Master's Thesis

Foramina Localization

Semantic Segmentation of Neural Foramina on Lumbar Spine MRIs with Convolutional Neural Networks

AUTHOR
K. Lamerigts

FIRST SUPERVISOR
dr. A.J. Feelders

SECOND SUPERVISOR
prof. dr. A.P.J.M. Siebes

EXTERNAL SUPERVISOR
ir. M.J. Harte

July 7, 2017

ABSTRACT

Lumbar foraminal stenosis is the diseases of compressed nerve roots in the lumbar foramina and can cause great pain with the patient. Traditionally, this disease is diagnosed by a radiologist on lumbar MRI scans. In the past decade *computer aided diagnosis* (CAD) has made its rise due to the recent successes of deep neural networks. Especially in the domain of automatic medial image analyses CAD has seen a tremendous growth of interest among researchers. This work presents a deep neural network that automatically localizes the lumbar foramina from MRI scans. The pipeline consists of two stages, first a semantic segmentation on a MRI volume is performed, and secondly the coordinates of the foramina are determined. Contrary to past research, our network makes no use of data driven postprocessing techniques or hand-crafted features.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The branch of medical science that uses medical imaging techniques to diagnose diseases is called *radiology*. Interpreting such images is a hard task that is usually not carried out by the treating physician, but by a specialized *radiologist*. A radiologist interprets the image and consults the physician on further proceedings and possible treatments. The advances in various science fields have been very beneficial to radiology. Based on methods from computer science, physics, and engineering numerous modalities for medical imaging have been developed over the years. Currently, radiologists have a large number of modalities at their disposal. Among the common modalities used in hospitals are magnetic resonance imaging (MRI), computed tomography (CT), ultrasound, and positron emission tomography (PET).

Radiologists process a large number of scans. Due to the advancement of technology the amount of data per scan is increasing [1]. For example, MRI scans are performed with less spacing between the slices. Therefore, the workload of radiologists has steadily been increasing [2]. Despite the fact that the prediction for a radiologist shortage [3] has not come true [4], this trend still continues.

Increased computational power has made *computer-aided diagnoses* (CAD) possible and in the past decades CAD has seen a tremendous growth. Presently, CAD has evolved to one of the major research subjects in medical imaging [5, 6]. CAD is especially suitable for radiologists since they mostly analyze images, thus there are no additional factors that complicate the process. It has the potential to both reduce the number of errors and speed up the diagnosis by localizing objects, segmenting images, or even diagnosing diseases. Particularly the potential speedup is much needed due to the increasing workload of radiologists [2]. In some areas CAD already rivals the performance of trained radiologists [6]. Governments also seem to have picked up on the importance of CAD. The Japanese Ministry of Economy, Trade and Industry proclaimed in their *Technology Strategy Map 2010* that the development of CAD is an important issue [7].

The need for after-hour radiologists has spurred the appearance of *teleradiology* [8]. Teleradiology is radiology done by a radiologist at a different site than where the image was taken. Companies who offer teleradiology as a service are relatively new, but their use has been increasing rapidly [7, 9]. Additionally, such companies can offer their service using a pay-per-scan business model causing the costs for hospitals that have a low volume of scans to be lower than when an in-house radiologist is hired. Furthermore, not every hospital can have an in-house specialized radiologist for all fields. With the aid of teleradiology hospitals can offer health care with better quality by letting specialized radiologists diagnose complicated cases. Teleradiology is an excellent candidate for integrated CAD systems, since their success already depends on solid technological foundations.

CAD makes extensive use of *machine learning* [10]. Machine learning algorithms detect regularities from data and use those to make predictions on new data. Data driven predictions are useful for tasks where there is no clear cut method by means of traditional algorithms. One might say these are problems that require human intuition. For example, there is no way to determine how to predict the price of a house other than learning it from available data. Machine learning algorithms automate this process.

Machine learning algorithms rely heavily on the representation of the data. This representation usually consists of derived properties from the data. When predicting whether a customer will buy a certain product or not from its purchase history, the raw data is hard to interpret. The algorithm does detect regularities from the purchase history directly, but rather via features engineered by humans, often domain experts. A good feature for example, is whether the customer has previously bought a product from the same brand. The hardest part of applying machine learning to real problems (provided academia already designed good machine learning algorithms) is the design of successful features [11]. However, for some data it is hard to define features. In these cases even the crafting of features requires human intuition and cannot be programed directly. This is for instance the case with image classification. A human has no problem recognizing a wheel (the presence of a wheel is a very useful feature for image recognition), yet to define a wheel in terms of pixels is nearly impossible.

There are machine learning algorithms with a layered approach, that allows them to iteratively modify the input before performing the final task. Recently, this kind of machine learning has regain traction in the community. *Deep learning* [12] uses many hierarchical layers. It concerns the training of *deep neural networks* (DNN). Each layer in a DNN transforms the data from the previous layers. This way high-level features are generated from lower-level features. In the case of image classification a network first detects edges based on the pixel values, from these edges corners and contours are generated, which in turn are used to detect object parts. Finally, based on the highest level of features a classification is made. Real world DNNs have many more layers, ranging from 12 to 1000.

Deep learning achieved great successes in many tasks, in particular tasks that require the interpretation of images. This is due to the fact that it is hard to express image features in terms of numbers (pixel values), and therefore a difficult problem for non-hierarchical machine learning algorithms. Medical imaging is concerned with generating and analyzing medical images. Evidently, deep learning has lots of potential in the field of medical imaging and radiology. In fact, many medical imaging groups are urging to implement deep learning in their models, with promising results [13].

In collaboration with Aidence[1] we aim to design a deep neural network that can localize *lumbar foramina* from MRI scans. Current approaches use machine learning based postprocessing techniques. Our intent is to design a network that can do without complex postprocessing. This will make the network more robust to changes in the requirements and more broadly applicable. The question we will answer in this study is as follows: *Is it possible to design and train a deep neural network that has viable performance on the task of lumbar foramina localization on MRI scans without using data driven postprocessing techniques or hand-crafted features?* We consider what is means to be viable in the task description (Ch. 5).

1.1 STRUCTURE

In order to understand the usefulness of localizing foramina in MRI scans, the reader is required to have a basic understanding of the lumbar spine. Moreover, to interpret the dataset the reader is has to be familiar with the DICOM format. A short overview of these subjects is given in Ch. 2. In Ch. 3 techniques that are commonly used in DNNs are introduced. This includes how to train, regularize, and optimize DNNs in general. Regarding image processing, a specific type of DNN has proven to be effective. This so called *convolutional neural network* (CNN) is described in Ch. 4. The literature on CNNs is extensive. There are many architectures published for classification [14] and segmentation [15] among other tasks. A selection of noteworthy architectures is included in Ch. 4.

After a comprehensive overview of the required knowledge, we explain the task and research question in more detail in Ch. 5. Furthermore, we show the role of this project in the big picture. There have been works published over the last few year that have overlap with our task or approach, a short overview of these works are described in Ch. 6. In Ch. 7 we present our methodologies, experiments and results. Finally,

1 http://www.aidence.com/

we conclude and provide a discussion with issues that arise from our results in Ch. 8.

2 MEDICAL BACKGROUND

To understand the methods and techniques in this thesis, the reader is required to be familiar with some medical concepts. This chapter brings the reader up to speed in this regard. In Sec. 2.1 lumbar foraminal stenosis and a grading system are introduced. Lumbar foraminal stenosis is diagnosed by radiologists using MRI scans [16]. How MRI scans work and what they show, together with the structure and a few relevant meta tags of DICOM files, is explained in Sec. 2.2.

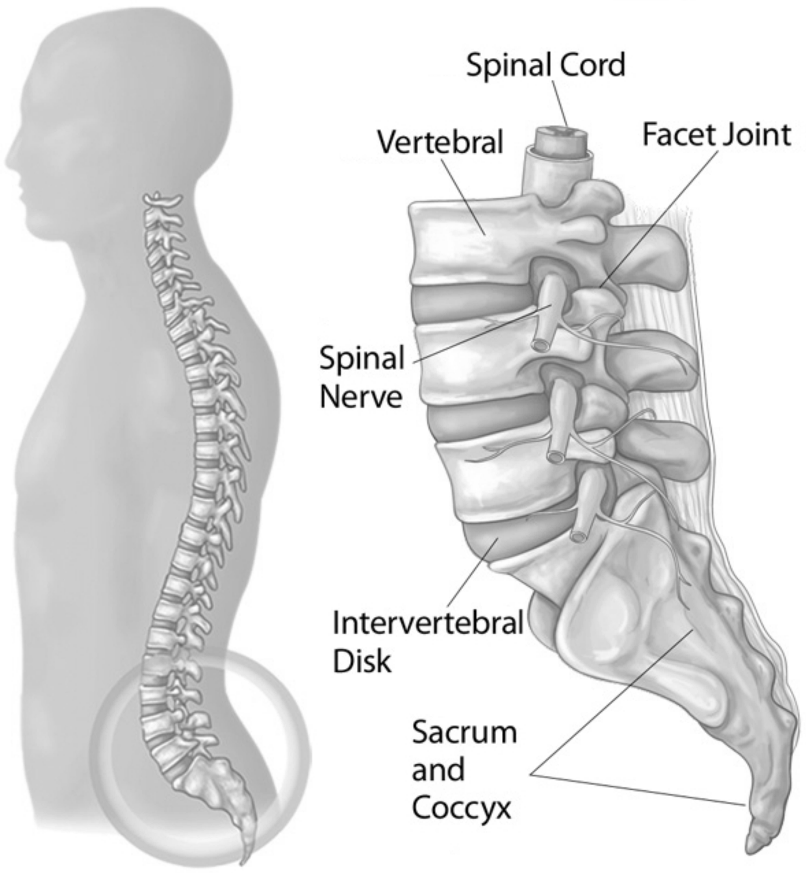


Figure 2.1: A close-up of the L3, L4, and L5 vertebrae.
Image taken with permission from http://www.orthoinfo.org/.

2.1 LUMBAR FORAMINAL STENOSIS

The spine connects the brain to the nerves in the rest of the body. A spine consists of *vertebrae* that are separated by *intervertebral discs* and connected towards the back by *facet joints* (Fig. 2.1). The five lowest vertebrae form the *lumbar spine*, more commonly known as the lower back. They are top-down labeled as L1-5. Below the lumbar spine starts the *sacrum*. The sacrum also consists of vertebrae, the top vertebral is labeled as S1.

In the vertical direction, the spinal cord runs through a hole between the vertebral body and the facet joint, the *spinal canal* (Fig. 2.2). At each level the nerves split of the spinal cord and travel trough a hole in the horizontal direction, the *intervertebral-* or *neural foramen*. There are two intervertebral foramina per vertebral, one on the left and one on the right. Nerves that exit from the lumbar spine innervate the lower limbs.

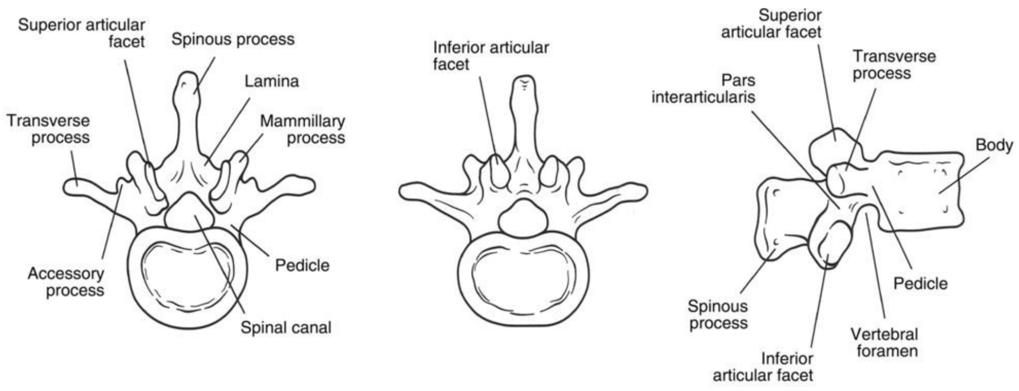


Figure 2.2: The axial and sagittal view of a lumbar vertebral.

Two adjacent lumbar vertebrae together with their intermediate intervertebral disc form a *motion segment*. For example the motion segment at L4/L5 consists of the L4 vertebral, the L4 intervertebral disc, and the L5 vertebral. A foramina is indicated by the top vertebral of its motion segments. So, left and right foramina that are formed by the L4/L5 motion segment, are indicated as the left and right L4 foramina.

The nerves in the spinal cord (*traversing nerves*) split off around the height of the upper vertebral body center in the segment. Subsequently, they (*exiting nerves*) travel in a downward and sideways motion towards and trough the foramen. Exiting nerves pass trough three zones (Fig. 2.3). After they split of they enter the *lateral recess* or *nerve root canal* (Fig. 2.3a), the area between the spinal canal and intervertebral foramen. The zone between the two facet joints is the *foramen* or *vertical interpedicular zone* (Fig. 2.3b). Finally, the area after exiting the foramen is the *extraforaminal zone*.

By compression or even contact with other tissue, nerves can become irritated. Irritated nerves cause pain to the patient. When a nerve root

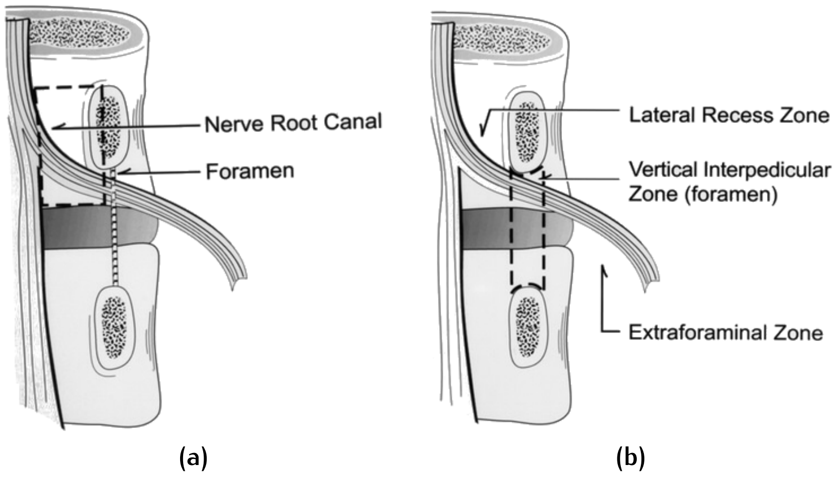


Figure 2.3: Coronal view of the zones that an exiting nerve passes trough [17].

in the lumbar spine is irritated, the effect (e.g. pain) is felt in the area that is innervated by the respective nerve [18]. Usually the nerves at level L5/S1 or L4/L5, which innervate the tissue below the knee, get irritated. But is it not uncommon for nerves to be irritated at the three higher levels, which innervate the knee and upper leg [19–21].

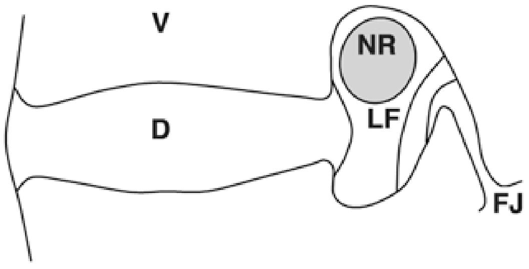
As said above, nerves become irritated when they are compressed or touched. This happens when the space in which they are located, is severely narrowed and therefore the nerves have no *perineural fat* (the layer of fat around the nerve) left. This disease is called *lumbar spinal stenosis*. There are three primary forms of stenosis. *Central*, *lateral*, and *foraminal stenosis* indicate stenosis in the spinal canal, lateral recess, and intervertebral foramen, respectively. There are multiple causes for stenosis.

First of all *facet arthrosis* [22], as humans get older the facet joint cartilage deteriorates. Cartilage is soft tissue that protects the facet joint on places where it touches other bone. Facets depend on the cartilage to stay flexible and mobile. To counter the deterioration of cartilage and relief pressure the facet joint creates a bigger surface by growing more bone. This extra bone can travel into the intervertebral foramen or the lateral recess, reducing its size.

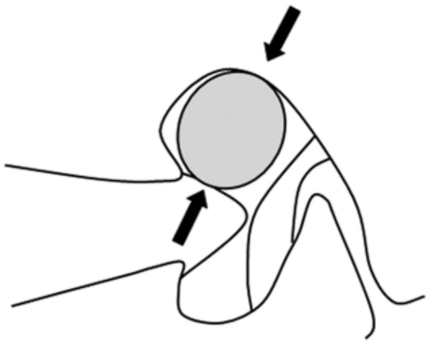
Secondly, two possible diseases that affect the intervertebral disk can cause stenosis. A intervertebral disc consist of an inner and outer substance, the *nucleus pulposus* and *annulus fibrosus* respectively. The annulus can ‘bulge’ into the spinal canal and intervertebral foramen, possibly causing stenosis. If the annulus has a tear it is a *hernia* [23]. This allows for the softer inner portion to prolapse out of the disk. A propulsion could cause stenosis depending on where the tear is located.

Finally, *spondylolisthesis* [24, 25] and *scoliosis* [26] can both deform the spine in such a way that the intervertebral foramina get narrowed.

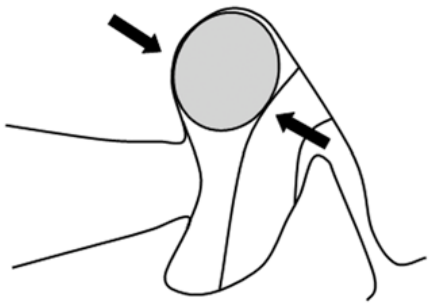
Spondylolisthesis is the situation where a vertebral slips forward over the vertebral below. This reduces the size of the foramina both sides, depending on how far the vertebral slipped. Scoliosis is the condition where the spine is deformed with a sideways curve. Again depending on the severity, this could cause stenosis.



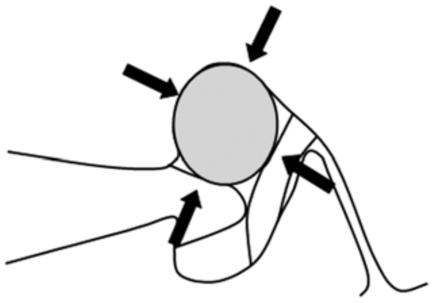
(a) Grade 0



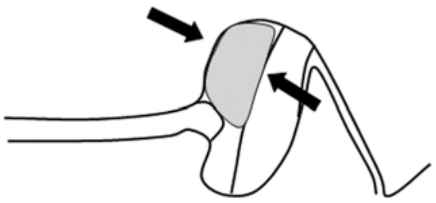
(b) Grade 1



(c) Grade 1



(d) Grade 2



(e) Grade 3

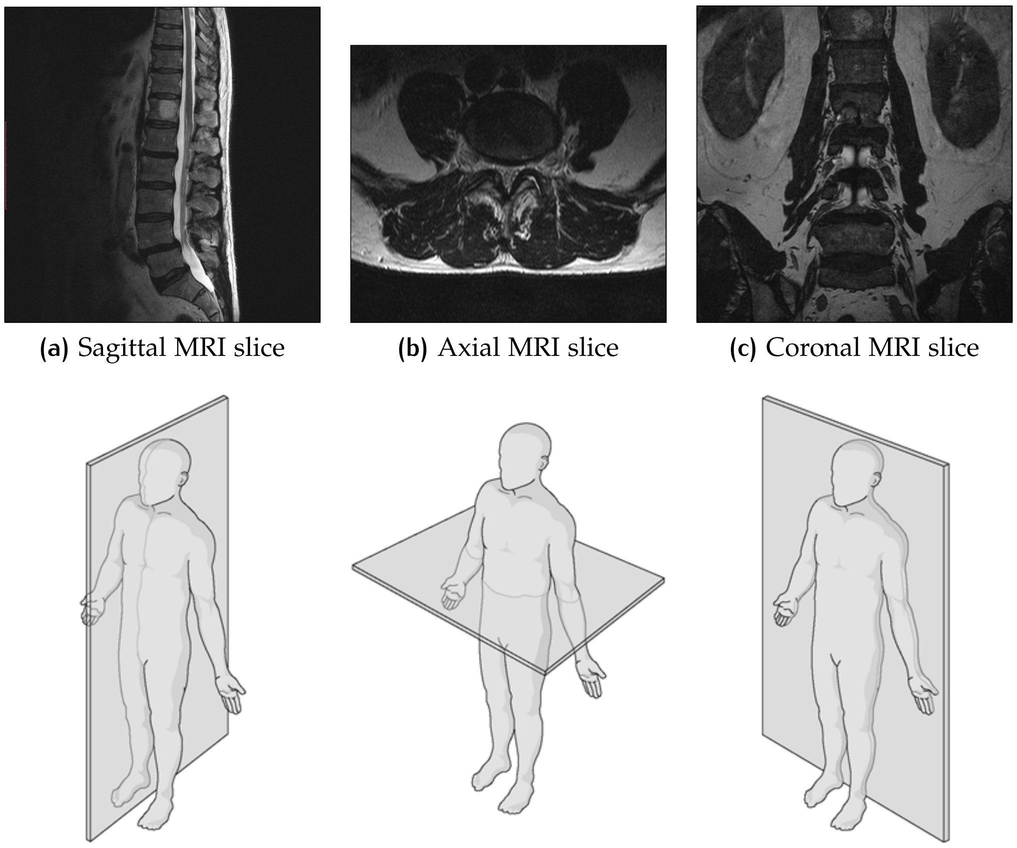
Figure 2.4: Schematic overview of the grading system from Lee *et al.* [27] for foraminal stenosis.

V: Vertebral body. D: Intervertebral disk. NR: Nerve root. LF: Ligamentum flavum. FJ: Facet joint.

2.1.1 Grading system

Since there are a multitude of possible causes, it is ambiguous how to grade the amount of stenosis that occurs [27–29]. Lee *et al.* [27] recently designed a grading system that can be applied to the results of the classic MRI protocol for the lumbar spine, i.e. it does not require altering the MRI protocol. Furthermore, it has a higher interobserver agreement value than other grading systems [30]. The system consist of four grades (Fig. 2.4).

Grade 0 indicates that no lumbar foraminal stenosis occurs. Grade 1 indicates mild foraminal stenosis. The perineural fat is obliterated in either the vertical (Fig. 2.4b) or transverse (Fig. 2.4c) direction. When the perineural fat is obliterated in both directions the grade indication is 2, referring to moderate foraminal stenosis. Finally, grade 3 indicates severe foraminal stenosis where the nerve root shows morphological changes.



(a) Sagittal MRI slice (b) Axial MRI slice (c) Coronal MRI slice

(d) Sagittal plane (e) Axial plane (f) Coronal plane

Figure 2.5: A T2 weighted MRI slice and plane visualization for each of the three body axes (sagittal, axial, and coronal).

2.2 MAGNETIC RESONANCE IMAGING

Magnetic resonance imaging (MRI) is one of the most commonly applied techniques to map the human body in hospitals. It requires no radiation, is able to visualize all three planes (Fig. 2.5), and provides sufficient details of the anatomy. To achieve this a MRI scanner utilizes a very powerful magnet and the properties of atomic nuclei. Normally, the orientation of atomic nuclei is random. But within a strong uniform magnetic field the protons in water atoms align with the magnetic field, this is called *magnetization*. By sending radio frequency (RF) signals, the magnetization gets disrupted. Subsequently, by realigning with the magnetic field, the protons send out the received radio frequency signals (echoes). The echoes are captured and analyzed, from the analysis an image is formed. Usually multiple RF's are applied, the time between these pulses is the *repetition time* (TR). The time between sending the RF and capturing the echo is the *time to echo* (TE).

By altering the TR and TE parameters the echoes change, thereby changing the difference that can be seen between tissues (Tab. 2.1, 2.2). There are two typical parameter settings; *T1-* and *T2-weighted* scans. T1- and T2-weighted scans use short and long TR and TE times, respectively. As a consequent, tissues that might be hard to distinguish in T1-, can be easy to distinguish in T2-weighted scans and vice versa.

TIME	T1-WEIGHTED	T2-WEIGHTED
Repetition time	500	4000
Time to echo	14	90

Table 2.1: The TR and TE in miliseconds for T1- and T2-weighted scans.

TISSUE	T1-WEIGHTED	T2-WEIGHTED
Cerebrospinal fluid	Dark	Bright
Muscle	Gray	Dark gray
Spinal cord	Gray	Light gray
Fat	Bright	Light
Intervertebral disc	Gray	Bright
Air	Very dark	Very dark

Table 2.2: The tissue colors for T1- and T2-weighted scans.

3

DEEP NEURAL NETWORKS

Machine learning is a set of automated techniques that can recognize patterns from data and make predictions about new data. In other words, machine learning algorithms are able to learn from data. Their performance at a certain task increases the more experience they have. In the case of machine learning algorithms, experience means processing an example. The techniques used in machine learning are closely related to statistics, but have more emphasis on using computations (i.e. with computers) to estimate complicated functions.

Traditional machine learning techniques require a human (often a domain expert) to carefully engineer relevant features. The raw data is transformed into a internal feature space from which the machine learning algorithm can learn patterns. In some cases it is hard for humans to hand-craft features and at the same time traditional machine learning algorithms lack the ability to extract patterns from the raw data directly.

The approach to let an algorithm learn to map the data to an internal feature space by itself, is called *representation learning*. *Deep learning* uses multiple layers of representations[2] where each layers builds on the previous one. By stacking multiple layers complex functions are approximated. Ideally, each layer learns features that are from a high abstraction level than the previous layer. For example, given a network that has to classify images, the first layer could learn basic geometric shapes such as edges and curves. The second layer uses such shapes learn more complex shapes such as circles and squares. The main idea of deep learning is that the all layers will learn these features by itself from the raw data.

While training a deep neural network, several steps are executed:

1. Process a mini-batch with a forward pass.
2. Calculate the loss.
3. Calculate the gradient with backpropagation in a backward pass.
4. Update the weights with the gradient.

We will call an execution of these four steps an *iteration*. Deep neural networks require many iterations to perform well. In this chapter each

2 It is undefined what number of layers is required for a network to be called deep. Let us assume that a network is deep if it has more than 3 layers.

of these steps is explained. Furthermore, the problems that arise are pointed out, together with their corresponding solutions.

To understand why a mini-batch instead of a full batch is processed, we first provide a short introduction to *gradient descent*, the most common machine learning optimization technique. Subsequently, the three variants of gradient descent are explained.

The core of the problems that arise specifically in deep neural networks, comes from the way neural networks determine their gradient. This algorithm is called *backpropagation* and is explained in Sec. 3.2. The problem that stems directly from the backpropagation is called the *unstable gradient problem*. Which is, together with possible solutions, discussed in Sec. 3.3.

Since the rise of deep learning, more sophisticated weight update algorithm have been developed. In Sec. 3.4 we give an overview of relevant algorithms. To prevent overfitting of the network, a few common regularization techniques can be applied. Two of these techniques are described in the final section of this chapter.

3.1 GRADIENT DESCENT

Training deep neural networks involves optimization. In this context optimization refers to the task of minimizing of maximizing a function $f(w)$ by modifying w. One of the most common optimization techniques for machine learning is *gradient descent*. In most implementations gradient descent is designed to minimize $f(w)$, for maximization one can minimize $-f(w)$.

The derivative of $f(w) = y$, denoted as $f'(x)$ or $\frac{dy}{dw}$, is useful for minimizing $f(w)$ because it gives information on how to change w for a small reduction in y. If many small changes are made to w successively, a (local) minimum for y will be reached. For function $f(\mathbf{x}) = y$ with more than one input variable, we take the partial derivative with respect to each input variable w_i, $\frac{\partial}{\partial w_i} f(w)$. The vector of partial derivatives for each variable is called the *gradient* and is denoted as $\nabla f(\mathbf{x})$.

We denote the loss function for data $\mathbf{x}$, function f, and targets $\mathbf{y}$ as $L(f(\mathbf{x}), \mathbf{y})$. Alternatively, we use denote the loss as a function of the weights $\boldsymbol{w}$ at step t; $L(\boldsymbol{w}_t)$ where the function, data, and target values are left out for simplicity. Each time the weights are updated, only a small step is taken into the direction of the gradient. The step size is called the *learning rate* and denoted as η. An update of the weights is now defined as follows:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \cdot \nabla L(\boldsymbol{w}_t) \tag{3.1}$$

There are three variants of gradient descent, they differ in the number of samples used to determine the gradient. *Batch* gradient descent uses all available (trainings) samples from $\mathbf{x}$. *Stochastic* gradient descent, on the other hand, determines the gradient based on only one sample $x_i \in \mathbf{x}$. As a result the gradients are based on less accurate information and will be more noisy overall. Even though the gradients are more noisy, on average they will 'point' in the right direction. Additionally, the calculation of the gradients will be a lot faster. A trade off between speed and noisiness is made by increasing the number of samples. When more than one, but not all, samples are used we call it *mini-batch* gradient descent[3].

Since the loss function is non-convex, we would normally worry about local minima and take precautions to prevent trapping the network. However, as empirical evidence suggest, for high-dimensional functions local minima are not as big of a problem as they are for lower dimensional functions [31]. That is not to say there are no local minima in high dimensional non-convex functions, but rather that all minima will have more or less the same value.

3.2 BACKPROPAGATION

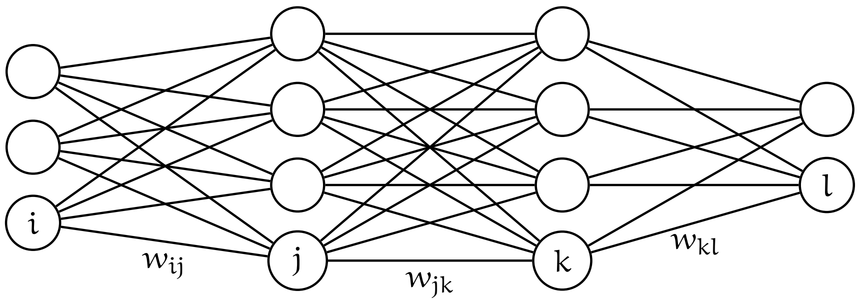
Directly applied, gradient descent is not feasible for deep neural networks. This is is due to the fact that the gradient is computed by taking the partial derivative of the loss function with respect to a single weight $\nabla_{w_i} L = \partial L / \partial w_i(\boldsymbol{W})$, for all weights $w_i \in \boldsymbol{W}$. Deep neural networks can have up to millions of weights, it is intractable to compute a partial derivative for each weight every iteration. Multilayer networks can compute the gradient for each weight by recursively backpropagating the *error signal* trough the network. This algorithm is called *backpropagation* [32]. The key insight to backpropagation is that a derivative of the error signal can flow back trough the network by means of the chain rule. This requires less computations and therefore faster.

During the forward pass we do a weighted sum over the connections to a node i to compute its pre-activation z_i. Its output is the activation function φ applied to the pre-activation, $y_i = \varphi(z_i)$. In this manner we pass over the whole network and calculate the predictions $\hat{y}$ in the output nodes. The gradient of output node l is the loss of the prediction $\partial L / \partial \hat{y}_l = L(\hat{y}_l, y_l)$. The gradient indicates what the slope of the loss given w.r.t the weights is. Following the slope would increase the loss, thus we following the opposite direction of the slope. In other words,

3 In most literature mini-batch gradient descent is refered to as stochastic gradient descent.

Forward:

$$z_j = \sum w_{ij} x_i \qquad z_k = \sum w_{jk} y_j \qquad z_l = \sum w_{kl} y_k$$
$$y_j = \varphi(z_j) \qquad y_k = \varphi(z_k) \qquad \hat{y}_l = \varphi(z_l)$$



Backward:

$$\frac{\partial L}{\partial y_j} = \sum w_{jk} \frac{\partial L}{\partial z_k} \qquad \frac{\partial L}{\partial y_k} = \sum w_{kl} \frac{\partial L}{\partial z_l} \qquad \frac{\partial L}{\partial \hat{y}_l} = L(\hat{y}_l, y_l)$$
$$\frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial z_j} \qquad \frac{\partial L}{\partial z_k} = \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial z_k} \qquad \frac{\partial L}{\partial z_l} = \frac{\partial L}{\partial \hat{y}_l} \frac{\partial \hat{y}_l}{\partial z_l}$$

Figure 3.1: Equations of the for- and backward pass of a multilayer neural network.

the gradient gives information whether the output should be higher or lower than the current prediction value. To compute the gradient of the other nodes in the network we use the chain rule. The gradient of a node on the loss is the weighted sum of the gradients of its outgoing connections $\partial L/\partial y_i = \sum w_{ij} \partial L/\partial z_j$. To compute the gradient of each weight we first compute the influence of each connection to the output $\partial y_i/\partial z_i = \phi'(z_i)$. With the gradient of its outgoing connection, its own influence on the output, and the chain rule at hand, the gradient of a weight is computed as $\partial L/\partial z_i = \partial L/\partial y_i \cdot \partial y_i/\partial z_i$. The weights are updated with the gradient from there *receiving* node, i.e. all weights to node k are updated with the error signal $\partial L/\partial z_k$. How the gradient is used to update the weights is discussed in Sec. 3.4.

3.3 UNSTABLE GRADIENTS

Deep neural networks have more layers than shallow neural networks. As a results numerous computations transform the activation and gradient during the forward and backward pass respectively. Historically, the *sigmoid* function is commonly used as activation function in the hidden layer. However, for deep neural networks the sigmoid function causes a problem.

Consider a deep neural network, the weights are initialized randomly from a Gaussian distribution[4], and the activation function for nodes is the sigmoid function. During backpropagation the gradient flows back trough all nodes. The derivative of the sigmoid function, $\text{sig}'(x) = \text{sig}(x)(1-\text{sig}(x))$, has a maximum value of 0.25 (Fig. 3.3). Consequently, the gradient will at least decrease with 75% at each sigmoid it passes. After multiple layers the gradient is so small, that learning is no longer feasible. This is problem is commonly known as the *vanishing gradient problem* [33].

The vanishing gradient problem is not the only difficulty with the sigmoid function. Given that the weights are initialized from a Gaussian distribution with mean 0, the activations during the forward pass will either explode or vanish depending on the standard deviation and the number of nodes in the previous layer. The derivative of a sigmoid function with a low or high (saturated) input, has a low derivative value. We will this problem combined with the vanishing gradients, the *unstable* gradient problem.

There have been attempts to prevent unstable gradients by unsupervised pre-training [34, 35]. However, better methods to stabilize the learning process have been developed since. The first is to carefully initialize the weights such that the activations do not saturate. Secondly, numerous activation functions are designed to avert the problems of the sigmoid function. Finally, an extra type of layer is introduced, the *batch normalization* layer.

3.3.1 Weight initialization

Early deep neural networks had a hard time converging. Partially, this was due to uncareful initialization of the weights. If the weights are too small, the activations will decrease after each passing layer. Given a network with the sigmoid function, decreasing activations results in saturated activations. If there is little variance in the activations, they are basically linear. A similar situation appears when the weights are too big. The activations saturate to 1, with little variance and gradients close to zero.

3.3.1.1 *Xavier & He initialization*

Too avoid saturated activations, *Xavier* initialization [36] sets the weights in such a way that the variance before and after a sigmoid layer is the same. Weights are randomly sampled from a Gaussian distribution $\mathcal{N}(0, \text{Var})$, the variance Var differs per layer and is determined by its

4 Sampling from a Gaussian distribution is the commonly accepted way of initializing weights [12]

number of inputs $n_{l,in}$. We denote the initialization variance for the weights from layer l as $\text{Var}(W_l)$.

$$\text{Var}(W_l) = \frac{1}{n_{l,in}} \tag{3.2}$$

Xavier initialization (Eq. 3.2) is designed with a sigmoid function in mind. More recently, the ReLU activation function (Sec. 3.3.2.3) have become common in networks. Since a ReLU function returns 0 for roughly half its input space, *He* initialization [37] (Eq. 3.3) doubles the variance of the output to keep the variance of the activations before and after the ReLU layer the same.

$$\text{Var}(W_l) = \frac{2}{n_{l,in}} \tag{3.3}$$

The crucial observation made by Glorot and Bengio [36] is that whether the activations, amplify or dampen can be controlled with the variance of the distribution from which the weights are randomly sampled.

3.3.2 Activation functions

Activation functions add nonlinearity to neural networks. Since a combination of linear transformations (pre-activations) can be expressed as a single linear transformation, without the nonlinearities neural networks would have the same capacity as a linear classifier.

Choosing effective activation functions has proven to be difficult. At this moment, designing activation function that perform well is an active field of research.

3.3.2.1 *Sigmoid*

In addition to the problems of saturation, the sigmoid function operates only in the positive real number space $\mathbb{R}_{\geqslant 0}$. This means that the weights will only be updated with a positive value. Although this isn't a problem for learning theoretically, it is a problem in practice since the learning will take longer [12].

$$\text{sig}(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

3.3.2.2 *Softmax*

Contrary to the sigmoid, the *softmax* considers the whole vector. The values of the resulting vector will sum to 1, thus the softmax can be

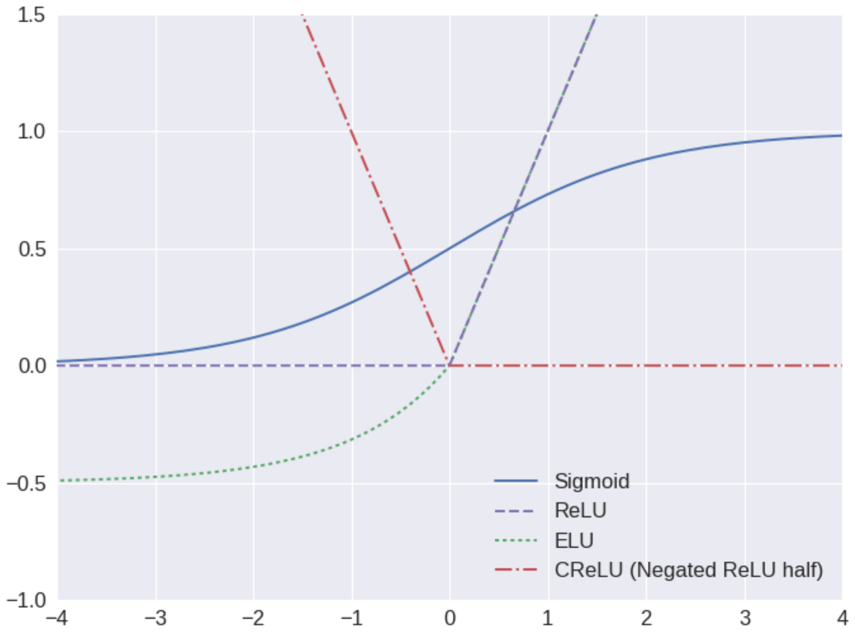


Figure 3.2: Several activation functions. ELU: $\alpha = 0.5$

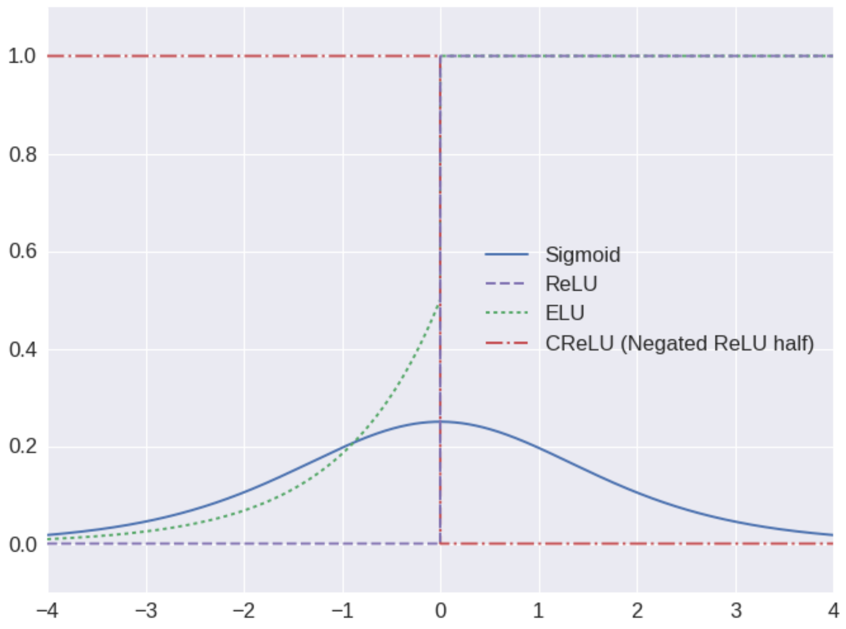


Figure 3.3: The derivatives of several activation functions. ELU: $\alpha = 0.5$

interpreted as a probability distributor. Assigning probabilities proportional to the corresponding input value.

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=0}^{n} e^{x_j}} \tag{3.5}$$

The softmax function is usually not used as activation in the network, but rather as an output activation. It converts the unscaled logits to a probability per class.

3.3.2.3 *Rectified linear unit*

One the big breakthroughs in making deep neural networks viable was the *rectified linear unit* (ReLU) [38]. The ReLU is a simple, yet effective and computationally efficient function. Intuitively, one might say that disallowing negative values hinders learning. However, empirical evidence shows that learning with ReLU activations can improve the converge time with a factor of 6 [39]. For positive values the gradient flows trough a ReLU node unchanged and does not vanish.

$$\text{relu}(x) = \max(0, x) \tag{3.6}$$

For all that, ReLUs are not zero centered and kill the gradient completely if the pre-activation of the node is lower than zero. As a consequence, if a ReLU node does not activate, it will not update its weights. It is possible that a ReLU is 'pushed' out of the range where it will ever activate, effectively killing the node.

3.3.2.4 *Parametric rectified linear unit*

To prevent ReLUs from dying one can use a *leaky* ReLU (LReLU) [40]. LReLUs do not cap negative values to zero, but multiple them with 0.01. This greatly reduces the influence of negative values, but allows for gradients to flow through and revive inactive nodes. The generalization of the LReLU is the *parametric* ReLU (PReLU) [37]. Instead of multiplying negative values with a constant, the PReLU uses a parameter a. This is differentiable, so the network can learn what is the best value to use. It does however, introduce an extra learnable parameter per node, making the network bigger.

$$\text{prelu}(x) = \begin{cases} x, & \text{if } x > 0 \\ ax, & \text{if } x \leqslant 0 \end{cases} \tag{3.7}$$

3.3.2.5 *Exponential linear unit*

Rather than multiplying the negative values with a constant as in PReLus, the *exponential linear units* (ELU) [41] saturates negative pre-activations at a learnable value a. The network can learn to set a at a high value, pushing the mean activation closer to zero. Additionally, for pre-activations lower than a a (small) gradient is propagated back, therefore giving the network the opportunity to revive the node if necessary. A disadvantage of ELUs compared to ReLUs, is the use of e^x, which is computationally expensive.

$$\mathrm{elu}(x) = \begin{cases} x, & \text{if } x > 0 \\ a(e^x - 1), & \text{if } x \leqslant 0 \end{cases} \tag{3.8}$$

3.3.2.6 *Concatenated rectified linear unit*

Another way of maintaining variance, yet keeping the activation non-linear, is to concatenate the positive and negative information from the pre-activation. The *concatenated* ReLU (CReLU) [42] does two ReLU computations; on the 'standard' pre-activation and on the negated pre-activation. In other words, it selects both the information from positive and the negative values. A major difference with other activation functions is that the CReLU gives two outputs. Thus, the number of activations doubles at each CReLU layer.

$$\mathrm{crelu}(x) = [\mathrm{relu}(x), \mathrm{relu}(-x)] \tag{3.9}$$

3.3.3 Other strategies

3.3.3.1 *Batch normalization*

A more radical approach to make activations follow a useful gaussian distribution, is to simply normalize the activations. *Batch normalization* [43] calculates the mean $\mu_\mathcal{B}$ and variance $\sigma^2_\mathcal{B}$ for activations $\mathbf{x} \in \mathbb{R}^m$ and normalizes them. Ideally, we want to let the network decide if batch normalization will help performance. Therefore, batch normalization introduces two learnable variables β and γ, which control the scale and variation of the output respectively. This means that the network has the ability to 'turn off' batch normalization in a certain layer or learn a better distribution. A fixed small variable $\epsilon = 1\text{e}{-6}$ is used to prevent division by 0.

$$
\begin{aligned}
\mu_{\mathcal{B}} &\leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \\
\sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \\
\text{bn}_{\text{train}}(x_i) &= \gamma\left(\frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right) + \beta
\end{aligned}
\tag{3.10}
$$

With batch normalization as defined in Eq. (3.10) the output depends on the batch that is processed. During inference however, the network should be deterministic. Without a deterministic network it is hard to measure its performance accurately. Besides, it is also desirable from an intuitive perspective that a network during inference gives the same output no matter what the other samples in the batch are. So, during training we keep track of a running average of $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$, denoted as $E[x_i]$ and $\text{Var}[x_i]$ respectively, and use those values during inference.

$$
\text{bn}_{\text{test}}(x_i) = \gamma\left(\frac{x_i - E[x_i]}{\sqrt{\text{Var}[x_i] + \epsilon}}\right) + \beta \tag{3.11}
$$

Batch normalization has made learning more stable. As a result, weight initialization techniques and activation functions require less careful consideration. Furthermore, the learning rate can be set higher since the gradient is more likely to point to the 'correct' direction. Because the activations are normalized (based on a randomly constructed batch), the network can no longer 'remember' the values, thus is less likely to overfit. This reduces the need for regularization. The disadvantage of batch normalization lies in the computational cost to normalize activations. However, the advantages typically outweigh the extra computational cost because of the faster convergence, resulting in less computation time overall.

3.4 OPTIMIZATION

The learning rate is a very important hyperparameter. If the learning rate is too small it will take a long time before the network converges, making training it infeasible. Too high, makes learning unstable because it cannot make fine-grained modifications to the weights. Usually the approach take is to set a quite high learning rate in the beginning and when learning stops (due to overshooting in the surface loss), lower the learning rate. Recently, more sophisticated weight update algorithms have been developed that increase convergence speed.

3.4.1 Momentum

The optimization problem can be seen from a physical perspective. Imagine the loss surface as a landscape and the loss as a ball in that landscape. In this metaphor the gradient is the slope of the surface. *Momentum* [44] accelerates learning by giving the gradient a velocity v as if it rolls down. Every time step a velocity is computed by adding a fraction γ (usually set to 0.9) from the previous velocity to current gradient. For dimensions where the gradient is going downward the velocity will increase, and the velocity will decrease for dimension where the gradient is going up.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla L(\boldsymbol{W_t}) \\ \boldsymbol{W}_{t+1} &= \boldsymbol{W}_t - v_t \end{aligned} \tag{3.12}$$

3.4.2 Nesterov momentum

To further increase convergence rate *Nesterov momentum* [45] looks ahead. Instead of slowing down when the ball (loss) is actually going up, Nesterov momentum approximates the next weight variables by subtracting the previous velocity from the weights. It can not accurately predict the weights because for that it needs the next gradient. Now the ball slows down when approaching a hill and speeds up when approaching a cliff.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla L(\boldsymbol{W}_t - \gamma v_{t-1}) \\ \boldsymbol{W}_{t+1} &= \boldsymbol{W}_t - v_t \end{aligned} \tag{3.13}$$

3.4.3 Adaptive gradient algorithm

Deep neural networks have multiple layers and a huge number of parameters in total. *Adaptive gradient algorithm* (Adagrad) [46] tries to make better updates by giving each parameter a individual learning rate. It keeps track of all past gradients per weight and updates this 'cache' $g_{t,i} \in \mathbf{G}_t$ (the cache for weight i at step t as element from the vector of $\mathbf{G}_t$) by adding the square of the current gradient to it. Subsequently, the learning rate is divided by this cache. As a results, parameters that get small or infrequent gradients will have their gradient scaled up, and parameters that get big or frequent gradients will have their gradient scaled down.

$$\begin{aligned} \mathbf{G}_t &= \mathbf{G}_t + (\nabla L(\boldsymbol{W}_t))^2 \\ \boldsymbol{W}_{t+1} &= \boldsymbol{W}_t - \frac{\eta}{\sqrt{\mathbf{G}_t} + \epsilon} \odot \nabla L(\boldsymbol{W}_t) \end{aligned} \tag{3.14}$$

Adagrad requires more memory than optimization algorithms that do not fine tune the learning rate per parameter, but when Adagrad is applied it tremendously increase the rate of converge [47]. However, for deeper neural network that have to train for a long time, Adagrad is too aggressive. Eventually every parameter has accumulated a big cache, effectively dispersing the learning rate for each parameter and stop learning.

3.4.4 Root mean square propagation

An unpublished method by Geoff Hinton from his Coursera course on neural networks, reduces the aggressiveness of Adagrad by making the cache leaky. *Root mean square propagation* (RMSprop) [48] only retains a portion γ (usually 0.9) of the previous cache value. Leaking the cache prevents the algorithm from stopping learning too early.

$$\begin{aligned} \mathbf{G}_t &= \gamma \mathbf{G}_t + (1-\gamma)(\nabla L(\boldsymbol{W}_t))^2 \\ \boldsymbol{W}_{t+1} &= \boldsymbol{W}_t - \frac{\eta}{\sqrt{\mathbf{G}_t} + \epsilon} \odot \nabla L(\boldsymbol{W}_t) \end{aligned} \tag{3.15}$$

3.4.5 Adaptive moment estimation

Adaptive moment estimation (Adam) [49] combines the ideas above into one algorithm. It keeps track of the exponentially decaying average of the past gradients $m_{t,i} \in \boldsymbol{m}_t$ (the mean) and the exponentially decaying average of the squared gradients $v_{t,i} \in \boldsymbol{v}_t$ (the variance), both in a leaky manner with β_1 and β_2 respectively.

$$\begin{aligned} \boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\nabla L(\boldsymbol{W}_t) \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)(\nabla L(\boldsymbol{W}_t))^2 \end{aligned} \tag{3.16}$$

The authors notice that after initialization both $\boldsymbol{m}$ and $\boldsymbol{v}$ are biased towards zero. This is especially true when the decay rates are low (i.e. β_1 and β_2 are close to 1). Therefore, $\boldsymbol{m}$ and $\boldsymbol{v}$ are normalized. The normalized values are used to update the weights. Since β_1 and β_2 are smaller than 1 and taken to the power of t this will only have (significant) influence when t is small, as intended.

$$
\begin{aligned}
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
W_{t+1} &= W_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t
\end{aligned}
\tag{3.17}
$$

3.5 REGULARIZATION

3.5.1 Data augmentation

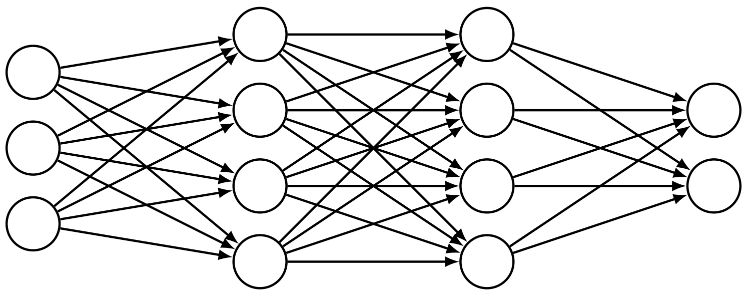
Ultimately, the best way to generalize better is to train on more data. Obviously, this is in practice never the solution, since we are most likely already using all the data we have. To still increase the training set *data augmentation* can be performed. The idea is simple, we consider all the variations that our network has to be robust against and apply such variations at random to our trainings samples.

3.5.2 Dropout

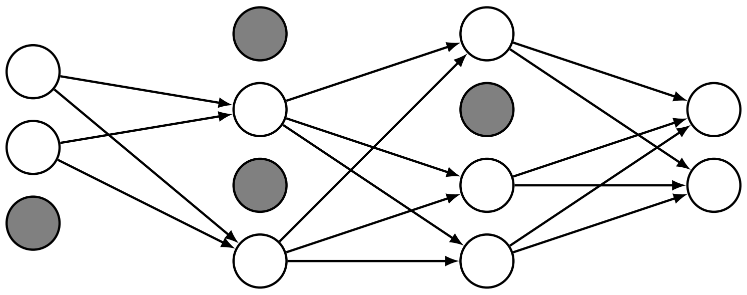
A *dropout* layer [50] sets the output of its nodes with a probability p to zero (i.e. it drops several nodes). Dropped nodes do not contribute to training during the forward and backward pass (Fig. 3.4). In a network without dropout layers the gradient tells how a node should change to decrease the loss given what all the other nodes are doing. This may lead to what Srivastava *et al.* [50] call *co-adaptations*. Nodes could learn to fix the mistake made by other nodes. Co-adaptations do not generalize well to unseen data. By dropping nodes at random, mistakes can not be fixed in a reliable way. Thus, nodes are not given the opportunity to let other nodes make up for their mistakes and will have to change themselves. At the same time it reduces the capacity of the network during training. Since networks with more parameters are more likely to overfit, reducing the capacity also reduces the chance of overfitting.

Using dropout in a single network is more or less the same as training several smaller networks (with shared parameters) and ensemble them during test time. Because during training nodes are dropped with probability p, nodes have less incoming connections than during test time. This skews the activation values. Therefore, the activations during test time should be scaled to simulate the activation size during training time. To that end, the output of a node is multiplied with $q = 1-p$.

Scaling the activations during test time works well, but imposes a problem. The probability p is a hyper parameter and can be changed. Each time we change p the network at test time changes. An alternative is too scale the activations down during training with 1/q. This way the difference in activation size is accounted for during training and eliminates the need to change the network at test time depending on how the network is trained.



(a) Before



(b) After

Figure 3.4: Possible effect of dropout with a probability of 0.5.

4 CONVOLUTIONAL NEURAL NETWORKS

A human eye catches incoming light on its retina. The brain processes this raw information and creates an interpretation of the image. By measuring the activity of a single neuron in the brain of a cat, Hubel and Wiesel [51] showed that neurons respond to contours with a specific orientation. This was one of the first steps towards an understanding of how the brain processes raw visual information. The authors hypothesized that the visual system of cats and primates is hierarchical [52, 53]. That is, complex neurons transform and depend on the output of simple cells. DNNs resemble such a hierarchy. However, in traditional DNNs layers are fully connected. For visual systems the spatial information is important and neurons in the brain seem to retain this information [51]. The part of the incoming image (light caught on the retina) that excites a neuron is called its *receptive field*.

The earliest attempt to model a hierarchical visual system in the form of a neural network was as early as 1982, called the *neocognitron* [54]. Even tough the findings of Hubel and Wiesel [51] inspired the modelling of the neocognitron, neuroscience provides little guidance on how to train such a model. As a result, training a neocognitron proved to be difficult and made the neocognitron as a whole ineffective. The first network that was trainable (with the, by then discovered, backpropagation algorithm [Sec. 3.2]) was LeNet [55] which is commonly accepted as the first *convolutional neural network* (CNN). However, for reasons now known (unstable gradients [Sec. 3.3]) LeNet was not more successful than traditional computer vision techniques. The uprise of CNNs started with AlexNet [39]. It was the first truly successful CNN applied to image classification and object localization. Its success relied upon techniques to overcome the unstable gradient problem and the utilization of GPUs, yet is similar in architecture to LeNet.

In this chapter the core of a CNN; the *convolution* and its corresponding layer are analyzed. First we analyse the convolution as standalone operation and then how CNNs use convolutions in layers. In addition to the convolutional layer, the *pooling* and *fully connected* layers are commonly used in CNNs. These will also be explained. Furthermore, we have a closer look at two tasks that can be solved with CNNs; *image classification* and *segmentation*. Classification networks should learn a function that maps the input (pixels) to one of k possible categories, that

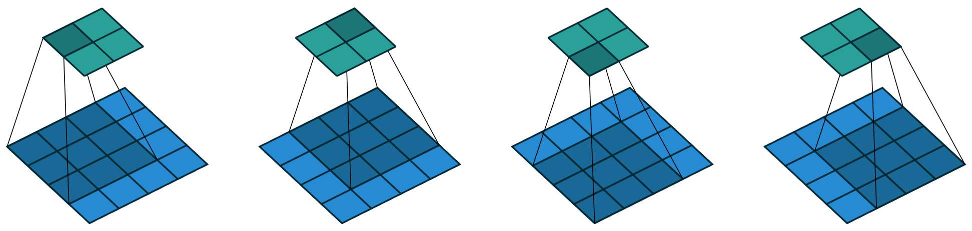


Figure 4.1: A 3×3 convolution on a 4×4 input with 'valid' padding and stride 1.

is $f : \mathbb{R}^n \times \mathbb{R}^m \to \{1, ..., k\}$. A few key architectures for classification are described.

Segmentation does something similar, but instead of classifying the whole image to one of k categories, it classifies each pixel to one of k categories, i.e. $f : \mathbb{R}^n \times \mathbb{R}^m \to \{1, ..., k\}^n \times \{1, ..., k\}^m$. Segmentation architectures make use of supplementary layers that are used for upscaling an input. These so-called *unpool* and *transposed convolutional* layers are first described before delving deeper into key architectures for segmentation.

4.1 THE CONVOLUTION OPERATION

In its essence a convolution operation is a linear transformation. It takes an input and a kernel matrix, and slides (convolves) the kernel over the input (Fig. 4.1). For each position is output one value y. This is calculated by taking the sum over the element wise multiplication of the kernel and the input window. Or in other words, the dot product of the vectorized kernel $\boldsymbol{W}$ and vectorized input window $\mathbf{X}$:

$$y = \text{vec}(\mathbf{X}) \cdot \text{vec}(\boldsymbol{W}) = \text{vec}(\mathbf{X})^\top \text{vec}(\boldsymbol{W}) \tag{4.1}$$

We can modify two parameters that determine how a kernel convolves over the input. First is the amount of padding of the input. If we do no padding ($p = 0$) and the kernel has size greater than 1, then the output will not have the same size as the input (Fig. 4.1). No padding is also called *valid padding*. Given a kernel of size 3, the output will have the same size as the input if we pad the input with 1 in each axis (Fig. 4.2). If the padding is set in such a way that the input and output are the same, it is called *same* padding. Secondly, we can determine how big the steps are. This is called the *stride*. If one uses same padding and sets the stride to 2, the output will roughly shrink in half (Fig. 4.3).

A convolution in the semantical sense is a *feature extractor*. What feature is extracted depends on the definition of the kernel. Since a convolution filters out everything except the extracted features, the kernel is

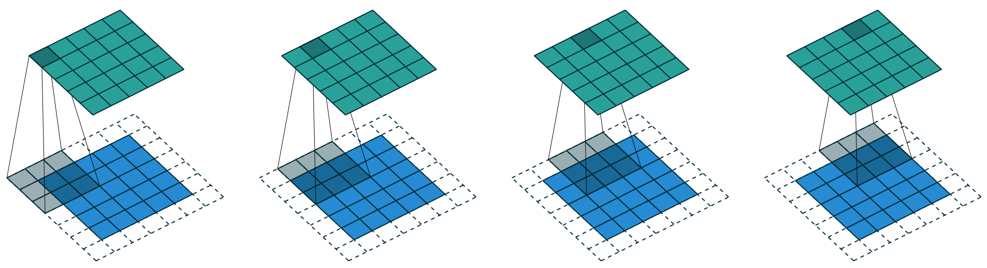


Figure 4.2: A 3×3 convolution on a 5×5 input with 'same' padding and stride 1.

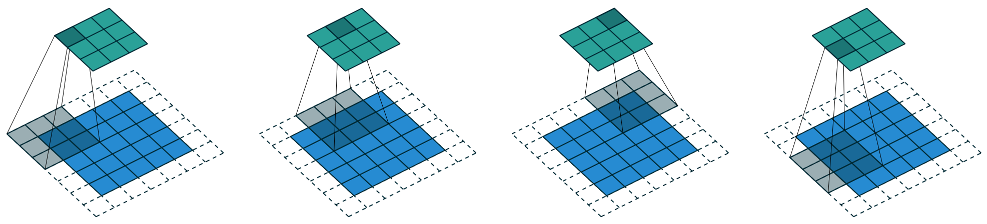


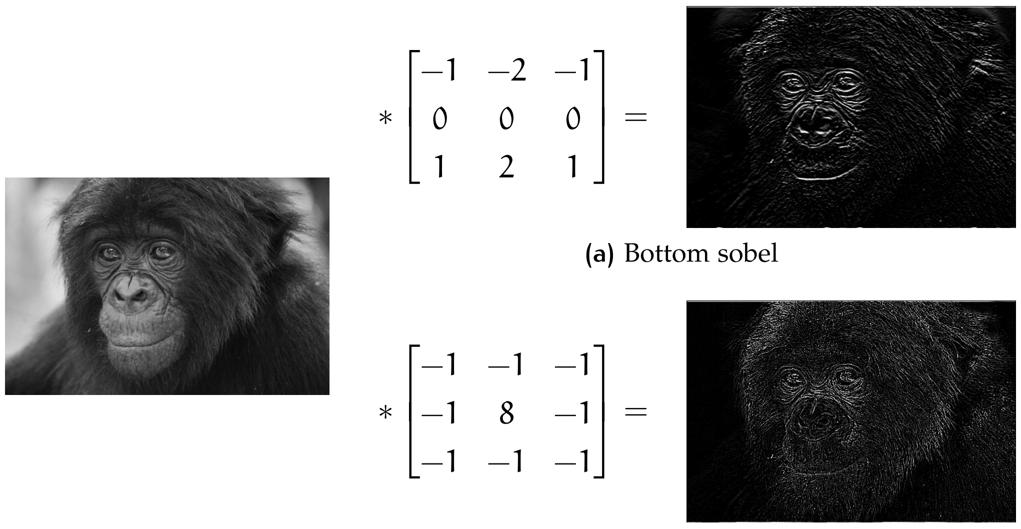
Figure 4.3: A 3×3 convolution on a 5×5 input with 'same' padding and stride 2.

also called a *filter*. What makes the convolution useful is that it respects the spatial information that the input contains because it operates on small spatially connected portions of the input.

Let us consider Fig. 4.4. All convolutions from Fig. 4.4 are performed with a 3×3 filter, same padding and stride 1. The bottom sobel filter (Fig. 4.4a) ignores the middle row and has the same weights for the top and bottom row, although the top row is negated. When the bottom row has higher values than the bottom row, thereby not allowing the top row to undo the contribution of the bottom row, the output will be positive. In other words, the output is positive if there is some horizontal contrast with an emphasis to the bottom. Finally, the edge filter strengthens the center pixel and weakens surrounding pixels. Consequently, if a pixel has a high value and is not countered by high surrounding pixels that undo its contributions completely, it will have a positive output value. Edges are exactly that, bright pixels that are (in most directions) surrounded by dark(er) pixels.

4.2 CONVOLUTIONAL NEURAL NETWORK LAYERS

The convolution has 3 properties that can make it beneficial for DNNs: *sparse connectivity*, *parameter sharing*, and *equivariant representation*. First we will explain each of these properties and why they can be helpful. Next we show what a convolutional layer looks like in a CNN. Finally,



$$* \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} =$$

(a) Bottom sobel

$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

(b) Edge

Figure 4.4: The bottom sobel and edge convolutions applied to an image of a bonobo.

we describe the other 2 common layers in CNNs; the *pooling* and *fully connected* layer.

4.2.1 Convolutional layer

Traditional DNN layers have a connection between every input-output pair. Given that the kernel size of a convolution is smaller than the input size (which is practically always the case), a convolution has *sparse connections*. Not all input-output pairs have a connection. This reduces the number of weights and computations. For example, consider a layer with an input of 100×100 and an equal size output. A traditional DNN requires $100^2 \cdot 100^2 = 10^8$ weights and an equal number of computations for such a layer. A convolutional layer with a kernel size of 3×3 on the other hand, only requires $3 \cdot 3 = 9$ weights and $100^2 \cdot 9 = 9 \cdot 10^4$ computations.

As stated above, the number of weights and computations in a traditional DNN is equal. But in a CNN the number of weights is smaller. This is because the same kernel is used to calculate every output node. Consequently, CNNs are more efficient in terms of memory requirements. This means that bigger inputs can be processed. Also, it makes them less likely to overfit.

As a result of the sparse connectivity and parameter sharing, a convolutional layer is *equivariant to translation*. It does not matter where a feature is located, the output will be the same. For example, consider a convolution that detects edges. An edge in the upper left corner of the input will produce a high value in the upper left output. Likewise, a edge in the lower right corner produces a high value in the lower right output. This is desirable because we do not want to differentiate between edges based on their location. A convolution is not equivariant to other transformations, such as rotation and scale[5].

A convolutional layer is similar to a traditional layer, but sparsely connected and with weight sharing. Thus, the backpropagation algorithm can be applied to learn the kernel weights. Just as with regular DNNs, at each layer representations are learned that depend on the representation of the previous layer. A convolutional layer can be seen as a feature extractor. However, we would like to learn multiple features per level in the hierarchy. For instance, as a first layer a CNN usually learn lines and edges.

Furthermore, higher in the hierarchy more complex features emerge, these features are composed of multiple lower features. In other words, each layer has a certain number of features that are composed from the features of the previous layer. This adds an extra dimension to the input. The extra dimension is the *channel* or *feature map* dimension. As said before, each channel is composed of channels from the previous layer. Consequently, the number of weights for a convolution layer increases with the input and output channels, i.e. the number of weights is $C_{in} \cdot s \cdot C_{out}$ where C_{in} is the number of input channels, C_{out} is the number of output channels, and s is the kernel size[6].

4.2.2 Pooling & fully connected layers

Early architectures often made use of pooling layers. Pooling layers are very simple layers to downscale the input. Pooling is similar to convolutions in that they both slide a window over the input and output 1 value for each position. Two main pooling variations exist. First *max pooling*, the output is the maximum value of the window. Intuitively, this works because when looking for features we care for the highest value because that apparently is present in the input. For instance, assume we max pool a feature map that has high values at positions with dog tails[7]. A max pool will produce a smaller output size that contains the same highest indication for dog tails, but with less spatial information. *Average pool* takes, as the name suggests, the average of the window. This approach is less common. In fact, pooling altogether is disappearing from network architectures as it turns out that changing the convolution in front of the pooling layer to have a stride of 2 and removing the pooling layer results in a network with less weights and an equivalent performance [56].

5 Data augmentation is needed to account for rotation and scale.
6 A 3×3 kernel has a size of 9 and a 3×3×3 kernel has a size of 27.
7 Actual CNNs are learned and therefore never have such clearly defined feature maps.

Architectures may also have fully connected final layers. The main part of a CNN learns the features with convolution layers (and optionally pooling layers). The final layers act as a traditional neural network that classifies based on the learned features.

4.3 CLASSIFICATION ARCHITECTURES

In this section we cover some key architectures for image classification with CNNs. The classification task lies at the foundation of other, more complex tasks.

4.3.1 VGG

VGG [57], by the Visual Geometry Group from the University of Oxford, is designed for image classification, i.e. it has a single output node[8]. textscVGG gradually decreases the resolution while increasing the depth (Tab. 4.5). As a result, there are less 'simple' features than 'complex' features. Simple features have less previous layers that increases complexity, at the same time their receptive field is small. Complex features have more previous layers to build on and simultaneously a bigger receptive field.It makes sense to dedicate more channels to complex and high level features such as faces, and less to simple low level features such as lines.

The downscaling is done with a max pool layer. Only 3×3 convolutions are used. The reason for using only 3×3 convolutions is that stacking smaller convolutional layers gives the same receptive field as a single bigger convolution, for less parameters (Fig. 4.1) and more nonlinearities in between. Due to the intuitive architecture of VGG, it has become one of the default architectures for CNNs.

FILTER	RECEPTIVE FIELD	WEIGHTS
7×7	7×7	$C\cdot(7\cdot7\cdot C) = 49C^2$
3×3	3×3	$C\cdot(3\cdot3\cdot C) = 9C^2$
$3 \cdot 3\times3$	7×7	$3 \cdot (C\cdot(3\cdot3\cdot C)) = 27C^2$

Table 4.1: The receptive field per output neuron and number of weights for convolutions on a layer with $C_{in} = C_{out} = C$ channels.

8 The single output node has multiple channels, namely the number of classes.

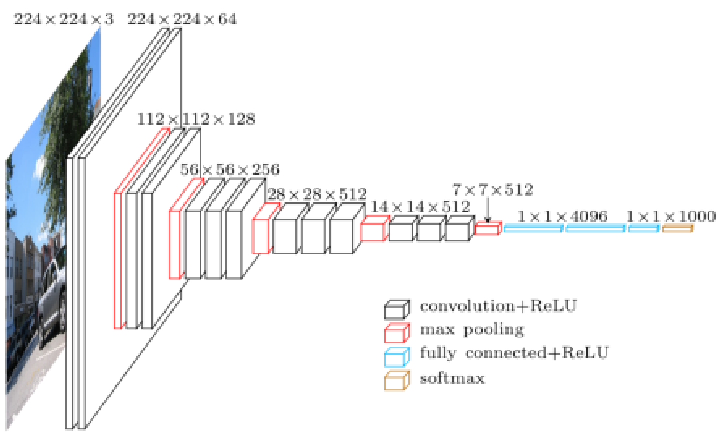


Figure 4.5: The VGG architecture. Figure extracted from Matthieu Cord's talk.

4.3.2 GoogleNet

An insight that lead to NETWORK-IN-NETWORK [58] was that an efficient way to create more complex features is by using 1×1 convolutions. A 1×1 convolution is similar to a fully connected layer from traditional DNNs, but applied to each neuron. GOOGLENET [59] uses 1×1 convolutions to achieve high accuracy without the burden of too many weights. The foundation of the architecture is the *inception module* (Fig. 4.6). Before applying the expensive 3×3 and 5×5 convolutional layers, the numbers of channels are reduced with cheap 1×1 convolutions. The results are concatenated.

Additionally, to reduce the number of parameters GOOGLENET does not use fully connected layers as final layers. Instead, it uses a average pooling at the end.

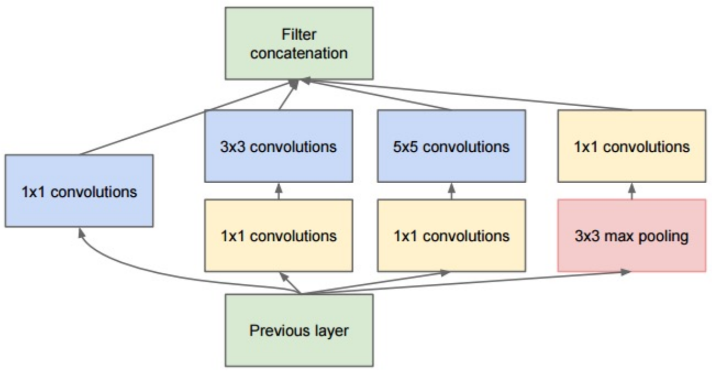
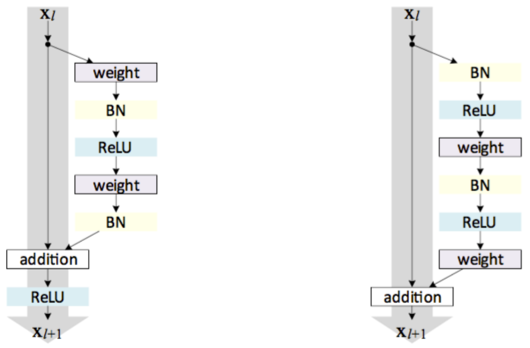


Figure 4.6: Inception module from GOOGLENET. Image by [59].

4.3.3 ResNet

Due to the unstable gradient problem, it is hard to stabilize learning with deep neural networks. To overcome this problem architecturally ResNet [14, 60] uses *residual skips* (Fig. 4.7). Instead of applying operations and send the results to the next layer, the input is added to the output. Recall that a gradient flows back trough the network by recursively multiplying the derivative of the operation with the gradient. The derivative of an addition is 1. In other words, the gradient flows unchanged trough the addition operation. Consequently, the gradient does not vanish no matter how deep the network is.

With residual skips, networks that have more than 1000 layers have been trained successfully.



(a) Original **(b)** Improved

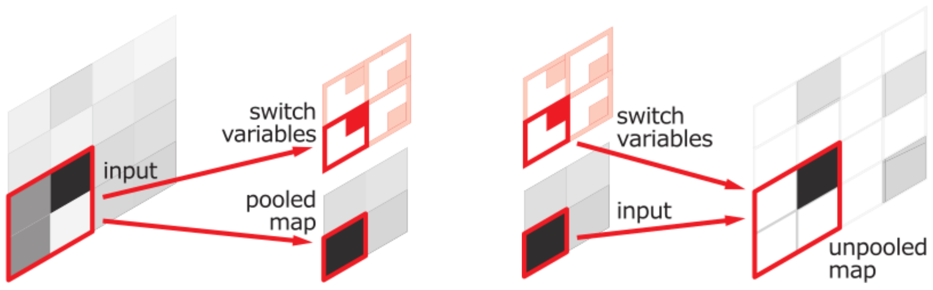
Figure 4.7: The original [14] and improved [60] residual block from ResNet.

4.4 UPSCALING LAYERS

Classification networks do no require upscaling layers, but networks designed for segmentation often do. The two key upscaling layers are the *unpool* and *transposed convolutional* layers.

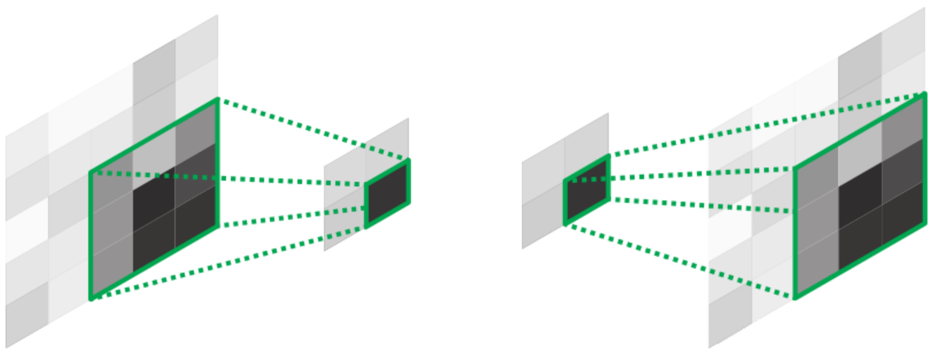
4.4.1 Unpooling

The most straightforward way to upscale the input, is by doing a reverse pooling operation. The so-called *unpool* operation. Two variations are possible. First, every output gets the same value as the input. This creates a dense, but coarse output. Alternatively, if the network stores the index that had the max value (the switch variable), the same index can be filled during unpooling (Fig. 4.8). Other pixels are set to zero,



(a) Pool **(b)** Unpool

Figure 4.8: A pool and corresponding unpool operation with switch variables. Images by Noh *et al.* [61].



(a) Convolution **(b)** Transposed convolution

Figure 4.9: A convolution and transposed convolution with the same kernel. Images by Noh *et al.* [61].

resulting in a sparse matrix. For average pooling the contribution per index is stored and during unpooling the input is divided accordingly.

4.4.2 Transposed convolution

Sometimes called a *deconvolution*[9], the *transposed convolution* performs a convolution with the weight matrix transposed. In other words, normally a $n \times n$ matrix from the input is multiplied with a filter and summed to 1 value (Fig. 4.9). But a transposed convolution takes 1 value and maps it to a $n \times n$ matrix. To do so, first the kernel $\boldsymbol{W}$ is vectorized and transposed. This is mulitplied with the input value x. The result is a $1 \times n^2$ vector. Devectorizing this gives the final window. Consequently, the stride and padding parameters concern the output rather than the input. Naturally, transposed convolutions are suitable for upscaling. It

9 The name was introduced by Zeiler *et al.* [62]. It is an unfortunate name because it implies that a deconvolution undoes a convolution, which is not the case.

is possible that the output windows $\mathbf{Y}$ overlap, the overlapping values are summed together.

$$\mathbf{Y} = \operatorname{devec}(x \cdot \operatorname{vec}(\boldsymbol{W})^\top) \tag{4.2}$$

4.5 SEGMENTATION ARCHITECTURES

A (very) naive approach to segmentation would be to extract patches around each pixel that fit into state-of-the-art classification networks and use the result as label for the pixel. However, this is extremely inefficient. In this section we describe two key architectures for semantic segmentation.

4.5.1 Fully convolutional networks

The *fully convolutional network* (FCN) [63] tries to optimize segmentation by mimicking the final fully connected layers with 1×1 convolutions. In this manner, the network can be scaled to bigger inputs. As a result, the output is no longer a single value, but multiple values (Fig. 4.10). Note that prediction on a bigger image with a FCN requires less computation than doing the same with multiple independent patches.

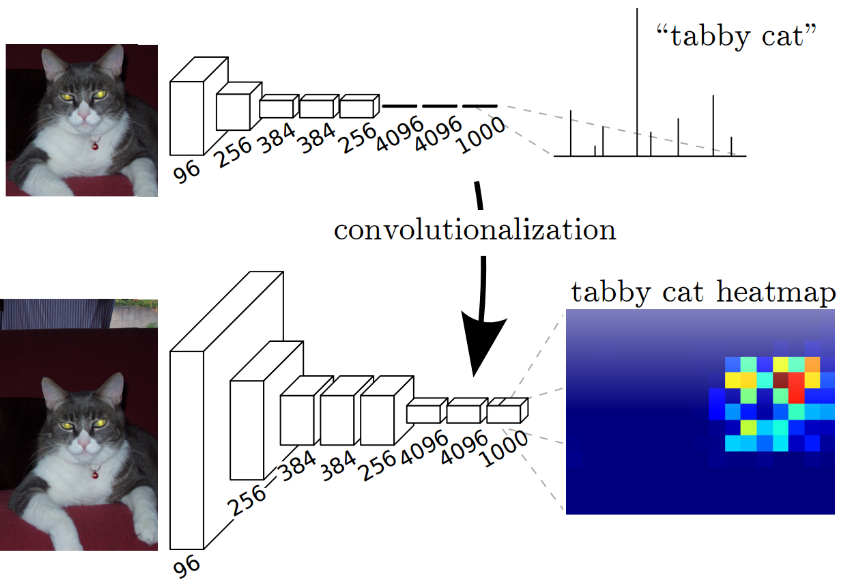


Figure 4.10: Convolutionalization of a classification network enables a efficient pixel-wise prediction, i.e. segmentation. Image by Shelhamer *et al.* [63].

The output resolution is smaller than the input resolution; how much smaller depends on what the network architecture is. To scale the smaller output up to the original size, a transposed convolution is used. However, it is likely that the filter size of the transposed convolution is very big. Or in other words, the transposed convolution has to create a segmentation from a neuron with a large receptive field. This will not result in fine-grained segmentations. To counter this, FCNs upscale to the size of an earlier feature map and sum them together (Fig. 4.11). Earlier neurons have a smaller receptive field and thus the information is spatially more constricted. After the summations a final transposed convolution upscales the activations back to the original size. There is a trade off to be made about how much layers one considers during upscaling. Fewer layers give a better semantic understanding of the pixel and emphasize features with large receptive fields more (useful for segmenting large objects). More layers have more emphasis on the fine-grained spatial boundaries of objects (useful for small objects and object borders).

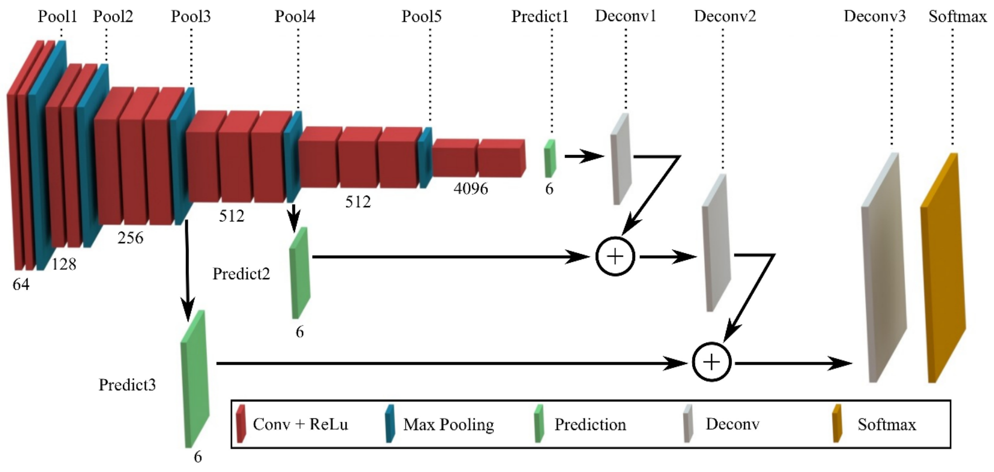


Figure 4.11: A VGG FCN that includes two extra layers in upscaling. Image by Tai *et al.* [64].

4.5.2 Decoder networks

Instead of applying transposed convolutions with large filter sizes, DeconvNet [61] and SegNet [65] gradually increase the size[10]. This gives the networks a structure similar to *stacked auto-encoders* (Fig. 4.12). Both networks use an unpool layer (with remembered pooling indices from the pool layer) for upscaling. After an unpooling layer the resulting feature map is sparse. To make a dense feature map a transposed convolution is used.

10 DeconvNet and SegNet are very similar, they have been published around the same time.

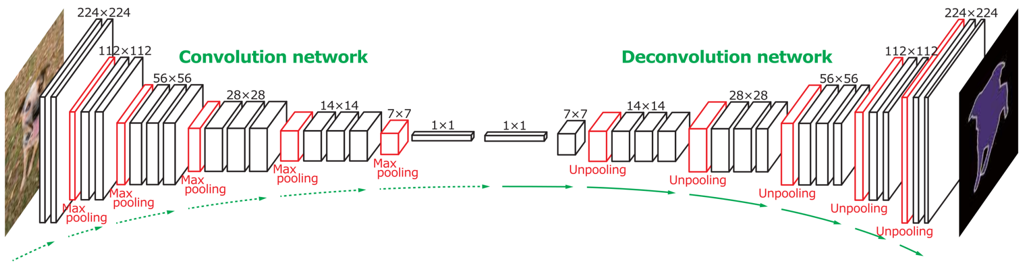


Figure 4.12: The DeconvNet architecture. Image by Noh *et al.* [61].

5 TASK AND DATASET

5.1 THE TASK IN CONTEXT

The ultimate goal of the project is to diagnose whether a foramen exhibits signs of stenosis based on a MRI scan. To train a network that does this with complete MRI scans as input would require enormous amounts of data. To avoid this we decided to split the problem into two subtasks. The first to localize the ten foramina of the lumbar spine with a network. The volume around each foramina is to be extracted and fed to the second network that classifies the amount of stenosis for each foramen. The localization network is trained with the complete MRI scans and requires the coordinates of foramina as target values. The classification network is trained with the extracted volumes around the foramina and requires the degree of stenosis as target value. The reports written by radiologists are available, but these are written in natural language. In order to use them as target values, the relevant information needs to be extracted. In summary, there are three subtasks to be performed;

1. Localize the 10 foramina of the lumbar spine.
2. Determine the presence of stenosis for each foramen as diagnosed by a radiologist from the corresponding report.
3. Classify the presence of stenosis for each foramen.

Given the time-constraint of the project, the focus of this thesis is on the first task: localizing the foramina from the MRI scan. We aim to answer the following question:

> *Is it possible to design and train a deep neural network that has viable performance on the task of lumbar foramina localization on MRI scans without using data driven postprocessing techniques or hand-crafted features?*

No target values (i.e. the foramina 3D coordinates) for this are available, thus, manual annotation is unavoidable. With manually annotated target values, there are two options for training the network. Either segment the MRI scan to a probability map and derive the coordinates with a (non data driven) postprocessing step, or train a network that directly outputs the coordinates or bounding boxes around the foramina. State of the art localization networks require complicated trainings and test pipelines [66]. To avoid this, we opt for a semantic segmentation of the volume to accomplish the first subtask.

5.2 THE DATASET

The dataset consists of MRI scans with corresponding reports that contain the findings of a radiologist. The MRI scans come from a *picture archiving and communication system* (PACS) in the *digital imaging and communications in medicine* (DICOM) format [67] (Fig. 5.1). When a patient undergoes an examination it is possible that multiple tests are performed. For example, the MRI technician could scan on multiple planes or with different weightings. A complete examination is called a *study*. Each scan is a *series*[11]. An MRI scan returns volumetric data, the DICOM format stores each slice from the volume as an *instance*.

$$\begin{tikzcd}
\boxed{\text{Patient}} \arrow[r, "1{:}n"] & \boxed{\text{Study}} \arrow[r, "1{:}n"] & \boxed{\text{Series}} \arrow[r, "1{:}n"] & \boxed{\text{Instance}}
\end{tikzcd}$$

Figure 5.1: A simplified view of the DICOM data model.

DICOM files are instance-based. In other words, each DICOM file is an instance that contains meta data (Tab. 5.1). By examining the meta data, other instance from the same series, study or patient can be determined. E.g. a series is constructed by combining all instances that share the same `SeriesUID` tag. In a similar manner the same can be done for patient and study. As a consequence, we are dependent on the meta data to reconstruct complete MRI scans.

In addition to the `SeriesUID` tag, more tags are needed to construct a complete MRI scan. The `Plane` (sagittal, axial (traverse), or coronal) and `Weight` (T1 or T2) are derived from the `SeriesDescription`. `PixelData` contains a 2D array with gray scale values of the slice. From the `PixelData` resolution we can derive the `Shape` of the image. To create a 3D array the `PixelData` arrays are concatenated. To concatenate the `PixelData` correctly, we order them based on the x, y or z value from `ImagePositionPatient`. Which axis to order depends on the `Plane`, i.e. given planes sagittal, axial, or coronal we order x, z, or y, respectively. The arrays are ordered from low to high. The DICOM standard dictates that the coordinates increase in value in the x-axis from right to left, in the y-axis from front to back, and in the z-axis from feet to head (Fig. 5.2). We only consider scans that have `PatientPosition` *Head First-Supine* (HFS) or *Feet First-Supine* (FFS), i.e. patients laying on their back with either head or feet first.

11 We use the terms *scan* and *series* interchangeably.

TAG	NAME	DESCRIPTION
0010 | 0020	`PatientUID`	The UID of the patient.
0020 | 000d	`StudyUID`	The UID of the study.
0020 | 000e	`SeriesUID`	The UID of the series.
0008 | 0018	`InstanceUID`	The UID of the instance (slice).
0008 | 103e	`SeriesDescription`	Describes the series plane and weight.
0018 | 5100	`PatientPosition`	The position of the patient relative to the scanner.
0018 | 0050	`SliceThickness`	The thickness in mm of each slice.
0018 | 0088	`SpacingBetweenSlices`	The space in mm between slices.
0028 | 0030	`PixelSpacing`	The space in mm that is represented between pixels.
0020 | 0032	`ImagePositionPatient`	The x, y and z coordinates of the instance image (slice).

Table 5.1: Selected meta tags (attributes) of DICOM images.

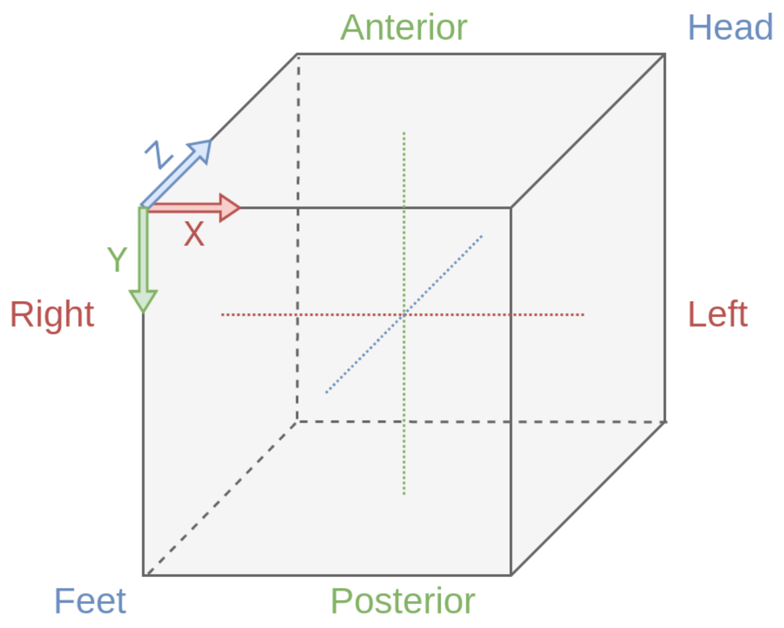


Figure 5.2: The DICOM reference coordinates system.

In total the dataset consists of 291.776 series, that is, there are 291.776 unique `SeriesUIDs` present in the meta data. Of those, 40.572 are so called *localizers*. Localizers are coarse scans used to establish where to aim the scanner[12]. Such scans are not usable for us. Moreover, for 59.900 scans the meta data is not complete. This leaves us with 2415.304 complete and precise scans.

Discussions with radiologists revealed that sagittal T1-weighted scans are most suitable to detect foraminal stenosis. Of the complete scans, 40.064 are sagittal T1-weighted. Not all sagittal T1-weighted scans are suitable. All scans that have no value for `SliceThinkness`, `SpacingBetweenSlices` or `PixelSpacing` are discarded. To remove outliers and corruptions in the meta data (e.g. a negative `SliceThinkess` is theoretically not possible) we specify restrictions on these attributes that describe around 99% of the data. More specifically, $3 \leqslant$ `SliceThinkness` $\leqslant 4$ and $3 \leqslant$ `SpacingBetweenSlices` $\leqslant 5$.

For the `Shape` (resolution) we choose values that are more strict than the values that describe 99% of the data. Since the `PixelSpacing` differs per scan, a network has to learn to compensate for this variation. To avoid this extra learning, we normalize the `PixelSpacing`. Based on the original and normalized distribution we choose a minimal resolution of 500×500. All normalized images that grow more than 20% during the normalization are discarded.

Keeping the future tasks in mind, only scans that have a corresponding report from a radiologist are used. In the end we have a dataset with 29.309 MRI scans that are uncorrupt, satisfy the restrictions, and have a corresponding report.

	INSTANCES	SERIES	STUDIES	PATIENTS
All	5.831.211	291.776	42.655	37.293
Suitable	-	29.309	28.825	26.660

Table 5.2: Statistics of the complete data set.

5.3 CREATION OF THE MANUAL ANNOTATIONS

The dataset does not contain the target values that are needed for training the localization network[13]. Therefore, we have to create our own target labels. To that end, we build an *annotation tool* (Fig. 5.3). Each lumbar spine has 10 foramina, with the annotation tools someone can scroll trough the slices and annotate relevant points. More than 1 slice per foramen might contain information that a radiologist uses to deter-

12 The time it takes to complete an MRI scan increases with more precise settings.

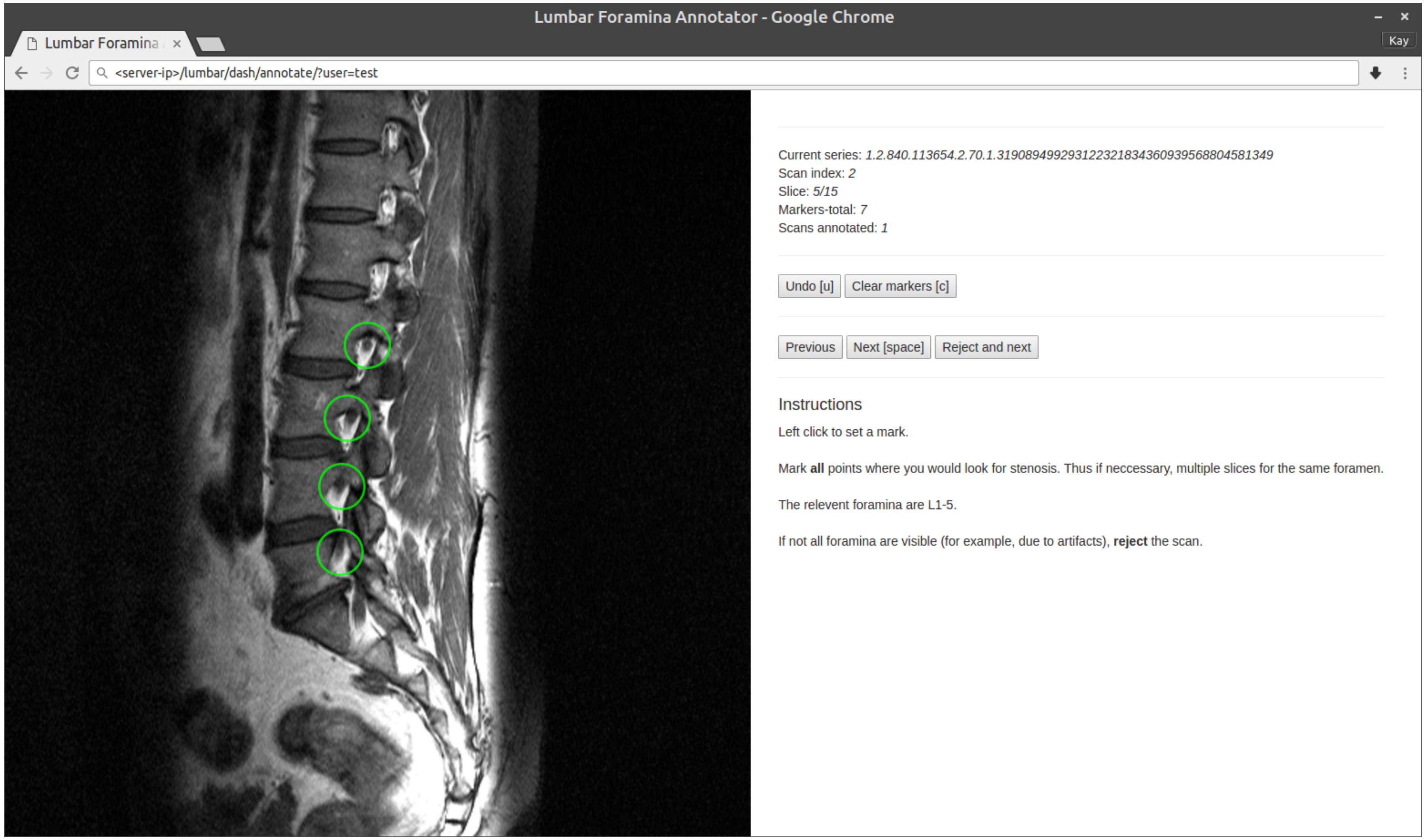
13 From here on when we refer to our 'localization' network, we refer to our network that does segmentation for the purpose of localization.

mine the presence of stenosis. Thus, per foramen an annotator can mark multiple points. Since the MRI scans are ordered during the preprocessing, we can derive the corresponding foramina for each coordinate in an annotation.

To ensure the trustworthiness of the annotations, a radiologists performed an independent trial of 50 scans. The annotations of the radiologist have been compared with our annotations. Since it is a subjective matter which slices should or should not be included, some difference between two annotators is inevitable. However, we found that in all cases there was at least 1 marker that overlapped between the annotations of the radiologist and our own for all foramina. Additionally, the differences have been evaluated together with the radiologist. The radiologist in question deems the differences negligible[14].

A total of 500 scans have been manually annotated with the locations of the 10 lumbar foramina.

14 An endorsement of our annotation skills by a certified radiologist is necessary for a CE certificate


Lumbar Foramina Annotator - Google Chrome
Lumbar Foramina
Kay
<server-ip>/lumbar/dash/annotate/?user=test
Undo [u]
Clear markers [c]
Previous
Next [space]
Reject and next

Current series: *1.2.840.113654.2.70.1.319089499293122321834360939568804581349*
Scan index: *2*
Slice: *5/15*
Markers-total: *7*
Scans annotated: *1*

Instructions

Left click to set a mark.

Mark **all** points where you would look for stenosis. Thus if neccessary, multiple slices for the same foramen.

The relevent foramina are L1-5.

If not all foramina are visible (for example, due to artifacts), **reject** the scan.

Figure 5.3: Interface of the annotation tool.

6

RELATED WORKS

As mentioned before in Chap. 1 radiologists interpret medical images. Radiology is a part of *medical image analysis*. The advances in computer vision (Chap. 4) due to deep learning have not gone unnoticed in automatic medical image analysis groups. The number of published papers concerning automatic medical image analysis with deep neural networks has exploded in the recent past (Fig. 6.1).

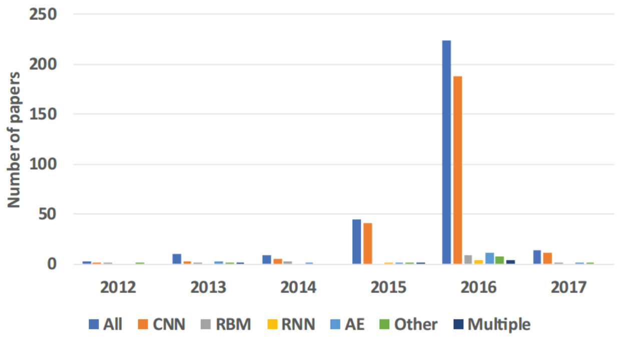


Figure 6.1: Number of papers about deep learning for automatic medical image analysis[15]. Image by Litjens *et al.* [68] from (February 19, 2017).

Typically, biomedical data is 3D since it represents parts of the human body (which a 3D object). However, computer vision mostly focuses on 2D data (images). Although 2D CNNs are applicable to volumes by applying them to 2D images along the depth axis, this approach does not take the spacial information along the depth axis into consideration. For CAD systems that analyze MRI scans, it is crucial to be applicable to volumetric data.

6.1 SEGMENTATION OF BIOMEDICAL VOLUMES

Segmentation is an important task for CAD systems. It helps radiologists to make a better diagnosis and often is a first step in the pipeline for other tasks. In this project we also use segmentation as a first step for our localization task. Consequently, is the most common task in published papers concerning deep learning in automatic medical image analysis [68].

15 Papers were looked for on *Pubmed*, *arXiv*, and the conferences proceedings of *MICCAI* (including workshops), *SPIE*, *ISBI*, and *EMBC*.

The best-known network for biomedical segmentation is U-NET [69] which combines the architectures of the FCN [63] and DECONVNET [61]. Its main contribution, the *long skip* between the down- and upscaling branches, has since been adopted among CNNs for natural image segmentation [15].

6.1.1 U-Nets

Even though its network is designed for biomedical data, U-NET [69] works well for segmentation in general. It combines the ideas of a auto-encoder CNN and taking into account information from multiple layers for fine-grained spatial information. This leads to an architecture that uses *skip connections*[16] between the down- and upscaling branches. Max pooling is used to downscale the input, to upscale U-NET first upsamples the image and then performs a 2×2 convolution.

Drozdzal *et al.* [70] investigated the importance of the long skip connections and residual skips (that were not invented when the original U-NET was published) for the performance of U-NETS. The authors stress the importance of such connections and note that without them the networks perform worse.

The architecture of the U-NET can easily be extended to support volumetric data. V-NET [71] (Fig. 6.2) and 3D U-NET [72] do not offer any substantial contributions besides making the architecture U-NET 3D compatible and implementing techniques that have been discovered after the U-NET paper was published, such as *batch normalization* [43] and *residual skips* [60].

U-Networks (denoting all networks that are variations of U-NET) share some limitations: class imbalance and data scarcity. Both are caused by the fact that U-Networks segment large patches and segmentation typically has an imbalanced distribution [73]. Moreover, manual segmentation is a time consuming task that most of the time require an expert, this holds especially for biomedical data. As a result, there are not many large biomedical segmentation datasets. A natural solution is to apply heavy data augmentation.

When segmentation output volumes are sparse (which is often the case when segmentation is used as a first step in a detection pipeline), there is class imbalance. There will be many more negative than positive pixels to classify. Typically, *cross entropy* [74] is used as a measure of performance for image classification. It is possible to use cross entropy for segmentation, since it is a pixel-wise classification. But a network that is trained on a dataset with a class imbalance and a cross entropy loss will

16 Sometimes residual skips are also referred to as skip connections. In this study a skip connection refers to the long skip connections as in U-NET.

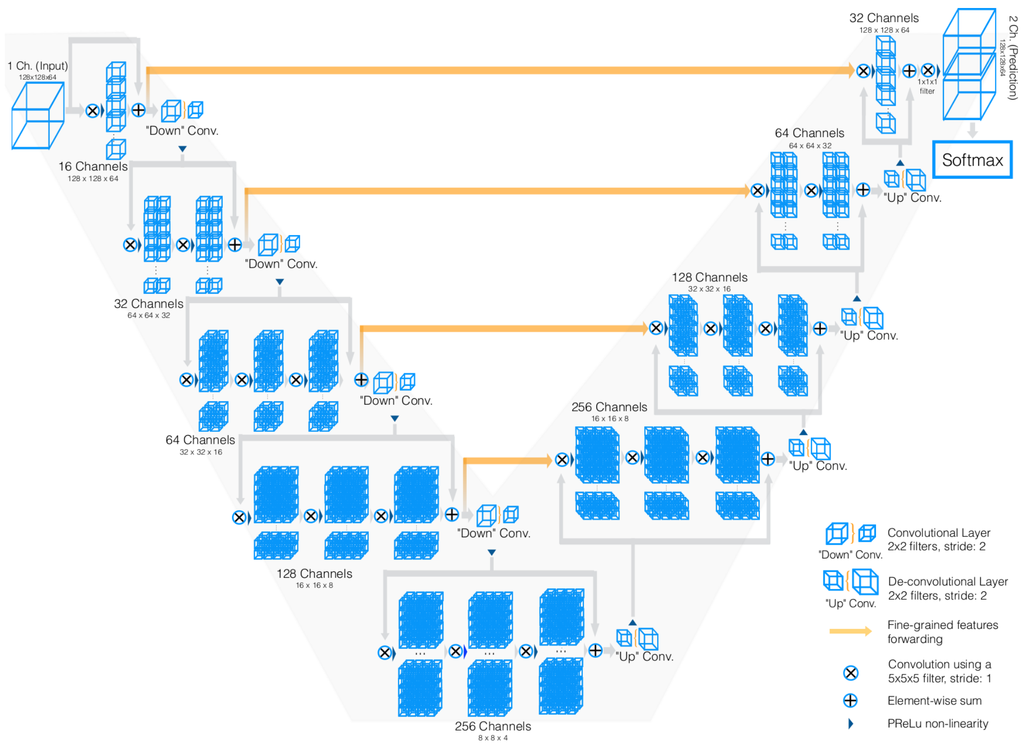


Figure 6.2: V-NET architecture. Image by Milletari *et al.* [71]

be biased towards the dominant class. The authors of the U-NET and 3D U-NET solved this by using a weight map in combination with cross entropy. A simpler approach that does not require any hyperparameters or weight maps is to use the *Dice coefficient* [75] (Sec. 7.1).

6.1.2 Deep supervision

Another approach to segmentation is to dispose of the expanding branch, i.e. no gradual expansion of a gradually compressed input. VOXRESNET [76] and 3D DSN [77] gradually compress the input. But instead of a gradual expansion, they upscale different layers to the original size, similar to FCN. VOXRESNET consists of stacked *VoxRes modules* (which are basically 3D residual blocks [60]), alternating strided convolutions to decrease the size (Fig. 6.3). The output of four layers is unscaled to the original size and summed together for the final output. In addition to the final classifier, the authors opted to use *auxiliary classifiers* [59] to ensure that the intermediate layers produce features that are directly useful for the segmentation. This has the disadvantage of introducing an extra hyperparameter that specifies the weight of each classifier. The authors argue that this approach is more efficient while maintaining the fine-grained spatial information from the lower features because of the auxiliary classifiers.

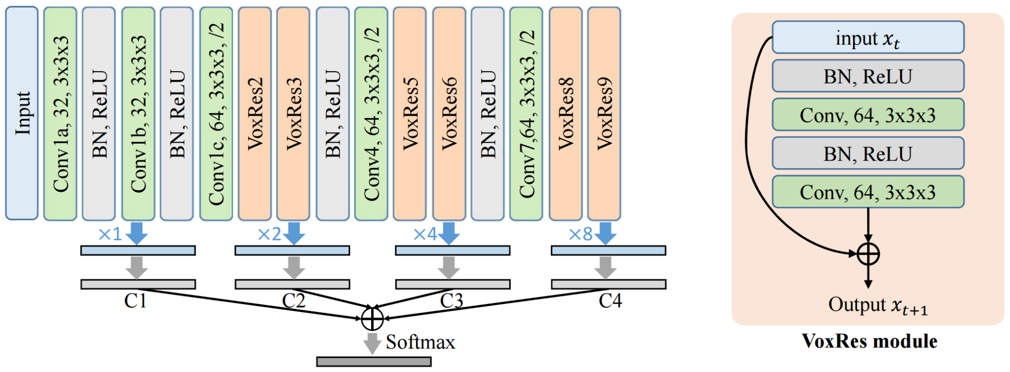


Figure 6.3: VoxResNet architecture. Image by Chen *et al.* [76].

6.1.3 U-Nets with deep supervision

More recently two architectures attempted to combine the ideas of U-Net and deep supervision. CUMED [78] (Fig. 6.4) uses direct deep supervision with auxiliary classifiers. Alternatively, the network by Kayalibay *et al.* [73] interpolate and sum layers together, similar to FCN (Fig. 4.11).

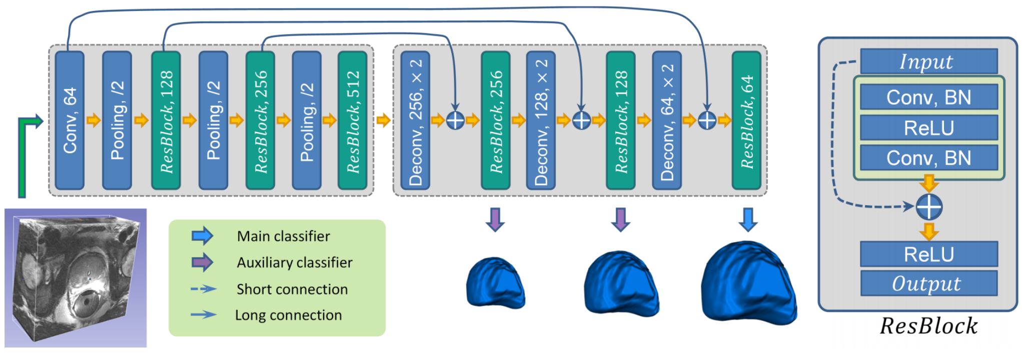


Figure 6.4: CUMED architecture. Image by Yu *et al.* [78]

6.2 VERTEBRAE DETECTION

Localization of the foramina is an uncommon task that has barely been studied. Vertebrae detection however, is much more common. It aids radiologists with diagnosing and planning treatments for spinal disorders. As showed before in Fig. 6.1 the deep learning revolution for automatic medical image analysis started in the year 2015, before that data driven approaches with traditional computer vision techniques were much more common. Examples are multi-based atlas registration [79], conditional random fields [80], and regression forests [81].

The similarity in morphological appearance makes it hard to distinguish the vertebrae from each other. This is due to the repetitive nature of a spine. Additionally, diseases that cause abnormalities in the spine are rare, making it hard for techniques to correctly label the vertebrae in such cases. We humans can take in the context to infer which vertebrae we are dealing with, e.g. we know that L1 is positioned above L2. State-of-the-art vertebrae detection networks all use multiple stages to somehow utilize this prior information.

One of the first convolutional neural networks for vertebrae detection is DeepSeg [82]. The authors of DeepSeg developed a two-stage pipeline. First they make rough estimations of the localization and segmentation. Secondly, they use the localizations to define a *region of interest* (ROI) and improve the segmentation in the ROI.

The output of DeepSeg's localization network is a coordinate per vertebrae, i.e. DeepSeg is a regression network. During inference multiple random patches are sampled and the results are aggregated into a probability map per vertebrae. The probability maps are refined with a *hidden Markov model* (HMM) that connects them all.

Chen *et al.* [83] do not train their network directly on the raw medical image. First HOG features are extracted and a binary random forest is trained to classify each voxel as part of a vertebral. The output is a binary volume, this volume is feeded trough their CNN, J-CNN. J-CNN outputs a probability per level for each candidate from the coarse locations. To improve the identification rate, the neighboring dependencies are taken into consideration while refining the probabilities.

Yang *et al.* [84] propose a 3-stage approach. The first stage is a CNN that predicts a probability map per level foramen. To avoid false positives the authors make use of a *message passing* scheme that leverages the mutual relations between the vertebrae. In its essence the message passing scheme is a graphical model that is also trained on the training set. This graphical model takes the prior information about how the vertebrae are structured into account.

A similar approach is taken by Forsberg *et al.* [85]. However, they use two CNNs; one for the general foramina and one for the deviant S1 vertebrae[17]. Again a graphical model is trained on probability maps during training, allowing to taking into account the prior information about mutual dependencies between the vertebrae. In other words, the graphical model assigns labels to the candidates that best resemble the spines seen during training.

17 The S1 vertebrae is morphological very different from the L1-5 vertebrae, the same holds for the corresponding foramina.

7 EXPERIMENTS

In this chapter we describe the performed experiments and their results. All experiments follow the same outline:

1. Train a network with 400 trainings volumes on a segmentation loss.
2. Perform inference on the resulting 100 test volumes with the trained network.
3. Determine the 10 coordinates from the each of inferred volumes, for a total of 1000 coordinate predictions.
4. Measure the performance of the network with a localization metric.
5. Test the localization performance for statistical significance against other networks.

As said before, our approach consists of two stages, segmentation and localization. The segmentation task is not our ultimate goal, yet we require a proxy loss function to train the network. Subsequently, we need to evaluate how well the segmentation of the network is suitable for the localization stage. The metrics used for segmentation and localization are described in Sec. 7.1. Additionally, we want to compare models and check if their difference is statistically significant. Our test for this is explained in Sec. 7.1.3.

Before segmenting the volumes, we first preprocess them. In Sec. 7.2 a walk trough of the preprocessing pipeline is given. Note that even though we describe the pipeline as if every step is mandatory, they are all optional.

We experimented with two different approaches to segmentation, a single-channel and multi-channel output. The corresponding networks SingleNet and MultiNet are explained in Sec. 7.4 and 7.5, respectively. Both sections follow the same outline. First a general overview of the base network is given. Next we list all variations we made of the network to improve it or test the influence of an architectural choice. As both approaches require slightly different methods of foramina localization from the segmented volumes, both sections contain a short explanation about how to do the localization. We compare the models on how well the localization is done and test for statistical significance.

For pre- and postprocessing we made extensive use of `NumPy` [86], `SciPy` [87], and `Pandas` [88] which are all libraries for scientific computing in `Python`. The neural networks are implemented in `TensorFlow` [89], an interface by Google that let us express and compute machine learning algorithms as a graph, and trained on a NVIDIA GeForce GTX 1080 Ti GPU.

7.1 EVALUATION METRICS

We opted for a localization by segmentation network, therefore we have to measure the performance of two tasks. First, we require a metric that measures the correctness of the segmentation and secondly a metric that measures how good the actual predicted coordinates for the foramina are.

7.1.1 Segmentation metrics

For the segmentation we use the Dice coefficient [75]. The Dice coefficient measures the overlap between two sets A and B:

$$\begin{aligned} D(A,B) &= \frac{2|A\cap B|}{|A|+|B|} \\ &= \frac{2|A\cap B|}{(|A\cap B|+|A\backslash B|)+(|A\cap B|+|B\backslash A|)} \\ &= \frac{|A\cap B|}{|A\cap B|+\frac{1}{2}|A\backslash B|+\frac{1}{2}|B\backslash A|)} \end{aligned} \tag{7.1}$$

It divides two times number of elements in the intersection of A and B by the total number of elements in A and B combined. It is easy to see that when the sets are the identical, the Dice coefficient is maximal $D(A,B) = 1$. For every element that A misses with respect to B, the numerator will shrink. For every element that A has extra with respect to B, the denominator will grow. Both will result in a lower score. We can see that the Dice coefficient is the same as the F1 score:

$$\begin{aligned} F1(A,B) &= \frac{2}{\frac{1}{\text{Precision}}+\frac{1}{\text{Recall}}} \\ &= \frac{2}{\frac{1}{\frac{|A\cap B|}{|A|}}+\frac{1}{\frac{|A\cap B|}{|B|}}} \\ &= \frac{2}{\frac{|A|}{|A\cap B|}+\frac{|B|}{|A\cap B|}} \\ &= \frac{2|A\cap B|}{|A|+|B|} \end{aligned} \tag{7.2}$$

Some literature [73] suggests using the Jaccard coefficient [90]. However, since the Jaccard coefficient is monotonic to the Dice coefficient this will only result in a different optimal learning rate.

$$\begin{aligned} J(A,B) &= \frac{|A \cap B|}{|A \cup B|} \\ &= \frac{|A \cap B|}{|A \cap B| + |A \backslash B| + |B \backslash A|)} \end{aligned} \tag{7.3}$$

The Dice coefficient as defined in Eq. 7.1 requires a discrete number of elements in both sets. Our results however, are probabilities. The `TensorFlow` library requires all operations to be differentiable. Therefore, it is not possible to round the probabilities to binary set. Thus, we have to modify the Dice coefficient to be compatible with probabilities.

$$D_{\text{fuzzy}}(\mathbf{p}, \mathbf{t}) = \frac{2\sum_i^n p_i t_i}{\sum_i^n p_i + \sum_i^n t_i} \tag{7.4}$$

The fuzzy Dice coefficient D_{fuzzy} takes two vectors[18] of length n; the prediction $\mathbf{p}$ and target $\mathbf{t}$. Note that this definition only works because target $\mathbf{t}$ is, in contrast to prediction $\mathbf{p}$, binary.

7.1.2 Localization metrics

A straightforward metric for the localization performance is the mean error distance from the predicted coordinates to the target coordinates. However, the end goal is too extract a subvolume around the foramen. As we will extract from multiple slices in the x-axis (from right to left on the patient) and the distance between slices is bigger than 3 and less than 5 millimeters, we can miss the foramen with approximately 10 millimeters in the x-axis and still correctly extract a subvolume.

For the other two axes however, this does not hold. The y and z-axis (from front to back and feet to head, respectively) are the axes that define the 2D space of a slice. The accuracy on the slice itself is required to be more precise because a predication that is 10 millimeters off in this space might very well be the coordinate of another foramen.

Therefore, we measure the distance of a prediction to the real center in two ways. The distance in the x-axis, which conveys how many slices the prediction is off. And the euclidean distance in the yz-axes space, which conveys how far the prediction is off on the slice itself. The latter is most important and is the subject of the reported mean error distance.

$$\begin{aligned} \text{Dis}_x(p_x, t_x) &= \sqrt{(t_x - p_x)^2} \\ \text{Dis}_{yz}(p_y, t_y, p_z, t_z) &= \sqrt{(t_y - p_y)^2 + (t_z - p_z)^2} \end{aligned} \tag{7.5}$$

18 The vectorized volumes.

It might occur that a network makes no prediction at all for a particular foramen. Obviously, this is wrong. Yet, this is hard to include such predictions in the mean error distance. For our purpose it is preferable if a model makes no prediction, rather than a wrong prediction. Thus, we ignore these 'missed' foramina in the mean error distance. In the appendix we will present the number of missed foramina for all networks (Tab. B.1 and **??**). As a result, the models can not be compared fairly by means of mean error distance, because a different set of foramina is considered for different models.

In addition to the mean error distance and number of missed foramina, we also calculate the recall of the model. Since recall considers binary classification we have too set a threshold that defines when a prediction is considered positive or negative. We decided that 5 millimeters is a fair threshold. The recall includes missed foramina as negatives. The recall takes all 1000 predictions into account, thus the models can fairly be compared with this metric.

7.1.3 Model comparison

To compare two models M_A and M_B for a significant difference, we consider them as binary classifiers. The same threshold of 5 millimeter is used, as for the recall. With both models we make a prediction on each foramen in the test set. This results in the 2×2 contingency table as shown in Tab. 7.1. We are interested in whether the difference is significant, thus we do not consider a and d. The null hypothesis is $H_0 : p_a = p_b$. Normally we would use the McNemar's test [91], however, the McNemar's test depends on the χ_1^2 distribution which is an approximation that breaks down for a low $b + c$. Therefore, we use the exact binomial test. The binomial test gives us a p-value, which is the chance of observing these values (b and c) given that H_0 is true. A low p-value suggest that H_0 is not true.

	M_A+	M_A-
M_B+	a	b
M_B-	c	d

Table 7.1: A 2×2 contingency table between 2 binary classifiers.

7.2 PREPROCESSING

We perform several steps of preprocessing before a volume is fed to the network. First we discard corrupted samples and outliers as described in Sec. 5.2. Accepted volumes go through the pipeline described below. For many machine learning applications sophisticated preprocessing techniques are absolutely necessary. However, computer vision usually requires very little preprocessing [12]. Every stage of the pipeline can be disabled if necessary. For simplicity we consider the full pipeline in this section.

First the `PixelSpacing` is normalized to a `NormSpacing` defined by us. We choose a `NormSpacing` of 0.7 millimeters because that was close to the average. Even though we are normalizing all scans, we would like to minimize the modification. The `ScaleFactor` is calculated with `PixelSpacing` /`NormSpacing`. To get the new size we multiply the `ScaleFactor` with the original size. Each slice is resized with bilinear interpolation.

Next, for each scan a corresponding target volume is generated with the manually annotated foramina markers. A volume, with an identical shape to the scan, is initialized with zeros. On the slices with markers a circle with radius `LabelRadius` (the default is 7 millimeters) is set to ones. The marker coordinates are multiplied with the previously determined `ScaleFactor` to account for the resize. Note that these are circles and not spheres, they extend in the y and z dimension. The result is a sparse 3D tensor with roughly the voxels that are part of a foramina set to 1.

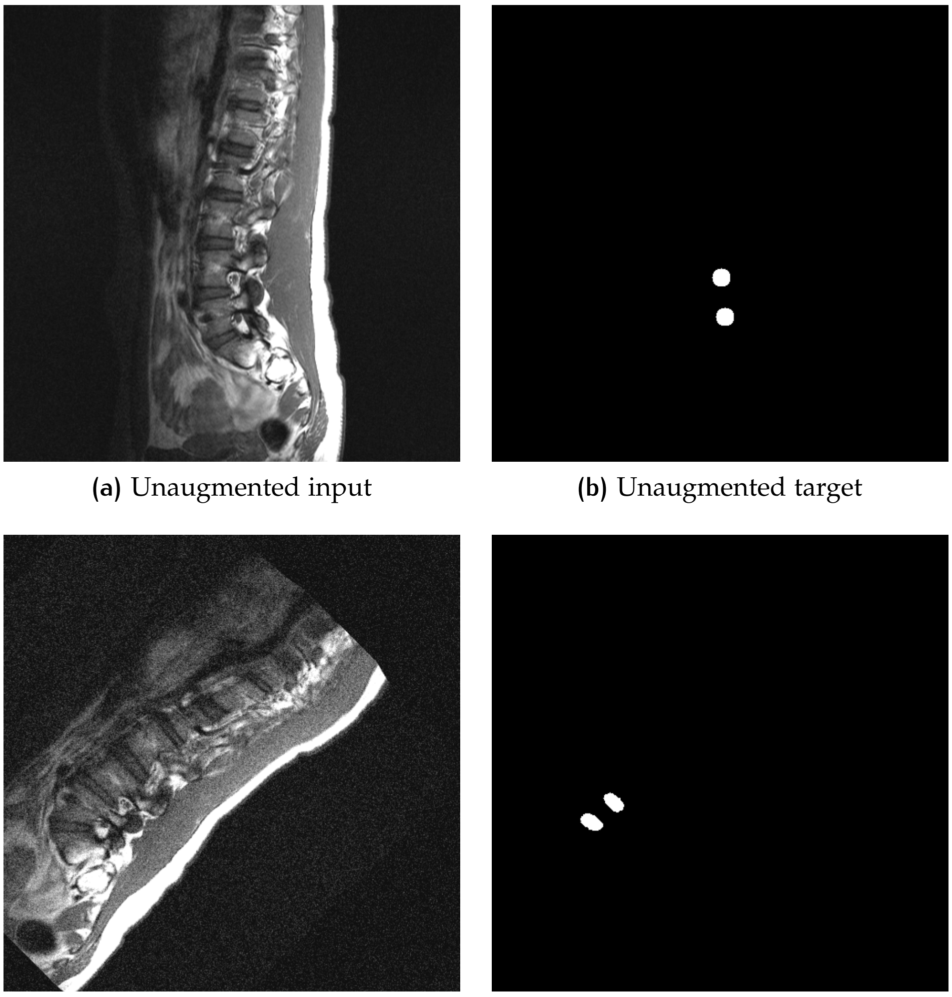
As is conventional in machine learning the features (in our case the voxels) are standardized, i.e. for each voxel $v_i = (v_i - \mu)/\sigma$ where μ and σ are the mean and standard deviation of all voxels respectively.

The preprocessing pipeline also includes data augmentation. A volume is warped with random nonlinear deformations. This results in strange looking lumbar spines that cannot possibly exist. But even unrealistic data augmentation can improve performance [69]. Secondly, volumes are rotated around the x-axis with a random angle between $-45°$ and $45°$. Finally, noise sampled from $\mathcal{N}(0, 0.25)$ is added to the volume. Due to this randomized augmentation, the network will never process two identical inputs during training.

A patch with size `PatchSize` is randomly extracted from the augmented volume. The compatible `PatchSizes` depend on the network architecture. An examples is shown in Fig. 7.1. Testing and inference is done on unaugmented volumes, and training is done on (partially) augmented volumes.

7.3 NETWORKS

For both approaches to segmentation (single-channel and multi-channel output), we first design a base version. We make modifications to this base version to either try and improve it, or test the influence of a tech-



(a) Unaugmented input (b) Unaugmented target

(c) Augmented input (d) Augmented target

Figure 7.1: A slice from a prepreprocessed volume, both un- and augmented.

nique. The base versions are trained for 20 hours to see how many steps it takes for the the Dice coefficient to converge. The variants of the network are trained for that many steps[19] (a single step is a single batch processed).

As will become clear in Sec. 7.4, it is important to find a balance between the network's size, `BatchSize`, and `PatchSize`. For this reason, it is desirable that the number of slices per trainings sample is kept low. However, we still want to utilize the 3D spatial information. We discarded the idea of 3 slices in and 3 slices out, because that means that the output of the 1st and 3rd sliced are based on the extra context from only 1 other slice, i.e. the 2nd slice. The 2nd slice however, has extra context from both the first and third. Therefore, we choose to design networks that take 3 slices in, and output only 1. We are using 3 slices

19 For a fair comparison we kept the weights from the base version at the same number of steps.

as input and not 5 or more, because our annotator has sufficient extra context with 3 slices. If a human can do with 3 slices, so can a neural network given enough training[20].

7.4 SINGLENET

Our base architecture for the single-channel approach is a 3D U-NET based on V-NET [71], let us call it SINGLENET (Fig. 7.2). Just as V-NET, SINGLENET consists of a compressing and expanding branch with long skips connecting them. Furthermore, SINGLENET also has residual blocks after each compression and before each expansion. First we explain the difference between SINGLENET and V-NET, next the process of determining the hyperparameters is explained. After that we describe the modifications we made leading to the variations of SINGLENET. Finally, the results and some examples are examined.

7.4.1 Architecture

Whereas V-NET uses 4 levels (4 compression and expanding blocks), SINGLENET only uses 2. Also, we start with half the channels. Therefore, at each level in the rest of the network SINGLENET also has half the channels as we double the channels per level. We choose to use only $3\times3\times3$ instead of $5\times5\times5$ convolutions and ReLU instead of PReLU as activation function. Furthermore, we added batch normalization and dropout.

In the final prediction block we use a $3\times1\times1$ valid convolution with 1 output channel. As a result the output has a dimension of $1\times481\times481\times1$, i.e. a single slice with height and width 481 and a single channel. The sigmoid operation converts this to a probability. The values should be interpret as the probability that a voxel is part of one the 10 lumbar foramina.

The total number of parameters for SINGLENET is $179,115$. With respect to state-of-the art natural image networks, this is not a lot. Especially considering that the input is 3D.

The network is trained with the ADAM optimization algorithm.

7.4.2 Random hyperparameter search

To get a sense of what set of hyperparameters are reasonable, we used *random search* [92]. The ranges for each hyperparameter explored can be

20 Assuming the network is able to perform the task at all.

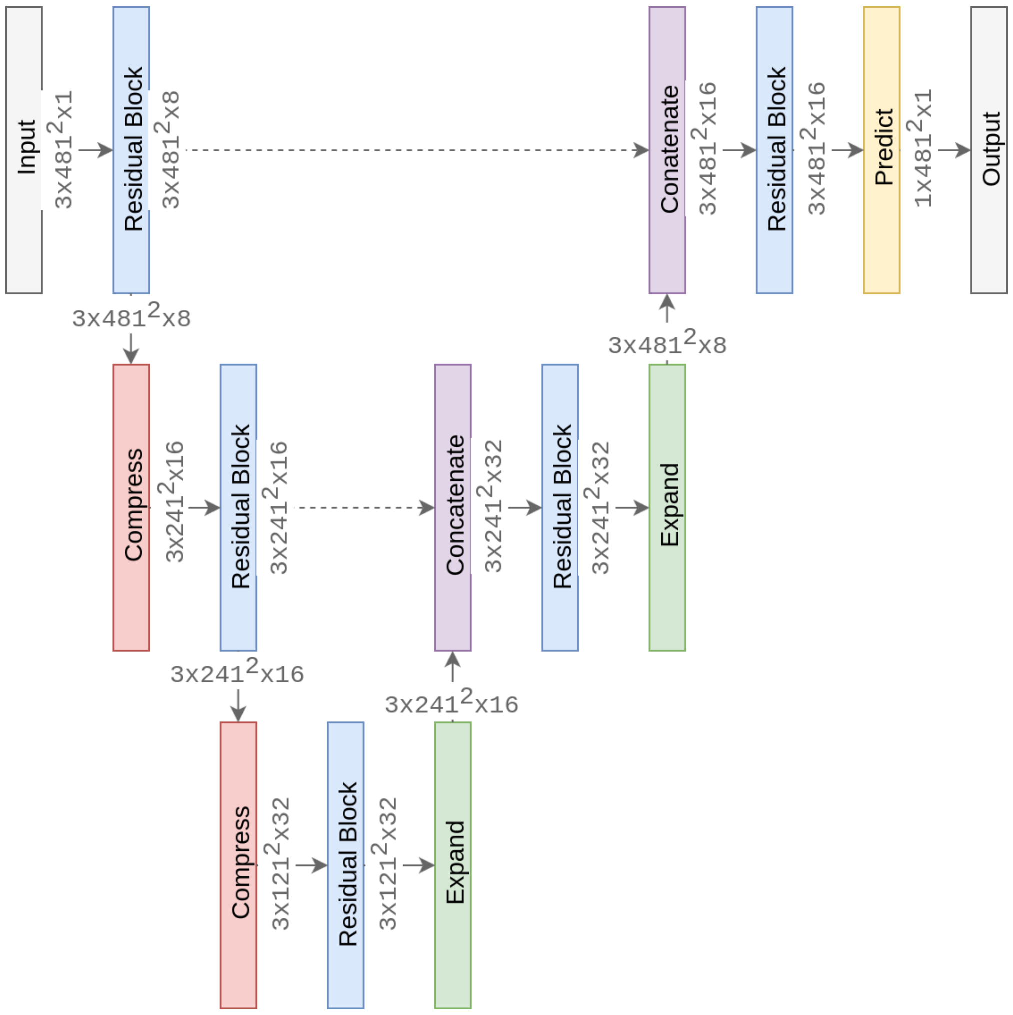


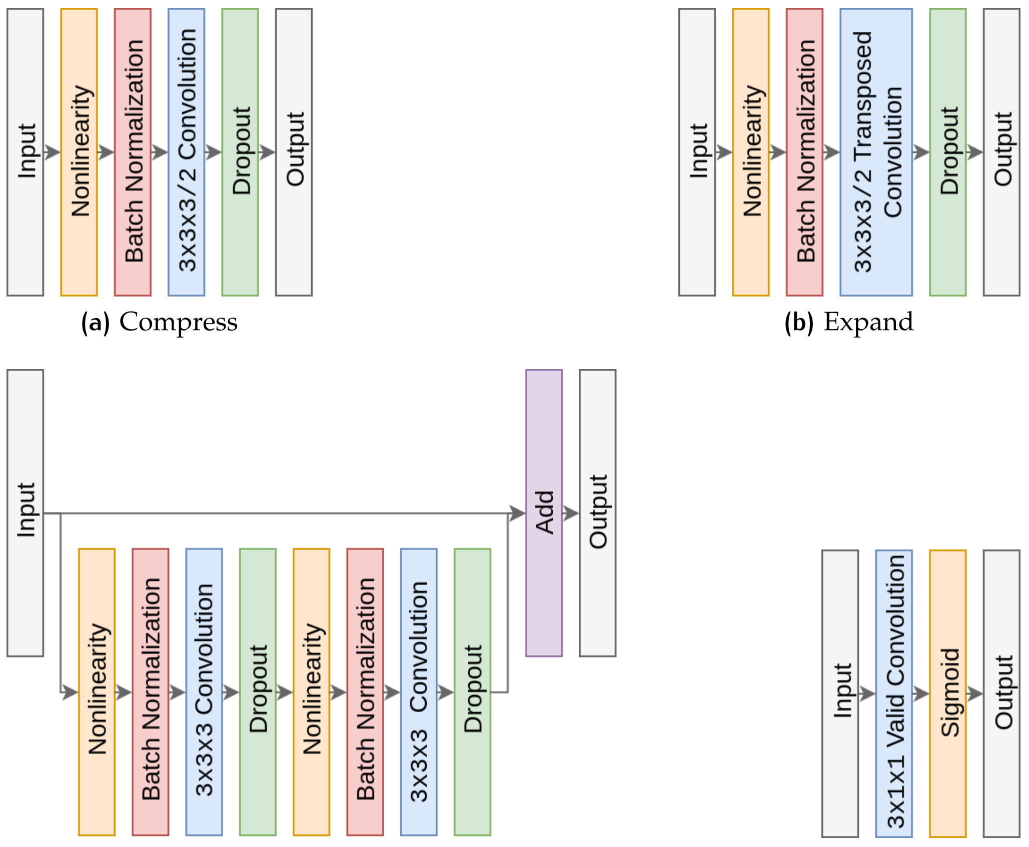
Figure 7.2: The SINGLENET architecture.

NAME	WEIGHT	P-VALUE	SINGLENET
`PatchSize`	14.869	0.0	481
`BatchSize`	9.171	0.0	4
`Blocks`	−4.456	0.061	2
`Channels`	4.146	0.096	8
`LearningRate`	−3.926	0.106	0.001
`KeepProbability`	−2.226	0.345	0.9

Table 7.2: The weights and p-values from the fitted linear regression model per hyperparameter and the final value for SINGLENET.

seen in Tab. A.1 and are based on the values from similar networks. We trained 51 networks with random hyperparameters each for 3 hours.

To aid us with the interpretation of the random hyperparameter search results, we fitted a linear regression model on the hyperparameter search



(a) Compress **(b)** Expand

(c) Residual Block **(d)** Predict

Figure 7.3: The modules used in the SINGLENET architecture.

results with Dice coefficient as target value (Tab. A.1). This gave us guidance on whether certain hyperparameters should be low or high (negative or positive weight) and how important they are (the p-value). Note that this is only used as a rough guideline and does not lead to the direct specification of SINGLENET.

As expected, the results indicate that some hyperparameters are more important than others. It is important to keep `PatchSize` and `BatchSize` as high as possible. Thus, processing more voxels per iteration increases performance. At the same time more channels (i.e. more parameters and activations) also increases performance. These two findings both require more memory. A balance between the two is necessary. In any case, the GPU's memory should be as full as possible. The random search suggests a `PatchSize` of 481×481 (the highest), a `BatchSize` of 4, and 8 initial channels.

Surprisingly, the number of levels is to be kept low. This is surprising because the long skips between down- and upscaling branches are the theoretical reason U-NETS perform well. However, for SINGLENET more than 2 blocks does not yield better performance. According to the data, a better approach to increase the model size is to keep the number of

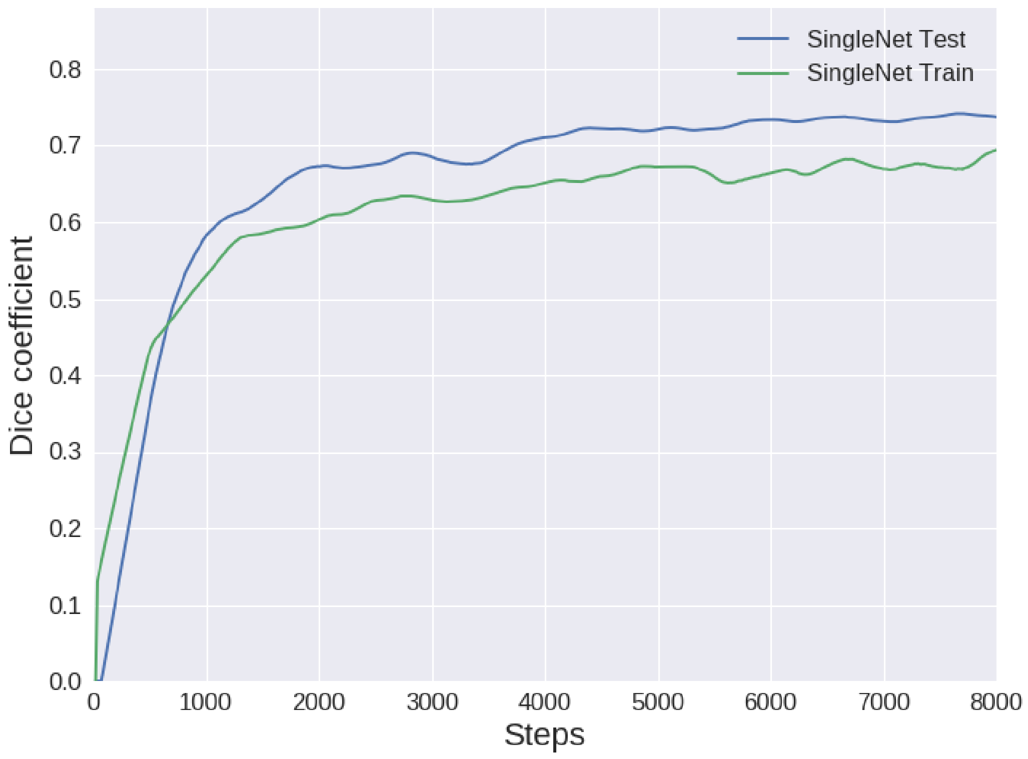


Figure 7.4: The Dice coefficients of SingleNet while training the network.

blocks low and increase the number of channels. We speculate that this is because the network does not require a large receptive field to determine whether a voxels is part of a foramina. SingleNet has a receptive field of 42×42×3 voxels. Adding more blocks would only increase the complexity, a better use for more parameters is to increase the number of channels.

The learning rate has little influence. This is because Adam uses adaptive strategies to optimize the weight update procedure. Therefore, it works with a wide range of (sensible) learning rates.

7.4.3 Variations

We experimented with numerous changes to the architecture that might increase its performance. The activation function might have significant impact on the performance on the model, so we trained models with the ELU and CReLU. Also, we increased the capacity of the network by defining a bigger network with `Channels` increased to 12 and at the same time `KeepProbability` decreased to 0.7. The Dice coefficient of the test batches while training these 'improved' networks are shown in Fig. B.1.

Too gain insight in the added value of batch normalization and the He initialization scheme, we trained the network without these methods

(Fig. B.2). To approximate how much more annotated data was needed, we also trained the network with subsets of the trainings set of different sizes (Fig. B.3).

The Dice coefficient can show how well the network learns. But since it is a proxy loss, it does not give a complete representation for how well the network performs on the final task. Nonetheless, the ability to learn is a crucial component of DNNs. Changing the activation function had no influence on the Dice coefficient curve. increasing the capacity even made it worse. Although this could also be caused by decreasing the `KeepProbability` too much. We can clearly see in Fig. B.2 that both He initialization and batch normalization have a positive effect on the curve. The theoretical explanations for this effect are given in Chap. 3. Surprisingly, the network seems to benefit little from more data. A trainings set size of 50 and more yields a similar Dice coefficient curve.

7.4.4 Single-channel localization

As mentioned before, SingleNet takes 3 slices as input and gives 1 slice as output, to create a full probability volume we pad the original volume with two empty (all zero) slices at the x-axis (a slice appended to the right and left side of the volume). Subsequently, we divide the volume into overlapping subvolumes of 3 slices, such that each slice of the original volume is the middle slice of a subvolume.

Recall that the SingleNets take a fixed input[21] which is likely not the same as the slice. Therefore, we divide the subvolume into 4 even smaller subvolumes with the same depth (3 slices) and compatible width and height. We do inference on each of these 4 subvolumes and merge them together for a probability map of the slice. Next we concatenate the probability maps together for the probability volume.

As said before, the segmentation is a proxy task that provides an easier way of determining the coordinates. The next task is to actually localize the 10 foramina from the probability volumes.

First we round the probabilities to either 1 or 0, as a result we a have binary volume that we will use to extract coordinates from. To do that, we detect all clusters[22]. These are the `candidates`. For each `candidate` we determine the center and size (the number of 1s in the cluster).

Because we want to predict the foramina L1-5 for both the left and right side, it is useful to split the volume into two sides. But it might be possible that the middle of the spine does not align with the middle of the volume. Therefore we take the x value of each candidate center

21 It is not required for the network to take the same input as the trainings patch, however, SingleNets (and U-Nets by extension) have a limited range of compatible input sizes depending on their architecture.

22 A contiguous shape of 1s in the 3D volume, our networks output cilinder shapes.

and cluster them into 2 clusters with the *k-means* algorithm [93]. We consider the slice in between the two clusters as the middle slice. Note that this approach requires that candidates (i.e. the predictions made by the network) have to be of a certain quality for it to work. If there are 5 or less candidates (poor quality prediction) or the spread among the candidates' x values is less than 4 (all candidates come from one side), we take the middle slice of the volume.

For each side we take the 5 candidates with the highest size, as this is an indication of the network confidence in the candidate. The 5 biggest candidates are sorted in the z-axis and labeled from top to down as L1 to L5.

7.4.5 Evaluation

The next stage is do inference on the resulting 100 test volumes and determine the predicted coordinates. A scatter plot of the distances of predictions made by SingleNet is shown in Fig. 7.5. A clear pattern is visible. Most predictions are close to the actual coordinate (within 5 millimeters), but then we see a jump to around 17 millimeter error distance. This is the approximately the distance to a neighboring foramen. The mean error distance, recall, and skipped foramen metrics, for all SingleNet variations can be found in Tab. 7.3, 7.4, and B.1, respectively. With a binary classification at hand, we can compare the models for significant differences. The p-value for all SingleNet variations pairs are displayed in Tab. B.2.

The ELU and CReLU networks perform better both in terms of mean error distance and number of foramina locations correctly predicted. Furthermore, this performance gain is statistically significant[23]. Among the ELU and CReLU networks themselves a weak difference is found ($p = 0.10$), they most likely produce a similar network. This make the ELU network favorable, because CReLU increases the number of parameters and memory requirements.

Interestingly, the networks that are trained with 150 and 250 trainings samples perform worse, yet the difference is not significant.

7.4.5.1 *Examples*

In Tab. 7.3 and 7.4 can be seen that SingleNet has more difficulties with L1 than the other foramina. Upon inspection of the cases that went wrong we saw that this was due to over generalization by the network. SingleNet has a small receptive field, it only has limited context around the foramen to determine if it is a (lumbar) foramen. As it turns out this makes it difficult to distinguish between the lumbar foramina and

23 Assuming the commonly used $p < 0.05$ threshold.

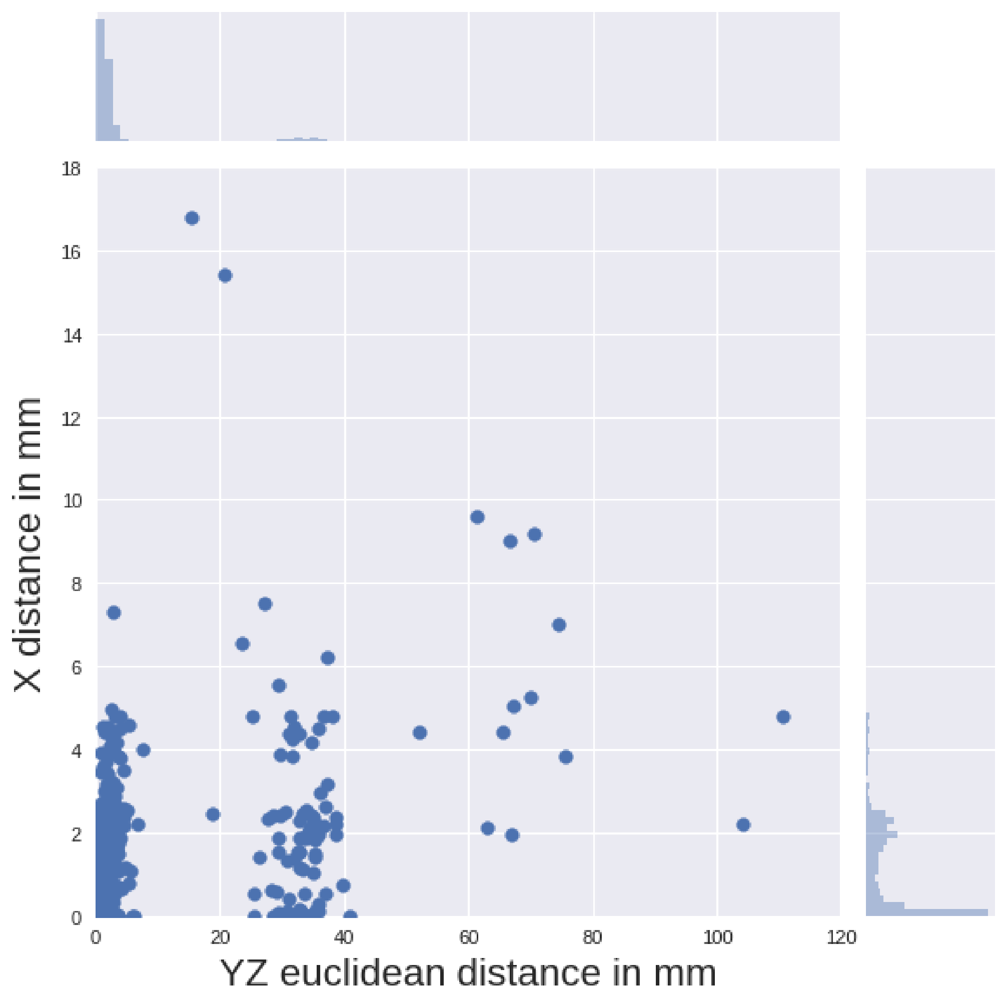


Figure 7.5: The distance from the predicted foramina centers to the real centers in the x and yz axises by the ELU variant of SingleNet.

thoracic foramina that reside above them. This is problematic as shown in Fig. 7.6 where the L5 candidate is not included the top 5 biggest candidates.

Although it makes the labeling of foramina more difficult for us, it is encouraging to see how the network generalizes from only lumbar foramina as positive samples, to the thoracic foramina. Arguably, this means that the network truly generalizes.

Another more disastrous flaw of single-channel output, is that the labeling of the foramina happens after prediction. As a result, we have no way of knowing whether the network missed a foramen. This has more far-reaching implications than might seem at first. An example is given in Fig. 7.7. Even though the network correctly distinguishes the T12 foramen from the lumbar foramina, as can be derived from the small size of the T12 candidate, it will miss label all foramen. By

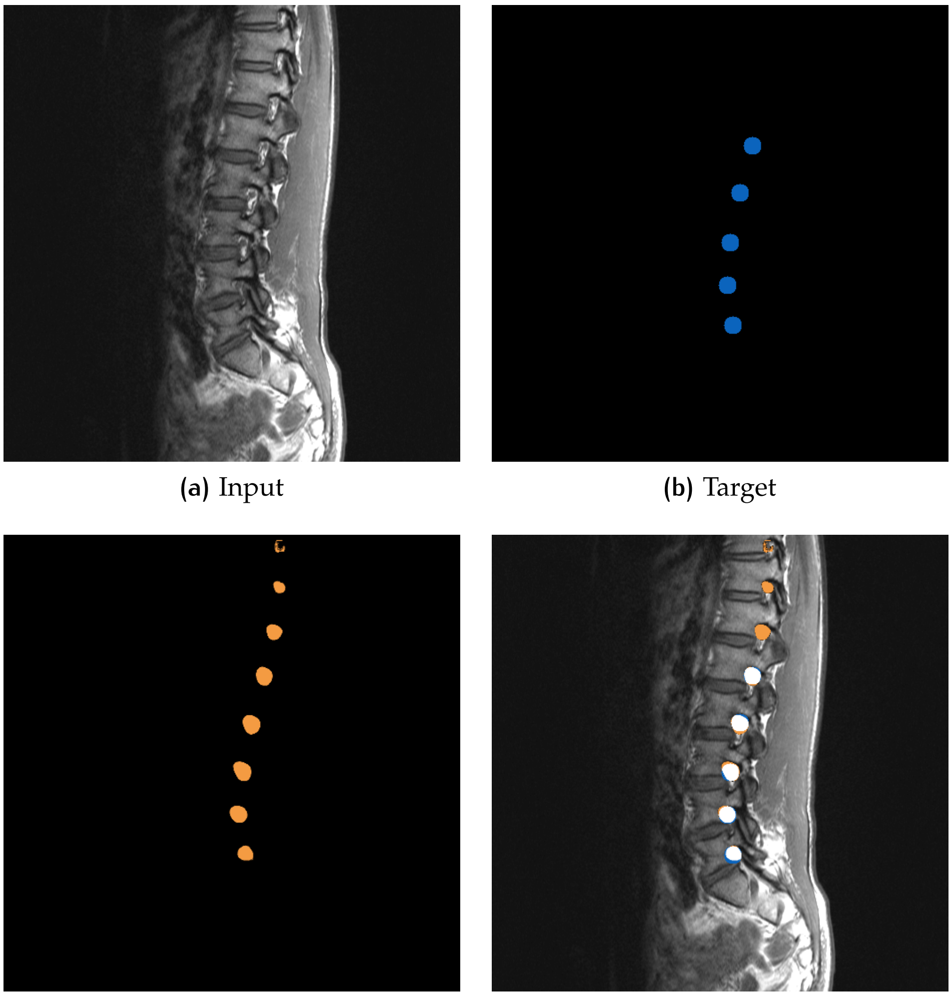
NETWORK	MEAN ERROR DISTANCE IN MM					
	ALL	L1	L2	L3	L4	L5
Base	5.78	10.04	5.08	4.78	4.36	4.70
CReLU	5.07	7.80	4.42	4.46	4.08	4.68
ELU	5.20	10.35	4.05	3.78	3.87	3.99
More capacity	9.79	17.51	8.19	6.25	5.91	11.11
No He Initialization	18.83	30.69	19.39	14.99	13.05	16.33
No Batch Normalization	11.17	21.42	10.83	7.89	6.66	9.04
Trainings Set Size 250	6.42	10.35	6.26	5.64	5.02	5.05
Trainings Set Size 150	7.40	13.69	6.43	5.02	4.02	7.86
Trainings Set Size 50	7.70	12.86	7.01	6.15	6.18	6.36
Trainings Set Size 5	16.78	25.47	17.41	13.50	12.11	16.29
Trainings Set Size 1	71.79	68.58	72.98	75.22	72.42	69.65

Table 7.3: The mean error distance for the SingleNet variations.

NETWORK	PERCENTAGE BELOW 5 MM					
	ALL	L1	L2	L3	L4	L5
Base	85.4	73.0	89.0	89.5	89.0	86.5
CReLU	87.5	$\boxed{79.5}$	89.5	90.0	90.0	88.5
ELU	$\boxed{89.2}$	75.5	$\boxed{92.5}$	$\boxed{93.0}$	$\boxed{93.0}$	$\boxed{92.0}$
More capacity	79.7	60.0	83.0	86.0	86.5	83.0
No He Initialization	62.3	42.0	58.5	65.5	70.5	75.0
No Batch Normalization	75.6	52.5	76.5	82.0	84.0	83.0
Trainings Set Size 250	84.1	73.0	86.0	87.5	87.5	86.5
Trainings Set Size 150	84.3	68.0	85.5	89.0	91.0	88.0
Trainings Set Size 50	82.3	69.5	84.5	85.5	86.0	86.0
Trainings Set Size 5	59.9	37.5	57.0	66.0	70.0	69.0
Trainings Set Size 1	6.3	3.0	4.0	9.5	7.5	7.5

Table 7.4: The percentage of coordinate predictions that were within 5 millimeter of the target coordinate for the SingleNet variations.

missing the L5 foramen it will label L4 as L5, L3 as L4, etc. Making every prediction wrong.


(a) Input (b) Target

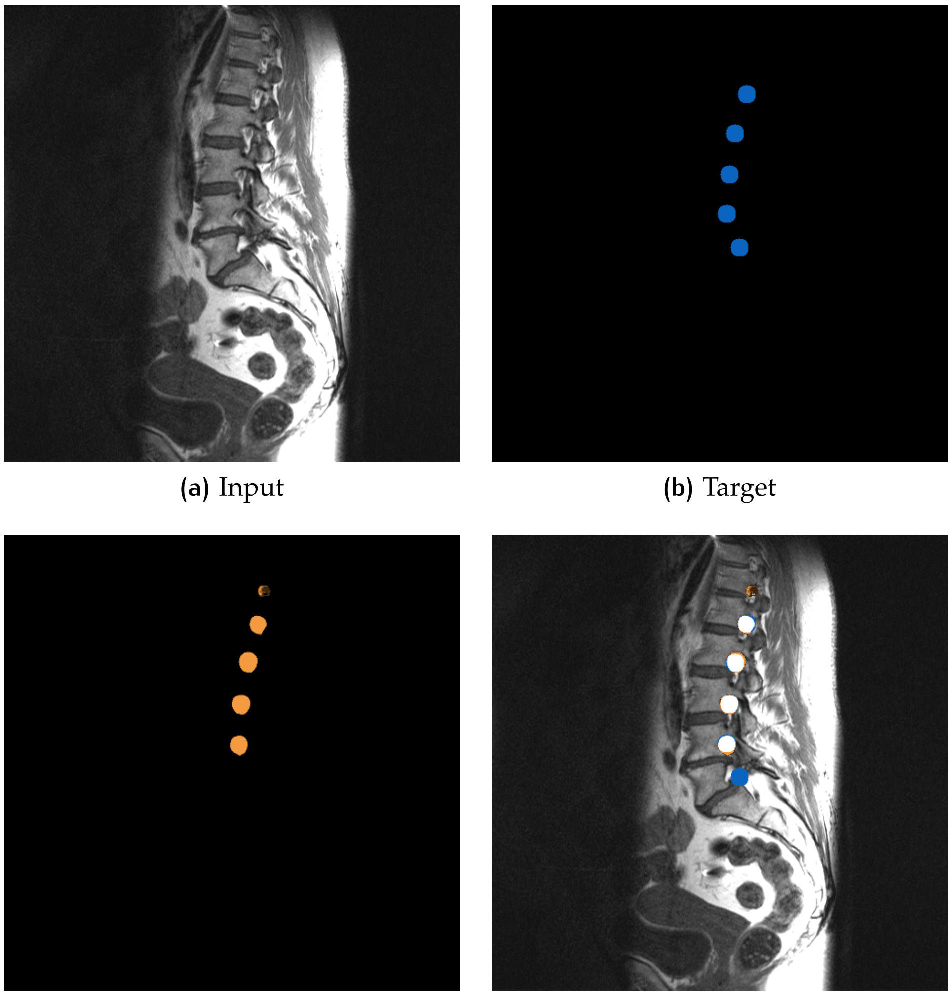
(c) Output (d) Combination

Figure 7.6: An example of overgeneralization to the T12 and T11 foramina by SINGLENET.
The target and prediction colors are chosen such that when they overlap they become white.

7.5 MULTINET

To prevent the problem that arises with a single-channel output, we create a network that outputs the probabilities whether a voxel belongs to one of the foramina at each level (i.e. a multi-channel output). With a multi-channel output we create a candidate per level. As a result, when L5 is not recognized, only L5 will have no candidate and have a wrong prediction. The rest of the levels still get a correct prediction. In other words, the labeling of the foramina is incorporated in the network architecture.

In the rest of the section we will first describe the architectural differences of MULTINET with SINGLENET. Subsequently, we explain the need


(a) Input (b) Target

(c) Output (d) Combination

Figure 7.7: An example of a missed L5 foramen by SingleNet.

for a modification to the Dice coefficient. Next, we enumerate all variations of MultiNet (Sec. 7.5.2). A multi-channel output requires a small modification to the coordinate extraction from the probability volume with respect to a single-channel output, which we will clarify in Sec. 7.5.3. Finally in Sec. 7.5.4 we will evaluate the results and give some examples of predications made by MultiNet.

7.5.1 Architecture

This network requires a bigger receptive field because it needs more context to determine to which level the foramina belongs. We increased the number of blocks the network has to 5, as this effectively increases the receptive field for each output node. To restrain the size of the network we introduce a new hyperparameter called `MaxChannels` and set it to 16. The layers that control the number of channels (i.e. the compress and expand layer) are capped in the number of output channels by `MaxChannels`. We call the new network MultiNet. The number of parameters is 397, 371. Even though we put a constraint on the size by using `MaxChannels`, the network is too big for a `BatchSize` of 4. Thus, we lower the batch size to 2. Lastly, we added supervision. Instead of extending the loss to take the auxillary classifiers from the lower levels into account, we iteratively interpolate the output to the size of the level above and add them together in similar fashion to Kayalibay *et al.* [73].

One of the first things that immediately becomes clear is that MultiNet has trouble learning. In Tab. (C.1) is shown how many foramina the MultiNet base version misses. It misses all L1, L2, and L3 foramina. We hypothesized that this was due to the fact that the lower foramina are deeper[24] than the higher foramina. As a result, the lower foramina are on average marked on more slices. In other words, we are dealing with a class imbalance problem. SingleNet did not have this problem because it had only 1 class to predict, i.e. foramen or non-foramen. To compensate for the class imbalance we normalize the Dice coefficient. For a correct normalization we counted how many markers each level has (Tab. 7.5). Instead of vectorizing the full output and calculating the Dice coefficient, we determine the Dice coefficient per channel and subsequently do a weighted sum. This normalized Dice coefficient deals with the class imbalance improving recall from 38.5% to 96.5% (Tab. 7.7). All variations made on MultiNet make use of the normalized Dice coefficient.

	L1	L2	L3	L4	L5
Count	1154	1274	1582	2060	2267
Weight	1.00	0.91	0.73	0.56	0.51

Table 7.5: The marker counts and weights for normalization per level.

7.5.2 Variations

To start, we trained a network without supervision to acquire insight into the influence of supervision.

As the normalization of the Dice coefficient had such a big impact, we explored the use of other loss functions. One of the most common is cross entropy. However, cross entropy is very susceptible to class bias with imbalanced data. Similar to the normalized Dice coefficient, we normalize the cross entropy loss. A positive and negative weight is determined, such that they each can contribute half of the total loss. Sub-

24 Pysically deeper and therefore visible on more slices.

sequently, we construct a weight map of the same size as the target segmentation, that has the positive weight at the corresponding positive voxels and the negative weight at the corresponding negative voxels. Next we multiply the cross entropy loss with this weight map. Both the sigmoid and softmax cross entropy loss are trained.

To improve the model we trained a networks with more capacity (`Channels` increased to 12, `MaxChannels` to 32, and `KeepProbability` decreased to 0.75), the ELU and CReLU activation functions — since CReLUs increased the number of parameters by a lot we had to lower the batch size to 1, therefore we also trained 'CReLU Less' which has slightly less channels (`MaxChannels` set to 14 instead of 16) and can process batch of size 2 — and a softmax activation as output layer.

By reducing the capacity of the network, we can train networks that are able to process bigger batches. The network 'Less capacity' has 6 channels at the first level and 8 in the rest of the network. This allows for training with batch size 4.

Another approach to reducing the memory requirements is by reducing the input. However, we know from the SingleNet random hyperparameter search that a big `PatchSize` is essential for good performance. This is even more so for MultiNet, which requires more context to determine the foramen level. For that reason we reduced the input by doubling the `NormSpacing` to 1.4 millimeters. As a result the volumes shrink half in size, yet are still complete volumes. The `PatchSize` for this network is set to 225×225. This allows for training with a batch size of 8. Additionally, a combination of less capacity and a lower resolution with a batch size of 16 was trained.

Finally, we experimented with the varying number of trainings set samples.

7.5.3 Multi-channel localization

As mentioned before, MultiNet has 5 output channels. This makes the extraction easier since the network tells us how to label the candidates. Similar to SingleNet the output is 1 slice and does (usually) not cover the entire slice as the network only takes limited set of `PatchSizes`. So again, we pad the original volume with two empty (all zero) slices at the x-axis (a slice appended to the right and left side) and divide the volume into overlapping subvolumes of 3 slices, such that each slice of the original volume is the middle slice of a subvolume. Subsequently, we divide the subvolume into 4 even smaller subvolumes with the same depth (3 slices) and compatible width and height. We do inference on each of these 4 subvolumes and merge them together for a probability map of the slice. Finally, we concatenate the probability maps together for the probability volume.

NETWORK	MEAN ERROR DISTANCE IN MM					
	ALL	L1	L2	L3	L4	L5
Base	2.36	—	—	—	2.18	2.54
Normalized	2.83	5.28	2.42	2.21	1.87	2.38
Unsupervised	2.98	4.93	2.72	2.59	2.23	2.48
ELU	2.48	3.07	2.22	2.21	2.20	2.69
CReLU	7.33	11.09	7.72	7.21	4.23	4.81
CReLU Less	2.52	3.64	2.31	2.24	1.92	2.47
More capacity	2.58	3.45	2.21	2.74	1.91	2.58
Softmax	3.12	5.74	2.90	2.32	2.37	2.30
Sigmoid Cross Entropy	13.61	9.08	6.21	3.44	10.23	41.46
Softmax Cross Entropy	14.58	14.91	5.04	5.76	11.1	35.62
Lower res.	2.08	2.17	1.97	1.94	1.89	2.45
Lower res. 60k	1.97	1.87	1.72	1.99	1.83	2.43
Lower res. 120k	2.44	3.01	2.44	2.34	2.01	2.39
Less capacity	2.32	3.62	1.93	1.89	1.78	2.39
Less capacity + lower res.	2.43	2.56	2.51	2.50	2.17	2.41
Training Set Size 10	4.91	10.75	6.68	3.47	2.13	2.46
Training Set Size 25	2.82	4.21	2.55	2.93	1.95	2.49
Training Set Size 50	3.27	5.05	3.50	3.04	2.41	2.41
Training Set Size 100	3.44	6.21	3.24	2.56	2.45	2.80
Training Set Size 200	3.46	5.44	3.33	3.39	2.38	2.75

Table 7.6: The mean error distance for the MultiNet variations.

As with SingleNet we split the volume into two sides with the *k-means* algorithm [93]. Instead of taking the 5 highest count candidates from the single-channel, we take the highest count candidate per channel as final coordinate prediction.

7.5.4 Evaluation

Supervision has no significance influence on the performance of the network.

The CReLU network performs significantly worse. Because the CReLU network with less capacity and a batch size of 2 performs on par with

NETWORK	PERCENTAGE BELOW 5 MM					
	ALL	L1	L2	L3	L4	L5
Base	38.5	0.0	0.0	0.0	97.0	95.5
Normalized	94.5	87.0	96.0	96.5	$\boxed{98.0}$	95.0
Unsupervised	93.8	85.5	95.5	96.0	96.5	95.5
ELU	95.6	94.5	96.0	96.5	97.0	94.0
CReLU	62.0	67.5	64.0	59.0	63.5	56.0
CReLU Less	95.7	92.5	97.0	97.0	97.5	94.5
More capacity	95.9	93.0	97.0	96.5	$\boxed{98.0}$	95.0
Softmax	93.7	84.5	95.0	96.5	97.5	95.0
Sigmoid Cross Entropy	67.3	77.0	76.5	82.5	56.5	44.0
Softmax Cross Entropy	58.2	52.5	79.0	68.5	48.5	42.5
Lower res.	97.3	97.0	98.0	$\boxed{98.0}$	$\boxed{98.0}$	95.5
Lower res. 60k	$\boxed{97.5}$	$\boxed{98.0}$	$\boxed{98.5}$	97.5	$\boxed{98.0}$	95.5
Lower res. 120k	96.1	94.5	96.5	96.5	97.5	95.5
Less capacity	95.8	90.5	98.0	97.5	$\boxed{98.0}$	95.0
Less capacity + lower res.	96.2	95.5	96.5	96.5	96.5	$\boxed{96.0}$
Training Set Size 10	83.9	57.0	79.5	91.5	97.0	94.5
Training Set Size 25	93.2	85.5	95.0	95.0	97.5	93.0
Training Set Size 50	92.6	86.0	93.5	94.0	96.0	93.5
Training Set Size 100	91.9	83.0	91.5	95.0	96.0	94.0
Training Set Size 200	92.9	87.5	94.0	94.0	96.0	93.0

Table 7.7: The percentage of coordinate predictions that were within 5 millimeter of the target coordinate for the MultiNet variations.

other networks, we have to conclude that a batch size of 1 is not enough for stable training. The softmax activation as output layer prevents the network from labeling a foramen as two level simultaneously, however, this seems to have no influence on the performance. Suggesting that this is no problem. The networks with more capacity, ELU activations and CReLU with less capacity are all better than the base variant with a statistical significant difference. Interestingly, among themselves there is no significant difference. It seems that the base variant can be improved and that the improved variants all compensate for the same flaws.

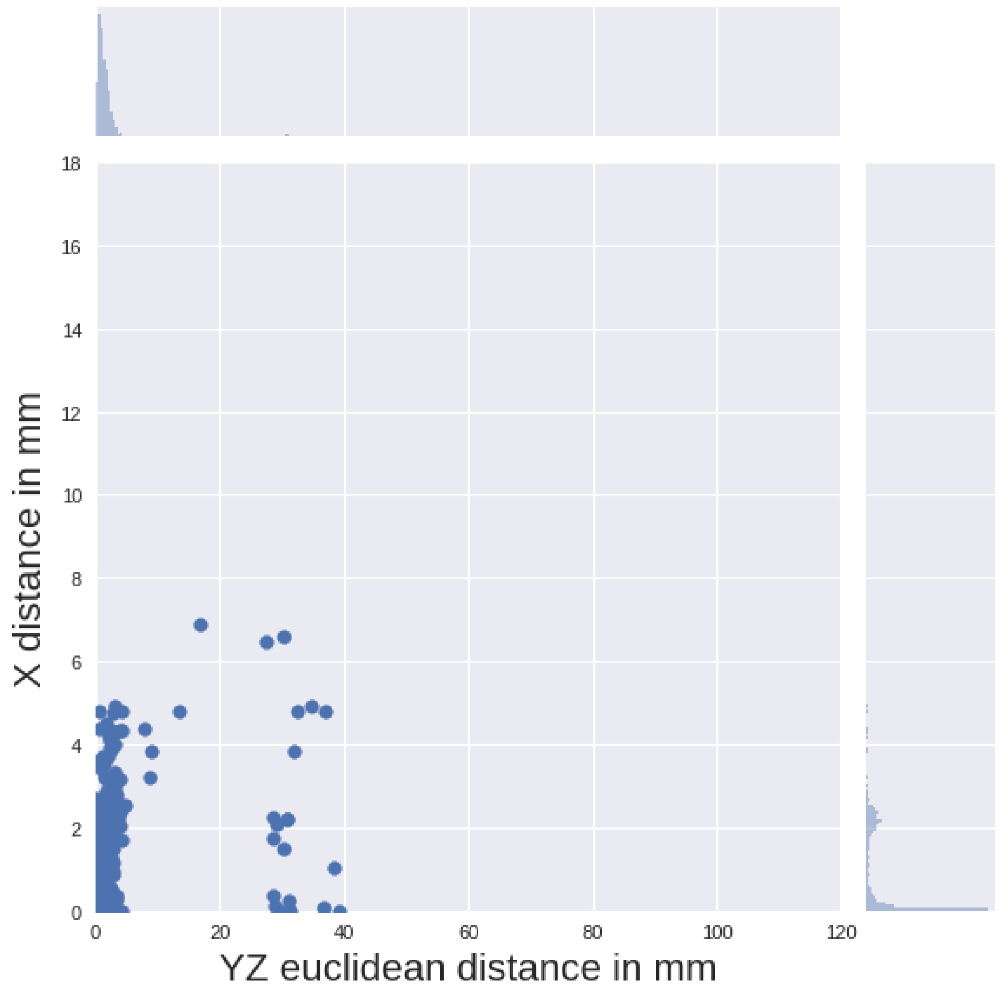


Figure 7.8: The distances of the predicted foramina centers to the target centers in the x and yz axises by the lower resolution variant of MultiNet.

Cross entropy punishes more severely if the predicted value is more wrong. For instance, assume that $y = 1$ and we make 3 different prediction $\hat{y} \in \{0.2, 0.4, 0.6\}$. The losses are $-\ln(0.2) \approx 1.61$, $-\ln(0.4) \approx 0.92$, and $-\ln(0.6) \approx 0.51$, respectively. Even though the difference between 0.2 and 0.4 is the same as the difference between 0.4 and 0.6, the difference in loss is bigger because it is further away from 1. This is not alike to D_{fuzzy}. A cross entropy loss makes the network perform significantly worse.

Remarkably, the network with less capacity manages to reach the same performance as the base network and networks with variations aimed at improving the performance. The smaller network has less memory requirements, making it faster to train and performance inference with. Overall the smaller network is favorable over the base version.

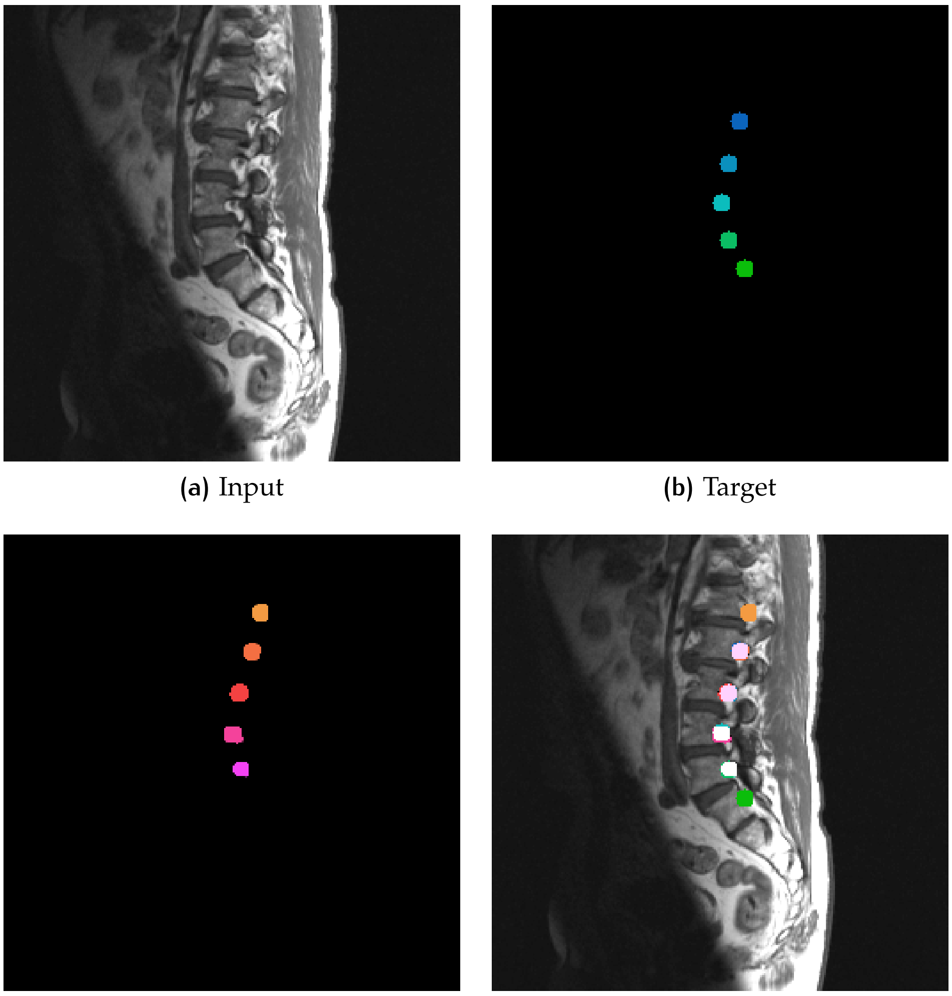
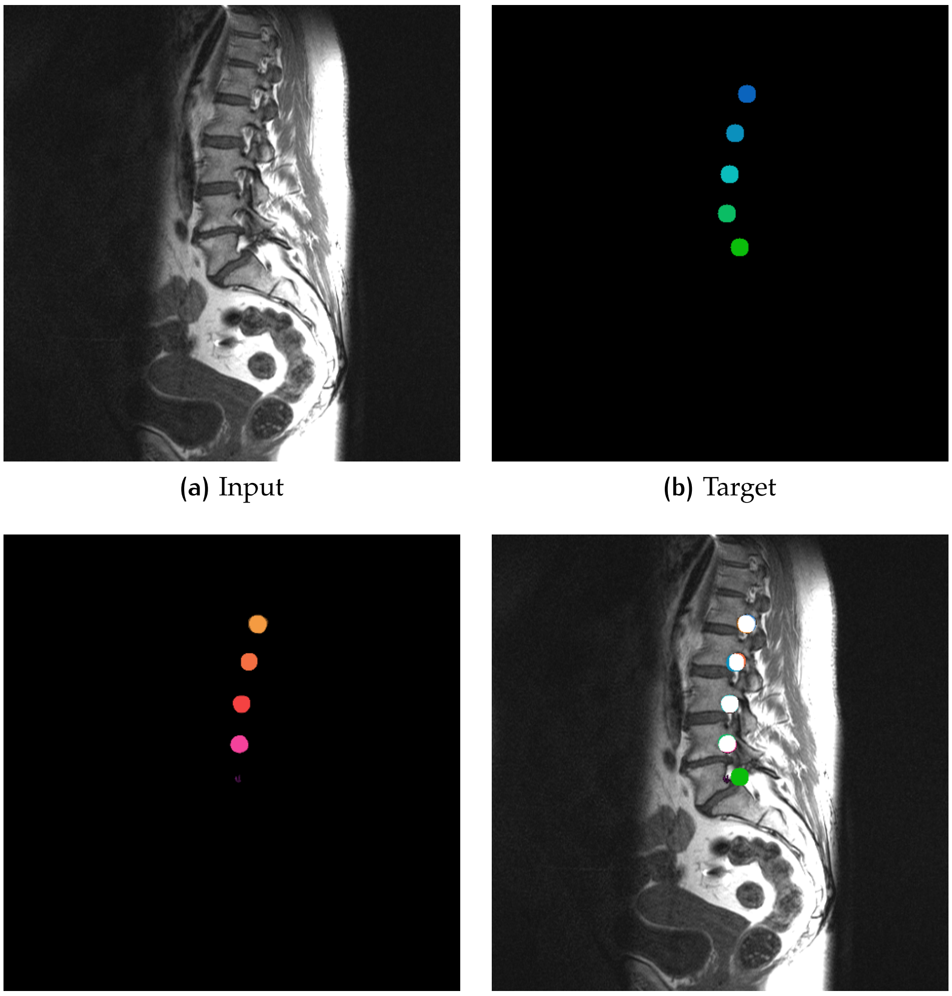


Figure 7.9: A slice that is predicted a full shift up by the lower resolution variant of MULTINET.
For each level the prediction and target color pairs are chosen such that when they overlap they become white.

When the input is normalized to a `PixelSpacing` of 1.4 millimeters, more samples can be processed per trainings iteration. Perhaps more importantly, the receptive field of nodes in the network effectively cover a bigger area of the scan[25]. With more context available the network can make a better prediction about the foramen's level. A network trained with lower resolution scans perform best of all variants. Moreover, the difference is significant. We wanted to create the best network possible and trained the network for 60,000 and 120,000 steps. These have little and insignificant difference in results. Confirming that after 30,000 steps the network is converged.

25 A bigger receptive field in terms of millimeters, not in terms of voxels.



(a) Input (b) Target

(c) Output (d) Combination

Figure 7.10: An example of a missed L5 foramen by MultiNet.

A network with less capacity performs equally well as the base variant. But, when we apply this strategy to a network that trains on scans with a lower resolution, this effect is not visible. In fact, lowering the resolution of patches for the smaller network does not improve performance. Or from an alternative perspective, reducing the network size of the 'lower resolution' network hurts performance.

As with SingleNet, we trained the (normalized) base variant of MultiNet using various number of trainings samples. Whereas the results indicated that SingleNet managed to reach a similar network with 150 samples, MultiNet requires at least more than 200.

7.5.4.1 *Examples*

Let us analyze some of the cases where MultiNet is inaccurate. Earlier is mentioned that SingleNet has trouble correctly labeling foramina candidates when one is missing, such as in the case of Fig. 7.7. In Fig. 7.10 the same slice is shown but the predication is made by MultiNet. As can be seen, MultiNet classifies all correctly, except for L5. SingleNet makes a similar prediction, but since we label after we created the segmentation, SingleNet misses all. This is an example where SingleNet is outperformed by MultiNet.

There is 1 case where even the best variant of MultiNet is completely wrong. As can be seen in Fig. 7.9, not a single foramen is correctly predicted. MultiNet predicted the labels one level too high, i.e. L1 as T12, L2 as L1, etc. All variants of MultiNet have make the same mistake in this case.

Strangely enough, there are 2 cases where MultiNet has no trouble with 1 side, but does a full shift upwards for the other side. In both cases the L4 disc is degenerated (Fig. C.7).

The rest of the mistakes come from 3 volumes and are all cases as Fig. 7.10 and C.6. These patient all have a disease such as spondylosis or heavy foraminal stenosis.

8 CONCLUSION AND DISCUSSION

8.1 CONCLUSION

First and foremost, we succeeded in our goal to design a deep neural network to segment a lumbar spine from a MRI scan for the purpose of localizing the lumbar foramina. The best network — a MultiNet trained on volume normalized with a `NormSpacing` of 1.4 millimeter — makes a prediction for all foramina, has a mean error distance of 1.97 millimeter, and 97.5% of all predictions are below 5 millimeter. We set out to answer to the following question: *Is it possible to design and train a deep neural network that has viable performance on the task of lumbar foramina localization on MRI scans without using data driven postprocessing techniques or hand-crafted features?* It is ambiguous to set a threshold for when the performance is viable. It is safe to say however, that 97.5% is viable.

Our first approach was to create a binary segmentation network, called SingleNet, that distinguishes between lumbar foramina and other voxels. SingleNet has difficulties distinguishing the lumbar foramina from non-lumbar (most notably, thoracic) foramina. Furthermore, labeling proved to be troublesome when one of the foramina is not recognized by the network.

To overcome these flaws we built a network with a bigger receptive field and 5 output channels, called MultiNet. After normalizing the Dice coefficient to account for the class imbalance among foramina levels, this improved performance by a lot. Surprisingly, we found that most architectural modifications made little difference or made it worse. Most performance gain was achieved by lowering the resolution of the input.

Our approach makes use of deep neural networks, that have the advantage over other techniques that we are not required to manually engineer features. Nor do we use data driven postprocessing techniques. Our network learns all it needs from the data itself. This makes our approach more robust to chances in the task requirements and more widely applicable than models designed in past research.

8.2 DISCUSSION

Although our best network — MultiNet trained on lower resolution inputs — performs excellent, it still makes wrong predictions. Mostly when dealing with rare cases such as spondylosis. An interesting follow up research question is to examine whether there is a correlation between the certainty of the network and the chance that the prediction is wrong. As for our goal it is worse to make a wrong prediction instead of no prediction, we could abstain from making a prediction if the network is not entirely certain.

Deep neural networks are inherently good at interpolating patterns from the trainings data rather than extrapolating. In other words, for a network to recognize rare cases it requires to have seen plenty of such cases. Our network might not have had favorable circumstances. To ensure that the network has seen all diseases one could create a separate trainings set filled with such cases and feed them, according to some ratio, interspersed with the 'common' trainings samples.

As stenosed foramina look different from common 'normal' foramina by definition, it is important that the localization network can deal with abnormal looking foramina. Otherwise it might do inaccurate predictions for stenosed foramina defeating its purpose.

This study shows that deep convolutional neural networks that segment with the purpose of localization can benefit from lowering the resolution of the input. This is plausible, as the ultimate goal is not to create an accurate segmentation that requires precise information from the input, but to create a probability volume for localization which requires less fine-grained information and more surrounding context.

Further research is needed to establish the optimal `NormSpacing`. The network benefits from the extra context, but must still be able to recognize foramina. Especially when dealing with infrequent diseases such as spondylosis, it might be even harder to recognize foramen at low resolutions.

Alternatively, an architecture that combines both the approaches of SingleNet and MultiNet can be designed. If SingleNet is trained to recognize all foramina with high resolution inputs and MultiNet is trained to label both recognize and label the foramina at low resolution, we could somehow combine the network for better performance. Where MultiNet expects a foramen but does not recognizes one, the output of SingleNet might be of aid.

Finally, the data validity can be questioned. A radiologist vowed for our foramina localization competency, yet in the end it would be ideal if a radiologist does all marker annotations or verifies all annotations done by us.

BIBLIOGRAPHY

[1] Rubin, G. D., "Data explosion: The challenge of multidetector-row CT," *European Journal of Radiology*, vol. 36, no. 2, pp. 74–80, **2000**. DOI: 10.1016/S0720-048X(00)00270-9 (see p. 1).

[2] Bhargavan, M., Kaye, A. H., Forman, H. P., *et al.*, "Workload of radiologists in united states in 2006–2007 and trends since 1991–1992," *Radiology*, vol. 252, no. 2, pp. 458–467, **2009**. DOI: 10.1148/radiol.2522081895 (see p. 1).

[3] Sunshine, J. H., Cypel, Y. S., and Schepps, B., "Diagnostic radiologists in 2000: Basic characteristics, practices, and issues related to the radiologist shortage," *American Journal of Roentgenology*, vol. 178, no. 2, pp. 291–301, **2002**. DOI: 10.2214/ajr.178.2.1780291 (see p. 1).

[4] Sunshine, J. H. and Meghea, C., "How could the radiologist shortage have eased?" *American Journal of Roentgenology*, vol. 187, no. 5, pp. 1160–1165, **2006**. DOI: 10.2214/AJR.06.0559 (see p. 1).

[5] Doi, K., "Computer-aided diagnosis in medical imaging: Historical review, current status and future potential," *Computerized Medical Imaging and Graphics*, vol. 31, no. 4, pp. 198–211, **2007**. DOI: 10.1016/j.compmedimag.2007.02.002 (see p. 1).

[6] Ginneken, B. van, Schaefer-Prokop, C. M., and Prokop, M., "Computer-aided diagnosis: How to move from the laboratory to the clinic," *Radiology*, vol. 261, no. 3, pp. 719–732, **2011**. DOI: 10.1148/radiol.11091710 (see p. 1).

[7] Katsumata, A. and Fujita, H., "Progress of computer-aided detection/diagnosis (CAD) in dentistry," *Japanese Dental Science Review*, vol. 50, no. 3, pp. 63–68, **2014**. DOI: 10.1016/j.jdsr.2014.03.002 (see p. 1).

[8] Bashshur, R. L., Krupinski, E. A., Thrall, J. H., *et al.*, "The empirical foundations of teleradiology and related applications: A review of the evidence," *Telemedicine and e-Health*, vol. 22, no. 11, pp. 868–898, **2016**. DOI: 10.1089/tmj.2016.0149 (see p. 1).

[9] Kaye, A. H., Forman, H. P., Kapoor, R., *et al.*, "A survey of radiology practices' use of after-hours radiology services," *Journal of the American College of Radiology*, vol. 5, no. 6, pp. 748–758, **2008**. DOI: 10.1016/j.jacr.2008.01.009 (see p. 1).

[10] Bishop, C. M., *Pattern Recognition and Machine Learning*. Springer, **2006** (see p. 2).

[11] Murphy, K. P., *Machine Learning: A Probabilistic Perspective*. MIT press, **2012** (see p. 2).

[12] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*. MIT Press, **2016** (see pp. 2, 15, 16, 53).

[13] Greenspan, H., Ginneken, B. van, and Summers, R. M., "Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1153–1159, **2016**. DOI: 10.1109/TMI.2016.2553401 (see p. 3).

[14] He, K., Zhang, X., Ren, S., *et al.*, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*, **2016**, pp. 630–645. DOI: 10.1007/978-3-319-46493-0_38. arXiv: 1603.05027 (see pp. 3, 32).

[15] Jégou, S., Drozdzal, M., Vázquez, D., *et al.*, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," **2016**. arXiv: 1611.09326 (see pp. 3, 44).

[16] Maus, T., "Imaging the back pain patient," *Physical Medicine and Rehabilitation Clinics of North America*, vol. 21, no. 4, pp. 725–766, **2010**. DOI: 10.1016/j.pmr.2010.07.004 (see p. 5).

[17] Jenis, L. G. and An, H. S., "Spine update: Lumbar foraminal stenosis," *Spine*, vol. 25, no. 3, pp. 389–394, **2000**. DOI: 10.1097/00007632-200002010-00022 (see p. 7).

[18] Bogduk, N., *Clinical Anatomy Of The Lumbar Spine And Sacrum*. Elsevier Health Sciences, **2005** (see p. 7).

[19] Epstein, N. E., Epstein, J. A., Carras, R., *et al.*, "Far lateral lumbar disc herniations and associated structural abnormalities an evaluation in 60 patients of the comparative value of CT, MRI, and myelo-CT in diagnosis and management," *Spine*, vol. 15, no. 6, pp. 534–539, **1990**. DOI: 10.1097/00007632-199006000-00019 (see p. 7).

[20] Hasegawa, T., Mikawa, Y., Watanabe, R., *et al.*, "Morphometric analysis of the lumbosacral nerve roots and dorsal root ganglia by magnetic resonance imaging.," *Spine*, vol. 21, no. 9, pp. 1005–1009, **1996**. DOI: 10.1097/00007632-199605010-00001 (see p. 7).

[21] Jenis, L. G., An, H. S., and Gordin, R, "Foraminal stenosis of the lumbar spine: A review of 65 surgical cases.," *American Journal of Orthopedics*, vol. 30, no. 3, pp. 205–211, **2001** (see p. 7).

[22] Epstein, J. A., Epstein, B. S., Lavine, L. S., *et al.*, "Lumbar nerve root compression at the intervertebral foramina caused by arthritis of the posterior facets," *Journal of Neurosurgery*, vol. 39, no. 3, pp. 362–369, **1973**. DOI: 10.3171/jns.1973.39.3.0362 (see p. 7).

[23] Takahashi, K., Shima, I., and Porter, R. W., "Nerve root pressure in lumbar disc herniation," *Spine*, vol. 24, no. 19, p. 2003, **1999**. DOI: 10.1097/00007632-199910010-00007 (see p. 7).

[24] Edelson, J. and Nathan, H, "Nerve root compression in spondylolysis and spondylolisthesis," *Bone & Joint Journal*, vol. 68, no. 4, pp. 596–599, **1986**. DOI: 10.1097/00007632-199910010-00007 (see p. 7).

[25] Herkowitz, H. N. and Kurz, L., "Degenerative lumbar spondylolisthesis with spinal stenosis: A prospective study comparing decompression with decompression and intertransverse process arthrodesis," *Journal of Bone and Joint Surgery. American Volume*, vol. 73, no. 6, pp. 802–808, **1991**. DOI: 10.2106/00004623-199173060-00002 (see p. 7).

[26] Ploumis, A., Transfledt, E. E., and Denis, F., "Degenerative lumbar scoliosis associated with spinal stenosis," *The Spine Journal*, vol. 7, no. 4, pp. 428–436, **2007**. DOI: 10.1016/j.spinee.2006.07.015 (see p. 7).

[27] Lee, S., Lee, J. W., Yeom, J. S., *et al.*, "A practical MRI grading system for lumbar foraminal stenosis," *American Journal of Roentgenology*, vol. 194, no. 4, pp. 1095–1098, **2010**. DOI: 10.2214/ajr.09.2772 (see p. 8).

[28] Kunogi, J. and Hasue, M., "Diagnosis and operative treatment of intraforaminal and extraforaminal nerve root compression," *Spine*, vol. 16, no. 11, pp. 1312–1320, **1991**. DOI: 10.1097/00007632-199111000-00012 (see p. 8).

[29] Wildermuth, S., Zanetti, M., Duewell, S., *et al.*, "Lumbar spine: Quantitative and qualitative assessment of positional (upright flexion and extension) MR imaging and myelography," *Radiology*, vol. 207, no. 2, pp. 391–398, **1998**. DOI: 10.1148/radiology.207.2.9577486 (see p. 8).

[30] Park, H.-J., Kim, S., Lee, S.-Y., *et al.*, "Clinical correlation of a new MR imaging method for assessing lumbar foraminal stenosis," *American Journal of Neuroradiology*, vol. 33, no. 5, pp. 818–822, **2012**. DOI: 10.3174/ajnr.a2870 (see p. 9).

[31] Dauphin, Y. N., Pascanu, R., Gülçehre, Ç., *et al.*, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in Neural Information Processing Systems 27*, **2014**, pp. 2933–2941. arXiv: 1406.2572 (see p. 13).

[32] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, **1986**. DOI: 10.1038/323533a0 (see p. 13).

[33] Bengio, Y., Simard, P. Y., and Frasconi, P., "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, **1994**. DOI: 10.1109/72.279181 (see p. 15).

[34] Hinton, G. E., Osindero, S., and Teh, Y. W., "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, **2006**. DOI: 10.1162/neco.2006.18.7.1527 (see p. 15).

[35] Bengio, Y., Lamblin, P., Popovici, D., *et al.*, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems 19*, **2006**, pp. 153–160 (see p. 15).

[36] Glorot, X. and Bengio, Y., "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, **2010**, pp. 249–256 (see pp. 15, 16).

[37] He, K., Zhang, X., Ren, S., *et al.*, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *IEEE International Conference on Computer Vision*, **2015**, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123. arXiv: 1502.01852 (see pp. 16, 18).

[38] Glorot, X., Bordes, A., and Bengio, Y., "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, vol. 15, **2011**, p. 275. DOI: 10.1109/iwaenc.2016.7602891 (see p. 18).

[39] Krizhevsky, A., Sutskever, I., and Hinton, G. E., "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, **2012**, pp. 1106–1114 (see pp. 18, 25).

[40] Maas, A. L., Hannun, A. Y., and Ng, A. Y., "Rectifier nonlinearities improve neural network acoustic models," in *International Conference on Machine Learning*, vol. 30, **2013** (see p. 18).

[41] Clevert, D., Unterthiner, T., and Hochreiter, S., "Fast and accurate deep network learning by exponential linear units," **2015**. arXiv: 1511.07289 (see p. 18).

[42] Shang, W., Sohn, K., Almeida, D., *et al.*, "Understanding and improving convolutional neural networks via concatenated rectified linear units," in *International Conference on Machine Learning*, **2016**, pp. 2217–2225. arXiv: 1603.05201 (see p. 19).

[43] Ioffe, S. and Szegedy, C., "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, vol. 37, **2015**, pp. 448–456. arXiv: 1502.03167 (see pp. 19, 44).

[44] Qian, N., "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, **1999**. DOI: 10.1016/S0893-6080(98)00116-6 (see p. 21).

[45] Nesterov, Y., "A method for unconstrained convex minimization problem with the rate of convergence O(1/k2)," in *Proceedings of the USSR Academy of Sciences*, vol. 269, **1983**, pp. 543–547 (see p. 21).

[46] Duchi, J. C., Hazan, E., and Singer, Y., "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, **2011** (see p. 21).

[47] Dean, J., Corrado, G., Monga, R., *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25*, Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C., *et al.*, Eds., **2012**, pp. 1232–1240 (see p. 22).

[48] Tieleman, T. and Hinton, G., "Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude," *Coursera: Neural networks for machine learning*, vol. 4, no. 2, **2012** (see p. 22).

[49] Kingma, D. P. and Ba, J., "Adam: A method for stochastic optimization," **2014**. arXiv: 1412.6980 (see p. 22).

[50] Srivastava, N., Hinton, G. E., Krizhevsky, A., *et al.*, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, **2014** (see p. 23).

[51] Hubel, D. H. and Wiesel, T. N., "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, pp. 574–591, **1959**. DOI: 10.1113/jphysiol.1959.sp006308 (see p. 25).

[52] ——, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, **1962**. DOI: 10.1113/jphysiol.1962.sp006837 (see p. 25).

[53] ——, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, **1968**. DOI: 10.1113/jphysiol.1968.sp008455 (see p. 25).

[54] Fukushima, K. and Miyake, S., "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*, Springer, **1982**, pp. 267–285. DOI: 10.1007/978-3-642-46466-9_18 (see p. 25).

[55] LeCun, Y., Boser, B. E., Denker, J. S., *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, **1989**. DOI: 10.1162/neco.1989.1.4.541 (see p. 25).

[56] Springenberg, J. T., Dosovitskiy, A., Brox, T., *et al.*, "Striving for simplicity: The all convolutional net," **2014**. arXiv: 1412.6806 (see p. 29).

[57] Simonyan, K. and Zisserman, A., "Very deep convolutional networks for large-scale image recognition," **2014**. arXiv: 1409.1556 (see p. 30).

[58] Lin, M., Chen, Q., and Yan, S., "Network in network," **2013**. arXiv: 1312.4400 (see p. 31).

[59] Szegedy, C., Liu, W., Jia, Y., *et al.*, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, **2015**, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594. arXiv: 1409.4842 (see pp. 31, 45).

[60] He, K., Zhang, X., Ren, S., *et al.*, "Deep residual learning for image recognition," in *Conference on Computer Vision and Pattern Recognition*, **2016**, pp. 770–778. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385 (see pp. 32, 44, 45).

[61] Noh, H., Hong, S., and Han, B., "Learning deconvolution network for semantic segmentation," in *IEEE International Conference on Computer Vision*, IEEE Computer Society, **2015**, pp. 1520–1528. DOI: 10.1109/ICCV.2015.178. arXiv: 1505.04366 (see pp. 33, 35, 36, 44).

[62] Zeiler, M. D., Krishnan, D., Taylor, G. W., *et al.*, "Deconvolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, **2010**, pp. 2528–2535. DOI: 10.1109/CVPR.2010.5539957 (see p. 33).

[63] Shelhamer, E., Long, J., and Darrell, T., "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640–651, **2017**. DOI: 10.1109/TPAMI.2016.2572683. arXiv: 1605.06211 (see pp. 34, 44).

[64] Tai, L., Ye, Q., and Liu, M., "PCA-aided fully convolutional networks for semantic segmentation of multi-channel fMRI," **2016**. arXiv: 1610.01732 (see p. 35).

[65] Badrinarayanan, V., Handa, A., and Cipolla, R., "SegNet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling," **2015**. arXiv: 1505.07293 (see p. 35).

[66] Ren, S., He, K., Girshick, R. B., *et al.*, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems 28*, **2015**, pp. 91–99. arXiv: 1506.01497 (see p. 38).

[67] NEMA, "Digital imaging and communications in medicine (DICOM)," National Electrical Manufacturers Association, Standard, PS3 / ISO 12052. (see p. 38).

[68] Litjens, G. J. S., Kooi, T., Bejnordi, B. E., *et al.*, "A survey on deep learning in medical image analysis," **2017**. arXiv: 1702.05747 (see p. 43).

[69] Ronneberger, O., Fischer, P., and Brox, T., "U-Net: convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer Assisted Intervention*, vol. 9351, **2015**, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28. arXiv: 1505.04597 (see pp. 44, 53).

[70] Drozdzal, M., Vorontsov, E., Chartrand, G., *et al.*, "The importance of skip connections in biomedical image segmentation," in *Deep Learning and Data Labeling for Medical Applications*, vol. 10008, **2016**, pp. 179–187. DOI: 10.1007/978-3-319-46976-8_19. arXiv: 1608.04117 (see p. 44).

[71] Milletari, F., Navab, N., and Ahmadi, S., "V-Net: fully convolutional neural networks for volumetric medical image segmentation," in *Fourth International Conference on 3D Vision*, IEEE Computer Society, **2016**, pp. 565–571. DOI: 10.1109/3DV.2016.79. arXiv: 1606.04797 (see pp. 44, 45, 55).

[72] Çiçek, Ö., Abdulkadir, A., Lienkamp, S. S., *et al.*, "3D U-Net: learning dense volumetric segmentation from sparse annotation," in *Medical Image Computing and Computer-Assisted Intervention*, vol. 9901, **2016**, pp. 424–432. DOI: 10.1007/978-3-319-46723-8_49. arXiv: 1606.06650 (see p. 44).

[73] Kayalibay, B., Jensen, G., and Smagt, P. van der, "CNN-based segmentation of medical imaging data," **2017**. arXiv: 1701.03056 (see pp. 44, 46, 51, 65).

[74] Lieber, D., Rubinstein, R. Y., and Elmakis, D, "Quick estimation of rare events in stochastic networks," *IEEE Transactions on Reliability*, vol. 46, no. 2, pp. 254–265, **1997**. DOI: 10.1109/24.589954 (see p. 44).

[75] Dice, L. R., “Measures of the amount of ecologic association between species,” *Ecology*, vol. 26, no. 3, pp. 297–302, **1945**. DOI: 10.2307/1932409 (see pp. 45, 50).

[76] Chen, H., Dou, Q., Yu, L., *et al.*, “VoxResNet: deep voxelwise residual networks for volumetric brain segmentation,” **2016**. arXiv: 1608.05895 (see pp. 45, 46).

[77] Dou, Q., Chen, H., Jin, Y., *et al.*, “3D deeply supervised network for automatic liver segmentation from CT volumes,” in *Medical Image Computing and Computer-Assisted Intervention*, vol. 9901, **2016**, pp. 149–157. DOI: 10.1007/978-3-319-46723-8_18. arXiv: 1607.00582 (see p. 45).

[78] Yu, L., Yang, X., Chen, H., *et al.*, “Volumetric ConvNets with mixed residual connections for automated prostate segmentation from 3D MR images,” in *AAAI Conference on Artificial Intelligence*, **2017**, pp. 66–72 (see p. 46).

[79] Wang, C. and Forsberg, D., “Segmentation of intervertebral discs in 3d MRI data using multi-atlas based registration,” in *Computational Methods and Clinical Applications for Spine Imaging*, vol. 9402, **2015**, pp. 107–116. DOI: 10.1007/978-3-319-41827-8_10 (see p. 46).

[80] Hutt, H., Everson, R., and Meakin, J., “3D intervertebral disc segmentation from MRI using supervoxel-based CRFs,” in *Computational Methods and Clinical Applications for Spine Imaging*, vol. 9402, **2015**, pp. 125–129. DOI: 10.1007/978-3-319-41827-8_12 (see p. 46).

[81] Urschler, M., Hammernik, K., Ebner, T., *et al.*, “Automatic intervertebral disc localization and segmentation in 3D MR images based on regression forests and active contours,” in *Computational Methods and Clinical Applications for Spine Imaging*, vol. 9402, **2015**, pp. 130–140. DOI: 10.1007/978-3-319-41827-8_13 (see p. 46).

[82] Chen, C., Belavy, D., Yu, W., *et al.*, “Localization and segmentation of 3D intervertebral discs in MR images by data driven estimation,” *IEEE Transactions On Medical Imaging*, vol. 34, no. 8, pp. 1719–1729, **2015**. DOI: 10.1109/TMI.2015.2403285 (see p. 47).

[83] Chen, H., Shen, C., Qin, J., *et al.*, “Automatic localization and identification of vertebrae in spine CT via a joint learning model with deep neural networks,” in *Medical Image Computing and Computer-Assisted Intervention*, vol. 9349, **2015**, pp. 515–522. DOI: 10.1007/978-3-319-24553-9_63 (see p. 47).

[84] Yang, D., Xiong, T., Xu, D., *et al.*, "Automatic vertebra labeling in large-scale 3D CT using deep image-to-image network with message passing and sparsity regularization," **2017**. arXiv: 1705.05998 (see p. 47).

[85] Forsberg, D., Sjöblom, E., and Sunshine, J. L., "Detection and labeling of vertebrae in MR images using deep learning with clinical annotations as training data," *Journal of Digital Imaging*, pp. 1–7, **2017**. DOI: 10.1007/s10278-017-9945-x (see p. 47).

[86] Walt, S. van der, Colbert, S. C., and Varoquaux, G., "The NumPy array: A structure for efficient numerical computation," *Computing in Science and Engineering*, vol. 13, no. 2, pp. 22–30, **2011**. DOI: 10.1109/MCSE.2011.37. arXiv: 1102.1523 (see p. 49).

[87] Jones, E., Oliphant, T., Peterson, P., *et al.*, *SciPy: Open source scientific tools for Python*, **2001** (see p. 49).

[88] McKinney, W. *et al.*, "Data structures for statistical computing in Python," in *Proceedings of the 9th Python in Science Conference*, van der Voort S, Millman J, vol. 445, **2010**, pp. 51–56 (see p. 49).

[89] Abadi, M., Agarwal, A., Barham, P., *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," **2016**. arXiv: 1603.04467 (see p. 49).

[90] Jaccard, P, "Nouvelles recherches sur la distribution florale," **1908** (see p. 51).

[91] McNemar, Q., "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, pp. 153–157, **1947** (see p. 52).

[92] Bergstra, J. and Bengio, Y., "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, **2012** (see p. 55).

[93] Lloyd, S., "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, **1982**. DOI: 10.1109/tit.1982.1056489 (see pp. 60, 67).

A

HYPERPARAMETER SEARCH

NAME	POSSIBLE VALUES	DESCRIPTION
`PatchSize`	$\{33, 65, 97, 129, 161, 193, 225, 257, 289, 321, 353, 385, 417, 449, 481\}$	The size of the patch extracted from a scan.
`BatchSize`	$\{2, \ldots, 6\}$	Number of patches in a batch.
`Block`	$\{2, \ldots, 5\}$	How many times BaseNet compresses and expands.
`Channels`	$\{2, \ldots, 16\}$	The number of channels at the highest level.
`LearningRate`	$\{0.01, 0.004, 0.001, 0.0004\}$	The initial learning rate.
`KeepProbability`	$\{0.4, 0.45, \ldots, 1\}$	The probability a neuron is not dropped.

Table A.1: The hyperparameter ranges search for the base version of SingleNet.

BATCH SIZE	BLOCK	CHANNELS	KEEP PROBABILITY	LEARNING RATE	PATCH SIZE	DICE COEFFICIENT
2	2	16	0.95	0.001	385	72.75
5	4	13	0.75	0.0004	321	71.48
3	2	12	0.85	0.001	353	68.98
4	5	8	0.9	0.004	289	67.75
3	5	13	0.9	0.004	385	67.61
3	3	6	0.9	0.0004	449	67.57
6	3	12	0.75	0.0004	193	64.34
3	3	2	0.85	0.0004	289	63.14
5	2	8	0.8	0.01	449	62.63
4	4	11	0.5	0.01	417	60.36
6	5	5	0.95	0.0004	385	59.54
6	2	13	0.75	0.001	225	59.35
3	5	8	0.95	0.0004	417	59.22
3	4	2	0.85	0.004	417	59.18
6	2	8	0.45	0.0004	321	58.13
4	2	16	0.5	0.0004	321	57.76
5	3	5	0.95	0.001	417	57.62
6	2	4	0.6	0.0004	481	57.49
2	5	4	0.75	0.001	353	57.42
4	3	15	1.0	0.0004	353	56.77

Figure A.1: Top 20 hyperparameter search results.

B

SINGLENET

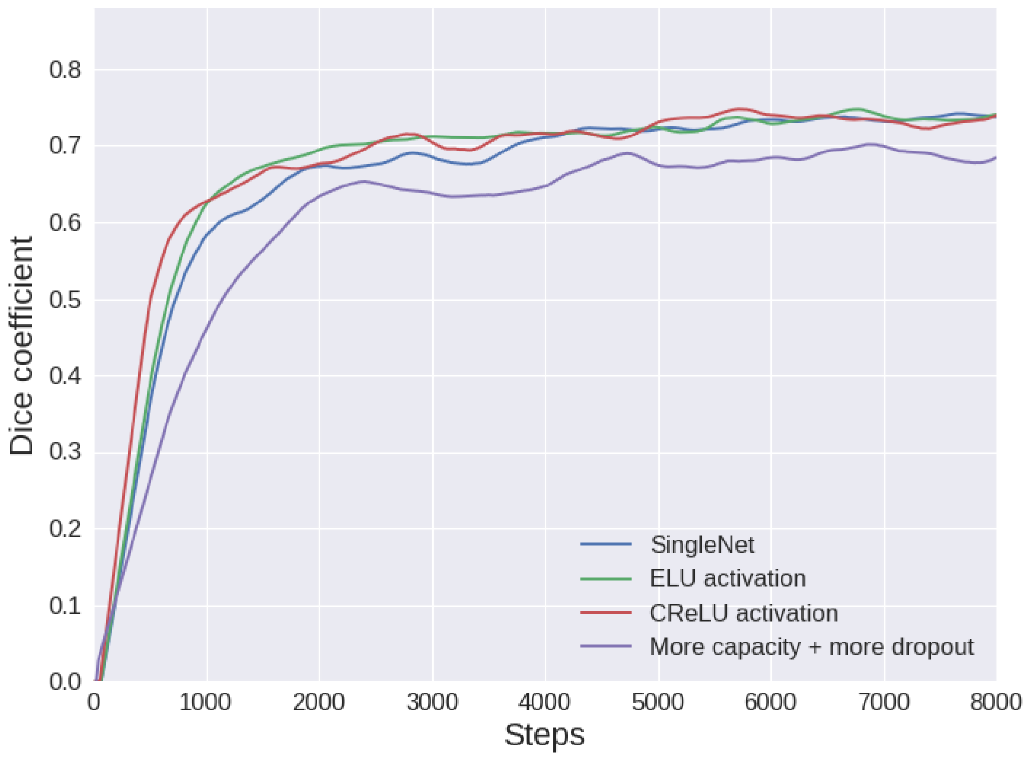


Figure B.1: The test Dice coefficients of the SingleNet 'ELU', 'CReLU', and 'more capacity' variations while training the network.

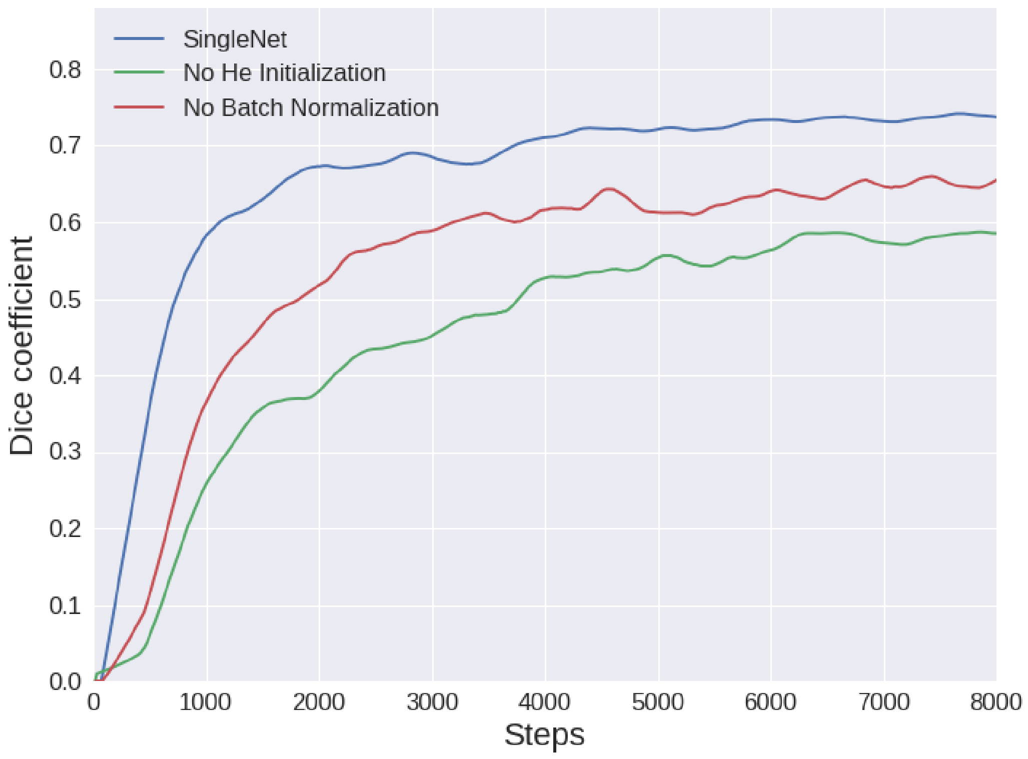


Figure B.2: The test Dice coefficients of the SingleNet 'no He Initialization' and 'no batch normalization' variations while training the network.

NETWORK	MISSED FORAMINA					
	ALL	L1	L2	L3	L4	L5
Base	3	3	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
CReLU	10	9	1	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
ELU	1	1	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
More capacity	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
No He Initialization	11	6	3	2	$\boxed{0}$	$\boxed{0}$
No Batch Normalization	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
Trainings Set Size 250	10	10	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
Trainings Set Size 150	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
Trainings Set Size 50	3	2	1	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
Trainings Set Size 5	28	20	6	2	$\boxed{0}$	$\boxed{0}$
Trainings Set Size 1	83	33	25	15	8	2

Table B.1: The number of foramina that the variants of SingleNet made no predication for.

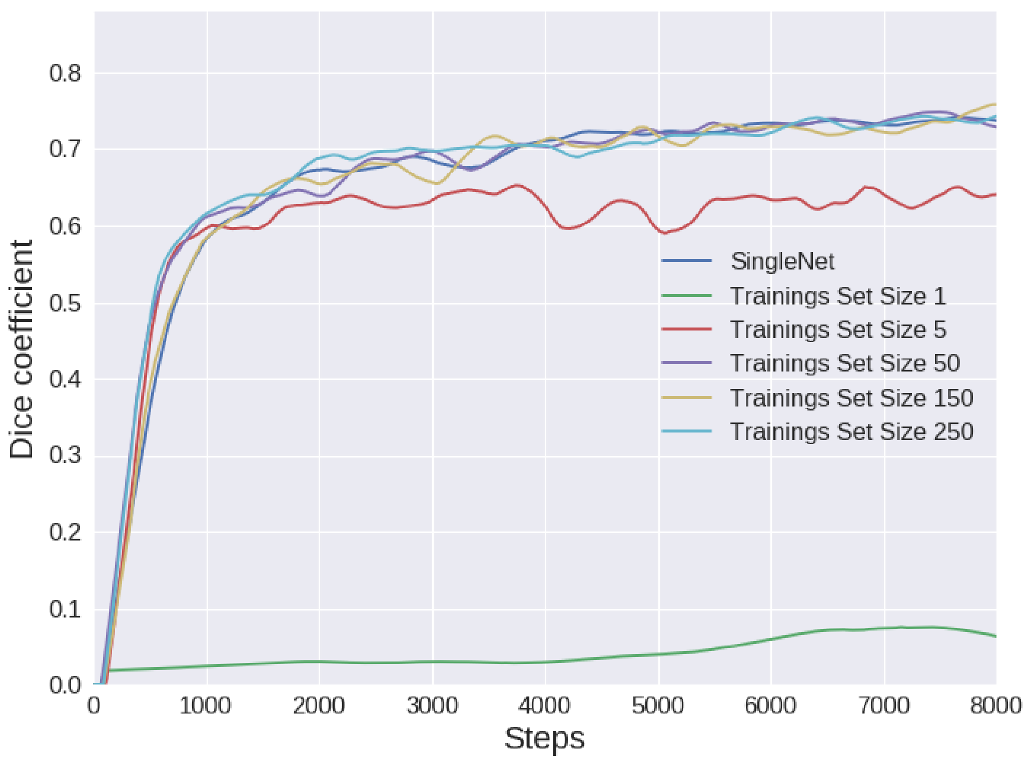


Figure B.3: The test Dice coefficients of SingleNet with various number of trainings samples while training the network.

MODEL A	MODEL B	P-VALUE
Trainings Set Size 150	Trainings Set Size 250	0.94
Trainings Set Size 150	Base	0.44
Trainings Set Size 250	Base	0.28
No He Initialization	Trainings Set Size 5	0.20
Trainings Set Size 50	Trainings Set Size 150	0.14
Trainings Set Size 50	Trainings Set Size 250	0.14
ELU	CReLU	0.10
Trainings Set Size 50	More capacity	0.05
Base	CReLU	0.04
Trainings Set Size 50	Base	0.01
Trainings Set Size 150	CReLU	0.01
	⋮	

Table B.2: The p-value for the SingleNet variation pairs. Only non-zero values are shown.

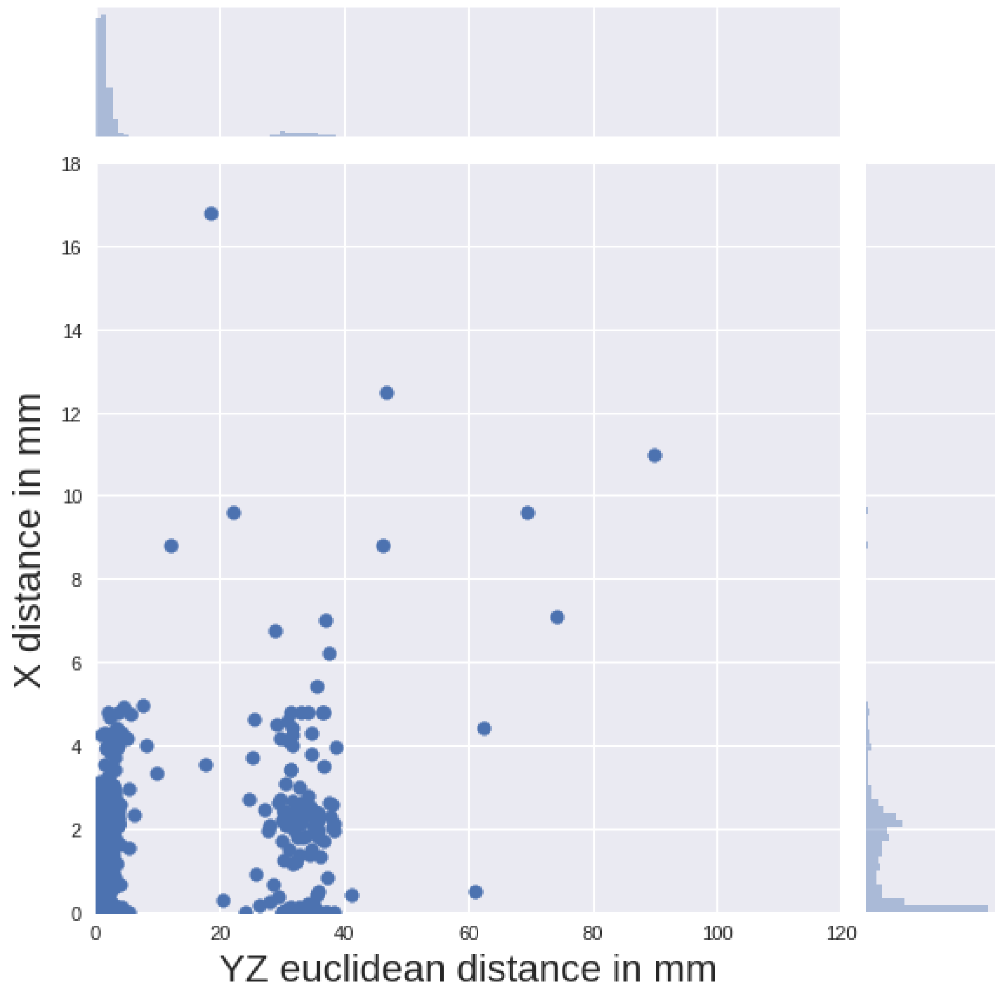
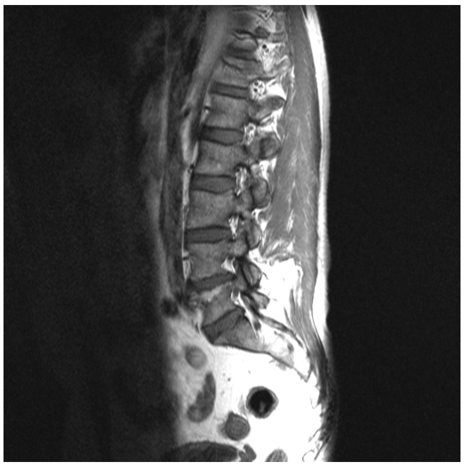
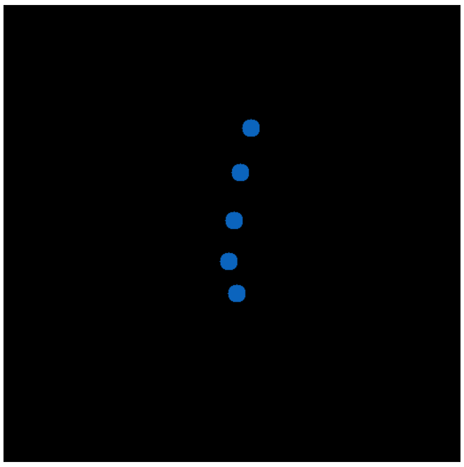
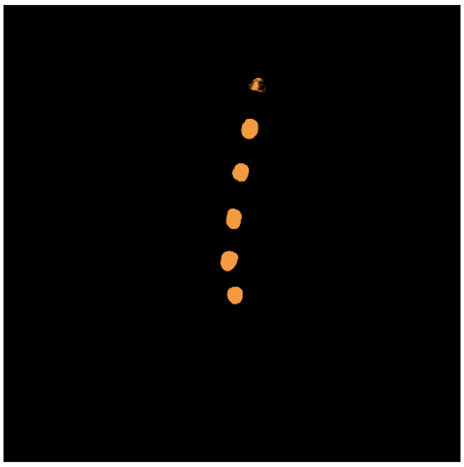


Figure B.4: The distance from the predicted foramina centers to the real centers in the x and yz axises by the base variant of SINGLENET.


(a) Input


(b) Target


(c) Output

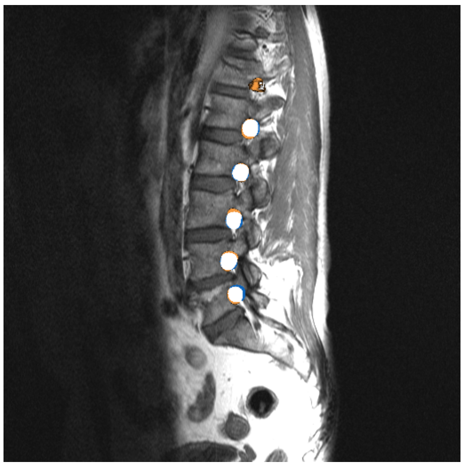

(d) Combination

Figure B.5: A complete slice correctly predicted by SingleNet.

C MULTINET

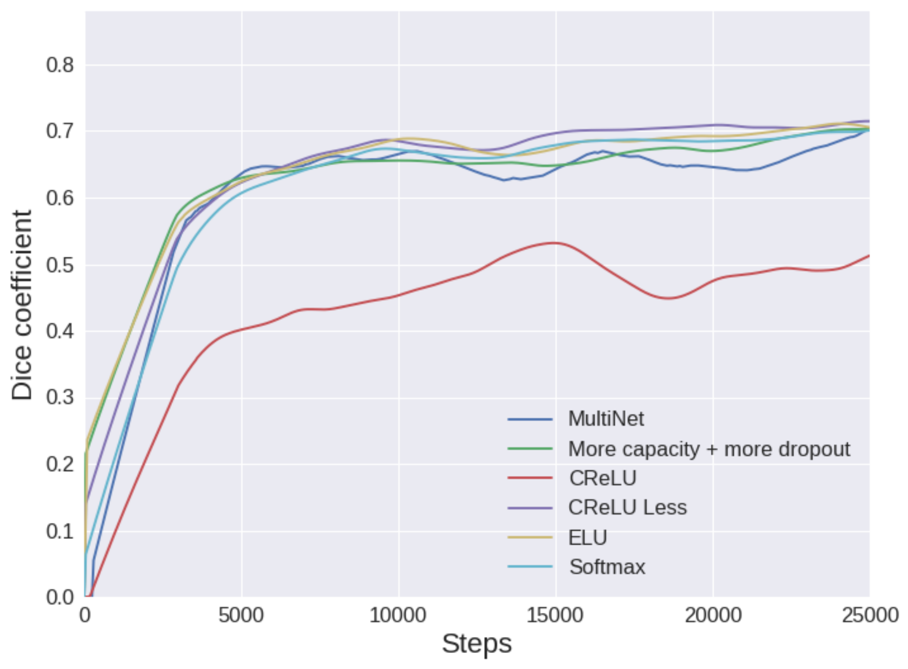


Figure C.1: The test Dice coefficients of the MultiNet 'CReLU', 'CReLU Less', 'ELU', and 'more capacity' variations while training the network.

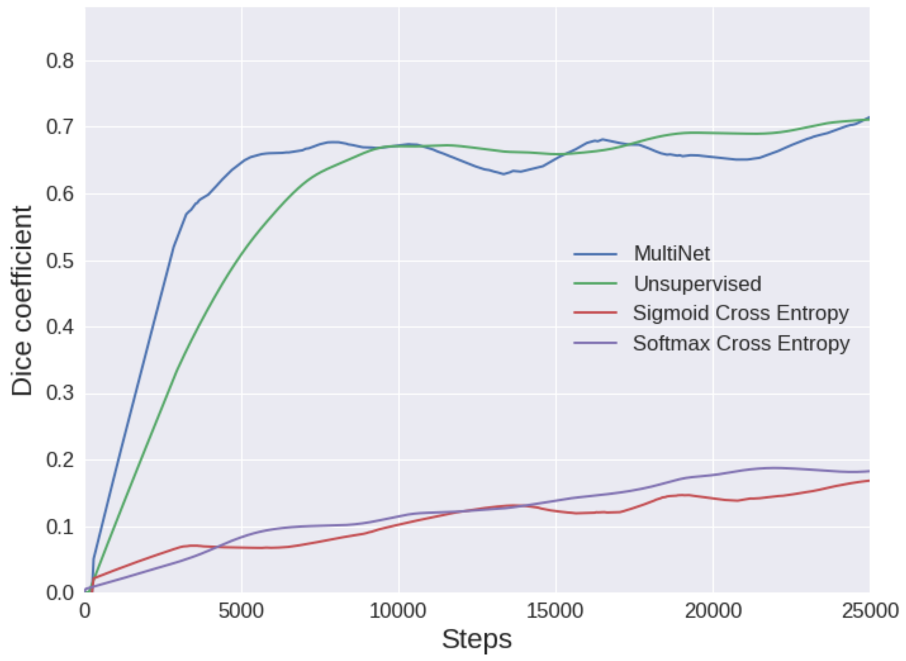


Figure C.2: The test Dice coefficients of the MultiNet 'softmax', 'sigmoid cross entropy', and 'softmax cross entropy' variations while training the network.

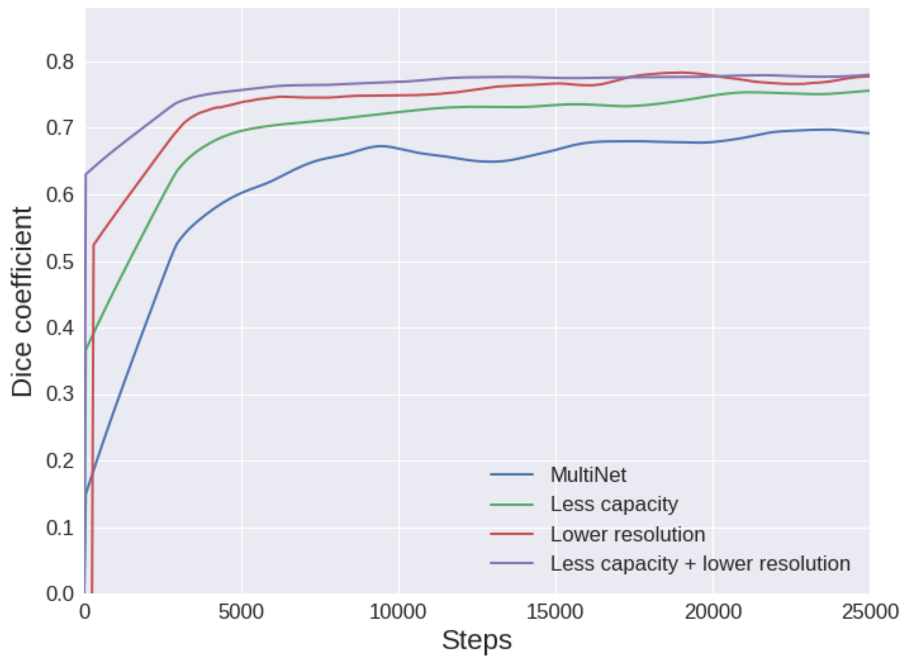


Figure C.3: The test Dice coefficients of the MultiNet 'less capacity', 'lower resolution', and 'less capacity + lower resolution' variations while training the network.

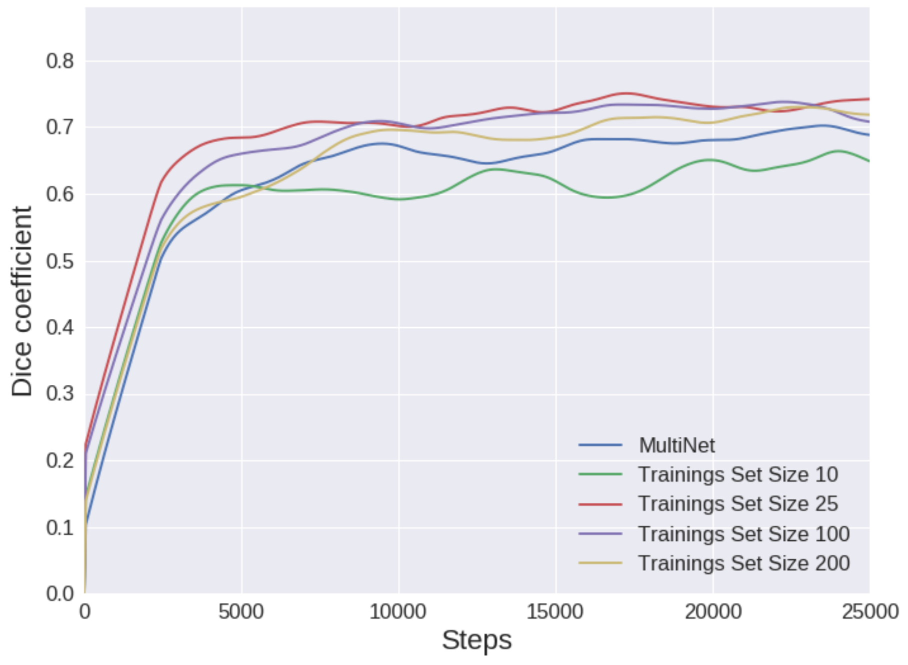


Figure C.4: The test Dice coefficients of MultiNet with various number of trainings samples while training the network.

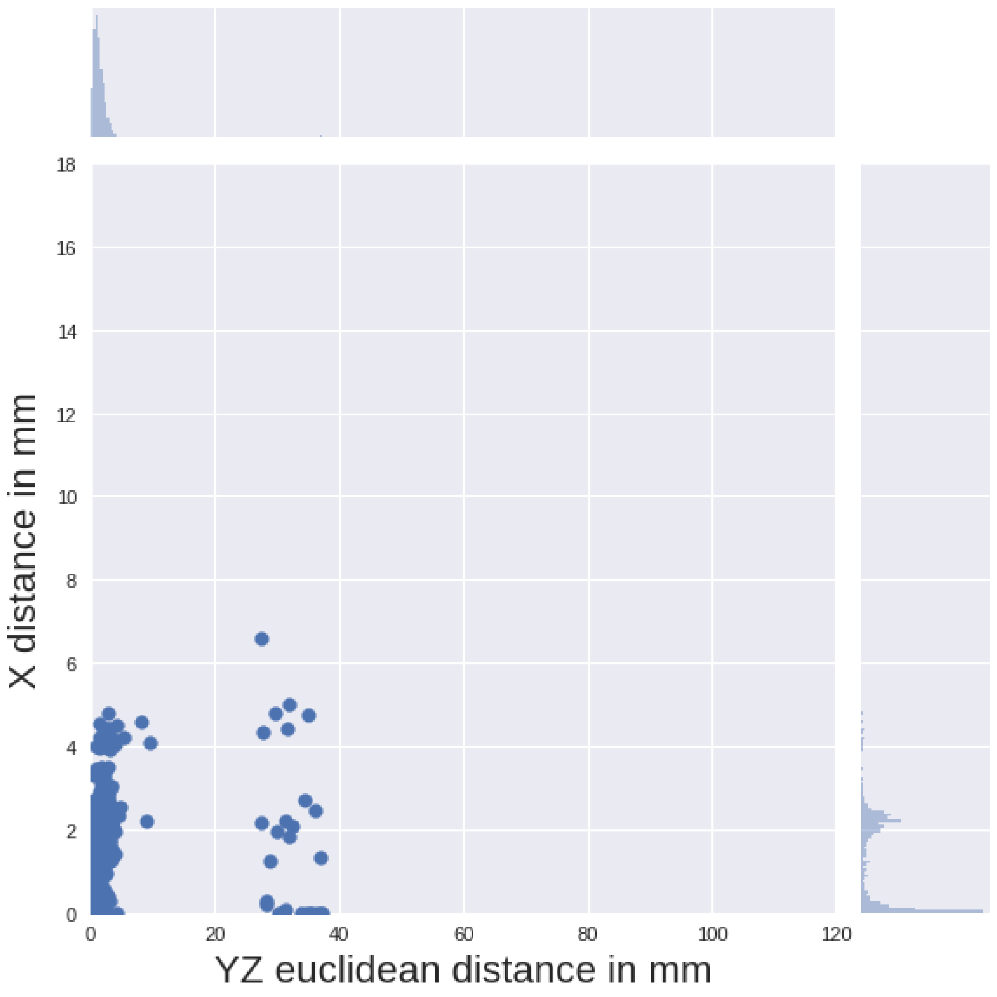


Figure C.5: The distance from the predicted foramina centers to the real centers in the x and yz axises by the (normalized) base variant of MultiNet.

NETWORK	MISSED FORAMINA					
	ALL	L1	L2	L3	L4	L5
Base	600	200	200	200	$\boxed{0}$	$\boxed{0}$
Normalized	5	2	1	1	$\boxed{0}$	1
Unsupervised	7	6	1	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
ELU	4	$\boxed{0}$	3	1	$\boxed{0}$	$\boxed{0}$
CReLU	219	5	38	51	57	68
CReLU Less	4	2	1	1	$\boxed{0}$	$\boxed{0}$
More capacity	4	1	1	1	$\boxed{0}$	1
Softmax	6	3	1	1	$\boxed{0}$	1
Sigmoid Cross Entropy	90	$\boxed{0}$	$\boxed{0}$	18	43	29
Softmax Cross Entropy	61	$\boxed{0}$	$\boxed{0}$	6	49	6
Lower res.	1	1	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
Lower res. 60k	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
Lower res. 120k	2	1	1	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
Less capacity	5	4	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	1
Less capacity + lower res.	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
Training Set Size 10	44	32	7	3	1	1
Training Set Size 25	12	8	2	1	$\boxed{0}$	1
Training Set Size 50	9	6	$\boxed{0}$	1	$\boxed{0}$	2
Training Set Size 100	12	6	4	2	$\boxed{0}$	$\boxed{0}$
Training Set Size 200	1	$\boxed{0}$	1	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$

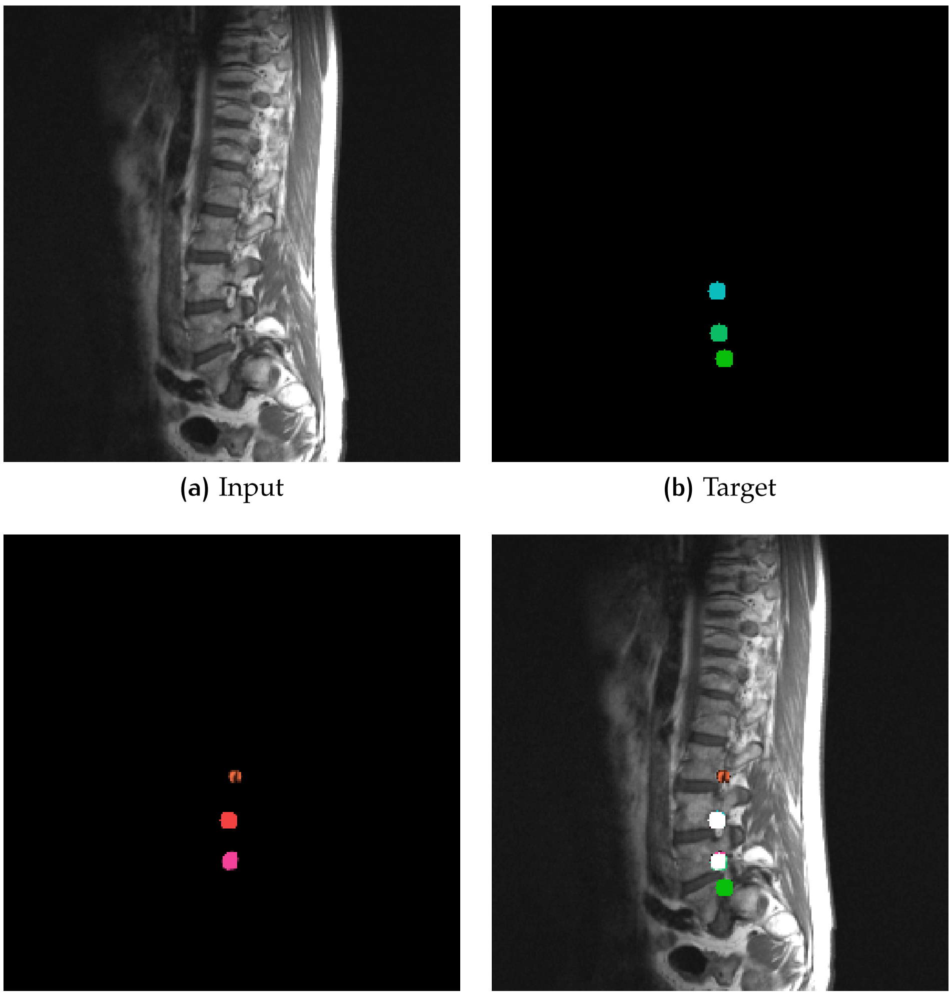
Table C.1: The number of foramina that the variants of MultiNet made no predication for.

MODEL A	MODEL B	P-VALUE
Lower res. 120k	Less capacity + lower res.	1.00
CReLU Less	Less capacity	1.00
Unsupervised	Softmax	1.00
ELU	CReLU Less	1.00
More capacity	Less capacity	1.00
ELU	Less capacity	0.85
More capacity	Lower res. 120k	0.82
Lower res.	Lower res. 60k	0.82
CReLU Less	More capacity	0.81
Training Set Size 25	Training Set Size 200	0.80
Training Set Size 50	Training Set Size 200	0.78
Lower res. 120k	Less capacity	0.71
More capacity	Less capacity + lower res.	0.70
ELU	More capacity	0.65
Softmax	Training Set Size 25	0.60
Less capacity	Less capacity + lower res.	0.58
CReLU Less	Lower res. 120k	0.52
Unsupervised	Training Set Size 25	0.50
Training Set Size 25	Training Set Size 50	0.49
ELU	Lower res. 120k	0.44
CReLU Less	Less capacity + lower res.	0.40
Training Set Size 50	Training Set Size 100	0.39
Normalized	Unsupervised	0.36
ELU	Less capacity + lower res.	0.36
Softmax	Training Set Size 200	0.33
Unsupervised	Training Set Size 200	0.31

Table C.2: The p-value for the MultiNet variation pairs. Only non-zero values are shown. First half.

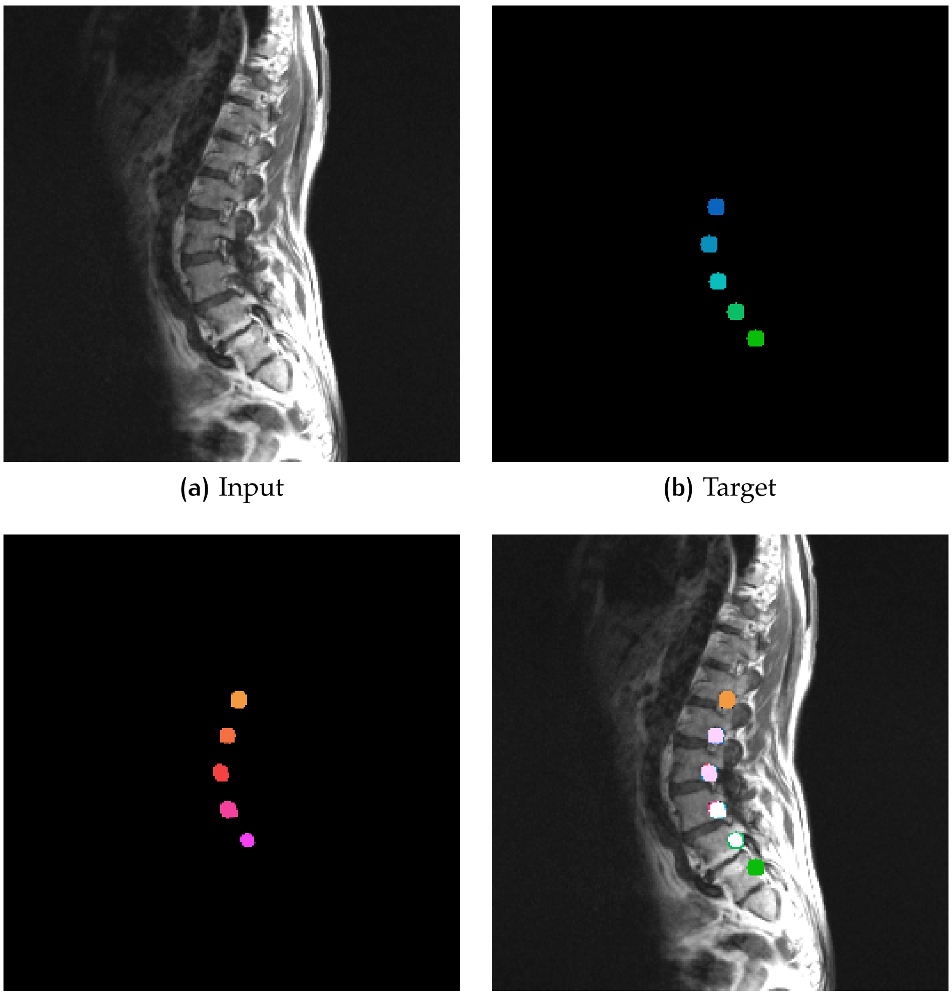
MODEL A	MODEL B	P-VALUE
Normalized	Softmax	0.31
Training Set Size 100	Training Set Size 200	0.23
Unsupervised	Training Set Size 50	0.14
Softmax	Training Set Size 50	0.14
Training Set Size 25	Training Set Size 100	0.12
Lower res.	Less capacity + lower res.	0.09
Normalized	Training Set Size 25	0.07
Normalized	ELU	0.07
CReLU	Softmax Cross Entropy	0.06
Lower res.	Lower res. 120k	0.05
Normalized	Training Set Size 200	0.04
Normalized	CReLU Less	0.04
Softmax	Training Set Size 100	0.02
Normalized	Less capacity	0.02
ELU	Lower res.	0.01
Unsupervised	ELU	0.01
Normalized	Training Set Size 50	0.01
More capacity	Lower res.	0.01
CReLU Less	Softmax	0.01
CReLU Less	Lower res.	0.01
Normalized	Less capacity + lower res.	0.01
Lower res.	Less capacity	0.01
ELU	Softmax	0.01
Unsupervised	Training Set Size 100	0.01
Normalized	Lower res. 120k	0.01
Normalized	More capacity	0.01
	⋮	

Table C.3: The p-value for the MultiNet variation pairs. Only non-zero values are shown. Second half.


(a) Input (b) Target

(c) Output (d) Combination

Figure C.6: An example of a missed L5 foramen by MultiNet due to an unrecognizable foramen because of heavy spondylosis.



(a) Input

(b) Target

(c) Output

(d) Combination

Figure C.7: An example of prediction made by MultiNet that is a shift upwards w.r.t. the target. The L4 disc is degenerated.