ICA-6317197

Utrecht University

Material Segmentation

MASTER'S THESIS

Author
Emiel Bos

Project supervisor
Dr. ir. Ronald W. Poppe

Second supervisor
Prof. dr. Remco C. Veltkamp

External supervisors
Berry van Someren, MSc
Dr. Thijs van Lankveld

Dr. ir. Bas Boom
Julien Vijverberg, MSc

October 2019

Abstract

Rapidly developing research on convolutional neural networks allows for generating semantic information on images in an increasing variety of ways and with increased segmentation performance. This work focuses on one such deep learning objective, which we termed material segmentation and which involves determining for each pixel in an image – in this research we work with street-level imagery – which of a predefined set of materials it represents. The context for this research was provided by a GIS company, which records and digitizes the public space and whose clients have requested such per-pixel material information. This company also employs deep learning to automatically detect and locate street furniture, and the material information additionally has the potential to improve this object detection.

We annotated our own dataset, which is deficient in both the number of annotated images and the ground truth coverage per image. In addition, this dataset suffers from severe class imbalance. In the first part of this research we explore techniques to either resolve this class imbalance or to mitigate its negative effect on material segmentation performance. In our case, the best loss function turns out to be class-weighted cross entropy loss, though only by small margin. We conjecture that our class imbalance is too severe, and renders the dataset intractable without merging small, non-performing classes together or acquiring more ground truth for the small classes.

In addition to colour values, we also have depth information to our disposal, which gives the distance from the camera to the surface at each pixel. Contrary to our expectations, our network cannot manage an increase in performance when working with a unified RGBD representation over only RGB colour input. Our experiments indicate that, even though depth maps hold some discriminatory value, they become superfluous in the presence of colour information. We show examples where depth falls short in terms of discriminatory value by visual inspection.

Lastly, we assessed the extent to which our lack of training data holds performance back. We deduce that two factors are likely to cause the largest gains in performance: increasing the number of segmentations for classes with little ground truth, and increasing the number of training images to a thousand. Such measures should be able to effectuate satisfactory performance. Further improvements in material segmentation performance are likely to be gained by swapping depth maps for intensity maps, and simultaneously performing instance segmentation and material segmentation, using a joint network like Panoptic FPN.

Contents

1 Introduction

This research was carried out at CycloMedia Technology, a company specialized in GIS and located in Zaltbommel, The Netherlands. Founded in 1994, CycloMedia performs large-scale panoramic photography, three-dimensional reconstruction and geographical mappings of the public space. Among other things, this digitization of the exterior world is used for inventorization of common objects therein, e.g. traffic lights and signs, benches, lampposts, etc. The company employs Ford Fiestas mounted with panoramic image cameras, yielding 360°, 250 megapixel[*] images called *cycloramas*, and Velodyne HDL-32E LiDAR sensors. These cars drive all public roads in commissioned municipalities (they have driven throughout The Netherlands) and capture their surroundings at five meter intervals using said photography and LiDAR measurements. LiDAR measurements yield point clouds; collections of three-dimensional coordinates corresponding to surface points with which the pulsed laser light – emitted by the LiDAR equipment – collided and reflected back. Since this equipment knows its geographical location through GPS and SLAM postprocessing, the azimuth and altitude of the emitted signal through IMU, and the distance from the equipment to the collision (and reflection) point (determined by halving the time between emission and reception of the signal), the three-dimensional position of the point of collision can be resolved.

These point clouds offer only a sparse sampling of the environment. Therefore, the points are connected by edges using a surface reconstruction algorithm in order to approximate the actual surfaces that were sampled. Such a surface reconstructed object or environment is called a mesh, and is the conventional method of storing 3D models within computer graphics and geometry processing. Figure 2 shows the process of generating a textured mesh from a point cloud.

As the camera location, orientation, and focal length are known for each cyclorama, the 3D mesh reconstructed from the point cloud recorded at the same location can be related to that cyclorama in such a way that the image and mesh are spatially coherent. Moreover, the absolute geographical locations of the points are known, due to GPS, IMU, and SLAM postprocessing. The geometric relationships between the 3D objects and their projection on the image plane of the cyclorama is illustrated in Figure 3. Besides the RGB colour values, an extra channel can then be added, which covers all pixels in a cyclorama and indicates the distance, or depth, towards the surface that was projected on that pixel. This channel is called a *depth map*[†], and can be combined with the colour information to form a *RGBD* representation of an image. The other way around, the reconstructed environment meshes can be enhanced by texturing the meshes using the colour information from the cycloramas, pictured in Figure 2c. The entire data collection pipeline for all CycloMedia's forms of data is visualized in Figure 1.

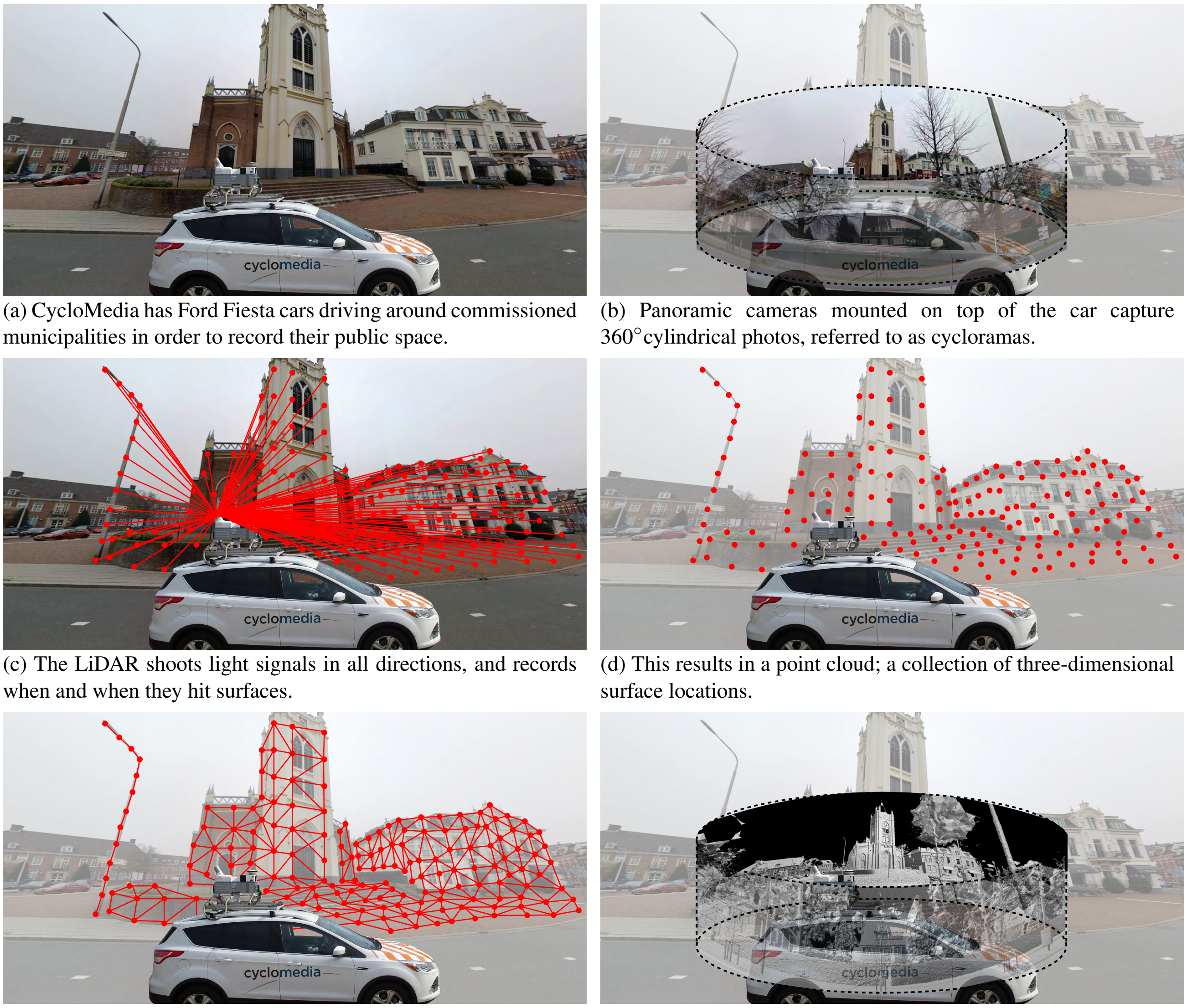
1.1 Problem statement

CycloMedia's customers mainly include municipalities requesting either inventorization and localization of their property in public spaces, or merely the panoramic captures. CycloMedia also offers a web-based portal, called Street Smart, which enables customers to measure, inspect, and take inventory of the public space themselves. Customers increasingly request more fine-grained attributes of objects, including their material properties. As an example, different materials often have different material cycles, and these can be planned with such segmentations. Inspection with respect to a specific material or set of materials can be eased by highlighting these materials in cycloramas. The total surface area of a material can be obtained instantaneously, e.g. the total square kilometers of asphalt roads in the province of Utrecht. This research is at least partially due to these demands. Attributing material information to a digitization of the public space requires a segmentation of the cycloramas based on the materials the pixels represent, i.e. each pixel will need to be classified according to a predefined set of materials. The focus of this research will be on material segmentation of the cycloramas and examines which models and settings yield what quality of material segmentation.

Secondly, these material segmentations could enhance the quality of the instance segmentations, since most materials only tend to appear in a limited set of object types. Therefore, a material prediction can alter the probability distribution over the possible object types. In turn, knowing where which objects are located in images – the result of instance segmentation – could benefit material segmentation. This is backed by the intuition that objects are often made out of a restricted range of materials, e.g. trees consist of the material types `wood` and `foliage`. This synergy between material segmentation and instance segmentation is reserved for future work, however. We note that in researching such a synergy, both material segmentation and instance segmentation can be performed separately, with the effects of either upon the

[*] As of the upcoming DCR11 iteration of CycloMedia's Digital Cyclorama Recorder (DCR) system.

[†] This is not strictly true, as CycloMedia defines depth maps to be separate image files into which the depth is encoded. In this work, however, the term refers to an image's fourth channel, encoding the depth.



(a) CycloMedia has Ford Fiesta cars driving around commissioned municipalities in order to record their public space.

(b) Panoramic cameras mounted on top of the car capture 360°cylindrical photos, referred to as cycloramas.

(c) The LiDAR shoots light signals in all directions, and records when and when they hit surfaces.

(d) This results in a point cloud; a collection of three-dimensional surface locations.

(e) Surface reconstruction algorithms transform the point clouds into 3D models. The texturing is visualized in Figure 2.

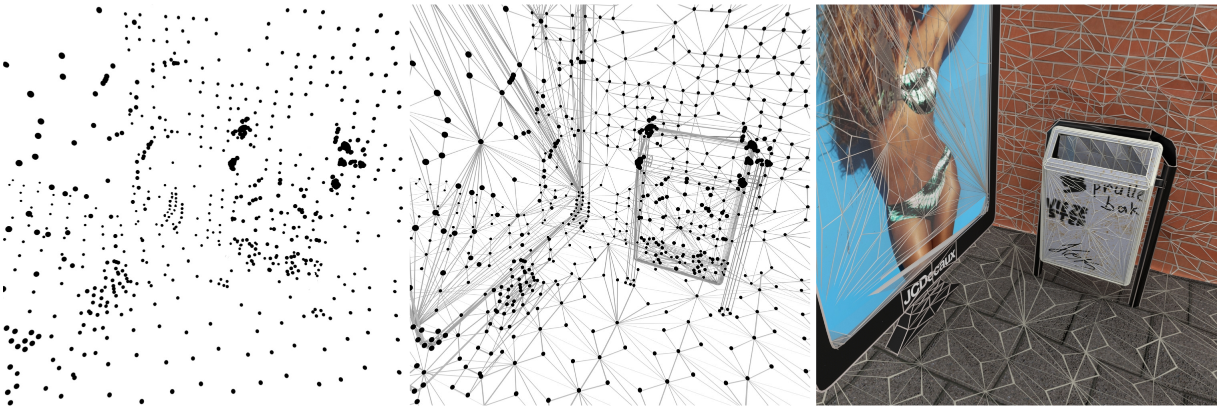
(f) These meshes are finally used to generate depth maps, which show for each pixel in a cyclorama the distance to its surface.

Figure 1: CycloMedia's recording process of the public space, and the data collection pipeline for generating novel data based on the recordings.

other being studied, but the two tasks could also be unified into a joint task, producing both material segmentations and instance segmentations simultaneously on the same image.

We restrict our solutions to convolutional neural networks, or CNNs for short. CNNs belong to the class of machine learning models known as deep learning, and have a proven track record of semantic image processing within computer vision [53]. Various computer vision tournaments and challenges are dominated by, and often solely consist of, CNN entries [47]. Tournaments organized during the year 2012 saw a leap in prediction quality across the various image processing tasks when CNNs were (re)introduced. This allows for arguing that deep learning outperforms more shallow machine learning models for computer vision in general and the aims of this research in particular, and provides us with reason to only consider deep learning for our objectives.

With the depth maps and geographical coordinates of the cycloramas, the detected objects in images can be related to real-world locations. However, CycloMedia currently indicates the location of objects within cycloramas with *bounding boxes*: two-dimensional rectangles that demarcate an object's pixels as tightly as possible. These bounding boxes will naturally also comprise superfluous pixels that do not belong to the object. These background pixels will pollute the approximation of object locations. A better alternative would be to use *segmentation masks*, where each detected object



(a) A point cloud

(b) A mesh after surface reconstructing the point cloud

(c) A textured mesh

Figure 2: Three stages in the process of surface mesh reconstruction.

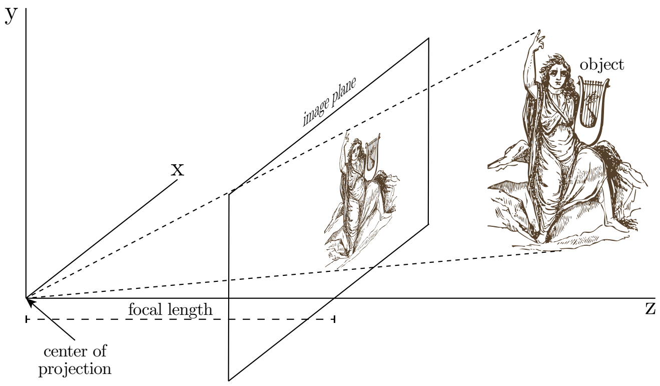


Figure 3: The relation of 3D point coordinates (of either captured real-world objects or reconstructed meshes) and their projections on the image plane.

has a binary mask indicating which pixels belong to it. In that case, only relevant depth values are taken into account.

1.2 Research questions

The goal of this work is to semantically segment street-level cycloramas as accurately as possible with respect to materials, i.e. each pixel needs a material label. This goal is formulated in our main research question:

RESEARCH QUESTION: *"What material segmentation performance are we able to obtain on outdoor imagery?"*

We quantify *performance* in this work using the mean intersection-over-union (mIoU).
Three aspects of material segmentation are investigated, to each of which we dedicate a subquestion. The first of these concerns the class imbalance of the dataset we annotated, which is certain to hamper performance. This means that there are classes with multiple orders of magnitude more ground truth annotations than others. This causes the networks to be biased towards the large classes, and since mIoU averages IoU of classes equally, this harms our performance. We therefore investigate this problem and formulate it our first subquestion:

SUBQUESTION I: *"What measures significantly reduce either class imbalance or the adverse effect of class imbalance on material segmentation performance?"*

We consider the negative influence of class imbalance significantly reduced when no class has a near zero mIoU due to lack of ground truth.

One would expect the addition of a depth map to benefit material segmentation, as this should facilitate distinguishing

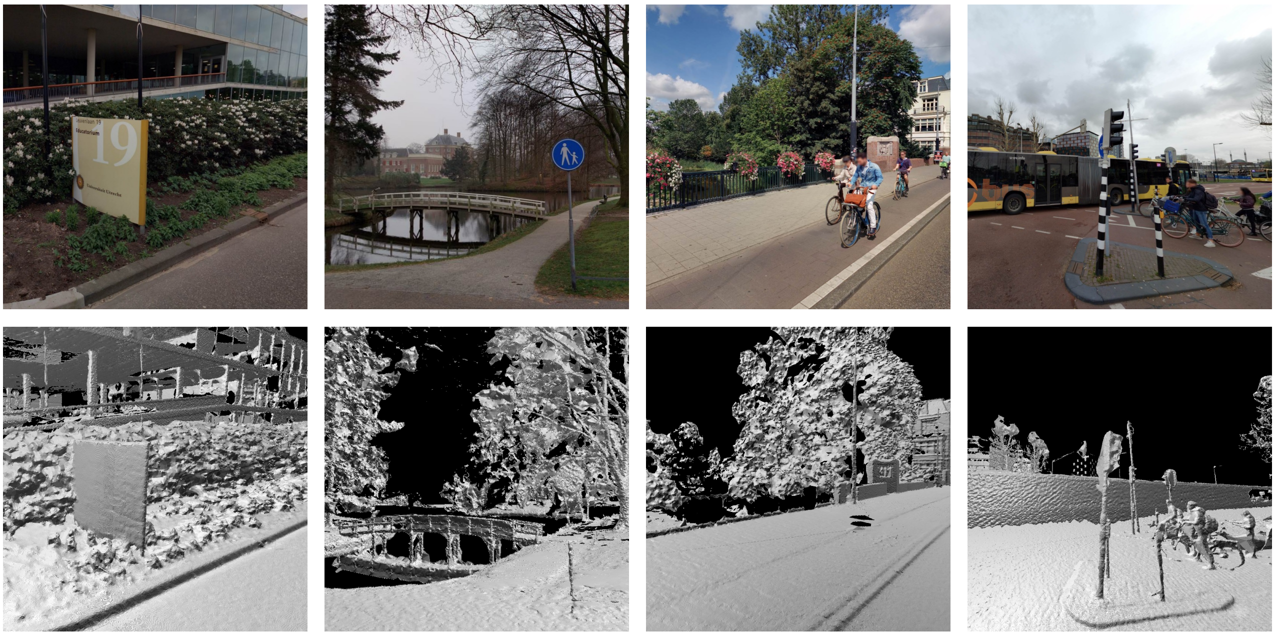


Figure 4: Visualizations of depth maps for four images. Since the difficulty of the human eye to perceive small differences in colour makes representing small deviations in depth challenging, we show shaded meshes instead of a visualization of the depth map. This allows one to observe that, though the depth is generally noisy, the amount and texture of the noise differs between some materials, and depth therefore still is expected to hold important discriminitve information.

different objects, indicates surface orientations – a surface's normal might be of influence to the material's posterior distribution – and might encode additional discriminatory properties. The depth maps are rather noisy, though the noise itself is characteristically distinct for some materials. See Figure 4 for some examples. The first image shows that the street and sign have relatively smooth depth while the dirt patch on the bottom left has a much coarser depth, and the depth of foliage is by far the coarsest. The second image shows a difference in coarseness between the asphalt and the grass. Barring the reflection, water has not been recorded, because the LiDAR signals were not reflected back to the sensor. Whenever signals do not return to the sensor, a depth of zero is recorded instead. Moving objects are difficult to manage, as they often create stretched artifacts of non-existent depth. In some cases, this can be mitigated by determining surfaces which show up in a point cloud as scanned from one position, but disappear in the point cloud generated from the next position. The third image shows that the bicyclists have been successfully omitted from the depth. Removal of moving objects and their artifacts does not always work; the fourth image depicts an imaginary wall created by the moving bus.

For these reasons it is reasonable to expect depth to contain discriminitive value in segmenting materials, since materials tend to differ in depth texture and orientation. We investigate the difference in performance between the two types of input:

SUBQUESTION II: *"How much does the performance of material segmentation with additional depth input improve compared to our network only receiving colour information?"*

In our research, we work with a limited dataset. Resolving the two above subquestions should enable us to give an adequate answer to our research question, given this small amount of data. We employ methods to assess the extent to which the data scarcity holds performance back, and assess what material segmentation performance is likely to be possible with a larger dataset. The third subquestion addresses this:

SUBQUESTION III: *"How much does material segmentation performance increase with more training data?"*

The answers to all these subquestions will give a good sense of material segmentation's feasibility.

2 Literature Review

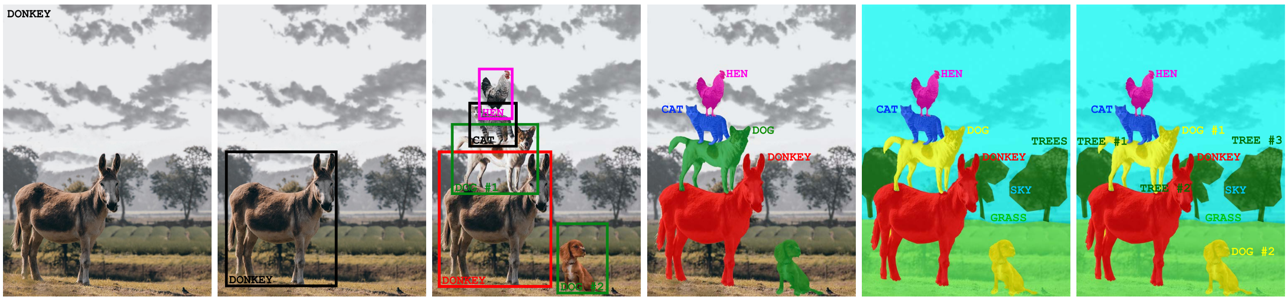
Nowadays, computer vision as discussed in this work is commonly approached with machine learning. Within the broad field of artificial intelligence, machine learning is the prominent branch in which algorithms and models are developed that make computers learn to perform increasingly complex tasks without explicit instructions. Instead, machine learning models infer these instructions themselves. More concretely, prior to the machine learning revolution, algorithms utilized man-made, fixed rules that defined how to operate on data. For all but the most trivial of tasks, hard-coding these rules is cumbersome, inefficient, error-prone, and poses the risk of having to be redone once their purpose or the characterstics of the data change. As a solution, algorithms and models in machine learning are tasked with automatically inferring these rules, using *training data*, which essentially teaches the model to infer these rules itself. The downside to this is that training data is needed, and, depending on the objective, often a large amount.
Machine learning has proven to be a good solution to a multitude of use cases, with some examples being recognizing and classifying objects in photos or videos, interpreting human speech, and performing medical diagnosis. A commonly used supervised machine learning model is that of neural networks, to which the next section provides an introduction. "Supervised" here refers to the model being explicitly taught what to do, as opposed to unsupervised models, which have to infer features from the data themselves. Deep neural networks are neural networks possessing a certain degree of complexity – manifested in their physical depth; the number of cascading processing steps that operate on top of each other – and comprise the subset of machine learning techniques known as deep learning. This work will center around *convolutional neural networks*, an adaptation of neural networks for use in computer vision, and these will be treated in Section 2.2.

Within computer vision, multiple objectives, or use cases, are defined. When considering only computer vision, the six prominent objectives are image classification, object localization, object detection, instance segmentation, semantic segmentation, and panoptic segmentation. Examples are given in Figure 5. Among these, image classification is the simplest, which concerns classifying an entire input image with a label from a predefined set of classes. Object localization aims to either indicate the location of the image's primary object or demarcate that objects spatial extent within the image. Bounding boxes, convex hulls, or other polygons could be used to effectuate demarcation. Additionally, the located object can be classified in a task aptly titled as object localization. Object detection performs object localization for potentially multiple objects, which can stem from the same class. Instance segmentation is identical to object detection, but yields pixel-based mask segments in order to describe an object's spatial extent. Lastly, semantic segmentation seeks to classify each individual pixel, with a disregard for individual instances. Adjacent pixels displaying different instances of the same class will therefore get partitioned into the same segment. Panoptic segmentation combines semantic segmentation and instance segmentation, classifying each pixel in an image while still maintaining instances. These five objectives are among the most common problems within still image deep learning. However, a multitude of other tasks have been formulated, with some examples accommodated in Figure 6.

This section starts with an introduction to neural networks in Section 2.1, and to convolutional neural networks in Section 2.2. The objectives that this work tackles, semantic segmentation, and its more specific variant material segmentation, are discussed in Section 2.3 and Section 2.6. Section 2.4 and Section 2.5 discuss the objectives of instance segmentation and panoptic segmentation respectively. We end our literature study with an overview of some existing datasets for the various objectives in Section 2.7.

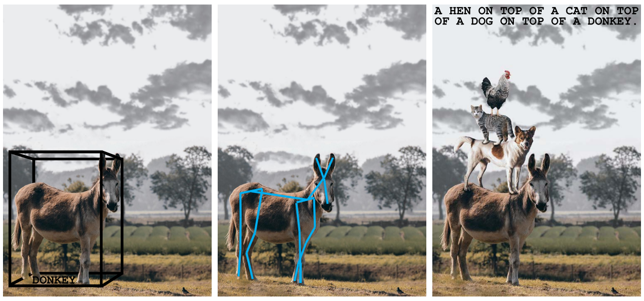
2.1 Neural networks

Artificial neural networks, henceforth simply referred to as *neural networks*, are processing mechanisms loosely based on biological neural networks as present in the brains of humans and animals. A human brain contains 86 billion neurons on average, connected to each other through outgoing axons and incoming dendrites. When an organism with a brain takes part in a particular activity, a certain subset of the neurons in its brain activate in a transmitted fashion. That is, a neuron receives input signals through its dendrites, and once the cumulative of these signals exceed a certain threshold, the neuron activates and passes the signal along its axons, which branch into synapses connected to other neurons' dendrites. When two neurons repeatedly activate together, or rather one neuron repeatedly causes activation in the other, then their connection strengthens and the efficiency of the one neuron carrying signal to the other increases. This is how brains learn, according to the Hebbian learning principle: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." [41] Neural networks are based on this concept, with its nodes – representing neurons – connected to each other in an arbitrarily complex function, and the strength of their connections



(a) Image classification (b) Object localization (c) Object detection (d) Instance segmentation (e) Semantic segmentation (f) Panoptic segmentation

Figure 5: Six common computer vision tasks. Image classification assigns a single label to the entirety of an input image. Object localization detects and classifies the primary object in the image and denotes its position and possibly its spatial extent with e.g. a bounding box. Object detection performs object localization for a potential multitude of objects. Instance segmentation does object detection, but segments the objects with a pixel-wise mask, denoting the exact shapes. Semantic segmentation assigns a class label to each pixel in the input image, and disregards different object instances. Panoptic segmentation classifies every pixel like semantic segmentation, but distinguishes between different instances from the same class, like instance segmentation. The image used of the town musicians of Bremen is CC0 public domain.



(a) 3D object localization (b) (Articulated) pose estimation (c) Image captioning

Figure 6: Some more computer vision tasks. 3D object detection localizes an object in three-dimensional space. Pose estimation attempts to reconstruct the skeletons of humans or animals visible in the image. Image captioning offers a natural language sentence describing the contents of its input image.

increasing with the magnitude of activations that pass through them during a so-called "training" procedure. This is commonly summarized as "neurons that wire together fire together", a phrased coined by Lowel et al. in 1992 [63].

Neural network are a class of highly popular and powerful machine learning models, capable of representing complex mathematical functions and learning latent features and patterns of digitally recorded phenomena. Neural networks can be used for classifying data, among other things. They are complex functions manifested as DAGs, taking as input a vector of input values $\mathbf{x} = \{x_1, ..., x_n\}$ corresponding to attributes of data that is to be classified, from which they produce k predicted probabilities $\hat{\mathbf{y}} = \{\hat{y}_1, ..., \hat{y}_k\}$ of the data belonging to each of the k respective classes.* The data can then be labeled with the class having the highest of these probabilities.

Nodes, the digital counterparts of the biological neurons, form the building blocks of a neural network. They receive, process, and transmit *activations*: values indicating the degree to which the corresponding node responds to the input data. Nodes are organized in *layers*, which are vectors of nodes connected to nodes in the preceding and/or subsequent layer, but are not inter-connected within a layer. Such layers are known as *fully-connected* layers, since each node is connected to all nodes of the following layer. Input data is supplied to the *input layer*, the nodes yielding the outputs constitute the *output layer*, and all layers in between are the *hidden layers*.

A template neural network is depicted in Figure 7, where n is the number of input nodes, j is the number of hidden layers, indicating the network's depth, and k is the number of output nodes, equalling the number of classes considered in the

*Originally, neural networks were formulated very differently, with one network per class. There are no defined rules or an official implementation, but the structure as presented here is what is commonly taught and used in practice.

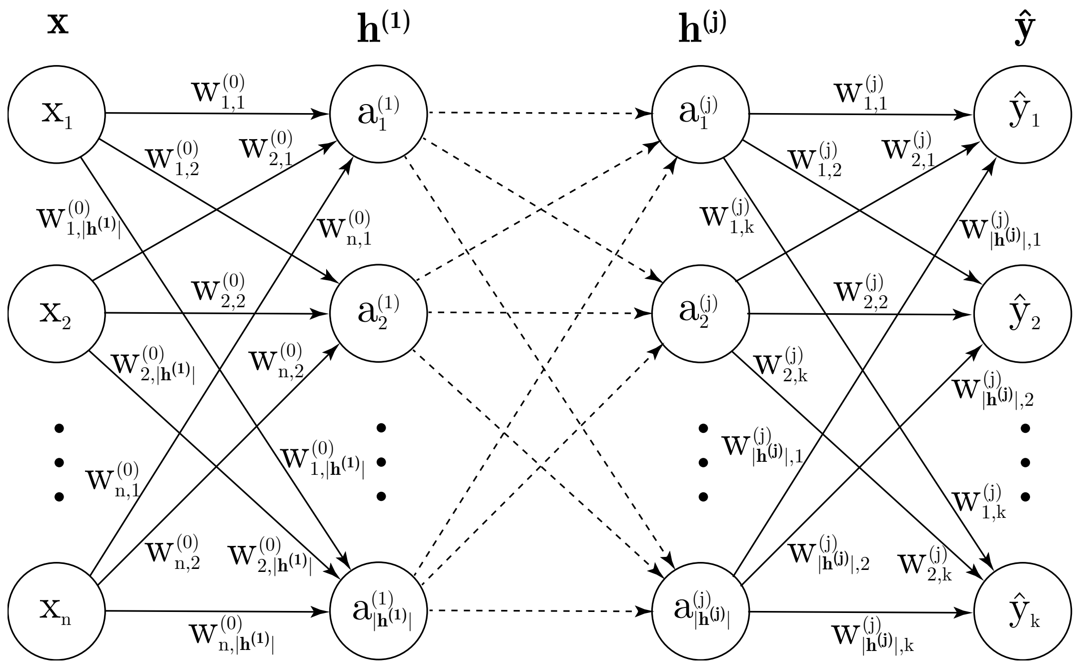


Figure 7: A template neural network, sketching their general structure. Before training, n, j, k, and each $|\mathbf{h^j}|$ are hyperparameters that are manually fixed. Not all weights are displayed due to spatial restrictions.

classification. The l^{th} hidden layer $\mathbf{h^{(l)}} = \{a_1^{(l)}, ..., a_{|\mathbf{h^{(l)}}|}^{(l)}\}$ contains $|\mathbf{h^{(l)}}|$ nodes. Each layer is a collection of nodes which together can be regarded as features of the input data.

Such a network *architecture*, or structure, is manually designed before it is being used, and is determined by a subset of the *hyperparameters*: (meta)parameters of the network that are not learned, but are tweaked manually to define some aspect of the network or training procedure. Using the notation at hand, n, k, j and each $|\mathbf{h^j}|$ are hyperparameters. Not all hyperparameters are concerned with the network's structure, however.* Each pair of nodes in subsequent layers of the network has an associated *weight*. Intuitively, weights determine the strength of node connections and indicate their correlation. If, during training, the network sees a lot of training samples where both nodes have either large or small activation, the weight of their connection will increase, in accordance with the aforementioned Hebbian learning principle. Additionally, each node in the hidden layers and output layer has another scalar, called a *bias*. The bias is an offset for a node in the case that it consistently, i.e. for the majority of training samples, has either a large or small activation. It is added to the sum of weighted activations from the previous layer before being activated to make sure the activation function is performed properly, since those are designed for zero-centered input. Because the sign and magnitude of weights are supposed to indicate the correlation between nodes, biases will make sure this is adhered to by shifting their norm towards zero. In this way, and intuitively speaking, only meaningful and indicative combinations of nodes will activate similarly. The weights and biases together form the *parameters* of the network, and the values of these parameters are determined during the *training* of the network, a process which is explained later. Assuming this network has already been trained, the classification of a data point – called *forward propagation* – consists of calculating the activations within the nodes layer-by-layer, until the output nodes are reached. At any hidden layer, the activations comprise a feature representation of the input, whatever that representation that the network has taught itself during training might be. The activation of the o^{th} nodes in the l^{th} hidden layer is then calculated according to

*Some authors debate whether the hyperparameters determining the structure (n, k, j and each $|\mathbf{h^j}|$) can be considered hyperparameters in the first place. They argue that these are mere characterstics of the chosen network topology, while hyperparameters determine the training of the network, e.g. learning rate, batch size, and other properties of training we will see later on.

$$a_o^{(l)} = \sigma([\underbrace{\sum_{i=1}^{|\mathbf{h^{(l-1)}}|}}_{\text{sum over all nodes in previous layer}} \underbrace{a_i^{(l-1)} w_{i,o}^{(l-1)}}_{\text{weighted activation from previous layer}}] + \underbrace{\gamma_o^{(l-1)}}_{\text{bias}}) \tag{1}$$

where $w_{i,p}^{(l-1)}$ is the weight associated with the connection from the i^{th} node in the previous hidden layer to the current node, $\gamma^{(l-1)}$ is the bias associated with the previous layer, σ is an activation function, and the nodes preceding the first hidden layer are the input nodes: $\mathbf{h^{(0)}} = \mathbf{x}$. The function is more clearly visualized in Figure 10, showing how the node activation is calculated by summing the activations in the previous layer multiplied by the weights of their associated connections, after which a bias term is added and the whole sum is processed by an activation function. This activation function has two purposes. The first is squashing the sum, which could be any real number, to a fixed interval (either ranging from 0 to 1 or from -1 to 1, depending on the specific activation function), in order to limit all activations to a universal range. Secondly, most activation functions introduce non-linearity. Without this non-linearity, each neural network would merely be a linear function of its inputs regardless of its depth, which would forego the computation of more interesting features and restricting its uses only to a small set of facile problems. Early neural networks used the *sigmoid* activation function [36]: $\sigma(x) = (1 + e^{-x})^{-1}$, pictured in Figure 9a, but it proved awkward for training due to two disadvantages that render the training procedure more difficult and more slow [81]. These downsides have to do with *backward propagation* (backpropagation or backprop for short), the algorithm with which the network is trained. Backpropagation calculates the gradients of the difference between ground truth and network output with respect to each layer's output layer-by-layer, from the deep layers to the shallow ones. The difference between ground truth and network output is called *loss* and is a measure of how erroneous the network has predicted a particular input. The gradients of the loss with respect to the parameters then indicate in which direction – namely the opposite of the gradient's direction – and by how much the network parameters should be adjusted in order to reduce this loss, which is the goal of the training process. When passing through sigmoid activations in the network, this so-called "gradient flow" is multiplied by the derivative of the sigmoid function. Nodes with either a small or large activation are referred to as *saturated nodes*, and the derivative of the sigmoid function at these nodes is near zero. This means that the gradient, after being multiplied with the sigmoid's derivative, is nearly diminished, which slows down training significantly. The second downside arises from the node following a sigmoid always receiving a positive activation. At any layer, activations are multiplied with weights W, and the derivatives of the layer weights with respect to the layer outputs ($\frac{\partial f}{\partial W}$, which we will multiply with the "upstream" gradient calculated thus far) are the layer inputs, because $\frac{\partial ax}{\partial a} = x$. The layer inputs of a layer succeeding a sigmoid activation are always positive, and therefore the values of the gradient on W after having backpropagated through such a layer will always be either all positive or all negative, depending on the sign of the incoming gradient flow, because they result from a multiplication with a positive value. In other words, at each parameter update step, either all parameters are increased or they are all decreased. If the optimal parameter configuration contains both positive and negative parameters, the training procedure will be inefficient. Figure 8 shows an intuitive interpretation, where the parameter updates are zigzagging around the optimal gradient descent path.

A solution to this last problem is to make sure that the inputs to the multiplicative layer are zero-centered. The *tanh* activation function, pictured in 9b, increases its range to $[-1, 1]$, making sure that the next layer can receive both positive and negative activations. However, any gradient flow will still be significantly reduced when backpropagating through saturated nodes. The *ReLU* activation function (first used in AlexNet, discussed later) trades the zero-centering in for good derivatives in the positive region of its domain (though not in the negative region), as can be seen in Figure 9c [51]. ReLU simply takes the maximum of 0 and its input: $\sigma(x) = \max(0, x)$ It also converges around six times as fast as sigmoid or tanh, and better fits the biological allegory of (convolutional) neural networks. A variant of ReLU, called *Leaky ReLU*, makes sure that also the negative domain has a small derivative by replacing the flat line with a slight upward slope, making sure that a network always gets at least some gradient flow [64]. What's more, it can output negative values instead of strictly positive ones. The amount of slope can be parameterized using a *Parametric Rectifier*, or *PReLU*, which learns the slope parameter during training [38].

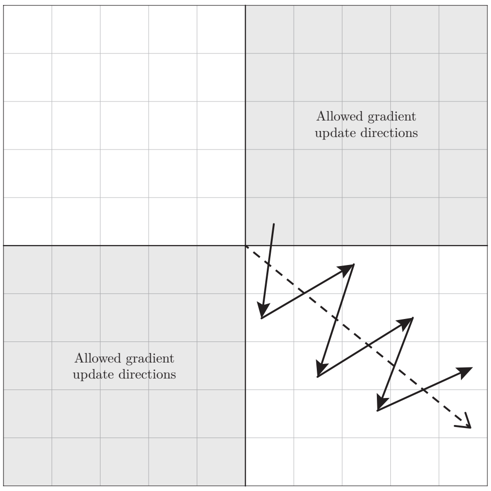

Allowed gradient update directions
Allowed gradient update directions

Figure 8: The sigmoid activation function always outputs positive values, which, because of the rules of derivation, causes the values of the gradient with respect to the weights of the corresponding layer to be either all positive or all negative. In other words, the parameters of a layer are either all incremented or all decremented per update step. Illustrated here is a hypothetical optimal weight vector, with only two parameters for demonstrative purpose, as a dashed line. The updates can be only either all positive (all possible update vectors in the first quadrant) or all negative (all possible update vectors in the third quadrant), resulting in the inefficient, zigzagging gradient descent updating around the optimal gradient descent path.

The calculation of an entire layer of activations can also be summarized as a matrix-vector multiplication of weights with previous activations:

$$\mathbf{a}^{(\mathbf{l})} = \sigma(\begin{pmatrix} w_{1,1}^{(l-1)} & \cdots & w_{|\mathbf{h}^{(\mathbf{l-1})}|,1}^{(l-1)} \\ \vdots & \ddots & \vdots \\ w_{1,|\mathbf{h}^{(\mathbf{l})}|}^{(l-1)} & \cdots & w_{|\mathbf{h}^{(\mathbf{l-1})}|,|\mathbf{h}^{(\mathbf{l})}|}^{(l-1)} \end{pmatrix} \begin{pmatrix} a_1^{(l-1)} \\ \vdots \\ a_{|\mathbf{h}^{(\mathbf{l-1})}|}^{(l-1)} \end{pmatrix} + \begin{pmatrix} \gamma_1^{(l-1)} \\ \vdots \\ \gamma_1^{(l-1)} \end{pmatrix}) = \mathbf{W}\mathbf{a}^{(\mathbf{l-1})} + \gamma^{(\mathbf{l-1})} \tag{2}$$

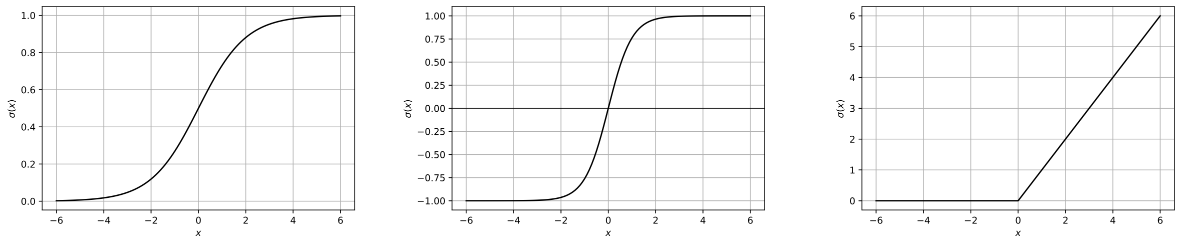
In order to have the activations at the output nodes interpretable as a probability distribution, these activations will need to sum to 1. To that end, a *softmax function* can be used [10, 11]:

$$\hat{y} = s(a_p^{(j+1)}) = \frac{e^{a_p^{(j+1)}}}{\sum_{i=1}^{k} e^{a_i^{(j+1)}}} \tag{3}$$

Training

The parameters of the network (the weight and bias values) are determined through training. In order to train a neural network, a designated *training set* is required, consisting of *training samples*: inputs for which the ground truth label is known. *Ground truth* is the manually assigned set of true labels which serve as examples for the network to learn from. Because these represent the optimal prediction the network could possibly make, it can be regarded as the golden standard. These training samples are iteratively classified using forward propagation. For each iteration, after the class probabilities are obtained, the *loss* quantifies the network's inability to correctly classify the training sample. This is calculated as some function of the difference between the ground truth and the output probabilities. An example of a loss function for neural network classification is the *cross entropy loss*, also sometimes termed (multinomial) logistic loss, as was originally used in combination with multinomial logistic regression [73]. It is defined as:

$$L_{\text{CE}} = -\sum_{i=0}^{C} y_i \log \hat{y}_i \tag{4}$$



(a) The sigmoid activation function: $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$ [36].

(b) The tanh activation function: $\sigma(x) = \tanh x$ [55].

(c) The ReLU activation function: $\sigma(x) = \max{(x, 0)}$ [51].

Figure 9: Different activation functions. The sigmoid function has neither zero-centered outputs nor does it provide sufficient gradient flow at both large and small activations. The tanh function stretched its range in order to output zero-centered activations, but still kills gradient flow at both ends of its domain. ReLU is linear in its positive domain – positive activations will always yield a sufficient gradient – but it is again zero-centered. Leaky ReLU (not pictured) has a slight slope in order to mitigate saturation in the negative domain and enable itself to output negative activations.

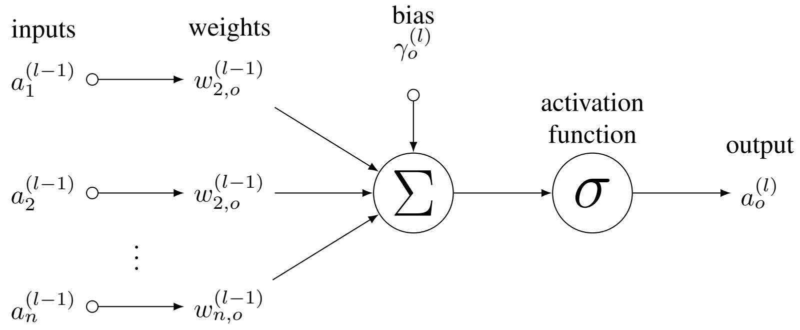


Figure 10: Diagram visualizing the process of activation calculation. The weighted sum is taken over all activations in the previous layer, with the weights belonging to the connections connecting each corresponding node in the previous layer with the node of the activation that is currently being calculated. A bias term is added to this sum, and the compound is processed by an activation function, such as sigmoid or ReLU. Reproduced with permission from [90].

where y_i is the ground truth and $\hat{y}_i$ the predicted probability[*] for each class i in C. Note that y_i is 1 if i happens to be the ground truth class, and 0 otherwise. For this reason, a simpler formulation takes the negative natural logarithm of the prediction score of the true class as denoted by the ground truth label:

$$L_{\text{CE}} = -\ln P(Y = y_s \mid X = x_s) = -\ln \hat{y}_s \tag{5}$$

where x_s is the particular training sample input vector being processed, y_s is the ground truth class of that training sample and $-\ln \hat{y}_s$ is the network's predicted probability for the ground truth class. The natural logarithm of the probability is taken, because those prove easier to optimize for than the probability itself. Multiplication is more computationally expensive than addition, and in logarithmic form, the product of probabilities will transform to the sum of log probabilities. Moreover, sufficiently small digits may cause underflow, meaning precision is lost as a result of a computers inability to store all its decimals. In log space, this is less of a problem. Since logarithms with base greater than 1 are monotonically increasing functions, optimizing with respect to the negative logarithm will yield the same results as optimizing with respect to the original probabilities. Lastly, the log probability is negated, as the loss should be inversely proportional to the probability of the true class.

Since neural networks of sufficient complexity oftentimes have too many parameters to be able to analytically solve for the weights that give the minimum loss, a numerical solution called *gradient descent* is used. The downside to this is that

[*] For binary classification, the probabilities are the result of an activation function, e.g. sigmoid. For multi-class classification, they are the output of a normalization function, e.g. softmax. However, not all variations of cross entropy loss work upon the final probabilities. For problems in which each sample can take on multiple labels, it is common for the cross entropy loss function to work on top of the network scores or logits, which are the penultimate activations that would normally feed into the final activation function.

finding the global optimum is not guaranteed. Instead of viewing the neural network as a big function that is parameterized by weights and that takes input vectors and outputs probability vectors ($f(\mathbf{x}; \mathbf{W}) \to \hat{\mathbf{y}}$), we can also view the network as a function with as inputs the weights and as output the loss of a training sample, given that sample: $f(\mathbf{W}; \mathbf{x}, \mathbf{y}) \to L$, where L is the loss when predicting sample $\mathbf{x}$ and comparing its output with ground truth $\mathbf{y}$ using a network with weight matrix $\mathbf{W}$. Since a neural network consists solely of additions, multiplications and activation functions (which are always derivable), we can calculate the partial derivatives of the loss with respect to each parameter. Disregarding the bias parameters for notational simplicity[*], a matrix containing all these partial derivatives is known as a gradient:

$$\nabla \mathbf{W} = \begin{pmatrix} \frac{\partial L}{\partial w_{1,1}^{(l-1)}} & \cdots & \frac{\partial L}{\partial w_{|\mathbf{h^{(l-1)}}|,1}^{(l-1)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial w_{1,|\mathbf{h^{(l)}}|}^{(l-1)}} & \cdots & \frac{\partial L}{\partial w_{|\mathbf{h^{(l-1)}}|,|\mathbf{h^{(l)}}|}^{(l-1)}} \end{pmatrix} \tag{6}$$

Note that the gradient has the same shape as the weight matrix itself, which is true for the gradient of any type of variable. Each partial derivative indicates how much the loss changes when changing only the corresponding parameter and keeping the other parameters fixed. Therefore, in order to reduce the loss, each parameter should be incremented by a fraction of the associated partial derivative. The reason why only a fraction is taken, is because these gradients can be arbitrarily large, which would result in arbitrarily large parameter updates. This fraction is another hyperparameter called the *learning rate*, and finding an optimal learning rate proves to be a challenge in itself. A low learning rate will make the network slower at finding a parameter setting with low loss, as it is taking smaller steps towards a minimum loss, while a high learning rate might constantly overshoot optimal parameter settings, which also renders learning slow. A more severe problem of too high a learning rate is its risk to land in sub-optimal areas of the *loss landscape*, the surface of the graph of the loss function. Ideally, training will make the network parameters *converge* to a global minimum, or local minimum with similar loss. However, there always exists the danger of the network landing in severely sub-optimal local minima or saddle points, examples of which are pictured in Figure 11 [25]. As the gradient in both these kinds of points is zero, it is impossible for the network to progress without a more sophisticated learning methodology. Even though saddle points seem a trivial, rarely occurring problem in low dimensions, they become increasingly prevalent in higher dimensions and will occur more frequently than local minima [25]. This corresponds with intuition, because a saddle point is essentially a point at which the loss increases in at least one direction and decreases in at least one other direction. The higher the dimensionality of the loss landscape, the higher the chance that one of the increasingly many dimensions has a negative gradient direction, while another is positive.
A second problem, also exacerbating with increased parameter complexity, is the problem of a jittery, inefficient gradient descent path caused by a point on the loss landscape on which the loss changes quickly in one direction while barely changing in another.[†] The gradient descent path will then progress slowly along the slowly moving direction, while jittering along the sensitive dimension – changes along which results in large changes of the loss. To combat these problems, the idea of *momentum* was introduced [85]. Instead of computing the next parameter setting at time $t+1$ as $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta * \nabla \mathbf{W}_t$, where η is the learning rate, we use $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta v_{t+1}$, where $v_{t+1} = \rho v_t + \nabla \mathbf{W}_t$ is the velocity or momentum that is being build up as a moving average of past gradients and ρ is a hyperparameter serving as friction. The latter is typically between 0.90 and 0.99. This momentum increases the chance of learning paths overshooting local minima and saddle points, and cancels out opposing directions within a dimension the path would else be jittering over.

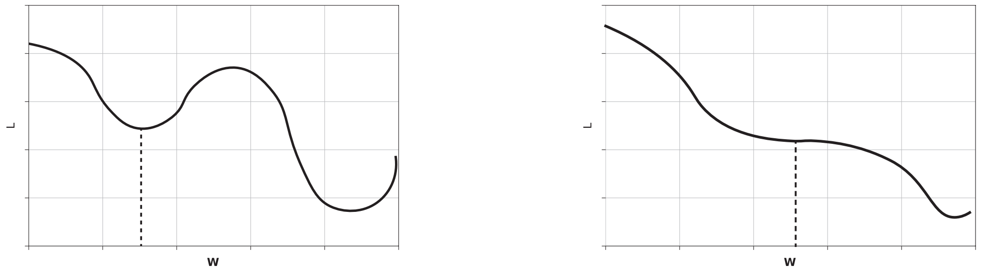
The gradient of the loss with respect to its inputs is calculated using the previosuly mentoined backpropagation algorithm. Using rules of derivation, especially the chain rule, partial derivatives of the loss with respect to a layer's parameters can be calculated using the partial derivatives previously calculated in the "next"[‡] layer. This is done layer-by-layer, working backwards through the network until the partial derivatives with respect to every parameter in the network are known (and the full gradient of the parameter matrix has been calculated).

During gradient descent, every training sample in the training set is forward propagated and their losses are averaged before one parameter update – one step on the gradient landscape – is performed using that average loss. Even though the gradient (more) accurately represents the optimal gradient on the loss landscape, this method is often impractically slow,

[*] Biases are typically treated as weights in a parameter matrix or gradient.

[†] This is referred to as the loss having a bad condition number at this point, which is characterized by a large ratio between the largest and smallest values in the parameters' Hessian matrix – the multi-dimensional matrix containing all possible second-order partial derivatives of the loss with respect to the weights

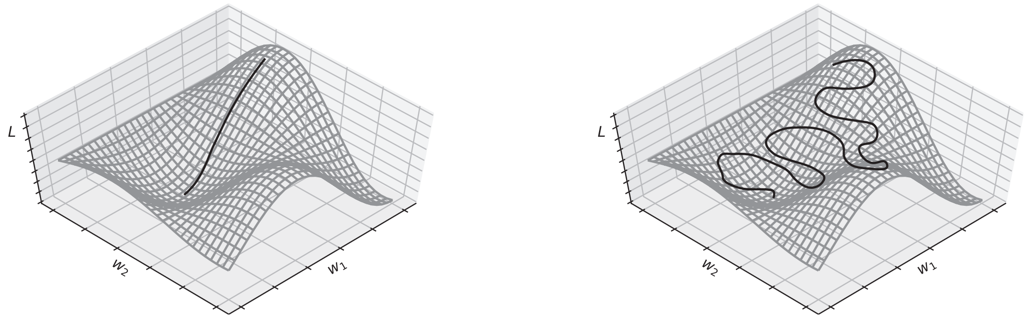
[‡] According to the sequence of layers as seen during forward propagation.



(a) Local minimum

(b) Saddle point

Figure 11: Parameter settings for which the loss function yields a zero gradient, halting learning progression. In a local minimum, all directions exclusively increase the loss, leading gradient descent to conclude, since it thinks it has reached an optimal parameter setting. At a saddle point, the loss goes up in some directions, and down in others, with a current gradient of zeroes, which offers no guidance to gradient descent as to what step to take. The latter problem increases in frequency the higher the dimensionality of the parameters. Momentum combats both these problems, by incorporating a moving average of past gradients, and thereby keeping some velocity to bypass such points.



(a) Gradient descent

(b) Stochastic gradient descent

Figure 12: Paths taken by different gradient descent approaches in a loss landscape as a function of two parameters. Loss landscapes depending on the many parameters in a neural network are challenging to visualize.

since training sets can be arbitrarily large. At the other extreme, stochastic gradient descent (SGD) uses only one random (hence the term "stochastic") training sample before taking a step, resulting in a larger number of steps in more inefficient directions (using momentum "smoothes out" the descent path, though). Figure 12 depicts a loss landscape as a function of only two weights.* It depicts characteristic descent paths for both gradient descent and SGD. In practice, a balance is struck by mini-batch SGD, which groups randomly sampled subsets of training data into (mini)batches and performs its steps per batch.

In short, neural networks contain many parameters allowing it to process input in a way that it has taught itself during its training. This dynamic way of discriminating between latent features gives neural networks a large advantage over traditional machine learning methods, in which feature representations are to be designed manually.

2.2 Convolutional neural networks

Convolutional neural networks (CNNs) are an adaptation of neural networks for use in computer vision. A regular CNN is displayed in Figure 13. It consists of two parts. The first is a *convolutional* part that downsamples and convolves an input image to a small representational feature vector. This is then fed into the second *fully-connected* part, which is essentially a neural network as discussed in the previous section (Section 2.1). The convolutional part contains layers, like a standard neural network does, but each layer has its neurons or activations ordered in three-dimensional arrays rather than one-dimensional vectors, with a predetermined width, height, and depth. The input layer takes images of size $w * h * 3$, i.e. the number of pixels times the number of colour channels per pixel. Forward propagation then consists of a series of convolutions, where at each layer the image representation reduces in spatial resolution while (commonly) increasing in depth.

*An actual neural network has potentially millions of parameters, but visualization becomes difficult in higher dimensions

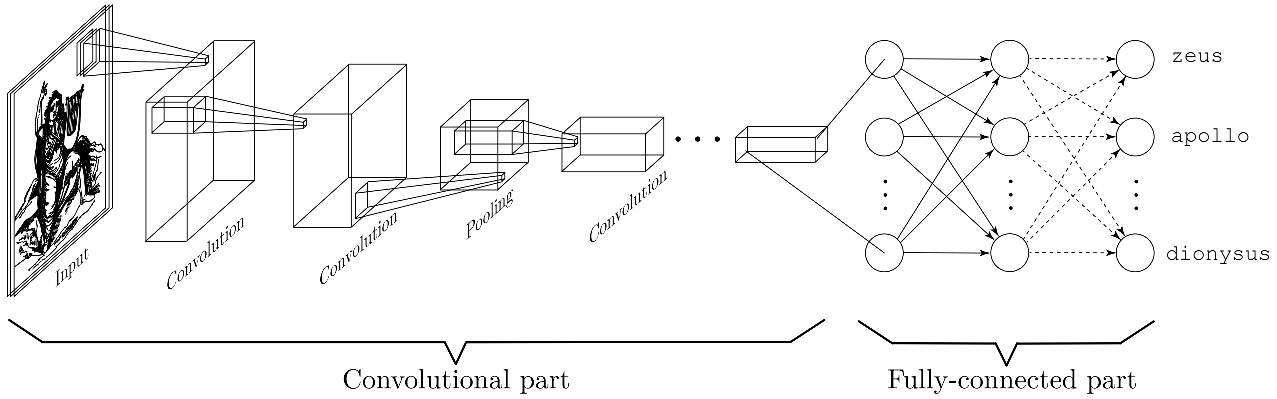


Figure 13: The structure of a regular CNN.

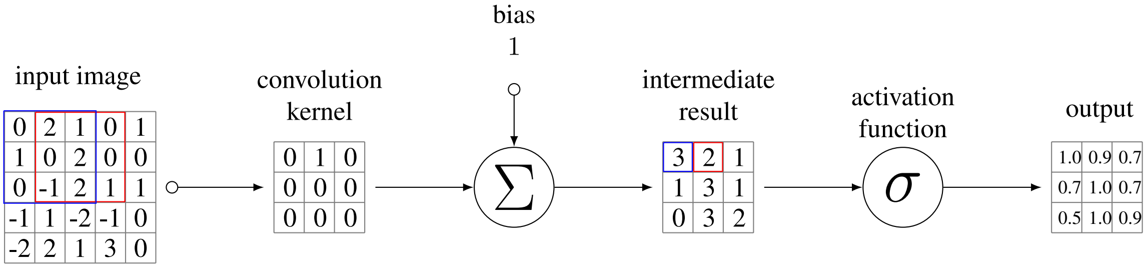


Figure 14: A convolutional layer in a convolutional neural network. Reproduced with permission from [90].

Each convolution – each transition from one layer to the next – has a number of square *kernels*, or *filters*, of equal size. Common filters sizes are 1x1, 3x3 and 5x5. The depth of these kernels is equal to the depth of the activation layer they act upon. In order to reduce only the depth dimension (or to increase it, which is sometimes needed before and after expensive convolutions in order to reduce compute), 1x1 filters can be utilized. The parameters of the convolutional part are contained in these filters. Conceptually, each filter in a convolution slides across the extent of the previous layer in a sliding-window fashion and produces one *activation map*, containing an activation for each spatial location of the kernel. Figure 14 illustrates how at each spatial position, the activations are element-wise multiplied by the corresponding weights in the kernel. These products are then summed together with an additional bias, of which each kernel has one, before being processed by an activation function. This yields the activation at the center pixel of the kernel at its current position in the new layer. Note that in Figure 14, the kernel is shown as a flat, two-dimensional matrix, while in practice kernels nearly always have a depth greater than one and extend over the depth of the previous layer. A kernel's depth equals the number of colour channels for the first layer, and the number of kernels of the previous layer for each subsequent layer.

Naturally, the filter should not exceed the layer boundaries, as there are no pixel values defined outside of it. This is not an issue when the filter moves over the layer in a pixel-wise manner. However, the filter can be made to move quicker using the *stride* hyperparameter, determining the interval of activation calculations. A convolution with stride 2 is shown in Figure 16. If the current settings of filter size and stride will make the filter move outside of the image's spatial extent, the layer can be *padded* with zero-valued activations that surround it. Figure 15 exemplifies a layer with a padding of 1. Instead of zero activations in the padding border, existing activations can be used, a method known as *reflection*.

The dimensionality of each hidden layer can be calculated according to

$$(\frac{W-F+2P}{S}+1)\times(\frac{H-F+2P}{S}+1)\times K \tag{7}$$

where W and H are the respective width and height of the preceding layer, F is the kernel size of the convolution, P is the number of padding borders, S is the stride, and K is the number of filters.

At the end of the convolutional part, the input image has been condensed to a small representation of it, encoding the important aspects of the input at a low resolution but with a much larger depth. This representation is transformed into a

0	0	0	0	0	0	0
0	0	2	1	0	1	0
0	1	0	2	0	0	0
0	0	-1	2	1	1	0
0	-1	1	-2	-1	0	0
0	-2	2	1	3	0	0
0	0	0	0	0	0	0

Figure 15: An input layer with pooling 1, meaning a border containing zeroes of thickness 1 is encircled around the layer. The additional zeroes are printed in red.

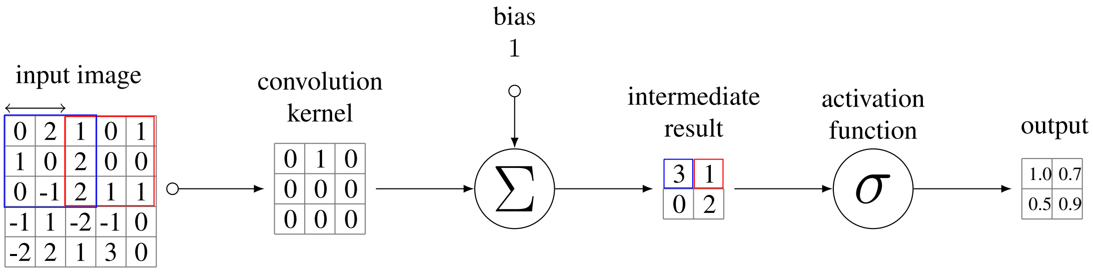


Figure 16: A convolution with kernel size 3 and stride 2.

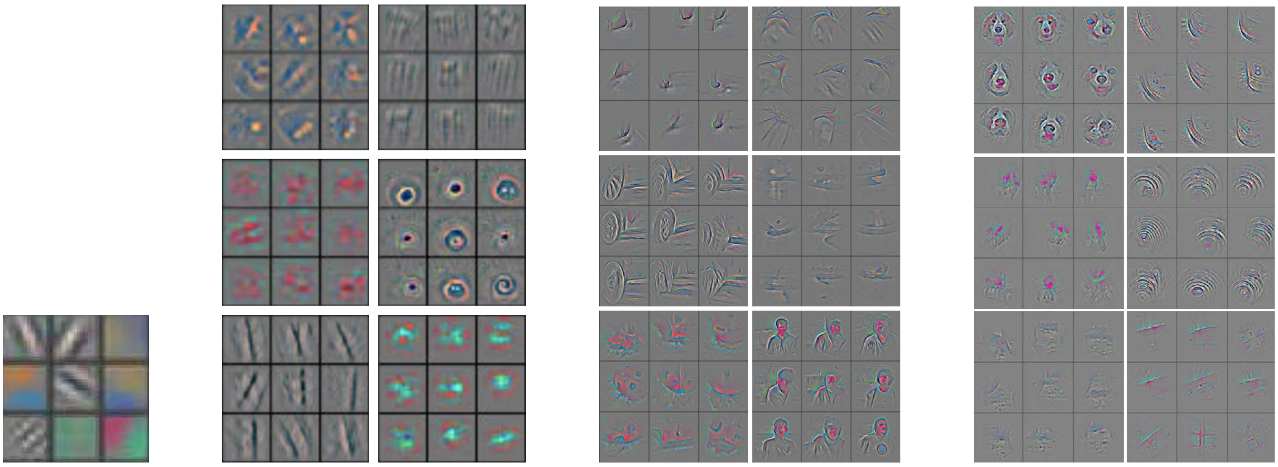
vector that serves as input to the fully connected part. Note that activations in an intermediate layer in the convolutional part only depend on and are connected to a small cube of activations in the preceding layer, whereas activations in the fully connected part are connected to all neurons in the preceding layer, hence the name. The fully connected part takes care of the classification of the input image, based on the feature representation produced by the preceding convolutional part.

Arguably the major drawback of neural networks is their uninterpretability. A machine learning model that is difficult to interpret is also called a *black box*, which produces results while its actual workings are difficult to gauge. They have a wealth of parameters, making it hard for humans to determine what consistencies, or features, the network has managed to discover. This contrasts with simpler machine learning classification models, like logistic regression and classification trees, which facilitate human understanding and insight. However, when dealing with convolutional neural networks, it is possible, to some extent, to assess the kind of semantic structures that kernels are looking for by visualizing their weights. Figure 17 shows reconstructed patterns that cause high activations for kernels belonging to convolutions at different depths. This process shows how kernels in shallow layers concern themselves only with basic patterns and deeper kernels look for abstractions of this simple geometry into more meaningful structures. Likewise, activations at deeper layers have a larger *receptive field*, the effective area that influences a kernel's activation. It can be thought of as the area of the input image that a neuron "sees".
Besides visualizing kernels, activation maps can also be visualized, with grayscale colors indicating the magnitude of activation at that location for a particular kernel. Figure 18 gives examples.

History and evolution of CNNs

The foundations for convolutional neural networks are arguably two works from the 1960s by Hubert and Wiesel, who demonstrated that a cat's visual cortex is highly hierarchical, with low-level neurons picking up on basic shapes, colors and movements, and neurons higher in the hierarchy abstracting upon their subordinates [44, 45]. In 1998, LeCun et al. formalized these observations into the first convolutional neural network, in which input pixels were processed in a similarly hierarchical way [54]. However, as computational power was not sufficient, deep learning image processing methods like theirs were not used in practice back then.

Multiple different conferences and institutions hosted and still host competitions on computer vision. Two notable historical competitions for image classification (Figure 5a) and object detection (Figure 5c) are the PASCAL VOC challenge [29], which ended in 2012, and the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [74], which held its



(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 17: A visualization of kernels with high activations from different layers in a CNN. As the image reaches deeper layers, the kernels capture more semantically meaningful structures and patterns. Images reproduced from [99] with permission.

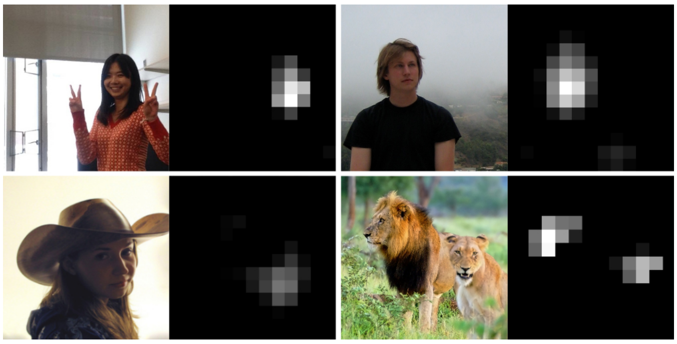


Figure 18: Images and their activation map processed with a kernel that appears to be looking for face-like inputs. Images from [96] with permission.

last competition in 2017. For the latter competition, the winning error margins are given per year in Figure 19. In 2012, the winning entry was a CNN by Krizhevsky et al., implementing LeNet as conceptualized in 1998 [51]. The increase is apparent in said figure. Their seminal work caused a renewed interest in deep convolutional neural networks, and every subsequent winning entry were CNNs of increasing complexity.

In the years following the conception of tractable CNNs in 2012 by AlexNet, newer architectures continuously achieved improved prediction quality, generally by becoming deeper; the deeper the network, the more it tends to be able to abstract and extract semantic information from its input. AlexNet is generally a CNN as described prior. It contains two branching paths of convolutions which both connect to the fully connected part, but this was due to a hardware restriction: the Nvidia GTX 580 graphics cards which were used did not possess sufficient VRAM to store all activations. Therefore, the network was spread across two GPUs, with each branch running on one of them. 2013's ZFNet kept the same general structure as AlexNet, but made slight changes to hyperparameters like stride and number of filters at certain layers [99]. 2014 saw two entries finishing close together: VGG (named after the Visual Geometry Group at the University of Oxford) [82] and GoogLeNet [86]. The former decreased the filter sizes and increased the number of layers from 8 to 16 (in VGG16) or 19 (in VGG19). The winner of that year was GoogLeNet, which is built using inception modules; local topologies that are carefully fine-tuned and stacked on top of each other to form the final network. Each inception module is therefore a network within a network. Since each module is designed to be efficient, the network as a whole is efficient as well. A naive version of GoogLeNet's topology is shown in Figure 20a. It performs three convolutions and one pooling operation in parallel. *Pooling layers* are solely used to reduce a feature representations dimensionality without utilizing parameters and therefore without adding semantic information, and likely losing such information. The most common pooling method is max pooling, illustrated in Figure 21 in which only the maximum activation is kept within each square

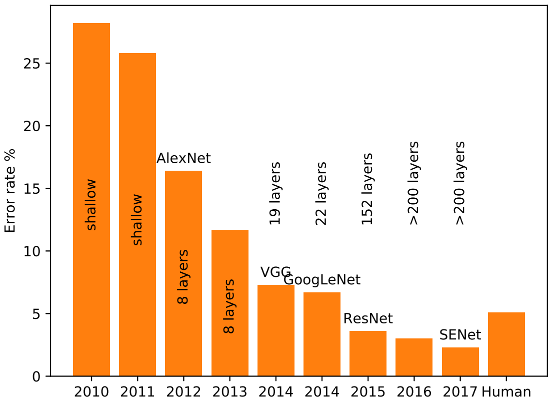
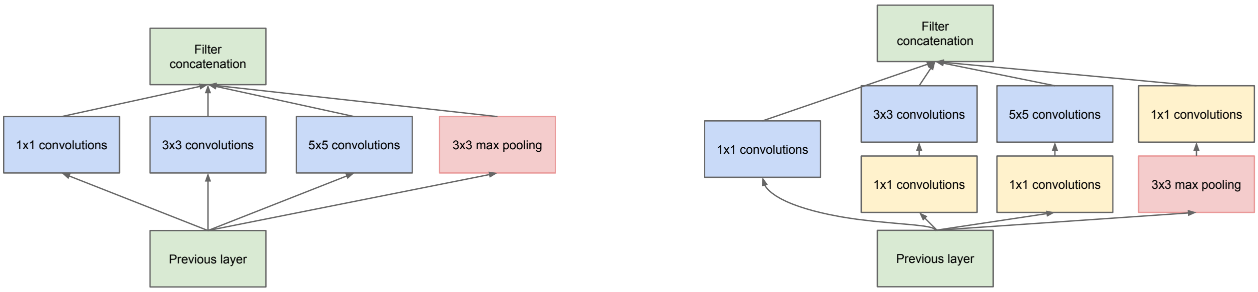


Figure 19: Top-5% error rates of the winning entries of ILSVRC challenges from years 2010 through 2017.



(a) Inception module, naive version

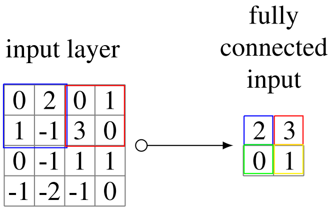
(b) Inception module with dimensionality reduction

Figure 20: The two iterations of GoogLeNet's inception module. Reproduced from [86] with permission.

region. Similarly to convolutions, pooling is also set using stride and region size hyperparameters.

Each activation layer operation within the inception module has a different kernel size. At the end, the resulting activation layers are concatenated, i.e. placed back-to-back to create a large block of data incorporating all incoming parts. Stride and zero padding are used to make sure the four activation layers are equal in resolution, enabling the concatenation to be a simple depth-wise stacking. However, this topology proved computationally complex due to the depth of inputs and number of kernels Szegedy et al. envisioned [86]. They addressed this by inserting "bottleneck" layers: 1x1 convolutions that only reduce the depth of an activation map. Three such layers are inserted: two before the 3x3 and 5x5 convolutions, and one after the pooling layer. In this way, the convolutions are performed on significantly less feature maps. These additional pooling layers contrast Figure 20b with Figure 20a. GoogLeNet contains nine of these modules, and with each module containing two layers the networks ends up having 22 layers, including fully connected layers. Despite GoogLeNet being significantly deeper than AlexNet, it only has five million parameters, twelve times as few as AlexNet. It can be concluded that a higher number of parameters of a network does not necessarily imply a higher prediction quality.

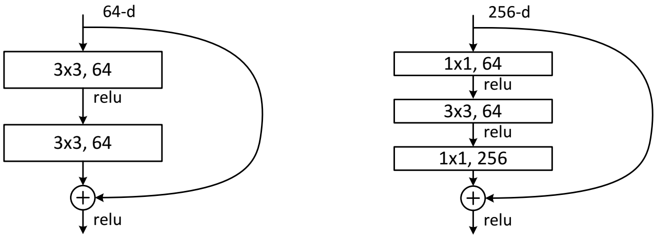
Starting in 2015, ILSVRC's winning entries all contained at least 152 layers. Intuition stated that deeper networks perform inherently better than shallower ones. However, deeper networks proved harder to optimize, though not as a result of *overfitting*, a scenario in which a model adapts too well to its training data and therein losing its ability to generalize to new and unseen data. Instead, this was caused by the magnitude of the gradient diminishing as it is backpropagated further back up the network. Repeated application of the chain rules decreases the partial derivatives, which makes training the shallow layers increasingly slow. This is known as the *vanishing gradient problem*. The 2015 ILSVRC winner, ResNet, adds *residual connections* or *skip connections* to deep architectures, pictured in Figure 22, which alleviates this problem [39]. Here, small groups of two or three convolutional layers are grouped into *residual blocks*. Besides the regular flow of convolution through such a block, activation layers are also routed around the convolutions, bypassing any processing, after which the layer is element-wise summed with its filtered counterpart. This skipping performs identity mapping, since the feature layer is not altered within the shortcut. The regular convolution flow, which is being bypassed by the



0	2	0	1
1	-1	3	0
0	-1	1	1
-1	-2	-1	0

2	3
0	1

Figure 21: Max pooling performed on an activation map. For pedagogical reasons, pooling is often taught using non-overlapping regions. We note that this is not a requirement; pooling regions may overlap.



(a) Residual block

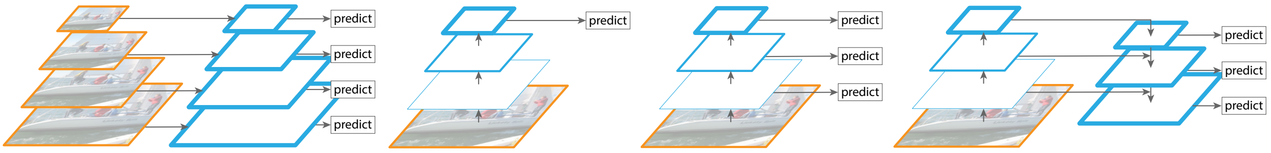
(b) Residual block with bottleneck layers

Figure 22: Two versions of a residual block in ResNet. Reproduced from [39] with permission.

skip connection, is called the residual mapping. In order to sum the input of a layer with its residual, they need to be equal in dimensions. In ResNet's case, input and output layers always equal in width and height. Any downsampling is offset by using padding. However, the input and output layers might differ in the number of channels, in which case the identity mapping becomes a linear projection.

The benefits of residual connections are twofold: the gradient signal in shallow layers becomes stronger as a result of it being copied and sent down the skip connections, avoiding the multiplications it would otherwise undergo when backpropagating through the convolutions. Secondly, in addition to the network learning the weights in the convolution layers, the skip connections allow it to learn to what extent the network wants to use these convolutions in the first place. In the extreme case, the weights within a residual block could converge to all zeroes, in which case the comprising convolutions are not used at all. This increased degree of autodidacticism at least partially explains ResNet's performance boost, which comes with barely any added computational complexity – apart from the negligible element-wise additions – and without additional parameters to train. Another interpretation is the network learning, at each residual block, what should be added or subtracted to the inputs, i.e. a residual block learns the delta to its input instead of changing it directly. For very deep implementations of ResNet, e.g. networks with over fifty layers, computational complexity again becomes a problem. Like GoogLeNet, this is resolved using bottleneck layers, i.e. by reducing the depth dimensionality at the beginning of the block (using a smaller number of kernels than before), and increasing the depth again just before the end of the block (with a number of kernels equal to the input depth). In this way, the core convolutional operations of the block are performed on fewer input channels. This is pictured in Figure 22b.

The authors improved upon the residual block design and created a more direct path for propagation (in both directions) by moving the activations to the residual mapping pathway [40]. Zagoruyko et al. made the residual blocks "wider" by using more kernels instead of making the whole network deeper, also improving performance [97]. Huang et al. stochastically disable a subset of residual blocks from the network for each batch during training, reducing depth and tackling the vanishing gradient problem to a further extent, and then use the full trained network at test time [43]. The same authors also proposed DenseNet in 2017 [42]. Instead of the skip connections routing a residual block's input only to the same block's summation stage, the inputs are additionally being routed to the concatenation stages of every subsequent block, thereby densely connecting the residual blocks. In other words, every residual block now ends in a summation of its own input, residual, and inputs of all its preceding blocks. This alleviates the vanishing gradient problem even further, strengthens feature propagation and encourages feature reuse. ResNet's original authors improved it again in 2017 with ResNeXt, which parallelized the residual pathways within each residual block [95]. Each pathways was made slightly "thinner", i.e. they used slightly fewer kernels, but there are now multiple branching paths, the number of which is the *cardinality* of the network. The writers argue that increasing cardinality gives better performance than increasing either



(a) Featurized image pyramid (b) Single features map (c) Pyramidal hierarchy (d) Feature Pyramid Network

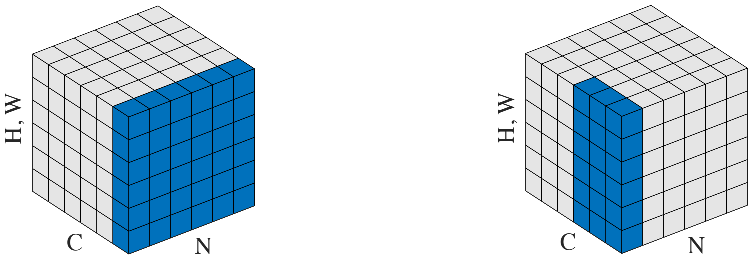
Figure 23: Different usage of a pyramidal structure for feature representation. (a) Ideally, each scale of a feature pyramid is generated from the corresponding image scale, as this would retain localization information at the low resolution scales as well as the higher resolution scales. (b) A depiction of a CNN, which typically predicts only upon the last, lowest resolution feature representation. (c) A pyramidal hierarchy additionally predicts using intermediate representations. This method is used by the single-shot detector [60]. (d) The feature pyramid network as proposed by Lin et al., which employs a top-down pyramid in which each scale is a concatenation of the layer at the previous scale in the same pyramid, as well as the layer at the same scale in the bottom-up pyramid. Reproduced from [56] with permission.

network depth or residual block width.

Further additions and improvements

Up until now the general evolution of CNNs has been discussed. This section treats some more specific instantiations. It is difficult for a network predicting on one feature layer based on one input image to detect both very large and very small objects. In allowing for this spatial prediction flexibility, images and feature representations of multiple different scales can be used. An *image pyramid* is a series of an image at regularly spaced scales. These are common in non-deep learning techniques, using hand-engineered features, e.g. in HOG (histogram of oriented gradients) [24] and SIFT (scale-invariant feature transform) [62]. Operating on such pyramids produces features that are less scale-specific, since an object's change of scale can be offset by a corresponding switch to a different pyramid level. Similarly, using equisized regions on all scales of a pyramid will inspect the image on a range of different granularities. Processing all scales of the image pyramid independently by a CNN will produce a feature pyramid, with feature maps at the same intervals of scales. Lin et al. refer to such a conceptually ideal pyramid as *featurized image pyramids*, but these featurized image pyramids were not used in CNNs due to their computational complexity [56]. Instead, CNNs typically take a single image scale, compute a succession of feature representations at different scales (i.e. the convolutional layers) during forward propagation, and predict on the smallest of these representations, as per a regular architecture shown in Figure 23b. However, the shallow layers are high resolution at the cost of semantic strength, since these layers have local, low-level features. Alternatively, the intermediate feature scales can also be used to predict upon, with those predictions aggregated, as shown in Figure 23c [60]. These two methods are fast, but less accurate than featurized image pyramids. Lin et al. managed to replicate the prediction quality of featurized image pyramids from only a single input scale, using a method that is as fast as a regular CNN [56]. After the regular forward pass of an image, for instance using ResNet, they construct a second pyramid in a top-down fashion, by upsampling, starting from the deepest layer (comparable to fully convolutional networks, treated later on). At each layer of this top-down pyramid, the semantically strong but spatially coarse preceding layer is upsampled. Additionally, it is added in an element-wise fashion to the layer of the feed forward network (the bottom-up pyramid) at the same resolution, whose activations are more accurately localized as it was subsampled fewer times. The latter layer is first convolved by a 1x1 kernel, prior to the addition. The end result is highly similar to a featurized image pyramid – having strong semantics and accurate localization at each level – on which predictions can be made and processing can be done at all scales by other architectures. The pyramid typically contains scales from 1/4$^{\text{th}}$ of the resolution to 1/32$^{\text{nd}}$, with each scale having the same depth. Lin et al. name their proposal *feature pyramid network* (FPN).

The convolutional neural networks discussed thus far are among the category of *backbones*, core CNN architectures on which more complicated networks can be built. In other words, besides inference performed using only these backbones, they can also be used as the core convolutional functionality within more complex architectures. Such advanced architectures are mostly tailored towards a specific computer vision task, and are therefore discussed in the task-specific subsections, namely Section 2.3, Section 2.4 and Section 2.5.



(a) Batch normalization

(b) Group normalization

Figure 24: Three dimensional visualizations of both batch and group normalization. For the former, mean and variance are calculated across the batch dimension (and pixel dimension), separately for each channel. Conversely, group normalization estimates mean and variance within groups of channels, but separately for image in a batch. Reproduced with permission from [94].

Normalization

Activation functions were briefly discussed in Section 2.1. There, we mentioned some activation functions with their advantages and disadvantages. Ideally, an activation function makes sure that its preceding layer gets zero-centered activations (or is at least able to output both positive and negative activations) and passes gradients of sufficient magnitude further down regardless of the value of its input. Both of these ideals can be enforced at the start of the network by zero-centering and normalizing the data during a data preprocessing stage. Zero-centering is done by subtracting the mean value from all sample features in each dimension/channel, and normalization is done by dividing each such dimension by its standard deviation.[*] The resulting data has a unit Gaussian distribution, in which each sample is in the same range and contributes equally during training. During testing, those same means, averaged over the training images, are naturally subtracted again from the test samples.

Data preprocessing increases the chance of a larger gradient with respect to the first couple of layers, but this effect quickly diminishes deeper into the network, as the activations – the transformed inputs – can take on arbitrarily small and large values again, depending on the weights they encounter and are multiplied by. It is desirable to have this constrained distribution of activations within the network as well, as opposed to only at the beginning. In order to force the activations to be unit Gaussian again at different point within the network, batch normalization can be used. Invented in 2015 by Ioffe et al., *batch normalization* is a normalization step that can be applied to a batch of activations at any point in the network, and is usually performed after fully connected or convolutional layers and before an activation function [46].[†] Like data preprocessing, it ensures the activations in the batch to be unit Gaussian distributed:

$$x^* = \frac{x - E[x]}{\sqrt{Var[x]}} \tag{8}$$

This is a differentiable function, and can therefore simply be backpropagated through. The mean and variance are calculated independently for each dimension/channel and across samples in the batch, visualized in Figure 24a. These are then used to normalize each neuron according to the above formula. This constrains the activations to regions of the activation function domain that are sure to backpropagate a gradient of sufficient magnitude (except in the case of regular ReLU). However, this does not necessarily have to be desirable. It could be that a certain degree of saturation works better for some layers. Therefore, batch normalization allows the network to learn the degree of normalization by giving it two additional parameters, γ and β, which are used in an operation inverse to the normalization:

$$y = \gamma x + \beta \tag{9}$$

The network could learn $\gamma = \sqrt{Var[x]}$ and $\beta = E[x]$ to undo the normalization entirely, $\gamma = 1$ and $\beta = 0$ to keep the normalization as is, or anything in between. This form of batch normalization improves gradient flow during backpropagation and allows for higher learning rates. An additional benefit is the regularization effect. *Regularization* combats

[*] In practice, when using deep learning for image processing, the normalization step is often skipped, as RGB values are already comparable in terms of scale and distribution, reducing the urgency of normalization.

[†] In deep learning frameworks or libraries, batch normalization is usually implemented, and can be conceptually thought of, as a separate kind of layer.

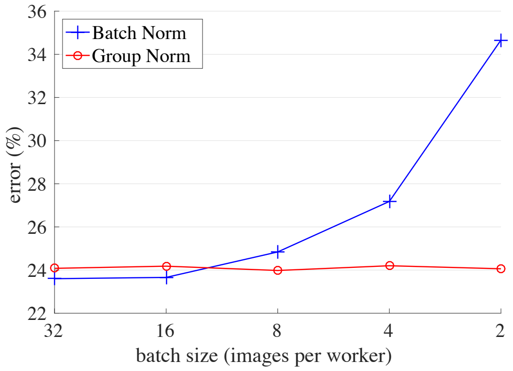


Figure 25: Comparison of prediction error rates between batch normalization and group normalization for different batch sizes. Reproduced from [94] with permission.

overfitting by forcing a machine learning model to be more general. For example, adding an additional term to a loss function which penalizes the model's complexity is a form of regularization, because complex models are more susceptible at capturing undesirable noise in the training data. In this case, batch normalization makes a forward pass during training non-deterministic; the output then depends on the empirical mean and variance of the batch it is shipped in, and this batch is sampled stochastically, which was found to improve generalization. During testing, batch normalization is not performed based on the mean and variance of the current (test) batch, but on some mean and variance acquired during training, e.g. a running averages of values seen during training. These values are then fixed for testing and deployment purposes.

As a sidenote, another popular technique for avoiding overfitting is *pretraining*. A network is then first trained on some general dataset for some general task, after which the resulting parameters are transferred and used as a starting point in another network. This network is then *finetuned*, i.e. trained until convergence, on the task and dataset at hand. Especially the shallow end of the network, where the low to mid-level features are inferred, benefits from this, as these features are often more or less the same for differing tasks. The deep layers often learn the task and subject specific features, and these glsplkernel are most affected by the finetuning. Pretraining not only generally speeds up training, but also usually prevents overfitting.

He et al. mention some problems of batch normalization, and propose an alternative: *group normalization* [94]. According to them, batch normalization requires a sufficiently large batch size. Figure 25 shows how the prediction error increases exponentially each time the batch size is halved. Only after a batch size of approximately 16 does the error rate plateau. For computer vision and other memory-intensive purposes, such a batch size is often infeasible due to memory limitations. He et al. also mention that, during test time, batch normalization uses a static mean and variance calculated during training time. Therefore, the activations in a batch aren't normalized by their own mean and variance, meaning they aren't actually normalized. Group normalization addresses these issues by making the normalization independent of batch size, or the batch dimension. Instead, each image is normalized independently. Its channels are grouped, and normalization is performed within each group. Normalizing within groups offers increased flexibility over normalizing over all channels (a technique called *layer normalization*), since the model is now able to learn a different distribution for each group. Figure 25 shows that group normalization has a consistent error rate over varying batch sizes. He et al. suggest 32 groups of channels.

2.3 Semantic segmentation

Semantic segmentation aims to classify every pixel in an input image, as exemplified in Figure 5e. *COCO* (Common Objects in Context) is a large-scale object detection, segmentation, and captioning dataset, originally developed by Microsoft and containing 330,000 images of which over 200,000 are labeled, covering 1.5 million object instances, 80 object categories, and 91 stuff categories [58]. COCO distinguishes between *things*, countable objects (instances) with a specific size and shape, and often composed of parts, like `car`, `person`, or `elephant`; and *stuff*, homogeneous, amorphous matter or repetitive patterns of fine-scale properties, e.g. `grass`, `sky`, and `wall`.

Additionally, COCO hosts competitions on different computer vision objectives. Even though COCO calls its Stuff Segmentation Task semantic segmentation, it differs from semantic segmentation as defined in this work in that not all pixels

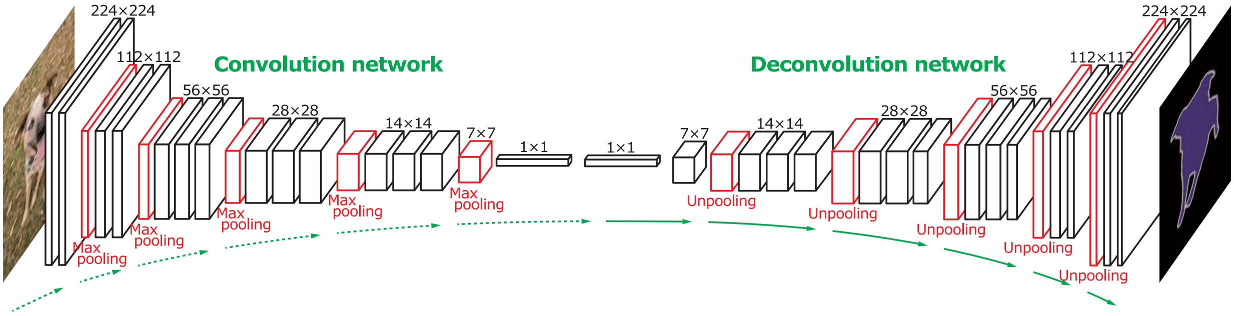


Figure 26: A fully convolutional network. The first half of such a network encodes the image to a vector representation by repeatedly downsampling. Noh et al. used VGG16 as backbone for this part. The latter half then upsampled this feature vector again, until it has the resolution of the input image, but with the depth equal to the number of classes and each channel being a probability map for its associated class. The whole image is being processed at once, as opposed to e.g. a sliding-window approach, which classifies an image pixel-by-pixel. This is faster, as becomes apparent once one views this method as essentially sharing computations between the patches being processed during sliding-window. In that perspective, the overlap between patches constitute the shared computations. Reproduced from [68] with permission.

have to be classified [13]; COCO's thing and stuff classes are inherently mutually exclusive, making it impossible to classify pixels that are supposed to belong to things. Therefore, COCO's Panoptic Segmentation task is closer to semantic segmentation, in which both things and stuff are simultaneously classified. The only difference is that architectures performing panoptic segmentation are supposed to differentiate between multiple instances of the same thing detected in case their pixels are adjacent, whereas this is not necessary for semantic segmentation. However, simply discarding this differentiation between instances will yield results similar to semantic segmentation, as only the pixel classifications remain.

Prior to 2015, deep learning semantic segmentation was performed using a sliding-window approach[*], in which each pixel was classified separately based on a small region centered on it. This is very slow, which is why Long et al. proposed the concept of semantic segmentation using *fully convolutional network* (FCNs) in their seminal paper [61]. Instead of classifying the image pixel-by-pixel based on their surrounding patches of pixels, the whole image is processed as a whole, thereby avoiding redundant computations for overlapping patches. Long et al. achieve this by "decapitating" the fully connected part of any CNN backbone, e.g. AlexNet, VGG or GoogLeNet, and instead upsampling again using bilinear interpolation to the image's original resolution. Therefore, an input image of WxHx3 is processed to yield a "probability image" of WxHxC, where C is the number of classes, meaning each channel in the output is a probability map for the corresponding class. Likewise, each pixel contains probabilities for each class of the pixel belonging to that class. As Long et al. used only a single bilinear interpolation transformation, they call the output results "dissatisfyingly coarse". They mitigated this by merging predictions from shallower layers, but at the cost of increased computational complexity and a more convoluted network. Noh et al. improved the concept by replacing the bilinear interpolation with a full-fledged upsampling part, mirroring the (down)convolutional part [68]. The architecture is shown in Figure 26. Because FCNs essentially first encode their input into a small representation, after which that representation is decoded into the final predictions, FCNs are sometimes termed *encoder-decoder networks*.[†]

Upsampling is done using transpose convolutions. Transpose convolutions are the intuitive opposite of regular convolutions, in which a single scalar activation from a spatially smaller representation is element-wise multiplied by all weights in the layer's kernel, resulting in an enlarged patch with the same size as the kernel. The aggregation of all patches, obtained from transpose convolving all activations, then becomes the upsampled layer, with overlapping areas of patches being summed. The process is visualized in Figure 27. The network is more accurate, with a significant speed-up during test time: both forward and backward propagation are approximately five times faster than AlexNet [61].

In Section 2.1, we briefly discussed the cross entropy loss. Semantic segmentation pipelines can be evaluated using this loss by applying it to the predicted probabilities and one-hot ground truth of every pixel in the segmented image. Especially in the field of bioinformatics, however, other loss functions have gained traction which are better suited for segmentation problems with a severe class imbalance. Dice loss was conceived to combat the class imbalance present

[*] Though we note there are other approaches that do not rely on deep learning, e.g. approaches based on selective search.

[†] However, this term is also applied to neural networks that are not CNNs, with examples of this usage common in natural language processing (NLP).

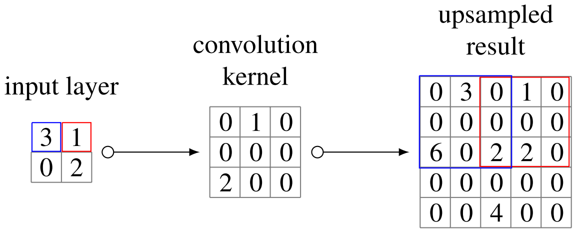


Figure 27: A transpose convolution with kernel size 3 and stride 2. It can be thought of as a convolution in reverse, where now an input pixel is multiplied by each element in the convolution kernel, obtaining the activation patch in the upsampled feature layer, centered at the same location as the input pixel. Overlapping pixels between upsampled patches are summed.

in semantic segmentation of brain lesions, which tend to only take up a minuscule fraction of a 3D brain scan. Such an imbalance will invariably bias the network towards the majority class(es). Dice loss is based on the Dice similarity coefficient or Dice score coefficient (DSC) [79], which measures the similarity between ground truth and prediction segmentations. The Dice coefficient is named after its creator: Lee Raymond Dice. For binary classification, DSC can be calculated using:

$$DSC = \frac{2|\hat{Y} \cap Y|}{|\hat{Y}| + |Y|} = \frac{2TP}{2TP + FP + FN} \tag{10}$$

where $|\hat{Y}|$ and $|Y|$ are the number of predicted and ground truth foreground pixels, and $|\hat{Y} \cap Y|$ is the number of pixels on which prediction and ground truth agree. However, this function is not differentiable, as it uses discrete values. Therefore, a continuous alternative can be used as loss function for binary classification [66]:

$$DSC = \frac{2\sum_p^N \hat{y}_p y_p}{\sum_p^N \hat{y}_p^2 + \sum_p^N y_p^2} \tag{11}$$

where $\hat{y}_p$ is the binary prediction and y_p the ground truth of pixel p. Negating this formula yields a valid loss function, though only for binary classification. Another formulation from [83] is as follows:

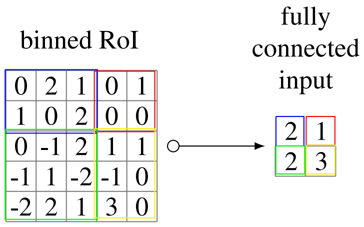
$$DSC = \frac{\sum_p \hat{y}_p y_p + \epsilon}{\sum_p \hat{y}_p + y_p + \epsilon} - \frac{\sum_p (1 - \hat{y}_p)(1 - y_p) + \epsilon}{\sum_p 2 - \hat{y}_p - y_p + \epsilon} \tag{12}$$

The smoothing term ϵ is used to avoid division by zero in the case none of the classes is either present in the ground truth or predicted.

Back in 2006 already, Crum et al. proposed the generalized Dice score (GDS) as a way of evaluating multiple class segmentations (naturally not within the context of deep learning image segmentation) [23]. GDS was adapted into the generalized Dice loss (GDL) by Sudre et al. in 2017 [83]:

$$GDL = 1 - 2\frac{\sum_l w_l \sum_p \hat{y}_{p_l} y_{p_l}}{\sum_l w_l \sum_p \hat{y}_{p_l} + y_{p_l}} \tag{13}$$

where $\hat{y}_{p_l}$ and y_{p_l} are the ground truth and predicted probability of label l at pixel p, and $w_l = 1/(\sum_p y_{p_l})^2$ is the class weight of label l, inversely proportional to the square of its area size. This loss function resembles the reference metric we use to grade the overall quality of a segmentation architecture: the *mIoU* metric. It averages over all classes the ratio of correctly predicted pixels to the union of predicted and ground truth pixels of that class. GDL will optimize this metric directly, whereas cross entropy, though easier to optimize, only serves as a "proxy" towards the real objective of optimizing mIoU. Attentive readers will note that the function actually poses a closer resemblance to the F1 score. Since F1 and the mIoU are positively correlated, and maximizing one necessarily maximizes the other, this does not have any practical implications [92]. We suspect that the F1 formula is easier to optimize than that of mIoU, hence to choice to represent GDL with the F1 formula.



0	2	1	0	1
1	0	2	0	0
0	-1	2	1	1
-1	1	-2	-1	0
-2	2	1	3	0

2	1
2	3

Figure 28: RoI pooling. A proposed region is subdivided into a fixed number of bins, of which the boundaries are rounded. Within each bin, only the highest activation is kept and transmitted to the fully connected layers.

2.4 Instance segmentation

Instance segmentation (Figure 5d) performs object detection (Figure 5c), but instead of provided bounding boxes, it uses pixel-wise masks in order to denote an object's spatial extent. Within COCO, instance segmentation is filed under the Object Detection Task, but they distinguish between object detection using bounding boxes and object detection using segmentation masks.

A highly effective instance segmenting architecture is Mask R-CNN, proposed by He et al. in 2017 [37], influencing multiple successors which have since exceeded it in performance [59]. It is the culmination of a sequence of works, each building off of its predecessor.
It's conception originated in 2014 with R-CNN (region-based CNN) [32]. R-CNN, like its two successors, is an object detection architecture, displayed in Figure 30a. It starts by executing a fixed, deterministic algorithm – i.e. this alsogirthm is not learned and is not altered during training – which is required to suggest *region-of-interest* (RoI) proposals. The algorithm in question is called selective search*, and it iteratively merges clusters of pixels that are similar in colour or texture [88]. More precisely, it takes an input image and segments it into relatively fine-grained groups of texturally similar pixels. Repeating this process, it acts upon this segmentation and merges groups instead of pixels. This process is iterated until a certain, manually specified stop condition is met, which decides upon the granularity of the clustering. The bounding boxes of these segmentations are then returned as RoI proposals, which are all separately processed by a CNN. Like most CNNs, the CNN used in R-CNN expects a fixed input size.† Therefore, the RoI proposals need to be rescaled to be of uniform resolution. This is done using an affine warping technique. Afterwards, the warped image regions are condensed by a CNN to a small feature representation, which is then classified using category-specific linear support vector machines (SVMs) [22, 87]. The authors use SVM loss function on the scores, and additionally linearly regress on the bounding box coordinate offsets of the original proposal regions. The aim is to provide the network with the ability to alter the bounding box, in case selective search offers an erroneous demarcation, e.g. a human head is excluded while detecting humans, while the network has learned that humans, more often than not, have heads.

Downsides to R-CNN are slow training time, slow inference, and high memory requirements. R-CNN's selective search typically suggest around 2,000 RoI proposals, each of which have to be processed by a small CNN. To that end, one of the authors enhanced the model, calling it Fast R-CNN [31]. He resolved the issue by processing the whole image by a (larger) CNN, with selective search acting on top of the resulting feature representation, demonstrated in Figure 30b. The RoIs now do not need to be separately convolved and warped, but are only pooled by an *RoI pooling layer*, since the fully connected part still expects a fixed size input. RoIs are subdivided into bins, with the number of bins equalling the input size to the fully connected part. In each bin, only the maximum activation is supplied to the fully connected input layer:

Bin boundaries are rounded to the nearest integer. The fully connected part yields softmax classification probabilities and bounding box offsets for each class, the latter of which is a four-tuple ((x, y, w, h)) containing scale-invariant translation and log-space height/width shift relative to an object proposal [33]. Each training RoI is labeled with a ground truth class and a ground truth bounding box regression target. For backpropagation, Girshick uses a multitask loss, unifying the softmax loss and regression loss [31]:

* Girshick et al. note that other region proposal algorithms are equally feasible: objectness [3], category-independent object proposals [27], constrained parametric min-cuts (CPMC) [14], multi-scale combinatorial grouping [4], and mitotic cells [20] could replace selective search.

† CNNs that produce variable output dimensions, e.g. FCNs, are the exception.

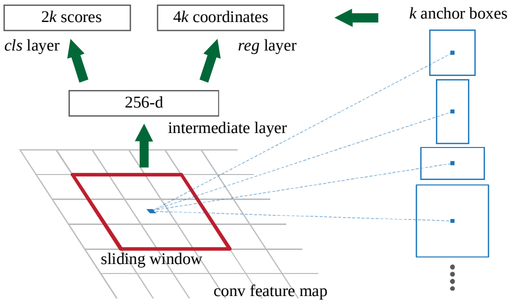


Figure 29: The region proposal network (RPN). At every spatial location in the feature layer generated by a backbone architecture, a single forward pass predicts the objectness and regressed coordinates of k candidate region proposals, or anchors. The location's surrounding 3x3 square patch is first convolved into a 256-dimensional feature layer, after which two parallel fully connected layers predict the objectness and coordinate regressions. Objectness is represented with two output nodes with probabilities for the anchor containing an object of interest and it not containing an object of interest. Reproduced from [72] with permission.

$$\underbrace{-\log p_u}_{\text{true class log loss}} \underbrace{+\lambda \sum_{i\in\{x,y,w,h\}} \text{smooth}_{L1}(t_i - v_i)}_{\text{true class bounding box regression loss}}, \text{ where } \text{smooth}_{L1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| \le 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{14}$$

Fast R-CNN is approximately ten times as fast as regular R-CNN, both during training and testing. In fact, Fast R-CNN's test times are bottlenecked significantly by selective search. In order to improve test time even further, Ren et al. omit the fixed region proposal algorithm, and instead additionally learn a *region proposal network* (RPN) during training [72], which proposes RoIs instead of a fixed algorithm using a sliding-window approach. They aptly titled their network Faster R-CNN, of which a diagram can be found in Figure 30c. Apart from the duty of proposing regions being handed to the RPN, Faster R-CNN is the same as Fast R-CNN. Like Fast R-CNN, the input image is first convolved into a feature layer, on top of which the classification and regression of proposed bounding boxes, and now also the generation of region proposals itself, is effectuated. Each of RPN's region proposals has an associated objectness score; the network's estimated probability of the region containing an object of interest. A threshold on this objectness score decides which proposals will be predicted upon, and which will be discarded. Ren et al. investigated two backbones for convolving the input image into the feature layer: ZFnet [98] and VGG-16 [82]. The proposals are obtained by sliding a 3x3 window – whose receptive field on the input image is naturally much larger – over the feature layer. At each spatial location of the feature layer, this 3x3 window is convolved by a 3x3 convolutional layer to either a 256-d feature, when working on ZFnet's feature layer, or a 512-d feature, when working on VGG-16's feature layer. This feature is in turn fed to a 1x1 convolutional layer in order to produce two sibling fully connected layers. The first fully connected layer predicts two probabilities for the window containing an object and for the window not containing an object.* The second fully connected layer predicts four scores – two spatial coordinates and two box dimensions – which together predict a regression off of the default sliding window bounding box into a more suitable bounding box. The architecture of the small network evaluating the sliding windows is shown in Figure 29.

At each sliding window location, not only one square candidate region proposal, but multiple candidate region proposals, called *anchors*, are evaluated. Ren et al. evaluate three aspect ratios (1:1, 1:2 and 2:1) at three different sizes (with area sizes 128^2, 256^2 and 512^2), resulting in nine different anchors being evaluated per sliding window, all simultaneously in the same network. This means that the objectness classification and bounding box regression fully connected layers actually have $2 * 9$ and $4 * 9$ outputs, respectively. For a convolutional feature map of width w and height h, a grand total of $wh9$ anchors are evaluated. Ren et al. contrast this method to other schemes that propose regions at different scales and sizes. One could use a featurized image pyramid, where each feature layer is built from scaled input, and run the RPN on each layer. Alternatively, one could scan an input image of a single scale with multiple sliding windows of different sizes and scales. Both methods are inefficient, as they require multiple passes of sliding a window, either on different input images or with different windows. The latter methods also entails training multiple different small convolutional network, one for each size and aspect ratio. The RPN evaluates multiple anchors at each location in parallel, and thus

*Ren et al. mention that objectness could also be represented with one probability, using logistic regression. This two-class output was chosen for simplicity.

requires going over the feature layer only once. Ren et al. note that their RPN is translation invariant; if the network picks up on an object in a certain image, then the network will always detect this object regardless of its exact location. One could Photoshop the object to a different spatial location, and the network will still guarantee its detection.[*]

In training the RPN, each anchor is assigned a binary label denoting whether or not it contains an object of interest. An anchor's label is positive if its IoU with any single ground truth box is larger than 0.7, or if it is (one of) the anchors with the highest such IoU. This second condition is employed to catch the rare case that the first does not yield sufficient positive anchors. A negative label is assigned to an anchor if there is no ground truth box with which it has an IoU larger than 0.3. The total loss then becomes:

$$L(\{\hat{p}_i\}, \{\hat{\mathbf{t}}_i\}) = \underbrace{\frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(\hat{p}_i, p_i)}_{\text{classification loss}} + \lambda \underbrace{\frac{1}{N_{\text{reg}}} \sum_i p_i L_{\text{reg}}(\hat{\mathbf{t}}_i, \mathbf{t}_i)}_{\text{regression loss}} \tag{15}$$

where i is the index of an anchor in a batch, $\hat{p}_i$ is its predicted objectness, p_i is its ground truth (1 if it is a positive anchor, 0 otherwise), $\hat{\mathbf{t}}_i$ is a vector of its four parameterized predicted regression coordinates, $\mathbf{t}_i$ are the parameterized coordinates of the ground truth box associated with it if it is a positive anchor. The parameterizations of the predicted box coordinates $\hat{t}$ and ground truth box coordinates t are carried out as follows:

$$\begin{aligned} \hat{t}_x &= (\hat{x} - x_a)/w_a & \hat{t}_y &= (\hat{y} - y_a)/h_a \\ \hat{t}_w &= \log(\hat{w}/w_a) & \hat{t}_h &= \log(\hat{h}/h_a) \\ t_x &= (x - x_a)/w_a & t_y &= (y - y_a)/h_a \\ t_w &= \log(w/w_a) & t_h &= \log(h/h_a) \end{aligned}$$

where x and y are a box's center's coordinates, h and w are its height and width, and $\hat{x}$, x_a and x denote predicted, anchor, and ground truth values respectively (similarly for y, h and w). The classification loss L_{cls} takes the log loss over the two classes. The regression loss $L_{\text{reg}}(\hat{\mathbf{t}}_i, \mathbf{t}_i) = R(\hat{\mathbf{t}} - \mathbf{t})$, where R is the robust loss function (smooth L_1), defined in 14.

As noted before, on each image with width w and height h there exist $w \cdot h \cdot 9$ possible anchors. Approximately 70% of these anchors cross the image boundaries, and such anchors are ignored, since they inhibit training to converge. As a result only around $0.3 * wh9$ anchors remain.[†] The majority of these will be negative samples, and in order to avoid network bias, Ren et al. sample 128 positive and 128 negative samples. If there are less than 128 positive samples, then the remainder is padded with negative samples.

The RPN is likely to propose many highly overlapping regions. Ren et al. apply non-maximum suppression to proposals with at least 0.7 IoU with the same ground truth bounding box. In other words, for a group of proposals that all have 0.7 IoU or larger with the same bounding box, only the proposal with the largest IoU is kept. Finally, only the top-N ranked proposals (in terms of IoU) are being passed to Fast R-CNN for detection.

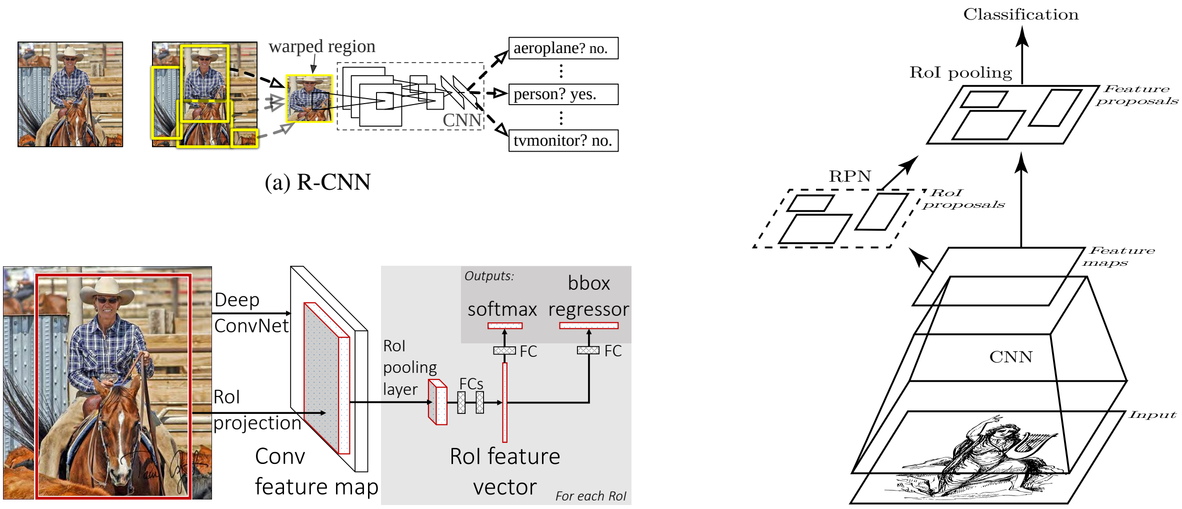
Training is done by first training RPN, then using the resulting proposals to train Fast R-CNN. Up until this point, the backbone convolutional layers are not shared and have been trained separately (initialized using parameters pretrained on ImageNet), and therefore this finetuned Fast R-CNN network is used to initialize RPN's backbone, though the backbone convolutional layers are now fixed and only the mininetwork is trained. The backbone is now shard between the two networks, and after the RPN mininetwork is finetuned, we lastly finetune Fast R-CNN unique layers.

Faster R-CNN brings object detection to interactive speeds: 0.2 seconds test time during Ren et al.'s experiments. Lin et al. note that the RPN can be implemented using an FPN [56]. In that case, the RPN simply slides equisized windows across the different scales of the FPN, effectively proposing regions of different sizes.

Mask R-CNN is an extension of Faster R-CNN, transforming it into an instance segmentation architecture rather than object detection [37]. The idea is conceptually simple: Figure 31 shows how, for each RoI proposal, a branching FCN predicts the segmentation mask of the object that is ostensibly present in the proposed region, parallel to the already exist-

[*] Though this property only holds up to the networks total stride, as is the case with all FCNs. In other words, if a detected object is translated to an unlucky position between strides, it is not guaranteed to be detected again.

[†] During testing, all anchors are considered, and overlapping prediction boxes are clipped.



(a) R-CNN

(b) Fast R-CNN

(c) Faster R-CNN

Figure 30: Developments towards Mask R-CNN. (a) R-CNN processes each of the RoI suggested by selective search by a separate CNN, after the RoI has been warped to a fixed size. (b) Fast R-CNN first processes the entire image using a CNN, after which selective search proposed RoIs on top of the resulting features representation. (c) Faster R-CNN outsourced the RoI selection to an RPN, a separate model that is now trained simultaneously along with the rest of the architecture. (a) and (b) reproduced from [32] and [31], respectively, with permission.

ing classifier branch.* Each RoI therefore gets class probabilities, bounding box offsets and a binary segmentation mask, indicating which pixels belong to it. However, the rounding of the bin boundaries during RoI pooling causes misalignment between the input pixels and the mask output pixels, which was not a problem for object detection. The authors therefore replace the RoIPool layer with the sample based RoIAlign. Four bilinear interpolation values are computed at four regularly spaced points within each bin, which are aggregated using either an average or the maximum. The avoidance of rounding mitigates the misalignment.

At the time of writing, six architectures of the top 7 in the COCO Object Detection Task (using segmentation) leaderboard are based on Mask R-CNN. Many use the combination of Mask R-CNN with FPN as backbone. Mask R-CNN itself currently holds the ninth place. The frontrunner is Sun et al.'s FishNet [84], a fish-shaped network that can be found in Figure 34. A CNN backbone, e.g. ResNet, constitutes the tail of the fish. The fish body upsamples its downsampled output again, in order to get strong semantics at a higher resolution, after which these are downsamples again by the fish head. Additionally, each layer in the network is concatenated along the depth dimension with the first preceding layer that has the same resolution. Sun et al. state that leveraging the complementarity of feature maps at different locations in the architecture improves the diversity of those features. They moreover point out that the skip connections in ResNet and ResNeXt are not actually identity mappings, but rather, what they call, I-convs. These I-convs are 1x1 convolutions in the skip connections, meant to change a layer's depth and make input and output layers of a residual block compatible for element-wise addition. Sun et al. claim that these hinders the gradient flow during backpropagation. They avoid I-convs, and therefore allow for an unhindered and unaltered gradient flow over the skip connections. Sun et al. note that FishNet is a framework, which does not specify the building block it consists of. They used both ResNet and ResNeXt's residual

*Using only one extra output and corresponding loss in the classification branch, Mask R-CNN can additionally be trained to perform pose estimation à la Figure 6b. Each vertex in the pose skeleton is then a one-hot binary mask which is additionally predicted. We also note that COCO has the Keypoint Detection Task for pose estimation.

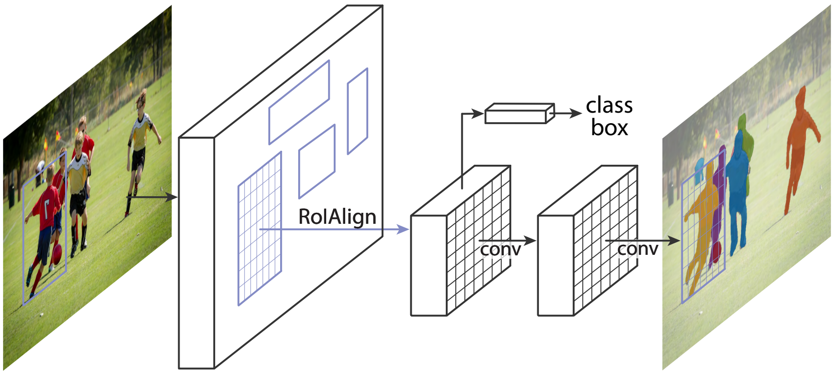


Figure 31: The Mask R-CNN process, clarifying the branching structure. The first half and top branch are essentially the same as Faster R-CNN, but Mask R-CNN adds the FCN branch additionally predicting segmentation masks for each RoI. Reproduced from [37] with permission.

blocks in their FishNet, with the latter combination termed FishNeXt.

All of these detectors and instance segmentators belong to the family of *two-stage* detectors, since they consists of two stages: first a sparse set of candidate locations are proposed, after which these are classified (as a class or as background) and their bounding boxes further regressed. Another family of object detectors is that of the one-stage or *single-shot* detectors. Such detectors do not bother with first proposing promising regions, and instead directly predict on a regular, dense grid of locations, scales and aspect ratios. One of the first such deep learning one-shot detectors was 2013's Overfeat [78], which has since been superseded by 2016's YOLO (short for You Only Look Once) [71] and SSD (for Single Shot Detector) [60]. YOLO and SSD are faster than state-of-the-art two-stage predictors, but trail behind in terms of accuracy. Their ideas are similar: an input image is divided into an $n \times n$ course grid, and each grid cell center has b bounding boxes of different scales and aspect ratios centered on top of it. From each bounding box, five regression coordinates (dx, dy, dh, dw, confidence) predict the offset of a potential object's true location. For each grid cell, class scores are predicted for each of c classes that estimate how likely the respective class is present at that grid cell. The network therefore predicts $n^2(5b + c)$ outputs in one "shot".

Lin et al. investigated why such detectors lack in accuracy [57]. They found the problem to be the large amount of easily classifiable background proposals that result from the regular sampling, and propose a dynamically scaled cross entropy loss function whose scaling factor decays to zero as confidence in the ground truth class increases. In other words, it decreases the contribution of the easy background samples, thereby obstructing it from polluting the loss. Their focal loss is defined as:

$$L_{\text{FL}}(p_t) = -\log(p_t)\alpha_t(1 - p_t)^\gamma \tag{16}$$

where p_t is either the predicted probability p when predicting the ground truth class or $1 - p$ when predicting the background class[*], $-\log(p_t)$ is the familiar cross entropy loss, γ is a hyperparameter, and α_t is a weighting factor defined as some inverse of class frequency α and analogously to p_t, $\alpha_t = 1 - \alpha$ for the background class. When p_t is small, i.e. the network is unsure of the correct class, then the modulating term $(1 - p_t)^\gamma$ is close to 1 and the loss is barely affected. For easy samples with a large p_t, the modulating term will be small and the loss will be diminished. The focusing hyperparameter γ determines the magnitude of this effect. Whith $\gamma = 0$, focal loss is equal to cross entropy. A loss curve for different values of γ is shown in Figure 32. Lin et al. found $\gamma = 2$ to be optimal, and add the extra α-weighting because it provided improved results over the loss without the α term.

Focal loss's effectiveness is demonstrated by using it in Lin et al.'s one-shot detector: RetinaNet. RetinaNet consists of an ResNet-FPN [56] backbone, on top of which two branching subnetworks perform classification and regression, as seen in Figure 33. The feature pyramid has levels P_3 through P_7, where level P_l's resolution is 2^l times as small as the input resolution. Each level has depth 256.[†] The regular sampling of the grid is highly similar to that of RPN, but windows are

[*] Lin et al. explain their focal loss in a binary classification context, but note that extending it to a multinomial setting is straightforward.

[†] This feature pyramid is slightly different than the original pyramid proposed in [56]: the highest resolution level P_2 is omitted for computational reasons, level P_6 is introduced and computed by a 3x3 stride-2 convolution instead of downsampling, and level P_7 is also introduced and calculated by applying ReLU and a 3x3 stride-2 convolution on P_6 in order to improve large object detection. These modifications improve speed while maintaining accuracy.

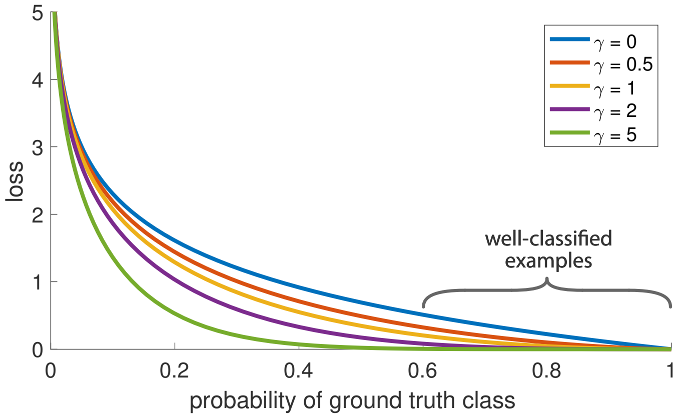


Figure 32: The focal loss for different network certainties and different values of γ. The blue line ($\gamma = 0$) is the regular cross entropy loss function, which has a more even distribution of loss values across certainties. Whenever there are many easy-to-classify examples, these can overwhelm the total loss of a batch. Setting γ to larger values will downweight the loss of easy samples more, which gives more room for the loss of hard examples, were the potential for learning lies. Reproduced with permission from [57].

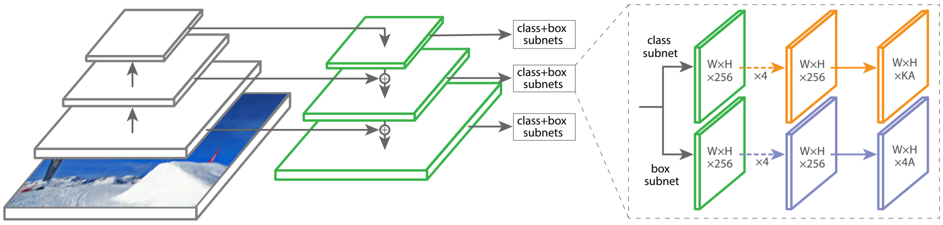


Figure 33: The network design of RetinaNet Reproduced with permission from [57].

naturally assigned to all levels of the pyramid. The receptive fields of these windows are 32^2, 64^2, 128^2, 256^2 and 512^2 for the respective levels. For each window, the same three aspect ratios are evaluated, and again at three different sizes: 1, $2^{\frac{1}{3}}$ and $2^{\frac{2}{3}}$ times the window size, resulting in 9 anchors per window as per regular FPN. Because RetinaNet samples anchors both at different levels and different sizes per level, the scale coverage is even more dense than regular RPN.

In comprising the training set, each anchor is associated to the ground truth box with which it has the largest IoU, as long as this IoU is larger than 0.5. It is assigned a one-hot encoded class and a 4-vector of box regression targets, which are derived from said box. If there is no ground truth box with which its IoU is 0.4 or larger, it is designated as background, and the class and regression targets become irrelevant.* The classification subnetwork consists of four 3x3 convolutional layers, each followed by ReLU, and a 3x3 convolution with $9c$ filters (one filter for each class and each anchor) followed by sigmoid or softmax activations. That is, each spatial location in the final output map contains c class scores for the 9 anchors centered around that location. The network is the same for each pyramid layer, including parameters. The regression network, though completely separate from and parallel to the classification network, is highly similar apart from the fact that it ends with $4 \cdot 9$ regression coordinates per spatial location. To improve speed, Lin et al. only decode box predictions from at most the 1,000 top-scoring predictions per FPN level, after thresholding detector confidence at 0.05. These top predictions from all levels are merged and non-maximum suppression with a threshold of 0.5 is applied to yield the final detections.

2.5 Panoptic segmentation

Panoptic segmentation, coined by Kirillov et al. [49], unifies the tasks of semantic segmentation and instance segmentation. At the time of writing, of all the currently ranked contenders of the Panoptic Segmentation Task, only one published a technical report [26]. The others either submitted proprietary architectures or have not published their work. Moreover, the state-of-the-art methods on this joint task use separate, architecturally distinct networks for both tasks, and simply aggregate the results. Recently, however, Kirillov et al. proposed Panoptic FPN [48], though it is not listed in the Panoptic

*All anchors which have an IoU between 0.4 and 0.5 with any box are ignored during training.

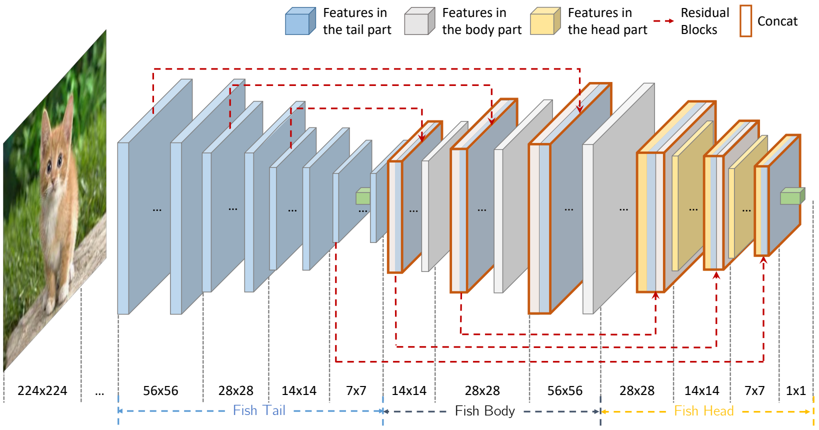


Figure 34: FishNet. The fish tail is a CNN backbone, like ResNet or ResNeXt. The fish body takes the condesned features representation and upsamples it. The fish head, downsamples the representation again, for stronger semantics and more features refinement. The dotted lines indicate skip connections, which, in contrast with the skip connections in ResNet and ResNeXt, are pure identity mappings, allowing for unhindered gradient flow. The layers with red outlines indicate the layers that are involved in these skip connections. These layers use concatenation to combine their previous layer in the network with the first preceding layer having the same resolution. Reproduced with permission from [84].

Segmentation Task leaderboards. Panoptic FPN aims to solve both tasks using a single network, namely Mask R-CNN with an FPN backbone (with this FPN backbone in turn having a ResNet/ResNeXt backbone). They show a synergistic effect which causes both semantic segmentation and instance segmentation to benefit from one another.

Similarly to how Mask R-CNN was birthed by adding a branch after the region proposal network, Panoptic FPN in turn was conceptualized by adding a branch, this time for semantic segmentation, to Mask R-CNN. In this case, it is positioned right after the FPN backbone and before the RPN (which is only necessary for instance segmentation). Figure 35 shows how Panoptic FPN uses the four scales resulting from FPN for the two respective tasks. Instance segmentation remains practically unchanged, with Mask R-CNN's region proposal network working on top of the feature pyramid. For semantic segmentation, all pyramid scales (with the exception of the largest) are upsampled to all have resolution equal to the largest scale. The resulting equisized feature maps are element-wise summed, after which they are processed one time – using a 1x1 convolution, 4x bilinear upsampling and softmax – to yield the final dense segmentation prediction. In addition to stuff classes, this branch also outputs a special wildcard class for all pixels belonging to things in order to avoid overlapping output with the instance segmentation branch. Any remaining overlap is resolved by favouring instance segmentations. The Panoptic FPN network without the instance segmentation branch – which then only does semantic segmentation – is given a separate name: Semantic FPN.

Kirillov et al. achieve semantic segmentation performance comparable with current state-of-the-art semantic segmentation methods, e.g. the heavily-engineered and performance enhanced DeepLabv3+ [15, 16, 17]. They achieve state-of-the-art results with only half the computational complexity. With a network equalling the state-of-the-art computational complexity, Panoptic FPN improves upon the performance otherwise acquired by two separate networks by a 1.1 PQ on the COCO dataset. The authors emphasize the modularity of their network, and stress that other backbones can be used as well.
The effects of joint training of instance and semantic segmentation on instance segmentation was also investigated, and in doing so Kirillov et al. employed a loss scaling weight λ_s for the instance segmentation branch. They found that the optimal scaling weight of $\lambda_s = 0.1$ offered only a 0.1 increase in AP on COCO. Interestingly enough, when testing on Cityscapes [21] – a dataset with high similarity to CycloMedia's data – the optimal weight is $\lambda_s = 1$, in which case the semantic segmentation loss counts as severely as the instance segmentation loss in the combined loss. What's more, simultaneously training semantic segmentation increases instance segmentation's AP by 1.

2.6 Material segmentation

In *material segmentation*, the goal is to classify every pixel in an input image from a predefined set of material classes. It is a specific subtask of the more general semantic segmentation, previously discussed in Section 2.3, where the classes do

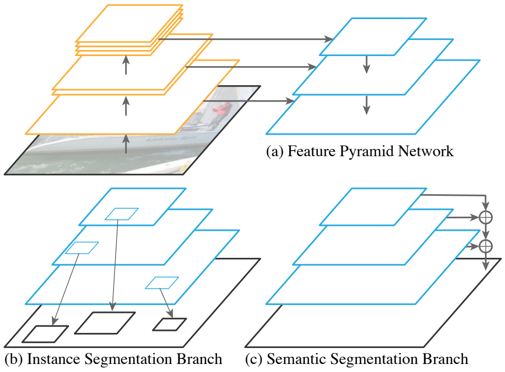

(a) Feature Pyramid Network
(b) Instance Segmentation Branch
(c) Semantic Segmentation Branch

Figure 35: Panoptic FPN. As the name suggests, this network uses FPN as its backbone, which gives it a featurized image pyramid – four feature maps at different scales – to work with. The instance segmentation is nearly identical to Mask R-CNN, accept it now predicts on four feature maps instead of one. The semantic segmentation branch, called Semantic FPN, merges the four scales and convolves the merger into a prediction map.

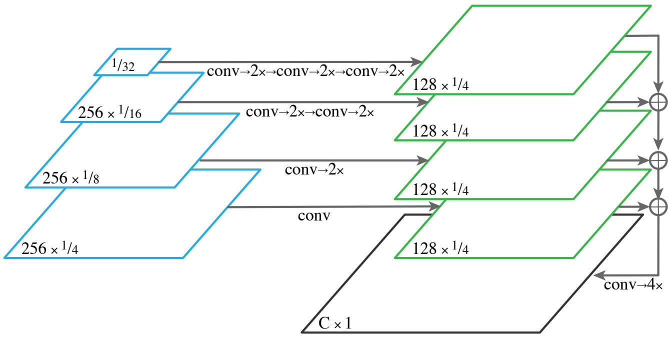


Figure 36: Semantic FPN, Panoptic FPN's semantic segmentation branch, in more detail. The smallest three feature maps are iteratively convolved with a 3x3 glskernel, batch normalized, ReLU activated, and bilinearly doubled in resolution, until the smallest three scales have the same resolution (128x128) as the largest scale. The equisized scales are subsequently element-wise summed, after which a final 1x1 convolution yields the segmentation map.

not necessarily have to be and typically are not materials, but can be any sort of perceptible attribute. This is a difficult endeavor, because wholly different materials may look visually similar in RGB images. An extreme examples is that of fake, plastic food and real, organic food, which tend to be difficult to distinguish, especially in an image.

Bell et al. employ a *sliding-window approach* for segmenting an image, in which every pixel in the image is classified by a CNN taking only a small region, or patch, around it as input. For training, they introduce the Material in Context Database (MINC), which contains three million images with material annotations. These annotations come in the form of segments, polygons demarcating materials, and clicks, single pixels for which a material is distinguished. In training the CNN, clicks are converted to training patches by centering the patch on the click point, and segments are converted into approximately nine patches by using Poisson-disc subsampling. Prior to training the CNN (or rather *finetuning* it, which takes the parameters from a model previously trained and adapts those to the dataset at hand), it was first pretrained on ImageNet. The writers ended up with 2,996,674 labeled patches, generated from 436,749 images.

During test time, they slide this trained CNN over an input image at three different scales, producing three probability maps that are upsampled and averaged together. Lastly, a fully connected conditional random field (CRF) predicts a final label for each pixel. They achieve 73.1% mean class accuracy (which calculates the error rate over all pixels) when *ensembling*, or combining, AlexNet and GoogLeNet. The main drawback of this work is its use of the sliding-window approach, which results in comparatively bad segmentation quality and slow compute, and has been superseded by fully convolutional network [30].

Bell et al. conclude that including the surrounding context and object shape is essential for material classification. However, Schwartz et al. doubt this claim, and argue that "we may only speculate as to the actual importance of context when simply given large patches as input and materials as output" [77]. Since Bell et al. use a highly specific dataset (mainly consisting of professional real-estate images), Schwartz et al. argue that Bell et al.'s network instead recognizes objects that are often made out of the same material, and classify the object with that material. In other words, they merely perform a semantic segmentation variant of object detection, but with the labels referring to materials instead of objects. Schwartz et al. agree that global context is important, as context may influence the probability distribution over possible materials, but claim this needs to be processed separately from the appearance of the material. Therefore, their work takes only small patches within objects' boundaries, ensuring shape and surroundings are not taken into account. They concatenate the penultimate feature layer at the end of their VGG16-based FCN with two additional feature layers containing the context. The first is a semantic segmentation – of which Schwartz et al. do not specify the network whence it came – with pixel-wise object predictions. The second is a probability distribution over the type of location the image was taken, acquired with the MIT Places CNN [102] and duplicated for every pixel, yielding a homogeneous feature layer. In this way, context information that Schwartz et al. deem essential is taken into account, but only after prior distributions

are calculated based only on materials' visual appearance. Schwartz et al. tried the introduction of context at multiple levels, and found that, indeed, introduction at the penultimate layer provided the highest accuracy. They also demonstrate that both object and location context provide more accuracy compared to either of the isolated contexts accounted for singularly, and compared to not taking context into account at all. Since all data in our research depicts the same type of location, i.e. cities, municipalities or highways, this result is less relevant to this work. More importantly, their fully convolutional network with context outperforms the patch-based network by Bell et al., even though the latter was pretrained on the three million samples of the MINC database whereas the former did not have this advantage.
Zhang et al. perform material segmentation on hyperspectral images [100].

More recent developments make use of *time-of-flight* (ToF) cameras for material segmentation. Such cameras measure the time is takes for a particle or wave to bounce back from a surface and use that time to infer the distance of that surface. This is done for every point in an image to obtain a depth map. Laser-based ToF is part of the broader class of scannerless[*] LiDAR sensors. Tanaka et al. exploit the fact that the measured depth of translucent objects are overestimations. Phrased differently, a time delay of the traveling signal incurred because of subsurface scattering, the depth of (partially) translucent surfaces is often erroneously predicted as being further away than they are in reality.

Hyperspectral images capture radiation across the entire electromagnetic spectrum, as opposed to merely within the visible range. Since this work concerns regular RGB images, [100] is noteworthy, though not relevant.

2.7 Datasets for material and instance segmentation

Table 1 gives a non-exhaustive overview of currently prominent datasets for material segmentation and instance segmentation. Most recent of these is the Natural Environment Dataset [5].[†] It was generated by Baslamisli et al. from 3D models depicting gardens and parks with the aim of *intrinsic image decomposition*. This entails separating an RGB image into what Baslamisli et al. call an albedo, or reflectance, map and a shading map. The former approximates the diffuse surface reflectance, or intuitively, the "true colour" of a surface. The latter indicates, confusingly, the reflectivity of the objects in the image. For each pixel, this depends on the direction of incoming light towards to surface, the direction of outgoing light from the surface to the camera sensor, the surface normal, and the surface's *bidirectional reflectance distribution function* (BRDF): the ratio of incoming irradiance to outgoing radiance. Using this data, they train a network in order to train this separation; they employ a three-headed FCN called ShapeNet [80], each predicting a reflectance map, a shading map, and a semantic segmentation, respectively. An overview of the architecture is shown in Figure 37. They claim that jointly learning the reflectance and albedo with semantic segmentation benefits the latter. The dataset itself was rendered using Blender, an open-source 3D modeling suite. However, the datasets' main subjects being gardens and parks renders it rather unsuitable for our purposes.

Another noteworthy dataset is Mapillary's Vistas dataset [67]. It contains 25,000 high-resolution images depicting street scenes, similar to CycloMedia's cyclorama's. It is used to perform panoptic segmentation, and since it concerns things and stuff, but not materials, also this dataset is ill-purposed for material segmentation. However, since it contains both things and stuff, it is highly suitable for panoptic segmentation. Cityscapes is similar, containing the same amount of images, also from street-level viewpoints [21]. This dataset contains fine segmentations for only 5,000 images though, with the other 20,000 being only coarsely and shoddily annotated. It also contains significantly less classes: where Mapillary Vistas contains 152 object classes[‡], Cityscapes only maintains 30.

The only dataset with semantic segmentations related to materials is MINC [7], introduced in 2015. It is an extension of the much smaller OpenSurfaces [6]; OpenSurfaces contains 105,000 segmented images, while MINC sports three million training samples. OpenSurfaces has many of its 28 supported classes under-sampled, and hence MINC brings the total number of images to three million, making sure each material category has sufficient samples. Each sample can contain both full-fledged segments (polygons denoting materials, originating from the OpenSurfaces dataset), as well as clicks (single points denoting materials), which were less costly to obtain. The dataset supports 23 material classes, meaning Bell et al. dropped five classes from OpenSurfaces. They use this dataset to train a material segmentation network. While MINC's purpose is material segmentation, it focuses almost exclusively on indoor imagery and contains classes of materials which are seldom found outside. For this reason, also MINC is not suitable for our street-view material segmentation task. Table 1 further highlights why none of the currently existing datasets can be used in this work.

[*] "Scannerless" here refers to a methodology which records its environment or image instantaneously, as opposed to pixel-by-pixel; this is known as scanning LiDAR.

[†] At the time of writing, the dataset has not yet been published, with the publication's website stating it is "coming soon".

[‡] Their Research Edition offers only 66 of the 152 classes available in the Commercial Edition.

Name	Publication	Objective	Contents	Advantages	Disadvantages
NED (Natural Environment Dataset) [5]	2018 - 2019	Intrinsic image decomposition, semantic segmentation	●35K images ●40 different gardens ●5 lighting conditions ●16 classes	Contains RGB , albedo and shading images (apart from ground truth segmentations).	Depicts only gardens and parks.
COCO (Common Objects in Context) [58]	2016 - 2018	Panoptic segmentation, keypoint detection	●330K images (>200K labeled) ●1.5 million object instances ●80 object categories ●91 stuff categories ●250,000 people with keypoints		No material segmentation; stuff annotations are rarely actual materials. Small percentage are street-view images.
Mapillary Vistas [67]	2018	Panoptic segmentation	●25,000 HD images covering global reach ●152 object categories ●100 instance annotated categories	Mainly street-view images.	No material segmentation; stuff annotations are rarely actual materials. Research Edition only contains annotations for 66 of 152 classes.
Places2 [101]	2017	Image classification	●>10 million images ●>400 classes ●5,000 - 30,000 images per class	Improved version of Places.	Only image classification. No material annotations.
ADE20K [103, 104]	2016 - 2017	Semantic segmentation, instance segmentation	●~20,000 images	Categorized by scene category.	Only about 20% of dataset is street-level imagery.
Cityscapes [21]	2016	Semantic segmentation	●5,000 HQ annotations ●20,000 coarse annotations ●Semantic, instance-wise, and dense pixel annotations ●30 classes	Mainly street-view images. More popular than Mapillary.	No material segmentation; stuff annotations are rarely actual materials.
MINC (Materials in Context Database) [7]	2015	Material segmentation	●3M labeled point samples ●7061 labeled material segmentations ●23 material categories	Annotates materials. Contains realistic and "difficult" images.	Mostly restricted to indoor scenes with man-made, indoor materials.
Places [102]	2014	Image classification	●2.5 million images ●205 scene categories		Only image classification. No material annotations.
DTD (Describable Textures Dataset) [19]	2014	Texture classification	●5,640 images ●47 terms (categories) ● 120 images per category		Only perceptible texture property annotations.
PASCAL-Context	2010	Panoptic segmentation, semantic segmentation	●21,738 images (from PASCAL VOC 2010) ●540 classes	Semantic segmentation extension to PASCAL VOC 2010, which only contains instance segmentations.	No material segmentation; stuff annotations are rarely actual materials
Stanford Background Dataset [35]	2009	Semantic segmentation	●715 images ●7 classes		No material segmentation; stuff annotations are rarely actual materials. Comparably small set.
CIFAR-10 [50]	2009	Image classification	●60K 32x32 images ●10 classes		No material segmentation; stuff annotations are rarely actual materials. Only 10 classes.

Table 1: A non-exhaustive comparison of annotated computer vision image datasets. The last two columns give an impression of a dataset's suitability for outdoor material segmentation. As apparent from this comparison, none of the datasets are inherently fit for the purpose of material segmentation.

3 Methodology

The methodology of this work is discussed in this section. An overview of the process is given in Section 3.2, in which the structure of this research is discussed. Before that, however, it is necessary to discuss our method, which is essentially Semantic FPN [48], in Section 3.1.

3.1 Method

This section describes our methods and settings used in training Semantic FPN [48] for each of our experiments. We reimplemented the recent Semantic FPN in TensorFlow, faithful to the specifications given in [48]. We selected this network as our material segmentation network of choice, due to its reportedly good performance despite its architectural simplicity, and its potential for further improvement. On the Cityscapes dataset [21], Semantic FPN performs 0.5% worse than DeepLabV3+ with a ResNeXt-101 backbone and 1.9% worse with a ResNet-101 backbone, while being significantly simpler to implement. It fuses a number of architectural concepts together. It start with ResNet-FPN: the input image is convolved to a small representation using ResNet-50 [39], a fairly typical CNN of which the outputs of residual blocks can be regarded as constituting a bottom-up pyramid. In the reverse direction and starting from the deepest feature layer, a top-down featurized image pyramid is built from this bottom-up pyramid by repeatedly bilinearly upscaling a previous feature layer and element-wise summing this with the corresponding feature layer in the bottom-up pyramid (i.e. ResNet).

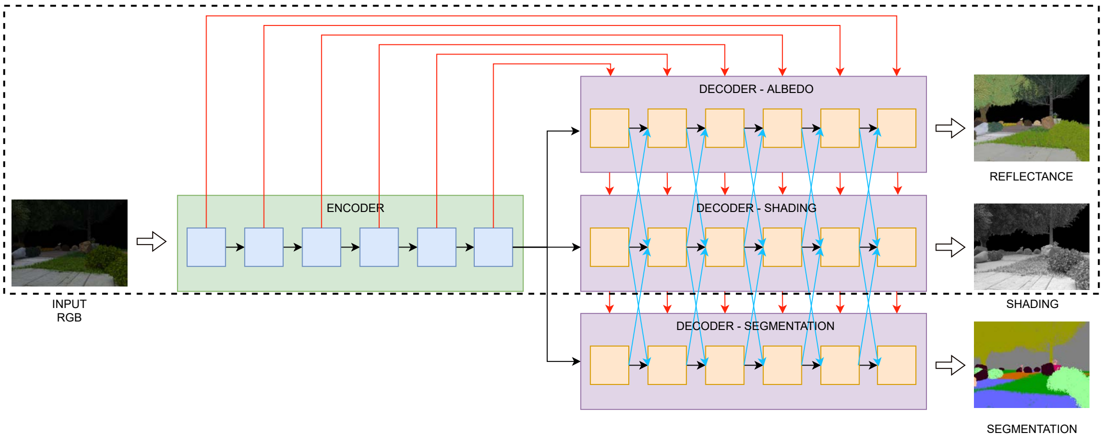


Figure 37: The ShapeNet FCN, predicting a semantic segmentation as well as a reflectance map and a shading map. The three decoders for the three output maps share the same encoder. In order to further strengthen the correlation between features in the three components, they are piece-wise connected using inter-connections. Additionally, features layers in the encoder are connected with the separate decoders with what Baslamisli et al. call mirror-links, in order to mix high-resolution, low-level semantics with the low-resolution, high-level semantics in the style of FPN, DenseNet and FishNet. Reproduced with permission from [5].

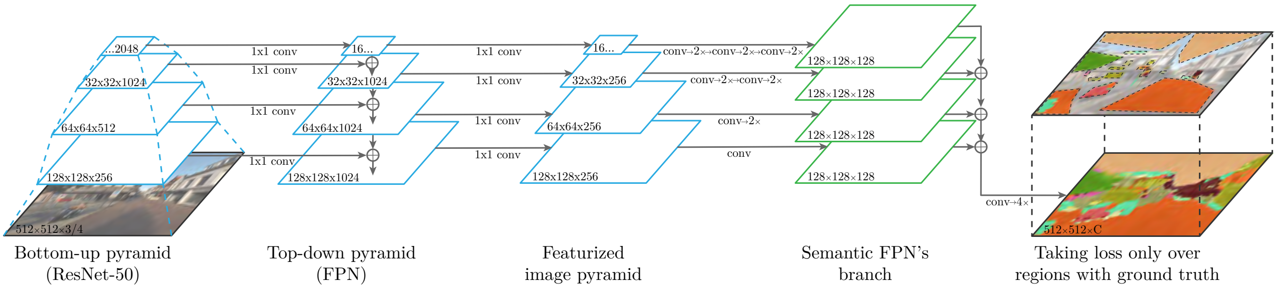


Figure 38: Our method, comprised mostly of Semantic FPN [48]. The 16x16 feature representation of ResNet-50 is taken and continually bilinearly doubled in size and element-wise summed with the corresponding layer in ResNet-50's pipeline. This creates four feature representations of different scales, which are convolved with 1x1 kernels in order to reduce depth. They are then bilinearly upscaled to make them equisized, after which they are element-wise summed, 1x1 convolved, and bilinearly quadrupled to yield the final predicted segmentation image. Only selected regions within our dataset images are annotated with ground truth, which only allows for taking the loss and evaluating performance within these ground truth regions.

This results in multiple feature representations of the input image at different scales. In our case, we obtain four scales: 16x16, 32x32, 64x64, and 128x128. To each of these, a 1x1 convolution is applied in order to reduce the depth of each feature scale from 1024 to 256. The network up until this point is known as FPN [56], with a diagram in Figure 23d. The second part combines the differently sized feature maps into a semantic probability map. The smallest three of the feature representations are upscaled to be of the same 128x128 resolution as the largest. Each doubling in size is effectuated by a 3x3 convolution, followed by batch normalization, followed by ReLU, followed by a 2x bilinear upscaling. This sequence of operations is, for example, executed three times for the smallest scale, and only once for the 64x64 scale. The now equisized feature layers are element-wise summed, and a final 1x1 convolution and 4x bilinear upsampling gives the segmented probability layer. Semantic FPN's way of transforming the FPN feature scales to the probabily image is summarized in 36. Semantic FPN is also discussed in Section 2.5. There it is explained how Semantic FPN is a subnetwork of Panoptic FPN, which, besides semantic segmentation, adds another instance segmentation branch which works on top of the four feature map scales. Our whole method is schematically summarized in Figure 38.
One essential deviation from the original specification of FPN [56] is the switch from nearest neighbour upsampling to bilinear upsampling. FPN was originally conceived for the task of object detection, which does not require feature maps of a very high resolution. For pixel-wise predictions, however, high resolution feature maps are essential.

As with any FCN, this network can take input images of any resolution. We chose 512x512 as our input dimensions, which seemed to strike a good balance between output resolution and memory consumption, and the images in the dataset are scaled accordingly prior to their processing. The depth maps are stored on disk as 1536x1536 PNG files. Before network processing, these are also downscaled to 512x512 using nearest neighbour, after which they are decoded into a depth map and concatenated with the RGB channels. Nearest neighbour is the most logical choice for the job, since any interpolation may cause non-existent, intermediate surfaces to appear between object boundaries. We initialize the weights of the ResNet-50 backbone with weights pretrained on ImageNet. For a network taking four-channeled input, such weights are not publicly available as far as we are aware. We simply initialized the additionally required weights – for the depth of the filters in the first convolutional layer – with reflected values (stemming from the same filters). Practically, this means that the weights for the depth channel take on the same initial values as those of the green colour channel.

3.2 Approach

This section discusses how we are to answer our RESEARCH QUESTION: "what material segmentation performance are we able to obtain on outdoor imagery?" We quantify material segmentation performance as mIoU. Our network that we use, Semantic FPN, is a subnetwork of Panoptic FPN, and by adding one branch can be turned into a panoptic segmentation network. This allows for simultaneous street-level imagery segmentation and street furniture detection, which, on the Cityscapes dataset, results in an increase of 1 of both mIoU (for semantic segmentation) and AP (for instance segmentation).

As described in the previous section, we annotated only certain segments (approximately 30%) of 800 street-level images. In training the network, these images are processed and predicted upon – as a whole – by a Semantic FPN network. However, since ground truth data is only available within the annotated regions, the loss is calculated only for the pixels contained in the annotated segments. Besides there being substantially less training data than when the images were more densely annotated, another important drawback is that the network will not learn about material boundaries. The edges of `foliage` look remarkably different than those of `cloud`, `soil`, `water`, or `steel`. With only patches of data to work with, the network only gets glimpses of the material texture at the center of segmentations. The network never learns the patterns of material edges, and we can therefore hardly expect it to accurately segment materials contours.

SUBQUESTION I

"What measures significantly reduce either class imbalance or the adverse effect of class imbalance on material segmentation performance?"

We distinguish and investigate three aspects which influence material segmentation performance, with the help of three subquestions. SUBQUESTION I deals with the severe class imbalance in our dataset. A dataset has class imbalance when the difference in size of the smallest and largest classes is disproportionately large, i.e. the dataset's ground truth follows a long tail distribution over its classes. We define class size as the number of ground truth pixels in our dataset pertaining to a class. The smaller classes are expected to perform worse than the large classes, which would drag down our performance. With the aim of reducing class imbalance and answering SUBQUESTION I, we try a number of methods: class-weighted loss, where we multiply the loss of each class with a weight that is inversely proportional to its relative size and thereby increases the importance of loss taken over pixels from small classes [52]; Dice loss, a different loss function which optimizes mIoU more directly and is therefore supposed to give equal importance to the small classes [83]; focal loss, which places more importance on classes which are more difficult to predict (with this difficulty expressed as network uncertainty) [57], and a number of combinations of class-weighted loss and one of the other loss functions. These measures fall under the category of cost sensitive learning, which addresses class imbalance on a classifier level. Another category of class imbalance countering measures instead operate on the dataset itself and aim to level the actual class sizes directly. The method of oversampling belongs to this category. Buda et al. found this to be the most optimal countermeasure when dealing with class imbalance for image classification [12], though they have not considered any cost sensitivity learning methods and it is uncertain whether their results carry over to the more complex task of semantic segmentation.[*]

In case none of these measures manage to significantly reduce the adverse effects of class imbalance, we resort to merging the smallest classes together. Of course, some important information is lost in this process. As an example, `iron` is amongst the smallest classes, and merging it with other classes would mean discarding useful information for, say, manhole detection. However, reasonable performance for all classes allows us to draw conclusions with respect to all those classes, instead only to the largest and most well-performing ones. Practically speaking, a sufficient overall performance due to all classes showcasing high mIoU would allow material segmentation to be used in production without extensive

[*] "Complexity" here refers to the number of predictions done on an image; image classification predicts one label per image, while semantic segmentation predicts one label per pixel.

manual correcting labour (which would otherwise be needed to correct the worst performing classes).
With this reduced set of classes, we train using cross entropy loss and focal loss, both with and without class weights. For each of these, we try three ways of dealing with the segmentation of the omitted classes: merging them into one `background` class, only using the `other` class as `background` (in which case it is simply renamed), and removing all those segmentations altogether without using them any further. The best method out of the 12 is chosen for use in the subsequent experiments.

SUBQUESTION II

"How much does the performance of material segmentation with additional depth input improve compared to our network only receiving colour information?"

Next, we compare two types of input data to the network: one only has colour information encoded in the RGB format, while the other adds a depth channel to these, which gives RGBD input. We determine whether one performs better than the other with SUBQUESTION II: "How much does the performance of material segmentation with additional depth input improve compared to our network only receiving colour information?" If RGBD appears to perform significantly better than RGB input, then this corresponds to our expectations. If RGBD input performs similar to or worse than RGB, we start by determining whether our depth maps hold any discriminitive value for material segmentation in the first place by training only on the depth. Then we train and test only with a selection of classes for which depth is clearly distinct. The classes we select in this latter experiment are preferably classes which have not performed too well on our initial experiment(s). This is because the performance of those classes might already be saturated with RGB input, in which case it becomes harder for a depth map to further increase it. By isolating these "depth-discriminatory" classes, we ensure that classes of which the depth could be noisy and confuse the network are ignored. This experiment allows the network to focus purely on classes of which the depth map contains helpful information, and therefore should theoretically benefit performance. If these classes show the same performance in this isolated experiment, then this suggests that depth provides no additional help to the network if it already works with RGB input, i.e. colours make depth superfluous and unnecessary.

In either case, we investigate the effect of pretraining on the difference between the respective input formats. Even though pretrained networks only have to be "finetuned" instead of trained from scratch, they start off with a bias for the task and/or dataset on which they have been pretrained. Our network was pretrained on ImageNet, which has little resemblance to our dataset, and which does not work with depth as input. It is possible that pretraining makes the network start on a suboptimal spot on the loss landscape, where it has started to converge based on colour information and where depth has little influence over the gradient direction anymore. We therefore train Semantic FPN from scratch, using Xavier initialization to initialize the weights, also known as Glorot initialization [34]. For each filter, Xavier initialization draws samples from a uniform distribution ranging between $-\sqrt{6/(w_{\text{in}} + w_{\text{out}})}$ and $\sqrt{6/(w_{\text{in}} + w_{\text{out}})}$, where w_{in} is the number of input units to the weight tensor (the filter) and w_{out} is the number of output units in the weight tensor (which equals 1, since each filter outputs one activation at each location of the input).* Xavier initialization ensures that the weights have variance equal to the input, which ideally is unity variance. If this experiment yields a measurable improvement of RGBD performance, then this suggests that the pretraining on ImageNet obstructs the depth map from being sufficiently exploited.
When loading our depth maps from disk in order to use them as a channel, we downscale them using nearest neighbour, in order to avoid any interpolation artifacts. We are curious whether the method of downscaling has any effect at all, and therefore attempt to train on depth that is bilinearly interpolated instead of downscaled with nearest neighbour.

SUBQUESTION III

"How much does material segmentation performance increase with more training data?"

Lastly, 800 sparsely annotated images can be considered a small amount of data for this relatively complex deep learning objective. SUBQUESTION III asks to what extent the small amount of data bottlenecks material segmentation performance: "How much does material segmentation performance increase with more training data on a similar dataset?" Accurately answering this question would require us to actually annotate a larger dataset. Instead, we perform a number of experiments in which we control the training data that we have, in order to estimate whether an increase in training data offers an increase in performance.

The first of these experiments concerns the relation between segmentation size and class IoU. We train with only with a selection of large classes: `asphalt`, `brick`, `foliage`, and `sky`, of which we will repeatedly reduce `asphalt` in number of ground truth pixels by "shrinking" each of its segmentations. All other classes are completely ignored, and no

*We assume here that each filter is initialized independently, instead of all filters of the same convolutional layer together. However, we are not certain if this is the common method, or even the method used by our implementation.

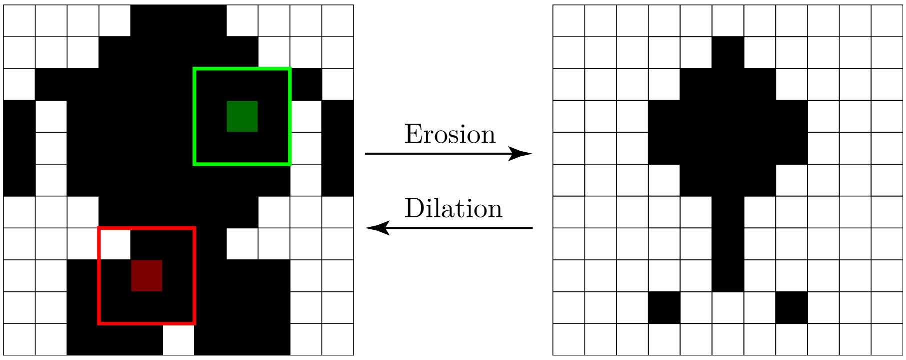


Figure 39: The process of eroding a binary, pixel-wise segmentation mask. In this examply, a kernel of size 3x3 is used. At each spatial location where the kernel contains at least one "inactive" pixel, i.e. a pixel that does not belong to the mask, then the pixel centered around the kernel will also by deactivated. Only pixels around which the kernel contains only active pixels will be kept. The inverse to erosion is called dilation, in which pixels are activated if its surrounding kernel contains *at least* one active pixel.

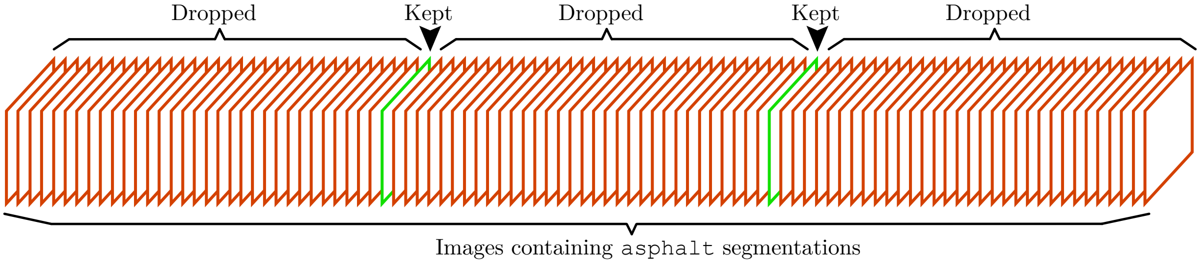
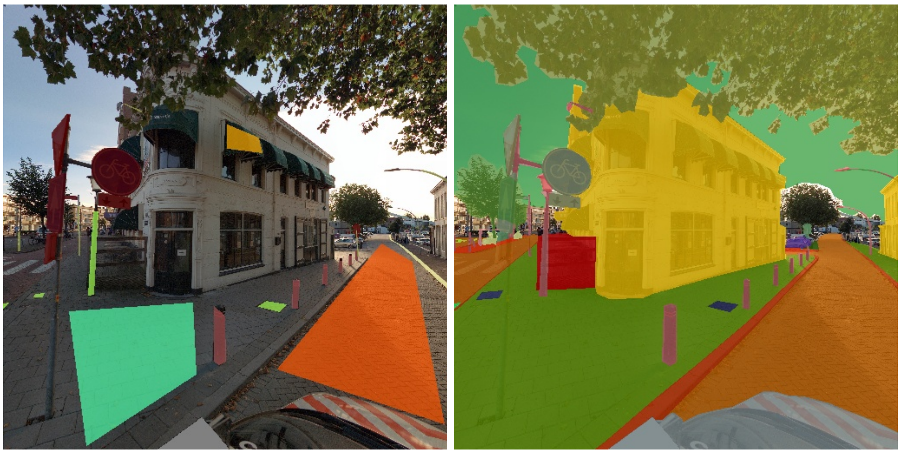


Figure 40: The selection of images with `asphalt` from which `asphalt` is removed, or dropped. The larger the dropout rate, the less segmentation will be trained with. The dropout rate is 32 in this example.

`background` class is used. The shrinking of `asphalt`'s segmentations is done using erosion, a common operation in computer vision which works on binary segmentations. It reduces the area of the target segmentation by skiving off pixels around its edges. The magnitude of this erasing of edge pixels its determined by the size of the kernel. The process is visualized in Figure 39. If all pixels of a segmentation are under the kernel at a certain location, then the segmentation's pixel in the center of the kernel at that location is kept, otherwise it is eroded (removed). In other words, only pixels for which a square kernel, centered on top of it, "fits" in the segment are retained. In this way, the larger the kernel size, the more ground truth is removed. We train with multiple kernel sizes with which `asphalt` is eroded: 0 (no erosion), 50x50, 100x100, 150x150, and 200x200. If its performance decreases accordingly, we can confirm that the typical size of a class's segmentations is proportional to its performance.

We also use another method of reducing the class size of `asphalt`, namely by discarding, or dropping, `asphalt` segmentations. Each run has a set value, which we will refer to as the dropout rate for ease of notation, and which determines the number of images containing `asphalt` from which `asphalt`'s ground truth is removed, after which `asphalt` is kept in one image. In other words, the dropout rate fixes a pattern upon the images with available `asphalt` ground truth; this patterns designates, say, 32 of such images to have their `asphalt` removed, after which one image gets to keep its `asphalt`, after which, again, 32 `asphalt`-containing images are stripped, etc. This pattern is visualized in Figure 40. We chose this method to ensure determinism. It also allowed us to make sure that `asphalt` ground truth decreases more or less proportionally with the dropout rate; setting a dropout probability does not necessarily ensure this. We also opted to additionally erode all `asphalt` segmentation with a 128x128 kernel, since the small classes have smaller segmentations than the large classes, and we felt this to result in a more fair comparison.

Lastly, we employ the two largest datasets, the instance-as-semantic and instance-as-material datasets, that were derived from the original instance segmentation dataset. The former will keep the original 96 instance classes, while the other has each instance class translated to a material class (the translations can be found in Appendix A). Both these new datasets have problems – which are summarized in Table 3 – and are not expected to perform as well. However, they can be useful in assessing the plot of mIoU (or any other measure of performance) against training set size, of the material segmentation problem. In this work, we will refer to such a curve as a "performance curve". We can only plot such a



(a) Material dataset

(b) Instance-as-semantic dataset

Figure 41: An example image with annotations from the three respective datasets. The material dataset is much sparser than the instance-as-semantic and instance-as-material datasets, besides it containing annotations for many fewer images.

Code	*Resolution*	*Range (in m)*
00	1 mm	0 - 16.384
01	4 mm	0 - 65.536
10	16 mm	0 - 262.144
11	64 mm	0 - 1048.576

Table 2: The four depth resolutions in which pixel values in a depth map can be expressed. The resolution defines the difference in depth between two consecutive values, and indicates how precise and fine-grained the depth measurements are. Every quadruple in maximum range forces the resolution to be four times as low.

curve for experiments on our material segmentation dataset for up to 600 images, since 200 images out of the total 800 are reserved for the test and validation sets. Performance curves for the other transformed dataset can be plotted up to 3800 images, and should give an indication of the performance potential for material segmentation.

4 Experiments and results

The experiments ran in order to answer the posed research questions, and the results obtained by performing these experiments are reported upon in this section. We start by giving an overview of the setup of our experimentation. Section 4.1 treats the datasets with which we train and test, one of which we annotated while the other two are transformations of an existing dataset. This section also names the 21 material classes that we adhere to, and motivates this selection. After that, we discuss our metrics in Section 4.2. Section 4.3 gives specifics of our training procedure and implementation. Section 4.4 then discusses the results of the experiments.

4.1 Datasets

This section concerns the data that we work with. The procurement of our training and evaluation data is illustrated in Section 4.1.1, which describes the data we had to our disposal from the outset, and Section 4.1.2, where we discuss the datasets we ourselves derived from the data we started with. Lastly, Section 4.1.3 mentions and motivates the classes we maintain in our material dataset.

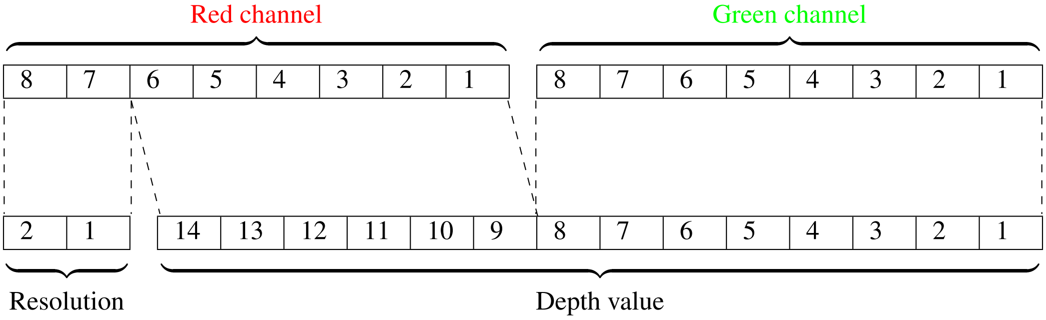


Figure 42: CycloMedia way of storing depth in the red and green channels of a PNG file. The first two bits of the red channel indicate the resolution at hand. There are four possible resolutions, given in Table 2. The remaining 14 bits represent the actual depth value at the indicated resolution. Naturally, depth is encoded at the highest possible resolution for which the value is still in range. This means that the resolution quarters right after the indicated range boundaries.

4.1.1 Pre-existing dataset

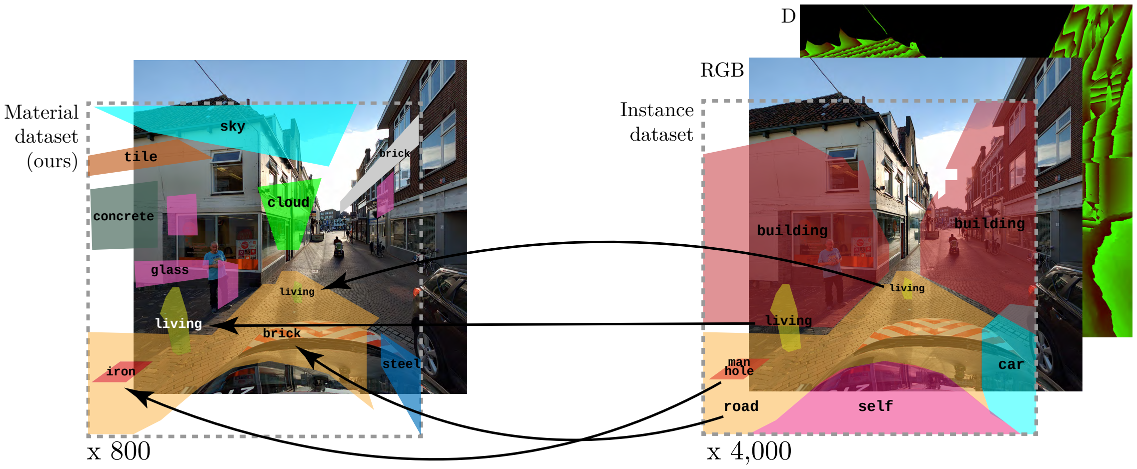
We have 4,000 street-view images, property of CycloMedia. They depict a variety of environments: residential areas, shopping districts, highways, industrial areas, rural landscapes, etc. The first half of the images were taken in Schiedam, The Netherlands, and the other half in the United States. For each of these RGB colour images there is a depth map available, which has been generated from 3D meshes constructed from LiDAR point clouds (we refer the reader to Figure 1. These depth maps are encoded and stored as PNG [93] files using the red and green colour channel, leaving the blue and alpha channels for future additional information.* The depth format they employ is shown in Figure 42. The first two bits of the red channel indicate one of four possible depth resolutions (not to be confused with image resolution) in which the rest of the bits – the remaining six bits in the red channel concatenated with the bits in the green channel – represent their depth value. The depth maps cover a total distance of 1,048.576 meters. Each of the four resolutions work with a different range, all starting from 0 up to at most the maximum depth. With two bits occupied to indicate the resolution, there are 14 bits left for the depth value itself, with which 2^{14} different values can be encoded. Each resolution uses 14 bits of precision to cover values across its range, meaning that depth values falling within a smaller range will be represented more accurately than those exceeding that range. The resolution of depth is therefore dependent on its magnitude. The four resolutions and their ranges are given in 2.
Before CNN processing, these two channels are converted back to a single channel of float values, which is how depth would logically be represented. We could supply the two channels directly to the network as RGBRG input, but this would require the network to additionally learn the transformation from two channels to one floating point channel, which we have now preprocessed for it. Some examples of the depth map can be found in Figure 4, showing that depth maps are noisy, though the severity and texture of this noise can be considered distinguishing characteristics for some classes. On the other hand, Figure 49 contains examples of materials where depth seems useless.

CycloMedia also provides their instance segmentation dataset, containing segmentations on all 4,000 images adhering to 96 instance classes (the table in Appendix A gives the class names). As is apparent from Figure 41b, this dataset contains much denser ground truth; nearly all pixels in an image have a label. Only distant buildings and vegetation, water, some small, enclosed regions of sky, and some incidental objects are missing. Even though the classes concern objects rather than materials, this dataset may prove useful in determining the increase in segmentation performance that a larger dataset can provide.

4.1.2 Constructed datasets

We created our own three separate ground truth datasets on top of (a subset of) the same 4,000 images. The first dataset, which we will refer to as the *material dataset*, contains material segmentations for 21 material classes as defined in Section 4.1. In order to keep the annotation effort at a minimum, only selected regions within an image are annotated, and most of the pixels remain unlabeled, as exemplified in Figure 41a. Some of these annotations have a rectangular or parallelogram shape – which are faster to draw though rarely cover the entirety of a material's surface – and others are material (instance) encompassing polygons. The latter were taken from CycloMedia's instance segmentation dataset,

*According to CycloMedia, as stated in their in-house documentation.



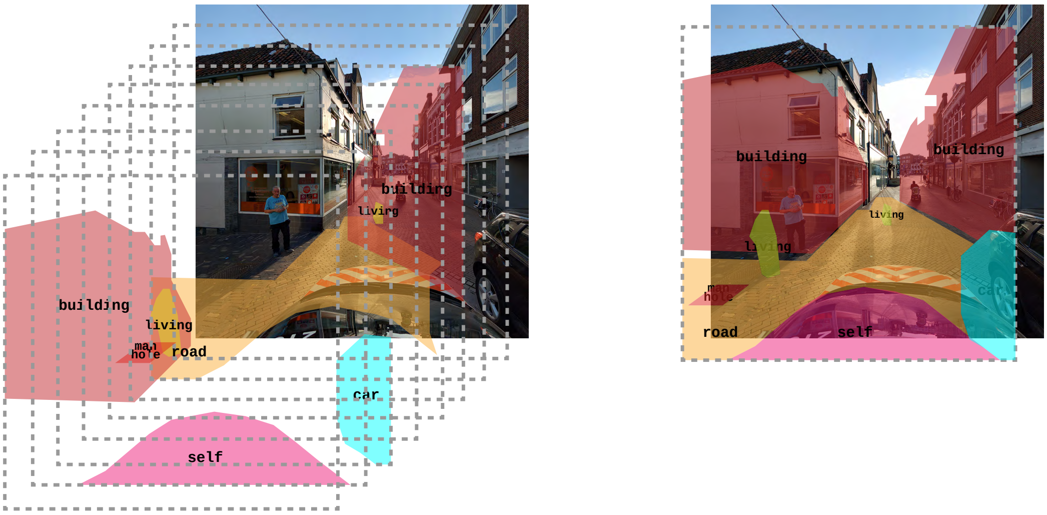
(a) Our material dataset. We annotated parallelogram material annotations. Whenever possible, we reused segmentations from the instance dataset (exemplified in (b)), and translated its label if necessary.

(b) All data which was supplied to us by CycloMedia and with which we started. Some instance annotations could be reused in our material dataset.

Figure 43: In creating our dataset for material segmentation, we copied the segmentations of selected classes from the instance dataset, and repurposed those as material segmentations. Those classes which often correspond to one material were taken. Manual verification ensured that these segmentations are correct. Sometimes, when an instance class can exist in multiple material forms, the labels had to be corrected. Sometimes a segmentation had to be removed, but since deleting a segmentation is more efficient than creating one, reusing instance segmentations still proved helpful in our annotation efforts.

as shown in Figure 43. Some of these classes in the instance segmentation dataset represent objects that in the real world often consist of the same material. For example, objects of class `object | support | pole` are likely to be made of steel, and `object | fire hydrants` are nearly always made of iron. The segmentations of these materially uniform classes were taken from this instance segmentation dataset and their labels translated to their associated materials, after each of these segmentation was verified for correctness. That is, if the translated material did not correspond with the material it encapsulated, this was corrected. If a translated segmentation contained multiple materials, and reshaping the polygon to encapsulate only on of the materials was deemed to much effort, we discarded it. We copied `construction | flat | road` annotations from the instance segmentation dataset because of its generally large image occupancy and translated it to `asphalt`. However, it is this class which often needs correcting, since our images contain brick or even tiled roads as well. The material dataset annotates on average approximately 30% of an image, and only contains annotations for the first 800 images in the dataset, i.e. the dataset only annotates images captured in The Netherlands.

The instance segmentation dataset has additionally been directly transformed into our second semantic dataset: the *instance-as-semantic* dataset. Transformation of the instance segmentation dataset into a dataset for semantic segmentation was a somewhat involved process, and a difficult one to optimize. An instance segmentation dataset generally has segmentation masks for multiple objects within an image, and these masks may overlap. For example, the segmentation mask of a manhole or a pothole may intersect the segmentation mask of a road. In order to "flatten" the instance segmentations to a semantic segmentation map, for each such overlapping segment it has to be determined which of the conflicting masks is supposed to be "on top". We do this by representing the colour and depth distributions of the pixels within the overlap as a histogram. Both conflicting masks are also each represented by such a histogram, and the mask with histogram distance closest to the histogram of their overlap gets its label selected and applied to the overlapping pixels. This ordering is only defined between every pair of masks, and thus constitutes a topological ordering. A topological order is not a total order, because it does not adhere to the properties of antisymmetry (if $a \leq b$ and $b \leq a$ then $a = b$) and transitivity (if $a \leq b$ and $b \leq c$ then $a \leq c$), and therefore cannot be solved with regular sorting algorithms. For ease of implementation, we opted for a brute-force approach, which simply keeps swapping masks in the ordering as long as a pair of out-of-order masks is detected. Though inefficient, this approach works relatively well according to a visual inspection, and manages to successfully resolve most overlap. Edge cases are sparsely present though; Figure 41b shows the pole of a lamppost being buried under masks which should be under it. Such inaccuracies happen infrequently enough for us to expect the network to be invariant to them. This dataset has annotations for nearly all pixels in an image, and does so for all 4,000 images.



(a) Each instance segmentation is a binary map, indicating which pixels belong to the instance. These instances may overlap.

(b) In order to "flatten", or merge, the separate instance maps together, the overlap for each pixel needs to be resolved.

Figure 44: An intuitive visualization of instance segmentations, and the semantic map obtained by flattening them. This flattening requires resolving overlap, which we did by comparing colour histograms of overlapping segmentations with the histogram of their overlap.

Material dataset	Instance-as-material dataset	Instance-as-semantic dataset
• Correct labels	• All 4,000 images annotated • Each image densely annotated	• Correct labels • All 4,000 images annotated • Each image densely annotated
• Only 800 of 4,000 images annotated • Roughly 30% of each image annotated • Severely imbalanced	• Erroneous labels	• Object labels rather than materials • Many (96) classes

Table 3: Positive and negative aspects of each of the three datasets. The former are indicated with green box colours, and the latter with red box colours.

The third dataset we call the *instance-as-material* dataset. We transformed the instance segmentation dataset in the same manner as described in the previous paragraph, and we additionally translate the instance labels to material labels, using a mapping from the 96 instance classes to the 21 material classes. This mapping can be found in Appendix A. The instance-as-material dataset therefore covers the same pixels as the instance-as-semantic dataset for the same number of images, but with different classes. Naturally, many of the instance classes from the instance segmentation dataset (can) consist of more than one material type, making the translation often erroneous. For instance, `construction | flat | road` could be translated to `asphalt`, `brick`, and even `tile`. In our experience of annotating roads, we found them most often to be made of asphalt, and we therefore decided that all roads are annotated as such, but this entails that all brick and tile will be wrongly annotated.

As may be clear, none of these datasets have full coverage, i.e. provide a label for every pixel in the image. The material dataset covers few images, has little coverage (per image), and is severely imbalanced. This latter downside makes training difficult, as CNNs are prone to be biased towards the larger classes, which results in poor performance for the small classes. The instance-as-material dataset annotates all 4,000 images and has a much denser coverage for each, but its mask labels are often erroneous, labeling the wrong material(s). The instance-as-semantic dataset has both a dense coverage as well as correct labels – excluding the discussed edge cases – but these labels address objects rather than materials. We recapitulate these points in Table 3.

4.1.3 Material classes

We define and maintain the following material classes: `aluminium`, `asphalt`, `brick`, `cloud`, `concrete`, `fabric`, `foliage`, `glass`, `grass`, `gravel`, `iron`, `living`, `other`, `plastic`, `sky`, `steel`, `stone`, `soil`, `tile`,

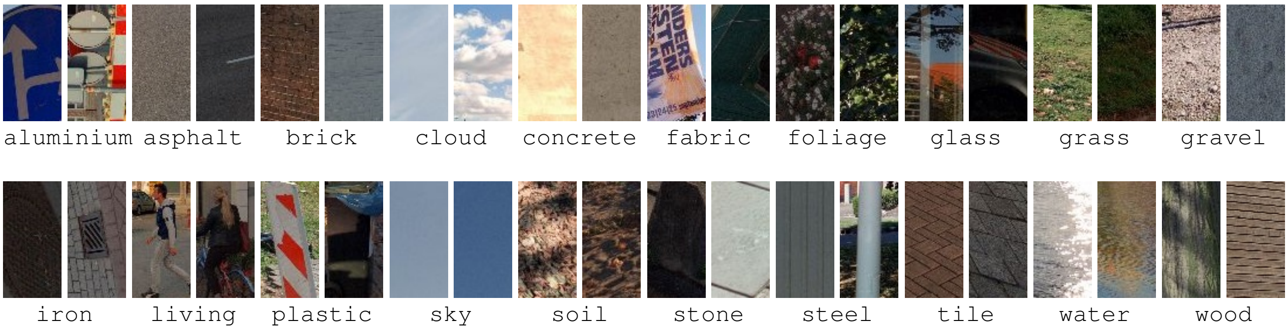


Figure 45: Samples of material segmentations. These highlight the difference in appearance, due to, for instance, differing lighting and weather conditions, but especially because the same material can be fashioned and processed into a variety of objects or constructions.

water, wood. We note that not all classes are actually materials, with cloud, living and brick being notable examples of classes that are not materials, but are descriptive and commonly occurring. The distinction between sky and cloud is not directly relevant to CycloMedia's objectives, but we still thought it interesting to investigate the extent the network would manage to distinguish the two, especially given the fact that their depth values are always zero, and their classification therefore relies purely on the colour channels. Moreover, though not currently relevant, the separation of aerial pixels into sky and cloud could be used for potential future purposes of determining whether pictures are taken on a sunny or on an overcast day, and even in determining the cloud cover.

In case multiple classes encompass the same material, we make sure the onthological boundaries between these is unambiguous. In case the encompassing material itself is also a class, this class serves as a catch-all for any instantiations of that material that do not belong to the more specific classes. For instance, brick and tiles are often stone, with the latter class designated for all stone that is not formed in bricks or tiles. Bricks and tiles both are made up of rectangular blocks of stone, but they differ in pattern: tiles are laid in such a way that all edges of a tile touch exactly one edge of a neighbouring tile, while a brick's edge may touch multiple other bricks, and are usually interleaved. Similarly, soil includes surfaces covered by leaves and fabric includes canvas and sailcloth. other serves as a catch-all for everything that does not naturally fall under the other categories; only pixels of which we were completely uncertain as to what they depicted are classified as other. Samples of the material classes can be found in Figure 45.

Some experiments involve the omission of certain classes, meaning materials are segmented with respect to only a subset of our original set of 21 material classes. Whenever classes are omitted, a new background class is technically supposed to replace those, or else the network is not able to correctly predict pixels belonging to omitted classes. The arrangement of this shift in classes can be effectuated in multiple ways. The first one merges all the omitted classes into the background class, in which case all their segmentation are relabeled to background. This makes intuitive sense, as background would factually contain all materials that are not primarily searched for. The downsize to this method is the possibility of background obtaining too much ground truth, depending on which classes are absorbed by it, which could exacerbate class imbalance. The second method simply translates the other class to background, and ignores (the annotations of) all omitted classes. Whereas the background class runs the risk of getting too large with the first method, it will be too small when using this method, since other is one of the smallest classes, with only a handful of annotations. A last method would be to completely ignore all omitted classes and to also forego the background class. Like the second method, this would be a waste of annotations, however, and this would technically be an improper way of organizing classes. After all, this method declares that all pixels in an image belong to one of the remaining classes, which clearly does not hold. This is no problem for our evaluation of this method, since we would only evaluate regions of the image annotated as one of the remaining classes, while everything outside will be disregarded. This appeal of this method lied in the fact that it focuses purely on the task of predicting the "foreground" classes, and any confusion which may arise from other classes is avoided. This method cannot be used in production though, because all pixels labeled as an omitted class will then be misclassified. Since there is no clear best method, we try all three when we select the optimal loss function for our reduced set of classes in our answering of SUBQUESTION I.

Whenever we do use background, it will not be taken into account in averaged performance metrics, since background contains all classes that er explicitly not of interest. Besides, this would be unfair in comparisons to methods without background.

The distinction between other and background may need clarification: whereas other has specifically been an-

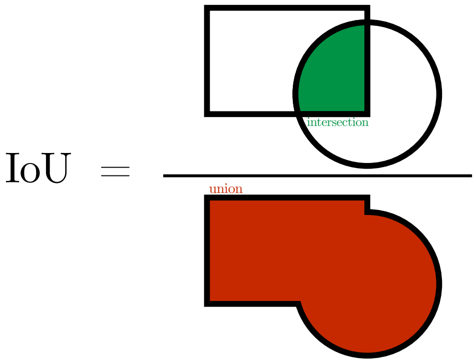


Figure 46: A depiction of both the union and the intersection of two shapes or regions. In the context of IoU calculation, one of these regions is a ground truth mask, while the other is a prediction. IoU is then their union, in red, divided by their intersection, in green. In this instance, one could reasonably state that the relative positioning of these two particular regions yield a rather low IoU.

		Predicted	
		Positive	Negative
True	Positive	True positive	False negative
	Negative	False positive	True negative

Table 5: The four types of classification that can be made. "Positive" and "negative" refer to the classification predicting that an object is present or is not present. "True" and "false" refer to whether or not the classification is correct.

notated for and, in that sense, can be regarded as a material in itself, the function of a `background` class is purely to allow the network to predict something else in case none of the other classes used in a certain experiment are suitable. This stems from the fact that every pixel in an image should theoretically belong to one or more of the original classes. When certain classes are disregarded in an experiment, this may no longer be the case. A `background` class is therefore needed to represent the omitted classes.

The visualizations of ground truth or predictions concerning the material and instance-as-material datasets make use of the colour coding of the material classes given in Table 4.

aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel
🟥	🟥	🟧	🟫	🟫	🟧	🟩	🟨	🟨	🟩

iron	living	other	plastic	sky	soil	stone	steel	tile	water	wood
🟩	🟥	⬛	🟥	🟧	🟩	🟨	🟨	🟩	🟩	🟪

Table 4: The colour coding of material classes as maintained in this work.

4.2 Metrics

The most prominent metric in our evaluation of material segmentations is the *mIoU* metric (alternatively called the Jaccard index), the de facto metric for semantic segmentations. COCO uses mIoU to effectuate a ranking among Stuff Segmentation Task contenders, which was originally introduced by PASCAL VOC [28]. It is calculated by dividing the number of pixels of the intersection between the predicted and ground-truth class segmentations by the union of these segmentations, averaged over classes [13]. Phrased differently, *intersection over union* (IoU) is the ratio of the intersection of a predicted mask with the corresponding ground truth mask to their union. The intersection and union of two regions are visualized in Figure 46. All IoU values for all class instances in all images are averaged to get the mIoU. This means that smaller segmentations are given as much weight in the final mIoU score as larger ones.
mIoU can alternatively be expressed in terms of individual pixels. Within machine learning, TP, FN and FP are common performance metrics for evaluating models, and are summarized in Table 5. *TP* (true positives) is the number of correct predictions, FP (false positives) – sometimes called Type I errors – are the incorrect predictions, and FN (false negatives)

– sometimes called Type II errors – are all ground truth instances the ML model did not manage to distinguish. A fourth class, TN (true negatives), containing correct predictions for something not being true or not being present, appears less frequently in metrics for computer vision. For instance, in object detection and instance segmentation, there are an infinite number of objects not present in every image, making it impossible to keep track of true negatives.
If we regard each pixel as a separate classification, a mask's IoU can be formulized as:

$$\text{IoU} = \frac{|TP|}{|TP| + |FP| + |FN|} \tag{17}$$

An image's mIoU is then calculated by averaging over the IoUs of the ground truth classes present in the image. Each class contributes equally in this averaging. A variation of mIoU is fIoU (frequency-weighted intersection over union), which weights a class's IoU by the relative frequency of its ground truth pixels. In a sense, fIoU is better at capturing how good a segmented image looks, since it rewards high IoUs of classes which take up the largest portion of the image. In contrast, mIoU disregards class size, and punishes small classes in the same way as large ones. However, mIoU will remain the leading metric, as we consider each class equally important regardless of the number of ground truth pixels. For instance, correctly predicting the pixels for a traffic sign is as desirable as correctly predicting the pixels of a road, even if the sign occupies a much smaller portion of the image. After all, one of the potential uses of material segmentation is to aid object detection or instance segmentation, in which case the IoU per object instance is what matters most.
If a class is both not predicted and not present in an image, then this class is ignored in mIoU and fIoU calculation for that image. This means that once even a single pixel of a class is present but not predicted, or predicted but not present, then this will significantly punish mIoU, as this class's IoU of zero is now mixed in, and with weight equal to the other classes. We feel that this is the biggest downside to the mIoU metric, and in this regard, fIoU might be more lenient.

Additional metrics are *(pixel) accuracy*, the fraction of correctly classified pixels, and *mean accuracy*, the fraction of correctly classified pixels per class averaged over classes [61]. For binary problems, accuracy is formally defined as:

$$\text{Accuracy} = \frac{|TP| + |TN|}{|TP| + |TN| + |FP| + |FN|} \tag{18}$$

Since we are not tackling a binary problem and true negatives are irrelevant to our overall semantic segmentation performance (it is dependent on the number of classes), this formula cannot be applied. We instead opt to define accuracy simply as the percentage of correctly classified pixels, which ultimately has the same outcome. For class-specific accuracy, the above formula can be used, since true negatives can be calculated per class. These class-specific accuracies can in turn be averaged to obtain mean average. However, in our case, class-specific accuracy is always extremely high, regardless of class size; if a class is large, then large regions are likely to be correctly predicted, resulting in a lot of true positive, whereas if a class is small, then it will be rarely predicted, which will yield a high number of true negatives. This means that mean accuracy and true negatives are not useful measures in our studies.

4.2.1 Precision and recall

The *precision* of a model is the percentage of correct predictions (over all predictions):

$$\text{Precision} = \frac{|TP|}{|TP| + |FP|} \tag{19}$$

This formula and the following ones are applicable to both binary and multinomial problems, and is therefore the formula as we use it. Its *recall* is the percentage of ground truth instances, i.e. the things or stuff that are actually present in an image, that the model managed to correctly identify, or detect:

$$\text{Recall} = \frac{|TP|}{|TP| + |FN|} \tag{20}$$

A high precision with low recall is seldom informative. In such a case, only a small number of predictions have been done, and even though these were largely correct, they only detected a small portion of ground truth, which is not very impressive. This also goes the other way around; a high recall paired with a low precision only tells us that the class at hand has been severely overpredicted. This naturally causes most ground truth to be found, but also yields many false positives. Therefore, precision and recall are often consolidated in the *F1* measure, or F-score:

$$\mathrm{F1} = 2\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{21}$$

Unlike averaging precision or recall over all classes with instance segmentation, averaging precision or recall for semantic segmentation can be done in multiple ways [89]. The first calculates precision and recall for each class, and then averages them over the classes. This is called *macro-averaging*. It weights each class equally, similarly to mIoU, and punishes the mistakes of small class mispredictions as severely as the more easily avoided mispredictions of large classes. We consider all classes equally important, regardless of screen presence, and therefore macro-averaged precision and recall therefore will be most significant to our research. Another method, *micro-averaging*, aggregates all true positives, false positives and false negatives over all pixels (possibly after thresholding), and calculates precision and recall based on these totals. However, each false positive for one class is a false negative for another, meaning the total number of false positives will always equal the number of false negatives, and thus precision will equal recall (and F1). In other words, micro-averaging weights each pixel prediction equally, while macro-averaging weights each class equally [65]. Lastly, we could macro-average the classes, but weight the classes with respect to their relative size, which is referred to as *weighted macro-averaging* and is comparable to fIoU.

When looking at the performance of individual classes, we value recall over precision. The reason is that it is easier and more efficient to remove (extraneous) mispredictions in a binary segmentation than it is to fill in any missed ground truth, given the right tools. Anyone with any raster graphics editing experience is likely to concur that erasing mispredicted segments is generally quicker than tracing regions which have not been predicted. Because of this, we will discuss precision and recall separately instead of using the F1 score. Because macro-averaging averages over the precision and recall of the classes, the same holds for the overall performance. F1 is reported in the complete accounts of our experiments in Appendix G, however. Our complete results in the appendices include precision-recall curves, though we do not use those in our analysis in this section. Some notes on precision-recall curves are in Appendix C.

The last analytical tool we discuss is the confusion matrix. A *confusion matrix* is a table denoting the number of predictions for each pair of predicted class and actual class. Each row contains the number of predictions done on pixels of the corresponding ground truth class per class that was predicted. Conversely, each column contains the numbers of times that class was predicted per actual class. Ideally, all counts are in the matrix diagonal, in which case all pixels would be predicted correctly. The number of counts in this diagonal as a percentage of the matrix sum equals accuracy. A confusion matrix is most useful in detecting whether a class is often confused for another. A relatively large number of predictions in a cell outside the matrix diagonal indicates such a situation.

4.3 Training

Our initial experiment uses a multinomial cross entropy loss function, calculated only over the pixels for which ground truth is available.
We tried 0.1, 0.01, 0.001, and 0.0001 for starting learning rates, with a value of 0.01 performing the best for cross entropy loss and Dice loss, and a value of 0.001 performing the best for focal loss. A learning rate of 0.1 converged infeasibly slow and gave worse performance, one of 0.001 converges slightly faster, but performed approximately 16% worse, and 0.0001 again performed and converged significantly worse. Next, we tested batch sizes and quickly found that the higher the batch size, the better the performance, as is nearly always the case in deep learning. Our available VRAM allows a batch size of 8 when using cross entropy loss, but this no longer fits when using focal loss, which forced us to use a batch size of 6 for all experiments in order to provide for a fair comparison. The reason for using a fixed batch size across experiments while adjusting the learning rate to the loss function between experiments, is the fact that batch size increases proportionally for all loss functions and is limited by external factors, while each loss functions has a specific optimal learning rate. We do not wish to unfairly favour some experiments because unimportant external factors happen to benefit their performance, and compare their results to those of experiments which are unfortunate enough to not get this advantage, which would not be very informative. Learning rate, however, is something we ourselves determine, and since we do want to get the most out of each experiment while keeping external factors fixed, it is a small effort to test for an experiment's optimal learning rate.
Lastly, we tried momentum values of 0.7, 0.8, and 0.9, and found 0.8 to perform slightly better than the other values. After three epochs without an improvement of at least 0.0001 in the validation loss, the learning rate is divided by four. This

allows the network to more precisely converge to an optimum. In order to prevent overfitting, training is stopped after six epochs without improvement in the validation loss. Our prioritisation did not permit the investigation of other loss decay and patience values.

Across all experiments, we fix the test set to each contain the same 100 images, which establishes the most informative and valuable comparison, even if the annotations on those images (between datasets) differ. We also keep the validation set the same, using a different set of 100 images. As we explained earlier, the validation set determines for how long the network is trained and when training has converged. Keeping the validation set fixed means that training for each experiment stops whenever there is no improvement in that same set, ensuring that each experiment is maximally exploited *in the same way*. We could have opted to keep the number of training epochs fixed rather than keeping the validation set fixed; the network would see each sample the same number of times for each experiment. In this case, we could better compare experiments on the same dataset with differing training set sizes, as training set size would be the only difference between them. However, when comparing experiments on different datasets, it is more informative to train to convergence in an equal manner, since we'll know what kind of segmentation performance is achievable for each experiment.
Our material dataset only concerns the first 800 images of the 4,000 images provided by CycloMedia, and therefore only contains images captured in The Netherlands (recall that the first half of our images was captured in The Netherlands, with the other half captured in the USA). Since we wish to have the same validation and test sets for experiments on the instance-as-material and instance-as-semantic datasets as well as the material dataset, the validation and test sets must contain images exclusively taken in The Netherlands. This might unfairly evaluate experiments in which Semantic FPN is trained on images from the United States. While naturally occurring materials have the same characteristics regardless of country, like `foliage`, `water` or `wood`, materials that constitute man-made constructs often differ in appearance across nations. For instance, `aluminium`, a class mostly made up by traffic signs, and `iron`, consisting of manholes, fire hydrants and catch basins, tend to look different across the Pacific. Networks trained on datasets of all 4,000 images will additionally need to adapt to US images, but are only evaluated on the Dutch images. Their learned representations will be more general, while networks trained exclusively on Dutch images are expected to be better at segmenting Dutch images. It is likely that this will negatively impact the results of experiments trained on the whole set of images. On the other hand, such training might improve the network's ability to generalize.

We attempted for as deterministic a training procedure to the extent to which this was feasible. The images were shuffled before being distributed across the training, validation, and test set, which remain fixed across experiments. No further shuffling is being done, and there are no other stochastic elements in our entire pipeline. However, training on a GPU is generally non-deterministic, due to asynchronous floating point operations.

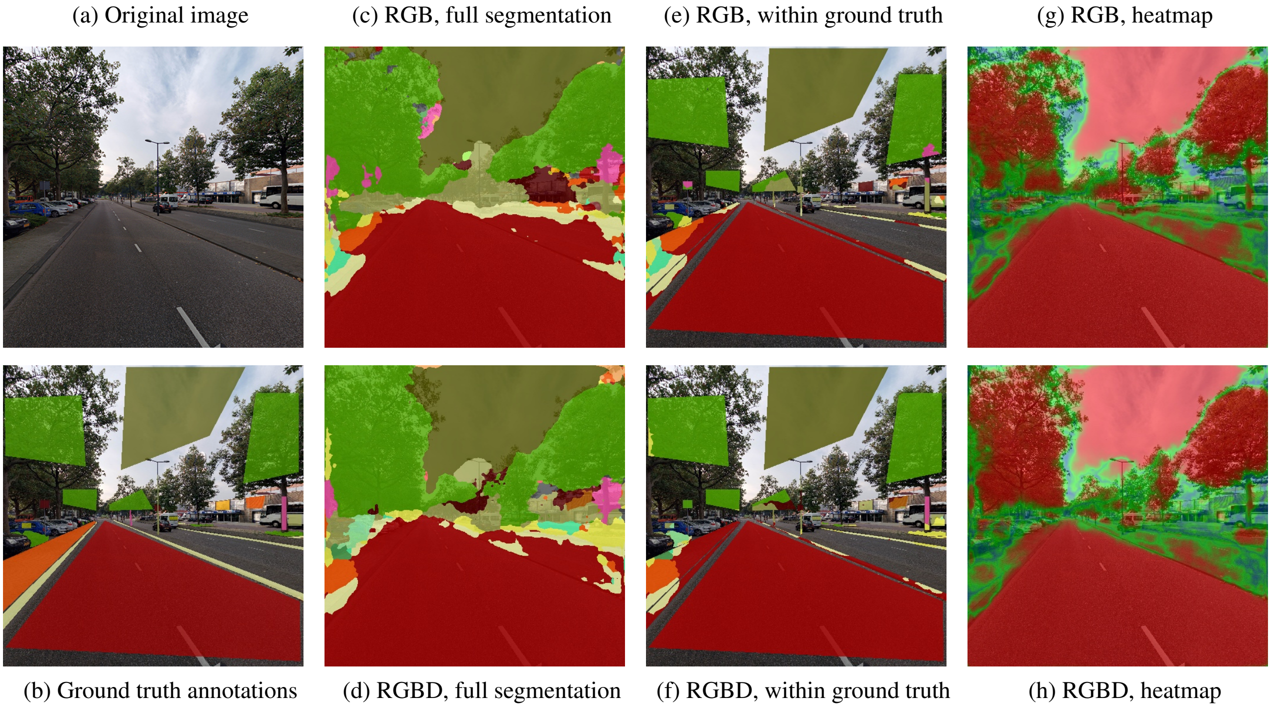
Our code has been exclusively written in Python 3.5. In implementing, training and testing Semantic FPN, we use the Keras API of TensorFlow 1.12, Google's machine learning framework [2, 1]. Though images are represented as TensorFlow's `Tensor` objects when being processed by the network, they are supplied and evaluated on as NumPy's multidimensional (`nd`)`array` objects [69], though the two formats are highly similar and conversion between them is easy. Images are read, written, and transformed with `opencv-python`, an unofficial wrapper package for the Python API of OpenCV 2 [8, 9], for which a Python wrapper package is readily available. For writing confusion matrices and AUC calculation, we use two corresponding functions from the Scikit-learn library [70]. Some models are trained using two Nvidia Tesla P100 PCIe GPUs, and some were trained on a single Nvidia Tesla K40m. Both GPUs sport 12 GB of VRAM. We use CUDA [76] 8.0.61 with cuDNN [18] 7.1.4. We include snippets of code for our loss function and network definition in Appendices D and E, respectively.

4.4 Experiments

We report on our experiments in more or less chronological order. We start with training and evaluation of Semantic FPN on our material dataset, which we consider our core experiment. All subsequent experiments are deviations from this experiment, and aim at a deeper understanding of this problem and investigate areas of improvement. Only figures of interest are included in this section; all figures for each experiment are printed in Appendix G.

4.4.1 Initial experiment

Figure 47 showcases Semantic FPN's predictions after having been trained on 600 images of our material dataset with cross entropy loss. Especially the large segmentations adhere somewhat to the shapes they try to encompass. We see difficulties in cluttered and distant areas, however. This was to be expected, since such far-away cityscapes are likely to



(a) Original image (c) RGB, full segmentation (e) RGB, within ground truth (g) RGB, heatmap

(b) Ground truth annotations (d) RGBD, full segmentation (f) RGBD, within ground truth (h) RGBD, heatmap

Figure 47: Visualizations of Semantic FPN's predictions after being trained on 600 images of our material dataset. (a) shows the original test image, (b) shows the ground truth annotations for the image, (c) and (d) show the visualized predicted segmentations when trained on RGB and RGBD respectively, (e) and (f) show the predictions only within the annotated regions, and (g) and (h) show the corresponding confidence of the softmax probabilities of the predicted classes. The "heatmaps" with which we visualize confidence range from blue, for low probabilities, to green, for probabilities around 50%, to red, for high probabilities. The center two columns show messy predictions, and the confidence indicate that the network is generally uncertain about its predictions.

be perceived similarly as noise by Semantic FPN and are too detailed and of too low a resolution to be usable. It is no wonder that RGBD does not perform better for these image regions; a small image region with a high variance in RGB values (i.e. a cluttered image region) is likely to entail a high variance in the corresponding depth map, and the depth map has a comparatively low depth resolution at large values. For pixels where ground truth is given, the network predicts correct in 86% of the time. The tiled sidewalk and the stone curb perform worse, though. The confidence (visualized as "heatmaps" in Figures 47g and 47h) indicate the network's certainty within large segmentations, and its uncertainty on borders and within smaller, more cluttered segmentations. This is in accordance with the training data, which focuses on large regions of uniform material rather than smaller regions. Especially the distant horizon is rarely annotated, mainly because the effort yields less ground truth than the large patches, but also because far-away materials are more difficult to determine. Another reason for the uncertain borders could be due to the training data giving the network little to no concept of pixel-wise boundaries, since its only frame of reference consists of variably shaped segments of singular materials. The actual boundaries between materials in an image are rarely covered with ground truth, and we therefore already expected the network to be bad at predicting border regions. The confidence shows the network's uncertainty around segmentation edges, of which it has seldom seen the patterns. Obviously, the differences in depth are then also not taken into account, while boundaries are potentially even more distinct in depth maps. We expect full image semantic training data to significantly improve segmentation quality. A more thorough account of our data is given in Section 4.1. More segmentations can be found in Appendix F.

Contrary to expectations, this experiment shows how the segmentation quality between RGB and RGBD predictions is nearly identical, especially for the large classes. It could be that depth maps either possess less discriminitive value than we expected, or their discriminitive value is made superfluous by the presence of RGB information. The similarities between the two become more apparent in Table 6. This table shows how the segmentations of some classes improve marginally, while those of others slightly deteriorate. We reserve the high similarity of both input types for later, when we investigate SUBQUESTION II, and instead focus first on class imbalance.

Overall mIoU for this experiment is a rather low 0.49 for RGB and 0.51 for RGBD. In general, we can observe a trend of a decreasing IoU with a decreasing class size. There are some notable exceptions to this rule, though. `tile` performs bad

	RGB			RGBD			RGBD - RGB		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.494	0.636	0.576	0.507	0.630	0.594	+0.013	-0.006	+0.018
Weighted overall	0.809	0.879	0.886	0.802	0.874	0.882	-0.007	-0.005	-0.004
brick	0.832	0.899	0.918	0.820	0.893	0.909	-0.012	-0.006	-0.009
sky	0.982	0.998	0.984	0.980	0.998	0.981	-0.002	0.000	-0.003
asphalt	0.858	0.907	0.940	0.832	0.880	0.937	-0.026	-0.027	-0.003
tile	0.597	0.796	0.704	0.583	0.807	0.677	-0.014	+0.011	-0.027
foliage	0.826	0.890	0.920	0.862	0.914	0.938	+0.036	+0.024	+0.018
cloud	0.963	0.967	0.996	0.958	0.962	0.995	-0.005	-0.005	-0.001
grass	0.715	0.785	0.889	0.663	0.766	0.831	-0.052	-0.019	-0.058
glass	0.674	0.753	0.865	0.702	0.784	0.870	+0.028	+0.031	+0.005
stone	0.340	0.563	0.461	0.369	0.578	0.505	+0.029	+0.015	+0.044
steel	0.569	0.712	0.739	0.596	0.733	0.761	+0.027	+0.021	+0.022
concrete	0.824	0.924	0.883	0.819	0.921	0.881	-0.005	-0.003	-0.002
water	0.563	0.837	0.632	0.644	0.792	0.776	+0.081	-0.045	+0.144
wood	0.402	0.683	0.495	0.473	0.740	0.568	+0.071	+0.057	+0.073
soil	0.265	0.501	0.359	0.191	0.348	0.298	-0.074	-0.153	-0.061
other	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
aluminium	0.412	0.509	0.685	0.432	0.558	0.657	+0.020	+0.049	-0.028
gravel	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
iron	0.000	0.002	0.000	0.004	0.022	0.005	+0.004	+0.020	+0.005
living	0.447	0.779	0.512	0.541	0.833	0.607	+0.094	+0.054	+0.095
fabric	0.105	0.843	0.108	0.048	0.455	0.051	-0.057	-0.388	-0.057
plastic	0.002	0.004	0.006	0.134	0.242	0.232	+0.132	+0.238	+0.226

Table 6: Evaluation metrics of training Semantic FPN on 600 images of our material dataset. Results are given both for a pipeline with RGB as input as well as RGBD. The overall metrics are unweighted averaged: mIoU, macro-averaged precision and macro-averaged recall, respectively. Naturally, the weighted overall metrics are fIoU, weighted macro-averaged precision and weighted macro-averaged recall. The materials are sorted descendingly by size; their number of available ground truth pixels in the training set. The cells are color coded to ease interpretation, with the colors ranging from light yellow (for zero values) to red (for 1's). RGB and RGBD perform highly similar, with some classes benefitting slightly from the additional depth map, and some being worse off. The third columns gives the difference between performance on the two input types.

compared to its neighbours of similar size. Looking at the confusion matrices in Table 8, it can be seen that approximately one-seventh of tile's pixels are confused as brick, which makes sense given the similarity of their pattern. Conversely, approximately a tenth of all tile predictions were actually brick. To a lesser extent, a similar confusion between classes happens between tile and stone, which is also understandable. stone largely consists of curbs, curb cuts or ramps, and small barriers, all of which appear relatively smooth and light grey (see Figure 45, which contains two stone samples, though from different lighting conditions). Tiles, more so than bricks, are often of the same appearance, which provides a possible explanation for the confusion. stone itself also performs badly relative to its class size. Naturally, the confusions between tile and stone will reflect even worse on stone, since is stone is smaller than tile. However, stone is most often misclassified as brick, even though we would say that stone and brick are less similar than stone and tile. We conjecture that this is due to brick's overwhelming class size. For RGB, plastic, iron, gravel, other are barely predicted, and significantly impact mIoU.

There are also classes that perform better than their size suggests: concrete, living, fabric, and aluminium are the most notable ones. The latter is easy to explain; aluminium almost exclusively comprises traffic signs, which have very recognizable patterns and colours. fabric, on the other hand, seems to perform well relative to its size, though this is partly an illusion caused by its high precision. Precision is less interesting than recall, and fabric exemplifies why. fabric only has 27,300 pixels annotated in the test set, according to Table 7. This is the same as a 165x165 square image, and only about a tenth of these pixels have been correctly detected with RGB input. In other words, the network predicted 3,497 pixels as fabric, and even though 2,951 of these were correct, $27,300 - 2,951 = 24,349$ pixels were missed.

aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	stone	steel	tile	water	wood
0.004	0.179	0.307	0.07	0.011	0.003	0.05	0.015	0.026	0.0001	0.002	0.003	0.005	0.0003	0.196	0.005	0.023	0.024	0.075	0.002	0.006

(a) Class size ratios; the ratio of a class's number of ground truth pixels to the total number of ground truth pixels.

aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	stone	steel	tile	water	wood
39,487	1,707,777	2,934,132	664,716	100,685	27,300	477,708	145,727	244,105	1,064	14,609	31,918	4,490	3,421	1,876,419	48,792	216,358	228,386	721,283	18,817	57,269

(b) Class sizes; expressed in number of ground truth pixels.

Table 7: Two tables, giving the sizes of the material classes within our test set in three different representations. (a) gives the relative class ratios. (b) gives the absolute number of ground truth pixels.

A first glance at the confusion matrices in Table 8 suggests decent segmentation quality, as most predictions are in the matrix diagonal. The sum of the diagonal – the number of correct predictions, or true positives – divided by the total number of ground truth pixels gives our accuracy. For both RGB and RGBD this accuracy is 0.89, meaning 89% of all pixels are predicted correctly. mIoU is still a low 0.49 for RGB and 0.51 for RGBD. The difference between these two metrics highlights an important observation. As discussed in Section 4.2, mIoU averages IoUs equally over classes, whereas accuracy is a percentage of the number of pixels in an image. This tells us that the accuracy is dominated by the large classes, which the network has the least trouble predicting and give the accuracy its high value. Contrast this with mIoU, which is being held back by the small classes with low IoUs. The network's difficulty of segmenting small classes is therefore at least partially responsible for the low mIoU. This can be read from the confusion matrices; there seems to be a proportional relation between the recall of a class (indicated by the redness of the class's cell in the diagonal) and its size.

Cells outside of the diagonal that show a large recall indicate a class which the network often confuses with another. Looking at the matrix for RGB input, most striking seems to be the mistaking of other pixels for brick and of gravel for stone, though the former ameliorates with additional depth input (indicating that depth helps distinguish other from brick) and the latter only concerns a relatively small number of mispredictions. Another salient confusion is that of both fabric and aluminium being predicted as steel. For fabric, this was to be expected, since plenty of steel annotations are painted or printed, in which case it resembles a coloured fabric. steel is the larger class, and it therefore makes sense that the network is biased towards steel and mistakes fabric for steel instead of the other way around. The confusion of aluminium and steel is more puzzling, since the aluminium class almost exclusively consists of traffic signs, which have, again, easily recognizable and repetitive colour patterns, and is therefore expected to be easily distinguishable from steel. However, traffic signs are frequently turned away from the camera, in which case the backside is visible. In nearly all cases, this backside is a smooth grey, which is easy to mistake for steel. The worst confusion, however, is that of iron being mistaken for both asphalt and brick; together they form 10,830 missed pixels out of the total 14,909. iron consists mostly of manholes, which are in turn mostly embedded in either asphalt or brick roads. A smaller portion of iron consists of catch basins, which in The Netherlands are most often built into curbs. Visually, iron is rather distinct from the classes with which it is confused, and we have trouble coming up with a reason. A possible explanation is that iron segmentations are simply too small. The larger scales of Semantic FPN's feature pyramid contain the fine details, like manholes, but these are mixed with the low-resolution feature scales (which contain more semantic information). The iron segmentations could potentially get muddled or faded by the large, encompassing asphalt or brick segmentations.

We ran the same experiment again, and determined that the small classes are rather volatile, i.e. the distribution of predictions among the small classes differs measurably between two runs of same experiment. This makes it difficult to draw any conclusions regarding those classes regarding the pattern on where and how the network makes its mistakes. We printed the results of this second run in Appendix G.3.

4.4.2 SUBQUESTION I, concerning class imbalance

At this point, the most pressing issue is that of the class imbalance hampering our performance. Our first attempt at mitigating class imbalance is to use per-class loss weights proportional to class size. All other hyperparameters and settings remain the same. The weight of a class is the ratio of its size to the size of the smallest class. Class weighing

Predicted labels

True labels

	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	27060	53	400	67	557	34	896	738	106	0	0	562	0	31	0	12	7760	71	267	45	828	39487
asphalt	0	1606151	80027	0	0	0	155	5	4209	0	0	0	0	88	0	1989	649	9150	5285	0	69	1707777
brick	4290	102103	2693963	0	2079	41	2592	15911	2510	39	442	174	162	0	781	3679	10255	18906	71139	25	5041	2934132
cloud	0	0	76	662268	0	0	94	0	0	0	0	0	0	0	1923	0	150	0	205	0	0	664716
concrete	38	0	5720	0	89000	119	103	985	54	27	8	0	0	16	27	0	3549	0	1018	0	21	100685
fabric	8603	0	1302	38	206	2951	13	7880	86	0	0	21	154	0	249	0	3846	1354	571	9	17	27300
foliage	272	97	2997	133	532	0	439810	154	23073	20	10	333	0	34	3	1107	5088	2249	457	171	1168	477708
glass	2508	6	5635	0	9	194	173	126184	151	0	0	727	89	178	0	0	6495	54	2460	0	864	145727
grass	26	1175	2854	0	1554	0	6541	6	217142	0	0	22	0	11	0	911	3701	5940	3203	458	561	244105
gravel	0	0	22	0	0	0	49	0	0	0	0	0	0	0	0	0	69	820	104	0	0	1064
iron	0	5076	5754	0	5	0	20	0	104	0	9	5	0	0	0	0	49	2425	1129	0	33	14609
living	283	1063	1837	0	0	9	1161	3650	37	32	69	16352	1183	87	117	11	3377	200	2389	0	61	31918
other	0	12	3256	0	0	0	64	0	15	0	9	0	0	17	0	0	84	39	990	0	4	4490
plastic	204	1	389	503	0	15	87	0	0	0	0	741	0	21	0	0	1032	37	313	0	78	3421
sky	95	0	227	20195	0	16	7680	0	0	0	0	0	121	0	1847543	0	129	0	413	0	0	1876419
soil	70	10226	4227	0	0	0	902	0	9981	0	0	3	0	0	0	17560	1602	2242	1005	345	629	48792
steel	7901	2198	16955	1543	1789	116	1870	4636	2232	1	1130	1402	0	2165	15	227	160061	4736	5496	93	1792	216358
stone	335	19635	49254	0	135	0	813	940	5852	48	1912	509	18	2444	74	3352	3897	105369	32006	1018	775	228386
tile	631	21654	115390	36	360	2	21258	301	10431	88	187	22	57	0	0	6165	3375	32954	508135	29	208	721283
water	169	217	424	0	0	0	4017	548	0	0	0	0	0	0	0	0	56	107	382	11904	993	18817
wood	665	30	5165	7	21	0	5746	5576	530	0	445	100	0	81	0	8	9365	224	813	123	28370	57269
Total	53150	1769697	2995874	684790	96247	3497	494044	167514	276513	255	4221	20973	1784	5173	1850732	35021	224589	186877	637780	14220	41512	9564463

(a) RGB

Predicted labels

True labels

	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	25959	3	344	98	386	287	1428	611	110	0	0	45	0	28	10	0	8117	515	106	3	1437	39487
asphalt	59	1601384	83862	0	0	0	212	77	5129	0	2	0	10	12	0	1909	1157	10574	3196	31	163	1707777
brick	2374	156426	2669142	0	844	0	1295	5919	3019	0	571	877	14	111	68	2460	7537	19244	60540	1213	2478	2934132
cloud	0	0	60	661891	89	0	0	41	0	0	0	0	0	0	1762	0	481	0	392	0	0	664716
concrete	0	0	2528	0	88713	0	143	856	0	0	0	0	0	12	22	0	6714	0	1697	0	0	100685
fabric	4531	1345	1665	0	60	1393	80	10608	0	0	8	0	116	0	213	427	6099	0	329	0	426	27300
foliage	1422	542	2282	163	680	0	448399	1014	13759	0	49	217	17	54	1	1056	3371	2363	475	213	1631	477708
glass	2232	43	3984	0	1024	343	703	126881	0	0	0	1452	13	0	176	0	6913	726	874	0	363	145727
grass	6	2543	5981	0	1672	0	13142	58	202856	0	84	0	0	0	0	3742	3274	6208	2848	751	940	244105
gravel	0	0	13	0	0	0	44	0	0	0	0	0	0	0	0	0	10	724	273	0	0	1064
iron	0	4236	6400	0	0	0	0	6	125	0	81	10	0	0	0	5	118	2257	1371	0	0	14609
living	432	501	4395	0	0	0	203	3325	55	0	37	19386	0	77	647	10	1153	310	1208	30	149	31918
other	0	12	1641	0	0	0	29	0	0	0	3	0	0	0	0	0	29	572	2179	25	0	4490
plastic	14	0	203	417	0	0	88	52	0	0	64	298	41	797	0	0	1107	180	160	0	0	3421
sky	0	0	32	25145	0	0	7691	23	0	0	0	0	0	0	1842102	0	85	0	1341	0	0	1876419
soil	0	9498	2737	0	0	0	222	0	15643	0	0	7	0	0	0	14582	2114	2078	1085	0	826	48792
steel	8246	2088	11604	64	2652	886	2400	5072	2792	0	347	800	23	954	11	383	164830	3564	6757	832	2053	216358
stone	157	14851	50315	0	14	0	607	938	7297	0	495	147	0	1112	85	2048	3514	115396	30665	557	188	228386
tile	628	24738	128934	10	119	79	9031	1524	12910	73	987	3	0	0	72	15122	4269	34048	488560	23	153	721283
water	363	1	708	0	0	0	152	0	268	0	837	0	0	95	0	0	822	359	0	14608	604	18817
wood	77	63	10292	0	39	70	4596	4640	694	0	11	7	0	39	10	55	2947	200	843	156	32530	57269
Total	46500	1818274	2987122	687788	96292	3058	490465	161645	264657	73	3576	23249	234	3291	1845179	41799	224661	199318	604899	18442	43941	9564463

(b) RGBD

Table 8: Confusion matrices for predictions made by Semantic FPN on our material dataset with RGB and RGBD input. A cell's color reflects the ratio of its number of predictions to the total number of ground truth pixels for the corresponding true class. For cells in the diagonal, this ratio equals the class's precision.

increases the loss taken over pixels belonging to a small class, and devaluates the loss for large class pixels. This increases gradient magnitude for the small classes, allowing those to learn more and take steeper gradient descent steps than without class weights. The class weights, together with class sizes within our training set, are given in Table 9a.

The results in Table 10 show how a class-weighted cross entropy loss performs marginally better. The small classes are most affected by the weighted loss, which is to be expected, though they not always change for the better. There does not seem to be a coherent pattern in the difference between class-weighted and unweighted loss, both between classes and between RGB and RGBD; it seems rather arbitrary. fabric sees a significant increase in precision. In order to avoid any unnecessary analysis, we also ran this exact experiment again, with the outcomes, differences to the current experiment, and a small analysis located in Appendix G.4. This time also, the small classes appear highly volatile. We think that using class weights is insufficient, given the extremity of our class imbalance (as indicated by the ratios in Table 9b). Literature agrees that class-weighted loss function are suitable for datasets with a slight imbalance, but does not work well for datasets with a severe imbalance [75].

We next try Dice loss. Since we express performance in mIoU, and since Dice loss theoretically optimizes for mIoU directly, Dice loss is expected to be a viable alternative. The results are disappointing however. The results for both class-weighted and unweighted Dice loss are summarized in Table 11, and a more complete account can be found in Appendix G.5 and G.6. Apparently, Dice loss's idea of increasing mIoU is to not predict most classes, and instead to gamble on others. It is strange, however, that it neglects to predict brick, the biggest class, but chooses to predict living – it manages to detect half of the latter class. We ran the experiment before, though with a different learning rate and batch size, and found that another set of classes were not predicted. When combining Dice loss with class weights, performance increases somewhat, yet again a different set of classes is skipped.

aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel
0.166	0.004	0.003	0.012	0.049	0.805	0.011	0.03	0.013	0.396

iron	living	other	plastic	sky	soil	stone	steel	tile	water	wood
0.402	0.516	0.146	1	0.004	0.09	0.035	0.034	0.011	0.071	0.088

(a) Class weights; the weight of a class is the ratio of its size to the size of the smallest class.

aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel
0.004	0.182	0.256	0.061	0.015	0.001	0.065	0.024	0.055	0.002

iron	living	other	plastic	sky	soil	stone	steel	tile	water	wood
0.002	0.001	0.005	0.0007	0.189	0.008	0.021	0.021	0.067	0.010	0.008

(b) Class size ratios; the ratio of a class's number of ground truth pixels to the total number of ground truth pixels.

aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel
258,488	10,677,635	15,048,695	3,605,311	884,032	53,374	3,824,702	1,427,588	3,240,765	108,582

iron	living	other	plastic	sky	soil	stone	steel	tile	water	wood
106,922	83,242	293,620	42,988	11,127,619	475,160	1,212,214	1,249,003	3,931,732	608,546	487,980

(c) Class sizes; expressed in number of ground truth pixels.

Table 9: Three tables, giving the sizes of the material classes within our training set in three different quantities. (a) gives the weights as we use them for weighting our loss. (b) gives the relative class ratios. (c) gives the absolute number of ground truth pixels.

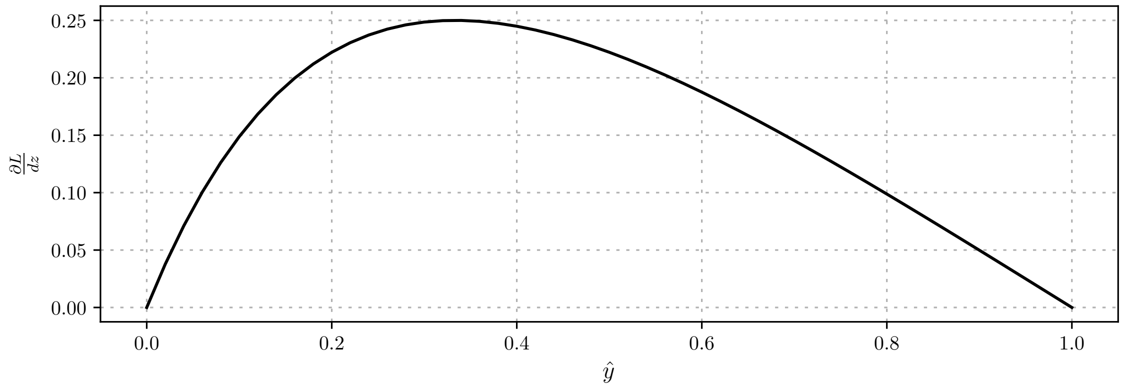


Figure 48: A plot of the gradient of Dice loss with respect to logits against the probability resulting from the logits. Ideally, this plot should be monotonically decreasing, since Dice loss only optimizes for probabilities of true labels (i.e. true positives or false negatives). Now, false negatives get the same loss as true positives, which results in the low recall we observe when using Dice loss.

The derivative of Dice loss with respect to the logits z is:

$$\frac{\partial L_{\text{Dice}}}{\partial z} = \frac{-2y\hat{y}^2(1-y)}{(y+\hat{y})^2} \tag{22}$$

which results in unstable training; whenever both y and $\hat{y}$ are small, the gradient quickly becomes unwieldily large. Compare this to the more stable binary cross entropy loss, whose gradient with respect to the logits is:

$$\frac{\partial L_{\text{CE}}}{\partial z} = y - \hat{y} \tag{23}$$

Also, Dice loss only gives a gradient for true positives and false negatives, since there is no gradient whenever $\hat{y} = 0$. For all possible probabilities of a true class, the gradient of Dice loss with respect to logits is plotted against probability in Fig 48. This function should be monotonically decreasing, but is not. This means that false negatives are getting incorrect gradients, which results in a low recall. The stronger optimization of true positives reduces false positives, improving the precision. This is a possible explanation for Dice loss's erratic behaviour.

	Class-weighted cross entropy loss						Difference with unweighted cross entropy loss					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.509	0.659	0.591	0.503	0.646	0.586	+0.015	+0.023	+0.015	-0.004	+0.016	0.000
Weighted overall	0.808	0.879	0.886	0.807	0.880	0.886	-0.001	0.000	0.000	+0.005	+0.006	+0.004
brick	0.830	0.892	0.922	0.828	0.902	0.909	-0.002	-0.007	+0.004	+0.008	+0.009	0.000
sky	0.975	0.994	0.980	0.977	0.998	0.979	-0.007	-0.004	-0.004	-0.003	0.000	-0.002
asphalt	0.858	0.901	0.947	0.825	0.881	0.928	0.000	-0.006	+0.007	-0.007	+0.001	-0.009
tile	0.604	0.816	0.698	0.621	0.808	0.728	+0.007	+0.020	-0.006	+0.038	+0.001	+0.051
foliage	0.843	0.903	0.926	0.849	0.894	0.944	+0.017	+0.013	+0.006	-0.013	-0.020	+0.006
cloud	0.945	0.957	0.986	0.952	0.954	0.997	-0.018	-0.010	-0.010	-0.006	-0.008	+0.002
grass	0.691	0.795	0.841	0.711	0.827	0.835	-0.024	+0.010	-0.048	+0.048	+0.061	+0.004
glass	0.710	0.798	0.864	0.715	0.785	0.888	+0.036	+0.045	-0.001	+0.013	+0.001	+0.018
stone	0.332	0.567	0.444	0.398	0.617	0.529	-0.008	+0.004	-0.017	+0.029	+0.039	+0.024
steel	0.573	0.703	0.757	0.614	0.718	0.809	+0.004	-0.009	+0.018	+0.018	-0.015	+0.048
concrete	0.806	0.870	0.916	0.829	0.908	0.905	-0.018	-0.054	+0.033	+0.010	-0.013	+0.024
water	0.652	0.774	0.805	0.467	0.614	0.660	+0.089	-0.063	+0.173	-0.177	-0.178	-0.116
wood	0.406	0.709	0.487	0.440	0.720	0.531	+0.004	+0.026	-0.008	-0.033	-0.020	-0.037
soil	0.237	0.531	0.299	0.339	0.636	0.421	-0.028	+0.030	-0.060	+0.148	+0.288	+0.123
other	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
aluminium	0.450	0.600	0.643	0.501	0.671	0.665	+0.038	+0.091	-0.042	+0.069	+0.113	+0.008
gravel	0.143	0.542	0.163	0.000	0.000	0.000	+0.143	+0.542	+0.163	0.000	0.000	0.000
iron	0.006	0.062	0.007	0.044	0.297	0.049	+0.006	+0.060	+0.007	+0.040	+0.275	+0.044
living	0.394	0.764	0.449	0.353	0.695	0.418	-0.053	-0.015	-0.063	-0.188	-0.138	-0.189
fabric	0.235	0.660	0.268	0.092	0.639	0.098	+0.130	-0.183	+0.160	+0.044	+0.184	+0.047
plastic	0.003	0.008	0.004	0.007	0.013	0.017	+0.001	+0.004	-0.002	-0.127	-0.229	-0.215

Table 10: Results obtained by training Semantic FPN with class-weighted cross entropy loss for both RGB and RGBD input. The differences with unweighted cross entropy loss are on the right hand side.

The last loss function we try is focal loss. Focal loss downweights the contribution of samples for which the softmax probability is high, which indicates that the sample is already easy to predict. In this way, the more difficult samples – of which the network is uncertain – get a higher priority. The confidence in Figures 47g and 47h show the network's uncertainty on small or cluttered regions, and these are the pixels which will therefore contribute more to the loss. Ideally, this makes the network more skilled at predicting the smallest classes, which in turn decreases their contribution and makes the network focus on the next difficult class.
Table 12 contains the results. Performance is slightly worse than that of cross entropy loss, though not nearly as bad as Dice loss. Instead of getting better at predicting smaller classes than cross entropy loss, this loss is less capable at it. Results of weighted focal loss are in Appendix G.8. Performance of class-weighted focal loss is highly similar, though the deviations in the small classes are slightly exacerbated.

We conjecture that the class imbalance is so severe that it renders our dataset unusable. The volatility of the small classes in our cross entropy experiments suggests the same. Table 9a shows how the ratio of the smallest to the largest class is 0.003. fabric and plastic both have around 50,000 pixels with which they are trained. Besides the imbalance in the training set, we also evaluate with very little data. Table 7b tells us that gravel and plastic are evaluated on only 3,421 and 1,064 ground truth pixels, respectively.

A potential solution aimed at improving the performance of the small classes would be to predict those classes using an instance segmentator, e.g. Mask R-CNN [37], or even using object detectors with an added segmentation head, e.g. RetinaNet [57]. The predicted instances can then be flattened into a semantic segmentation map and merged with the semantic predictions of the other material classes. Most of the small classes largely consists of small, recognizable objects. For instance, the majority of plastic's annotation exist in the form of traffic lights, traffic cones, and small movable guiding signs as shown in Figure 45 as the left plastic example. iron exclusively consists of catch basins, fire hydrants and manholes, and fabric contains flags, billboards and banners. Especially RetinaNet, when used with the focal loss with which it was introduced, is adept at detecting infrequently occurring objects. The others materials are semantically segmented in isolation, after which the resulting segmented image is "overwritten" with the instance segmentations of the small classes. Evaluating this method is outside the scope of this research, however.

In order to analyze any subsequent experiment without the negative effects of class imbalance, we resort to merging the smallest, worst performing classes. Those with mIoU near zero across experiments are plastic, fabric, iron, gravel, and other. We attempt three methods of eliminating these five classes: merging them into a background

	Unweighted Dice loss						Class-weighted Dice loss					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.330	0.780	0.387	0.420	0.783	0.474	0.426	0.835	0.480	0.468	0.787	0.524
Weighted overall	0.554	0.615	0.593	0.631	0.716	0.685	0.766	0.849	0.857	0.784	0.866	0.872
brick	0.000	—	0.000	0.783	0.874	0.882	0.764	0.817	0.921	0.769	0.842	0.899
sky	0.985	0.997	0.987	0.981	0.996	0.984	0.984	0.997	0.986	0.985	0.999	0.985
asphalt	0.823	0.889	0.917	0.000	—	0.000	0.814	0.870	0.926	0.791	0.843	0.928
tile	0.484	0.796	0.553	0.550	0.828	0.621	0.566	0.840	0.634	0.562	0.811	0.646
foliage	0.882	0.943	0.931	0.847	0.899	0.935	0.867	0.901	0.958	0.885	0.930	0.948
cloud	0.970	0.978	0.991	0.958	0.969	0.988	0.966	0.975	0.990	0.974	0.976	0.997
grass	0.654	0.803	0.778	0.628	0.761	0.783	0.677	0.773	0.844	0.712	0.815	0.849
glass	0.720	0.813	0.862	0.739	0.849	0.851	0.748	0.817	0.898	0.700	0.765	0.890
stone	0.000	—	0.000	0.360	0.658	0.443	0.321	0.631	0.395	0.364	0.654	0.450
steel	0.571	0.776	0.684	0.589	0.795	0.694	0.578	0.741	0.724	0.619	0.766	0.763
concrete	0.000	—	0.000	0.780	0.945	0.817	0.000	—	0.000	0.804	0.967	0.826
water	0.000	—	0.000	0.416	0.990	0.418	0.231	0.978	0.233	0.681	0.903	0.734
wood	0.243	0.870	0.253	0.269	0.927	0.275	0.284	0.922	0.291	0.266	0.873	0.277
soil	0.000	—	0.000	0.000	—	0.000	0.010	0.444	0.010	0.101	0.458	0.115
other	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000
aluminium	0.000	—	0.000	0.566	0.801	0.659	0.552	0.766	0.665	0.514	0.810	0.584
gravel	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000
iron	0.112	0.578	0.122	0.064	0.293	0.075	0.000	—	0.000	0.110	0.961	0.111
living	0.494	0.910	0.519	0.284	0.944	0.289	0.555	0.946	0.573	0.000	—	0.000
fabric	0.000	—	0.000	0.000	—	0.000	0.029	0.946	0.029	0.000	—	0.000
plastic	0.000	0.000	0.528	0.000	0.000	0.242	0.000	—	0.000	0.000	0.000	0.000

Table 11: Per-class results of training Semantic FPN with both unweighted and class-weighted Dice loss on RGB and RGBD input.

class, renaming other to background and removing the others, and removing all of them. All these methods are tried with our four best performing loss functions: both unweighted and class-weighted cross entropy loss, and both unweighted and class-weighted focal loss. The results per loss function are in Appendices G.9, G.10, G.11, and G.12, respectively. For all three methods and all four loss functions, mIoU increases significantly, although this is largely due to the absence of any non-performing classes. The performance of the remaining classes has not changed much. When merging the omitted classes into background and thereby increases its size, it now gets a small number of predictions across all experiments, even though the recall of these classes is insignificantly small. Two experiments are tied in terms of mIoU on RGB input: cross entropy loss without background, and class-weighted cross entropy loss where the background is made up of all irrelevant classes (in order to serve as a catch-all for anything that does not fall under any of the remaining classes). Since the former is technically an incorrect way of segmenting an image, since there is no way of correctly labeling pixels from one of the deleted classes, our preference goes towards the latter method: class-weighted cross entropy loss with a composite background class. This is also the method that performs the best on RGBD input, which further confirms our choice.

We can now answer SUBQUESTION I: the most optimal measures to combat the adverse effects of class imbalance on material segmentation performance – as far as we have found – are to combine the worst five performing classes into a background class, and to use class-weighted cross entropy loss. Using instance segmentation for the small classes is a promising alternative, but we can only speculate as to its effectiveness in our situation.

4.4.3 SUBQUESTION II, concerning the difference between RGB and RGBD performance

Having obtained an optimal loss function in class-weighted cross entropy, and a way to remedy class imbalance, we next investigate why material segmentation performance on RGB and RGBD input are as similar as they are. The method from the previous section becomes our new baseline to which we compare the experiments in this section.
We first ascertain whether a depth channel holds any discriminatory power in the first place; if it does not, then that already answers SUBQUESTION II. This we do by training only on depth. The results are given in Table 13, which shows the outcome using three different methods of depth weight initialization. It seems that predicting based solely on depth is feasible to some extent. Depth is not nearly as indicative of material than colour is, but that was to be expected based on visual inspections like those in Figure 4. These findings are corroborated by comparing the activation maps of training on

	Unweighted focal loss						Difference with unweighted cross entropy loss					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.470	0.666	0.555	0.434	0.554	0.517	-0.024	+0.030	-0.021	-0.073	-0.076	0.000
Weighted overall	0.782	0.859	0.870	0.771	0.849	0.860	-0.027	-0.020	-0.016	-0.031	-0.025	-0.022
`brick`	0.797	0.881	0.893	0.792	0.874	0.894	-0.035	-0.018	-0.025	-0.028	-0.019	-0.015
`sky`	0.978	0.998	0.980	0.980	0.996	0.983	-0.004	0.000	-0.004	0.000	-0.002	+0.002
`asphalt`	0.816	0.869	0.930	0.814	0.880	0.914	-0.042	-0.038	-0.010	-0.018	0.000	-0.023
`tile`	0.568	0.779	0.677	0.547	0.739	0.678	-0.029	-0.017	-0.027	-0.036	-0.068	+0.001
`foliage`	0.851	0.914	0.926	0.844	0.900	0.932	+0.025	+0.024	+0.006	-0.018	-0.014	-0.006
`cloud`	0.955	0.958	0.996	0.956	0.964	0.991	-0.008	-0.009	0.000	-0.002	+0.002	-0.004
`grass`	0.618	0.721	0.812	0.597	0.698	0.804	-0.097	-0.064	-0.077	-0.066	-0.068	-0.027
`glass`	0.668	0.750	0.860	0.659	0.739	0.859	-0.006	-0.003	-0.005	-0.043	-0.045	-0.011
`stone`	0.293	0.510	0.408	0.247	0.449	0.355	-0.047	-0.053	-0.053	-0.122	-0.129	-0.150
`steel`	0.529	0.662	0.725	0.525	0.668	0.711	-0.040	-0.050	-0.014	-0.071	-0.065	-0.050
`concrete`	0.740	0.846	0.855	0.664	0.783	0.813	-0.084	-0.078	-0.028	-0.155	-0.138	-0.068
`water`	0.672	0.747	0.869	0.358	0.604	0.468	+0.109	-0.090	+0.237	-0.286	-0.188	-0.308
`wood`	0.409	0.631	0.537	0.353	0.631	0.445	+0.007	-0.052	+0.042	-0.120	-0.109	-0.123
`soil`	0.186	0.375	0.270	0.134	0.314	0.189	-0.079	-0.126	-0.089	-0.057	-0.034	-0.109
`other`	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
`aluminium`	0.398	0.708	0.476	0.373	0.579	0.512	-0.014	+0.199	-0.209	-0.059	+0.021	-0.145
`gravel`	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
`iron`	0.000	0.000	0.000	0.001	0.032	0.002	0.000	-0.002	0.000	-0.003	+0.010	-0.003
`living`	0.293	0.689	0.337	0.281	0.784	0.304	-0.154	-0.090	-0.175	-0.260	-0.049	-0.303
`fabric`	0.106	0.617	0.114	0.000	0.004	0.000	+0.001	-0.226	+0.006	-0.048	-0.451	-0.051
`plastic`	0.000	0.000	0.000	0.000	0.000	0.000	-0.002	0.000	-0.006	-0.134	-0.242	-0.232

Table 12: Results of Semantic FPN working with unweighted focal loss, and the difference between focal loss and unweighted cross entropy loss.

depth in Appendix G.13 and G.10, which shows how the activations of colour information hold more semantic information than those resulting from depth.

A quick glance is enough to designate which classes have the most distinct depth characterstics: `foliage`, which has by far the most coarse and bumpy texture pattern (see Figure 4), and `sky`, of which the depth is strictly zero. It is interesting to see how `sky` performs better than `cloud`. Both classes have the exact same depth, but `sky` is the larger class, hence its better performance. It is strange that, given this fact, `cloud` still performs reasonably well. It is our guess that the network predicts randomly according to the ground truth distribution of the two respective classes. We also noticed that, more often than not, clouds appear in the lower half of aerial pixel regions, which might entice the network to develop a bias towards `cloud` at lower parts of image regions where depth is zero. Indeed we see that `cloud` predictions are often in the lower regions. They also somewhat adhere to the shapes of our annotations, which may indicate that `cloud` is inferred purely on the basis of the shape of segmentations and the location of those segmentation in the images.

`glass`, `concrete`, and `water` all have a high recall, but a low precision. This is because the network predicts those classes rather carelessly, and therefore incurs a lot of mispredictions. `water` and `glass` both have noisy depth segmentations; Figure 49 shows examples. LiDAR signals are reflected away from the sensor when colliding with `water`. Therefore, `water` pixels should all have a depth of zero. However, many objects that are reflected in the water in an RGB image are also present in the depth map, and are captured relatively accurately at that. It cannot be that the light signals are bouncing back and forth over the water and return to the sensor, since the chance of a signal following the exact same path back over water is nil. We therefore suspect these depth reflections to be postprocessing artifacts. This is not necessarily an issue, especially since `sky` and `cloud` already have zero depth. The combination of zero depth pixels interspersed with strictly positive depth pixels can ultimately be rather characterstics. Even though such a mixture of zero and positive depth values is also typical for `foliage`, which sometimes has sky visible through the leafs around the contours of trees, the network rarely confuses the two, as evident from the confusion matrix in Table 14. It may be that, because `water` can contain the depth map for whatever material or object is reflected in the water, instead indirectly learns those reflected materials. This could have led to the network overconfidently predicting such materials as `water`. A similar reasoning can be applied to `glass`. Light signals simply travel through it, and instead capture everything behind it, making glass invisible in depth maps.

The confusion matrix in Table 14 also shows the obvious confusion between `sky` and `cloud`. `soil` is frequently mistaken for `grass`, but not the other way around. This is similar to what we see for `cloud` and `sky`; `grass` is the

	Reflection initialized			Average RGB initialized			Xavier initialized		
	D			D			D		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.399	0.547	0.561	0.423	0.559	0.598	0.404	0.550	0.564
Weighted overall	0.529	0.674	0.685	0.553	0.694	0.709	0.537	0.678	0.695
`brick`	0.527	0.689	0.691	0.540	0.712	0.692	0.530	0.703	0.684
`sky`	0.770	0.867	0.873	0.793	0.868	0.902	0.790	0.852	0.916
`asphalt`	0.464	0.619	0.650	0.494	0.651	0.672	0.488	0.621	0.696
`tile`	0.306	0.508	0.435	0.294	0.508	0.411	0.246	0.478	0.336
`foliage`	0.770	0.839	0.903	0.799	0.860	0.918	0.780	0.863	0.891
`cloud`	0.456	0.641	0.612	0.470	0.693	0.593	0.452	0.711	0.554
`grass`	0.314	0.488	0.469	0.383	0.495	0.628	0.319	0.422	0.567
`glass`	0.451	0.531	0.751	0.514	0.600	0.783	0.455	0.583	0.675
`stone`	0.328	0.535	0.458	0.429	0.606	0.594	0.377	0.576	0.521
`steel`	0.398	0.589	0.551	0.455	0.620	0.630	0.388	0.525	0.598
`concrete`	0.471	0.524	0.822	0.472	0.529	0.814	0.526	0.623	0.771
`water`	0.484	0.560	0.781	0.488	0.542	0.830	0.481	0.546	0.800
`wood`	0.303	0.566	0.395	0.305	0.517	0.426	0.264	0.456	0.386
`soil`	0.191	0.426	0.256	0.137	0.306	0.199	0.179	0.547	0.210
`aluminium`	0.136	0.199	0.302	0.178	0.226	0.454	0.181	0.243	0.413
`living`	0.021	0.167	0.024	0.022	0.205	0.024	0.005	0.050	0.005
`background`	0.027	0.107	0.034	0.016	0.034	0.030	0.017	0.067	0.022

Table 13: Per-class results for training solely on depth, using three ways of initializing the weight of the first convolution. Reflection simply takes the weights of the RGB network's green channel, the second method averaged the pretrained RGB weights over the three channels, and the latter does not use any pretrained weights, but instead uses glorot initialization.

dominant class and therefore suffers less from the resemblance of their depth maps. Also striking are the many incorrect predictions for `brick`. It impacts the recall of `asphalt`, `living`, `tile`, and `wood`, but the precision of `brick` itself is still a reasonable 0.712. The confusion with `tile`, `asphalt`, and to a lesser extent with `wood`, are understandable, but the confusion with `living` is more puzzling. Whether a person or animal is captured in the depth map depends on the speed with which it moves. Even a person walking at a slow pace is rarely encoded. This is due to postprocessing; a cluster of points present in one point of view but absent in another are removed. It is fair to assume that most living being in the public space are moving, in which case the underlying depth is recorded, just like `glass`. If a lot of images depict people moving in front of brick buildings, then the network learns `living` as `brick.`

Looking at these results, it becomes clear why RGBD shows so little improvement. If the network already has much more discriminitive colour information at its disposal, it is no wonder that the less discriminitive depth information offers little additional help. Comparing the different initialization methods, we see that they are perform more similar than we expected – initialization is known to have a larger influence on performance than we observe here – but taking the average of the RGB channels does perform noticeably better. This concurs with expectation, as this method uses pretrained weights from all three colour channels, rather than only one channel or no channels. Our `foliage` example in Figure 49 illustrates how colour and depth tend to correspond in texture.

We trained two networks, one on RGBD with the depth weights Xavier initialized and one on RGBD with the average RGB weights (making this experiment similar to the previous one, except we add colours), in order to determine the effect of different weight initialization when colour information is present. Without reproducing the results here, we refer the reader to Appendices G.16 and G.17. In summary, while the method of initialization has some effect on performance when training only with depth, this effect is further diminished when RGB channels are used besides the depth. This reinforces our hypothesis that depth offers nearly no valuable information that colour has not already supplied. Also, the different initialization methods only affect a small portion of the weights: $64 * 7 * 7 * 1 = 3,136$ compared to the 38,535,953 weights in the rest of Semantic FPN, hence the insignificant difference between them.

We nevertheless experiment with Xavier initialized weights rather than weights pretrained on ImageNet. Such pretraining may start the network on a suboptimal location on the loss landscape as it exists with depth input. Results are in Appendix

		Predicted labels																
		background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
True labels	background	1536	1601	4385	20004	0	584	1579	7480	2750	311	178	197	3629	3303	2329	50	968	50884
	aluminium	1278	17914	0	1092	0	529	2522	2999	154	284	160	0	6501	103	1093	268	4590	39487
	asphalt	20320	14	1146896	434456	0	18473	840	543	19854	72	9	61	1835	10154	51237	2990	23	1707777
	brick	8195	54657	487777	2030477	6	24888	13023	44548	24281	292	2452	9058	15051	23119	188350	1844	6114	2934132
	cloud	1121	0	0	665	394392	2	16492	58	0	0	251129	0	660	0	197	0	0	664716
	concrete	3	5	18	9133	0	81965	482	411	50	619	647	0	6221	487	570	74	0	100685
	foliage	831	1521	155	3849	285	747	438528	949	6085	51	711	54	13689	988	2873	1622	4770	477708
	glass	778	62	8092	9831	195	5	1348	114068	379	185	164	0	9305	107	1187	0	21	145727
	grass	4144	19	16987	13350	0	4170	3944	105	153266	7	0	6553	8981	13253	17252	1647	427	244105
	living	191	188	177	13812	0	32	1858	5851	374	764	0	0	4425	1116	522	259	2349	31918
	sky	244	80	0	100	174111	299	7755	0	0	0	1693120	0	630	0	0	0	80	1876419
	soil	168	0	3448	717	0	0	2315	0	26465	71	0	9726	1195	2128	2033	221	305	48792
	steel	1424	2543	1001	23494	208	9658	8862	7642	8307	285	2132	456	136402	4176	5653	1351	2764	216358
	stone	2015	89	17441	32849	0	0	2303	132	15615	123	3	3041	4484	135613	13698	854	126	228386
	tile	1377	135	76548	233790	0	13713	5263	4664	51288	666	18	2624	3744	29058	296481	1740	174	721283
	water	1207	8	0	110	0	0	1481	0	31	5	0	0	250	0	23	15616	86	18817
	wood	487	378	76	25444	0	0	1209	751	556	0	0	8	2980	60	642	259	24419	57269
	Total	45319	79214	1763001	2853173	569197	155065	509804	190201	309455	3735	1950723	31778	219982	223665	584140	28795	47216	9564463

Table 14: Confusion matrix for training on depth, with weight initialized from the average of the RGB weights.

	RGB			RGBD			RGBD - RGB		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.489	0.495	0.961	0.473	0.479	0.945	-0.016	-0.016	-0.016
Weighted overall	0.592	0.595	0.990	0.589	0.592	0.989	-0.003	-0.003	-0.001
sky	0.737	0.739	0.996	0.733	0.735	0.995	-0.004	-0.004	-0.001
foliage	0.661	0.669	0.983	0.653	0.659	0.987	-0.008	-0.010	+0.004
concrete	0.201	0.203	0.951	0.203	0.205	0.944	+0.002	+0.002	-0.007
water	0.345	0.362	0.880	0.274	0.294	0.802	-0.071	-0.068	-0.078
brick	0.501	0.502	0.995	0.503	0.504	0.996	+0.002	+0.002	+0.001

Table 15: Per-class results.

G.18, which gives mIoUs of 0.36 and 0.39 for RGB and RGBD – compared to 0.49/0.51 with pretraining – which is a noticeably bigger difference. However, running the experiment again (Appendix G.19), this difference does not repeat, and we expect the difference in the first experiment to be due to luck. Since we use no pretraining, it can be interpreted that the network has no preconceived notions of patterns and object; that its learning process is starting from scratch, and it needs to learn more. We therefore repeated the experiment again with a learning rate of 0.1 (rather than 0.01, the default for cross entropy experiments). This did result in a large difference, though this time in favour of RGB instead of RGBD. These experiments thus do not offer a clue as to the cause of RGB and RGBD similarity. They do tell us that pretraining not only speeds up training, but also gives quite a significant performance boost. Without pretraining, our network is prone to overfitting given the scarce amount of training data, which is likely the cause to the performance we see here. Using a pretrained network prevents this, at least to some extent. Performance is significantly worse across the board, with the smallest classes being slightly more affected. It turns out that, in this instance, pretraining not only makes the network converge faster, but proves also essential for its performance. What's more, RGB and RGBD are still highly similar, and this experiment only emphasized the essence of pretraining.

We use nearest neighbour to downscale the depth maps as stored on disk to the network's input resolution. Purely to determine whether the method of scaling has any influence at all, we tried our baseline experiment, but with bilinear downsampling instead of nearest neighbour. Again, we leave the results for Appendix G.21, and note here that it barely influences material segmentation performance. This is not altogether strange, given the fact that the method of downscaling mostly affects edges of objects: bilinear interpolation will create non-existent surfaces between objects at different distances, while nearest neighbour does not. Since our dataset does not contain any boundaries, this has barely an effect.

Up until now, everything points to superfluity of depth when colour is supplied. A last experiment tests this. We select classes which, when predicted in isolation, have to increase in IoU with extra depth information: brick, concrete, foliage, sky, and water. We chose these classes based on their performance in Table 13, and ascertain that no pair is easy to confuse. With "predicting in isolation" we mean that we only work with these classes, i.e. no background class is used or evaluated with, and no other classes can therefore confuse the network. Class-weighted cross entropy loss is used. Results are in Table 15. Metrics for RGB and RGBD are, once again highly similar. We answer SUBQUESTION II therefore with the conclusion that additional depth offers no benefit to material segmentation performance, and that we suspect this to be because performance is already saturated on colour information. We reiterate that depth maps are

		Predicted labels					
		brick	concrete	foliage	sky	water	Total
True labels	brick	5803672	376185	223527	658452	28998	7090834
	concrete	4478	95797	31	379	0	100685
	foliage	7012	856	469654	0	186	477708
	sky	22	1	7709	1868687	0	1876419
	water	1162	0	1093	0	16562	18817
	Total	5816346	472839	702014	2527518	45746	9564463

Predicted labels					
brick	concrete	foliage	sky	water	Total
5783664	366384	232696	672529	35561	7090834
4192	95124	1257	112	0	100685
4703	367	471925	48	665	477708
121	0	7691	1868607	0	1876419
1382	0	2343	0	15092	18817
5794062	461875	715912	2541296	51318	9564463

Confusion matrices containing the five classes that are worked with. Left: RGB, right: RGBD

		Class			
		asphalt	brick	foliage	sky
Kernel size	0	0.2625	0.3699	0.0940	0.2735
	50	0.1481	0.4273	0.1086	0.3160
	100	0.0625	0.4703	0.1195	0.3477
	150	0.0225	0.4903	0.1246	0.3626
	200	0.0081	0.4976	0.1265	0.3679
	250	0.0030	0.5001	0.1271	0.3698
	300	0.0008	0.5012	0.1274	0.3706

(a) Reduction through erosion. With a kernel size of 300, this experiment closely resembles the class imbalance as present in our material dataset.

		Class			
		asphalt	brick	foliage	sky
No. of consecutive drops	1	0.0362	0.4834	0.1229	0.3575
	2	0.0149	0.4941	0.1256	0.3654
	4	0.0073	0.4979	0.1266	0.3682
	8	0.0038	0.4997	0.1270	0.3695
	16	0.0022	0.5004	0.1272	0.3701
	32	0.0011	0.5011	0.1273	0.3705
	64	0.0008	0.5012	0.1274	0.3706

(b) Reduction through erosion with kernel size 128 and dropping asphalt segmentations. When ignoring 64 asphalt segmentations after keeping one, this experiment closely resembles the class imbalance as present in our material dataset.

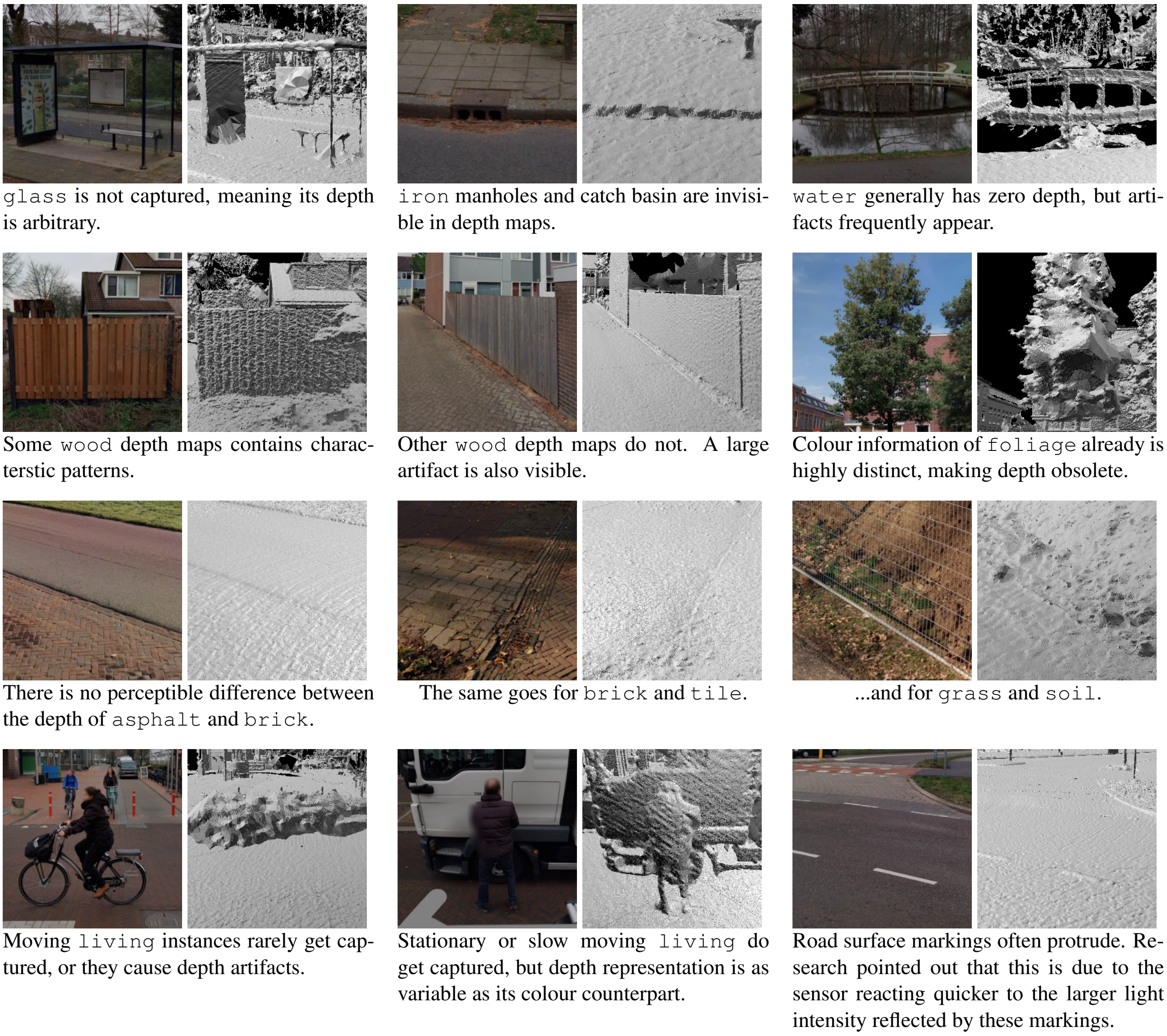
Table 17: Class size ratios among the four classes with decreasing size of asphalt. This decreasing of class size is done in two ways, for two experiments.

inherently noisy, and for some material classes often useless. We compiled some depth visualizations that cement these point in Figure 49.

4.4.4 SUBQUESTION III, concerning the estimation of performance curves

We investigated both class imbalance and prediction difference between RGB and RGBD input. What remains is to assess to what extent our lack of data is responsible for the disappointing performance. Our dataset is deficient in four respects: number of images, image coverage, number of segmentations, and segmentation size. We start with examining the correlation between segmentation area size and class performance by selecting a few large classes: asphalt; brick; foliage; and sky, and train those in a couple of successive experiments, each time shrinking all of asphalt's segmentations further using erosion. We use cross entropy loss, and we completely ignore the other classes, i.e. we do not make use of a background class, and therefore have four-dimensional output. The continual reduction of asphalt's segmentation size changes the relative ground truth distribution among the four classes. The ratios for each kernel size we use to erode asphalt are in Table 17a. It is only with a kernel size of 300, 34% of an image, with which the degree of class imbalance in our dataset is simulated. At that kernel size asphalt shares 0.08% of available ground truth, while in plastic occupies 0.07% of our training set (plastic and gravel furthermore occupy 0.03% and 0.01%, respectively, of our test set).

Plots of both the IoU of asphalt and mIoU for different kernel sizes, and both for weighted and unweighted cross entropy loss, are in Figure 50. Both IoU curves start with a slight increase, suggesting that the edges of asphalt segmentations are most difficult to predict. A possible explanation is that dirt and other such natural waste often accumulate in the grooves between road and curb, as seen in Figure 51. We repeat that nearly all segmentations for roads are derived from the instance dataset, and therefore completely encapsulates these roads. There is also frequently a tiled lining at the road edges. These may confuse the network into misclassifying these pixels. On top of that, our network is expected to be bad at predicting material edges and boundaries, as those are not annotated in our dataset. Surprisingly, the curves for asphalt remain relatively stagnant up until a kernel size of 250. It is only the last reduction which considerably decreases their performance. This fact leads us to believe that small segmentation size is not as much of a detriment to performance as we thought. It is only after there is barely any asphalt left that the erosion suddenly impacts perfor-



glass is not captured, meaning its depth is arbitrary.

iron manholes and catch basin are invisible in depth maps.

water generally has zero depth, but artifacts frequently appear.

Some wood depth maps contains characterstic patterns.

Other wood depth maps do not. A large artifact is also visible.

Colour information of foliage already is highly distinct, making depth obsolete.

There is no perceptible difference between the depth of asphalt and brick.

The same goes for brick and tile.

...and for grass and soil.

Moving living instances rarely get captured, or they cause depth artifacts.

Stationary or slow moving living do get captured, but depth representation is as variable as its colour counterpart.

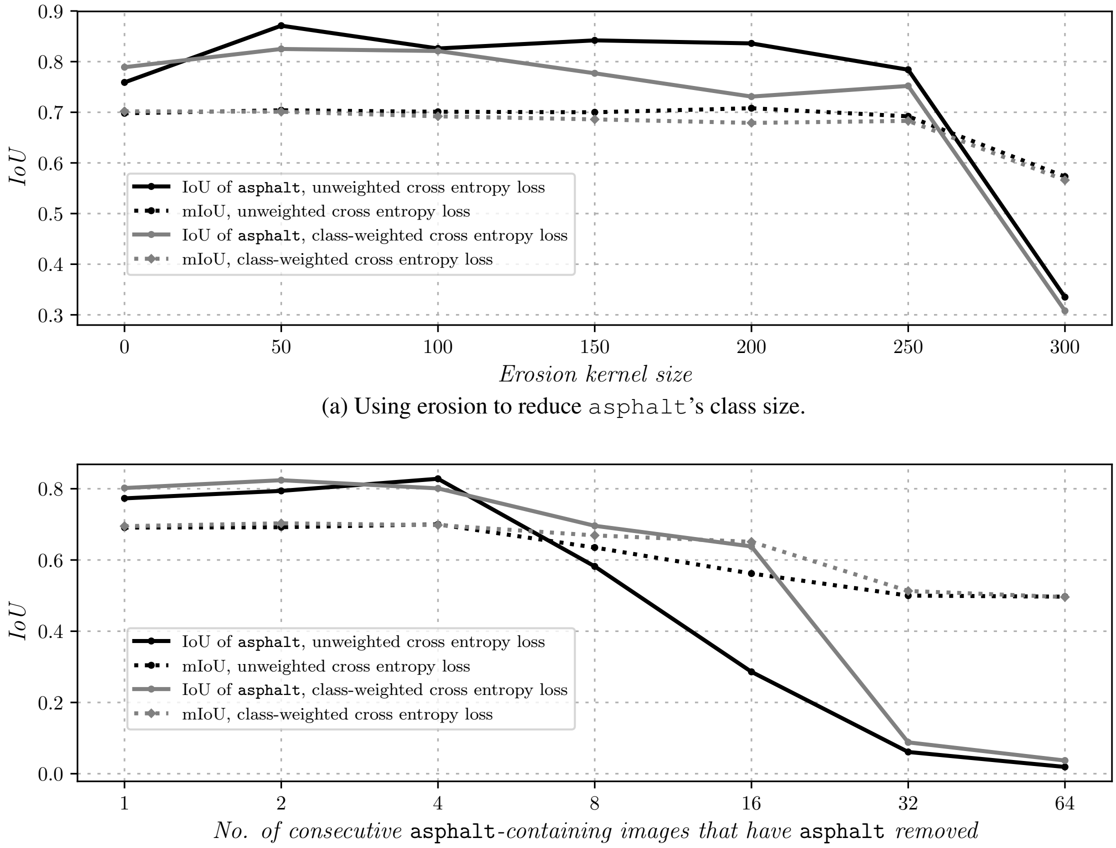
Road surface markings often protrude. Research pointed out that this is due to the sensor reacting quicker to the larger light intensity reflected by these markings.

Figure 49: A compilation of materials and examples where depth is either worthless or adds noise and confusion. These examples are not incidental, but appear frequently.

mance significantly. fabric, living, gravel, and iron's ratios are all between 0.01% and 0.02%, while the ratio of asphalt after being eroded with a 250x250 kernel is 0.03%. The mIoU of these classes can therefore be expected to be between 0.3 and 0.8. The actual results are much worse, however, with mIoUs near zero. At a loss for a better explanation, we ascribe their low performance to their difficulty caused by their diverging appearances.

We execute a similar series of experiments, this time adjusting for the number of asphalt segmentations rather than their surface area. The controlled variable in this experiment is a parameter which we refer to as the dropout rate. The higher the dropout rate, the more asphalt segmentations are removed. How this variable affects the number of segmentations is shows in Figure 40. The relative class sizes for each dropout rate are in Table 17b, and the resulting curves are plotted in Figure 50b. It is clear that this method sees mIoU decrease sooner, i.e. for less severe class imbalances. We note that the degree of class imbalance between the two methods differs greatly at the first iterations of class size reduction, but this difference decreases exponentially as the class sizes are reduced more. After a dropout rate of 8, the ratios share the same order of magnitude, and we deem the class imbalances comparable in severity. It is from that point that using dropout seems to affect performance, and more than using only erosion does. Using a class-weighted loss has the mIoU decrease only after a dropout rate of 16, however.

From these graphs we can infer that a large number of segmentations is more important than the size of the segmentations.



(a) Using erosion to reduce `asphalt`'s class size.

(b) Using dropout to reduce `asphalt`'s class size.

Figure 50: Plots of the IoU of `asphalt` and mIoU, for different kernel sizes and both for weighted and unweighted cross entropy loss. This decreasing of class size is done in two ways, using erosion to shrink segmentations, and by dropping whole segmentations.

This makes intuitive sense, as a wider range of segmentations is likely to introduce the network to a larger variety of material appearances. In contrast, having a same set of segmentations increase in size is less informative for the network, since the added pixels are probably similar to the ones that were already there. We also note that `asphalt` is rather uniform of appearance, whereas the classes we found to be problematic exists in a much wider variety of colours and shapes.

Lastly, we employ our other two datasets to estimate the extent to which lack of training images impacts overall performance. We do this by adjusting for the number of training images while plotting performance. Initial experiments with the instance-as-semantic dataset showed alarmingly low performance, which was caused by the severe undersampling of some classes. Five classes had no ground truth whatsoever. We therefore merged all classes with less than 83,242 annotated pixels (the size of the smallest material class with *some* performance) together, which left us with only 43 classes, given in Appendix B. Figure 52 makes clear that the material dataset outperforms the other datasets, undoubtedly caused by the flaws of the instance-as-material and instance-as-semantic listed in Table 3. The instance-as-material likely deals with too unpredictable a set of classes, and the instance-as-semantic dataset likely has too many classes to warrant reasonable performance for each, even after we merged the smallest ones together. We also observe that performance for the instance-as-material and instance-as-semantic datasets is saturated; adding more images to the training set will barely entail a performance improvement. This is not true for the material dataset, whose curve is still sloping upwards, though decreasingly so. We regressed this curve, and estimate that performance will saturate around a thousand training images with an mIoU around 0.7.

We answer SUBQUESTION III as follows: in terms of the number of training images, we expect that 400 additional training images, with a ground truth class distribution equal to that of the existing 600 images, will increase material segmentation performance by a reasonable margin. The biggest improvement in performance is anticipated to be attained

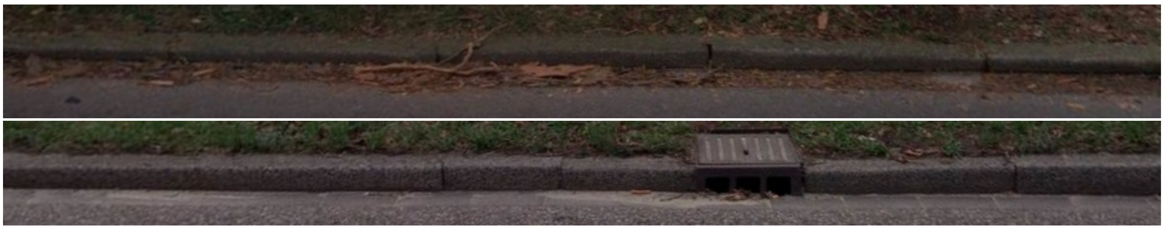


Figure 51: Two images of the sides of roads. Often, there is a significant amount of natural waste, or the paving is of a different material, like `stone` or `brick`. Water could also accumulate there, though CycloMedia usually drives in good weather. The fact that these road edges can deviate from the usual appearance of a road, it is likely that the network has most trouble with segmenting such edges.

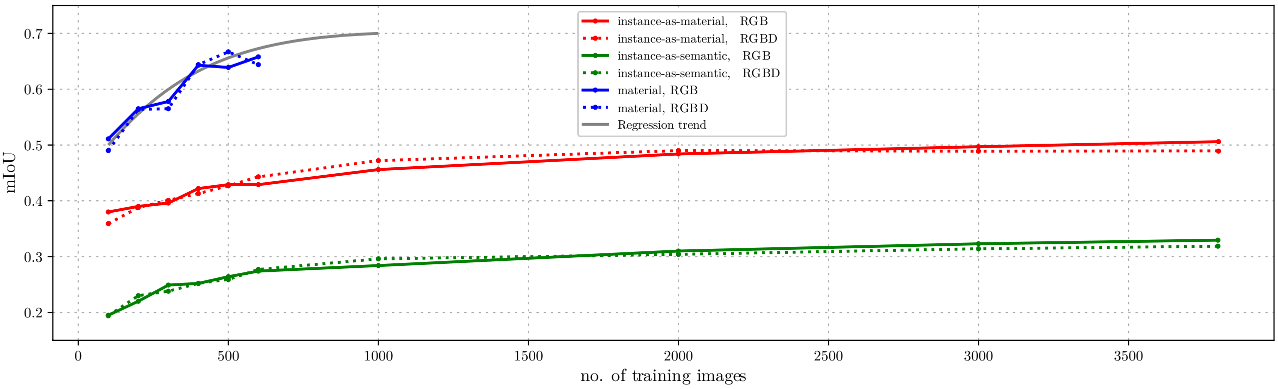


Figure 52: Performance curves of all three datasets trained with both unweighted as well as class-weighted cross entropy loss.

by additional annotations for the smallest classes, however. We therefore recommend focusing on the smaller classes when annotating additional images. Table 17b and Figure 50b convey that doubling a class's proportion of ground truth by adding segmentations is anticipated to significantly increase mIoU. In turn, ameliorating the performance of small classes will greatly benefit overall performance. Our recommendation would be to annotate at least 400 images in the same manner as the first 600, and then to focus the annotation effort on small classes.

5 Conclusions

Our research tackled the problem of material segmentation; semantic segmentation with respect to materials. To this end, we chose Semantic FPN, due to its reportedly good performance, especially in regard to the conceptual simplicity of its architecture [48]. It performs slightly worse than state-of-the-art semantic segmentation networks, even though those mostly use complicated techniques for improving performance, e.g. CRFs and atrous convolutions. What's more, Semantic FPN is part of Panoptic FPN, which additionally performs instance segmentation simultaneous to semantic segmentation, and the authors observed an increase in the quality of both outputs when performed together. This bodes well for any potential joint material segmentation and street furniture detection which may be performed by CycloMedia.

Our dataset has a severe class imbalance. Class imbalance in a dataset makes any network working with it biased towards the large classes; since the prior distribution gets shifted in favour of the large classes, it will teach itself to predict the larger classes more often, because such predictions have a higher a priori chance of being correct. The applicability of conclusions regarding a class are proportional to its performance; conclusions on classes with low and volatile performance are seldom valid or serviceable. SUBQUESTION I asked whether the impact on material segmentation performance caused by this class imbalance was salvageable with different training procedures. We tried a number of promising loss functions, but found none to improve the situation; the best loss function was class-weighted cross entropy loss, but only by a slight margin. Our conjecture is that the class imbalance is too extreme to be solved without creating additional ground truth. We proposed to handle the segmentation of small classes using an instance segmentator like RetinaNet with focal loss, and then to then mix the results with a material segmentation of the large classes, but we did not try this method.

We instead merged the five smallest and worst performing classes together into one `background` class, and performed our next experiments with this setup.

SUBQUESTION II treated the difference in material segmentation when working with and without additional depth input. Our experiments carried out for SUBQUESTION I already showed that working with per-pixel depth values did not improve performance, even though those do encode the material characterstics of some classes. We trained and tested exclusively on depth maps, and the outcomes hinted that depth does have distinguishing characterstics, though not as much as colour information. Subsequent experiments reinforced our conclusion that colour information already saturates performance, making depth obsolete.

Lastly, we looked into our data scarcity, and asked to what extent additional training data is expected to benefit performance with SUBQUESTION III. We found the number of a class's segmentations to be correlated with its performance, more so than the size of its segmentations. Accumulating plenty of segmentations is therefore essential for a class to perform satisfactory, even if those additional segmentation are comparatively small. We advocate to focus on small classes for future annotation efforts. This is not to say that more training images with ground truth distribution similar to the existing images is not worthwhile; we anticipate that an extra 400 training images will increase material segmentation performance up to 0.7 mIoU.

We starting this research with the following RESEARCH QUESTION: "What material segmentation performance are we able to obtain on outdoor imagery?" As an answer, we are able to achieve a performance of 0.67 with class-weighted cross entropy loss, but we only achieved this after merging the smallest, non-performing classes into a `background` class. With the material dataset as it currently stands, we suspect a higher performance is viable by using instance segmentation for small classes. Furthermore, we conclude that an increase in performance can be gained by either annotating 400 additional training images, or by only focusing new annotations on the small classes. A doubling of the ground truth of a small class can already result in at least a tripling of its performance. In short, a satisfying material segmentation performance is achievable, though likely not with our current dataset.

5.1 Future research

While we have shown an additional depth channel to have little to no effect on segmentation performance, we note that some LiDAR systems return reflectivity values in addition to the depth. This reflectivity is a ratio of incoming (returned) light to outgoing (sent) light, and is therefore dependent on the reflectivity – or inversely, the roughness – of a surface. This is obviously a very revealing characterstic of a material, and we therefore expect that an reflectivity map, added with colour and/or depth information into an RGBI or RGBDI representation, will benefit material segmentation performance. The LiDAR used by CycloMedia – Velodyne's HDL-32E – "measures reflectivity of an object independent of laser power and distances involved. [...] For each laser measurement, a reflectivity byte is returned in addition to distance. Reflectivity byte values are segmented into two ranges, allowing software to distinguish diffuse reflectors (e.g. tree trunks, clothing) in the low range from retroreflectors (e.g. road signs, license plates) in the high range." [91] This makes it a highly promising direction for research.

Naturally, a larger dataset entails a higher performance. While we concluded that the typical size of a class's segmentations have little effect on its performance, we expect that an actual semantic segmentation dataset – with all pixels annotated – will be beneficial. In this case, all edges are annotated, and therefore the network will be better at dividing material boundaries. Such boundaries are already apparent with colour information, but they still can be blurry or ambiguous, e.g. around trees, or due to interpolation or aliasing. Depth does not have this issue (when downscaled using nearest neighbour), and always shows "hard", or discrete, boundaries. A full semantic dataset will teach the network such boundaries, which is anticipated to improve segmentation boundaries.

`tile` also includes roof tiles, as we initially felt these did not occur often enough in our images to warrant a separate class, and that such a class would aggravate class imbalance. We expected the network to have little trouble in recognizing two types of manifestation of a material. However, it would have been better to create a separate `rooftile` class, since the materials used in roof tiles (e.g. ceramic, clay, concrete, or even plastic) are different from those in (stone) sidewalk tiling. Though not as visually present as one might think, there is still plenty of roof tiling visible in CycloMedia imagery, and amassing a reasonably sized `rooftile` class should be feasible.

Though we did not do so, it would in hindsight have been a good idea to designate pixels containing distant enough scenery as `other`, since those pixels may contain materials from multiple different classes, and since `other` is the only logical prediction the network could make for these regions. We encourage future research to actively annotate distant pixels as `other` or `background`.

A potentially informative way of evaluating material segmentation performance which we did not carry out is to analyze it per instance segmentation class. In other words, we isolate one of the classes from the instance segmentation dataset, and measure material segmentation performance only within the segmentations of the selected instance class. In this way, the segmentation quality between classes can be compared, and objects of which the materials are easily segmented can be distinguished from the hard-to-material-segment objects. This could provide valuable insights into material segmentation bottlenecks.

Our set of classes could be considered rather fine-grained. Instead of annotating one larger `metal` class, we distinguish between different types of metal, namely `aluminium` and `steel`, and instead of only `stone`, we gave `brick` and `tile` their own classes. Alternatively, one could create a hierarchy of classes. In this way, if the network is bad at distinguishing different metals, but better at detecting metal in general, then one could roll those prediction into one `metal` class and be assured of sufficient performance.

In our creation of the instance-as-material dataset, we mapped each instance class to one material class, resulting on many erroneous labelings. Another method would be a one-t-many instance-to-material translation. A car often consists of both `steel` and `glass`, and one could translate the `object | vehicle | car` class to both `steel` and `glass`, in which case those pixels get a two-hot encoding (rather than a one-hot encoding). During evaluation and loss taking, high certainties of either class are rewarded.

We only supplied our depth information to the network as a single channel of float values. Alternatively, the depth map can be transformed into a normal map, which encodes the direction of the surface normal for each pixel in another set of three RGB channels. The network then receives RGBRGB input, of which the latter three channels contain the depth information. Such an encoding essentially preprocesses depth into more meaningful information, as the orientation of a surface might hint at its material type. The distance information is lost however, but we have shown depth to have no effect anyways.

Lastly, we wish to discuss the potential for the use of material segmentations in object detection or instance segmentation. Since the type of an object and the material of which it consists are often, sometimes highly, correlated, it is expected that object detectors and instance segmentators will perform better with material segmentations. Knowing, for example, that an object is made of wood, will drastically change its posterior probability distribution, since the range of possible and/or probable object candidates is significantly reduced. This also holds for the opposite direction; knowing an object will, in most cases, reduce the possible set of materials that it encapsulates. Trees are never made of wood, for example.[*] This synergistic effect between material segmentation and instance segmentation is a promising research direction. Possibly the best way to exploit it would be to use Panoptic FPN, which outputs both instance and material segmentations and is trained with a unifying loss function [48]. Kirillov et al. found that the two tasks reinforce each other when trained simultaneously, and it is reasonable to expect a similar increase for material segmentation.

Acronyms

AUC Area-under-(the)-curve. 80

BRDF Bidirectional Reflectance Distribution Function. 35

CNN Convolutional neural network. 6, 16–19, 22, 25, 27, 30, 33, 34, 36, 42, 44, 67

COCO Common Objects in Context. 24, 25, 27, 30, 33

CRF Conditional random field. 34, 63

DAG Directed acyclic graph. 10, 70

DCR Digital cyclorama recorder. 68

[*] Any art installation that defies this rule is likely not annotated as `tree`.

DSC Dice score coefficient. 26

FCN fully convolutional network. 25, 27, 29, 31, 34, 35, 37

FPN Feature pyramid network. 22, 29–34, 36–38, 49

GDL Generalized dice loss. 26

GDS Generalized dice score. 26

GIS Geographic information system. 5, 68

GPS Global positioning system. 5

GPU Graphics processing unit. 19, 49

HOG Histogram of oriented gradients. 22

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 18, 20

IMU Inertial measurement unit. 5

IoU Intersection-over-union. 29, 32, 39

LiDAR Light Detection and Ranging. 6, 64, 69

MINC Material in Context Database. 34, 35

mIoU Mean intersection-over-union. 26, 38

MIT Massachusetts Institute of Technology. 34

NLP Natural language processing. 25

RGB Red-green-blue. 5, 23, 34–36, 38, 39, 42, 68

RGBD Red-green-blue-depth. 5, 39

RoI Region-of-interest. 27–31

RPN Region porposal network. 28–33

SGD Stochastic gradient descent. 16, 67, 68, 71

SIFT Scale-invariant feature transform. 22

SLAM Simultaneous localization and mapping. 5

SSD Single Shot Detector. 31

SVM Support-vector machine. 27

ToF Time-of-flight. 35

YOLO You Only Look Once. 31

Glossary

activation The value of a node as a reaction to the activations of preceding nodes and indirectly to the input data, reflecting the degree to which the node responds to the input, and therefore the extent to which the feature that the node represents is present in the input. 9–14, 16–27, 32, 67, 68, 71, 72

activation function A function used in calculating a neuron's activation, or output. It takes as input the summed activations of previous neurons multiplied by weights, and non-linearly scales this to a fixed range for output activations. The non-linearity is needed for the network to solve a much larger number of non-trivial problems without using an intractable number of nodes. 11–15, 17, 23

affine Allowing for or preserving parallel relationships. 27

architecture A specific instance of neural network, with a predetermined structure, depth, and number of neurons per layer. The term "network" is often used more loosely to mean "architecture". 11

autodidacticism The property of being self-taught. Within the context of neural networks, this refers to metalearning; the network teaches itself how to learn. 21

backbone A core CNN architecture on top of which more complicated CNNs can be built. A backbone is at least able to output smaller feature representation of input images. 22, 25, 28–31, 33, 34

backpropagation Shorter synonym of backward propagation. 12, 23, 27, 30, 67, 72

backward propagation The calculation of the gradient of the loss with respect to each parameter in the network, which determines the adjustments for those parameters. The gradient of the loss with respect to each parameter is calculated from the deepest layer to the shallowest, hence the name. Called "backpropagation" or "backprop" for short. 12, 67, 69

batch A subset of the dataset into which training samples are grouped, and which are processed simultaneously when performing minibatch SGD. 16, 21, 23, 24, 29, 32, 69, 71

batch size The number of samples contained in each batch during SGD. 11, 24, 48, 69

bias A type of parameter associated with a neuron in a network. 11–15, 17, 18, 26, 29, 70

bounding box Rectangular outlines demarcating the spatial extent of objects, i.e. the collection of all pixels that make it up – in images as tightly as possible. 6, 9, 10, 27, 28, 31, 69, 70

channel A slice of a three-dimensional tensor representing an image. Each channel is two-dimensional, with each value corresponding to a pixel of the image. Such a tensor often consists of three channels: one for the red, one for the greed, and one for the blue colour values, though many other colour representations are also possible. In this work, we additionally concern ourselves with a fourth channel: the depth map. 5, 16, 17, 21, 23–25, 38, 42, 45, 68

class One of multiple (named) categories which can be predicted during multinomial classification and segmentation tasks. When referring to the chosen category of a specific predictions, we use "label". 9, 10, 13, 14, 25–36, 38–46, 49, 52–54, 63, 64, 67, 69

class imbalance A significant difference between the number of samples of at least two classes in a dataset. 25, 38, 45, 63

cloud cover The fraction of sky covered by clouds. 45

computer vision An interdisciplinary field, though overlapping mostly with computer science and artificial intelligence, that researches the automatic analysis and interpretation of digital imagery. 6, 9, 10, 16, 18, 22

convolution A transition between two layers in a CNN, in which the second layer has different dimensions, and is calculating by moving a filter, or kernel, over the first layer and computing the summed products between activations in the first layer and weights in the filter. 16–21, 25, 26, 30, 31, 34, 37, 67, 68, 71

convolutional neural network A neural network which contains convolutional layers, specialized in processing images and videos. Abbreviated as CNN. 6, 9, 16–19, 22, 65, 68, 71

cyclorama Historically, a large cylindrical image or painting, designed to give the spectator the impression to be standing in the middle of the depicted scenery. Within CycloMedia, this term has been repurposed to refer to the 360°panoramic images captured by their car-mounted DCR-system, and even colloquially extends to any street-level image. 5–7, 35, 68

deep learning A subfield of machine learning, which uses (artificial) neural networks to achieve the inference of characterstics, or features, from training data. 6, 9, 18, 23, 25, 26, 39, 68

depth map An "image", or rather a channel, which instead of an RGB colour value contains a single depth value; generally a decimal value. Can also refer to the image file into which depth information is encoded and stored. 5, 6, 8, 35, 41, 42, 50, 51, 67, 70

digital cyclorama recorder Car-mounted capture devices designed and operated by Dutch company CycloMedia, which captures the public space at five meter intervals. The panoramic image camera captures 360°, 250 megapixel cycloramas, and a LiDAR sensor sends light signals in order to record depth in the form of points in three-dimensional space. 65

epoch One iteration over all training samples. 48, 49, 68, 71

equisized The property of being of the same size. 22, 29, 33, 34, 37

feature An attribute of data. Traditional machine learning techniques uses manually designed features, while deep learning has models infer these automatically. The range of possible data points spanned by the collection of its features is known as the feature space of a certain type of data. 9–12, 16, 18–23, 25–31, 33, 34, 36, 37, 66–68, 70, 71

featurized image pyramid The ideal feature image pyramid, which, instead of a regular colour image at different scales, contains (condensed) feature representations of the differently sized images. Useful for predicting objects of varying size upon. 22, 34, 36

filter A small, three-dimensional matrix of convolution weights, which is moved across a layer in order to produce activations for the subsequent layer. Used synonymously with kernel. 17, 19, 32, 67, 69

forward propagation The processing of input data by a neural network, which results in output data, or predictions. The activations of the nodes are calculating from the shallowest input layer to the deepest layer, hence the name. 11, 13, 16

fully connected A layer's property of having each of its nodes be connected to every node in the preceding and/or succeeding layer. 18–21, 23, 25, 27, 28, 34, 68

fully convolutional network A convolutional neural network without any fully connected layers, instead outputting a three-dimensional feature representation of the input image. 25, 34, 35, 66

geographic information system An umbrella term encompassing (computer) systems designed to capture, store, manipulate, analyze, manage, and present spatial or geographic data. Abbreviated as GIS. 66

gradient A generalization of the derivative for multivariate functions. It is an array, vector or matrix, of the same shape as the function input, containing the partial derivatives of the function with respect to each variable. Analogously to how a derivative indicates the slope at a certain point on a curve, a gradient point in the direction of steepest ascent at any point (i.e. specific set of inputs), and essentially tells how the inputs should be changed in order to increase the function output. 12, 14–16, 20, 21, 23, 68–72

gradient descent An algorithm for optimizing a loss function. In an iterative manner, and until the network has converged, the gradient of the loss with respect to the network parameters are determined at the network's current point on the loss landscape, and the parameters are adjusted based on the magnitude and (opposite) direction of that gradient, which makes the network take a step in the direction of steepest descent. Gradient descent only takes a step after calculation of and based on the gradient of a whole epoch. While the steps taken are better approximations of the gradient and the optimal direction, gradient descent converges slow, and has therefore been superseded by (minibatch) SGD. 12–16, 71

gradient flow An intuition for how the gradients of the loss with respect to all network parameters are calculated. Since the gradient of the parameter of a layer are dependent on the gradients of the succeeding layer, and all gradients are therefore being calculated from front to back during backward propagation, this calculation of the gradient calculation can be regarded as a flow going back through the network. 12, 14, 23, 30, 33

ground truth Manually annotated truth or true values belonging. Ground truth can be regarded as the answer that a network is supposed or desired to give when supplied with the associated input data, and is therefore also sometimes called the "golden standard" of the data; it is the ideal prediction. 7, 12–15, 25–27, 29, 31, 32, 36–40, 42, 45, 46, 48, 50, 52, 54, 69, 71

hyperparameter A metaparameter that is not learned or otherwise part of the network, but which is set manually before training and defines an aspect of the network or the training procedure. Examples are learning rate, network depth, batch size, etc. 11, 15, 17, 19, 20, 31

image classification The computer vision objective of classifying an image (with one label) with respect to the depicted subject. 9, 10, 38, 80

image pyramid A series of the same image at regularly spaced scales. A specific type of pyramid. 22, 68

instance segmentation Either a specific collection of detected objects in an image and their pixel-wise segmentation masks – demarcating their exact spatial extent, or the general computer vision objective of generating these instance segmentations. Similar to object detection, except segmentation masks are used to denote an object's spatial extent rather than a mere bounding box. 5, 6, 9, 10, 27, 29, 32–35, 37, 38, 42–44, 70, 80

kernel Used synonymously with filter. 17–22, 25, 26, 37, 67, 68, 71

label Often used synonymously with "class", though we use "label" to refer to the category of a specific prediction. 9, 26, 34, 42–44, 67

layer An organizational unit into which neurons of a neural network are grouped. Each layer can be considered as an exhaustive feature space of the input data, and the collection of activations in a layer as a complete representation of particular input. 10–30, 32–37, 67–69, 71, 72

learning rate A multiplier between 0 and 1 with which the gradient of the weights is multiplied before it is added to the weights. This determines the pace at which a network is learning. Too large a learning rate might lead to too aggressive steps across the loss landscape, while too slow a learning rate might make the network take an impractically long time to converge. 11, 15, 48, 69

LiDAR A three-dimensional capturing system which sends light signals in many outgoing directions and measures the time between their emission and their return. Using the speed of light, the distance between the sensor and the point of collision can be calculated. This results in a three-dimensional point for every received signal. It stands for Light Detection and Ranging. 5, 8, 35, 42, 68, 70

loss A measure of the difference between a networks output, or prediction, and the prediction it is supposed and desired to output according to the ground truth of the training data. There are multiple loss functions, each of which calculate loss in a different way and thereby prioritize different aspects of performance. 12–16, 25–33, 48, 49, 54, 69, 71

loss function A function that calculates loss given a networks output and the ground truth corresponding to the network input. Also sometimes called a cost function or an objective function, though this would not be strictly correct. A loss function calculates the difference of a data point, or batch of data points, while a cost function is more general, and can sum losses over a dataset and potentially adds a regularization penalty. An objective function is even more general, and refers to any function that is optimized during training. Cost and objective are not used in this work however, and we always refer to any difference as loss. There are multiple loss functions, each of which calculate loss in a different way and thereby prioritize different aspects of performance. 13–16, 24–27, 29, 31, 32, 38, 48, 68, 69

loss landscape The multivariate function with all parameters of a neural network as domain and with a loss, as calculated according to a loss function and depending on the input data, as output. In non-realistic examples featuring a network with only two weights (for visualization purposes), such a function tends to resemble a hilly landscape, hence the name. Naturally, the actual landscape depends on the selection of input. 15, 16, 68, 69

machine learning A subfield of artificial intelligence, studying the (meta-)algorithms and statistical models that make computers infer patterns and characterstics – needed to perform a certain task – automatically from training data, instead of having these features explicitly programmed. Machine learning allows for efficiently carrying out complicated tasks, like image recognition and natural language processing, and increased scalability compared to traditional algorithms, though it is notorious for its trend of requiring large amounts of training data. 6, 9, 10, 16, 18, 24, 49, 68, 71

mask A subset, or selection, of specific pixels of an image (or feature representation), usually encoded as a binary map. 9, 10, 27, 29–31, 43, 44, 71

material segmentation Either a specific per-pixel image classification with respect to a predefined set of labels relating to materials, or the general computer vision objective of generating these material segmentations. A sub-objective of semantic segmentation, with the extra constraint that labels have to refer to materials or something resembling material properties. 5–9, 35, 36, 38–42, 46

momentum A different method of gradient calculation, which takes the moving average of past gradients into account when updating parameters. This reduces the chance of the learning process stagnating whenever the gradient is (close to) zero. 15, 16, 48

neural network Complicated functions resembling DAGs and involving many operations and parameters, the values of which have to be inferred from training data, that are able to perform complicated tasks by transforming input values into output values. 9–16, 18, 25, 67–72

neuron Can be used synonymously with "node", although we use this term when discussing the biological basis of neural networks. 9, 10, 16, 18, 23, 67, 69, 70, 72

node A single unit of compute which serves as the building block of a neural network. It received an input, which is the activated sum of products of node activations in the previous layer with the weights associated with the connections to the node at hand. It outputs this value to all neurons in the following layer. Also referred to as "neuron", though we reserve that for the biological context. 9–14, 28, 67, 68, 70–72

normalization The operation of transforming (translating and scaling) a set of values in such a way that the set has zero mean and unit variance. 14, 23, 24, 37

object detection The computer vision objective of detecting one or more subjects in an image and demarcating them with bounding boxes. Similar to instance segmentation, but more simple, since only bounding boxes are used instead of the more complex binary segmentation masks. 9, 10, 18, 24, 27, 29–31, 34, 37, 69, 70, 80

object localization The computer vision objective of labeling the image with respect to the depicted subject and demarcating this subject with a bounding box (or perhaps even a segmentation mask). Similar to object detection, except that only one subject of interest is being detected for. 9, 10

objective A specific form of analysis to be performed on imagery data. It defines a format of prediction that should be inferred from images. 9, 24, 71

overfitting The overtailoring of a network to its training data. In overfitted network has adapted itself too much to the data with which it was trained, and thereby has lost its ability to generalize to other data. Since testing or production data is likely to have different characterstics than training data, such a model often performs worse when needing to process such alien data. 20, 49, 71

panoptic segmentation The computer vision objective combining both instance and semantic segmentation, i.e. instance segmentations are returned and pixels are labeled. 9, 10, 25, 32, 35, 38

parameter One of many values that embody a neural network's learned representation of the data that is has seen during training. In most neural networks, these consist of weights and biases. 11–21, 23, 29, 32, 34, 67–72

point cloud A collections of three-dimensional points, collected using a LiDAR sensor and corresponding to surfaces in the public space. Using surface reconstruction algorithms, these point clouds can be transformed into a 3D mesh, and these meshes are used to generate depth maps. 5–8, 42

pooling A layer in a convolutional network which reduces the dimension of the activation map and which is not learned, i.e. the activation map is reduced using straightforward operation, like taking the maximum value within each 4x4 square of activations. 18–21, 27, 30, 72

pyramid A series of activation maps (or other multidimensional arrays) at regularly spaced scales. 36, 69

receptive field The subset of input neurons to a convolutional neural network which are connected to, and which therefore influence the activation of, a certain node. 18, 28

regularization Any measure used to prevent overfitting by forcing the machine learning model to stay simple and interpretable, as complex models are more susceptible to capturing noise, outliers and irregular data. The most common form of regularization is a penalty added to the loss function, based on the models complexity. 23, 24, 69, 71

sample A pair of input data and correct, ground truth output data, which are used to train and test a network with. Also called "examples", because it intuitively shows a network what to learn and look for. 11, 13–16, 23, 29–32, 35, 36, 68, 71

segmentation Either a subset, or selection, of specific pixels of an image (or feature representation), or a per-pixel, potentially multinomial classification of an image. The former is synonymous with a mask. 25–27, 29–31, 33–35, 37, 42, 43, 45, 49

semantic segmentation Either a specific per-pixel image classification with respect to a predefined set of labels relating to either things or stuff, or the general computer vision objective of generating these semantic segmentations. 9, 10, 24–26, 32–35, 37, 38, 43, 70, 80

softmax A function taking a range of arbitrary values and dividing each by their sum, in order to make them sum to 1. A softmax function is used at the output layer of neural networks, to transform the activations received by the output layer – which can have any value – to interpretable probabilities. 13, 27

stochastic gradient descent A variant of gradient descent, in which the gradient calculation and parameter update is done after each processed training sample, instead of after each epoch. When the updating is done after processing of a batch of samples instead of just one sample, then this is called minibatch stochastic gradient descent. Abbreviated as SGD. 66

stuff Uncountable material matter. 24, 25, 33, 35, 71

supervised learning A training procedure in which a network is being explicitly told what to learn. More concretely, besides the input training data to a model, it also receives associated ground truth with which it can be evaluated and adapted. This contrasts with unsupervised learning, where only input data is given. Common supervised models are (supervised) neural networks, support vector machines, and random forests. 9

task Synonymous with objective. 9, 25, 32, 33, 67

tensor A multidimensional array. Within the context of this work, this generally means a three-dimensional array containing the colour values and potentially the depth values of an image. 49, 67

thing A countable unit of material matter, or object. 24, 25, 33, 35, 71

training The process of feeding specially compiled training data to an (untrained) neural network, quantifying the difference between the network output and the desired output – called the loss, and tweaking the networks many parameters using the derivative of the loss with respect to these parameters, in order to make it perform better next time. This process is iterated until some condition defining satisfactory performance is met, in which case the network has converged. 9, 11–16, 20, 21, 23, 24, 27–29, 32–36, 38–41, 49, 50, 67–71

transpose convolution The inverse of a convolution. Synonymous with upsampling. 25, 26

unsupervised learning A training procedure in which a model is only given input data, and has to infer patterns and features on its own. Common unsupervised models are clustering models and autoencoders. 9, 71

upsampling The inverse of a convolution; a learned layer which increases instead of decreasing an activation map's resolution using learned kernels. 22, 25, 33, 71

upscaling The inverse of a pooling operation; a layer which increases instead of decreasing an activation map's resolution by remembering how the activation map was originally pooled. 36, 37

vanishing gradient The problematic phenomenon present in deep neural networks, where repeated application of the chain rule diminishes the gradient during backpropagation, until the weights at the shallowest nodes receive next to no gradient signal. 20, 21

weight A type of parameter associated with the connection between two neurons in a network, the value of which is learned and depends on the correlation of their activations . 11–18, 21, 23, 25, 26, 33, 38, 53, 54, 67, 68, 70, 72

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[3] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2189–2202, 2012.

[4] P. Arbeláez, J. Pont-Tuset, J. T. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 328–335, 2014.

[5] A. S. Baslamisli, T. T. Groenestege, P. Das, H.-A. Le, S. Karaoglu, and T. Gevers. Joint learning of intrinsic images and semantic segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 286–302, 2018.

[6] S. Bell, P. Upchurch, N. Snavely, and K. Bala. Opensurfaces: A richly annotated catalog of surface appearance. *ACM Transactions on Graphics (TOG)*, 32(4):111, 2013.

[7] S. Bell, P. Upchurch, N. Snavely, and K. Bala. Material recognition in the wild with the materials in context database. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3479–3487, 2015.

[8] G. Bradski. The opencv library. *Dr Dobb's J. Software Tools*, 25:120–125, 2000.

[9] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[10] J. S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

[11] J. S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in neural information processing systems*, pages 211–217, 1990.

[12] M. Buda, A. Maki, and M. A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, 2018.

[13] H. Caesar, J. Uijlings, and V. Ferrari. COCO-stuff: Thing and stuff classes in context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1209–1218, 2018.

[14] J. Carreira and C. Sminchisescu. CPCM: Automatic object segmentation using constrained parametric min-cuts. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (7):1312–1328, 2011.

[15] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[16] L. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[17] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *arXiv preprint arXiv:1802.02611*, 2018.

[18] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[19] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613, 2014.

[20] D. C. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 411–418. Springer, 2013.

[21] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The Cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[22] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[23] W. R. Crum, O. Camara, and D. L. Hill. Generalized overlap measures for evaluation and validation in medical image analysis. *IEEE transactions on medical imaging*, 25(11):1451–1461, 2006.

[24] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[25] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[26] D. de Geus, P. Meletis, and G. Dubbelman. Panoptic segmentation with a joint semantic and instance segmentation network. *arXiv preprint arXiv:1809.02110*, 2018.

[27] I. Endres and D. Hoiem. Category independent object proposals. In *European Conference on Computer Vision*, pages 575–588. Springer, 2010.

[28] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[29] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[30] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017.

[31] R. Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[32] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[33] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[34] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[35] S. Gould, R. Fulton, and D. Koller. Decomposing a scene into geometric and semantically consistent regions. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1–8. IEEE, 2009.

[36] J. Han and C. Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.

[37] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[38] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[39] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[40] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[41] D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[42] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[43] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016.

[44] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[45] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[46] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[47] Y. Jaafra, J. L. Laurent, A. Deruyver, and M. S. Naceur. A review of meta-reinforcement learning for deep neural networks architecture search. *arXiv preprint arXiv:1812.07995*, 2018.

[48] A. Kirillov, R. Girshick, K. He, and P. Dollár. Panoptic feature pyramid networks. *arXiv preprint arXiv:1901.02446*, 2019.

[49] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár. Panoptic segmentation. *arXiv preprint arXiv:1801.00868*, 2018.

[50] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[52] S. Lawrence, I. Burns, A. Back, A. C. Tsoi, and C. L. Giles. Neural network classification and prior class probabilities. In *Neural networks: tricks of the trade*, pages 299–313. Springer, 1998.

[53] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[54] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[55] Y. A. LeCun, L. Bottou, G. B. Orr, and K. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[56] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *CVPR*, volume 1, page 4, 2017.

[57] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[58] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[59] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[60] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[61] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[62] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[63] S. Lowel and W. Singer. Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity. *Science*, 255(5041):209–212, 1992.

[64] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[65] C. Manning, P. Raghavan, and H. Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

[66] F. Milletari, N. Navab, and S. Ahmadi. V-Net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571. IEEE, 2016.

[67] G. Neuhold, T. Ollmann, S. Rota Bulo, and P. Kontschieder. The Mapillary Vistas dataset for semantic understanding of street scenes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4990–4999, 2017.

[68] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[69] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[70] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[71] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[72] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[73] C. Robert. Machine learning, a probabilistic perspective, 2014.

[74] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[75] S. S. M. Salehi, D. Erdogmus, and A. Gholipour. Tversky loss function for image segmentation using 3d fully convolutional deep networks. In *International Workshop on Machine Learning in Medical Imaging*, pages 379–387. Springer, 2017.

[76] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[77] G. Schwartz and K. Nishino. Material recognition from local appearance in global context. *arXiv preprint arXiv:1611.09394*, 2016.

[78] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[79] C. Shen, H. R. Roth, H. Oda, M. Oda, Y. Hayashi, K. Misawa, and K. Mori. On the influence of dice loss function in multi-class organ segmentation of abdominal CT using 3D fully convolutional networks. *arXiv preprint arXiv:1801.05912*, 2018.

[80] J. Shi, Y. Dong, H. Su, and S. X. Yu. Learning non-lambertian object intrinsics across ShapeNet categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1685–1694, 2017.

[81] J. Šíma. Training a single sigmoidal neuron is hard. *Neural computation*, 14(11):2709–2728, 2002.

[82] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[83] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 240–248. Springer, 2017.

[84] S. Sun, J. Pang, J. Shi, S. Yi, and W. Ouyang. Fishnet: A versatile backbone for image, region, and pixel level prediction. In *Advances in Neural Information Processing Systems*, pages 762–772, 2018.

[85] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[86] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[87] Y. Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[88] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[89] V. Van Asch. Macro-and micro-averaged evaluation measures [[basic draft]]. *Belgium: CLiPS*, pages 1–27, 2013.

[90] B. Van Someren. Neural multi-view segmentation-aggregation for joint Lidar and image object detection. 2017.

[91] Velodyne LiDAR, I. HDL-32E User's Manual, 2012.

[92] willem. F1/Dice-Score vs IoU. Cross Validated. URL:https://stats.stackexchange.com/q/276144 (version: 2017-11-13).

[93] World Wide Web Consortium and others. Portable network graphics (PNG) specification. `http://www.w3.org/TR/PNG/`, 2003.

[94] Y. Wu and K. He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[95] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

[96] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[97] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[98] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

[99] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[100] Y. Zhang, K. N. Ngan, C. P. Huynh, and N. Habili. Learning deep spatial-spectral features for material segmentation in hyperspectral images. In *Digital Image Computing: Techniques and Applications (DICTA), 2017 International Conference on*, pages 1–7. IEEE, 2017.

[101] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1452–1464, 2018.

[102] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using Places database. In *Advances in neural information processing systems*, pages 487–495, 2014.

[103] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ADE20K dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, page 4. IEEE, 2017.

[104] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba. Semantic understanding of scenes through the ADE20K dataset. *International Journal of Computer Vision*, pages 1–20, 2016.

Appendix A Instance-to-material mapping

Instance class	Material class
animal | ground animal	living
construction | barrier | fence	steel
construction | barrier | guard rail	steel
construction | barrier | other barrier	stone
construction | barrier | separator	tile
construction | barrier | wall	brick
construction | flat | bike lane	brick
construction | flat | crosswalk plain	asphalt
construction | flat | curb	stone
construction | flat | curb cut	stone
construction | flat | parking	brick
construction | flat | pedestrian area	stone
construction | flat | rail track	asphalt
construction | flat | road	asphalt
construction | flat | road shoulder	asphalt
construction | flat | service lane	asphalt
construction | flat | sidewalk	tile
construction | flat | traffic island	stone
construction | structure | bridge	concrete
construction | structure | building	brick
construction | structure | garage	steel
construction | structure | pylon	steel
construction | structure | tunnel	concrete
human | person | group	living
human | person | individual	living
human | rider | bicyclist	living
human | rider | motorcyclist	living
human | rider | other rider	living
marking | continuous | dashed	asphalt
marking | continuous | solid line	asphalt
marking | discrete | crosswalk zebra	asphalt
marking | discrete | other marking	asphalt
marking | discrete | stop line	asphalt
marking | discrete | symbol	asphalt
marking | discrete | text / number	asphalt
nature | desert	soil
nature | grass	grass
nature | mountain	stone
nature | other terrain	soil
nature | sand	soil
nature | sky	sky
nature | vegetation | high | group	foliage
nature | vegetation | high | single	foliage
nature | vegetation | low | group	foliage
nature | vegetation | low | single	foliage
nature | water	water
object | object | banner	fabric
object | object | bench	wood
object | object | bike rack	steel
object | object | billboard	plastic
object | object | catch basin	iron
object | object | cctv-camera	plastic
object | object | fire hydrant	iron
object | object | junction box	steel
object | object | mailbox	steel
object | object | manhole	iron
object | object | parking meter	steel
object | object | phone booth	glass
object | object | pothole	asphalt
object | object | street light	plastic
object | object | traffic cone	plastic
object | object | trash can	plastic
object | object | wire group	sky
object | support | pole	steel
object | support | traffic sign frame	plastic
object | support | utility pole	steel
object | traffic-sign | back	aluminium
object | traffic-sign | front	aluminium
object | traffic light | back | 1 light	plastic
object | traffic light | back | 2 light	plastic
object | traffic light | back | 3 light	plastic
object | traffic light | front | 1 light	plastic
object | traffic light | front | 2 light	plastic
object | traffic light | front | 3 light	plastic
object | vehicle | bicycle | group	steel
object | vehicle | bicycle | single	steel
object | vehicle | boat | group	plastic
object | vehicle | boat | single	plastic
object | vehicle | bus | group	steel
object | vehicle | bus | single	steel
object | vehicle | car | group	steel
object | vehicle | car | single	steel
object | vehicle | caravan	plastic
object | vehicle | motorcycle | group	steel
object | vehicle | motorcycle | single	steel
object | vehicle | on rails	steel
object | vehicle | other vehicle | single	steel
object | vehicle | trailer	steel
object | vehicle | truck	steel
object | vehicle | wheeled slow	steel
void | car mount	other
void | dynamic	other
void | ego vehicle	other
void | ground	other
void | static	other
void | unlabeled	other

Table 18: Our choice of instance classes to material classes mapping that we used in constructing the instance-as-material dataset.

Table 18 outlines for each of the 96 instance classes in CycloMedia's instance segmentation dataset to which of the 21 material classes it was translated when we constructed our instance-as-material dataset. Some of our choices of translation are obvious some may raise questions. When we were in doubt as to which material class an instance class should be translated we inspected a few ground truth instances to get an impression of which material is visually most prominent and based our choice on that small sample of segmentations. A good example is that of `object | object | wire group` a class which segments (overhead) power lines. In the selection of segmentations which we inspected the power lines themselves occupied only a small portion of the segmentation pixels but encompassed much larger regions of sky. Hence it would be more apt to translate this class to `sky` as this would cause the least number of pixels to be mislabeled. Some instance class names were unclear to us. We did not know beforehand what a `construction | barrier | separator` or a `object | vehicle | other vehicle | single` would look like. Other classes like `construction | flat | road` can assume a number of material classes. Naturally such segmentations will often get mislabeled. In all these cases we resolved the material class by visual inspection as described above.

Appendix B Instance classes included in the instance-to-semantic dataset

Instance class	In instance-as-semantic?	Instance class	Material class
animal | ground animal	No	object | object | bike rack	No
construction | barrier | fence	Yes	object | object | billboard	Yes
construction | barrier | guard rail	Yes	object | object | catch basin	No
construction | barrier | other barrier	Yes	object | object | cctv-camera	No
construction | barrier | separator	Yes	object | object | fire hydrant	No
construction | barrier | wall	Yes	object | object | junction box	No
construction | flat | bike lane	Yes	object | object | mailbox	No
construction | flat | crosswalk plain	No	object | object | manhole	No
construction | flat | curb	Yes	object | object | parking meter	No
construction | flat | curb cut	Yes	object | object | phone booth	No
construction | flat | parking	Yes	object | object | pothole	No
construction | flat | pedestrian area	Yes	object | object | street light	No
construction | flat | rail track	No	object | object | traffic cone	No
construction | flat | road	Yes	object | object | trash can	Yes
construction | flat | road shoulder	No	object | object | wire group	No
construction | flat | service lane	Yes	object | support | pole	Yes
construction | flat | sidewalk	Yes	object | support | traffic sign frame	No
construction | flat | traffic island	Yes	object | support | utility pole	No
construction | structure | bridge	Yes	object | traffic-sign | back	No
construction | structure | building	Yes	object | traffic-sign | front	Yes
construction | structure | garage	No	object | traffic light | back | 1 light	No
construction | structure | pylon	No	object | traffic light | back | 2 light	No
construction | structure | tunnel	No	object | traffic light | back | 3 light	No
human | person | group	No	object | traffic light | front | 1 light	No
human | person | individual	No	object | traffic light | front | 2 light	No
human | rider | bicyclist	No	object | traffic light | front | 3 light	No
human | rider | motorcyclist	No	object | vehicle | bicycle | group	Yes
human | rider | other rider	No	object | vehicle | bicycle | single	No
marking | continuous | dashed	Yes	object | vehicle | boat | group	Yes
marking | continuous | solid line	Yes	object | vehicle | boat | single	No
marking | discrete | crosswalk zebra	No	object | vehicle | bus | group	No
marking | discrete | other marking	Yes	object | vehicle | bus | single	No
marking | discrete | stop line	No	object | vehicle | car | group	No
marking | discrete | symbol	No	object | vehicle | car | single	Yes
marking | discrete | text / number	No	object | vehicle | caravan	No
nature | desert	No	object | vehicle | motorcycle | group	No
nature | grass	Yes	object | vehicle | motorcycle | single	No
nature | mountain	No	object | vehicle | on rails	No
nature | other terrain	Yes	object | vehicle | other vehicle | single	No
nature | sand	Yes	object | vehicle | trailer	Yes
nature | sky	Yes	object | vehicle | truck	Yes
nature | vegetation | high | group	Yes	object | vehicle | wheeled slow	No
nature | vegetation | high | single	Yes	void | car mount	No
nature | vegetation | low | group	Yes	void | dynamic	No
nature | vegetation | low | single	Yes	void | ego vehicle	No
nature | water	Yes	void | ground	No
object | object | banner	No	void | static	Yes
object | object | bench	No	void | unlabeled	Yes

Table 19: Selection of classes which we trained with when using our instance-as-semantic dataset.

Table 19 outlines which of the 96 instance classes in CycloMedia's instance segmentation dataset are included in our instance-as-semantic dataset. All classes with less than 83,242 ground truth pixels were merged into a `background` class, resulting in a total of 43 remaining classes.

Appendix C Notes on precision-recall curves

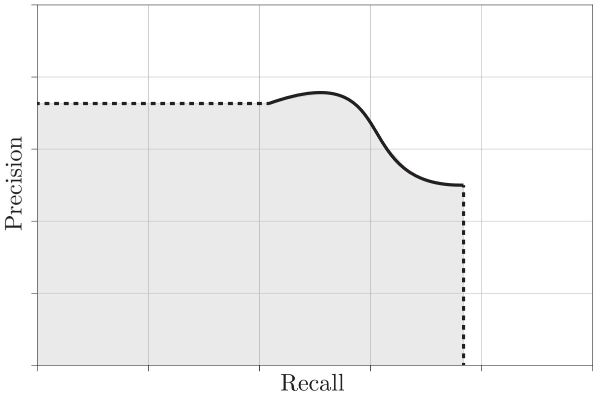


Figure 53: AUC calculation for curves which do not touch either the x-axis or the y-axis, i.e. curves that do not span the full domain of recall values and/or do not span the full range of precision values. Calculating AUC strictly under the curve would bias curves that occupy the widest interval of recall values. Instead, we attach a straight horizontal line from the leftmost endpoint (containing the smallest recall) to the y-axis, and a straight vertical from the right endpoint (with the largest recall) to the x-axis. AUC is then calculated for this resulting curve. AUC is indicated with light grey.

Precision can be plotted against recall in a *precision-recall curve*. This is often a decreasing function, since an increased recall might indicate a more careless prediction policy, causing more instances to be found at the cost of precision. However, a model that increases its number of predictions might consistently guess correct, in which case precision stagnates or even increases (with increasing recall), though this is admittedly a rare occurrence. Recall, on the other hand, is always monotonically decreasing with an increasing threshold. The predictions done at a high threshold always form a subset of the predictions done at any lower threshold, and such a subset can therefore only omit any detections from a lower threshold, resulting in an equal or lower recall.

We plot precision-recall curves by thresholding on prediction probability. Within the context of instance segmentation, thresholding can then be done by simply considering only predicted instances above a certain threshold of prediction certainty. The more thresholding and precision-recall pairs, the more faithfully this curve can be assessed. Such a curve can be drawn for each class separately, as well as for all classes combined. For semantic segmentation, it might initially be less obvious how this thresholding is accomplished, as every pixel needs exactly one label. For each class and each threshold, we remove any prediction of that class with softmax probability lower than the threshold, in which case those pixels have no predicted labels and will incur false negatives. In this way, a higher threshold will incur more mispredictions and a lower recall, but the remaining predictions are more likely to be correct, increasing precision.

Precision-recall curves can be a useful tool in determining optimal thresholding values, especially for the tasks of image classification, object detection, and instance segmentation. Based on the relative importance of precision versus recall for the purposes of the analysis, an optimal trade-off between the two can be found easily using such graphs. For semantic segmentation, however, PR curves prove less useful, as each pixel needs a predictions anyways, and thresholding will only remove pixel predictions. This can be done when performing binary segmentation, though, in which case PR curves might prove a useful analytical tool. A measure integrating precision and recall over all thresholds is the area under the precision-recall curve: the *AUC*, which stands for area-under-curve. In this work, we do not discuss PR curves or their AUC, but they are enclosed, for each experiment, in the appendices. Figure 53 shows how AUC is calculated.

For the sake of clarity, we mention that we require both precision and recall to be valid values for each threshold for them to get plotted. If no predictions have been made for a certain class, then the precision on that class constitutes a zero-division and is therefore undefined. Likewise, when there is no ground truth present for a certain class, recall is undefined. If either one or both of these measures is undefined, then the pair cannot be assigned a location in the graph and is skipped. In practice, though, precision-recall pairs can only get omitted at one of either ends of a curve. The predictions done at a high threshold always form a subset of the predictions done at any lower threshold, and therefore it cannot happen that precision is defined at threshold A, while there suddenly is no valid precision at a threshold B ($> A$), and at another threshold C ($> B$) there exists a valid precision again (the same holds for recall). As a consequence, "intermediate" points are never omitted.

Appendix D Code for loss functions

Loss functions play an important role in our research, especially SUBQUESTION I. For reference, we reproduce here our code for loss functions we used. These can be readily used in Keras' `tf.keras.Model` or `tf.keras.Model.fit_generator` functions. Class weighting was effectuated using the `class_weight` parameter of these functions.

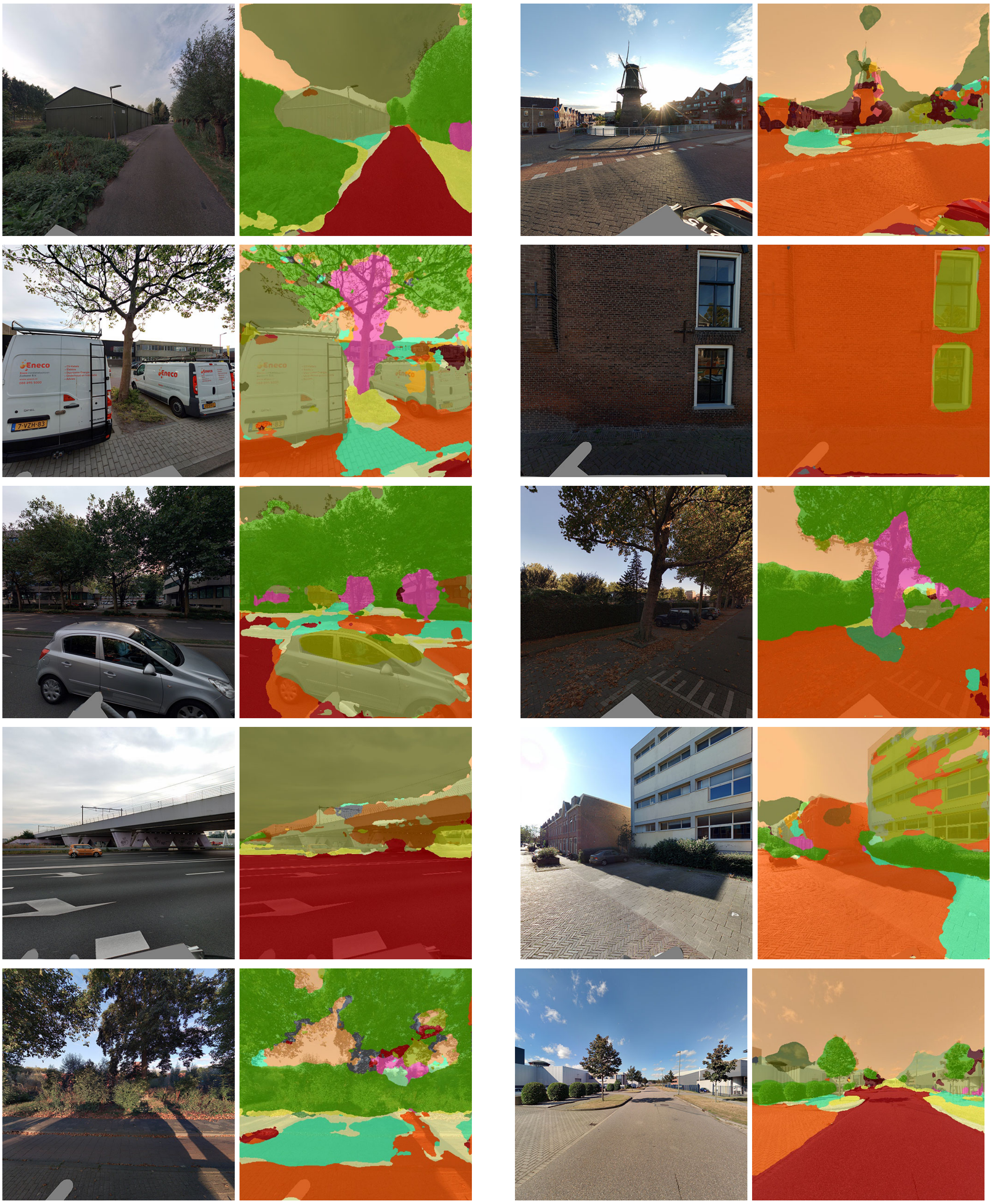
```python
def cross_entropy_loss(onehots_true, logits): # Inputs are [BATCH_SIZE, height, width, 3]
   logits, onehots_true = mask_pixels(onehots_true, logits)
   return tf.losses.softmax_cross_entropy(onehots_true, logits)

def dice_loss(onehots_true, logits):
   logits, onehots_true = mask_pixels(onehots_true, logits) # Both become of shape (num_pixels,
       num_classes)
   probabilities = tf.nn.softmax(logits)
   numerator = 2 * tf.reduce_sum(onehots_true * probabilities, axis=0)
   denominator = tf.reduce_sum(onehots_true + probabilities, axis=0)
   loss = 1.0 - (numerator + 1) / (denominator + 1)
   return loss

def focal_loss(onehots_true, logits):
   GAMMA = 2
   probabilities = tf.nn.softmax(logits)
   probabilities = tf.clip_by_value(probabilities, tf.keras.backend.epsilon(), 1 -
       tf.keras.backend.epsilon())
   cross_entropy = cross_entropy_loss(onehots_true, logits)
   loss = cross_entropy * tf.pow(1-probabilities, GAMMA) * onehots_true
   return tf.reduce_sum(loss, axis=1)
```

Appendix E Code for network definition

For reference, we include code implementing Semantic FPN:

```
def bilinearly_upsample(input_tensor, size):
   output_tensor = tf.image.resize_bilinear(images=input_tensor, size=size, align_corners=True)
   return output_tensor

def get_fpn(num_channels, config):
   input = tf.keras.Input(shape=(SCALED_HEIGHT, SCALED_WIDTH, num_channels))

   x = tf.keras.layers.Conv2D(64, (7, 7), strides=(2, 2), padding=’same’, name=’conv1’)(input)
   x = tf.keras.layers.BatchNormalization(axis=3, name=’bn_conv1’)(x)
   x = tf.keras.layers.Activation(’relu’)(x)
   x = tf.keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding=’same’)(x)
   x = conv_block(x, 3, [64, 64, 256], stage=2, block=’a’, strides=(1, 1))
   x = identity_block(x, 3, [64, 64, 256], stage=2, block=’b’)
   bottomup_xl = identity_block(x, 3, [64, 64, 256], stage=2, block=’c’) # (?, 127, 127, 256)

   x = conv_block(bottomup_xl, 3, [128, 128, 512], stage=3, block=’a’)
   x = identity_block(x, 3, [128, 128, 512], stage=3, block=’b’)
   x = identity_block(x, 3, [128, 128, 512], stage=3, block=’c’)
   bottomup_l = identity_block(x, 3, [128, 128, 512], stage=3, block=’d’) # (?, 64, 64, 512)

   x = conv_block(bottomup_l, 3, [256, 256, 1024], stage=4, block=’a’)
   x = identity_block(x, 3, [256, 256, 1024], stage=4, block=’b’)
   x = identity_block(x, 3, [256, 256, 1024], stage=4, block=’c’)
   x = identity_block(x, 3, [256, 256, 1024], stage=4, block=’d’)
   x = identity_block(x, 3, [256, 256, 1024], stage=4, block=’e’)
   bottomup_m = identity_block(x, 3, [256, 256, 1024], stage=4, block=’f’) # (?, 32, 32, 1024)

   x = conv_block(bottomup_m, 3, [512, 512, 2048], stage=5, block=’a’)
   x = identity_block(x, 3, [512, 512, 2048], stage=5, block=’b’)
   bottomup_s = identity_block(x, 3, [512, 512, 2048], stage=5, block=’c’) # (?, 16, 16, 2048)

   topdown_s = tf.keras.layers.Conv2D(1024, (1, 1), padding=’same’)(bottomup_s) # (?, 16, 16, 1024)
   # x = tf.keras.layers.UpSampling2D(size=(2,2))(topdown_s) # For nearest neighbour upscaling
   x = tf.keras.layers.Lambda(lambda x : bilinearly_upsample(x, (32,32)))(topdown_s) # For bilinear
       upsampling
   topdown_m = tf.keras.layers.add([x, tf.keras.layers.Conv2D(1024, (1, 1),
       padding=’same’)(bottomup_m)])

   # x = tf.keras.layers.UpSampling2D(size=(2, 2))(topdown_m) # (?, 64, 64, 1024) # For nearest
       neighbour upsampling
   x = tf.keras.layers.Lambda(lambda x: bilinearly_upsample(x, (64, 64)))(topdown_s) # For bilinear
       upsampling
   topdown_l = tf.keras.layers.add([x, tf.keras.layers.Conv2D(1024, (1, 1),
       padding=’same’)(bottomup_l)])
   # x = tf.keras.layers.UpSampling2D(size=(2, 2))(topdown_l) # (?, 128, 128, 1024) # For nearest
       neighbour upsampling
   x = tf.keras.layers.Lambda(lambda x: bilinearly_upsample(x, (128, 128)))(topdown_s) # For
       bilinear upsampling
   topdown_xl = tf.keras.layers.add([x, tf.keras.layers.Conv2D(1024, (1, 1),
       padding=’same’)(bottomup_xl)])

   pyramid_s = tf.keras.layers.Conv2D(256, (3, 3), padding=’same’)(topdown_s)
   pyramid_m = tf.keras.layers.Conv2D(256, (3, 3), padding=’same’)(topdown_m)
   pyramid_l = tf.keras.layers.Conv2D(256, (3, 3), padding=’same’)(topdown_l)
   pyramid_xl = tf.keras.layers.Conv2D(256, (3, 3), padding=’same’)(topdown_xl)

   resnet = applications.resnet50.ResNet50(include_top=False, weights=’imagenet’,
       input_shape=(SCALED_HEIGHT, SCALED_WIDTH, 3), pooling=None)
   weights = resnet.get_weights()
   if "d" in CHANNELS:
      if DEPTH_WEIGHT_INITIALIZATION == "glorot":
         glorot_input = tf.keras.Input(shape=(SCALED_HEIGHT, SCALED_WIDTH, num_channels))
         glorot_layer = tf.keras.layers.Conv2D(64, (7, 7), strides=(2, 2), padding=’same’,
             kernel_initializer=tf.keras.initializers.glorot_uniform())(glorot_input)
         glorot_model = tf.keras.Model(glorot_input, glorot_layer)
         weights[0] = np.concatenate([weights[0], glorot_model.weights[0][:,:,np.newaxis,-1,:]],
             axis=-2)
      elif DEPTH_WEIGHT_INITIALIZATION == "average_rgb":
         averaged_weights = np.expand_dims(np.average(weights[0], axis=-2), axis=-2)
         weights[0] = np.concatenate([weights[0], averaged_weights], axis=-2)
      else:
         paddings = tf.constant([[0, 0, ], [0, 0], [0, 1], [0, 0]])
         weights[0] = tf.pad(weights[0], paddings, mode='REFLECT')
   if CHANNELS == "d": weights[0] = tf.expand_dims(weights[0][:,:,-1,:], axis=-2)

   model = tf.keras.Model(input, [pyramid_s, pyramid_m, pyramid_l, pyramid_xl])
   if not NO_PRETRAINING: model.set_weights(weights)
   return model

def get_semantic_fpn(num_channels, config):
   def upsample(tensor, repetitions):
      if repetitions == 0:
         tensor = tf.keras.layers.Conv2D(128, (3, 3), padding='same')(tensor)
         tensor = tf.keras.layers.BatchNormalization()(tensor)
         #tensor = tf.contrib.layers.group_norm(tensor)
         return tf.keras.layers.ReLU()(tensor)
      # for _ in range(repetitions): # For nearest neighbour upscaling
      for dim in [32, 64, 128][repetitions-1:]: # For bilinear upscaling
         tensor = tf.keras.layers.Conv2D(128, (3, 3), padding='same')(tensor)
         tensor = tf.keras.layers.BatchNormalization()(tensor)
         tensor = tf.keras.layers.ReLU()(tensor)
         # tensor = tf.keras.layers.UpSampling2D(size=(2, 2))(tensor) # For nearest neighbour
             upscaling
         tensor = tf.keras.layers.Lambda(lambda x: bilinearly_upsample(x, (dim, dim)))(tensor) #
             For bilinear upscaling
      return tensor

   input = tf.keras.Input(shape=(SCALED_HEIGHT, SCALED_WIDTH, num_channels))
   fpn = get_fpn(num_channels, config)
   #fpn = get_fpn_resnext152(num_channels)
   pyramid_s, pyramid_m, pyramid_l, pyramid_xl = fpn(input)
   from_s = upsample(pyramid_s, 3)
   from_m = upsample(pyramid_m, 2)
   from_l = upsample(pyramid_l, 1)
   from_xl = upsample(pyramid_xl, 0)

   x = tf.keras.layers.add([from_s, from_m, from_l, from_xl])
   x = tf.keras.layers.Conv2D(len(LABELS), (1, 1), padding='same')(x)
   # output = tf.keras.layers.UpSampling2D(size=(4, 4))(x) # (?, 512, 512, 21) # For nearest
       neighbour upscaling
   output = tf.keras.layers.Lambda(lambda x: bilinearly_upsample(x, (512, 512)))(x) # For bilinear
       upscaling
   return tf.keras.Model(input, output)
```

Appendix F Prediction visualizations



RGB

RGBD

Prediction visualizations of Semantic FPN after training on 600 images of our material dataset. Only either RGB or RGBD predictions are shows for each unique image, since RGB and RGBD predictions are highly similar, to the extent shows in Figure 47c and 47d. Predictions were selected arbitrarily without bias towards higher quality segmentations.

Appendix G Full experimental results

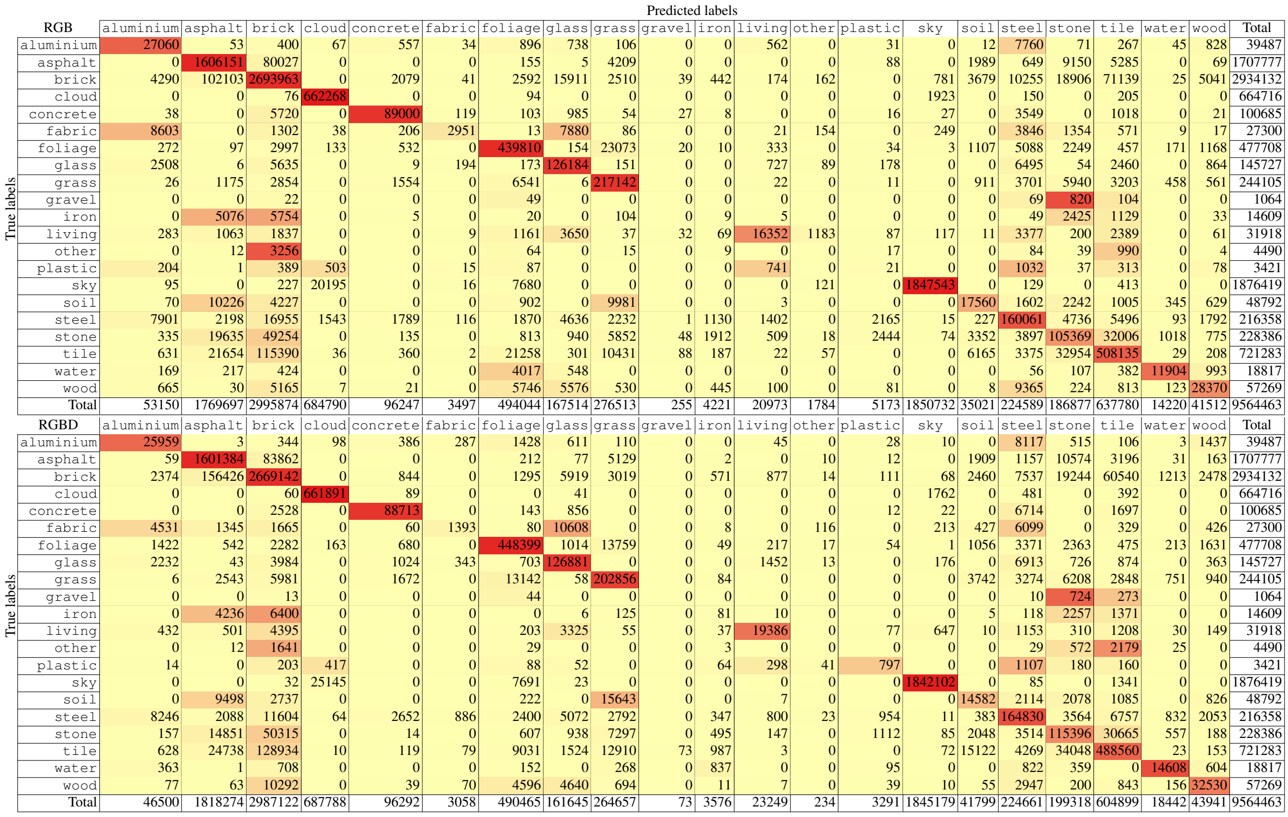
G.1 learning rate 0.01 • all 21 classes • cross entropy loss

	RGB	RGBD
mIoU	0.494	0.507
fIoU	0.809	0.802
accuracy	0.894	0.890
micro-averaged precision/recall/F1	0.894	0.890
macro-averaged precision	0.636	0.630
macro-averaged recall	0.576	0.594
macro-averaged F1	0.588	0.602
weighted macro-averaged precision	0.879	0.874
weighted macro-averaged recall	0.886	0.882
weighted macro-averaged F1	0.881	0.877

Performance overview

	RGB			RGBD			RGBD - RGB		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.494	0.636	0.576	0.507	0.630	0.594	+0.013	-0.006	+0.018
Weighted overall	0.809	0.879	0.886	0.802	0.874	0.882	-0.007	-0.005	-0.004
brick	0.832	0.899	0.918	0.820	0.893	0.909	-0.012	-0.006	-0.009
sky	0.982	0.998	0.984	0.980	0.998	0.981	-0.002	0.000	-0.003
asphalt	0.858	0.907	0.940	0.832	0.880	0.937	-0.026	-0.027	-0.003
tile	0.597	0.796	0.704	0.583	0.807	0.677	-0.014	+0.011	-0.027
foliage	0.826	0.890	0.920	0.862	0.914	0.938	+0.036	+0.024	+0.018
cloud	0.963	0.967	0.996	0.958	0.962	0.995	-0.005	-0.005	-0.001
grass	0.715	0.785	0.889	0.663	0.766	0.831	-0.052	-0.019	-0.058
glass	0.674	0.753	0.865	0.702	0.784	0.870	+0.028	+0.031	+0.005
stone	0.340	0.563	0.461	0.369	0.578	0.505	+0.029	+0.015	+0.044
steel	0.569	0.712	0.739	0.596	0.733	0.761	+0.027	+0.021	+0.022
concrete	0.824	0.924	0.883	0.819	0.921	0.881	-0.005	-0.003	-0.002
water	0.563	0.837	0.632	0.644	0.792	0.776	+0.081	-0.045	+0.144
wood	0.402	0.683	0.495	0.473	0.740	0.568	+0.071	+0.057	+0.073
soil	0.265	0.501	0.359	0.191	0.348	0.298	-0.074	-0.153	-0.061
other	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
aluminium	0.412	0.509	0.685	0.432	0.558	0.657	+0.020	+0.049	-0.028
gravel	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
iron	0.000	0.002	0.000	0.004	0.022	0.005	+0.004	+0.020	+0.005
living	0.447	0.779	0.512	0.541	0.833	0.607	+0.094	+0.054	+0.095
fabric	0.105	0.843	0.108	0.048	0.455	0.051	-0.057	-0.388	-0.057
plastic	0.002	0.004	0.006	0.134	0.242	0.232	+0.132	+0.238	+0.226

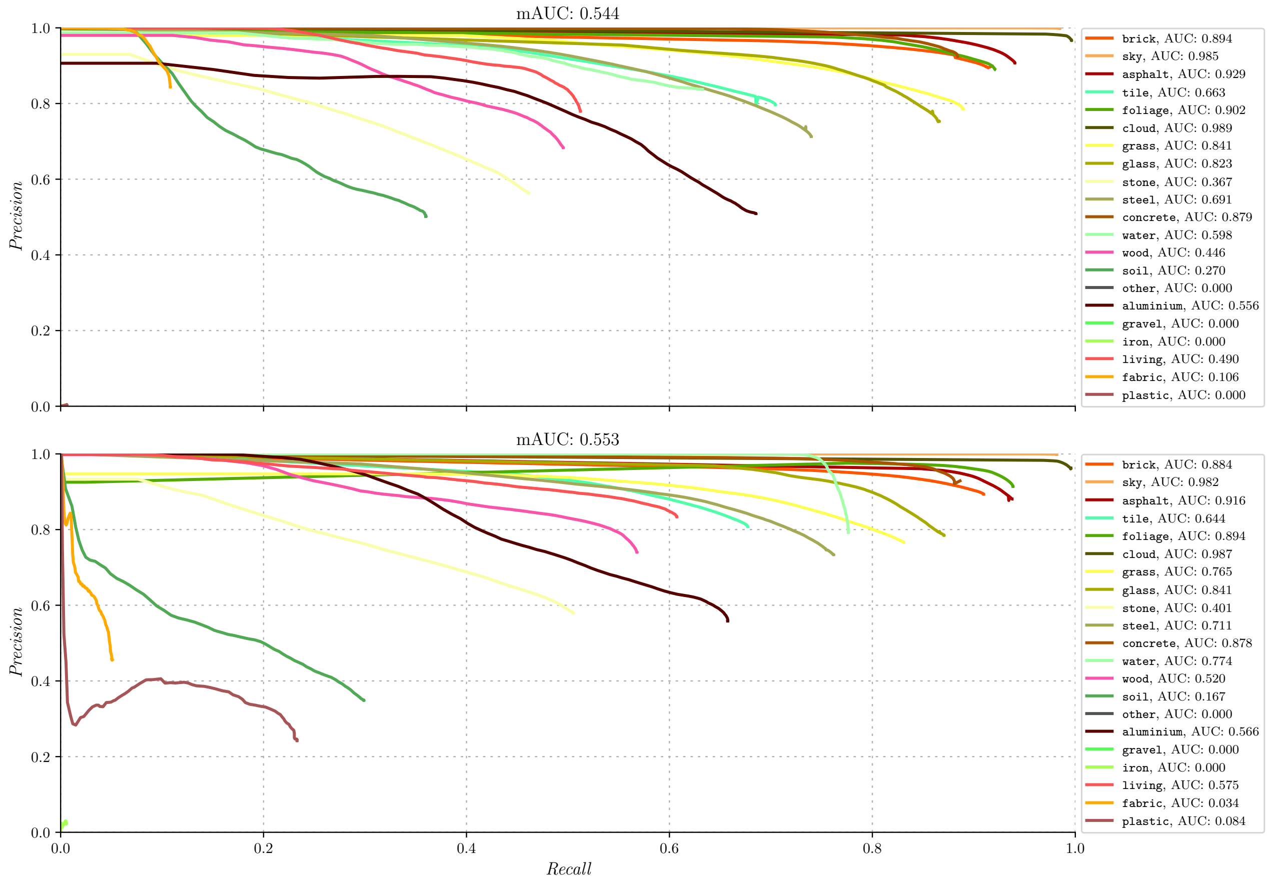
Per-class results


Predicted labels
True labels

RGB	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	27060	53	400	67	557	34	896	738	106	0	0	562	0	31	0	12	7760	71	267	45	828	39487
asphalt	0	1606151	80027	0	0	0	155	5	4209	0	0	0	0	88	0	1989	649	9150	5285	0	69	1707777
brick	4290	102103	2693963	0	2079	41	2592	15911	2510	39	442	174	162	0	781	3679	10255	18906	71139	25	5041	2934132
cloud	0	0	76	662268	0	0	94	0	0	0	0	0	0	0	1923	0	150	0	205	0	0	664716
concrete	38	0	5720	0	89000	119	103	985	54	27	8	0	0	16	27	0	3549	0	1018	0	21	100685
fabric	8603	0	1302	38	206	2951	13	7880	86	0	0	21	154	0	249	0	3846	1354	571	9	17	27300
foliage	272	97	2997	133	532	0	439810	154	23073	20	10	333	0	34	3	1107	5088	2249	457	171	1168	477708
glass	2508	6	5635	0	9	194	173	126184	151	0	0	727	89	178	0	0	6495	54	2460	0	864	145727
grass	26	1175	2854	0	1554	0	6541	6	217142	0	0	22	0	11	0	911	3701	5940	3203	458	561	244105
gravel	0	0	22	0	0	0	49	0	0	0	0	0	0	0	0	0	69	820	104	0	0	1064
iron	0	5076	5754	0	5	0	20	0	104	0	9	5	0	0	0	0	49	2425	1129	0	33	14609
living	283	1063	1837	0	0	9	1161	3650	37	32	69	16352	1183	87	117	11	3377	200	2389	0	61	31918
other	0	12	3256	0	0	0	64	0	15	0	9	0	0	17	0	0	84	39	990	0	4	4490
plastic	204	1	389	503	0	15	87	0	0	0	0	741	0	21	0	0	1032	37	313	0	78	3421
sky	95	0	227	20195	0	16	7680	0	0	0	0	0	121	0	1847543	0	129	0	413	0	0	1876419
soil	70	10226	4227	0	0	0	902	0	9981	0	0	3	0	0	0	17560	1602	2242	1005	345	629	48792
steel	7901	2198	16955	1543	1789	116	1870	4636	2232	1	1130	1402	0	2165	15	227	160061	4736	5496	93	1792	216358
stone	335	19635	49254	0	135	0	813	940	5852	48	1912	509	18	2444	74	3352	3897	105369	32006	1018	775	228386
tile	631	21654	115390	36	360	2	21258	301	10431	88	187	22	57	0	0	6165	3375	32954	508135	29	208	721283
water	169	217	424	0	0	0	4017	548	0	0	0	0	0	0	0	0	56	107	382	11904	993	18817
wood	665	30	5165	7	21	0	5746	5576	530	0	445	100	0	81	0	8	9365	224	813	123	28370	57269
Total	53150	1769697	2995874	684790	96247	3497	494044	167514	276513	255	4221	20973	1784	5173	1850732	35021	224589	186877	637780	14220	41512	9564463

RGBD	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	25959	3	344	98	386	287	1428	611	110	0	0	45	0	28	10	0	8117	515	106	3	1437	39487
asphalt	59	1601384	83862	0	0	0	212	77	5129	0	2	0	10	12	0	1909	1157	10574	3196	31	163	1707777
brick	2374	156426	2669142	0	844	0	1295	5919	3019	0	571	877	14	111	68	2460	7537	19244	60540	1213	2478	2934132
cloud	0	0	60	661891	89	0	0	41	0	0	0	0	0	0	1762	0	481	0	392	0	0	664716
concrete	0	0	2528	0	88713	0	143	856	0	0	0	0	0	12	22	0	6714	0	1697	0	0	100685
fabric	4531	1345	1665	0	60	1393	80	10608	0	0	8	0	116	0	213	427	6099	0	329	0	426	27300
foliage	1422	542	2282	163	680	0	448399	1014	13759	0	49	217	17	54	1	1056	3371	2363	475	213	1631	477708
glass	2232	43	3984	0	1024	343	703	126881	0	0	0	1452	13	0	176	0	6913	726	874	0	363	145727
grass	6	2543	5981	0	1672	0	13142	58	202856	0	84	0	0	0	0	3742	3274	6208	2848	751	940	244105
gravel	0	0	13	0	0	0	44	0	0	0	0	0	0	0	0	0	10	724	273	0	0	1064
iron	0	4236	6400	0	0	0	0	6	125	0	81	10	0	0	0	5	118	2257	1371	0	0	14609
living	432	501	4395	0	0	0	203	3325	55	0	37	19386	0	77	647	10	1153	310	1208	30	149	31918
other	0	12	1641	0	0	0	29	0	0	0	3	0	0	0	0	0	29	572	2179	25	0	4490
plastic	14	0	203	417	0	0	88	52	0	0	64	298	41	797	0	0	1107	180	160	0	0	3421
sky	0	0	32	25145	0	0	7691	23	0	0	0	0	0	0	1842102	0	85	0	1341	0	0	1876419
soil	0	9498	2737	0	0	0	222	0	15643	0	0	7	0	0	0	14582	2114	2078	1085	0	826	48792
steel	8246	2088	11604	64	2652	886	2400	5072	2792	0	347	800	23	954	11	383	164830	3564	6757	832	2053	216358
stone	157	14851	50315	0	14	0	607	938	7297	0	495	147	0	1112	85	2048	3514	115396	30665	557	188	228386
tile	628	24738	128934	10	119	79	9031	1524	12910	73	987	3	0	0	72	15122	4269	34048	488560	23	153	721283
water	363	1	708	0	0	0	152	0	268	0	837	0	0	95	0	0	822	359	0	14608	604	18817
wood	77	63	10292	0	39	70	4596	4640	694	0	11	7	0	39	10	55	2947	200	843	156	32530	57269
Total	46500	1818274	2987122	687788	96292	3058	490465	161645	264657	73	3576	23249	234	3291	1845179	41799	224661	199318	604899	18442	43941	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.2 learning rate 0.01 • all 21 classes • class-weighted cross entropy loss

	RGB	RGBD
mIoU	0.509	0.503
fIoU	0.808	0.807
accuracy	0.895	0.895
micro-averaged precision/recall/F1	0.895	0.895
macro-averaged precision	0.659	0.646
macro-averaged recall	0.591	0.586
macro-averaged F1	0.610	0.599
weighted macro-averaged precision	0.879	0.880
weighted macro-averaged recall	0.886	0.886
weighted macro-averaged F1	0.881	0.882

Performance overview

	Class-weighted cross entropy loss						Difference with unweighted cross entropy loss					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.509	0.659	0.591	0.503	0.646	0.586	+0.015	+0.023	+0.015	-0.004	+0.016	0.000
Weighted overall	0.808	0.879	0.886	0.807	0.880	0.886	-0.001	0.000	0.000	+0.005	+0.006	+0.004
brick	0.830	0.892	0.922	0.828	0.902	0.909	-0.002	-0.007	+0.004	+0.008	+0.009	0.000
sky	0.975	0.994	0.980	0.977	0.998	0.979	-0.007	-0.004	-0.004	-0.003	0.000	-0.002
asphalt	0.858	0.901	0.947	0.825	0.881	0.928	0.000	-0.006	+0.007	-0.007	+0.001	-0.009
tile	0.604	0.816	0.698	0.621	0.808	0.728	+0.007	+0.020	-0.006	+0.038	+0.001	+0.051
foliage	0.843	0.903	0.926	0.849	0.894	0.944	+0.017	+0.013	+0.006	-0.013	-0.020	+0.006
cloud	0.945	0.957	0.986	0.952	0.954	0.997	-0.018	-0.010	-0.010	-0.006	-0.008	+0.002
grass	0.691	0.795	0.841	0.711	0.827	0.835	-0.024	+0.010	-0.048	+0.048	+0.061	+0.004
glass	0.710	0.798	0.864	0.715	0.785	0.888	+0.036	+0.045	-0.001	+0.013	+0.001	+0.018
stone	0.332	0.567	0.444	0.398	0.617	0.529	-0.008	+0.004	-0.017	+0.029	+0.039	+0.024
steel	0.573	0.703	0.757	0.614	0.718	0.809	+0.004	-0.009	+0.018	+0.018	-0.015	+0.048
concrete	0.806	0.870	0.916	0.829	0.908	0.905	-0.018	-0.054	+0.033	+0.010	-0.013	+0.024
water	0.652	0.774	0.805	0.467	0.614	0.660	+0.089	-0.063	+0.173	-0.177	-0.178	-0.116
wood	0.406	0.709	0.487	0.440	0.720	0.531	+0.004	+0.026	-0.008	-0.033	-0.020	-0.037
soil	0.237	0.531	0.299	0.339	0.636	0.421	-0.028	+0.030	-0.060	+0.148	+0.288	+0.123
other	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
aluminium	0.450	0.600	0.643	0.501	0.671	0.665	+0.038	+0.091	-0.042	+0.069	+0.113	+0.008
gravel	0.143	0.542	0.163	0.000	0.000	0.000	+0.143	+0.542	+0.163	0.000	0.000	0.000
iron	0.006	0.062	0.007	0.044	0.297	0.049	+0.006	+0.060	+0.007	+0.040	+0.275	+0.044
living	0.394	0.764	0.449	0.353	0.695	0.418	-0.053	-0.015	-0.063	-0.188	-0.138	-0.189
fabric	0.235	0.660	0.268	0.092	0.639	0.098	+0.130	-0.183	+0.160	+0.044	+0.184	+0.047
plastic	0.003	0.008	0.004	0.007	0.013	0.017	+0.001	+0.004	-0.002	-0.127	-0.229	-0.215

Per-class results

Predicted labels

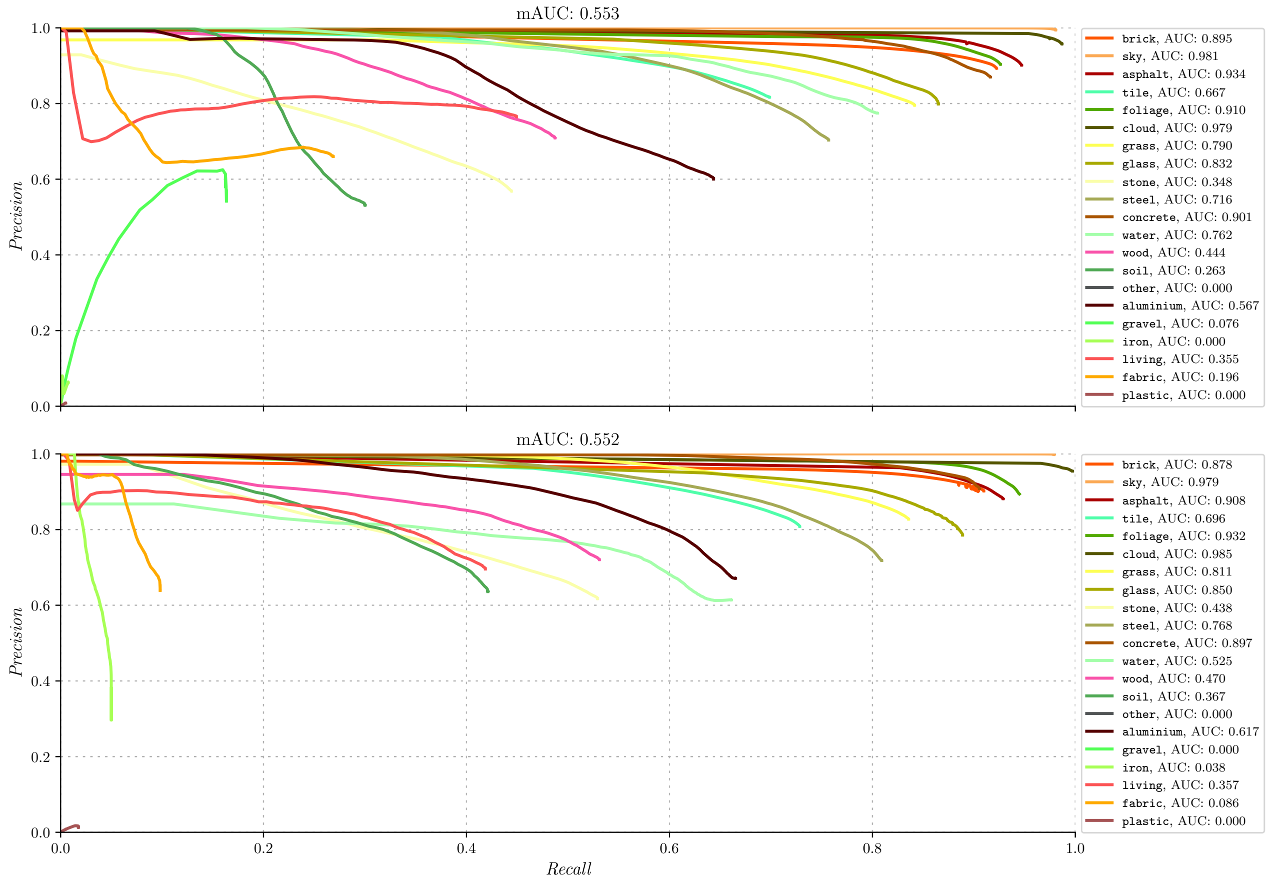
True labels

RGB	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	25415	0	547	45	464	1256	776	968	186	0	0	155	24	0	107	0	8381	369	43	36	715	39487
asphalt	65	1617394	79728	0	0	0	50	21	1657	0	0	0	17	78	0	62	452	5568	1815	650	220	1707777
brick	794	119366	2706214	0	10639	102	3154	7132	2121	0	5	80	2	0	376	3401	10566	12599	55703	564	1314	2934132
cloud	0	0	288	656024	0	0	2	0	0	0	0	0	0	0	8342	0	60	0	0	0	0	664716
concrete	1828	10	220	0	92254	223	173	4	90	0	0	0	0	0	83	0	3395	0	1753	409	243	100685
fabric	6508	19	1276	37	0	7332	8	8332	0	0	0	0	0	32	40	0	2439	29	0	0	1248	27300
foliage	555	44	1116	605	335	48	442452	1065	21194	2	0	297	2	0	109	638	4875	1288	1432	255	1396	477708
glass	1837	24	4872	0	0	102	153	126046	81	0	0	2738	0	0	0	0	7215	48	1950	34	627	145727
grass	5	3568	5705	0	196	0	7797	12	205430	0	88	83	0	0	0	2786	6260	6733	4146	924	372	244105
gravel	0	0	194	0	0	0	68	0	0	174	0	0	0	0	0	0	41	334	253	0	0	1064
iron	0	4727	6220	0	0	0	374	6	104	0	108	0	0	2	0	185	165	1935	783	0	0	14609
living	446	289	3248	3	0	39	1722	2794	90	0	2	14347	0	146	241	0	6163	2237	151	0	0	31918
other	0	18	3600	0	0	0	456	8	20	0	0	0	0	0	0	0	12	58	288	30	0	4490
plastic	221	0	361	452	0	0	138	0	0	0	7	177	0	17	0	0	1630	152	266	0	0	3421
sky	0	0	23	27601	0	2	7818	74	0	0	0	0	0	0	1840163	0	3	0	728	0	7	1876419
soil	0	9320	4082	0	27	0	1107	0	11702	0	0	81	0	0	0	14636	1092	3664	2317	0	764	48792
steel	2590	1699	19211	148	1926	1973	2570	4505	1394	0	849	533	5	110	337	275	163798	3478	7140	573	3244	216358
stone	401	21247	56843	0	161	30	366	267	2861	0	593	72	0	1419	0	3027	5175	101462	33309	619	534	228386
tile	411	16229	128262	0	2	2	14919	812	10513	138	38	135	796	17	86	2254	3727	38226	504032	78	606	721283
water	433	0	330	0	0	0	1956	0	50	0	0	0	0	0	0	1	575	183	0	15150	139	18817
wood	825	22	9183	0	0	0	3464	5772	748	7	40	59	299	142	174	280	6831	277	1011	232	27903	57269
Total	42334	1793976	3031523	684915	106004	11109	489523	157818	258241	321	1730	18757	1145	1963	1850058	27545	232855	178640	617120	19554	39332	9564463

True labels

RGBD	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	26269	0	1383	203	579	312	821	1355	599	0	0	889	0	23	18	0	5972	208	0	141	715	39487
asphalt	0	1586394	89590	0	167	0	634	58	6881	5	197	0	0	53	1	1650	285	10370	11357	12	123	1707777
brick	2974	150908	2669977	8	1885	452	2558	6389	1457	0	252	52	1	495	60	2704	12444	21247	57154	659	2456	2934132
cloud	0	0	0	662879	64	0	81	0	0	0	0	0	0	0	1248	0	444	0	0	0	0	664716
concrete	113	0	335	0	91185	0	0	757	0	0	0	0	532	57	321	0	5859	0	1432	0	94	100685
fabric	2181	216	657	11	122	2677	372	12512	0	0	0	53	9	0	458	0	6616	167	932	0	317	27300
foliage	1257	1757	778	247	1029	0	451415	727	9378	0	83	739	0	0	6	476	3845	1164	1518	260	3029	477708
glass	1238	351	5144	0	152	497	110	129530	39	0	0	967	0	0	11	11	5845	26	1459	0	347	145727
grass	453	3383	3735	0	1212	0	10846	264	204011	72	0	0	13	68	0	231	5692	4140	3459	5754	772	244105
gravel	0	0	10	0	0	0	262	0	0	0	0	0	0	0	0	0	18	608	166	0	0	1064
iron	0	2468	7227	0	0	0	385	3	133	0	730	5	0	10	0	0	58	2752	836	0	2	14609
living	775	195	8177	0	0	0	443	3748	49	0	0	13359	0	899	0	0	1689	1031	1543	0	10	31918
other	0	12	1350	0	0	0	81	656	0	0	0	0	0	0	0	0	97	51	2137	25	81	4490
plastic	142	0	264	0	90	0	129	0	0	0	0	398	0	60	0	0	1701	154	483	0	0	3421
sky	0	0	68	31014	0	0	7719	14	0	0	0	0	0	0	1837216	0	2	0	386	0	0	1876419
soil	21	12536	2497	0	0	0	846	0	6333	0	0	27	0	5	0	20546	1821	1805	1105	143	1107	48792
steel	2172	3918	12467	24	2783	75	2338	2477	1019	0	588	1661	20	956	331	89	175122	4747	4286	189	1096	216358
stone	43	16920	42801	0	12	0	297	145	4631	0	554	261	8	1456	0	1671	3873	120900	34232	462	120	228386
tile	480	16989	107781	0	936	0	21061	1696	11399	8	0	673	0	298	36	4844	3458	25566	525505	74	479	721283
water	843	0	277	0	0	0	2003	40	104	0	31	0	0	0	0	21	1773	240	0	12438	1047	18817
wood	167	3368	4313	0	176	176	2518	4540	483	0	19	113	49	175	843	61	7196	642	1928	84	30418	57269
Total	39128	1799415	2958831	694386	100392	4189	504919	164911	246516	85	2454	19197	632	4555	1840549	32304	243810	195818	649918	20241	42213	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.3 learning rate 0.01 • all 21 classes • cross entropy loss, second run

	Cross entropy loss, second run						Difference with first run					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.496	0.615	0.580	0.502	0.619	0.588	+0.002	-0.021	+0.004	-0.005	-0.011	0.000
Weighted overall	0.804	0.876	0.883	0.815	0.883	0.890	-0.005	-0.003	-0.003	+0.013	+0.009	+0.008
brick	0.825	0.891	0.917	0.838	0.899	0.925	-0.007	-0.008	-0.001	+0.018	+0.006	+0.016
sky	0.979	0.998	0.981	0.980	0.997	0.983	-0.003	0.000	-0.003	0.000	-0.001	+0.002
asphalt	0.832	0.884	0.935	0.876	0.925	0.942	-0.026	-0.023	-0.005	+0.044	+0.045	+0.005
tile	0.614	0.829	0.703	0.588	0.794	0.694	+0.017	+0.033	-0.001	+0.005	-0.013	+0.017
foliage	0.856	0.926	0.918	0.843	0.889	0.942	+0.030	+0.036	-0.002	-0.019	-0.025	+0.004
cloud	0.955	0.960	0.993	0.961	0.966	0.994	-0.008	-0.007	-0.003	+0.003	+0.004	-0.001
grass	0.711	0.802	0.862	0.721	0.794	0.888	-0.004	+0.017	-0.027	+0.058	+0.028	+0.057
glass	0.660	0.730	0.872	0.727	0.788	0.903	-0.014	-0.023	+0.007	+0.025	+0.004	+0.033
stone	0.354	0.597	0.465	0.349	0.543	0.495	+0.014	+0.034	+0.004	-0.020	-0.035	-0.010
steel	0.538	0.700	0.698	0.561	0.712	0.726	-0.031	-0.012	-0.041	-0.035	-0.021	-0.035
concrete	0.850	0.914	0.923	0.806	0.913	0.873	+0.026	-0.010	+0.040	-0.013	-0.008	-0.008
water	0.607	0.822	0.699	0.551	0.867	0.601	+0.044	-0.015	+0.067	-0.093	+0.075	-0.175
wood	0.333	0.673	0.398	0.535	0.777	0.632	-0.069	-0.010	-0.097	+0.062	+0.037	+0.064
soil	0.232	0.339	0.426	0.129	0.406	0.160	-0.033	-0.162	+0.067	-0.062	+0.058	-0.138
other	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
aluminium	0.452	0.599	0.648	0.448	0.555	0.699	+0.040	+0.090	-0.037	+0.016	-0.003	+0.042
gravel	0.000	0.000	0.000	0.002	0.028	0.002	0.000	0.000	0.000	+0.002	+0.028	+0.002
iron	0.013	0.109	0.014	0.093	0.304	0.118	+0.013	+0.107	+0.014	+0.089	+0.282	+0.113
living	0.552	0.766	0.664	0.456	0.721	0.553	+0.105	-0.013	+0.152	-0.085	-0.112	-0.054
fabric	0.016	0.318	0.017	0.001	0.012	0.001	-0.089	-0.525	-0.091	-0.047	-0.443	-0.050
plastic	0.026	0.060	0.044	0.079	0.111	0.216	+0.024	+0.056	+0.038	-0.055	-0.131	-0.016

Per-class results. The right hand side shows the difference with the exact same experiment retrained and retested. This table highlight the volatility of some of the small classes. Strangely enough, the probabilities are very similar between RGB and RGBD for both runs of the experiment, even though those are themselves separate experiments. Both runs have been run on different GPUs and were executed a few days apart (which might have an influence on any random seed derived from a datetime), but we are still unable to pinpoint were the similarity comes from.

Predicted labels

True labels

RGB	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	-1449	-15	+928	-67	-67	+211	-148	+257	+31	+0	+3	+35	+0	+411	+64	+8	-807	+285	+220	-45	+145	+0
asphalt	+2	-9201	+10196	+0	+361	+0	-149	-2	-546	+0	+0	+0	+0	-88	+0	+1380	+364	-2112	-227	+10	+12	+0
brick	-4061	+34817	-2362	+1000	+3064	-32	-806	-10209	+730	+40	+419	+1109	+81	+64	+185	+579	+5750	-2425	-26150	+1576	-3369	+0
cloud	+0	+0	+119	-1581	+0	+0	+59	+58	+0	+0	+0	+0	+0	+0	-91	+6	+1474	+0	-44	+0	+0	+0
concrete	+501	+0	-4053	+0	+4020	-119	-82	-490	-54	-27	-8	+81	+0	-16	+155	+0	-367	+968	-1007	+0	+498	+0
fabric	-2232	+414	-974	+352	+12	-2471	+6	+6653	-86	+0	+0	-21	-154	+0	+9	+0	-1545	-1174	+883	-9	+337	+0
foliage	+2236	+6	-1091	+38	-263	+31	-1094	+554	-688	-20	+179	-327	+0	-30	+8	+101	+218	-1347	-260	+123	+1626	+0
glass	-1497	-6	+2120	+0	-5	-57	-101	+995	-78	+0	+0	+1090	-89	-178	+0	+15	-779	+117	-741	+0	-806	+0
grass	-5	+1279	+477	+0	-1160	+0	+1859	+190	-6497	+5	+8	+110	+0	-11	+0	+5909	-642	-2500	+1264	-352	+66	+0
gravel	+0	+0	-22	+0	+0	+0	+1	+0	+0	+0	+0	+0	+0	+0	+0	+0	-44	-168	+233	+0	+0	+0
iron	+0	-482	+185	+0	-5	+0	+2	+0	+133	+0	+205	-5	+0	+0	+0	+0	+88	-308	+210	+10	-33	+0
living	+198	-1019	+2767	+0	+0	+10	-1092	-1218	-32	-32	-36	+4866	-1176	-63	-117	-11	-957	-157	-1984	+5	+48	+0
other	+0	+0	+109	+0	+0	+0	-58	+102	-11	+0	-9	+0	+0	-17	+0	+0	-34	+613	-780	+25	+60	+0
plastic	+10	-1	-101	-456	+0	-15	+0	+8	+0	+0	+0	-426	+0	+130	+0	+0	+895	+76	-43	+0	-77	+0
sky	-95	+0	+290	+4783	+0	-16	+90	+0	+0	+0	+0	+0	-121	+0	-5374	+0	+346	+0	+97	+0	+0	+0
soil	-69	+987	-396	+0	+0	+0	-162	+0	-3421	+0	+65	+28	+0	+1	+0	+3246	-1558	+654	+1093	-345	-123	+0
steel	-3319	+1939	-2457	-1342	-82	+168	+169	+10278	+304	+12	-888	+150	+0	-1199	-5	-122	-8848	+5792	-1308	-49	+807	+0
stone	-43	-2487	+3056	+0	-134	+71	-517	-877	+514	+42	-1674	+116	-11	-1613	-74	-566	+349	+997	+3887	-490	-546	+0
tile	-453	+8634	+6056	-36	-353	+10	-17342	+1075	-4348	-46	-85	+7	-56	+4	+311	+15670	+402	-9103	-542	+37	+158	+0
water	+225	-217	+336	+0	+0	+0	-1067	-548	+0	+0	+0	+0	+0	+1	+0	+0	+735	+550	-379	+1264	-900	+0
wood	-410	+1988	+8116	+251	+73	+220	-225	-312	-63	+0	-443	-100	+38	-65	+30	+121	-3847	+293	-146	+39	-5558	+0
Total	-10461	+36636	+23299	+2942	+5461	-1989	-20657	+6514	-14112	-26	-2264	+6713	-1488	-2669	-4899	+26336	-8807	-8949	-25724	+1799	-7655	+0

True labels

RGBD	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	+1651	+9	-56	-98	-55	+499	-488	+128	+355	+0	+43	+387	+0	-15	-10	+0	-751	-438	-34	-2	-1125	+0
asphalt	+21	+7391	-7074	+0	+208	+0	+141	-67	-1703	+0	-2	+3	-10	-12	+0	-1686	-245	+1713	+1148	-31	+205	+0
brick	-1269	-71633	+47434	+1	+379	+300	+759	+2391	-896	+0	+61	-115	-4	-96	+277	-10	+8516	+7919	+6949	-518	-445	+0
cloud	+3	+0	-60	-753	-89	+129	+0	+18	+0	+0	+0	+0	+0	+0	+1375	+0	-335	+0	-288	+0	+0	+0
concrete	+8	+78	+1411	+0	-796	+0	+1636	-200	+45	+0	+0	+0	+0	+39	-22	+0	-1709	+323	-975	+0	+162	+0
fabric	+4470	-1193	+879	+66	+1196	-1358	-74	-2575	+0	+0	-8	+11	-99	+0	+482	-427	-3631	+24	+2612	+0	-375	+0
foliage	-55	-542	-753	+85	-417	+0	+1945	-809	-2222	+83	-49	+874	-17	+43	+3	-866	+1615	-1031	+346	+63	+1704	+0
glass	-1363	+168	-539	+0	-1004	-138	-663	+4838	+36	+0	+0	-328	-5	+101	-176	+0	+290	-708	-332	+0	-177	+0
grass	+4	-1100	-2469	+0	-949	+0	-6635	-18	+13979	+0	+67	+3	+0	+12	+0	-2516	+409	-910	+1147	-600	-424	+0
gravel	+0	+0	-13	+0	+0	+0	-39	+0	+0	+3	+0	+0	+0	+0	+0	+0	+12	-245	+282	+0	+0	+0
iron	+4	-1550	-67	+0	+0	+0	+63	-6	-47	+0	+1652	-5	+0	+0	+0	+23	-32	-314	+277	+2	+0	+0
living	-132	-177	-1175	+0	+0	+0	+165	+3235	-32	+0	+58	-1714	+150	-39	-349	-10	+495	-62	-379	-23	-11	+0
other	+0	+4	+746	+0	+0	+0	-17	+0	+0	+0	+22	+0	+0	+0	+0	+0	+52	-546	-236	-25	+0	+0
plastic	+144	+17	+91	-173	+0	+90	+20	+43	+0	+0	-64	+386	-41	-55	+0	+0	-272	-136	-50	+0	+0	+0
sky	+8	+0	-32	-2861	+0	+3	+9	+115	+0	+0	+0	+0	+0	+0	+2867	+0	+606	+0	-715	+0	+0	+0
soil	+0	-896	+296	+0	+0	+0	-160	+0	+7393	+0	+0	+82	+0	+0	+0	-6771	-1096	+343	+776	+42	-9	+0
steel	-583	-910	+7843	+226	+1538	-732	+743	+1790	-1354	+2	+262	+761	-23	+1433	+215	-201	-7611	+115	-2403	-830	-281	+0
stone	+61	+502	-4577	+0	-14	+399	-193	-646	-848	+0	+1207	+717	+0	+2020	-85	-1290	-722	-2250	+5947	-178	-50	+0
tile	-488	-10986	-3566	-10	+12	-78	+15028	-468	-5828	-72	-987	+148	+0	+23	+75	-8912	-1446	+5048	+12528	+10	-31	+0
water	+291	-1	-294	+0	+0	+332	+3780	+34	-268	+0	-208	+0	+0	-75	+0	+0	-164	+23	+0	-3282	-168	+0
wood	+428	-29	-3629	+22	-18	+364	+2	-2430	-210	+15	+61	+38	+0	-39	+21	+104	+2024	+38	-442	-18	+3698	+0
Total	+3203	-80848	+34396	-3495	-9	-190	+16022	+5373	+8400	+31	+2115	+1248	-49	+3340	+4673	-22562	-3995	+8906	+26158	-5390	+2673	+0

Difference in confusion matrices between the two runs

G.4 learning rate 0.01 • all 21 classes • class-weighted cross entropy loss, second run

	Class-weighted cross entropy loss, second run						Difference with first run					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.515	0.636	0.601	0.504	0.648	0.590	+0.006	-0.023	+0.010	+0.001	+0.002	0.000
Weighted overall	0.809	0.880	0.888	0.811	0.881	0.888	+0.001	+0.001	+0.002	+0.004	+0.001	+0.002
brick	0.824	0.898	0.908	0.840	0.900	0.926	-0.006	+0.006	-0.014	+0.012	-0.002	+0.017
sky	0.977	0.997	0.979	0.982	0.997	0.983	+0.002	+0.003	-0.001	+0.005	-0.001	+0.004
asphalt	0.841	0.887	0.941	0.853	0.906	0.935	-0.017	-0.014	-0.006	+0.028	+0.025	+0.007
tile	0.598	0.813	0.694	0.618	0.812	0.722	-0.006	-0.003	-0.004	-0.003	+0.004	-0.006
foliage	0.844	0.889	0.944	0.840	0.901	0.925	+0.001	-0.014	+0.018	-0.009	+0.007	-0.019
cloud	0.952	0.955	0.996	0.965	0.967	0.997	+0.007	-0.002	+0.010	+0.013	+0.013	0.000
grass	0.731	0.830	0.859	0.689	0.785	0.849	+0.040	+0.035	+0.018	-0.022	-0.042	+0.014
glass	0.710	0.786	0.880	0.683	0.776	0.850	0.000	-0.012	+0.016	-0.032	-0.009	-0.038
stone	0.355	0.588	0.472	0.370	0.579	0.507	+0.023	+0.021	+0.028	-0.028	-0.038	-0.022
steel	0.571	0.681	0.780	0.575	0.724	0.736	-0.002	-0.022	+0.023	-0.039	+0.006	-0.073
concrete	0.832	0.907	0.909	0.778	0.890	0.860	+0.026	+0.037	-0.007	-0.051	-0.018	-0.045
water	0.660	0.761	0.832	0.491	0.656	0.662	+0.008	-0.013	+0.027	+0.024	+0.042	+0.002
wood	0.471	0.736	0.567	0.418	0.682	0.519	+0.065	+0.027	+0.080	-0.022	-0.038	-0.012
soil	0.439	0.679	0.554	0.262	0.508	0.352	+0.202	+0.148	+0.255	-0.077	-0.128	-0.069
other	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
aluminium	0.506	0.678	0.665	0.460	0.616	0.645	+0.056	+0.078	+0.022	-0.041	-0.055	-0.020
gravel	0.008	0.027	0.012	0.000	0.000	0.000	-0.135	-0.515	-0.151	0.000	0.000	0.000
iron	0.014	0.075	0.017	0.019	0.075	0.025	+0.008	+0.013	+0.010	-0.025	-0.222	-0.024
living	0.431	0.690	0.534	0.475	0.753	0.562	+0.037	-0.074	+0.085	+0.122	+0.058	+0.144
fabric	0.036	0.419	0.038	0.083	0.694	0.086	-0.199	-0.241	-0.230	-0.009	+0.055	-0.012
plastic	0.023	0.068	0.033	0.177	0.389	0.246	+0.020	+0.060	+0.029	+0.170	+0.376	+0.229

Per-class results. As with unweighted cross entropy loss, performance of the smallest classes appears to change measurably between different runs of the same experiment. other remains unchanged in that it is still not being predicted. aluminium and living have changed little, but they already performed quite well. gravel and iron show variable precision values, but these can be ascribed to noise given their low recalls. fabric and plastic are the most volatile, which reinforces the fact that they are most susceptible to noise due to their class size.

Predicted labels

True labels

RGB	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	+859	+36	+171	-22	+570	-761	+397	-437	+37	+0	+97	+871	-24	+51	-107	+15	-1706	-333	+249	+148	-111	+0
asphalt	-65	-9511	-832	+0	+0	+0	-3	+554	+2559	+73	+26	+0	-6	-64	+0	+58	-1	+3043	+4437	-582	+314	+0
brick	-416	+23764	-39363	+188	-8665	+50	-303	-319	-983	+0	+1630	+1563	+96	+30	+849	+2549	+9435	+10019	-1015	+1247	-356	+0
cloud	+79	+0	+114	+6501	+0	+0	+533	+0	+0	+0	+0	+0	+0	+0	-7502	+0	+233	+0	+42	+0	+0	+0
concrete	-1799	-10	+1759	+10	-701	-223	-159	+321	-21	+0	+0	+0	+0	+0	-83	+0	+1048	+1065	-1231	-409	+433	+0
fabric	-2529	+262	-79	-9	+127	-6272	+25	+2911	+0	+0	+211	+115	+1116	-32	+492	+23	+4275	+276	+300	+0	-1212	+0
foliage	+132	+1415	+2869	-188	-173	-48	+8715	-1004	-10407	-2	+454	+79	+55	+25	-109	+178	-844	-378	-902	+13	+120	+0
glass	+399	-24	-1817	+0	+43	+37	-153	+2337	-49	+0	+0	-2077	+0	+126	+1089	+0	+1035	+126	-778	-34	-260	+0
grass	-5	-456	-3454	+0	+209	+0	+4566	+124	+4351	+181	-88	-83	+0	+195	+5	-248	-3081	-2979	+87	+71	+605	+0
gravel	+0	+0	-194	+0	+0	+0	+71	+0	+0	-161	+0	+0	+0	+0	+0	+0	-41	+408	-174	+0	+91	+0
iron	+0	+46	-579	+0	+3	+0	-9	+34	-67	+0	+153	+2	+0	-2	+0	-172	-60	+407	+244	+0	+0	+0
living	-34	-177	-1485	-3	+0	+472	-1520	+1556	-79	+0	+3	+2713	+0	-145	-241	+0	-1264	-1786	+1858	+29	+103	+0
other	+42	+49	-2573	+0	+0	+0	-345	+693	+53	+0	+0	+0	+0	+120	+0	+8	+80	+56	+1788	-30	+59	+0
plastic	+102	+1	+27	-284	+0	+0	+50	+0	+10	+0	-7	+340	+0	+99	+0	+0	-433	-138	+216	+0	+17	+0
sky	+150	+0	-23	+1772	+0	-2	-138	-74	+0	+0	+0	+0	+0	+0	-1339	+0	+181	+0	-520	+0	-7	+0
soil	+8	-2769	-565	+0	-27	+0	-898	+0	-6213	+0	+0	+12	+0	+0	+0	+12437	+157	-2273	-300	+0	+431	+0
steel	+254	+1498	-5580	+112	+1395	-1968	-352	+370	+484	+0	-674	+794	+50	+277	-67	-246	+5128	+3453	-4560	-402	+34	+0
stone	-160	-2411	-10456	+0	-108	-30	+276	-146	+4996	+12	-293	+146	+22	-1035	+0	-1797	+2342	+6403	+1947	+449	-157	+0
tile	+111	+4846	+3794	+0	-2	+2	+7989	+2281	-759	+53	+254	+491	-796	+126	-86	-272	-954	-13492	-3329	+26	-283	+0
water	-352	+69	-213	+0	+0	+0	-595	+0	-50	+0	+0	+219	+0	+0	+0	-1	-97	+127	+0	+519	+374	+0
wood	-400	+214	-4010	+40	+2224	+160	-229	-3775	+372	-7	-32	+755	-299	-46	+364	-224	-297	+509	+125	-22	+4578	+0
Total	-3624	+16842	-62489	+8117	-5105	-8583	+17918	+5426	-5766	+149	+1734	+5940	+214	-275	-6735	+12308	+15136	+4513	-1516	+1023	+4773	+0

True labels

RGBD	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	-797	+208	-492	-179	-142	+150	+327	-500	-438	+0	+20	-486	+7	+31	-9	+0	+1073	+465	+141	-102	+723	+0
asphalt	+218	+12019	-8850	+0	-167	+0	-315	-19	-4306	-5	-75	+0	+0	-53	-1	+1345	+334	+6650	-6848	-10	+83	+0
brick	-2055	-44675	+49563	-8	-65	-451	+834	-1224	+2727	+0	+979	-18	+102	-495	+118	+2619	-4037	-1360	-1570	-424	-560	+0
cloud	+0	+0	+709	+157	-64	+9	-81	+174	+0	+0	+0	+0	+0	+0	-460	+0	-444	+0	+0	+0	+0	+0
concrete	+111	+79	+3302	+0	-4537	+0	+0	-226	+0	+0	+238	+0	-532	-57	-210	+0	+1576	+22	+328	+0	-94	+0
fabric	+2619	+2070	+993	+30	-122	-306	+13	-2661	+0	+0	+0	-46	-9	+0	+434	+0	-3758	+1152	-369	+0	-40	+0
foliage	-965	-1264	+961	-104	-599	+0	-9165	-192	+9054	+2	+87	-361	+0	+152	-3	+15	+305	+1102	+549	+120	+306	+0
glass	+246	-341	+176	+0	+618	-479	-99	-5604	-23	+0	+0	+1628	+487	+101	-7	-11	+1852	+18	-41	+0	+1479	+0
grass	-450	-1175	+3291	+0	-851	+0	-2406	-261	+3324	-61	+10	+91	-13	-43	+0	+1532	-3219	+1444	+303	-1175	-341	+0
gravel	+0	+194	-10	+0	+0	+0	-236	+0	+0	+0	+0	+0	+0	+0	+0	+0	+1	-134	+185	+0	+0	+0
iron	+0	+1867	-2232	+0	+0	+0	-211	-1	-46	+0	-354	-3	+0	-10	+0	+9	+42	-109	+1037	+13	-2	+0
living	-705	+75	-4902	+0	+11	+0	+889	+2276	-22	+0	+92	+4606	+0	-793	+277	+0	-234	-490	-1270	+0	+190	+0
other	+0	+0	+131	+0	+0	+0	-33	-656	+0	+0	+0	+0	+0	+0	+0	+0	+2	+6	+631	+0	-81	+0
plastic	-11	+0	-154	+439	-35	+0	+11	+63	+0	+0	+0	-222	+0	+783	+0	+0	-764	+181	-295	+0	+4	+0
sky	+0	+0	+451	-9879	+0	+0	+26	-14	+0	+0	+0	+0	+0	+0	+9164	+0	+24	+0	+228	+0	+0	+0
soil	-21	-2270	+1817	+0	+0	+0	-453	+0	+6287	+0	+0	-2	+0	-5	+0	-3339	-854	-576	-135	-143	-306	+0
steel	+3086	+162	+424	+142	+2879	+469	+182	+3429	+1503	+21	+61	-789	+309	-355	+865	+98	-15752	+2471	-760	+31	+1524	+0
stone	+321	-1320	-1608	+0	-12	+0	-211	-88	+2120	+0	+853	+231	+112	-1361	+0	+46	+577	-5062	+4945	+162	+295	+0
tile	-136	-819	+14611	+95	-276	+8	-6105	+1400	-3772	+39	+193	-226	+44	-114	+268	-788	+1425	-1232	-4663	+249	-201	+0
water	-322	+0	+403	+0	+0	+0	+932	-40	+333	+0	+425	+222	+0	+0	+0	-21	-1034	-87	+91	+21	-923	+0
wood	+1038	-609	+2108	+1	+280	-176	+1927	-1209	+810	+0	-19	+9	-49	-174	-826	+32	-1038	-438	-1024	-3	-640	+0
Total	+2177	-35799	+60692	-9306	-3082	-776	-14174	-5353	+17551	-4	+2510	+4634	+458	-2393	+9610	+1537	-23923	+4023	-8537	-1261	+1416	+0

Difference between the confusion matrices of the two runs.

G.5 learning rate 0.01 • all 21 classes • dice loss

	RGB	RGBD
mIoU	0.330	0.420
fIoU	0.554	0.631
accuracy	0.566	0.700
micro-averaged precision/recall/F1	0.566	0.700
macro-averaged precision	0.780	0.783
macro-averaged recall	0.387	0.474
macro-averaged F1	0.384	0.502
weighted macro-averaged precision	0.615	0.716
weighted macro-averaged recall	0.593	0.685
weighted macro-averaged F1	0.600	0.696

Performance overview

	Dice loss						Difference with cross entropy loss					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.330	0.780	0.387	0.420	0.783	0.474	-0.164	+0.144	-0.189	-0.087	+0.153	0.000
Weighted overall	0.554	0.615	0.593	0.631	0.716	0.685	-0.255	-0.264	-0.293	-0.171	-0.158	-0.197
`brick`	0.000	—	0.000	0.783	0.874	0.882	-0.832	—	-0.918	-0.037	-0.019	-0.027
`sky`	0.985	0.997	0.987	0.981	0.996	0.984	+0.003	-0.001	+0.003	+0.001	-0.002	+0.003
`asphalt`	0.823	0.889	0.917	0.000	—	0.000	-0.035	-0.018	-0.023	-0.832	—	-0.937
`tile`	0.484	0.796	0.553	0.550	0.828	0.621	-0.113	0.000	-0.151	-0.033	+0.021	-0.056
`foliage`	0.882	0.943	0.931	0.847	0.899	0.935	+0.056	+0.053	+0.011	-0.015	-0.015	-0.003
`cloud`	0.970	0.978	0.991	0.958	0.969	0.988	+0.007	+0.011	-0.005	0.000	+0.007	-0.007
`grass`	0.654	0.803	0.778	0.628	0.761	0.783	-0.061	+0.018	-0.111	-0.035	-0.005	-0.048
`glass`	0.720	0.813	0.862	0.739	0.849	0.851	+0.046	+0.060	-0.003	+0.037	+0.065	-0.019
`stone`	0.000	—	0.000	0.360	0.658	0.443	-0.340	—	-0.461	-0.009	+0.080	-0.062
`steel`	0.571	0.776	0.684	0.589	0.795	0.694	+0.002	+0.064	-0.055	-0.007	+0.062	-0.067
`concrete`	0.000	—	0.000	0.780	0.945	0.817	-0.824	—	-0.883	-0.039	+0.024	-0.064
`water`	0.000	—	0.000	0.416	0.990	0.418	-0.563	—	-0.632	-0.228	+0.198	-0.358
`wood`	0.243	0.870	0.253	0.269	0.927	0.275	-0.159	+0.187	-0.242	-0.204	+0.187	-0.293
`soil`	0.000	—	0.000	0.000	—	0.000	-0.265	—	-0.359	-0.191	—	-0.298
`other`	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000
`aluminium`	0.000	—	0.000	0.566	0.801	0.659	-0.412	—	-0.685	+0.134	+0.243	+0.002
`gravel`	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000
`iron`	0.112	0.578	0.122	0.064	0.293	0.075	+0.112	+0.576	+0.122	+0.060	+0.271	+0.070
`living`	0.494	0.910	0.519	0.284	0.944	0.289	+0.047	+0.131	+0.007	-0.257	+0.111	-0.318
`fabric`	0.000	—	0.000	0.000	—	0.000	-0.105	—	-0.108	-0.048	—	-0.051
`plastic`	0.000	0.000	0.528	0.000	0.000	0.242	-0.002	-0.004	+0.522	-0.134	-0.242	+0.010

Per-class results

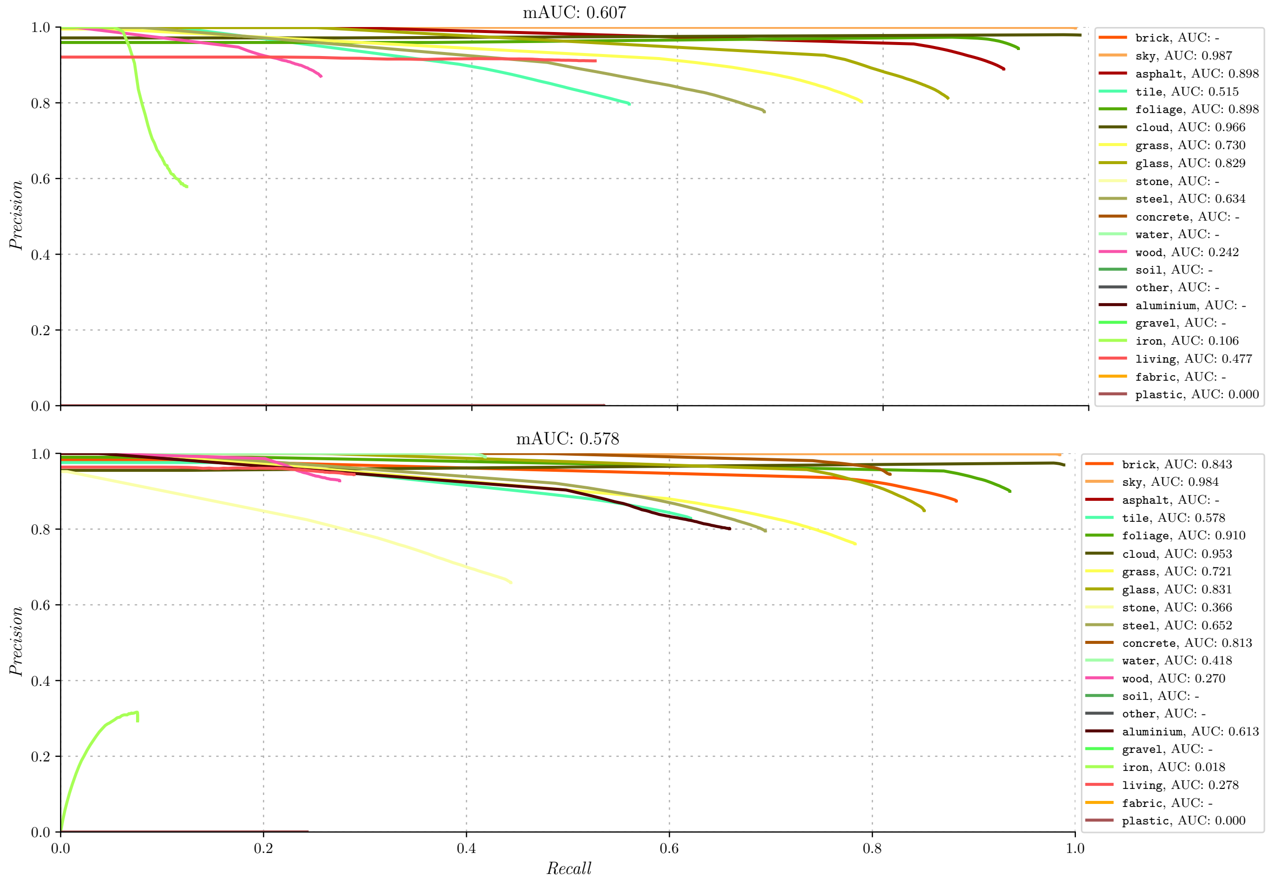
Predicted labels

True labels	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	0	4	0	0	0	0	939	264	24	0	0	8	0	33470	174	0	4582	0	0	0	22	39487
asphalt	0	1566542	0	0	0	0	201	0	3988	0	0	0	0	135639	0	0	8	0	1392	0	7	1707777
brick	0	136203	0	0	0	0	967	5122	1166	0	1300	4	0	2735901	0	0	5474	0	47909	0	86	2934132
cloud	0	0	0	659108	0	0	137	116	0	0	0	0	0	1931	3377	0	44	0	3	0	0	664716
concrete	0	0	0	0	0	0	0	0	0	0	0	0	0	90731	90	0	8729	0	1135	0	0	100685
fabric	0	0	0	0	0	0	34	6405	0	0	0	0	0	18168	16	0	2676	0	0	0	1	27300
foliage	0	5212	0	98	0	0	445035	183	6799	0	0	0	0	16073	12	0	3355	0	183	0	758	477708
glass	0	0	0	0	0	0	60	125702	0	0	0	1456	0	11453	0	0	5750	0	1306	0	0	145727
grass	0	3763	0	0	0	0	5054	183	190074	0	0	0	0	38127	0	0	3869	0	3020	0	15	244105
gravel	0	0	0	0	0	0	3	0	0	0	0	0	0	1049	0	0	12	0	0	0	0	1064
iron	0	3081	0	0	0	0	168	0	18	0	1784	0	0	8247	0	0	1	0	1310	0	0	14609
living	0	790	0	0	0	0	337	5038	157	0	0	16595	0	7044	0	0	1033	0	924	0	0	31918
other	0	12	0	0	0	0	38	0	0	0	0	0	0	1584	0	0	4	0	2812	0	40	4490
plastic	0	0	0	417	0	0	111	0	0	0	0	25	0	1808	0	0	798	0	262	0	0	3421
sky	0	0	0	13603	0	0	7681	0	0	0	0	0	0	2682	1852330	0	123	0	0	0	0	1876419
soil	0	15010	0	0	0	0	74	0	14728	0	0	11	0	16751	0	0	394	0	1243	0	581	48792
steel	0	997	0	50	0	0	1313	3365	1315	0	0	121	0	58094	74	0	148101	0	2484	0	444	216358
stone	0	16674	0	0	0	0	286	48	5025	0	0	0	0	168091	0	0	1064	0	37189	0	9	228386
tile	0	12894	0	0	0	0	5599	3577	9152	0	0	0	0	289840	0	0	1088	0	398961	0	172	721283
water	0	0	0	0	0	0	1650	0	2465	0	0	0	0	14668	0	0	12	0	0	0	22	18817
wood	0	74	0	0	0	0	1979	4520	1670	0	0	0	0	30226	0	0	3617	0	689	0	14494	57269
Total	0	1761256	0	673276	0	0	471666	154523	236581	0	3084	18220	0	3681577	1856073	0	190734	0	500822	0	16651	9564463

Predicted labels

True labels	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	26035	0	1534	0	381	0	584	464	89	0	0	47	0	3912	84	0	4909	8	1323	0	117	39487
asphalt	0	0	98022	0	0	0	596	98	6779	0	0	0	0	1594107	0	0	133	6882	1159	0	1	1707777
brick	263	0	2589952	0	539	0	3622	4195	2708	0	0	54	0	265779	0	0	7749	15118	43900	0	253	2934132
cloud	21	0	547	657201	0	0	184	0	0	0	0	0	0	1065	5649	0	24	0	25	0	0	664716
concrete	0	0	566	0	82306	0	0	465	0	0	0	0	0	10648	0	0	5031	0	1669	0	0	100685
fabric	3971	0	2135	7	0	0	17	4033	0	0	0	0	0	16604	41	0	492	0	0	0	0	27300
foliage	46	0	3246	153	111	0	446908	138	11305	0	0	97	0	10879	0	0	3905	27	269	44	580	477708
glass	196	0	8074	0	0	0	199	124028	0	0	0	0	0	7752	0	0	4838	11	629	0	0	145727
grass	7	0	7868	0	1415	0	12342	0	191139	0	0	0	0	25504	0	0	2344	2389	1095	0	2	244105
gravel	0	0	0	0	0	0	228	0	0	0	0	0	0	221	0	0	24	204	387	0	0	1064
iron	0	0	5757	0	0	0	365	0	77	0	1108	0	0	4279	0	0	1	1611	1411	0	0	14609
living	181	0	6894	0	0	0	462	5824	29	0	1	9230	0	5068	0	0	1882	2178	169	0	0	31918
other	0	0	268	0	0	0	0	0	0	0	0	0	0	159	0	0	13	0	4050	0	0	4490
plastic	153	0	437	145	0	0	95	0	0	0	0	64	0	831	0	0	862	5	829	0	0	3421
sky	0	0	147	20379	0	0	7703	0	0	0	0	0	0	403	1847700	0	87	0	0	0	0	1876419
soil	13	0	5319	0	0	0	407	0	16194	0	0	0	0	23219	0	0	502	1382	1671	0	85	48792
steel	1426	0	16345	4	2328	0	2590	2268	1769	0	498	285	0	32907	0	0	150287	1197	4329	0	125	216358
stone	35	0	54862	0	0	0	638	0	5345	0	2168	0	0	33854	0	0	1144	101345	28942	8	45	228386
tile	65	0	151678	0	0	0	12666	1534	13313	0	0	0	0	71342	4	0	1433	21292	447937	0	19	721283
water	46	0	337	0	0	0	2797	4	0	0	0	0	0	7589	0	0	140	31	0	7873	0	18817
wood	36	0	8183	0	0	0	4229	3025	2400	0	0	0	0	19340	0	0	3193	135	953	23	15752	57269
Total	32494	0	2962171	677889	87080	0	496632	146076	251147	0	3775	9777	0	2135462	1853478	0	188993	153815	540747	7948	16979	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.6 learning rate 0.01 • all 21 classes • class-weighted dice loss

	RGB	RGBD
mIoU	0.426	0.468
fIoU	0.766	0.784
accuracy	0.874	0.878
micro-averaged precision/recall/F1	0.874	0.878
macro-averaged precision	0.835	0.787
macro-averaged recall	0.480	0.524
macro-averaged F1	0.504	0.549
weighted macro-averaged precision	0.849	0.866
weighted macro-averaged recall	0.857	0.872
weighted macro-averaged F1	0.845	0.864

Performance overview

	Class-weighted dice loss						Difference with cross entropy loss					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.426	0.835	0.480	0.468	0.787	0.524	-0.068	+0.199	-0.096	-0.039	+0.157	0.000
Weighted overall	0.766	0.849	0.857	0.784	0.866	0.872	-0.043	-0.030	-0.029	-0.018	-0.008	-0.010
brick	0.764	0.817	0.921	0.769	0.842	0.899	-0.068	-0.082	+0.003	-0.051	-0.051	-0.010
sky	0.984	0.997	0.986	0.985	0.999	0.985	+0.002	-0.001	+0.002	+0.005	+0.001	+0.004
asphalt	0.814	0.870	0.926	0.791	0.843	0.928	-0.044	-0.037	-0.014	-0.041	-0.037	-0.009
tile	0.566	0.840	0.634	0.562	0.811	0.646	-0.031	+0.044	-0.070	-0.021	+0.004	-0.031
foliage	0.867	0.901	0.958	0.885	0.930	0.948	+0.041	+0.011	+0.038	+0.023	+0.016	+0.010
cloud	0.966	0.975	0.990	0.974	0.976	0.997	+0.003	+0.008	-0.006	+0.016	+0.014	+0.002
grass	0.677	0.773	0.844	0.712	0.815	0.849	-0.038	-0.012	-0.045	+0.049	+0.049	+0.018
glass	0.748	0.817	0.898	0.700	0.765	0.890	+0.074	+0.064	+0.033	-0.002	-0.019	+0.020
stone	0.321	0.631	0.395	0.364	0.654	0.450	-0.019	+0.068	-0.066	-0.005	+0.076	-0.055
steel	0.578	0.741	0.724	0.619	0.766	0.763	+0.009	+0.029	-0.015	+0.023	+0.033	+0.002
concrete	0.000	—	0.000	0.804	0.967	0.826	-0.824	—	-0.883	-0.015	+0.046	-0.055
water	0.231	0.978	0.233	0.681	0.903	0.734	-0.332	+0.141	-0.399	+0.037	+0.111	-0.042
wood	0.284	0.922	0.291	0.266	0.873	0.277	-0.118	+0.239	-0.204	-0.207	+0.133	-0.291
soil	0.010	0.444	0.010	0.101	0.458	0.115	-0.255	-0.057	-0.349	-0.090	+0.110	-0.183
other	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000
aluminium	0.552	0.766	0.665	0.514	0.810	0.584	+0.140	+0.257	-0.020	+0.082	+0.252	-0.073
gravel	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000	0.000	—	0.000
iron	0.000	—	0.000	0.110	0.961	0.111	0.000	—	0.000	+0.106	+0.939	+0.106
living	0.555	0.946	0.573	0.000	—	0.000	+0.108	+0.167	+0.061	-0.541	—	-0.607
fabric	0.029	0.946	0.029	0.000	—	0.000	-0.076	+0.103	-0.079	-0.048	—	-0.051
plastic	0.000	—	0.000	0.000	0.000	0.000	-0.002	—	-0.006	-0.134	-0.242	-0.232

Per-class results

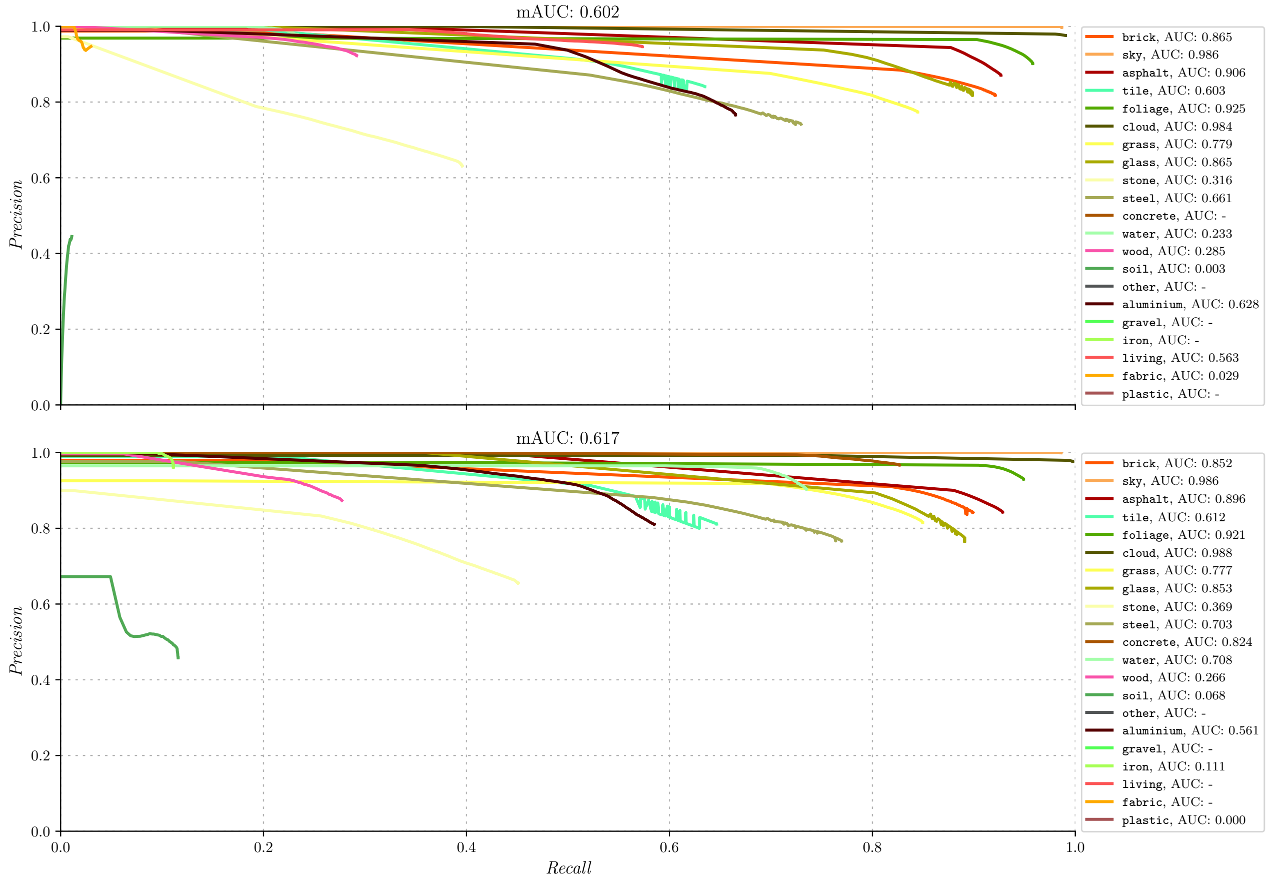
Predicted labels

True labels	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	26268	12	2740	0	0	46	2137	792	293	0	0	6	0	0	246	0	6255	78	505	0	109	39487
asphalt	0	1582766	104446	0	0	0	1639	0	6984	0	0	0	0	0	0	105	535	9981	1321	0	0	1707777
brick	36	165595	2702632	0	0	0	2861	3541	2041	0	0	14	0	0	376	0	7202	10449	39166	0	219	2934132
cloud	0	0	3719	658255	0	0	42	0	0	0	0	0	0	0	2669	0	31	0	0	0	0	664716
concrete	0	52	91535	0	0	0	4	419	0	0	0	0	0	0	584	0	8091	0	0	0	0	100685
fabric	4797	500	7982	15	0	817	45	7092	0	0	0	0	0	0	466	0	5551	0	0	0	35	27300
foliage	343	896	8335	71	0	0	457681	0	6679	0	0	0	0	0	65	0	2695	289	150	0	504	477708
glass	60	399	6936	0	0	0	394	130936	0	0	0	724	0	0	0	0	5590	44	644	0	0	145727
grass	0	2933	17600	0	0	0	6486	0	206236	0	0	0	0	0	0	0	4436	4730	1674	0	10	244105
gravel	0	0	855	0	0	0	209	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1064
iron	0	3928	7529	0	0	0	0	0	196	0	0	0	0	0	0	0	23	1400	1533	0	0	14609
living	489	495	6904	0	0	0	3	4146	72	0	0	18299	0	0	0	0	1418	18	74	0	0	31918
other	0	12	2478	0	0	0	482	258	25	0	0	0	0	0	0	0	10	0	1225	0	0	4490
plastic	155	25	1146	245	0	0	89	0	0	0	0	22	0	0	163	0	665	34	877	0	0	3421
sky	3	0	824	16066	0	0	7960	0	0	0	0	0	0	0	1851404	0	162	0	0	0	0	1876419
soil	0	14967	12497	0	0	0	136	0	16466	0	0	0	0	0	0	536	860	938	2060	0	332	48792
steel	1611	5147	36653	5	0	0	3540	5037	1918	0	0	272	0	0	57	0	156716	1486	3778	0	138	216358
stone	29	19033	78136	0	0	0	325	66	5178	0	0	0	0	0	0	564	2435	90351	32242	0	27	228386
tile	85	19898	184557	0	0	0	14358	3619	16161	0	0	0	0	0	30	0	1625	22926	457996	0	28	721283
water	374	779	3428	0	0	0	5063	0	2459	0	0	0	0	0	3	0	2111	197	14	4388	1	18817
wood	36	14	23032	0	0	0	4247	4230	1789	0	0	0	0	0	211	0	5009	191	1707	98	16705	57269
Total	34286	1817451	3303964	674657	0	863	507701	160136	266497	0	0	19337	0	0	1856274	1205	211420	143112	544966	4486	18108	9564463

Predicted labels

True labels	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	23098	0	5962	0	97	0	1912	735	204	0	0	0	0	0	138	0	6841	7	196	0	297	39487
asphalt	0	1585492	99867	0	0	0	330	0	6870	0	0	0	0	0	0	924	589	9593	4085	0	27	1707777
brick	98	203878	2638360	0	0	0	1225	6097	2608	0	65	0	0	0	45	4829	9383	8283	58482	612	167	2934132
cloud	0	0	983	663098	0	0	63	0	0	0	0	0	0	0	568	0	4	0	0	0	0	664716
concrete	16	254	8669	0	83233	0	135	107	1	0	0	0	0	0	163	0	6813	0	1294	0	0	100685
fabric	2245	4	4759	36	0	0	85	16250	0	0	0	0	0	0	186	0	3675	0	0	0	60	27300
foliage	298	655	9050	233	251	0	453344	867	6007	0	0	0	0	0	0	172	4832	70	895	171	863	477708
glass	208	484	9891	0	0	0	84	129838	124	0	0	0	0	0	0	0	4480	0	618	0	0	145727
grass	0	5185	18005	0	788	0	3953	31	207435	0	0	0	0	0	0	0	3157	3329	2209	0	13	244105
gravel	0	0	446	0	0	0	387	0	0	0	0	0	0	0	0	0	43	109	79	0	0	1064
iron	0	3528	6005	0	0	0	6	0	37	0	1623	0	0	0	0	0	3	2520	887	0	0	14609
living	239	532	28293	0	0	0	0	754	59	0	0	0	0	0	0	0	1229	457	355	0	0	31918
other	0	12	2329	0	0	0	38	39	0	0	0	0	0	0	0	0	21	0	2051	0	0	4490
plastic	169	19	1583	30	0	0	154	0	0	0	0	0	0	0	0	0	557	59	850	0	0	3421
sky	0	0	2891	15559	0	0	7716	0	0	0	0	0	0	0	1850112	0	139	0	2	0	0	1876419
soil	10	18368	9925	0	0	0	134	0	10427	0	0	0	0	0	0	5648	1173	1691	829	0	587	48792
steel	1235	2851	26105	65	1680	0	1819	4563	2472	0	0	0	0	17	10	0	165292	6017	4068	0	164	216358
stone	98	22315	65120	0	0	0	101	158	5630	0	0	0	0	0	0	754	750	102970	29977	513	0	228386
tile	501	36881	169687	0	0	0	9043	3189	11097	0	0	0	0	0	65	0	2345	21908	466481	17	69	721283
water	286	0	2219	0	0	0	172	0	36	0	0	0	0	0	0	0	2099	90	35	13830	50	18817
wood	0	121	22188	0	0	0	6751	6909	1410	0	0	0	0	0	0	0	2195	117	1546	159	15873	57269
Total	28501	1880579	3132337	679021	86049	0	487452	169537	254417	0	1688	0	0	17	1851287	12327	215620	157220	574939	15302	18170	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.7 learning rate 0.001 • all 21 classes • focal loss

	RGB	RGBD
mIoU	0.470	0.434
fIoU	0.782	0.771
accuracy	0.877	0.871
micro-averaged precision/recall/F1	0.877	0.871
macro-averaged precision	0.666	0.554
macro-averaged recall	0.555	0.517
macro-averaged F1	0.565	0.527
weighted macro-averaged precision	0.859	0.849
weighted macro-averaged recall	0.870	0.860
weighted macro-averaged F1	0.863	0.854

Performance overview

	Unweighted focal loss						Difference with unweighted cross entropy loss					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.470	0.666	0.555	0.434	0.554	0.517	-0.024	+0.030	-0.021	-0.073	-0.076	0.000
Weighted overall	0.782	0.859	0.870	0.771	0.849	0.860	-0.027	-0.020	-0.016	-0.031	-0.025	-0.022
brick	0.797	0.881	0.893	0.792	0.874	0.894	-0.035	-0.018	-0.025	-0.028	-0.019	-0.015
sky	0.978	0.998	0.980	0.980	0.996	0.983	-0.004	0.000	-0.004	0.000	-0.002	+0.002
asphalt	0.816	0.869	0.930	0.814	0.880	0.914	-0.042	-0.038	-0.010	-0.018	0.000	-0.023
tile	0.568	0.779	0.677	0.547	0.739	0.678	-0.029	-0.017	-0.027	-0.036	-0.068	+0.001
foliage	0.851	0.914	0.926	0.844	0.900	0.932	+0.025	+0.024	+0.006	-0.018	-0.014	-0.006
cloud	0.955	0.958	0.996	0.956	0.964	0.991	-0.008	-0.009	0.000	-0.002	+0.002	-0.004
grass	0.618	0.721	0.812	0.597	0.698	0.804	-0.097	-0.064	-0.077	-0.066	-0.068	-0.027
glass	0.668	0.750	0.860	0.659	0.739	0.859	-0.006	-0.003	-0.005	-0.043	-0.045	-0.011
stone	0.293	0.510	0.408	0.247	0.449	0.355	-0.047	-0.053	-0.053	-0.122	-0.129	-0.150
steel	0.529	0.662	0.725	0.525	0.668	0.711	-0.040	-0.050	-0.014	-0.071	-0.065	-0.050
concrete	0.740	0.846	0.855	0.664	0.783	0.813	-0.084	-0.078	-0.028	-0.155	-0.138	-0.068
water	0.672	0.747	0.869	0.358	0.604	0.468	+0.109	-0.090	+0.237	-0.286	-0.188	-0.308
wood	0.409	0.631	0.537	0.353	0.631	0.445	+0.007	-0.052	+0.042	-0.120	-0.109	-0.123
soil	0.186	0.375	0.270	0.134	0.314	0.189	-0.079	-0.126	-0.089	-0.057	-0.034	-0.109
other	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
aluminium	0.398	0.708	0.476	0.373	0.579	0.512	-0.014	+0.199	-0.209	-0.059	+0.021	-0.145
gravel	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
iron	0.000	0.000	0.000	0.001	0.032	0.002	0.000	-0.002	0.000	-0.003	+0.010	-0.003
living	0.293	0.689	0.337	0.281	0.784	0.304	-0.154	-0.090	-0.175	-0.260	-0.049	-0.303
fabric	0.106	0.617	0.114	0.000	0.004	0.000	+0.001	-0.226	+0.006	-0.048	-0.451	-0.051
plastic	0.000	0.000	0.000	0.000	0.000	0.000	-0.002	0.000	-0.006	-0.134	-0.242	-0.232

Per-class results

Predicted labels

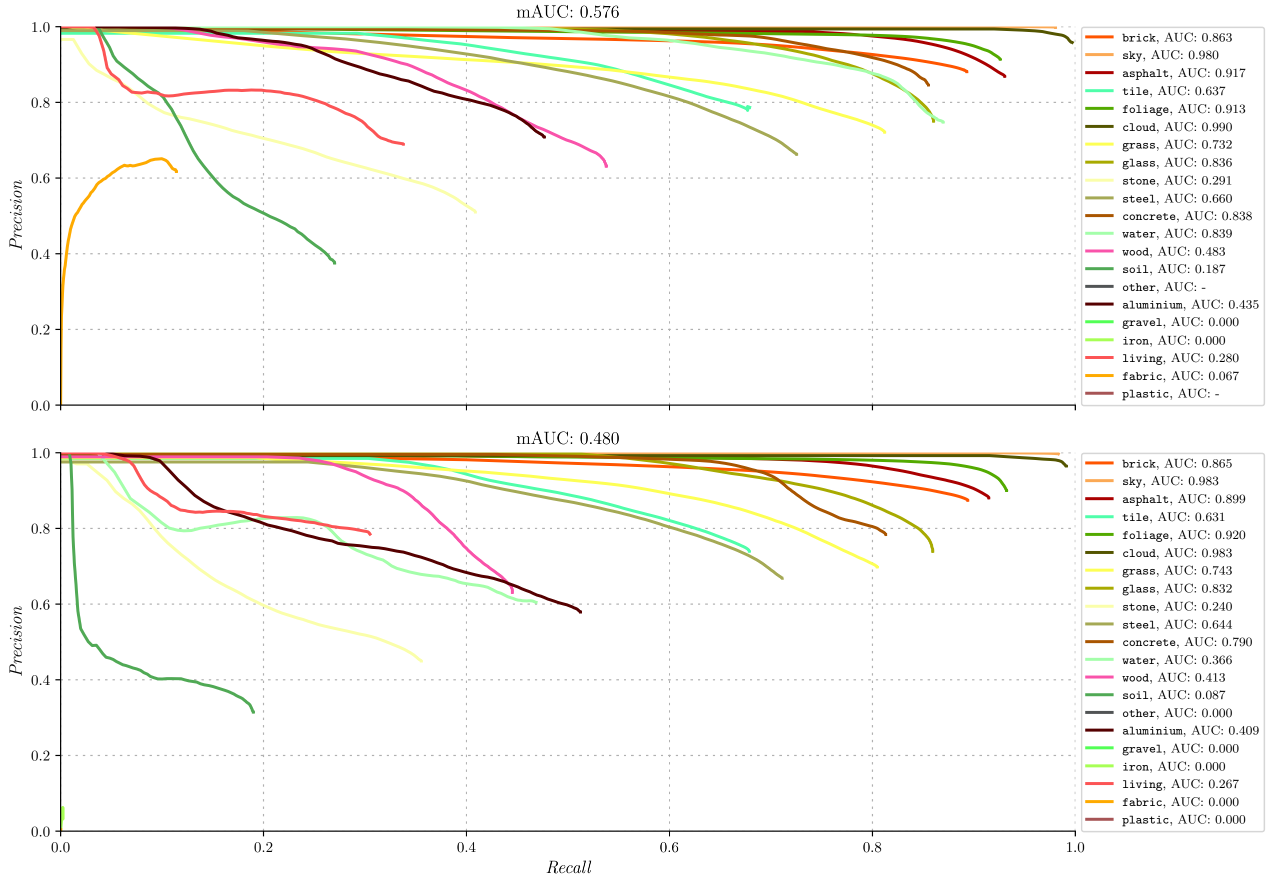
True labels

RGB	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	18818	17	2373	0	642	173	2847	1611	607	0	0	674	0	0	202	0	9611	7	410	61	1434	39487
asphalt	2	1588986	91290	0	2080	0	67	135	4906	0	12	0	0	0	0	2456	1369	11122	4390	930	32	1707777
brick	423	170703	2620797	97	5532	570	5565	15853	4049	79	0	839	0	0	540	2729	11401	28475	61607	1132	3741	2934132
cloud	0	0	7	662712	0	0	18	34	0	0	0	0	0	0	1431	0	400	0	114	0	0	664716
concrete	624	0	3017	0	86105	628	99	56	408	0	0	0	0	0	0	0	6317	5	3117	143	166	100685
fabric	2718	1643	492	64	287	3116	49	7396	0	0	0	4	0	0	316	0	5380	32	536	2	5265	27300
foliage	205	754	2110	197	583	6	442372	520	22797	0	0	376	0	0	17	575	3619	856	984	363	1374	477708
glass	1370	104	6242	0	221	402	69	125363	116	0	0	1258	0	0	91	0	8684	15	1502	0	290	145727
grass	0	5550	3223	0	560	0	10849	116	198250	0	105	13	0	0	0	1723	7187	4704	10165	883	777	244105
gravel	0	0	346	0	0	0	368	0	0	0	0	0	0	0	0	0	26	0	324	0	0	1064
iron	0	5376	5282	0	7	0	422	0	69	0	0	0	0	0	0	0	76	2171	1206	0	0	14609
living	113	871	5867	41	0	0	506	5408	141	0	0	10780	0	0	0	0	6131	647	1255	9	149	31918
other	5	12	1173	0	0	0	136	53	25	0	0	101	0	0	0	24	24	4	2866	0	67	4490
plastic	164	24	811	209	0	0	128	0	0	0	0	241	0	0	99	0	1447	47	216	0	35	3421
sky	0	0	451	27882	40	0	7886	12	0	0	0	0	0	0	1839754	0	9	0	385	0	0	1876419
soil	1	12307	4906	0	22	0	1169	0	12865	13	0	27	0	0	0	13180	1916	667	1404	0	315	48792
steel	1423	3776	18414	256	3451	109	3550	6572	4416	0	6	1117	0	0	484	47	156961	6243	5798	131	3604	216358
stone	415	21086	57209	0	221	0	536	54	8325	2	0	109	0	0	0	3487	1941	93308	40693	863	137	228386
tile	53	17181	142286	35	2004	42	2845	1506	16871	0	129	41	0	0	210	10689	4088	34059	488399	414	431	721283
water	16	0	736	0	10	0	594	88	0	0	0	0	0	0	0	0	295	319	227	16366	166	18817
wood	221	33	6967	0	0	0	3859	2290	1020	0	0	52	0	0	0	198	9892	183	1177	588	30789	57269
Total	26571	1828423	2973999	691493	101765	5046	483934	167067	274865	94	252	15632	0	0	1843144	35108	236774	182864	626775	21885	48772	9564463

True labels

RGBD	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	20240	11	2241	411	554	142	2407	3695	76	0	0	1164	1	0	47	0	6036	149	922	39	1352	39487
asphalt	5	1562243	106318	0	3469	0	556	90	11507	0	44	5	0	0	0	1085	1087	13476	7214	398	280	1707777
brick	1104	130787	2623321	0	1861	258	5599	14197	7461	0	482	206	822	179	136	3255	15596	28835	93318	2818	3897	2934132
cloud	7	0	508	658829	15	117	243	237	0	0	0	0	0	0	4719	0	0	0	41	0	0	664716
concrete	0	560	3538	0	81867	41	1284	876	468	0	0	0	0	5	0	1	10206	0	1805	34	0	100685
fabric	7903	3350	2504	42	0	3	51	8258	0	0	0	32	0	0	638	0	3270	0	297	0	952	27300
foliage	1460	411	3629	219	1130	10	445343	868	17213	6	0	50	0	0	5	94	3695	917	316	357	1985	477708
glass	849	75	6196	0	19	121	311	125257	14	0	0	653	47	0	0	0	9561	81	1807	0	736	145727
grass	0	4596	10706	0	295	0	6230	0	196446	36	0	0	0	0	0	4662	9189	4916	6108	363	558	244105
gravel	0	0	386	0	0	0	390	0	0	0	0	0	0	0	0	0	13	0	275	0	0	1064
iron	0	4932	6225	0	20	0	416	0	128	0	31	0	0	0	0	0	47	1701	1101	0	8	14609
living	817	1105	5546	0	0	0	1742	1492	63	0	369	9732	0	903	0	57	3045	4446	2426	5	170	31918
other	0	12	2042	0	0	0	372	25	0	0	0	0	0	0	0	0	44	34	1961	0	0	4490
plastic	123	8	639	391	0	0	253	0	0	0	0	134	0	0	0	0	808	264	780	21	0	3421
sky	1	0	113	22470	0	0	7730	0	0	0	0	0	0	4	1845034	0	98	0	969	0	0	1876419
soil	0	14903	3825	0	22	0	908	165	14275	0	0	85	0	0	0	9268	1389	1821	1127	0	1004	48792
steel	1532	4169	17247	365	8909	43	4569	6924	1869	0	29	258	49	16	31	126	153838	8865	5530	228	1761	216358
stone	208	22890	66409	0	13	0	504	40	6368	0	0	16	10	0	1	2797	1768	81187	45135	916	124	228386
tile	445	23111	126093	0	954	9	10225	1081	23832	18	0	41	0	0	1	7921	4050	33466	489491	258	287	721283
water	27	27	541	0	0	0	3561	698	684	46	0	10	0	0	0	0	2448	177	1	8816	1781	18817
wood	224	186	11784	0	5332	2	1995	5455	902	0	0	13	59	0	15	192	4038	223	1034	327	25488	57269
Total	34945	1773376	2999811	682727	104460	746	494689	169358	281306	106	955	12399	988	1107	1850627	29458	230226	180558	661658	14580	40383	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.8 learning rate 0.001 • all 21 classes • class-weighted focal loss

	RGB	RGBD
mIoU	0.447	0.445
fIoU	0.783	0.780
accuracy	0.878	0.876
micro-averaged precision/recall/F1	0.878	0.876
macro-averaged precision	0.558	0.581
macro-averaged recall	0.531	0.526
macro-averaged F1	0.534	0.531
weighted macro-averaged precision	0.857	0.855
weighted macro-averaged recall	0.870	0.869
weighted macro-averaged F1	0.862	0.860

Performance overview

	Class-weighted focal loss						Difference with unweighted cross entropy loss					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.447	0.558	0.531	0.445	0.581	0.526	-0.047	-0.078	-0.045	-0.062	-0.049	0.000
Weighted overall	0.783	0.857	0.870	0.780	0.855	0.869	-0.026	-0.022	-0.016	-0.022	-0.019	-0.013
brick	0.808	0.876	0.912	0.804	0.879	0.903	-0.024	-0.023	-0.006	-0.016	-0.014	-0.006
sky	0.981	0.998	0.982	0.977	0.995	0.981	-0.001	0.000	-0.002	-0.003	-0.003	0.000
asphalt	0.834	0.892	0.928	0.827	0.892	0.919	-0.024	-0.015	-0.012	-0.005	+0.012	-0.018
tile	0.561	0.774	0.670	0.534	0.730	0.666	-0.036	-0.022	-0.034	-0.049	-0.077	-0.011
foliage	0.811	0.887	0.905	0.853	0.904	0.938	-0.015	-0.003	-0.015	-0.009	-0.010	0.000
cloud	0.963	0.965	0.997	0.950	0.959	0.990	0.000	-0.002	+0.001	-0.008	-0.003	-0.005
grass	0.610	0.710	0.812	0.608	0.680	0.851	-0.105	-0.075	-0.077	-0.055	-0.086	+0.020
glass	0.661	0.738	0.863	0.626	0.685	0.879	-0.013	-0.015	-0.002	-0.076	-0.099	+0.009
stone	0.272	0.504	0.370	0.281	0.535	0.371	-0.068	-0.059	-0.091	-0.088	-0.043	-0.134
steel	0.510	0.647	0.706	0.520	0.658	0.712	-0.059	-0.065	-0.033	-0.076	-0.075	-0.049
concrete	0.713	0.867	0.801	0.747	0.877	0.834	-0.111	-0.057	-0.082	-0.072	-0.044	-0.047
water	0.672	0.739	0.881	0.557	0.728	0.703	+0.109	-0.098	+0.249	-0.087	-0.064	-0.073
wood	0.288	0.490	0.411	0.441	0.697	0.546	-0.114	-0.193	-0.084	-0.032	-0.043	-0.022
soil	0.134	0.322	0.187	0.104	0.285	0.142	-0.131	-0.179	-0.172	-0.087	-0.063	-0.156
other	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
aluminium	0.455	0.669	0.587	0.451	0.728	0.543	+0.043	+0.160	-0.098	+0.019	+0.170	-0.114
gravel	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
iron	0.000	0.000	0.000	0.000	0.000	0.000	0.000	-0.002	0.000	-0.004	-0.022	-0.005
living	0.124	0.630	0.134	0.070	0.976	0.071	-0.323	-0.149	-0.378	-0.471	+0.143	-0.536
fabric	0.000	0.014	0.000	0.000	0.000	0.000	-0.105	-0.829	-0.108	-0.048	-0.455	-0.051
plastic	0.000	0.000	0.000	0.000	0.000	0.000	-0.002	-0.004	-0.006	-0.134	-0.242	-0.232

Per-class results

Predicted labels

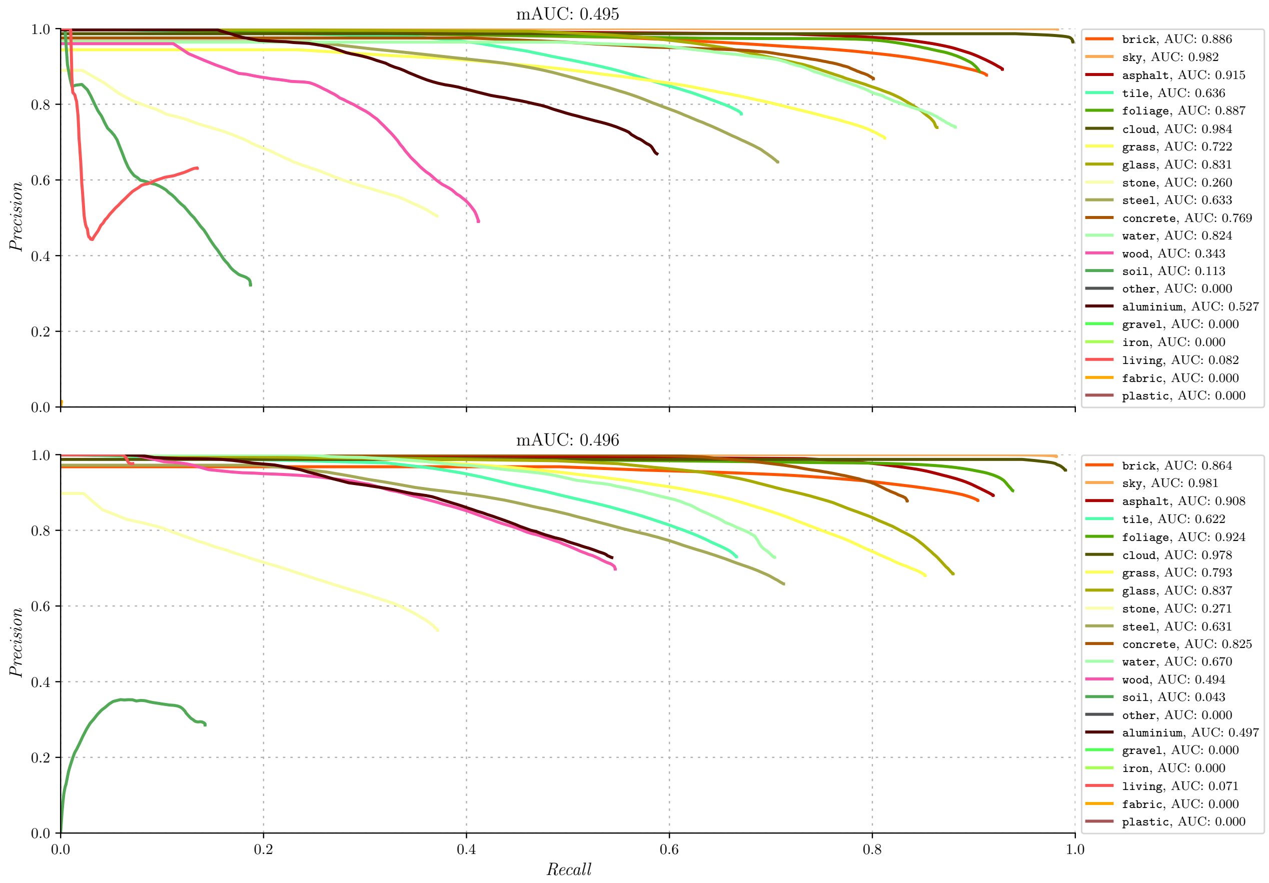
True labels

RGB	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	23207	12	1099	57	329	290	1710	1273	232	0	0	1252	0	62	73	0	8213	394	429	103	752	39487
asphalt	16	1585051	97546	0	116	0	529	12	4142	0	0	17	0	0	0	936	1288	12045	5367	358	354	1707777
brick	1073	112131	2677499	333	6499	17	2704	8556	4923	0	0	10	116	0	58	8119	12922	24187	66071	409	8505	2934132
cloud	112	0	528	663146	0	0	18	0	0	0	0	0	0	0	779	0	133	0	0	0	0	664716
concrete	114	0	6337	85	80654	382	316	288	417	0	0	0	0	0	3	219	8805	12	2620	0	433	100685
fabric	2324	714	1491	28	105	21	41	11805	0	0	0	125	689	0	1021	606	5371	1398	364	0	1197	27300
foliage	479	678	3356	170	456	0	432556	256	26448	0	0	38	0	8	3	238	8375	888	790	178	2791	477708
glass	1308	1	6075	0	500	3	225	125865	139	0	0	0	0	67	0	0	8410	127	1259	0	1748	145727
grass	3	6765	5438	0	144	7	10026	90	198275	0	0	86	0	0	0	3856	6277	6864	4106	1476	692	244105
gravel	0	0	429	0	0	0	158	0	11	0	0	0	0	0	0	0	33	126	307	0	0	1064
iron	0	3671	7221	0	0	0	135	0	34	0	0	1	0	0	0	15	61	1736	1711	0	24	14609
living	71	1	9536	0	0	132	361	7876	12	19	19	4299	45	413	0	135	5228	1077	844	16	1834	31918
other	0	12	4086	0	0	0	59	1	25	0	0	0	0	0	0	0	21	0	286	0	0	4490
plastic	87	161	635	133	0	0	219	95	0	0	0	248	0	0	0	0	877	24	828	113	1	3421
sky	456	0	50	22496	21	0	7954	273	0	0	0	0	0	0	1843561	0	355	0	1245	0	8	1876419
soil	0	13879	4424	0	37	0	842	0	15473	0	0	0	0	0	0	9130	949	1638	1244	249	927	48792
steel	2794	7478	19878	594	1698	518	3184	7644	2512	0	167	628	68	0	112	147	152910	3146	8524	653	3703	216358
stone	496	26804	57595	1	82	35	415	408	6053	0	55	62	44	0	0	3818	3165	84726	42761	963	903	228386
tile	1003	17566	140658	0	361	0	21458	1679	18930	0	0	0	0	24	295	1033	3697	29055	483873	1164	487	721283
water	323	24	600	0	0	0	438	5	245	0	0	0	0	0	0	0	197	40	186	16594	165	18817
wood	810	821	9293	0	1951	0	4275	4304	1118	0	0	47	0	0	0	63	8729	303	1815	162	23578	57269
Total	34676	1775769	3053774	687043	92953	1405	487623	170430	278989	19	241	6813	962	574	1845905	28315	236016	167786	624630	22438	48102	9564463

True labels

RGBD	aluminium	asphalt	brick	cloud	concrete	fabric	foliage	glass	grass	gravel	iron	living	other	plastic	sky	soil	steel	stone	tile	water	wood	Total
aluminium	21451	0	2743	11	411	0	3088	1359	725	0	0	0	0	0	20	0	7729	372	375	168	1035	39487
asphalt	0	1569494	107088	0	133	0	433	142	7504	0	0	0	0	0	0	3785	577	9424	9128	2	67	1707777
brick	754	114281	2652348	12	5081	15	5210	14538	6422	0	124	0	0	0	32	2482	15894	18518	93448	847	4126	2934132
cloud	0	0	548	658328	0	0	158	0	0	0	0	0	0	0	5401	0	139	0	142	0	0	664716
concrete	14	18	4070	327	84002	0	64	148	202	0	0	0	0	0	0	0	9389	110	2341	0	0	100685
fabric	1639	2837	1995	60	42	0	103	12241	0	0	0	0	0	0	19	0	7453	234	559	0	118	27300
foliage	185	266	3252	134	646	0	448236	353	15262	0	0	0	282	0	13	753	4406	780	1150	240	1750	477708
glass	1056	76	4352	0	1	0	60	128175	1	0	0	55	0	1	0	0	9639	31	1528	0	752	145727
grass	0	4703	5132	0	616	0	8566	0	207976	0	0	0	0	0	1	1844	4289	2555	6437	1669	317	244105
gravel	0	0	607	0	0	0	185	0	0	0	0	0	0	0	0	0	26	0	246	0	0	1064
iron	8	3562	6531	0	18	0	106	0	408	0	0	0	0	0	0	85	51	1966	1868	0	6	14609
living	765	1562	6150	0	0	0	598	6585	508	0	49	2270	0	0	0	86	7898	1294	2443	123	1587	31918
other	0	12	905	0	0	0	38	447	0	0	0	0	0	0	0	0	33	28	3027	0	0	4490
plastic	87	27	536	485	0	0	153	0	0	0	0	0	0	0	0	0	1154	96	883	0	0	3421
sky	3	0	38	26649	0	0	7776	21	0	0	0	0	0	0	1841223	0	1	0	708	0	0	1876419
soil	32	13575	2464	0	0	0	30	19	19037	0	0	0	4	0	0	6944	661	2371	3078	70	507	48792
steel	2547	3063	15512	340	4209	3	3457	12011	4233	0	0	0	0	0	441	332	154163	7811	6423	106	1707	216358
stone	141	18997	62392	0	15	0	288	402	11628	0	353	0	0	0	0	3356	2363	84879	41780	602	1190	228386
tile	371	25973	131232	0	193	0	13363	1442	29620	2	0	0	0	0	1921	4585	3759	27081	480463	898	380	721283
water	315	9	722	0	0	0	1237	850	939	0	0	0	0	0	0	0	965	509	0	13241	30	18817
wood	97	51	7563	0	309	0	2372	8300	1145	0	0	0	0	0	112	45	3584	332	1852	201	31306	57269
Total	29465	1758506	3016180	686346	95676	18	495521	187033	305610	2	526	2325	286	1	1849183	24297	234173	158391	657879	18167	44878	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.9 learning rate 0.01 • -5 small classes • cross entropy loss

	Cross entropy loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.642	0.781	0.748	0.644	0.782	0.755	+0.148	+0.145	+0.172	+0.137	+0.152	0.000
Weighted overall	0.820	0.891	0.895	0.817	0.888	0.895	+0.011	+0.012	+0.009	+0.015	+0.014	+0.013
brick	0.837	0.895	0.928	0.830	0.896	0.917	+0.005	-0.004	+0.010	+0.010	+0.003	+0.008
sky	0.975	0.996	0.979	0.980	0.997	0.982	-0.007	-0.002	-0.005	0.000	-0.001	+0.001
asphalt	0.853	0.906	0.936	0.859	0.905	0.944	-0.005	-0.001	-0.004	+0.027	+0.025	+0.007
tile	0.603	0.817	0.697	0.605	0.805	0.709	+0.006	+0.021	-0.007	+0.022	-0.002	+0.032
foliage	0.873	0.924	0.941	0.857	0.910	0.936	+0.047	+0.034	+0.021	-0.005	-0.004	-0.002
cloud	0.947	0.955	0.991	0.959	0.961	0.997	-0.016	-0.012	-0.005	+0.001	-0.001	+0.002
grass	0.751	0.846	0.869	0.694	0.813	0.825	+0.036	+0.061	-0.020	+0.031	+0.047	-0.006
glass	0.690	0.775	0.863	0.665	0.738	0.871	+0.016	+0.022	-0.002	-0.037	-0.046	+0.001
stone	0.351	0.568	0.479	0.354	0.588	0.471	+0.011	+0.005	+0.018	-0.015	+0.010	-0.034
steel	0.564	0.706	0.738	0.556	0.700	0.730	-0.005	-0.006	-0.001	-0.040	-0.033	-0.031
concrete	0.764	0.872	0.860	0.772	0.883	0.859	-0.060	-0.052	-0.023	-0.047	-0.038	-0.022
water	0.438	0.714	0.532	0.528	0.670	0.713	-0.125	-0.123	-0.100	-0.116	-0.122	-0.063
wood	0.491	0.730	0.600	0.390	0.700	0.468	+0.089	+0.047	+0.105	-0.083	-0.040	-0.100
soil	0.277	0.548	0.360	0.379	0.533	0.567	+0.012	+0.047	+0.001	+0.188	+0.185	+0.269
aluminium	0.399	0.516	0.636	0.440	0.627	0.596	-0.013	+0.007	-0.049	+0.008	+0.069	-0.061
living	0.465	0.739	0.557	0.445	0.791	0.504	+0.018	-0.040	+0.045	-0.096	-0.042	-0.103
background	0.067	0.204	0.090	0.065	0.190	0.090	—	—	—	—	—	—

(a) Omitted classes merged into background

	Cross entropy loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.658	0.784	0.769	0.626	0.768	0.731	+0.164	+0.148	+0.193	+0.119	+0.138	0.000
Weighted overall	0.824	0.891	0.900	0.813	0.883	0.894	+0.015	+0.012	+0.014	+0.011	+0.009	+0.012
brick	0.838	0.894	0.930	0.829	0.895	0.918	+0.006	-0.005	+0.012	+0.009	+0.002	+0.009
sky	0.983	0.998	0.984	0.981	0.999	0.982	+0.001	0.000	0.000	+0.001	+0.001	+0.001
asphalt	0.859	0.915	0.933	0.852	0.904	0.936	+0.001	+0.008	-0.007	+0.020	+0.024	-0.001
tile	0.591	0.809	0.687	0.616	0.796	0.731	-0.006	+0.013	-0.017	+0.033	-0.011	+0.054
foliage	0.871	0.913	0.950	0.858	0.906	0.941	+0.045	+0.023	+0.030	-0.004	-0.008	+0.003
cloud	0.965	0.969	0.996	0.964	0.965	0.998	+0.002	+0.002	0.000	+0.006	+0.003	+0.003
grass	0.716	0.809	0.862	0.647	0.748	0.828	+0.001	+0.024	-0.027	-0.016	-0.018	-0.003
glass	0.712	0.770	0.903	0.678	0.733	0.900	+0.038	+0.017	+0.038	-0.024	-0.051	+0.030
stone	0.364	0.571	0.502	0.344	0.584	0.456	+0.024	+0.008	+0.041	-0.025	+0.006	-0.049
steel	0.603	0.725	0.781	0.571	0.710	0.745	+0.034	+0.013	+0.042	-0.025	-0.023	-0.016
concrete	0.780	0.919	0.837	0.774	0.890	0.856	-0.044	-0.005	-0.046	-0.045	-0.031	-0.025
water	0.520	0.659	0.712	0.565	0.747	0.699	-0.043	-0.178	+0.080	-0.079	-0.045	-0.077
wood	0.377	0.646	0.475	0.379	0.689	0.458	-0.025	-0.037	-0.020	-0.094	-0.051	-0.110
soil	0.304	0.500	0.436	0.164	0.405	0.217	+0.039	-0.001	+0.077	-0.027	+0.057	-0.081
aluminium	0.498	0.731	0.609	0.405	0.566	0.588	+0.086	+0.222	-0.076	-0.027	+0.008	-0.069
living	0.543	0.708	0.699	0.389	0.754	0.445	+0.096	-0.071	+0.187	-0.152	-0.079	-0.162
background	0.000	0.000	0.000	0.000	0.000	0.000	—	—	—	—	—	—

(b) other renamed to background

	Cross entropy loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.666	0.790	0.774	0.655	0.788	0.762	+0.172	+0.154	+0.198	+0.148	+0.158	0.000
Weighted overall	0.827	0.893	0.902	0.819	0.888	0.898	+0.018	+0.014	+0.016	+0.017	+0.014	+0.016
brick	0.841	0.894	0.934	0.832	0.899	0.917	+0.009	-0.005	+0.016	+0.012	+0.006	+0.008
sky	0.983	0.999	0.983	0.986	0.998	0.987	+0.001	+0.001	-0.001	+0.006	0.000	+0.006
asphalt	0.864	0.918	0.936	0.840	0.896	0.931	+0.006	+0.011	-0.004	+0.008	+0.016	-0.006
tile	0.611	0.814	0.710	0.600	0.787	0.716	+0.014	+0.018	+0.006	+0.017	-0.020	+0.039
foliage	0.864	0.916	0.938	0.876	0.921	0.946	+0.038	+0.026	+0.018	+0.014	+0.007	+0.008
cloud	0.964	0.966	0.997	0.975	0.977	0.997	+0.001	-0.001	+0.001	+0.017	+0.015	+0.002
grass	0.705	0.793	0.864	0.673	0.755	0.859	-0.010	+0.008	-0.025	+0.010	-0.011	+0.028
glass	0.771	0.833	0.912	0.721	0.798	0.882	+0.097	+0.080	+0.047	+0.019	+0.014	+0.012
stone	0.356	0.577	0.482	0.366	0.611	0.477	+0.016	+0.014	+0.021	-0.003	+0.033	-0.028
steel	0.580	0.713	0.757	0.566	0.693	0.756	+0.011	+0.001	+0.018	-0.030	-0.040	-0.005
concrete	0.756	0.904	0.822	0.754	0.869	0.850	-0.068	-0.020	-0.061	-0.065	-0.052	-0.031
water	0.617	0.711	0.823	0.693	0.851	0.788	+0.054	-0.126	+0.191	+0.049	+0.059	+0.012
wood	0.424	0.748	0.495	0.403	0.678	0.499	+0.022	+0.065	0.000	-0.070	-0.062	-0.069
soil	0.235	0.457	0.327	0.302	0.543	0.404	-0.030	-0.044	-0.032	+0.111	+0.195	+0.106
aluminium	0.493	0.615	0.713	0.454	0.574	0.684	+0.081	+0.106	+0.028	+0.022	+0.016	+0.027
living	0.584	0.784	0.697	0.434	0.759	0.504	+0.137	+0.005	+0.185	-0.107	-0.074	-0.103

(c) No background class is used

Results of training without the five smallest classes, using class-weighted cross entropy loss. The omission of these classes and the generation of a background class is carried out in three different ways. When a background class is used, its performance is not recorded in the averaging metrics. The difference of the results of the corresponding classes when trained with all classes, and also using cross entropy loss, are in the right table for comparison. The rows are sorted by class size, except for background, which has been manually moved to the bottom.

G.10 learning rate 0.01 • -5 small classes • class-weighted cross entropy loss

	Class-weighted cross entropy loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.666	0.800	0.771	0.659	0.799	0.760	+0.172	+0.164	+0.195	+0.152	+0.169	0.000
Weighted overall	0.820	0.892	0.897	0.819	0.890	0.897	+0.011	+0.013	+0.011	+0.017	+0.016	+0.015
brick	0.827	0.895	0.916	0.828	0.894	0.917	-0.005	-0.004	-0.002	+0.008	+0.001	+0.008
sky	0.986	0.999	0.987	0.982	0.997	0.984	+0.004	+0.001	+0.003	+0.002	-0.001	+0.003
asphalt	0.845	0.894	0.939	0.856	0.905	0.940	-0.013	-0.013	-0.001	+0.024	+0.025	+0.003
tile	0.608	0.819	0.703	0.594	0.783	0.711	+0.011	+0.023	-0.001	+0.011	-0.024	+0.034
foliage	0.854	0.919	0.923	0.863	0.909	0.944	+0.028	+0.029	+0.003	+0.001	-0.005	+0.006
cloud	0.972	0.976	0.995	0.965	0.969	0.996	+0.009	+0.009	-0.001	+0.007	+0.007	+0.001
grass	0.691	0.794	0.843	0.703	0.808	0.844	-0.024	+0.009	-0.046	+0.040	+0.042	+0.013
glass	0.713	0.791	0.879	0.666	0.740	0.869	+0.039	+0.038	+0.014	-0.036	-0.044	-0.001
stone	0.358	0.602	0.470	0.351	0.634	0.440	+0.018	+0.039	+0.009	-0.018	+0.056	-0.065
steel	0.577	0.701	0.766	0.588	0.715	0.767	+0.008	-0.011	+0.027	-0.008	-0.018	+0.006
concrete	0.803	0.933	0.853	0.779	0.935	0.823	-0.021	+0.009	-0.030	-0.040	+0.014	-0.058
water	0.674	0.912	0.720	0.632	0.826	0.730	+0.111	+0.075	+0.088	-0.012	+0.034	-0.046
wood	0.428	0.679	0.536	0.504	0.714	0.631	+0.026	-0.004	+0.041	+0.031	-0.026	+0.063
soil	0.448	0.634	0.605	0.264	0.434	0.403	+0.183	+0.133	+0.246	+0.073	+0.086	+0.105
aluminium	0.398	0.526	0.622	0.536	0.720	0.678	-0.014	+0.017	-0.063	+0.104	+0.162	+0.021
living	0.470	0.723	0.573	0.438	0.808	0.489	+0.023	-0.056	+0.061	-0.103	-0.025	-0.118
background	0.051	0.144	0.074	0.025	0.094	0.034	—	—	—	—	—	—

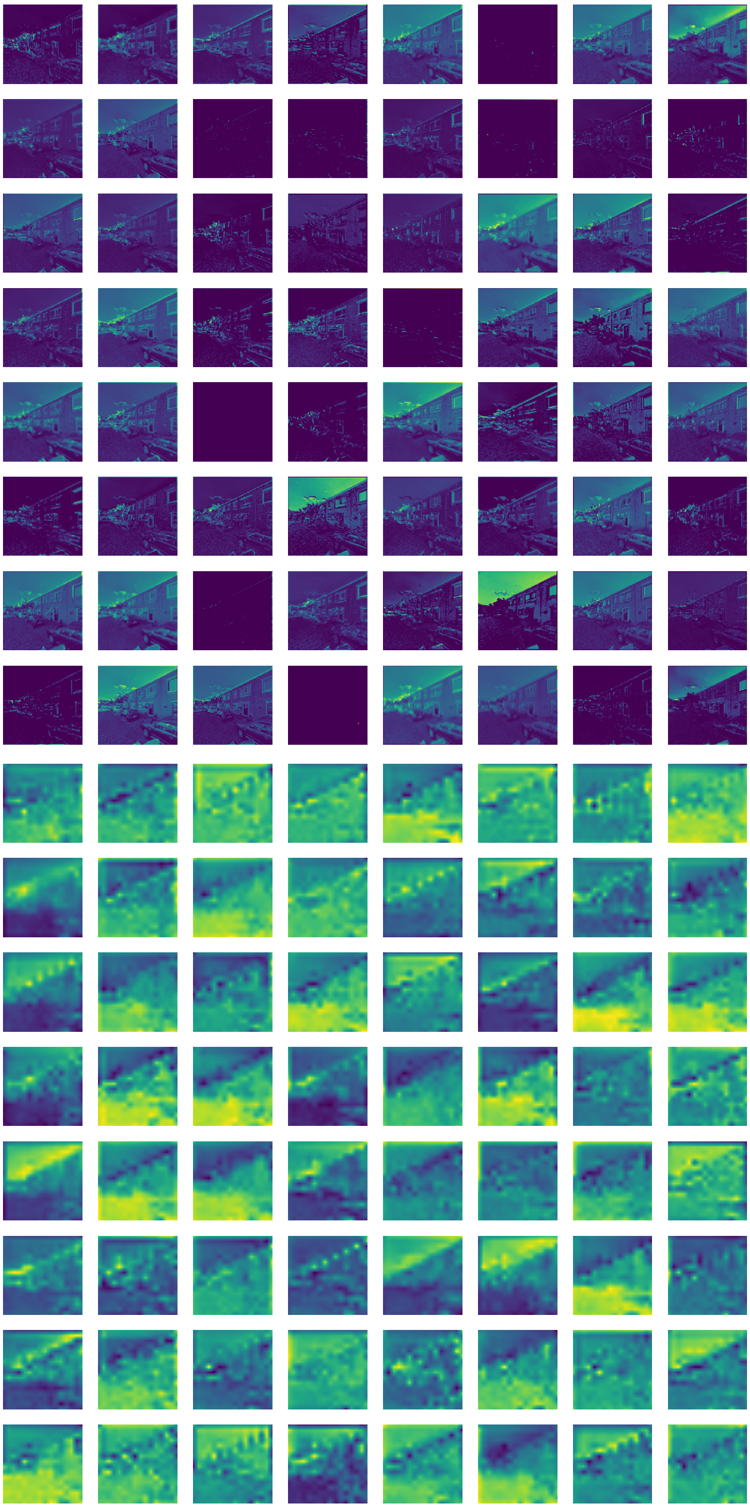
(a) Omitted classes merged into `background`

	Class-weighted cross entropy loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.649	0.791	0.754	0.646	0.782	0.758	+0.155	+0.155	+0.178	+0.139	+0.152	0.000
Weighted overall	0.816	0.889	0.894	0.810	0.883	0.892	+0.007	+0.010	+0.008	+0.008	+0.009	+0.010
brick	0.825	0.892	0.916	0.816	0.897	0.900	-0.007	-0.007	-0.002	-0.004	+0.004	-0.009
sky	0.982	1.000	0.983	0.976	0.997	0.979	0.000	+0.002	-0.001	-0.004	-0.001	-0.002
asphalt	0.845	0.895	0.938	0.844	0.890	0.942	-0.013	-0.012	-0.002	+0.012	+0.010	+0.005
tile	0.589	0.798	0.693	0.596	0.780	0.716	-0.008	+0.002	-0.011	+0.013	-0.027	+0.039
foliage	0.860	0.913	0.937	0.879	0.920	0.952	+0.034	+0.023	+0.017	+0.017	+0.006	+0.014
cloud	0.962	0.963	0.998	0.947	0.954	0.991	-0.001	-0.004	+0.002	-0.011	-0.008	-0.004
grass	0.725	0.821	0.862	0.665	0.761	0.840	+0.010	+0.036	-0.027	+0.002	-0.005	+0.009
glass	0.686	0.761	0.874	0.672	0.753	0.862	+0.012	+0.008	+0.009	-0.030	-0.031	-0.008
stone	0.364	0.580	0.494	0.358	0.592	0.476	+0.024	+0.017	+0.033	-0.011	+0.014	-0.029
steel	0.590	0.725	0.760	0.588	0.701	0.783	+0.021	+0.013	+0.021	-0.008	-0.032	+0.022
concrete	0.829	0.916	0.898	0.764	0.929	0.811	+0.005	-0.008	+0.015	-0.055	+0.008	-0.070
water	0.529	0.920	0.555	0.589	0.803	0.689	-0.034	+0.083	-0.077	-0.055	+0.011	-0.087
wood	0.353	0.687	0.421	0.471	0.720	0.576	-0.049	+0.004	-0.074	-0.002	-0.020	+0.008
soil	0.295	0.463	0.449	0.272	0.475	0.390	+0.030	-0.038	+0.090	+0.081	+0.127	+0.092
aluminium	0.450	0.563	0.691	0.423	0.529	0.678	+0.038	+0.054	+0.006	-0.009	-0.029	+0.021
living	0.500	0.767	0.590	0.480	0.814	0.539	+0.053	-0.012	+0.078	-0.061	-0.019	-0.068
background	0.000	0.000	0.000	0.000	0.000	0.000	—	—	—	—	—	—

(b) `other` renamed to `background`

	Class-weighted cross entropy loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.663	0.797	0.768	0.635	0.778	0.743	+0.169	+0.161	+0.192	+0.128	+0.148	0.000
Weighted overall	0.823	0.891	0.901	0.807	0.881	0.890	+0.014	+0.012	+0.015	+0.005	+0.007	+0.008
brick	0.835	0.894	0.927	0.818	0.894	0.906	+0.003	-0.005	+0.009	-0.002	+0.001	-0.003
sky	0.981	0.996	0.984	0.979	0.997	0.983	-0.001	-0.002	0.000	-0.001	-0.001	+0.002
asphalt	0.855	0.904	0.940	0.836	0.888	0.934	-0.003	-0.003	0.000	+0.004	+0.008	-0.003
tile	0.618	0.832	0.706	0.591	0.780	0.710	+0.021	+0.036	+0.002	+0.008	-0.027	+0.033
foliage	0.865	0.921	0.934	0.853	0.912	0.929	+0.039	+0.031	+0.014	-0.009	-0.002	-0.009
cloud	0.962	0.970	0.991	0.955	0.963	0.991	-0.001	+0.003	-0.005	-0.003	+0.001	-0.004
grass	0.691	0.769	0.871	0.664	0.756	0.845	-0.024	-0.016	-0.018	+0.001	-0.010	+0.014
glass	0.728	0.801	0.889	0.734	0.808	0.888	+0.054	+0.048	+0.024	+0.032	+0.024	+0.018
stone	0.353	0.586	0.470	0.359	0.576	0.488	+0.013	+0.023	+0.009	-0.010	-0.002	-0.017
steel	0.611	0.728	0.793	0.546	0.678	0.737	+0.042	+0.016	+0.054	-0.050	-0.055	-0.024
concrete	0.736	0.862	0.833	0.699	0.842	0.804	-0.088	-0.062	-0.050	-0.120	-0.079	-0.077
water	0.643	0.792	0.774	0.560	0.773	0.670	+0.080	-0.045	+0.142	-0.084	-0.019	-0.106
wood	0.523	0.734	0.646	0.441	0.731	0.527	+0.121	+0.051	+0.151	-0.032	-0.009	-0.041
soil	0.284	0.571	0.362	0.265	0.448	0.393	+0.019	+0.070	+0.003	+0.074	+0.100	+0.095
aluminium	0.488	0.665	0.647	0.508	0.654	0.696	+0.076	+0.156	-0.038	+0.076	+0.096	+0.039
living	0.432	0.721	0.518	0.346	0.745	0.393	-0.015	-0.058	+0.006	-0.195	-0.088	-0.214

(c) No `background` class is used



Activation maps when training on RGB with cross entropy loss and merging the five smallest classes. Only the first 64 channels of each layer are shown. Top: The XL bottom-up layer, bottom: the XL feature pyramid layer

G.11 learning rate 0.001 • -5 small classes • focal loss

	Focal loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.621	0.778	0.723	0.592	0.744	0.698	+0.127	+0.142	+0.147	+0.085	+0.114	0.000
Weighted overall	0.799	0.874	0.885	0.790	0.866	0.877	-0.010	-0.005	-0.001	-0.012	-0.008	-0.005
brick	0.809	0.883	0.906	0.803	0.886	0.895	-0.023	-0.016	-0.012	-0.017	-0.007	-0.014
sky	0.977	0.993	0.984	0.980	0.998	0.982	-0.005	-0.005	0.000	0.000	0.000	+0.001
asphalt	0.816	0.871	0.928	0.820	0.874	0.931	-0.042	-0.036	-0.012	-0.012	-0.006	-0.006
tile	0.595	0.787	0.709	0.572	0.766	0.693	-0.002	-0.009	+0.005	-0.011	-0.041	+0.016
foliage	0.875	0.935	0.931	0.846	0.902	0.931	+0.049	+0.045	+0.011	-0.016	-0.012	-0.007
cloud	0.953	0.968	0.984	0.957	0.961	0.995	-0.010	+0.001	-0.012	-0.001	-0.001	0.000
grass	0.651	0.737	0.848	0.638	0.733	0.832	-0.064	-0.048	-0.041	-0.025	-0.033	+0.001
glass	0.658	0.723	0.879	0.641	0.716	0.858	-0.016	-0.030	+0.014	-0.061	-0.068	-0.012
stone	0.266	0.514	0.355	0.282	0.504	0.391	-0.074	-0.049	-0.106	-0.087	-0.074	-0.114
steel	0.528	0.689	0.694	0.489	0.640	0.675	-0.041	-0.023	-0.045	-0.107	-0.093	-0.086
concrete	0.783	0.922	0.839	0.759	0.884	0.843	-0.041	-0.002	-0.044	-0.060	-0.037	-0.038
water	0.608	0.727	0.788	0.392	0.607	0.526	+0.045	-0.110	+0.156	-0.252	-0.185	-0.250
wood	0.354	0.642	0.441	0.375	0.612	0.492	-0.048	-0.041	-0.054	-0.098	-0.128	-0.076
soil	0.327	0.658	0.394	0.150	0.404	0.193	+0.062	+0.157	+0.035	-0.041	+0.056	-0.105
aluminium	0.425	0.662	0.542	0.445	0.672	0.568	+0.013	+0.153	-0.143	+0.013	+0.114	-0.089
living	0.312	0.733	0.352	0.326	0.739	0.368	-0.135	-0.046	-0.160	-0.215	-0.094	-0.239
background	0.086	0.367	0.101	0.036	0.162	0.044	—	—	—	—	—	—

(a) Omitted classes merged into background

	Focal loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.578	0.724	0.700	0.590	0.743	0.695	+0.084	+0.088	+0.124	+0.083	+0.113	0.000
Weighted overall	0.785	0.861	0.877	0.795	0.868	0.880	-0.024	-0.018	-0.009	-0.007	-0.006	-0.002
brick	0.802	0.874	0.906	0.804	0.886	0.896	-0.030	-0.025	-0.012	-0.016	-0.007	-0.013
sky	0.982	0.998	0.983	0.983	0.997	0.985	0.000	0.000	-0.001	+0.003	-0.001	+0.004
asphalt	0.800	0.870	0.908	0.829	0.879	0.935	-0.058	-0.037	-0.032	-0.003	-0.001	-0.002
tile	0.573	0.786	0.678	0.563	0.756	0.688	-0.024	-0.010	-0.026	-0.020	-0.051	+0.011
foliage	0.864	0.929	0.925	0.871	0.914	0.949	+0.038	+0.039	+0.005	+0.009	0.000	+0.011
cloud	0.961	0.965	0.996	0.965	0.972	0.992	-0.002	-0.002	0.000	+0.007	+0.010	-0.003
grass	0.611	0.685	0.849	0.651	0.745	0.838	-0.104	-0.100	-0.040	-0.012	-0.021	+0.007
glass	0.642	0.724	0.850	0.625	0.706	0.845	-0.032	-0.029	-0.015	-0.077	-0.078	-0.025
stone	0.255	0.505	0.340	0.277	0.489	0.389	-0.085	-0.058	-0.121	-0.092	-0.089	-0.116
steel	0.520	0.672	0.698	0.523	0.673	0.702	-0.049	-0.040	-0.041	-0.073	-0.060	-0.059
concrete	0.669	0.726	0.894	0.804	0.891	0.891	-0.155	-0.198	+0.011	-0.015	-0.030	+0.010
water	0.418	0.482	0.757	0.255	0.442	0.377	-0.145	-0.355	+0.125	-0.389	-0.350	-0.399
wood	0.414	0.764	0.475	0.378	0.620	0.491	+0.012	+0.081	-0.020	-0.095	-0.120	-0.077
soil	0.083	0.277	0.106	0.270	0.492	0.375	-0.182	-0.224	-0.253	+0.079	+0.144	+0.077
aluminium	0.419	0.605	0.577	0.426	0.696	0.524	+0.007	+0.096	-0.108	-0.006	+0.138	-0.133
living	0.238	0.716	0.263	0.219	0.733	0.238	-0.209	-0.063	-0.249	-0.322	-0.100	-0.369
background	0.000	0.000	0.000	0.000	—	0.000	—	—	—	—	—	—

(b) other renamed to background

	Focal loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.588	0.724	0.699	0.582	0.725	0.705	+0.094	+0.088	+0.123	+0.075	+0.095	0.000
Weighted overall	0.789	0.864	0.878	0.793	0.866	0.882	-0.020	-0.015	-0.008	-0.009	-0.008	0.000
brick	0.797	0.875	0.899	0.812	0.888	0.903	-0.035	-0.024	-0.019	-0.008	-0.005	-0.006
sky	0.982	0.997	0.985	0.979	0.996	0.982	-0.000	-0.001	+0.001	-0.001	-0.002	+0.001
asphalt	0.812	0.870	0.924	0.830	0.883	0.933	-0.046	-0.037	-0.016	-0.002	+0.003	-0.004
tile	0.541	0.744	0.666	0.560	0.762	0.679	-0.056	-0.052	-0.038	-0.023	-0.045	+0.002
foliage	0.856	0.913	0.932	0.856	0.911	0.933	+0.030	+0.023	+0.012	-0.006	-0.003	-0.005
cloud	0.960	0.968	0.992	0.957	0.962	0.995	-0.003	+0.001	-0.004	-0.001	0.000	0.000
grass	0.656	0.767	0.819	0.616	0.721	0.808	-0.059	-0.018	-0.070	-0.047	-0.045	-0.023
glass	0.629	0.702	0.857	0.626	0.694	0.863	-0.045	-0.051	-0.008	-0.076	-0.090	-0.007
stone	0.294	0.543	0.390	0.254	0.513	0.335	-0.046	-0.020	-0.071	-0.115	-0.065	-0.170
steel	0.506	0.671	0.672	0.507	0.631	0.720	-0.063	-0.041	-0.067	-0.089	-0.102	-0.041
concrete	0.819	0.902	0.899	0.732	0.820	0.872	-0.005	-0.022	+0.016	-0.087	-0.101	-0.009
water	0.559	0.651	0.798	0.470	0.540	0.784	-0.004	-0.186	+0.166	-0.174	-0.252	+0.008
wood	0.348	0.562	0.478	0.443	0.726	0.532	-0.054	-0.121	-0.017	-0.030	-0.014	-0.036
soil	0.081	0.184	0.126	0.219	0.402	0.326	-0.184	-0.317	-0.233	+0.028	+0.054	+0.028
aluminium	0.446	0.620	0.615	0.425	0.601	0.593	+0.034	+0.111	-0.070	-0.007	+0.043	-0.064
living	0.125	0.616	0.136	0.024	0.555	0.025	-0.322	-0.163	-0.376	-0.517	-0.278	-0.582

(c) No background class is used

G.12 learning rate 0.001 • -5 small classes • class-weighted focal loss

	Class-weighted focal loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.601	0.755	0.712	0.560	0.731	0.660	+0.107	+0.119	+0.136	+0.053	+0.101	0.000
Weighted overall	0.795	0.871	0.881	0.782	0.859	0.870	-0.014	-0.008	-0.005	-0.020	-0.015	-0.012
brick	0.808	0.882	0.906	0.803	0.870	0.912	-0.024	-0.017	-0.012	-0.017	-0.023	+0.003
sky	0.982	0.998	0.984	0.967	0.991	0.975	-0.000	0.000	-0.000	-0.013	-0.007	-0.006
asphalt	0.822	0.878	0.928	0.841	0.899	0.928	-0.036	-0.029	-0.012	+0.009	+0.019	-0.009
tile	0.572	0.773	0.688	0.521	0.754	0.628	-0.025	-0.023	-0.016	-0.062	-0.053	-0.049
foliage	0.849	0.926	0.911	0.849	0.897	0.940	+0.023	+0.036	-0.009	-0.013	-0.017	+0.002
cloud	0.963	0.966	0.997	0.924	0.943	0.978	0.000	-0.001	+0.001	-0.034	-0.019	-0.017
grass	0.620	0.721	0.816	0.607	0.706	0.813	-0.095	-0.064	-0.073	-0.056	-0.060	-0.018
glass	0.647	0.715	0.872	0.664	0.744	0.861	-0.027	-0.038	+0.007	-0.038	-0.040	-0.009
stone	0.267	0.504	0.363	0.265	0.484	0.369	-0.073	-0.059	-0.098	-0.104	-0.094	-0.136
steel	0.503	0.666	0.673	0.536	0.669	0.729	-0.066	-0.046	-0.066	-0.060	-0.064	-0.032
concrete	0.823	0.911	0.895	0.736	0.858	0.837	-0.001	-0.013	+0.012	-0.083	-0.063	-0.044
water	0.484	0.675	0.631	0.120	0.261	0.182	-0.079	-0.162	-0.001	-0.524	-0.531	-0.594
wood	0.356	0.597	0.469	0.345	0.606	0.445	-0.046	-0.086	-0.026	-0.128	-0.134	-0.123
soil	0.403	0.619	0.536	0.180	0.501	0.220	+0.138	+0.118	+0.177	-0.011	+0.153	-0.078
aluminium	0.398	0.543	0.599	0.428	0.654	0.553	-0.014	+0.034	-0.086	-0.004	+0.096	-0.104
living	0.117	0.714	0.122	0.177	0.865	0.182	-0.330	-0.065	-0.390	-0.364	+0.032	-0.425
background	0.020	0.107	0.024	0.024	0.110	0.030	—	—	—	—	—	—

(a) Omitted classes merged into `background`

	Class-weighted focal loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.582	0.734	0.681	0.602	0.766	0.710	+0.088	+0.098	+0.105	+0.095	+0.136	0.000
Weighted overall	0.788	0.865	0.876	0.791	0.867	0.878	-0.021	-0.014	-0.010	-0.011	-0.007	-0.004
brick	0.813	0.876	0.919	0.802	0.879	0.901	-0.019	-0.023	+0.001	-0.018	-0.014	-0.008
sky	0.962	0.995	0.966	0.984	0.996	0.988	-0.020	-0.003	-0.018	+0.004	-0.002	+0.007
asphalt	0.833	0.889	0.930	0.836	0.892	0.929	-0.025	-0.018	-0.010	+0.004	+0.012	-0.008
tile	0.556	0.761	0.674	0.547	0.744	0.673	-0.041	-0.035	-0.030	-0.036	-0.063	-0.004
foliage	0.846	0.907	0.927	0.830	0.885	0.930	+0.020	+0.017	+0.007	-0.032	-0.029	-0.008
cloud	0.913	0.923	0.988	0.972	0.979	0.992	-0.050	-0.044	-0.008	+0.014	+0.017	-0.003
grass	0.631	0.753	0.797	0.588	0.726	0.755	-0.084	-0.032	-0.092	-0.075	-0.040	-0.076
glass	0.650	0.720	0.870	0.576	0.642	0.848	-0.024	-0.033	+0.005	-0.126	-0.142	-0.022
stone	0.246	0.475	0.338	0.260	0.476	0.364	-0.094	-0.088	-0.123	-0.109	-0.102	-0.141
steel	0.515	0.673	0.687	0.534	0.667	0.727	-0.054	-0.039	-0.052	-0.062	-0.066	-0.034
concrete	0.799	0.918	0.861	0.812	0.928	0.866	-0.025	-0.006	-0.022	-0.007	+0.007	-0.015
water	0.463	0.662	0.607	0.548	0.646	0.783	-0.100	-0.175	-0.025	-0.096	-0.146	+0.007
wood	0.386	0.635	0.496	0.405	0.675	0.503	-0.016	-0.048	+0.001	-0.068	-0.065	-0.065
soil	0.081	0.231	0.111	0.254	0.489	0.346	-0.184	-0.270	-0.248	+0.063	+0.141	+0.048
aluminium	0.411	0.697	0.500	0.441	0.751	0.516	-0.001	+0.188	-0.185	+0.009	+0.193	-0.141
living	0.204	0.634	0.231	0.238	0.878	0.246	-0.243	-0.145	-0.281	-0.303	+0.045	-0.361
background	0.000	0.000	0.000	0.000	0.000	0.000	—	—	—	—	—	—

(b) `other` renamed to `background`

	Class-weighted focal loss, without five small classes						Difference with cross entropy loss, all classes					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.615	0.763	0.719	0.600	0.761	0.709	+0.121	+0.127	+0.143	+0.093	+0.131	0.000
Weighted overall	0.802	0.877	0.885	0.789	0.865	0.878	-0.007	-0.002	-0.001	-0.013	-0.009	-0.004
brick	0.819	0.890	0.911	0.804	0.884	0.899	-0.013	-0.009	-0.007	-0.016	-0.009	-0.010
sky	0.985	0.997	0.987	0.980	0.998	0.982	+0.003	-0.001	+0.003	0.000	0.000	+0.001
asphalt	0.821	0.875	0.930	0.831	0.884	0.933	-0.037	-0.032	-0.010	-0.001	+0.004	-0.004
tile	0.585	0.783	0.698	0.555	0.757	0.675	-0.012	-0.013	-0.006	-0.028	-0.050	-0.002
foliage	0.846	0.899	0.934	0.812	0.866	0.928	+0.020	+0.009	+0.014	-0.050	-0.048	-0.010
cloud	0.968	0.974	0.994	0.961	0.965	0.995	+0.005	+0.007	-0.002	+0.003	+0.003	0.000
grass	0.662	0.788	0.804	0.582	0.705	0.770	-0.053	+0.003	-0.085	-0.081	-0.061	-0.061
glass	0.691	0.770	0.870	0.640	0.694	0.891	+0.017	+0.017	+0.005	-0.062	-0.090	+0.021
stone	0.306	0.525	0.423	0.295	0.530	0.399	-0.034	-0.038	-0.038	-0.074	-0.048	-0.106
steel	0.554	0.695	0.732	0.528	0.676	0.707	-0.015	-0.017	-0.007	-0.068	-0.057	-0.054
concrete	0.736	0.864	0.832	0.770	0.842	0.900	-0.088	-0.060	-0.051	-0.049	-0.079	+0.019
water	0.516	0.736	0.633	0.533	0.703	0.688	-0.047	-0.101	+0.001	-0.111	-0.089	-0.088
wood	0.396	0.695	0.480	0.329	0.574	0.435	-0.006	+0.012	-0.015	-0.144	-0.166	-0.133
soil	0.197	0.378	0.292	0.186	0.463	0.238	-0.068	-0.123	-0.067	-0.005	+0.115	-0.060
aluminium	0.455	0.604	0.648	0.460	0.706	0.568	+0.043	+0.095	-0.037	+0.028	+0.148	-0.089
living	0.305	0.737	0.343	0.334	0.933	0.342	-0.142	-0.042	-0.169	-0.207	+0.100	-0.265

(c) No `background` class is used

G.13 learning rate 0.01 • -5 small classes • class-weighted cross entropy loss • only depth • reflect init.

	D
mIoU	0.399
fIoU	0.529
accuracy	0.685
micro-averaged precision/recall/F1	0.682
macro-averaged precision	0.546
macro-averaged recall	0.560
macro-averaged F1	0.542
weighted macro-averaged precision	0.674
weighted macro-averaged recall	0.685
weighted macro-averaged F1	0.677

Overall results

	Reflection initialized		
	D		
	IoU	Precision	Recall
Overall	0.399	0.547	0.561
Weighted overall	0.529	0.674	0.685
brick	0.527	0.689	0.691
sky	0.770	0.867	0.873
asphalt	0.464	0.619	0.650
tile	0.306	0.508	0.435
foliage	0.770	0.839	0.903
cloud	0.456	0.641	0.612
grass	0.314	0.488	0.469
glass	0.451	0.531	0.751
stone	0.328	0.535	0.458
steel	0.398	0.589	0.551
concrete	0.471	0.524	0.822
water	0.484	0.560	0.781
wood	0.303	0.566	0.395
soil	0.191	0.426	0.256
aluminium	0.136	0.199	0.302
living	0.021	0.167	0.024
background	0.027	0.107	0.034

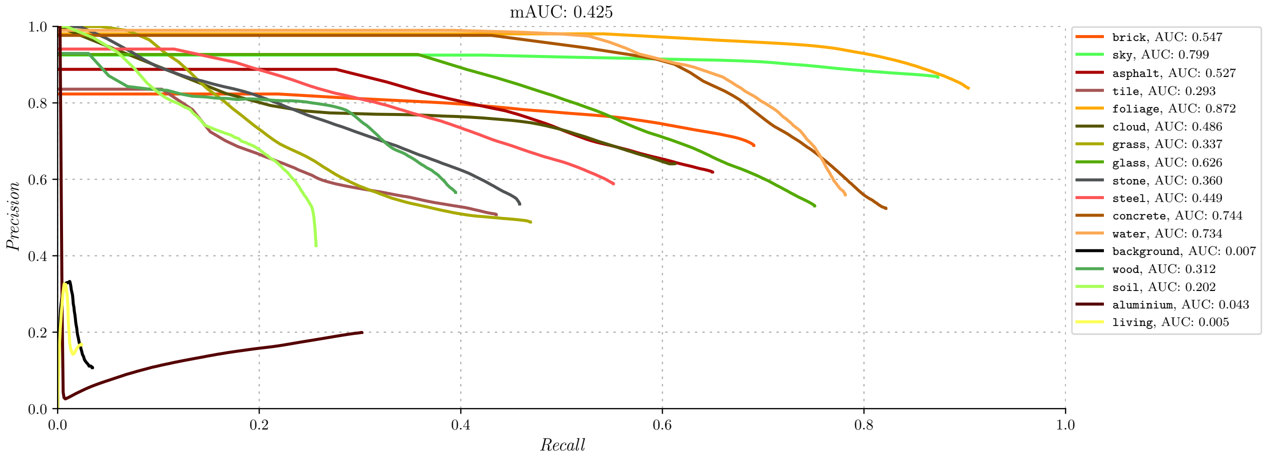
Per-class results

Predicted labels

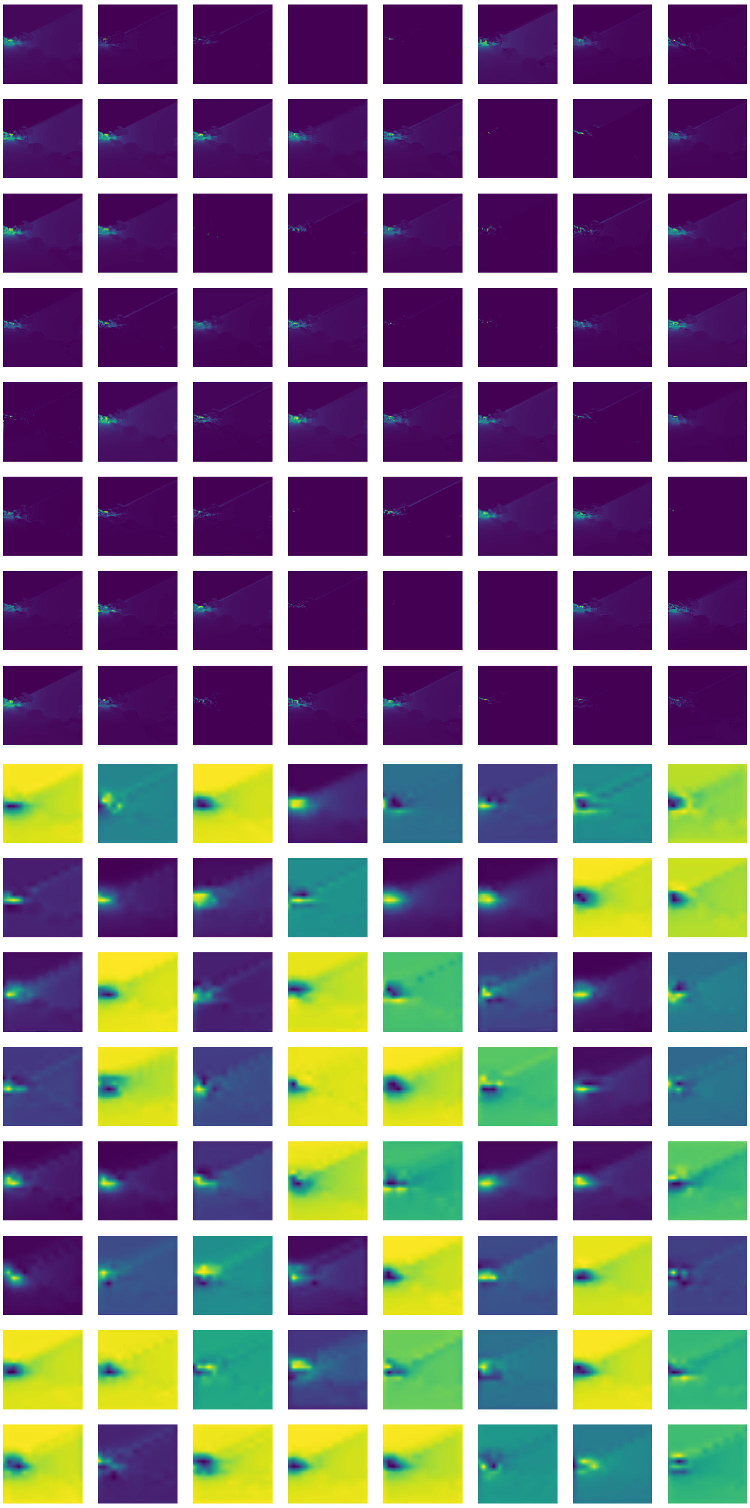
True labels

	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	1752	510	4612	16267	0	649	1640	9396	350	90	564	0	6984	2693	5307	12	58	50884
aluminium	1472	11916	17	2995	225	537	5095	4493	247	698	224	0	5632	4	70	15	5847	39487
asphalt	1056	0	1109582	465900	0	18902	863	14	23770	123	3	343	1185	9907	76012	50	67	1707777
brick	4177	42288	546109	2026794	0	25669	23606	53561	7371	793	1816	2689	18246	18871	159052	357	2733	2934132
cloud	211	5	0	432	407010	0	12666	2968	0	0	241241	0	90	41	52	0	0	664716
concrete	547	165	7	8805	644	82741	838	2286	17	0	0	0	3097	31	1334	173	0	100685
foliage	1301	1565	178	5022	298	229	431596	8064	6592	19	55	200	15556	1319	2672	1255	1787	477708
glass	321	0	8138	13593	0	1094	2296	109444	703	31	0	0	7498	15	880	205	1509	145727
grass	280	41	17376	32246	0	7681	4912	312	114514	57	31	4020	9714	18308	29624	4782	207	244105
living	403	302	47	16596	0	509	3205	4530	550	751	0	0	3037	593	833	197	365	31918
sky	0	0	0	806	226738	101	9343	0	0	0	1638589	0	556	5	274	0	7	1876419
soil	1	0	1219	1502	0	0	889	0	26054	173	0	12509	429	2545	2920	200	351	48792
steel	2853	2049	1133	28699	165	11187	9932	7760	10907	304	1994	4946	119297	4162	4464	2375	4131	216358
stone	1404	38	7177	70314	0	0	2819	713	12667	89	21	3268	3674	104682	20289	1147	84	228386
tile	377	139	96669	227136	42	7145	932	1110	30336	486	4781	1261	3911	32198	313881	741	138	721283
water	0	86	47	133	0	0	2751	60	270	365	0	0	265	0	75	14704	61	18817
wood	183	680	36	24535	0	1378	1110	1503	266	523	0	98	3507	146	634	55	22615	57269
Total	16338	59784	1792347	2941775	635122	157822	514493	206214	234614	4502	1889319	29334	202678	195520	618373	26268	39960	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size.



Activation maps when training only on depth. Only the first 64 channels of each layer are shown. Top: The XL bottom-up layer, bottom: the XL feature pyramid layer

G.14 learning rate 0.01 • -5 small classes • class-weighted cross entropy loss • only depth • Avg RGB init.

	RGB
mIoU	0.423
fIoU	0.553
accuracy	0.703
micro-averaged precision/recall/F1	0.702
macro-averaged precision	0.558
macro-averaged recall	0.598
macro-averaged F1	0.564
weighted macro-averaged precision	0.693
weighted macro-averaged recall	0.708
weighted macro-averaged F1	0.698

Overall results

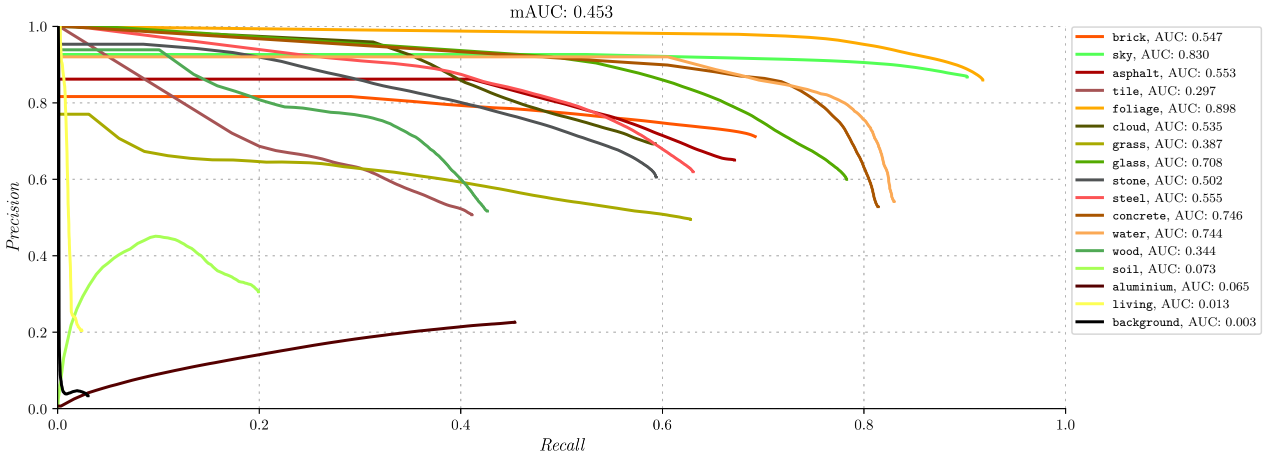
	Initialization by averaging RGB		
	D		
	IoU	Precision	Recall
Overall	0.423	0.559	0.598
Weighted overall	0.553	0.694	0.709
brick	0.540	0.712	0.692
sky	0.793	0.868	0.902
asphalt	0.494	0.651	0.672
tile	0.294	0.508	0.411
foliage	0.799	0.860	0.918
cloud	0.470	0.693	0.593
grass	0.383	0.495	0.628
glass	0.514	0.600	0.783
stone	0.429	0.606	0.594
steel	0.455	0.620	0.630
concrete	0.472	0.529	0.814
water	0.488	0.542	0.830
wood	0.305	0.517	0.426
soil	0.137	0.306	0.199
aluminium	0.178	0.226	0.454
living	0.022	0.205	0.024
background	0.016	0.034	0.030

Per-class results

Predicted labels

True labels	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	1536	1601	4385	20004	0	584	1579	7480	2750	311	178	197	3629	3303	2329	50	968	50884
aluminium	1278	17914	0	1092	0	529	2522	2999	154	284	160	0	6501	103	1093	268	4590	39487
asphalt	20320	14	1146896	434456	0	18473	840	543	19854	72	9	61	1835	10154	51237	2990	23	1707777
brick	8195	54657	487777	2030477	6	24888	13023	44548	24281	292	2452	9058	15051	23119	188350	1844	6114	2934132
cloud	1121	0	0	665	394392	2	16492	58	0	0	251129	0	660	0	197	0	0	664716
concrete	3	5	18	9133	0	81965	482	411	50	619	647	0	6221	487	570	74	0	100685
foliage	831	1521	155	3849	285	747	438528	949	6085	51	711	54	13689	988	2873	1622	4770	477708
glass	778	62	8092	9831	195	5	1348	114068	379	185	164	0	9305	107	1187	0	21	145727
grass	4144	19	16987	13350	0	4170	3944	105	153266	7	0	6553	8981	13253	17252	1647	427	244105
living	191	188	177	13812	0	32	1858	5851	374	764	0	0	4425	1116	522	259	2349	31918
sky	244	80	0	100	174111	299	7755	0	0	0	1693120	0	630	0	0	0	80	1876419
soil	168	0	3448	717	0	0	2315	0	26465	71	0	9726	1195	2128	2033	221	305	48792
steel	1424	2543	1001	23494	208	9658	8862	7642	8307	285	2132	456	136402	4176	5653	1351	2764	216358
stone	2015	89	17441	32849	0	0	2303	132	15615	123	3	3041	4484	135613	13698	854	126	228386
tile	1377	135	76548	233790	0	13713	5263	4664	51288	666	18	2624	3744	29058	296481	1740	174	721283
water	1207	8	0	110	0	0	1481	0	31	5	0	0	250	0	23	15616	86	18817
wood	487	378	76	25444	0	0	1209	751	556	0	0	8	2980	60	642	259	24419	57269
Total	45319	79214	1763001	2853173	569197	155065	509804	190201	309455	3735	1950723	31778	219982	223665	584140	28795	47216	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size.

G.15 learning rate 0.01 • -5 small classes • class-weighted cross entropy loss • only depth • Xavier init.

	D
mIoU	0.403
fIoU	0.536
accuracy	0.691
micro-averaged precision/recall/F1	0.688
macro-averaged precision	0.549
macro-averaged recall	0.563
macro-averaged F1	0.544
weighted macro-averaged precision	0.677
weighted macro-averaged recall	0.694
weighted macro-averaged F1	0.682

Overall results

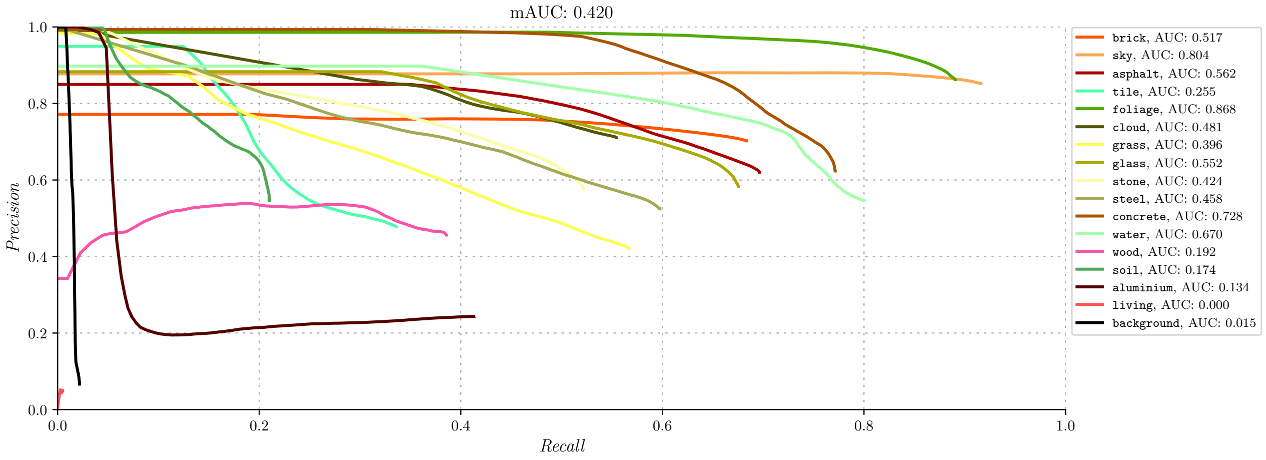
	Xavier initialized		
	D		
	IoU	Precision	Recall
Overall	0.404	0.550	0.564
Weighted overall	0.537	0.678	0.695
brick	0.530	0.703	0.684
sky	0.790	0.852	0.916
asphalt	0.488	0.621	0.696
tile	0.246	0.478	0.336
foliage	0.780	0.863	0.891
cloud	0.452	0.711	0.554
grass	0.319	0.422	0.567
glass	0.455	0.583	0.675
stone	0.377	0.576	0.521
steel	0.388	0.525	0.598
concrete	0.526	0.623	0.771
water	0.481	0.546	0.800
wood	0.264	0.456	0.386
soil	0.179	0.547	0.210
aluminium	0.181	0.243	0.413
living	0.005	0.050	0.005
background	0.017	0.067	0.022

Per-class results

Predicted labels

True labels	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	1107	361	5154	19666	0	281	1049	5395	414	97	409	0	5490	3281	7779	12	389	50884
aluminium	387	16307	113	569	2	382	4306	808	220	437	274	0	8763	0	154	250	6515	39487
asphalt	4065	0	1189089	387418	0	0	947	81	41999	6	8	3094	766	9054	71242	5	3	1707777
brick	4208	39439	588704	2006000	1568	25254	18383	44838	25676	626	1352	873	17630	18253	131207	1004	9117	2934132
cloud	533	13	0	0	368451	0	1322	0	0	0	294166	0	231	0	0	0	0	664716
concrete	0	0	0	3378	0	77664	3807	369	366	0	644	0	13623	651	139	0	44	100685
foliage	900	3863	17	8647	499	3090	425448	722	9727	57	326	61	12019	3251	4962	758	3361	477708
glass	233	1162	8083	13593	0	1859	5485	98437	681	1076	0	0	13992	24	655	55	392	145727
grass	490	83	33503	18002	0	1284	2394	444	138377	2	0	3990	14297	14579	12529	3825	306	244105
living	228	112	361	14090	0	569	2768	3620	1282	159	0	0	3571	1219	1553	279	2107	31918
sky	6	102	0	94	147040	0	9697	8	77	0	1718904	0	378	56	15	9	33	1876419
soil	13	0	1475	3211	0	0	565	0	16946	0	0	10246	4037	3643	8316	167	173	48792
steel	1653	5313	1318	28840	614	7471	8361	3596	14302	177	1431	206	129294	2929	5125	2843	2885	216358
stone	822	152	17062	49413	0	0	1242	620	10275	42	0	111	7836	119076	19859	1473	403	228386
tile	1560	0	71252	281924	11	6715	2378	9493	66732	487	264	163	6133	30565	242374	840	392	721283
water	0	23	0	57	0	0	841	2	646	0	0	0	1005	27	958	15063	195	18817
wood	388	141	41	20613	0	0	4202	505	472	6	0	0	7354	72	399	985	22091	57269
Total	16593	67071	1916172	2855515	518185	124569	493195	168938	328192	3172	2017778	18744	246419	206680	507266	27568	48406	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size.

G.16 learning rate 0.01 • -5 small classes • class-weighted cross entropy loss • with colour • Xavier init.

	RGBD	RGB
mIoU	0.663	-0.002
fIoU	0.823	+0.003
accuracy	0.903	+0.003
micro-averaged precision/recall/F1	0.901	+0.004
macro-averaged precision	0.796	-0.003
macro-averaged recall	0.773	+0.003
macro-averaged F1	0.781	0.000
weighted macro-averaged precision	0.893	+0.001
weighted macro-averaged recall	0.899	+0.003
weighted macro-averaged F1	0.895	+0.002

Performance overview

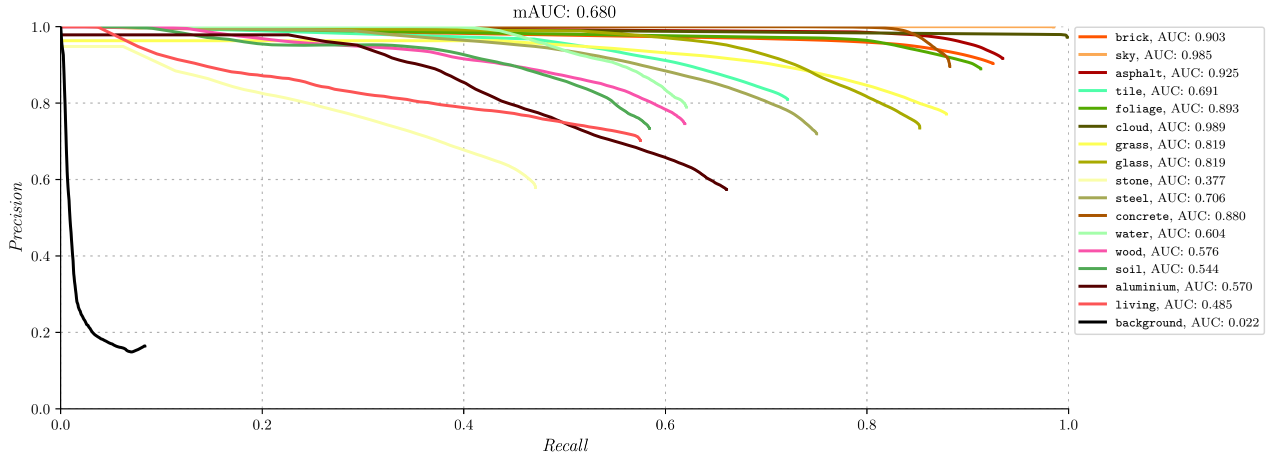
	Xavier initialized			Difference with reflect initialization		
	RGBD			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.663	0.796	0.773	-0.002	-0.003	+0.003
Weighted overall	0.824	0.893	0.900	+0.004	+0.001	+0.004
brick	0.842	0.904	0.925	+0.015	+0.010	+0.009
sky	0.985	1.000	0.985	-0.000	+0.002	-0.002
asphalt	0.862	0.917	0.935	+0.018	+0.024	-0.004
tile	0.617	0.810	0.721	+0.010	-0.008	+0.019
foliage	0.820	0.890	0.913	-0.034	-0.029	-0.010
cloud	0.971	0.973	0.999	0.000	-0.003	+0.005
grass	0.697	0.772	0.879	+0.006	-0.021	+0.037
glass	0.652	0.735	0.852	-0.061	-0.055	-0.027
stone	0.351	0.579	0.471	-0.007	-0.022	+0.002
steel	0.580	0.719	0.750	+0.003	+0.018	-0.016
concrete	0.800	0.896	0.882	-0.003	-0.036	+0.030
water	0.533	0.789	0.621	-0.140	-0.122	-0.099
wood	0.512	0.746	0.619	+0.085	+0.067	+0.083
soil	0.482	0.734	0.584	+0.034	+0.101	-0.020
aluminium	0.443	0.574	0.661	+0.045	+0.049	+0.040
living	0.462	0.702	0.575	-0.007	-0.020	+0.002
background	0.059	0.165	0.084	+0.008	+0.021	+0.011

Per-class results

Predicted labels

True labels	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	4249	5906	3216	8278	113	1447	544	9433	161	410	253	49	8131	3414	3914	57	1309	50884
aluminium	1844	26086	225	822	26	134	449	946	69	785	20	16	6855	184	157	89	780	39487
asphalt	82	9	1596266	81784	0	0	466	0	6946	0	0	400	2046	11927	7141	619	91	1707777
brick	5870	2073	98873	2714323	410	3731	1164	10663	2882	1697	360	1032	13401	18270	56951	225	2207	2934132
cloud	0	0	0	12	663853	0	3	0	0	0	204	0	319	0	325	0	0	664716
concrete	62	347	0	922	0	88822	36	474	0	0	51	0	2407	133	6129	619	683	100685
foliage	210	2641	1	1168	385	216	436145	2017	26263	262	14	851	4342	871	807	171	1344	477708
glass	978	4229	180	5289	0	0	189	124220	3	2315	0	0	5995	165	1387	0	777	145727
grass	258	0	1506	6506	0	1356	8332	187	214526	276	0	962	2782	4379	2522	130	383	244105
living	525	299	81	3207	0	0	603	6707	14	18356	0	0	1721	71	249	0	85	31918
sky	7	41	0	1046	17808	0	7884	37	0	0	1848734	0	323	0	539	0	0	1876419
soil	106	0	2507	3006	0	0	527	0	7122	138	0	28500	2025	1471	2322	0	1068	48792
steel	3216	2585	2264	10961	13	1478	2171	11306	3136	984	0	281	162279	8271	5081	173	2159	216358
stone	5665	843	15707	53560	0	150	473	95	4938	139	0	1133	3237	107629	33693	721	403	228386
tile	2055	92	19658	106552	0	1467	21375	1049	11193	241	0	5488	3200	28547	520174	0	192	721283
water	145	99	0	140	0	0	6009	0	0	0	0	0	48	122	0	11682	572	18817
wood	555	184	12	5527	0	350	3769	1875	804	537	0	122	6469	379	904	311	35471	57269
Total	25827	45434	1740496	3003103	682608	99151	490139	169009	278057	26140	1849636	38834	225580	185833	642295	14797	47524	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size.

G.17 learning rate 0.01 • -5 small classes • class-weighted cross entropy loss • with colour • Avg RGB init.

	RGBD	RGB
mIoU	0.648	-0.017
fIoU	0.822	+0.002
accuracy	0.902	+0.002
micro-averaged precision/recall/F1	0.899	+0.002
macro-averaged precision	0.787	-0.012
macro-averaged recall	0.758	-0.012
macro-averaged F1	0.766	-0.015
weighted macro-averaged precision	0.893	+0.001
weighted macro-averaged recall	0.898	+0.002
weighted macro-averaged F1	0.894	+0.001

Performance overview

	Initialized using averaged RGB			Difference with reflect initialization		
	RGBD			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.649	0.788	0.758	-0.016	-0.011	-0.012
Weighted overall	0.823	0.893	0.898	+0.003	+0.001	+0.002
`brick`	0.834	0.902	0.917	+0.007	+0.008	+0.001
`sky`	0.982	0.999	0.983	-0.003	+0.001	-0.004
`asphalt`	0.856	0.905	0.940	+0.012	+0.012	+0.001
`tile`	0.626	0.815	0.730	+0.019	-0.003	+0.028
`foliage`	0.847	0.899	0.936	-0.007	-0.020	+0.013
`cloud`	0.963	0.965	0.998	-0.008	-0.011	+0.004
`grass`	0.737	0.840	0.857	+0.046	+0.047	+0.015
`glass`	0.703	0.794	0.860	-0.010	+0.004	-0.019
`stone`	0.346	0.543	0.488	-0.012	-0.058	+0.019
`steel`	0.575	0.711	0.750	-0.002	+0.010	-0.016
`concrete`	0.736	0.854	0.841	-0.067	-0.078	-0.011
`water`	0.479	0.609	0.692	-0.194	-0.302	-0.028
`wood`	0.529	0.765	0.633	+0.102	+0.086	+0.097
`soil`	0.336	0.717	0.387	-0.112	+0.084	-0.217
`aluminium`	0.468	0.595	0.687	+0.070	+0.070	+0.066
`living`	0.364	0.693	0.433	-0.105	-0.029	-0.140
`background`	0.038	0.103	0.057	-0.013	-0.041	-0.016

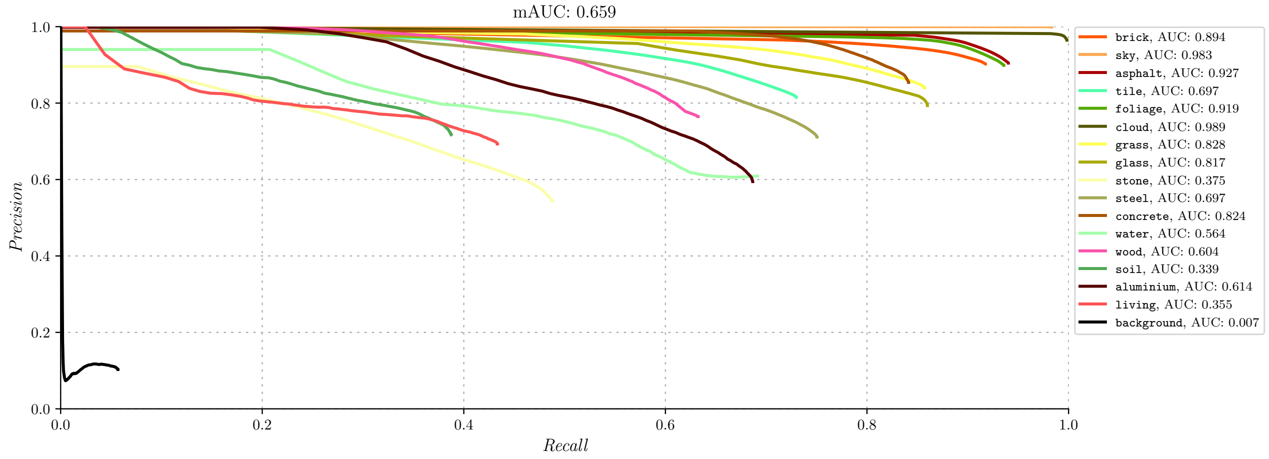
Per-class results

Predicted labels

True labels

	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	2895	8284	6268	8909	243	4	807	6338	136	449	83	47	7731	3723	4719	17	231	50884
aluminium	1157	27115	1	454	143	418	931	755	185	413	0	0	6790	59	92	62	912	39487
asphalt	141	0	1605828	80434	0	93	275	0	3025	3	0	340	468	9016	6773	1280	101	1707777
brick	7040	1181	115235	2692024	0	5567	3372	6874	562	596	2	1158	12564	31815	54872	776	494	2934132
cloud	0	3	0	57	663685	0	135	0	0	0	421	0	244	0	171	0	0	664716
concrete	789	297	0	6123	0	84704	17	299	23	0	163	4	4995	791	2084	0	396	100685
foliage	185	614	383	2431	90	517	447044	281	13683	559	20	696	4863	2594	383	327	3038	477708
glass	147	2182	1	5607	0	110	88	125323	124	2327	0	0	7470	66	883	0	1399	145727
grass	702	0	2540	3190	0	858	9210	64	209270	144	0	921	3784	4791	4363	4074	194	244105
living	1100	377	227	4085	0	0	941	8066	9	13831	0	0	1818	499	325	8	632	31918
sky	0	0	64	61	23433	0	7680	9	0	0	1844685	0	170	0	317	0	0	1876419
soil	32	83	7008	7870	0	0	1654	0	6999	12	0	18901	1893	3083	855	0	402	48792
steel	4901	2917	2538	13448	109	5954	2435	4062	2732	1146	256	26	162373	6681	5011	50	1719	216358
stone	6805	436	14262	47547	0	149	845	88	5063	136	0	639	3935	111471	35961	584	465	228386
tile	1139	1056	20524	101944	274	701	20372	3594	5593	138	382	3588	3843	30253	526504	975	403	721283
water	467	180	38	37	0	0	1070	0	986	202	0	0	627	226	1212	13012	760	18817
wood	537	877	87	9813	43	91	405	2160	765	0	0	34	4693	148	1201	192	36223	57269
Total	28037	45602	1775004	2984034	688020	99166	497281	157913	249155	19956	1846012	26354	228261	205216	645726	21357	47369	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size.

G.18 learning rate 0.01 • -5 small classes • class-weighted cross entropy loss • no pretraining

	RGB	RGBD
mIoU	0.359	0.390
fIoU	0.579	0.568
accuracy	0.715	0.714
micro-averaged precision/recall/F1	0.712	0.711
macro-averaged precision	0.502	0.604
macro-averaged recall	0.459	0.493
macro-averaged F1	0.455	0.508
weighted macro-averaged precision	0.688	0.703
weighted macro-averaged recall	0.709	0.705
weighted macro-averaged F1	0.695	0.692

Performance overview

	No pretraining						Difference with pretraining					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.359	0.502	0.459	0.391	0.605	0.493	-0.306	-0.297	-0.311	-0.268	-0.194	0.000
Weighted overall	0.579	0.688	0.709	0.569	0.703	0.705	-0.241	-0.204	-0.187	-0.250	-0.187	-0.192
brick	0.530	0.659	0.731	0.565	0.632	0.842	-0.297	-0.235	-0.185	-0.263	-0.262	-0.075
sky	0.968	0.991	0.976	0.922	0.992	0.929	-0.017	-0.007	-0.011	-0.060	-0.005	-0.055
asphalt	0.486	0.647	0.661	0.407	0.675	0.506	-0.358	-0.246	-0.278	-0.449	-0.230	-0.434
tile	0.140	0.292	0.212	0.160	0.372	0.220	-0.467	-0.526	-0.490	-0.434	-0.411	-0.491
foliage	0.738	0.785	0.924	0.753	0.833	0.888	-0.116	-0.134	+0.001	-0.110	-0.076	-0.056
cloud	0.925	0.941	0.981	0.809	0.830	0.969	-0.046	-0.035	-0.013	-0.156	-0.139	-0.027
grass	0.445	0.596	0.636	0.456	0.619	0.635	-0.246	-0.197	-0.206	-0.247	-0.189	-0.209
glass	0.385	0.559	0.552	0.483	0.623	0.682	-0.328	-0.231	-0.327	-0.183	-0.117	-0.187
stone	0.119	0.318	0.161	0.117	0.346	0.149	-0.239	-0.283	-0.308	-0.234	-0.288	-0.291
steel	0.283	0.459	0.425	0.265	0.452	0.390	-0.294	-0.242	-0.341	-0.323	-0.263	-0.377
concrete	0.515	0.615	0.759	0.623	0.690	0.866	-0.288	-0.317	-0.093	-0.156	-0.245	+0.043
water	0.077	0.152	0.135	0.309	0.750	0.344	-0.596	-0.759	-0.585	-0.323	-0.076	-0.386
wood	0.047	0.098	0.082	0.111	0.301	0.150	-0.380	-0.581	-0.454	-0.393	-0.413	-0.481
soil	0.036	0.243	0.040	0.138	0.556	0.155	-0.412	-0.390	-0.564	-0.126	+0.122	-0.248
aluminium	0.042	0.139	0.057	0.098	0.312	0.124	-0.356	-0.386	-0.564	-0.438	-0.408	-0.554
living	0.004	0.539	0.004	0.039	0.690	0.040	-0.465	-0.183	-0.569	-0.399	-0.118	-0.449
background	0.012	0.095	0.014	0.017	0.073	0.022	-0.039	-0.049	-0.059	-0.008	-0.021	-0.012

Per-class results

Predicted labels

True labels

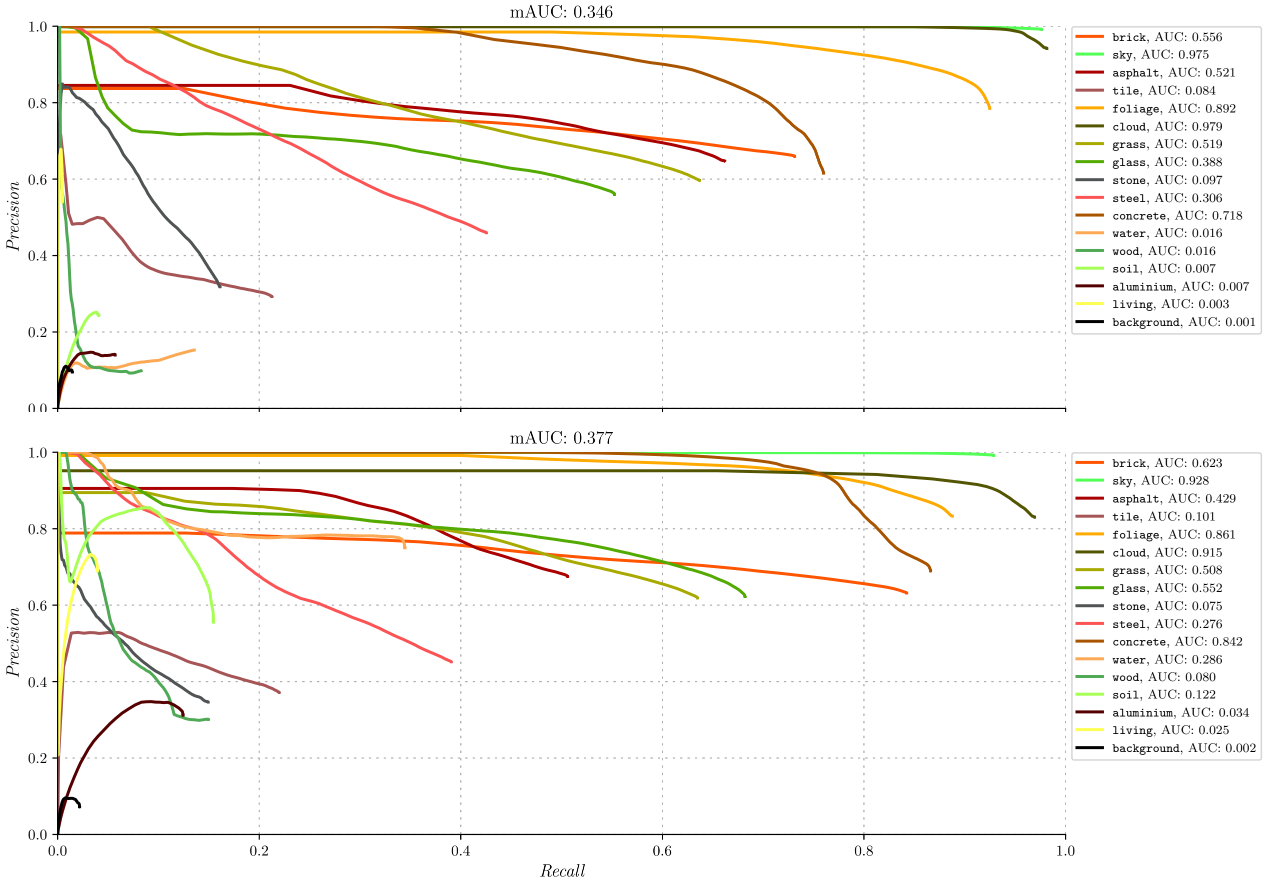
	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	742	4900	5030	14485	387	114	821	10000	478	33	138	68	5906	2349	4351	132	950	50884
aluminium	590	2252	371	3702	100	499	4628	2579	5760	0	221	17	10784	206	4143	348	3287	39487
asphalt	2	257	1130051	500259	0	6160	6995	333	15083	0	0	288	3210	11680	33182	76	201	1707777
brick	1713	2023	389425	2145960	2454	26203	17559	16638	18274	31	74	2016	16000	18842	253077	786	23057	2934132
cloud	0	0	0	1309	652423	0	0	29	0	0	9918	0	1020	0	17	0	0	664716
concrete	12	10	0	10172	0	76512	390	70	0	0	133	0	9097	44	2998	0	1247	100685
foliage	89	115	539	2775	253	2	441765	1548	21210	0	5	4	5459	2396	1129	7	412	477708
glass	628	2496	967	18422	0	122	6675	80491	0	12	2540	0	24667	134	4476	0	4097	145727
grass	87	11	5470	13344	0	242	37630	836	155435	0	0	941	8596	12861	7099	627	926	244105
living	414	279	0	11525	0	0	2318	6228	403	142	0	0	3681	1180	2825	24	2899	31918
sky	60	2	0	177	35411	0	7385	338	0	0	1831961	0	833	0	252	0	0	1876419
soil	129	0	143	13908	0	5543	616	9	14448	0	0	1988	2422	5859	2612	42	1073	48792
steel	203	1515	15994	43258	1404	8467	13444	5884	1695	45	2196	266	92005	11231	16008	816	1927	216358
stone	674	103	40437	87831	0	291	1734	8890	15307	0	0	2228	3799	36771	28604	1446	271	228386
tile	371	516	154222	369229	189	105	6648	4398	6946	0	40	166	3758	11202	153320	9704	469	721283
water	0	0	725	1307	0	0	1171	219	3545	0	0	182	3617	366	2468	2550	2667	18817
wood	2070	1639	603	15552	0	0	12831	5280	1795	0	0	0	5194	332	7081	151	4741	57269
Total	7784	16118	1743977	3253215	692621	124260	562610	143770	260379	263	1847226	8164	200048	115453	523642	16709	48224	9564463

Predicted labels

True labels

	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	1109	59	4370	16370	30	536	720	12915	633	27	305	26	8865	2103	2429	39	348	50884
aluminium	1361	4910	6	4236	55	1077	3231	1146	7383	45	474	0	12037	442	1320	101	1663	39487
asphalt	39	45	864378	749485	0	14914	1196	676	6656	0	0	1848	13186	11166	43535	102	551	1707777
brick	453	3275	217433	2471518	4438	8965	7507	12356	6708	0	0	1053	16980	13797	164060	101	5488	2934132
cloud	76	0	0	10496	644208	0	0	30	0	0	8830	0	1076	0	0	0	0	664716
concrete	210	0	24	4186	0	87189	766	181	155	0	514	0	3621	106	3312	0	421	100685
foliage	655	1469	535	4784	123	412	423995	3754	30626	164	9	154	4475	1541	2829	28	2155	477708
glass	4145	2251	614	18703	105	140	2716	99387	199	158	946	0	11092	381	3060	261	1569	145727
grass	901	3	22093	12004	0	1727	27494	359	154941	0	0	922	8356	6688	7198	542	877	244105
living	1472	119	947	11541	0	0	1760	4563	155	1262	0	0	4382	1259	3286	0	1172	31918
sky	0	18	0	383	124126	0	7143	1557	0	0	1742850	0	305	0	37	0	0	1876419
soil	158	0	1438	19708	0	0	261	33	8158	0	0	7540	1697	5912	3292	192	403	48792
steel	2173	2427	9381	51213	2680	10353	9947	8113	8948	127	3283	1196	84469	7683	10317	320	3728	216358
stone	503	315	31080	123230	252	63	2099	4583	8818	0	0	103	2353	34104	20526	133	224	228386
tile	495	474	125451	390647	1	852	9847	4313	9991	43	208	591	5334	13213	158548	109	1166	721283
water	85	77	2072	140	0	0	571	135	5004	0	0	0	3043	43	1047	6482	118	18817
wood	1419	287	43	21636	2	148	9935	5532	1897	3	224	136	5608	3	1601	229	8566	57269
Total	15254	15729	1279865	3910280	776020	126376	509188	159633	250272	1829	1757643	13569	186879	98441	426397	8639	28449	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.19 learning rate 0.01 • -5 small classes • class-weighted cross entropy loss • no pretraining, second run

	RGB	RGBD
mIoU	0.383	0.388
fIoU	0.563	0.583
accuracy	0.702	0.723
micro-averaged precision/recall/F1	0.699	0.720
macro-averaged precision	0.520	0.540
macro-averaged recall	0.498	0.496
macro-averaged F1	0.499	0.498
weighted macro-averaged precision	0.681	0.700
weighted macro-averaged recall	0.701	0.717
weighted macro-averaged F1	0.688	0.704

Performance overview

	No pretraining						Difference with pretrainng					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.383	0.520	0.498	0.388	0.540	0.496	-0.282	-0.279	-0.272	-0.271	-0.259	0.000
Weighted overall	0.563	0.681	0.701	0.583	0.700	0.717	-0.257	-0.211	-0.195	-0.236	-0.190	-0.180
brick	0.515	0.654	0.709	0.552	0.659	0.773	-0.312	-0.240	-0.207	-0.276	-0.235	-0.144
sky	0.939	0.990	0.947	0.948	0.994	0.953	-0.046	-0.008	-0.040	-0.034	-0.003	-0.031
asphalt	0.461	0.631	0.631	0.470	0.679	0.605	-0.383	-0.262	-0.308	-0.386	-0.226	-0.335
tile	0.152	0.303	0.233	0.191	0.387	0.274	-0.455	-0.515	-0.469	-0.403	-0.396	-0.437
foliage	0.690	0.730	0.927	0.733	0.781	0.922	-0.164	-0.189	+0.004	-0.130	-0.128	-0.022
cloud	0.844	0.871	0.965	0.873	0.890	0.977	-0.127	-0.105	-0.029	-0.092	-0.079	-0.019
grass	0.444	0.576	0.659	0.439	0.550	0.684	-0.247	-0.217	-0.183	-0.264	-0.258	-0.160
glass	0.386	0.498	0.633	0.420	0.562	0.625	-0.327	-0.292	-0.246	-0.246	-0.178	-0.244
stone	0.109	0.282	0.151	0.104	0.268	0.146	-0.249	-0.319	-0.318	-0.247	-0.366	-0.294
steel	0.307	0.510	0.434	0.298	0.484	0.438	-0.270	-0.191	-0.332	-0.290	-0.231	-0.329
concrete	0.640	0.752	0.812	0.676	0.753	0.868	-0.163	-0.180	-0.040	-0.103	-0.182	+0.045
water	0.217	0.472	0.287	0.090	0.272	0.118	-0.456	-0.439	-0.433	-0.542	-0.554	-0.612
wood	0.080	0.240	0.108	0.119	0.309	0.163	-0.347	-0.439	-0.428	-0.385	-0.405	-0.468
soil	0.131	0.274	0.201	0.094	0.354	0.114	-0.317	-0.359	-0.403	-0.170	-0.080	-0.289
aluminium	0.217	0.543	0.266	0.195	0.414	0.269	-0.181	+0.018	-0.355	-0.341	-0.306	-0.409
living	0.000	0.002	0.000	0.006	0.290	0.007	-0.469	-0.720	-0.573	-0.432	-0.518	-0.482
background	0.004	0.022	0.005	0.008	0.040	0.010	-0.047	-0.122	-0.068	-0.017	-0.054	-0.024

Per-class results

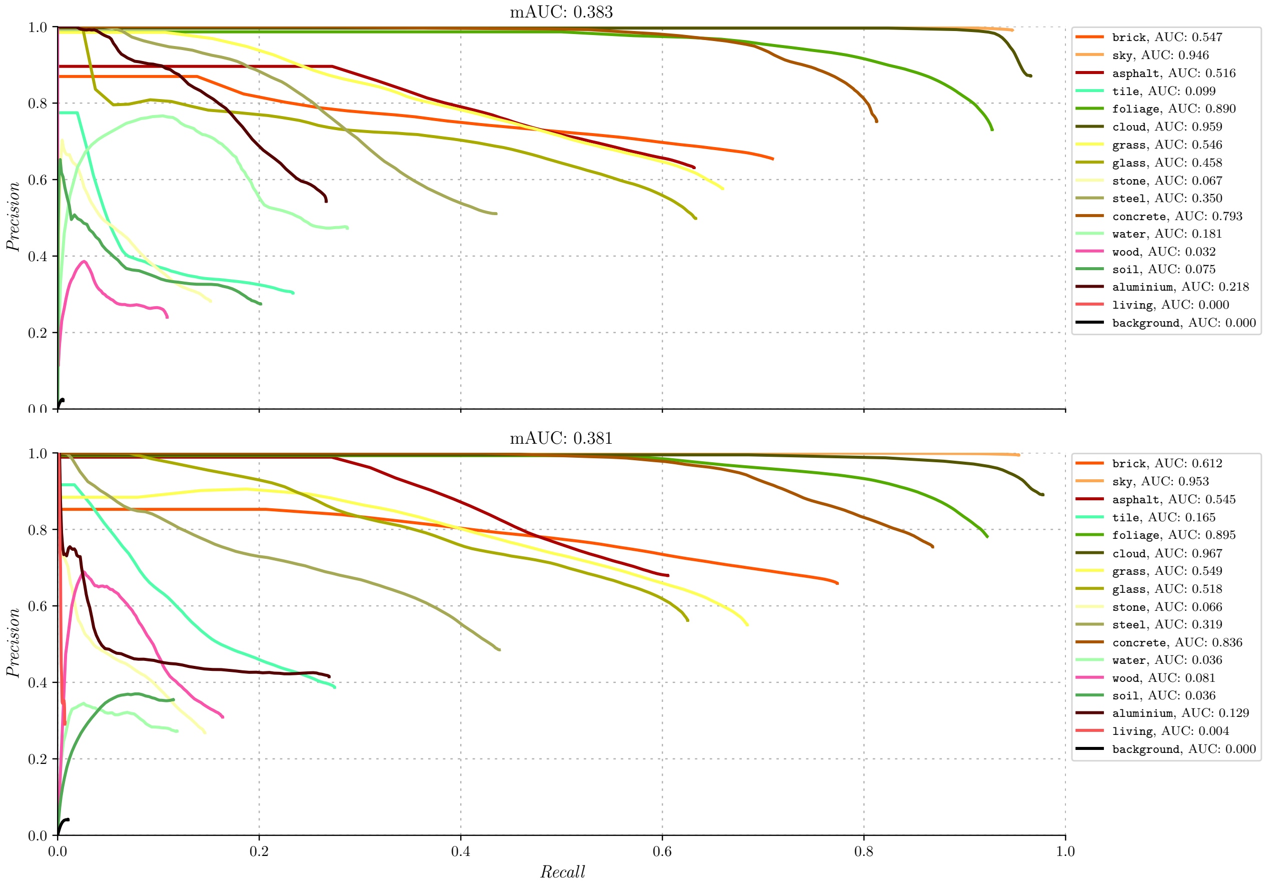
Predicted labels

True labels	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	274	4775	4852	15150	54	517	1226	13087	139	12	194	0	5220	2232	1720	90	1342	50884
aluminium	917	10522	96	5136	11	907	3960	1431	2635	20	74	0	10134	259	965	281	2139	39487
asphalt	336	0	1078346	495513	0	14654	3618	6726	21963	0	0	4007	1312	23558	56675	363	706	1707777
brick	5141	434	446383	2080890	2390	4514	34495	27514	17331	16	0	8322	12604	26118	257673	951	9356	2934132
cloud	0	0	0	7448	641688	0	0	175	0	0	11559	0	2124	0	1722	0	0	664716
concrete	120	0	0	8385	0	81809	1962	376	0	0	520	0	3389	0	3703	0	421	100685
foliage	112	19	18	4744	114	207	442922	637	19051	0	1	348	4938	461	2955	9	1172	477708
glass	1087	112	19	15180	0	0	10673	92255	0	280	1935	0	18027	570	4307	13	1269	145727
grass	61	46	4250	17898	0	206	33524	1289	161050	0	0	1883	8835	5627	8882	424	130	244105
living	838	1	60	12475	0	0	5219	5768	58	1	0	13	4401	1058	623	0	1403	31918
sky	0	0	0	24	90777	0	7590	42	0	0	1777252	0	734	0	0	0	0	1876419
soil	0	0	8846	5277	0	16	807	0	12758	0	0	9830	1828	5904	2880	0	646	48792
steel	723	2868	7292	49520	1522	4268	19729	10601	3559	51	1990	1114	94084	7837	8314	2083	803	216358
stone	1593	90	41100	83910	0	930	3396	8003	18454	0	0	1419	3730	34647	30630	466	18	228386
tile	20	235	116323	363272	0	731	18084	8228	15833	98	0	8512	6341	13791	168391	1218	206	721283
water	803	17	0	112	0	0	4858	87	4627	0	0	0	1809	182	824	5408	90	18817
wood	315	252	73	15299	0	0	14205	8764	1866	0	0	299	4612	563	4663	133	6225	57269
Total	12340	19371	1707658	3180233	736556	108759	606268	184983	279324	478	1793525	35747	184122	122807	554927	11439	25926	9564463

Predicted labels

True labels	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	529	3301	4915	14848	230	12	4402	4582	374	0	100	24	11326	1624	4072	6	539	50884
aluminium	1150	10635	0	3963	4	818	4911	3272	1284	0	37	0	8890	280	768	59	3416	39487
asphalt	624	0	1033974	533453	0	12549	4734	1469	30903	0	0	2631	9143	22046	55948	94	209	1707777
brick	2105	1909	339237	2269617	353	725	32021	20994	20023	0	40	1519	14607	30079	194073	133	6697	2934132
cloud	4	0	0	5647	649814	0	0	99	0	0	7655	0	1055	0	442	0	0	664716
concrete	0	1273	0	5583	0	87401	640	0	468	0	179	0	3012	0	2125	0	4	100685
foliage	255	652	50	5536	121	208	440528	1603	17040	0	2	353	4353	2733	1998	1163	1113	477708
glass	172	2058	111	23127	0	75	8673	91091	58	102	0	0	16381	130	3034	0	715	145727
grass	208	107	4567	24295	0	2443	25622	1231	167044	34	0	2235	6017	5545	3383	679	695	244105
living	1417	136	33	11533	0	0	489	5803	66	224	0	0	8012	1378	1093	98	1636	31918
sky	65	136	0	809	76849	0	6822	650	0	0	1789097	0	762	0	0	0	1229	1876419
soil	58	1	1172	7517	0	0	1201	38	20315	0	0	5601	429	5488	6700	63	209	48792
steel	1835	4191	10428	39971	1950	10018	9522	11157	4230	392	1727	363	94808	7332	14380	2243	1811	216358
stone	2476	6	37066	105450	0	965	3085	5721	13925	0	0	1492	2524	33356	21519	669	132	228386
tile	1010	312	89428	373603	81	593	6530	8200	22963	0	13	946	4235	14021	198241	745	362	721283
water	737	178	745	81	0	0	427	115	3738	0	0	0	5974	256	2154	2230	2182	18817
wood	314	732	8	18766	0	113	14144	5995	1017	18	0	618	4013	173	1979	0	9379	57269
Total	12959	25627	1521734	3443799	729402	115920	563751	162020	303448	770	1798850	15782	195541	124441	511909	8182	30328	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.20 learning rate 0.1 • -5 small classes • class-weighted cross entropy loss • no pretraining

	RGB	RGBD
mIoU	0.357	0.315
fIoU	0.581	0.499
accuracy	0.727	0.656
micro-averaged precision/recall/F1	0.724	0.653
macro-averaged precision	0.513	0.469
macro-averaged recall	0.463	0.422
macro-averaged F1	0.454	0.407
weighted macro-averaged precision	0.705	0.644
weighted macro-averaged recall	0.718	0.641
weighted macro-averaged F1	0.703	0.620

Performance overview

	No pretraining						Difference with pretraining					
	RGB			RGBD			RGB			RGBD		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.357	0.513	0.463	0.315	0.469	0.422	-0.308	-0.286	-0.307	-0.344	-0.330	0.000
Weighted overall	0.581	0.705	0.718	0.499	0.644	0.641	-0.239	-0.187	-0.178	-0.320	-0.246	-0.256
`brick`	0.584	0.664	0.829	0.480	0.549	0.794	-0.243	-0.230	-0.087	-0.348	-0.345	-0.123
`sky`	0.909	0.995	0.913	0.932	0.994	0.937	-0.076	-0.003	-0.074	-0.050	-0.003	-0.047
`asphalt`	0.486	0.717	0.601	0.269	0.664	0.312	-0.358	-0.176	-0.338	-0.587	-0.241	-0.628
`tile`	0.195	0.407	0.273	0.159	0.289	0.261	-0.412	-0.411	-0.429	-0.435	-0.494	-0.450
`foliage`	0.752	0.807	0.916	0.655	0.727	0.869	-0.102	-0.112	-0.007	-0.208	-0.182	-0.075
`cloud`	0.792	0.803	0.983	0.815	0.854	0.945	-0.179	-0.173	-0.011	-0.150	-0.115	-0.051
`grass`	0.514	0.684	0.675	0.285	0.451	0.437	-0.177	-0.109	-0.167	-0.418	-0.357	-0.407
`glass`	0.346	0.411	0.687	0.406	0.456	0.789	-0.367	-0.379	-0.192	-0.260	-0.284	-0.080
`stone`	0.081	0.329	0.098	0.031	0.236	0.035	-0.277	-0.272	-0.371	-0.320	-0.398	-0.405
`steel`	0.256	0.439	0.380	0.255	0.606	0.306	-0.321	-0.262	-0.386	-0.333	-0.109	-0.461
`concrete`	0.628	0.725	0.824	0.608	0.667	0.872	-0.175	-0.207	-0.028	-0.171	-0.268	+0.049
`water`	0.060	0.177	0.083	0.138	0.355	0.184	-0.613	-0.734	-0.637	-0.494	-0.471	-0.546
`wood`	0.041	0.197	0.049	0.006	0.066	0.007	-0.386	-0.482	-0.487	-0.498	-0.648	-0.624
`soil`	0.015	0.087	0.019	0.000	0.000	0.000	-0.433	-0.546	-0.585	-0.264	-0.434	-0.403
`aluminium`	0.060	0.260	0.073	0.006	0.121	0.006	-0.338	-0.265	-0.548	-0.530	-0.599	-0.672
`living`	0.000	—	0.000	0.000	—	0.000	-0.469	—	-0.573	-0.438	—	-0.489
`background`	0.001	0.016	0.002	0.001	0.009	0.002	-0.050	-0.128	-0.071	-0.024	-0.085	-0.032

Per-class results

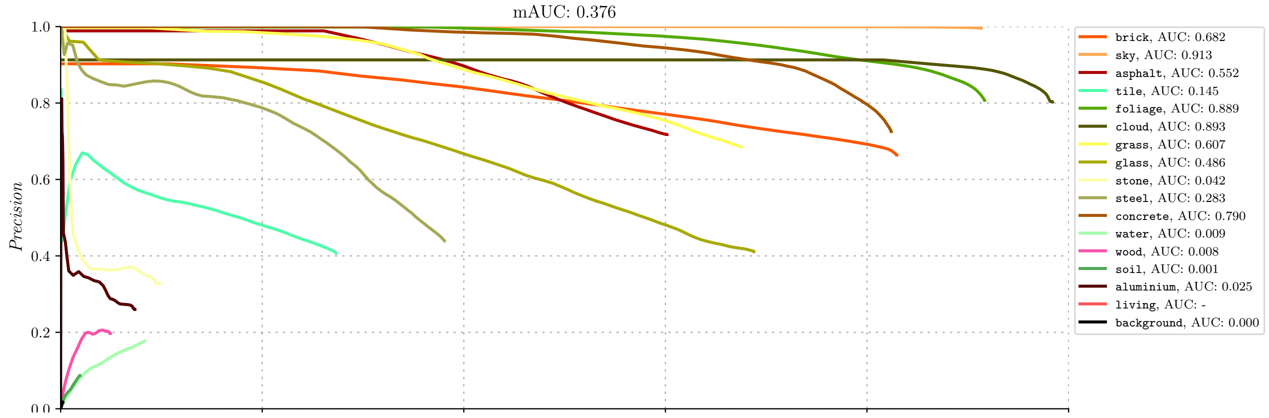
Predicted labels

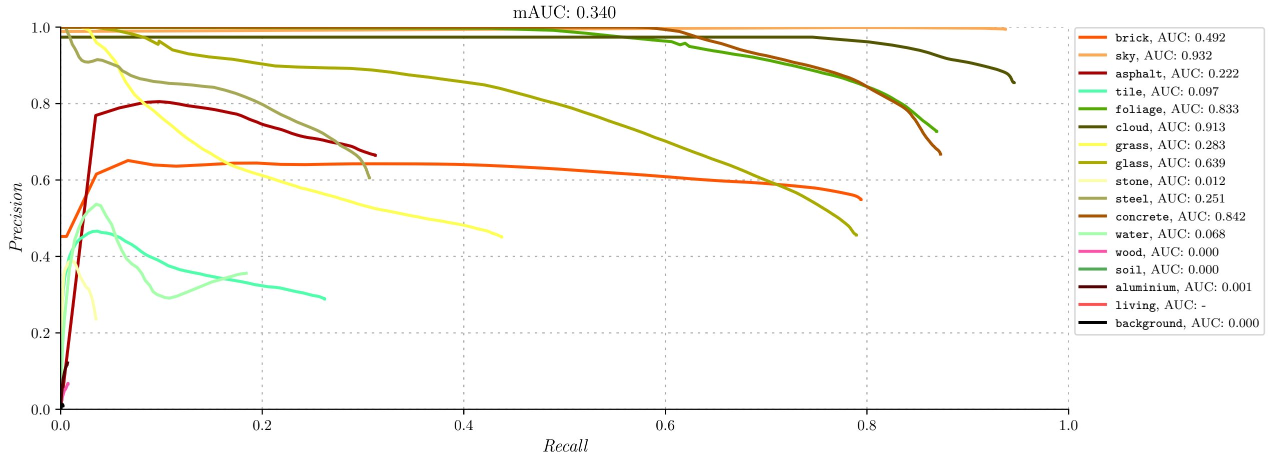
True labels	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	106	538	4478	13777	322	194	598	14289	142	0	144	230	9531	1584	4370	0	581	50884
aluminium	135	2910	0	5935	108	498	4250	8277	448	0	151	59	15007	175	1426	20	88	39487
asphalt	652	80	1027376	598690	0	370	0	1780	19857	0	0	1051	2377	10942	44597	0	5	1707777
brick	2308	873	276963	2433829	2779	9320	10337	32232	3830	0	91	265	9116	4282	144045	24	3838	2934132
cloud	0	0	0	213	654057	0	0	827	0	0	4249	0	2404	0	2966	0	0	664716
concrete	0	0	0	10204	69	82976	0	120	0	0	64	0	4533	0	2692	0	27	100685
foliage	0	664	0	3453	27	2050	437912	12189	11828	0	6	318	5858	1100	1055	326	922	477708
glass	273	3027	387	15568	528	18	2611	100232	0	0	677	0	17990	1000	2427	0	989	145727
grass	0	84	402	9503	0	1678	28713	3916	164816	0	0	3421	8262	9960	11091	455	1804	244105
living	1831	263	175	12249	0	0	3045	5984	46	0	0	0	6981	414	779	0	151	31918
sky	0	58	91	0	154654	0	4193	477	0	0	1713729	0	3159	0	58	0	0	1876419
soil	0	0	618	27252	0	0	2249	1298	10606	0	0	934	2355	1714	1074	0	692	48792
steel	27	1861	5702	34818	1659	14018	5258	29322	1830	0	1619	257	82364	3572	28703	4955	393	216358
stone	177	29	24164	120970	0	54	1425	1483	15231	0	0	1638	2999	22466	37341	394	15	228386
tile	694	430	91117	356975	2	1462	26003	13568	8162	0	113	2430	10201	10773	197170	774	1409	721283
water	35	0	0	1928	0	0	3525	5197	2786	0	0	109	2374	0	721	1568	574	18817
wood	22	369	25	19713	0	1755	12365	12581	1184	0	0	18	2083	262	3757	313	2822	57269
Total	6260	11186	1431498	3665077	814205	114393	542484	243772	240766	0	1720843	10730	187594	68244	484272	8829	14310	9564463

Predicted labels

True labels	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	116	84	1252	21160	526	834	3587	14627	69	0	251	0	3929	1694	2728	0	27	50884
aluminium	0	270	29	10383	444	1291	5311	6475	3922	0	309	12	7406	84	2059	37	1455	39487
asphalt	2355	0	533074	1053105	0	2703	6030	1503	26818	0	0	33	1362	3450	77086	258	0	1707777
brick	5020	260	162141	2329983	2084	15251	36934	37642	15601	0	558	0	2678	6933	318340	437	270	2934132
cloud	0	0	14073	13766	628782	0	0	426	0	0	6412	0	1013	0	241	3	0	664716
concrete	0	222	0	6189	129	87896	2456	294	0	0	0	0	1629	0	1660	0	210	100685
foliage	0	19	0	35678	234	93	415166	4927	15045	0	4	258	4218	594	819	244	409	477708
glass	0	178	0	18575	0	0	3325	115041	18	0	294	0	4383	1342	2413	0	158	145727
grass	6	0	2641	75031	0	2742	36892	1127	106829	0	0	16	7545	3973	5545	51	1707	244105
living	993	0	98	10699	0	0	4089	11737	956	0	0	0	1006	585	1462	65	228	31918
sky	3311	0	4366	543	100676	0	1	8374	0	0	1758655	0	478	0	0	15	0	1876419
soil	0	0	0	24260	0	71	996	0	18038	0	0	0	870	684	3332	107	434	48792
steel	0	677	881	72175	2580	16415	9200	25777	4301	0	1370	0	66284	1604	10529	4434	131	216358
stone	287	0	16122	160737	0	998	3736	1487	4514	0	0	0	837	8014	30901	360	393	228386
tile	103	0	67242	387280	0	3020	21298	12978	32909	0	0	0	3000	4053	188895	132	373	721283
water	0	0	0	150	0	265	1078	8	6890	0	0	0	654	737	5535	3464	36	18817
wood	0	513	0	21271	0	42	20615	9679	765	0	0	0	2071	75	1696	124	418	57269
Total	12191	2223	801919	4240985	735455	131621	570714	252102	236675	0	1767853	319	109363	33822	653241	9731	6249	9564463

Confusion matrices





Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.21 learning rate 0.01 • -5 small classes • class-weighted cross entropy loss • bilinear depth downscaling

	RGB	RGBD
mIoU	0.665	0.646
fIoU	0.820	0.819
accuracy	0.900	0.899
micro-averaged precision/recall/F1	0.897	0.896
macro-averaged precision	0.799	0.788
macro-averaged recall	0.770	0.747
macro-averaged F1	0.781	0.760
weighted macro-averaged precision	0.892	0.890
weighted macro-averaged recall	0.896	0.896
weighted macro-averaged F1	0.893	0.892

Performance overview. The RGB metrics are from Appendix G.10, our baseline.

	RGB			RGBD			RGBD - RGB		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.665	0.799	0.770	0.647	0.788	0.748	-0.018	-0.011	-0.022
Weighted overall	0.820	0.892	0.896	0.820	0.891	0.896	-0.000	-0.001	0.000
brick	0.827	0.894	0.916	0.825	0.893	0.916	-0.002	-0.001	-0.000
sky	0.985	0.998	0.987	0.986	0.999	0.987	+0.001	+0.001	-0.000
asphalt	0.844	0.893	0.939	0.854	0.902	0.941	+0.010	+0.009	+0.002
tile	0.607	0.818	0.702	0.602	0.809	0.702	-0.005	-0.009	-0.000
foliage	0.854	0.919	0.923	0.858	0.917	0.930	+0.004	-0.002	+0.007
cloud	0.971	0.976	0.994	0.971	0.975	0.995	-0.000	-0.001	+0.001
grass	0.691	0.793	0.842	0.685	0.778	0.851	-0.006	-0.015	+0.009
glass	0.713	0.790	0.879	0.746	0.822	0.889	+0.033	+0.032	+0.010
stone	0.358	0.601	0.469	0.335	0.552	0.460	-0.023	-0.049	-0.009
steel	0.577	0.701	0.766	0.562	0.698	0.742	-0.015	-0.003	-0.024
concrete	0.803	0.932	0.852	0.823	0.931	0.876	+0.020	-0.001	+0.024
water	0.673	0.911	0.720	0.638	0.898	0.688	-0.035	-0.013	-0.032
wood	0.427	0.679	0.536	0.502	0.680	0.658	+0.075	+0.001	+0.122
soil	0.448	0.633	0.604	0.310	0.604	0.388	-0.138	-0.029	-0.216
aluminium	0.398	0.525	0.621	0.374	0.489	0.613	-0.024	-0.036	-0.008
living	0.469	0.722	0.573	0.283	0.663	0.331	-0.186	-0.059	-0.242
background	0.051	0.144	0.073	0.100	0.289	0.132	+0.049	+0.145	+0.059

Per-class results. The RGB metrics are from Appendix G.10, our baseline.

Predicted labels

True labels

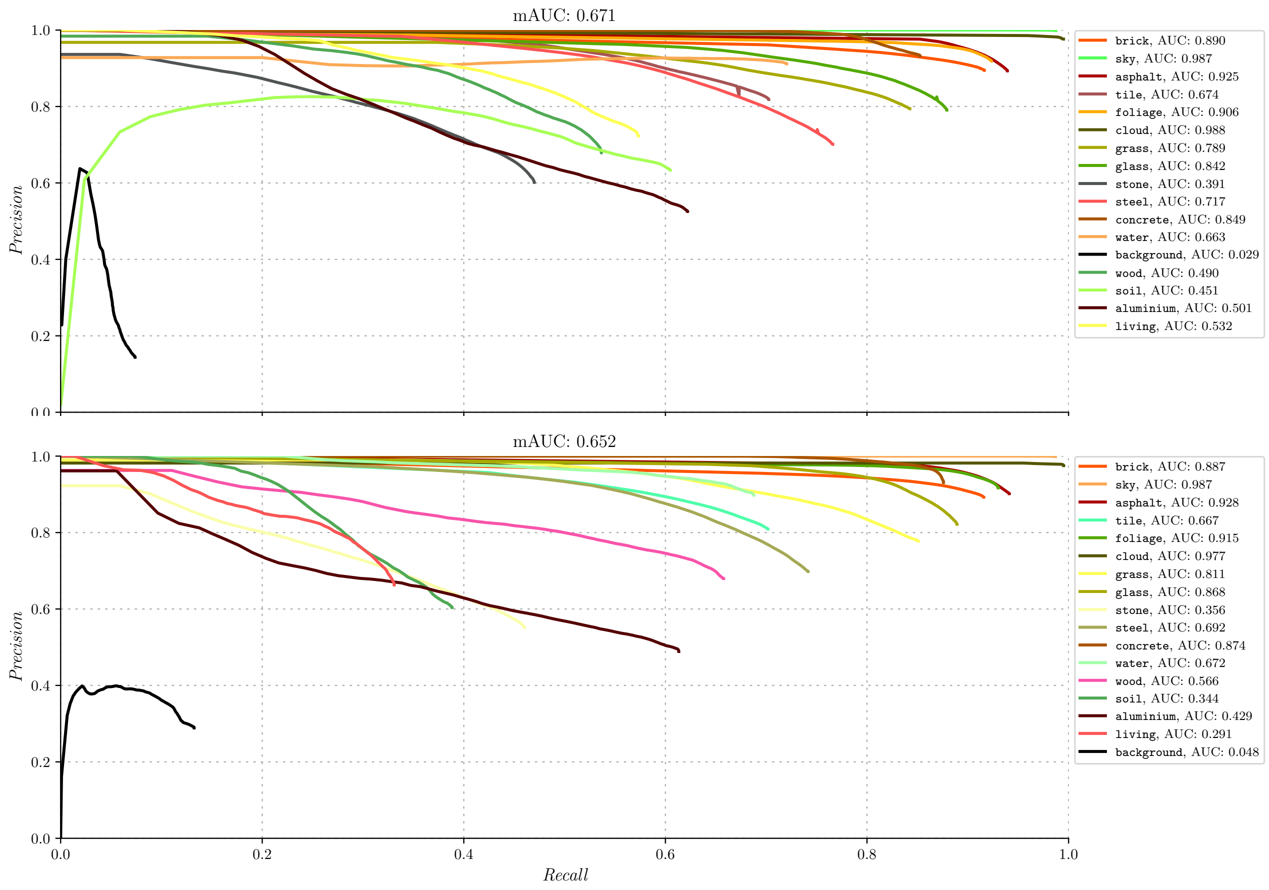
	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	3760	11172	2294	8468	275	11	284	5459	71	715	136	0	9918	3829	3774	53	665	50884
aluminium	1514	24560	2	641	142	460	842	690	63	720	19	0	8384	8	365	0	1077	39487
asphalt	895	2	1604133	81415	0	0	28	0	3708	0	0	0	917	9343	7280	0	56	1707777
brick	3862	913	140081	2689008	0	385	782	7421	1142	1589	327	1108	12993	15456	57193	0	1872	2934132
cloud	0	74	0	961	661384	0	8	25	0	0	1696	0	447	0	121	0	0	664716
concrete	0	381	0	2572	0	85845	0	882	13	0	77	0	7996	586	1099	0	1234	100685
foliage	1142	1217	137	3438	141	849	441161	542	20067	239	58	634	4628	1458	197	397	1403	477708
glass	1179	3070	156	4179	0	40	81	128132	38	1155	0	0	5691	5	716	0	1285	145727
grass	666	18	5082	2955	0	717	12025	0	205678	0	0	5248	3926	4172	3030	6	582	244105
living	1097	215	33	1196	0	0	50	5726	14	18296	0	0	1111	221	3939	0	20	31918
sky	127	0	0	7	15269	0	7698	19	0	0	1852721	0	60	0	518	0	0	1876419
soil	52	0	3172	2822	0	178	836	0	6673	44	0	29506	834	2228	1000	0	1447	48792
steel	3003	3650	1287	14303	146	3229	2114	6424	2296	1713	185	170	165750	4444	4618	100	2926	216358
stone	4885	154	18333	55883	0	335	910	56	4857	802	0	1991	4062	107319	27984	706	109	228386
tile	2196	915	20490	130659	0	7	6111	691	13059	34	116	7885	4021	28188	506718	24	169	721283
water	210	159	0	0	0	0	1683	421	416	0	0	0	365	297	34	13556	1676	18817
wood	1501	215	47	6431	0	0	5226	5568	972	1	0	25	5313	795	427	24	30724	57269
Total	26089	46715	1795247	3004938	677357	92056	479839	162056	259067	25308	1855335	46567	236416	178349	619013	14866	45245	9564463

Predicted labels

True labels

	background	aluminium	asphalt	brick	cloud	concrete	foliage	glass	grass	living	sky	soil	steel	stone	tile	water	wood	Total
background	6739	10178	4538	8842	350	65	238	5177	160	344	39	9	7520	3019	2663	0	1003	50884
aluminium	1074	24219	67	1492	27	573	904	644	108	413	0	0	8353	40	128	0	1445	39487
asphalt	43	0	1607274	76549	0	0	87	0	3059	0	0	36	240	12253	7982	0	254	1707777
brick	3459	3068	126053	2687646	350	2480	2920	8934	3038	235	63	1637	11973	20316	57959	39	3962	2934132
cloud	46	0	0	908	661457	0	13	0	0	0	1282	0	1010	0	0	0	0	664716
concrete	0	1297	0	3434	0	88192	0	776	0	0	292	0	4965	59	1670	0	0	100685
foliage	216	900	698	2839	490	225	444371	512	14587	171	35	7	5168	1489	1678	312	4010	477708
glass	140	1292	12	4385	0	102	250	129603	78	931	35	0	7432	177	481	0	809	145727
grass	856	21	2121	5883	0	1515	10097	53	207702	260	0	2815	4832	6018	1347	456	129	244105
living	2216	428	294	6462	0	0	976	5965	36	10564	0	2	2121	367	2152	0	335	31918
sky	0	13	0	0	14970	52	7681	0	0	0	1851891	0	312	0	1500	0	0	1876419
soil	13	0	4789	2143	0	52	66	0	15193	0	0	18946	2162	1810	2422	0	1196	48792
steel	2303	7116	3821	21103	57	1411	2069	4650	1741	1587	98	0	160444	2517	4712	1	2728	216358
stone	4744	32	18379	54727	0	31	359	98	5181	994	0	1832	3652	105082	32805	276	194	228386
tile	554	172	13902	126736	0	0	10932	461	15108	60	81	5996	3935	36862	506136	87	261	721283
water	48	640	0	193	0	0	2218	0	258	320	0	0	261	255	265	12942	1417	18817
wood	855	148	252	7475	562	0	1278	834	710	47	0	86	5379	219	1456	301	37667	57269
Total	23306	49524	1782200	3010817	678263	94698	484459	157707	266959	15926	1853816	31366	229759	190483	625356	14414	55410	9564463

Confusion matrices



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.22 learning rate 0.01 • only discriminatory classes • class-weighted cross entropy loss • no `background`

	RGB	RGBD
mIoU	0.488	0.473
fIoU	0.592	0.589
accuracy	0.993	0.993
micro-averaged precision/recall/F1	0.561	0.561
macro-averaged precision	0.494	0.479
macro-averaged recall	0.961	0.945
macro-averaged F1	0.631	0.614
weighted macro-averaged precision	0.595	0.592
weighted macro-averaged recall	0.990	0.989
weighted macro-averaged F1	0.734	0.732

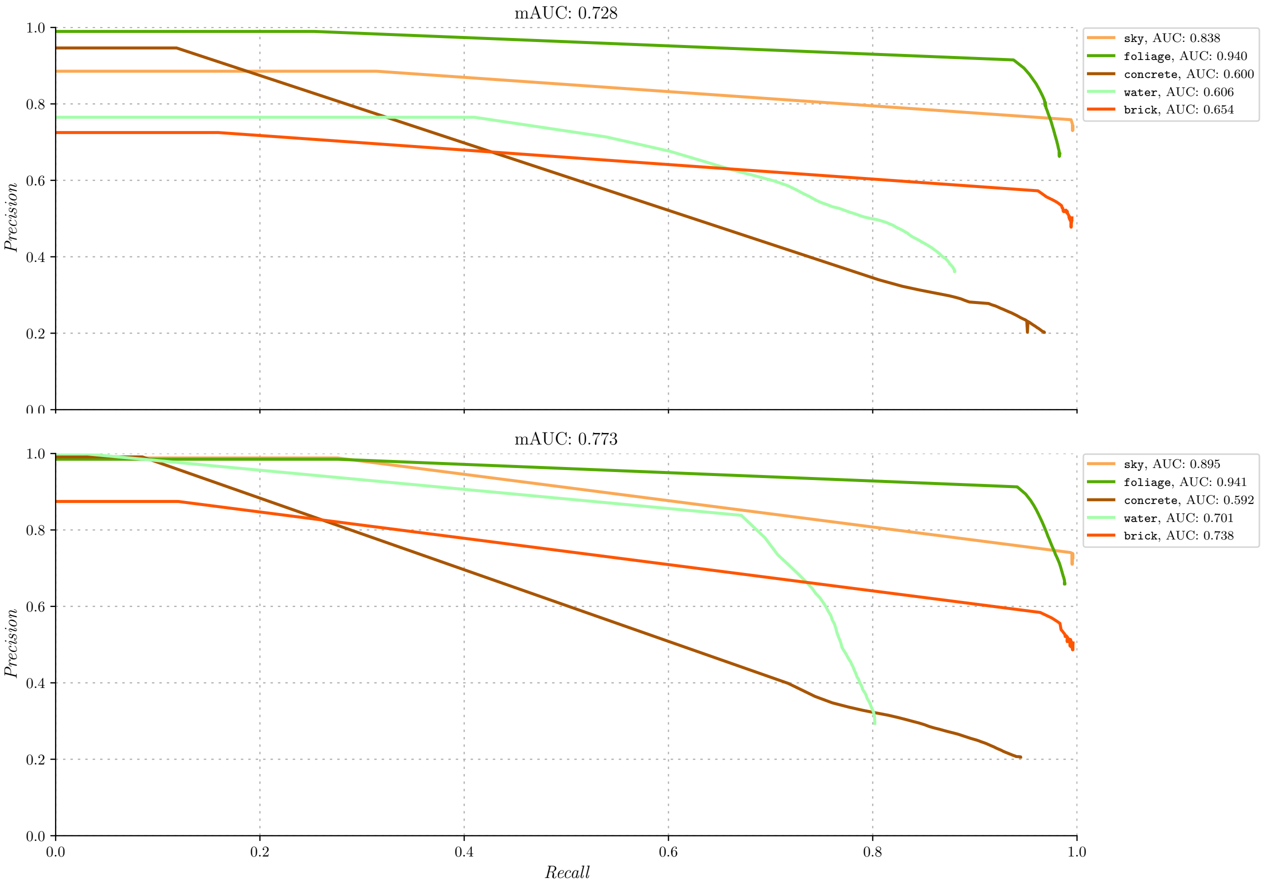
Performance overview.

	RGB			RGBD			RGBD - RGB		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.489	0.495	0.961	0.473	0.479	0.945	-0.016	-0.016	-0.016
Weighted overall	0.592	0.595	0.990	0.589	0.592	0.989	-0.003	-0.003	-0.001
`brick`	0.817	0.998	0.818	0.814	0.998	0.816	-0.003	0.000	-0.002
`sky`	0.737	0.739	0.996	0.733	0.735	0.995	-0.004	-0.004	-0.001
`foliage`	0.661	0.669	0.983	0.653	0.659	0.987	-0.008	-0.010	+0.004
`concrete`	0.201	0.203	0.951	0.203	0.205	0.944	+0.002	+0.002	-0.007
`water`	0.345	0.362	0.880	0.274	0.294	0.802	-0.071	-0.068	-0.078

Per-class results.

		Predicted labels						Predicted labels					
		`brick`	`concrete`	`foliage`	`sky`	`water`	Total	`brick`	`concrete`	`foliage`	`sky`	`water`	Total
True labels	`brick`	5803672	376185	223527	658452	28998	7090834	5783664	366384	232696	672529	35561	7090834
	`concrete`	4478	95797	31	379	0	100685	4192	95124	1257	112	0	100685
	`foliage`	7012	856	469654	0	186	477708	4703	367	471925	48	665	477708
	`sky`	22	1	7709	1868687	0	1876419	121	0	7691	1868607	0	1876419
	`water`	1162	0	1093	0	16562	18817	1382	0	2343	0	15092	18817
	Total	5816346	472839	702014	2527518	45746	9564463	5794062	461875	715912	2541296	51318	9564463

Confusion matrices. Left: RGB, right: RGBD



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD

G.23 learning rate 0.01 • instance-as-material • 3800 images • class-weighted cross entropy loss

	RGB	RGBD
mIoU	0.506	0.502
fIoU	0.818	0.820
accuracy	0.888	0.888
micro-averaged precision/recall/F1	0.887	0.888
macro-averaged precision	0.669	0.675
macro-averaged recall	0.633	0.622
macro-averaged F1	0.618	0.612
weighted macro-averaged precision	0.882	0.882
weighted macro-averaged recall	0.898	0.899
weighted macro-averaged F1	0.889	0.889

Performance overview.

	RGB			RGBD			RGBD - RGB		
	IoU	Precision	Recall	IoU	Precision	Recall	IoU	Precision	Recall
Overall	0.506	0.669	0.633	0.502	0.675	0.622	-0.004	+0.006	-0.011
Weighted overall	0.818	0.882	0.898	0.820	0.882	0.899	+0.002	0.000	+0.001
asphalt	0.787	0.834	0.933	0.793	0.836	0.939	+0.006	+0.002	+0.006
foliage	0.894	0.943	0.945	0.891	0.937	0.948	-0.003	-0.006	+0.003
sky	0.972	0.985	0.986	0.973	0.983	0.989	+0.001	-0.002	+0.003
brick	0.805	0.909	0.875	0.819	0.922	0.880	+0.014	+0.013	+0.005
other	0.916	0.950	0.962	0.923	0.959	0.961	+0.007	+0.009	-0.001
steel	0.659	0.795	0.794	0.680	0.805	0.814	+0.021	+0.010	+0.020
tile	0.494	0.718	0.613	0.457	0.703	0.566	-0.037	-0.015	-0.047
grass	0.641	0.718	0.857	0.617	0.722	0.808	-0.024	+0.004	-0.049
stone	0.336	0.550	0.464	0.331	0.505	0.490	-0.005	-0.045	+0.026
soil	0.307	0.570	0.400	0.320	0.566	0.425	+0.013	-0.004	+0.025
concrete	0.000	0.000	—	0.000	0.000	—	0.000	0.000	—
plastic	0.300	0.632	0.364	0.177	0.459	0.223	-0.123	-0.173	-0.141
water	0.294	0.476	0.435	0.200	0.685	0.220	-0.094	+0.209	-0.215
iron	0.109	0.382	0.132	0.199	0.444	0.265	+0.090	+0.062	+0.133
living	0.298	0.539	0.400	0.298	0.590	0.376	0.000	+0.051	-0.024
aluminium	0.291	0.700	0.333	0.352	0.686	0.419	+0.061	-0.014	+0.086

Per-class results.

Predicted labels

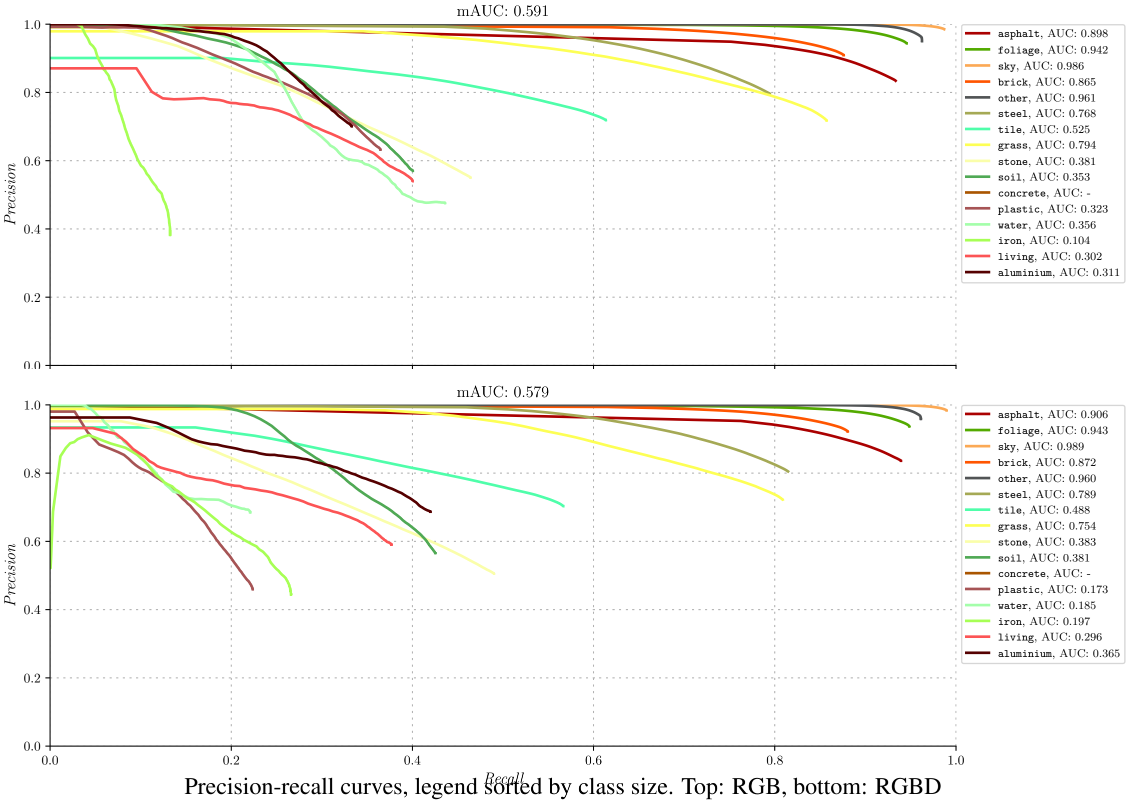
True labels

	aluminium	asphalt	brick	concrete	foliage	grass	iron	living	other	plastic	sky	soil	steel	stone	tile	water	Total
aluminium	14743	140	20401	0	6085	216	0	233	410	3154	923	93	5724	119	135	0	52376
asphalt	75	4898245	122787	0	1762	3519	504	354	16554	299	0	2242	10702	29793	161462	0	5248298
brick	1998	441075	5910626	5842	59791	9385	838	3584	7838	9769	37369	2445	92705	23332	139626	1439	6747662
concrete	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
foliage	602	2329	76099	0	3013059	22405	30	155	93	756	34954	3582	26187	3325	2854	934	3187364
grass	0	1603	3132	0	19852	251385	237	0	1	0	0	6521	3694	3608	2841	403	293277
iron	0	8827	1326	0	84	87	2686	0	0	162	0	113	406	4869	1716	0	20276
living	8	1097	5863	0	254	0	0	13808	1327	1282	0	0	10296	219	354	0	34508
other	0	15250	10390	0	1010	133	0	109	867784	771	95	334	3255	1299	1172	0	901602
plastic	986	2396	38664	54	4194	983	106	20	286	46798	1878	433	24199	138	7187	0	128322
sky	84	2	26034	2153	40830	0	0	0	0	1036	5560724	0	3798	27	0	0	5634688
soil	0	3120	3871	0	8606	34759	0	48	105	3	0	44054	5650	4181	5498	96	109991
steel	1280	14754	135415	47	26839	8254	1010	5147	4173	7480	8786	2573	958431	11425	20683	51	1206348
stone	701	71285	34102	0	3786	8899	748	401	7246	1187	299	2822	25511	199414	73019	242	429662
tile	21	410644	109667	0	4693	10025	858	1712	7375	1269	0	12013	32241	80294	1066690	868	1738370
water	0	0	848	0	1795	61	0	0	6	1	0	9	1581	0	439	3664	8404
Total	20498	5870767	6499225	8096	3192640	350111	7017	25571	913198	73967	5645028	77234	1204380	362043	1483676	7697	25741148

True labels

	aluminium	asphalt	brick	concrete	foliage	grass	iron	living	other	plastic	sky	soil	steel	stone	tile	water	Total
aluminium	18518	71	16060	0	5565	220	0	18	329	3672	962	105	6644	80	132	0	52376
asphalt	3	4929819	81002	0	2344	3654	2440	294	17522	407	0	1297	14671	36601	158244	0	5248298
brick	3079	382374	5941026	29456	67325	9021	556	1928	4683	12300	37885	4936	64721	33239	154584	549	6747662
concrete	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
foliage	1241	1465	67623	1044	3022923	16295	0	108	304	413	41059	4259	26096	1456	3062	16	3187364
grass	0	2742	1477	0	27531	237211	299	0	2	0	0	13841	3283	3783	2945	163	293277
iron	0	6873	1676	0	664	137	5391	0	0	37	0	59	155	3409	1875	0	20276
living	10	866	8432	0	522	1	0	13004	986	1717	0	0	8264	383	323	0	34508
other	0	16746	9824	0	1035	156	13	1	866632	1570	118	324	2834	893	1456	0	901602
plastic	453	4310	40523	0	4507	638	23	42	116	28715	1734	715	39010	1842	5694	0	128322
sky	193	0	17582	306	33805	0	0	0	11	892	5575788	0	6103	8	0	0	5634688
soil	0	2060	2303	0	13457	29114	30	37	66	17	0	46771	5808	4677	5651	0	109991
steel	2701	11078	109298	2976	34199	8955	787	3911	4018	8735	8669	627	982695	8962	18737	0	1206348
stone	112	68617	31747	0	3159	11231	1331	234	3381	2253	469	2325	31533	210555	62715	0	429662
tile	18	467988	112015	0	6340	11127	1267	2450	5621	1598	0	7367	27380	110496	984578	125	1738370
water	31	135	1187	3	2779	563	0	0	0	148	0	0	1462	0	240	1856	8404
Total	26359	5895144	6441775	33785	3226155	328323	12137	22027	903671	62474	5666684	82626	1220659	416384	1400236	2709	25741148

Confusion matrices. Left: RGB, right: RGBD



Precision-recall curves, legend sorted by class size. Top: RGB, bottom: RGBD