Utrecht University

Artificial Intelligence

Master Thesis

On the classification of imbalanced image datasets

A machine learning approach towards classifying the imbalanced COCO image dataset

Author:
Olivier Claessen

Supervisors:
dr. Cassio de Campos
dr. Ad Feelders
drs. Wilbert Weterings

August 1, 2019

Abstract

In certain complex real-world problems such as fraud detection and disaster prediction, some instances of classes are more rare than other instances of classes in the dataset making the dataset imbalanced. When working in such important domains, some classes might have very few instances while still being very important for the classification task. An intuitive example of a minority class which is important to learn and predict in an imbalanced dataset is an instance of a fraud case when trying to detect fraud. As fraudulent transactions are not as common as non-fraudulent transactions, these instances might be hard to learn as there is less data to train the classifiers on while this class is still the most important class to predict in the dataset. In order to solve these problems, the problem of imbalanced datasets has to be addressed. The goal of this thesis is to construct a classification model that can predict classes of image data in an imbalanced dataset. The dataset that is used for this research is the Common Objects in Context (COCO) dataset by Microsoft, this dataset contains 80 classes with occluded and cluttered images of these instances and is quite imbalanced. The goal is to research what kind of classifiers perform well on such types of imbalanced dataset, by using a convolutional Neural Network (CNN), random forest (RF) and support vector machine (SVM). The RF and SVM classifiers use pre-extracted features such as histogram of oriented gradients (HOG) and DAISY. Out of the three classifiers that were used, the neural network generally performed better than the RF and SVM. This is probably due to these predictors needing pre-extracted features which makes them less flexible in extracting meaningful features from images. The Neural network performance F1 micro (0.425) outperformed the baseline dummy classifier F1 micro (0.306) while only using 10% of the entire COCO dataset. This research shows that different approaches have the potential to construct models which have the potential to be used for multi-class classification tasks on imbalanced datasets. There are implications that the techniques which were used during this research can be finetuned and optimized even further which in turn leads to better results.

Contents

List of Figures

List of Tables

1 Introduction

In this modern age where data is more abundant than ever, there is a huge need to gain knowledge and insights from data. People generally share a huge amount of personal data on social media platforms such as Facebook, Instagram, LinkedIn and Flickr (Manovich, 2011; Naaman, 2012). On platforms such as Instagram and Flickr where every piece of content is visually inclined, there is an abundant amount of data available. These days social media marketers still struggle to sift through the gigabytes of personal data in order to find useful and meaningful content. One popular technique to infer knowledge from data is machine learning (Bose and Mahapatra, 2001; Fayyad, Piatetsky-Shapiro, and Smyth, 1996). Data from these photos can be used to infer knowledge by using machine learning techniques. Because of this recent trend and the ever-growing dataset from the world wide web, the question arises whether it is possible to train classifiers on imbalanced datasets while still getting good results. More research is needed to fully explore the scope in which sampling can utilized for machine learning to use imbalanced datasets and quickly train classifiers for prototyping or business cases while maintaining good results.

As image data sets are quite large these days, it is quite interesting to see what impact different data sampling methods have on the results of classifiers. This immediately raises the question whether it is possible to improve a classifier on a set of data while using sampling techniques and achieve good results on the test set. If achieving good results with a sample of the data is possible, what are the differences in results when comparing different sampling methods. Some datasets also tend to be imbalanced which might pose a challenge for the data scientist trying to engineer a model to solve the problem (Maloof, 2003). Domains where unbalanced datasets occur naturally are fraud detection and NLP (Lemaître, Nogueira, and Aridas, 2017; S. Kotsiantis, Kanellopoulos, Panayiotis Pintelas, et al., 2006). More research is needed in the domain of imbalanced multi-class datasets and effective sampling techniques.

The models generated by these machine learning techniques can be used in the industry to get an insight into the data that has been collected. Machine learning can be an effective tool for the domain of marketing especially when combined with the descriptive power of images. There are two main domains of marketing: mass marketing and direct marketing. In mass marketing the newspaper and television are used to reach a large audience and in direct marketing the marketing is tailored to groups that might already be interested in buying the product yielding higher response rates. By using machine learning, direct marketing can be tailored exactly to the consumers' needs in order to save money for the company and yield a higher response rate. Machine learning can also be incorporated in companies to find possible future customers or to analyze which customers might be interested in their new product (Ling and Li, 1998; Witten et al., 2016).

1.1 Problem statement

Training a classifier and trying to make it learn the classes present in the data set is much harder when the data set is imbalanced. This means that at least one of the classes present in the data set is significantly larger than other classes. The problem is that the classes with few instances have a low error cost and a low prior probability. Machine learning algorithms have different learning strategies when training on imbalanced data sets. Some algorithms might try to balance the true-positive and false-positive rates while others might simply predict he majority class most of the time. As imbalanced

data sets occur often in real life applications, this problem is interesting to study (Maloof, 2003; Lemaître, Nogueira, and Aridas, 2017). This raises the question: What are good sampling algorithms for a machine learning algorithm?

Sometimes, the dataset is larger than the classification algorithms can deal with on the hardware where it is running on (Provost, 2000). In this case sampling can be a good technique to see which models perform best on a part of the data set and later train it on the dataset. Applications such as market segmentation, new opportunity detection, risk management and software cost prediction are a big part of applications for companies where much data is available (Bose and Mahapatra, 2001).

A standard single label classification has a domain χ that must be classified. Y is the set of labels and H is the set of classifiers so that $\chi \rightarrow Y$. The objective in a classification task is to find the classifier $h \in H$ that maximizes the amount of times the correct class is predicted, h(x) = y where $y \in Y$ is the true label of the instance that is classified (Boutell et al., 2004). Single-label classification tries to classify cases with one label l from a set of disjoint labels L where $|L| > 1$ and if $|L| = 2$ the problem is called a binary classification problem. Even though single label classification has quite some success, there are still problems which do not suit the framework of single label classification (Zhou and M.-L. Zhang, 2007). The single label classification problem is quite limited because images can belong to multiple classes at the same time, if $|L| > 2$ the problem is a multi-class classification problem.

By constructing a model which can automatically classify which objects are in an image, much context and knowledge can be automatically inferred from photos without having to rely on humans. By using image recognition techniques together with feature extraction approaches, local features can be obtained from the photos which can be used for classification models. To build statistical models, machine learning can be used to automate the process of gathering information. Using a statistical model has the advantage that it detects objects faster than humans can, is likely more accurate and will be a lot cheaper in the long run. This use case also applies to Avanade, which is a joint-venture between Accenture and Microsoft. Avanade is an internationally operating Microsoft-based IT company specialized in IT consulting and services, they are currently using machine learning and computer-vision in their Analytics department to create client-tailored solutions.

1.2 Main objective of this study

The goal of this research project is to discover what the effects of imbalanced data sets are on the classification results and what the effect of different sampling methods are regarding the results of the original data set. By comparing different machine learning techniques, tuning parameters, trying several methods of data pre-processing and sampling of the data set, the goal is to find how the results of different classifiers change with different subsets of data obtained by sampling techniques.

To optimize the models, research has to be done to discover what optimizations are most effective in the process of constructing a good model for different sizes and structures of the data set on images and why the classifiers' results are different for particular sampling methods. The effect of the size of the data set used to train the model will also be researched, as this is a phenomenon that occurs often when working with data sets. Examples of possible optimizations are: Tuning the parameters of the models, pre-processing the data and possibly combining different machine learning techniques. The effect of undersampling and oversampling will also be studied and compared to stratified and random sampling.

As an additional step, several open-source machine learning libraries such as OpenCV, TensorFlow and scikit will be used to compare their feature extractors and classifiers. Open-source data sets with labeled data might be used to improve the usefulness of the model and to make a reliable proof of concept, this will also allow us to build a proof of concept if the provided data is not labeled or high quality. Using open-source data next to using the available data also allows for potentially automatically generating labels for the unlabeled data. One of the sub goals is that this research can provide a guideline on how to sample data sets when the data is imbalanced and give insight in when to use which sampling techniques and classifiers. Furthermore, the classifiers will be compared against several baseline algorithms to measure the performance of the models.

1.3 Scope of this thesis

For this thesis the COCO (Common objects in context) data set will be used as scope for this research to train and validate the models. The COCO data set is a good starting point to implement a proof of concept and to test the techniques as it is quite extensive. To further elaborate on this, the data set contains classes which occur often in images such as persons, vehicles and items that are used in everyday life. Furthermore, the images in the set are not always fully visible which makes it more of a general-purpose data set for training robust classifiers. The COCO 2014 train data set contains 82783 images with 604907 annotations which equals 7.3 annotations on average per image. An example of an instance from the data set with bounding boxes can be seen in Figure 1. The data set is roughly split into $\frac{1}{2}$ training data, $\frac{1}{4}$ validation data and $\frac{1}{4}$ test data which will help with validating the models (Lin et al., 2014). As the data sets are quite large the focus will be on using the train 2014 and validation 2014 sets because of computability. Due to time constraint, not every sampling method can be tested, because of this the sampling methods that will be considered are random sampling, stratified sampling and one over and undersampling algorithm. Another limitation is that only the COCO data set is used to verify the results, the classifiers and sampling methods might perform differently and thus yield different results on other data sets.

The main goal of this research is to find how the classifiers perform on different subsets of the original data set and which classifier is good for training on imbalanced data sets using the mentioned sampling methods. This thesis will only focus on supervised learning as this might give us more insight into what features to select and what the relation between subsets of data and the results is. Another reason to use supervised learning is that the performance of the sampling methods and classifiers can be measured. In the domain of data set subsetting the focus will mainly be on the different sampling techniques for data, the size of the data set and the relation with the results of the various classifiers.

1.4 Research Question

When taking the problem statement and scope of this thesis into account there is one question that that we want to answer in this research: What is the influence of sampling methods for training a classifier to classify images in an imbalanced multi-class data set?

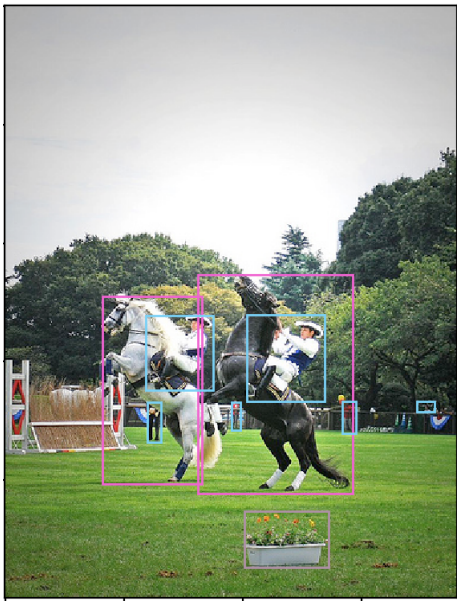


FIGURE 1: Image from data set with bounding boxes.

2 Literature review

In this section the intermediate results of the literature study will be discussed. First some comparable multi-class classification research will be analyzed. Then some literature regarding the problem transformation approach will be discussed.

2.1 Imbalanced data set classification

Imbalance inherently occurs in the real-world when decision systems are used to detect cases which are both rare and important, domains such as fraud detection, NLP and detecting oil spills are examples of real-world examples with class-imbalance (Lemaître, Nogueira, and Aridas, 2017; G. M. Weiss, 2004; Chawla, Japkowicz, and Kotcz, 2004). The problem with class-imbalance is that small classes are underrepresented in the dataset which might make it harder to classify these classes correctly while they might be important for the classification task (Lemaître, Nogueira, and Aridas, 2017; S. Kotsiantis, Kanellopoulos, Panayiotis Pintelas, et al., 2006). Imbalanced data sets still pose a big problem for data scientists. When using an imbalanced data set, the question rises whether the classifier will learn every class or whether it will simply predict the classes which are big in the data set (Maloof, 2003). One of the biggest factors in this is which metrics are being used to evaluate the performance of the classifiers. When using the accuracy as evaluation metric on a heavily imbalanced data set where the classifier predicts the majority class most of the time, the accuracy might be very high as there are many instances of that particular class in the dataset. The error rate of the smaller classes might be very high and the minority classes might be less likely to be predicted correctly while these classes might be very important. Metrics that are representative for the problem that is being solved should be chosen for evaluating models (G. M. Weiss and Provost, 2003).

If the data set that is being used has negative examples that outnumber the positive examples, some discriminative classification algorithms will overfit on one of the classes. A recognition-based classification algorithm might perform better than a discriminative one as the model is created only using the target class instances. One example of such a recognition-based classifier is the Support Vector Machine(SVM), this machine learning algorithm tends to perform extremely well in domains where the data is unbalanced (Chawla, Japkowicz, and Kotcz, 2004). Another problem next to class imbalance is the distribution of data in every class which is quite important (between-class versus within-class imbalance) (Zadrozny and Elkan, 2001; Chawla, Japkowicz, and Kotcz, 2004). While SVM's and Neural Networks generally perform well when the extracted features are continuous and multi-dimensional (S. B. Kotsiantis, Zaharakis, and Pintelas, 2007), the question arises how well they will perform on imbalanced data sets. An SVM can be used as a classifier with a one vs all approach as baseline where a classifier is built for every label.

Solutions for the class imbalance level can be implemented on both data and algorithmic level. When looking at the data level, several oversampling and undersampling such as random oversampling might be applied to counteract the class imbalance. One problem with sampling is that it is not always possible to sample the data. For example, when the minority classes do not have many instances it might be hard to generate new instances from the limited pool of existing instances without overfitting. At the algorithmic level, techniques such as adjusting the probability estimates, decision thresholds and one class learning might help improving the performance of the model (S. Kotsiantis, Kanellopoulos, Panayiotis Pintelas, et al., 2006; Maloof, 2003; Chawla, Japkowicz, and Kotcz, 2004; G. M. Weiss and Provost, 2003).

2.2 Sampling

Data sets as provided are not always optimally distributed for machine learning tasks (G. M. Weiss and Provost, 2003). One way to reduce the imbalance between classes is sampling. While sampling can offer a solution, the question remains: What is the optimal way to sample a data set into a smaller and more representative data set? Another interesting question regarding this is: What is the correct distribution for a learning algorithm? The distribution that occurs naturally is not always the optimal distribution for learning. A good reason to use undersampling is that it increases the sensitivity of the model to the minority class which makes the classifier more precise in its classification. Oversampling can be used to extend the number of examples for minority classes. While random undersampling can disregard some important cases in the data set and account for a loss of data, random oversampling might lead to overfitting on the training set. The problem with oversampling and undersampling is that the number of samples has to be detected empirically (Chawla, Japkowicz, and Kotcz, 2004; Chawla, Bowyer, et al., 2002; G. M. Weiss, 2004). Another problem which comes with oversampling is that extra computations have to be done to artificially create data which still resembles the data set (Chawla, Japkowicz, and Kotcz, 2004; Maloof, 2003; S. Kotsiantis, Kanellopoulos, Panayiotis Pintelas, et al., 2006).

The amount of undersampling and undersampling is mostly a process of experimentation and empirically set (Chawla, Japkowicz, and Kotcz, 2004). While undersampling and oversampling might improve the classifiers results, there are techniques that combine these two approaches. Well known examples of oversampling techniques which are also more advanced than random sampling are SMOTE (Synthetic Minority Oversampling Technique) and ADASYN (Adaptive Synthetic), whereas TOMEK links and ENN (Edited Nearest Neighbours) are well known undersampling techniques. When oversampling using these techniques the minority classes in the data set are fitted and synthetic minority class examples are added to the minority instances (Chawla, Bowyer, et al., 2002).

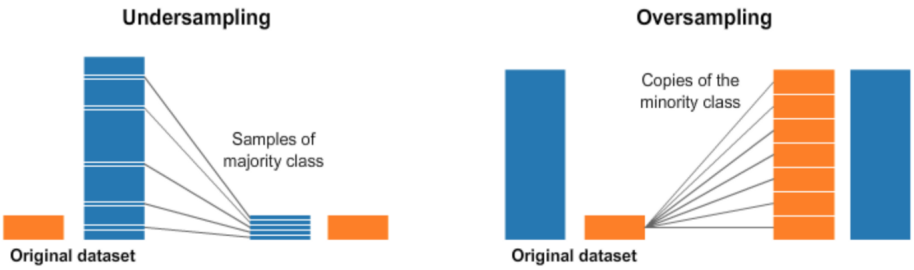


FIGURE 2: Undersampling & oversampling Badr, n.d.

2.3 Data stratification

The data set which is used to run experiments for machine learning tasks is very important. For supervised learning the data set is often split in a train, validation and a test set to evaluate the performance of the classifier. In the case of classification tasks, the stratification of data is quite important to keep the proportions of instances in the train, validation and test set close to the proportion of the original data set. When keeping these proportions similar by using stratification, the bias and variance of the classification are lowered (Kohavi et al., 1995).

Data stratification for image sets of labels can be achieved by using techniques such as the iterative stratification algorithm. This algorithm first calculates the desired number of instances per subset, then it calculates the number of class instances for every fold. When this is calculated, the algorithm finds the class with the least remaining instances and breaks ties randomly. Finally, it finds the subset with the highest number of examples for this particular label and breaks ties based on the number of labels for every fold (Sechidis, Tsoumakas, and Vlahavas, 2011).

2.4 Research gap

In the previous section it is evident that available literature regarding the field of data set sampling does not have a consensus on what data sampling techniques yield the best results in a general problem setting. One of the reasons is that the problem of data sampling is quite complex because of nature of the problem domains and the availability of the data (Chawla, Japkowicz, and Kotcz, 2004; Provost, 2000). Another possible reason is that image classification combined with imbalanced data sets is a combination of problems that is simply too hard to solve accurately for the current use cases. Images contain much information it makes classification more complex as features should possibly be selected (Hall, 1999). More research summarizing some of the popular techniques and their performance on artificial subsetted data is much needed in this research domain.

A topic that is not covered often is the stratification of sets of images which are all in one single image called image sets in this thesis. As image sets for every image in this data set could be regarded as a group of labels and sub-image called bounding boxes, sub-images from containing one object instead of an image with multiple objects, that belong together. The comparison between sampling image sets and separate bounding boxes might yield interesting results. Data stratification for single-label classification problems are well documented and this task is relatively easy. The stratification of the image set data proves to be harder as the item sets of the instances in the data set have to be considered. This makes stratification for these kind of image sets a non-trivial task which could be solved by techniques such as iterative stratification (Sechidis, Tsoumakas, and Vlahavas, 2011).

3 Research Objectives

The main objective of this research is to find out how different sampling methods influence the results of the chosen classifiers when using an imbalanced data set. More specifically, what are the effects of different sampling methods on classifiers such as the RF, SVM and CNN. As a great number of data sets are imbalanced in a way, it is quite important to research what the effects of unbalanced data sets are on the used classification algorithms.

3.1 Design

For this research standard machine learning practices will be applied. First the sets will be split into a training and test set, then the selected images will be pre-processed by using gaussian blurring and converting to gray-scale when necessary. After splitting, features have to be extracted from the images using techniques such as DAISY, Multi-Block Local Binary Pattern (multi-MLBP) and Histogram of Oriented Gradients (HOG) (Tola, Lepetit, and Fua, 2009;L. Zhang et al., 2007;Zhu et al., 2006). Classifiers such as a CNN, SVM and RF will be used to construct a classification model. When the models are trained, the performance of the classifiers will be evaluated. Scikit-image and openCV will be used to pre-process the images and extract features to be able to make a comparison between the different feature extraction techniques (Van der Walt et al., 2014; Bradski and Kaehler, 2000). TensorFlow and Scikit-learn will be used to train classifiers (Pedregosa et al., 2011; Abadi et al., 2016). Furthermore, to reduce the class-imbalance in the data set several oversampling and undersampling techniques will be used to make the data set more balanced and hopefully improve the classification results (Chawla, Bowyer, et al., 2002).

3.1.1 Research question

What is the influence of sampling methods for training a classifier to classify images in an imbalanced multi-class data set? Additionally, would it be possible to compare multiple models and find the best possible model within the scope of this research.

3.2 Research sub-questions

The main goal of this project is to research what the effect of sampling methods is on classification tasks using images and machine learning. This will help in order to generate valuable knowledge for Avanade. Extending consumer profiles will possibly give companies such as Avanade a better insight in their consumers which might in turn help optimize processes such as identifying business needs. The research is divided into several sub-questions which have to be answered to answer the main research question.

- Does pre-processing the data improve the model's results?
- Which metric(s) represent the performance of the models best?
- Which sampling methods represent the data in the best manner?
- Does a machine learning algorithm with feature extraction achieve a higher accuracy than using machine learning techniques without feature extraction?
- What machine learning technique or combination of techniques will yield the best results?

- How can the machine learning techniques be optimized further?
- Is there a possibility to account for possible biases when sampling such as the selection bias?

3.2.1 Does pre-processing the data improve the model's results?

Using pre-processing techniques might help training better classifiers. When pre-processing an image, noise can be reduced and uninteresting section can be smoothed. By reducing the noise and smoothing the images, a more robust classifier might be trained which performs better overall because it did not overfit as much as it would without pre-processing. Techniques such as Linear filtering are used to reduce noise in machine learning research, but how effective are these techniques? Some researches that use convolutional neural networks or kernel-based methods do not pre-process their images at all (Krizhevsky, Sutskever, and Hinton, 2012; Camps-Valls and Bruzzone, 2005). While in the classical machine learning context many pre-processing techniques such as whitening, Gaussian smoothing, Thinning or Edge Sharpening are available (Fayyad, Piatetsky-Shapiro, and Smyth, 1996; Bow, 2002). This raises the question which techniques generally improve the model's performance.

3.2.2 Which metric or metrics represent the performance of the model best?

Metrics are used to measure how well a machine learning model performs on a data set. While accuracy is often used as a metric to measure the performance of machine learning classifiers, it does not give a complete representation of the results of the model. More specifically, the rare or underrepresented classes in the data set which still might be very important do not influence the accuracy score on the same level as the majority class even if the minority class might be more important, this gives an intuitive reason why other metrics should also be taken into consideration (G. M. Weiss, 2004; S. Kotsiantis, Kanellopoulos, Panayiotis Pintelas, et al., 2006; Chawla, Japkowicz, and Kotcz, 2004). When using an imbalanced data set, metrics such as precision and recall in combination with accuracy might give a more complete overview of the performance of the model. This raises the question which metric(s) represents the performance of the model in the best way.

3.2.3 Which sampling methods represent the data in the best manner?

When a dataset is imbalanced, sampling methods might help to increase the performance of machine learning models. In the literature there are many different sampling methods for large datasets such as random sampling, TOMEK links, ENN (Edited Nearest Neighbours) and SMOTE (Chawla, Japkowicz, and Kotcz, 2004; Batista, Prati, and Monard, 2004; Wilson, 1972). This raises the question which sampling method represents the data in the best way and thus yields the best results. Another subject that might be interesting to study is why the sampling methods improve the results of the models. Can the best sampling method be generalized for other data sets or is the best sampling method dependent on the dataset where it is used on?

3.2.4 Does a machine learning algorithm with feature extraction achieve a higher accuracy than using machine learning techniques without feature extraction?

When looking at neural networks there are two possibilities when training the classifier. The first one is to extract features using feature extractors and using these extracted features to train the classifier. Another option is to give the raw image matrix as an input and let the neural network derive its own features from the images. An example of this is convolutional auto-encoders where neural networks are stacked upon eachother to use unsupervised learning to extract hierarchical features (Masci et al., 2011). To test this, TensorFlow will be used to train neural networks with and without pre-extracted features (Abadi et al., 2016).

3.2.5 What machine learning technique or combination of techniques will yield the best results?

As discussed in the previous section the main research question is which techniques perform best on the COCO data set. In order to derive the best techniques for training a classifier, experiments have to be done to get results that support a claim for best performing techniques. Furthermore extra literature research is needed to identify if there are additional techniques which are not yet mentioned in this proposal. All of the techniques that were mentioned before NN, SVM and RF will be implemented in python using the scikit-learn and TensorFlow libraries (Pedregosa et al., 2011; Abadi et al., 2016).

3.2.6 How can the machine learning techniques be optimized further?

While the standard machine learning techniques might yield decent results, are there any techniques that improve these results? The improvement might be achieved by pre-processing, optimizing hyperparameters or sampling methods. The techniques that further optimize the classification results are interesting because they improve the results but it also might give more insight in what is needed to build a good classifier.

3.2.7 Is there a possibility to account for possible biases such as the selection bias?

The COCO data set is quite skewed in the number of classes, for example the number of people in the data set is by far the largest class as can be seen in Figure 6a. Another problem that is present is how to select the training and testing set. Using non-randomly selected data might have a huge impact on the results and change outcome of the research (Heckman, 1977). It is very important that the results that are produced will be as free from bias as possible to fairly evaluate all of the techniques but also to make the research reproducible.

4 Experimental setup

In this section, the experimental setups used in this research will be explained and discussed. The following subjects will be discussed in the following subsections:

1. Data
2. Metrics
3. Models
4. Dummy classifier
5. Feature descriptors

The software and hardware are not extensively covered in this thesis as these specifications are not fully conclusive. To give a better insight in this project the software and hardware that were used will be summarized. For this research Python was used to implement all of the pre-processing, feature extractors and classifiers. Jupyter notebook was used to make visualization and re-using calculated features and data sets easier. To train and validate the classifiers, the Azure machine learning workspace on the Microsoft Azure platform was used as training on a laptop would take a long time. To train the SVM and RF a CPU cluster was instantiated. The CPU clusters that were used are the STANDARD_Ds2_V2 up until the STANDARD_Ds4_V2 VM which uses an Intel Xeon E5-2673 v3 @ 2.4 GHz CPU as can be seen in figure 3. As training a neural network is usually faster on a GPU, a GPU cluster was instantiated. The GPU cluster that was used is a Standard_NC6 VM, these VM's use a NVIDIA Tesla K80 GPU, the specifications can be seen in figure 4. Both the CPU and GPU cluster used two of the Microsoft VM's, this equals one Intel Xeon E5-2673 v3 @ 2.4 GHz CPU and one NVIDIA Tesla K80 GPU. To run additional experiments when the machine learning workspace was unavailable, a macbook was used. The specifications of this machine can be found in table 1.

Size	vCPU	Memory: GiB	Temp storage (SSD) GiB	Max data disks	Max cached and temp storage throughput: IOPS / MBps (cache size in GiB)	Max uncached disk throughput: IOPS / MBps	Max NICs / Expected network bandwidth (Mbps)
Standard_DS1_v2	1	3.5	7	4	4000 / 32 (43)	3200 / 48	2 / 750
Standard_DS2_v2	2	7	14	8	8000 / 64 (86)	6400 / 96	2 / 1500
Standard_DS3_v2	4	14	28	16	16000 / 128 (172)	12800 / 192	4 / 3000
Standard_DS4_v2	8	28	56	32	32000 / 256 (344)	25600 / 384	8 / 6000

FIGURE 3: Specifications of the used CPU for the RF and SVM experiments.

Size	vCPU	Memory: GiB	Temp storage (SSD) GiB	GPU	GPU memory: GiB	Max data disks	Max NICs
Standard_NC6	6	56	340	1	12	24	1
Standard_NC12	12	112	680	2	24	48	2

FIGURE 4: Specifications of the used GPU for the NN experiments.

TABLE 1: specifications of the macbook CPU used for NN, RF and SVM experiments.

Version	MacBook Pro (Retina, 15-inch, Mid 2014)
CPU	2,8 GHz Intel Core i7
Cores	4
Memory	16 GB 1600 MHz DDR3

4.1 Data

Meaningful data is an essential component of machine learning. In this section the data set which will be used will be discussed first. Secondly the stratification of the different subsets will be covered. Finally, upsampling and downsampling will be discussed and the potential uses and benefits that these techniques will bring to this research.

4.1.1 COCO data set

The COCO data set was made with the goal to create a data set which will help the advance of scene understanding. The data set covers three core research areas of scene-understanding:

- Detecting non-iconic views.
- Contextual reasoning between objects.
- Precise localization of objects.

The reasoning behind this is that when searching for an object online such as a bicycle, there exists an iconic view which is an unobstructed image. Nowadays classifiers perform quite well on unobstructed and non-occluded images but these classifiers seem to struggle when objects are occluded or cluttered as can be seen in Figure 5 (Lin et al., 2014). As mentioned in the introduction the COCO data set is used to train classifiers for multi-class classification, object detection and scene labelling.

The COCO data set consists of images with possibly multiple instances of the classes present in the data set. To find the objects in the training and validation data sets, a JSON file is provided with the instances of classes for every image and the location of this object as a bounding box. The average size of a bounding box in the COCO data set is 104x107, all of the bounding boxes were resized to 50x50 for a uniform input. Several training, validation and test data sets are available on the COCO data set website http://cocodataset.org.

4.1.2 Class imbalance

When training machine learning classifiers on data sets it is important to take certain biases into account such as the selection bias. The distributions of the COCO data sets are similar as can be seen in Figures 6a & 6b. However, it is easy to see that the number of classes is not equally distributed in the data set. This poses a problem for training the classifiers as some classes have less than 200 instances in the data set which means that these classes can be especially hard to learn.

The fact that this data set is skewed has to be taken into account when drawing conclusions from the results of the trained classifiers. Machine learning while using an unbalanced data set poses an interesting problem when trying to train a classifier as there are many possible solutions to this problem. When 1 class out of 80, in this case

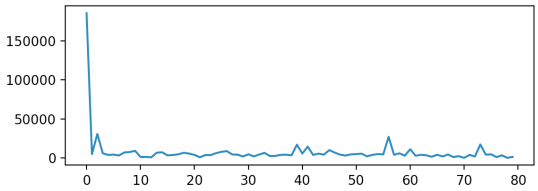
FIGURE 5: An example of an occluded object (bicycle) in the data set.

persons in the COCO data set, makes up for about 30% of the data set it would be an intelligent classification pattern for the classifier to predict person 100% of the time. This is due to some assumptions built into most classification algorithms: Maximizing accuracy and when the classifier is used in a production or predictive environment, the training data represents the test or production data. On one hand one could say that the data that is provided in the data set is the data that should be used to train the classifiers. On the other hand, one could say that artificially balancing the data set might lead to better classification results. Whether artificially balancing the data set always leads to better results is still debated as some classification methods are more robust to class imbalances than others (Provost, 2000; Chawla, Japkowicz, and Kotcz, 2004).

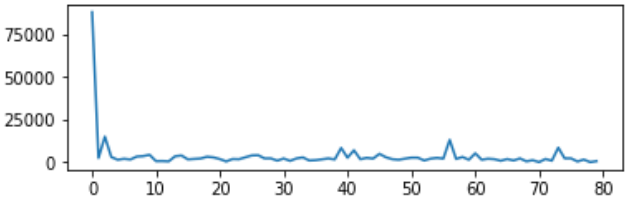
In some domains the class imbalance such does not pose a big problem as in some settings the number of instances of a certain class such as fraud is quite small in comparison to the number of non-fraudulent transactions. The dataset might still have to be sampled in a way to learn the important class, which is fraud, but the proportions of the data are natural to the domain. In other cases, the class imbalance occurs in domains where such an imbalance is not common which poses a problem, in these cases it is up to the data scientist to decide how to tackle this problem.

Over-sampling is a method where classes with a low prevalence in the data set are added in the train and test set in order to have more instances for the classifier to train on and create a more balanced representation of classes. Under-sampling is complementary in over-sampling in the sense that it reduces the amount of instances of classes which occur often in the data set (Lemaître, Nogueira, and Aridas, 2017). A problem that follows from this skewed data set is that the data should likely be stratified into subsets to measure the performance of classifiers on smaller data sets. The problem of stratified sampling on images is not necessarily trivial and multiple methods will be tested to measure the difference in the results.

Another interesting measure is the label density, which can be calculated using $LD(D) = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{|Y_i|}{|L|}$, where D is number of instances the dataset, Y_i number of labels for the current image and $|L|$ is the number of labels. As can be seen in Table 2 this data set is quite sparse as the label density is 0.091 which is low. However, the validation and test sets are similar in the distribution of labels and density and a proportional number of images, annotations and unique labelsets. If these metrics were too different the data sets would not exhibit the same properties which would make the research harder (Tsoumakas and Katakis, 2007).



(A) Distribution of labels in the train set.



(B) Distribution of labels in the val set.

TABLE 2: Dataset statistics.

	Train 2014	Val 2014
Images	82783	40504
Annotations	604907	291875
Annotations/image	7.307	7.206
Label density	0.0913	0.0901
Unique labelsets	82081	40137

4.1.3 Stratification of the subsets

The stratification of multi-label data is a non-trivial task as one image can have an arbitrary number of labels. Stratification splits a data set in a way that every smaller subset approximately contains the same proportions of the labels in the original data set, this way the subsets still represent the original data set. When using stratified data instead randomly split data in cross-validation, the bias and variance are decreased in comparison to non-stratified data (Kohavi et al., 1995). In order to see how important sampling is, three sampling methods are implemented for this research: Random sampling, probability mass split and iterative sampling. The random sampling method serves as a baseline to see which results are obtained when sampling from the original data set at random.

The probability mass function (PMF) is a measure that gives the possible values for a random variable. By using the PMF for splitting the original data set, the occurrence of labels in the labelsets in every fold is considered and should therefore yield better results than randomly splitting a skewed data set.

Finally, the iterative sampling method starts by calculating the number of desired labels for every subset. This algorithm iteratively examines the labelset and greedily picks the label with the fewest remaining examples to assign the labelsets containing this label to the folds. By greedily picking the labels with the least remaining examples the algorithm ensures that these labels are also stratified in the subsets. The labelsets are assigned to the subset which deviates most from the proportions of the original data set. When a tie occurs between two or more subsets, the data set with the highest number of desired examples is selected and further ties are broken randomly (Sechidis, Tsoumakas, and Vlahavas, 2011).

4.1.4 Sampling

As mentioned in earlier section, the COCO data set is not close to being uniformly distributed. This might be a problem when training the classification models or trying to generalize them. Class imbalance is an important problem to take into account when training classifiers as the data sets have a large influence on design and the results of the classifiers. It is often the case that misclassifying classes which are not common in a data set are most of the time more important to classify than the examples in the

data set that are common. Techniques that are often discussed in literature to tackle this problem are oversampling and undersampling. Two of the most known sampling methods are: random over-sampling and random under-sampling (Chawla, Japkowicz, and Kotcz, 2004; Maloof, 2003; Prati, Batista, and Monard, 2009; Lemaître, Nogueira, and Aridas, 2017).

When downsampling, only a subset of the large classes is used to make the distribution of the test set more uniform. Two downsides of using this method is that a portion of the large classes is never used and that bias is added by arbitrarily selecting a subset of the original data set (Chawla, Bowyer, et al., 2002; Maloof, 2003). Another problem that occurs when undersampling is that the undersampled data set might not be adequate to learn the classes which are in the data set (Maloof, 2003). This raises the question how many of the instances of the class should be sampled in order to get results which are not completely useless because of the bias. The downsampling in this research will be done by randomly downsampling large classes to a fixed number of instances to balance the data set.

Oversampling might increase overfitting on the train set as classes which are not represented equally in the data set might be sampled as exact copies of these instances of the class. Another option is to artificially create the classes which are underrepresented by using interpolation to create instances which are similar to the instances in the data set but are not exact copies. Even better results might be booked when noisy examples are removed from the classes and the classes are equalized by oversampling on examples which are not noisy (Prati, Batista, and Monard, 2009). As the definition of an optimal region to sample is not equal for every dataset, several sampling techniques were constructed. The difference between the sampling techniques is whether these techniques focus on synthesizing borders of classes to make the borders between classes clearer, whereas other sampling techniques focus on finding hard to learn classes and instances and focus on synthetically sampling these instances and classes (Branco, Torgo, and Ribeiro, 2016). However, the question still remains whether the generated data is not biased and whether the classification algorithms are prone to overfitting on the train set due to this generated data.

4.2 Metrics

The metrics that are chosen are quite important in machine learning tasks. When analyzing whether a machine learning classifier performs well on the task it is supposed to perform, metrics represent the performance in different ways. The metrics that are often used in machine learning are Accuracy, Precision, Recall and F1. Accuracy is the metric that is often used for measuring the performance of classifiers because it represents the ratio of correctly predicted true positives and true negatives to the complete predicted set. Precision is a metric which represents the ratio of the true positives to the true positives and false positives, this metric gives an insight in how many of the positive predicted classes were actually predicted correctly. The recall is the ratio of the true positives to the true positives and false negatives which represents the ratio of how many of the positive predicted classes that are correctly identified. Finally, the F1 metric represents the harmonic mean between the precision and recall.

Depending on the problem that has to be solved using an imbalanced dataset, there are three interesting metrics that have to be considered. As an example, the precision of this imaginary classifier will be calculated, let us imagine that there are 3 classes called A,B,C with a number of true positive (TP) and false positive (FP) predictions.

- A: 1 TP and 1 FP.

- B: 9 TP and 1 FP.
- C: 1 TP and 1 FP.

It is easy to see that the probabilities of a correct classification are Pr(A)=$\frac{1}{2}$, Pr(B)=$\frac{9}{10}$ and Pr(C) = $\frac{1}{2}$. The macro average is calculated by adding the probabilities and dividing by the number of classes. The macro precision of the classifier for this example = $\frac{0.5+0.9+0.5}{3}$ = 0.633. The macro average of the metrics may be quite important for imbalanced data sets as it gives as much weight to classes that do not occur often as to classes that are the majority class. The micro average is calculated by adding the true positives of every class and dividing by the total number of predictions, Pr(classifier)=$\frac{1+9+1}{2+10+2}$ = 0.786. A third way to calculate interesting metrics is to calculate the weighted precision, recall and F1. When calculating the weighted metrics, the metric values of different classes are added based and given a weight based on the size of the class. A final note that has to be made when discussing these different metric scores is that the micro score is equal for every metric in the multi-class domain. The metrics are also important for validating the model in a reliable way as the different metrics give different insights in how the classifier performs (Manning, Raghavan, and Schütze, 2010). As these metrics are biased it is important that multiple metrics are taken into account in order to evaluate the model as objective as possible (Powers, 2011). Measuring the performance of a classifier with accuracy is not always the best solution as the cost of classifying some classes incorrectly might have a higher cost (Chawla, Bowyer, et al., 2002).

$$Accuracy = \frac{TP + TN}{TP + FP}, \tag{1}$$

$$Precision = \frac{TP}{TP + FP}, \tag{2}$$

$$Recall = \frac{TP}{TP + FN}, \tag{3}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}. \tag{4}$$

Some classification tasks might require more focus on getting the cases that are predicted as true absolutely correct. For these cases the metrics precision and recall might be preferred. Precision represents the proportion of true positives to the true positives and false positives, this metric represents how good the classifier is in not making incorrect true predictions. Recall represents the proportion of the true positives compared to all of the 'positive' predictions that the classifier made. Finally, the F1 metrics might be the most popular metric lately, it represents the harmonic mean of the precision and recall metrics. Even though this performance metric is widely used, there are arguments such as that the relative importance of the precision and recall should be dependent on the problem and the researcher instead of always choosing the harmonic mean (Hand and Christen, 2018). Because accuracy does not give complete information on the performance of the classifier, the Precision, Recall and F1 metrics are also calculated in

Table 3: Example of a confusion Matrix.

	Actual positive	Actual Negative
Predicted positive	TP	FP
Predicted negative	FN	TN

order to see if there are any interesting patterns that might be discovered (He et al., 2008).

4.3 Models

One of the most important parts of this research is to compare classifiers such as Convolutional Neural Networks, Random Forests and Support Vector Machines and measure which classifiers perform best for different inputs. These inputs include subsetting the data in a stratified and non-stratified manner, pre-processing, time-limitations and limited computational resources. A comparison can be made between very straightforward such as random forests and more complex models such as the SVM and Neural Networks. It will also give an insight which factors are important when trying to classify which objects are on images. Another interesting comparison to make is between the performance of pre-extracted features by using feature extractors and feeding these features to the SVM and RF and using a CNN to extract its own features.

4.3.1 Convolutional Neural Networks

In the traditional models of machine learning, human made feature extractors have to gather relevant and distinct representation of the images and make these representations simpler than the input. Whenever all the features are collected, the extracted features can be given as input to a trainable classification model. This model than tries to categorize the input into classes and learn what representation classes should have. A different way to solve this problem is by having the model extract the features and reduce the size of the extracted features compared to dense layers using convolutional layers. This way the neural network can extract low-level features such as corners, edges and colors and combine these low-level features into higher level features in higher layers (LeCun, Bengio, et al., 1995). Another interesting step in the convolutional neural network is the pooling step, this step reduces the dimensionality of the feature map by downsampling.

The neural network uses an error rate to measure the progress and how well it is learning. The error rate is defined by Total Error_rate $= \Sigma\frac{1}{2}(targetprobability - outputprobability)^2$. The steps that a neural network goes through are:

1. Initialize parameters and weights with random starting values.
2. Network gets image as input and goes through forward propagation.
3. Total error of the output layer is calculated.
4. Backpropagate and calculate the gradients of the error rate and update weights accordingly using gradient descent.
5. Repeat previous steps for number of epochs.

When using a neural network as a classifier there are several design choices that should be made when building it. As only neural networks are used in this research the

scope can somehow be limited to exclude some architectures such as: Long Short Term Memory networks (LSTM) and Boltzmann Machine as the scope of the thesis has to be narrowed down. Some of these design choices include:

- Activation function.
- Stride length.
- Number of convolutional layers.
- Size of convolutional layers.
- Number of dense layers.
- Size of dense layers.
- Optimizer.
- Pooling layers.
- Dropout rates.
- mini batches.
- Data augmentation.

The search space that has to be searched to optimize all of these options is too large to fully search, so some assumptions have to be made from intermediate results. In order to get some benchmarks and establish a good baseline, the focus will be one architecture which have been used a lot in the past such as Conv-Pool-Conv-Pool on the ImageNet data set (Krizhevsky, Sutskever, and Hinton, 2012). The rectified linear unit (ReLU) activation function works really well with (convolutional) neural networks so this activation function will be chosen instead of the Sigmoid or Tanh activation functions which are used in many general-purpose neural networks (Nair and Hinton, 2010; Krizhevsky, Sutskever, and Hinton, 2012). The reason that the RELU activation function is picked for this research is the vanishing gradients problem. When training Neural networks which use gradient-based learning methods and backpropagation, each of the weights is updated proportional to the partial derivative of the used error function. The problem occurs if the gradient becomes small enough that it does not change the value of the weights anymore (Hochreiter, 1998).

- ReLU: $h = \max(0, x)$.
- Sigmoid: $h = \sigma(x) = \frac{1}{1+e^{-x}}$.
- TanH: $tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$.

Where x is the input to a neuron of the NN in the earlier listed formulas. Furthermore, the SoftMax function will be used as the activation function in the dense layers, the loss function is categorical cross entropy. The optimizer that is used for the CNN's in the experiments is the ADAM optimizer, ADAM is a gradient-based optimization algorithm for stochastic objective functions. Some of the upsides of this algorithm is that it is efficient, good for large problems and can handle noisy and sparse gradients (Kingma and Ba, 2014).

SoftMax: $y_k = \frac{e^{z_i}}{\sum_j^c e^{z_j}}$

For the data augmentation the comparison between color and grey-scale will be made in order to see whether much information is lost when disregarding the color. Furthermore, the images are normalized by dividing every pixel by 255 which helps removing some minor distortions and makes the images easier to work with. The Keras data generator object is also used to flip, mirror and shift the images to reduce the overfitting on the train set. Methods such as Dropout rates and mini-batches also help optimizing the model by preventing the overfitting of the Neural Network on the train data set. The dropout rate deactivates some neurons based on the dropout rate for every epoch (Krizhevsky, Sutskever, and Hinton, 2012).

4.3.2 Random Forest

A random forest fits multiple bootstrap samples of the data to decision tree classifiers and uses majority decision to make a classification. Using ensembles of trees that are trained on multiple bootstrap samples of the data should prevent overfitting and therefore lead to more accurate predictions. Because decision trees tend to overfit on the training set when the tree depth is deep with low bias but high variance, random forests seem to be a better choice as this classifier reduces the variance of the decision tree. The random forest achieves this by using bootstrap aggregating (bagging), bagging selects a random subset of the original data set by random sampling with replacement from the full data set (Liaw, Wiener, et al., 2002; Breiman, 2001).

The number of trees is a parameter that is not known in advance and the optimal number of trees in a random forest can be determined by cross-validation or using the out of bag error. Furthermore, the random forest uses at each split a random subset of the features that are given to the classifier, the thought process is that the best subset of features which are strong predictors are chosen for the classifier. The number of features used for a classification task with n features is often $\sqrt{n}$ when using a random forest (Breiman, 2001).

4.3.3 SVM

A Support Vector Machine (SVM) constructs a hyperplane which separates different classes in a classification task. Because there are many hyperplanes which can separate data, the hyperplane with the largest error margin is chosen. This makes SVM's quite robust, also when the training set is small (Foody and Mathur, 2004).The instances from the data set are projected to a n-dimensional space where n is the number of features for every instance in the data set, where every feature is a coordinate in this n-dimensional space. For classes which are not linearly separable or where the separation of the classes is not good, the kernel trick can be used. Instead of using a linear kernel, kernel functions that transform low dimension input and transforms it to a higher dimensional space. The kernel trick transform uses a kernel function to compute dot products between instance vectors in a high dimensional space without having to transform the data to this high dimensional space to find a kernel which separates the data in a non-linear manner (Camps-Valls and Bruzzone, 2005; Liu and Y. F. Zheng, 2005; Foody and Mathur, 2004).

Usually the SVM tries to separate two classes in the best way possible. In order to be able to use the SVM for multi-class problems it has to be transformed to either a one-vs-one classifier on a one-vs-rest classifier. The one-vs-rest approach trains the SVM on one class and uses all the other classes as negative examples. This way the classifier can predict whether an instance belongs to a specific class in a multi-class setting. On the other hand, the one-vs-one approach is trained for every possible combination of

two classes (Liu and Y. F. Zheng, 2005). For this research the one-vs-rest classifier is used as it is most suited for the current problem in terms of scalability.

For the Support Vector machine, both the linear and radial basis function (RBF) kernel will be compared. The disadvantage of the linear kernel is that the performance is often lower than that of the RBF kernel. An SVM with an RBF kernel takes a long time to compute, for data sets with more than 10000 samples it is recommended to use the linear kernel. The first benchmarks will be done with the linear kernel as the implementation of the non-linear SVM is based on the libsvm library implementation and the time complexity is more than quadratic with the number of samples in the train set. The SVM with the linear kernel is based on the liblinear library and should scale better with a bigger data set (*scikit-learn.org SVC* n.d.).

- Linear kernel = $K(x_i, x_j) = x_i \cdot x_j$.
- RBF kernel = $K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \sigma \in \mathbb{R}$.

Where x_i and x_j are two instances from the dataset, the dot product of x_i and x_j are 0 when the features are dissimilar. Furthermore the σ value is determined a priori and is subject to optimization methods such as grid search.

4.4 Dummy classifier

To establish a baseline for the classification task on the COCO data set, the dummy classifier from the scikit-learn library was used (Pedregosa et al., 2011). This dummy classifier uses simple rules to classify instances in the data set. The results are expected to be low, but it will give an insight in the minimal classification rates which should be achieved. The rules which are used for the dummy classifier are the following.

- Stratified.
- Most frequent.
- Prior.
- Uniform.

The stratified rule uses the distribution of the training data set in order to make predictions on the test set, this approach takes the proportional occurrence of classes into account. The most frequent rule always predicts the label that is represented most often in the training set. This most frequent rule might be quite simple, but the occurrence of the most common label person makes up for almost 30% in the data set. The prior rule always predicts the class label that maximizes the class prior. Finally, the uniform rule randomly generates uniform predictions for the data set. These baselines will be used to explain how the results of the different classifiers and sampling techniques are impacted.

4.5 Feature Descriptors

In this section the feature descriptors which are used to classify images with the Random Forest and Support Vector machine will be discussed. A feature is a type representation of an image that is obtained by simplifying the image and extracting the important information. There are two types of features, global features where one single feature vector describes the entire image and the local features where the image is described

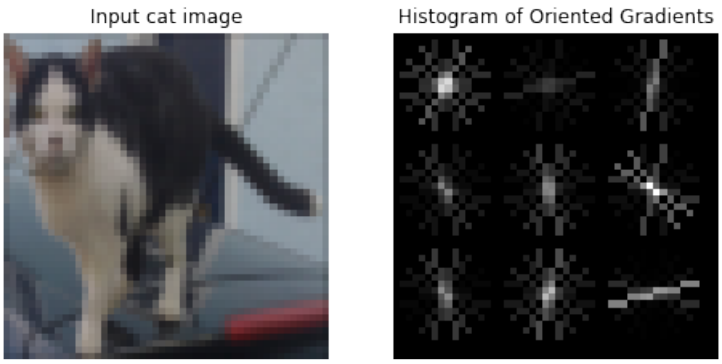


FIGURE 7: Histogram of oriented gradients on an image of a cat.

by interesting regions in the image instead of describing the entire image (Awad and Hassaballah, 2016). This section consists out of three sections where every feature used for the classification will be discussed. The feature descriptors used for this research are Histogram of Oriented Gradients (HOG), DAISY and Multi-Block Local Binary Pattern (MLBP).

4.5.1 Histogram of Oriented Gradients

The Histogram of Oriented Gradients (HOG) is a feature description method that has its basis in the evaluation of normalized local histograms of image gradient orientation in a dense grid. One of the assumptions of the HOG descriptor is that local objects and shape in images can be described by the distribution of gradients. When computing the features, the image is divided into connected regions which are called cells. For the pixels in every cell, the histogram of gradients is calculated. The calculation of the HOG feature is done by first calculating the horizontal and vertical gradients from the images. After this is done, the magnitude and direction of the gradients is computed. Finally, the histogram of gradients is constructed, which contains 9 bins from 0-180 degrees. An example of extracted features from a HOG feature extractor can be seen in figure 7 to get a better understanding of what this feature extractor is.

HOG can both be used for gray-scale and color images, when using color images to compute the features the gradients have to calculated for each color channel. From these calculated features, the one with the largest norm is picked as the gradient vector (Dalal and Triggs, 2005). The setup for the HOG descriptor for this experiment has 8 orientations, 12x12 pixels per cell and 1x1 cell patches. The block size is essential for the classification, the size has a big impact on the extracted features. While a small block size might miss global features, adding blocks with different sizes are likely to increase the computational costs (Zhu et al., 2006).

4.5.2 DAISY

DAISY is one of the most critically acclaimed feature extractors in the field today, the local region descriptors which are used often today are SIFT and GLOH. The problem with an algorithm like SIFT is that it is proprietary and that it might take a long time to compute, DAISY tries to compute the feature descriptors for every pixel in a faster manner. This is good for a bag-of-features representation of the image. The SIFT and GLOH algorithms use the gradient orientation histograms, these histograms are robust to distortions, occlusions and light. Another difference between DAISY and SIFT is

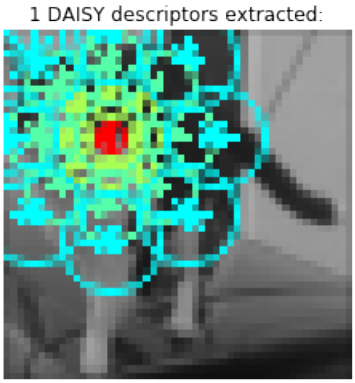


FIGURE 8: DAISY feature extractor on an image of a cat.

that DAISY uses a gaussian kernel, whereas SIFT uses a triangular kernel. The DAISY descriptor is a vector made out of the values from convolved orientation (Awad and Hassaballah, 2016). An example of the daisy feature extractor can be seen in figure 8

4.5.3 Multi-Block Local Binary Pattern

The Multi-Block Local Binary Pattern (MLBP) is a local feature descriptor that uses local binary pattern operator to encode rectangular areas. This feature descriptor can be computed quickly by using the integral image. The MLBP is quite similar to haar-like features but the set of MLBP features is smaller than the set of haar like features which gives it an advantage as the usage of the feature is simpler and it also makes the implementation of feature selection potentially easier. The MLBP is a descriptor that encodes the intensities of rectangular areas by local binary patterns which can capture large scale structures in images, an example can be seen in figure 9. The output of this feature descriptor can be used as a descriptor of the image (L. Zhang et al., 2007).

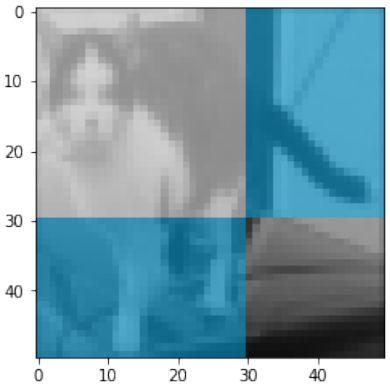


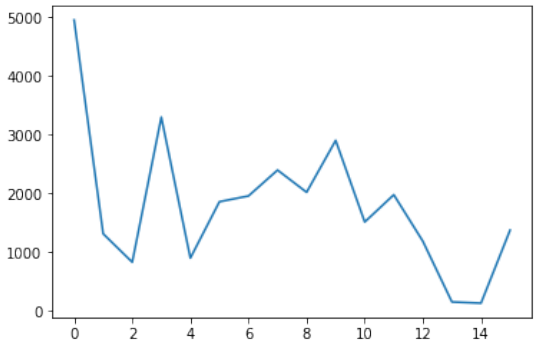
FIGURE 9: MLBP feature extractor on an image of a cat.

5 Results

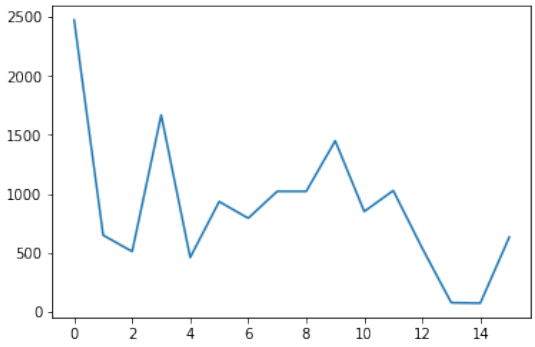
In this section the results of the conducted experiments for this research will be discussed. The goal of these results is to answer the question: What is the influence of sampling methods for training a classifier to classify images in an imbalanced multi-class data set? The subsections are divided in the different sampling methods. Some sampling methods use the imbalanced dataset as is, while other try to balance the dataset by sampling.

5.1 Data

The models used can compute any imbalanced data set, but to limit the study the decision was made to test and optimize the classifiers for a subset of classes of the original COCO data sets and stratified subset of the entire COCO data set. By doing this, the results and conclusions in this research will lead to more substantial claims as there are more metrics available for different models. This limitation to the research was made because the thesis would become too large without this limitation. The subset of classes chosen from the COCO data set is the following collection of classes (2,11,14,17,23,34,36,39,58,65,74,76,78,80,89,90) called the subset, the classes can be found in Table 31. This particular set was chosen as the classes in this set are spread across the various super categories present in the data set, because the set is also imbalanced making it a good set to test the various models and sampling methods and because the models can be computed within the resource limitations of this research. The subset and the original set do not necessarily have the same level of class imbalance, but it is a good place to start experimenting before moving on to the entire data set. The imbalance of the COCO data set can be seen in Figure 6a and Figure 6b, the distribution of the subset can be seen in Figure 10a and Figure 10b.



(A) Distribution of labels in the train subset.



(B) Distribution of labels in the val subset.

Another interesting thing to note about both datasets is that the clusters of classes overlap quite often, this means that the classifiers might have difficulties correctly classifying the instances of classes which are essentially outliers. This will most likely reflect in the results of the models but might be partially solved by undersampling the weakest instances of the class. The clusters of the data can be seen in Figure 11 and Figure 12.

5.2 Optimizing the models

In order to get optimal results from the classifiers, they have to be optimized by training on a (sub)set of the COCO dataset. This will give insight in the optimal hyperparameters or architecture of the classifiers. As mentioned in the previous section there are

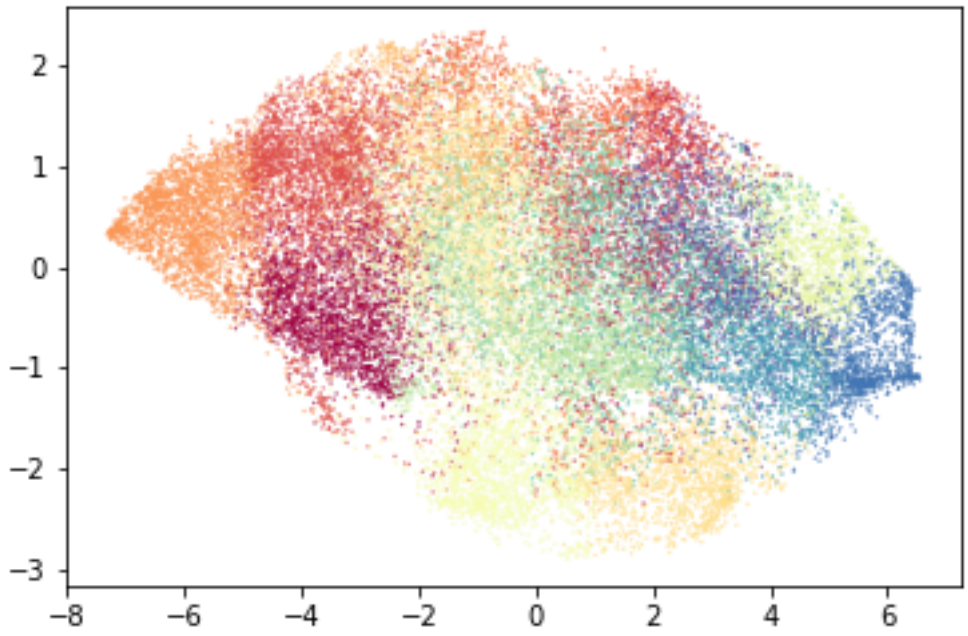


FIGURE 11: Clustered sampled class subset (16 classes) of the COCO subset, the axis are used to illustrate how far clusters of classes are from eachother. The closer clusters of classes are together, the more similar these classes are to eachother.

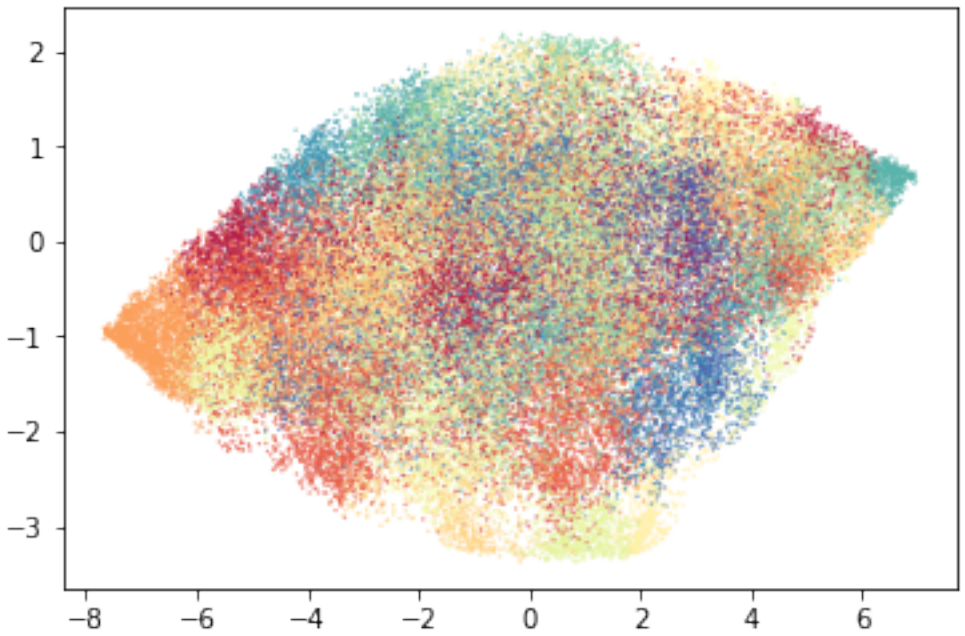


FIGURE 12: K Means and PCA to cluster the COCO dataset, the axis are used to illustrate how far clusters of classes are from eachother. The closer clusters of classes are together, the more similar these classes are to eachother.

two datasets on which the classifiers are trained and validated, the COCO dataset and a subset with a collection of classes, called the class subset, from the original dataset. To resample the data, several under and oversampling techniques are used next to the stratified dataset. For undersampling the undersampling techniques which are used are TOMEK links, ENN and random undersampling, the oversampling technique that is being used is SMOTE. Finally, one technique that uses both undersampling (TOMEK links) and oversampling (SMOTE) is used called SMOTETOMEK.

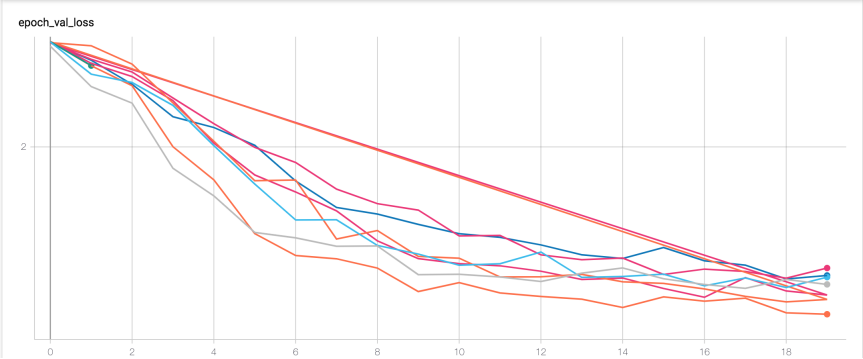


FIGURE 13: Validation loss for the best performing neural networks from the grid search, where the orange line is the 3-128-3-128 architecture

TABLE 4: The architecture of the neural network which found by grid searching different architectures.

Convolutional Layers	3
Kernel size	3
Dense layers	3
Convolutional layer size	128
Dense layer size	128
Pooling	(2,2)
Batch size	70
Dropout rate	0.35
Activation function	ReLU
Activation function dense	SoftMax
Loss function	Categorical cross entropy
Optimizer	ADAM
Early Stopping	10

The effect of these different sampling techniques on the different classifiers will be researched using this dataset. In order to find the optimal architecture of the Neural Network, a grid search was done on the different possible architectures as can be seen in Figure 13. The best Neural Network architecture of this grid search can be seen in Table 4, the chosen architecture is conv-pool-conv-pool-conv-pool. To further prevent overfitting, a gaussian noise layer was added as input layer of the model to introduce noise to the images which will make the learned features more robust and generalizable. Furthermore, an early stopping monitor is added, this callback stops the model when it is not improving anymore. Finally, drop out layers are added to prevent overfitting on the train set.

When looking at the feature extractors for the SVM and RF, there were some interesting findings and limitations. Firstly, images need to be in gray-scale for the DAISY feature extractor, so all of the images were pre-processed and converted to gray-scale to be able to extract features from the images. However, some information is lost when converting images to gray-scale as the colors of to be predicted classes might have much information about which class is being represented. To elaborate further, the neural network yielded better results when images had 3 channels (r,g,b). Furthermore, the SVM failed to converge when using the MLBP features so these features had to be omitted for the SVM. When optimizing the random forest, it became apparent that 1000 trees was the optimal number of trees. The number of trees used was 500 as the increase of performance between 500 and 1000 trees was marginal. Finally, the SVM had a better performance with the RBF Kernel than it had with a linear kernel as was to be expected as can be seen in Table 19 and Table 26 located in the appendix.

Another interesting finding is that the images can be greatly downsampled by using PCA, as can be seen 700 components describe approximately 97.5% of the data. If we compare a 50x50x3 image which has 7500 components to 700, 9.33% of the components have enough descriptive power which greatly reduces the size of the images as can be seen in figure 14. This process omits many non-meaningful features. As can be seen in figures 15a & 15b, the image of the bicycle is greatly blurred when PCA is used but the object in the image is still visible for the human eye.

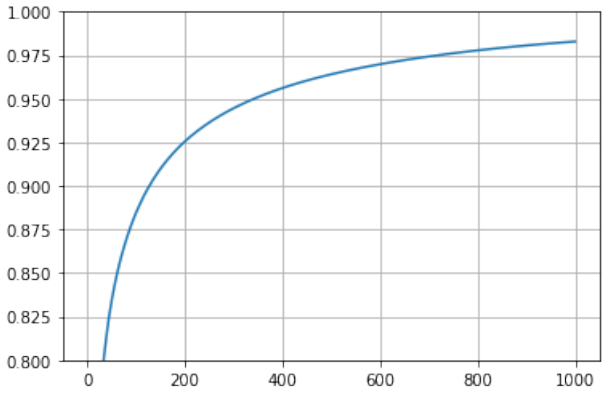
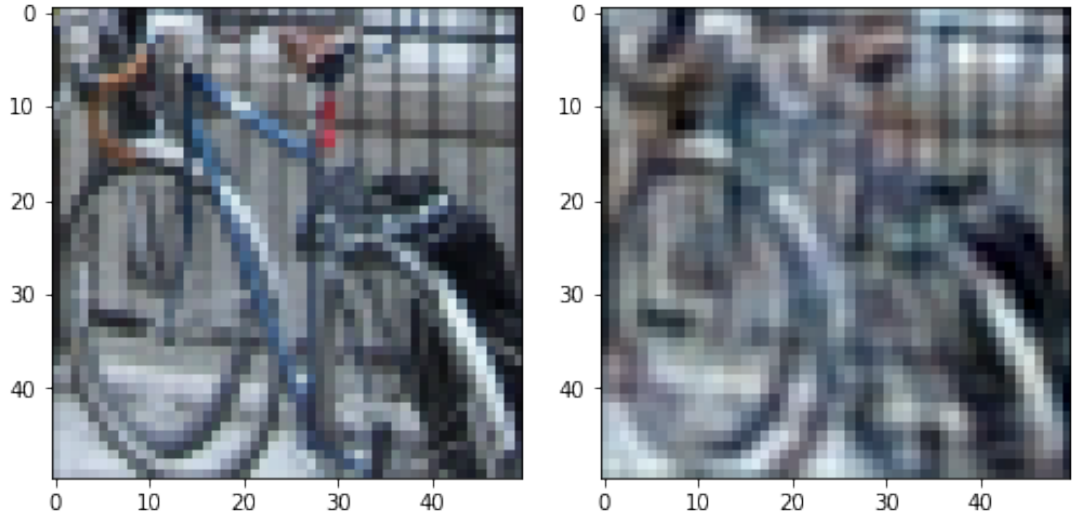


FIGURE 14: PCA curve for the class subset, 700 components describe about 97.5% of the dataset.

As can be seen in Figure 16, all of the classifiers outperform the dummy classifier by a large margin. The Neural Network has the best results as was to be expected as it is most likely the most flexible classifier when learning classes due to the convolutional layers instead of the pre-extracted features that the RF and SVM use. When looking at the classification results of the models which were trained on the undersampled dataset using the TOMEK links it is quite surprising to see that the classification results are either similar or even worse. The expected results were that TOMEK links would eliminate any instances of classes which were outliers and therefore improve the performance of the model. The other undersampling techniques, ENN and random undersampling, also did not improve the results of the classifiers and even severely decreased the performance of the models. This might be due to discarding too many images from the dataset due to overlapping with other classes, this shrinks the train dataset to a point where every class has an equal number of instances while there is also a clear imbalance in the validation set. Furthermore, the amount of datapoints to



(A) Image of a bicycle from the dataset without PCA.

(B) Image of a bicycle from the dataset with PCA on 700 components.

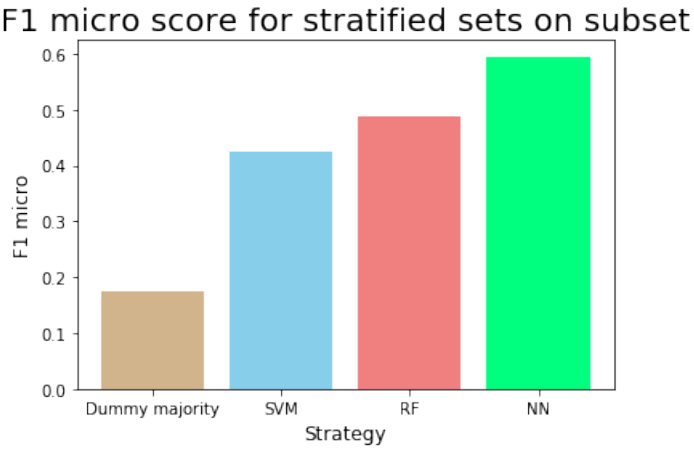


FIGURE 16: F1 micro for the class subset.

train on is severely decreased which most likely contributes to the loss in performance as can be seen in tables 7 & 9 which are located in the appendix.

Another surprise was the SMOTETOMEK sampling method were no significant improvements were found except for the random forest classifier. With the oversampling done with SMOTE and after this the removal of instances which overlapped with other classes by using TOMEK links, the expectation was that there would be a significant improvement of the results for all of the classifiers. As can be seen in tables 6, 14, 13 & 15, the Neural Network has the best overall performance and this is when no sampling methods are being used. The SVM's results do not fluctuate much between different methods, the random forest has the most interesting behavior where the results improve when SMOTETOMEK is used as would be expected. Even though the results might not improve as is expected when using sampling methods on this dataset, it might be that the sampling methods do improve the performance when using the entire COCO dataset. The best strategies which were found while testing on the class subset can be seen in figure 17 and table 5, the strategies are sorted on F1 micro performance. The best classifier is the neural network without sampling methods which is quite unexpected

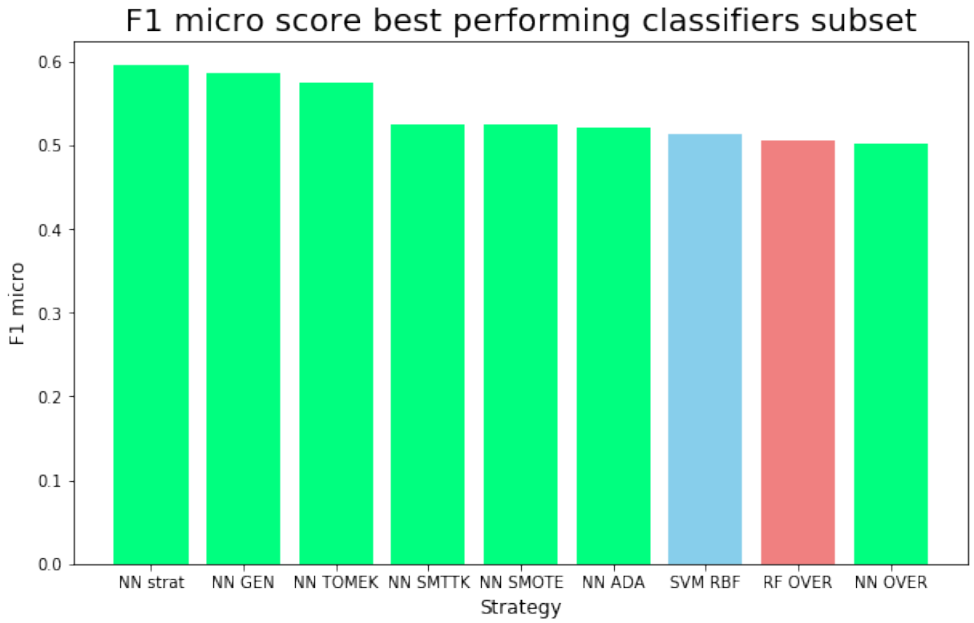


FIGURE 17: F1 micro for best strategies found for the class subset.

TABLE 5: Class subset classifier micro F1 score and standard deviation (n=5).

Classifier	F1 micro	SD
SVM	0.424	0.000
RF	0.486	0.002
NN	0.581	0.013

and interesting, the micro and macro ROC curves of this classifier can be found in figure 18.

5.3 Results COCO dataset

When comparing the results of the SVM, RF and NN with iterative stratification as can be seen in tables 16, 18 & 17 to the results of the dummy classifier as can be seen in table 25, all the metrics of the SVM, RF and NN have a higher score for every metric. This is for accuracy, precision, recall and the F1 score for both separate classes and overall. The only exception for this is when the dummy classifier always picks the majority class as predicted class, which is the person class. In this scenario both the accuracy and weighted recall of the dummy classifier are higher for smaller subsets of the COCO dataset as most of the instances in both the train and test set are persons with about 30.6%. When always picking the majority class, it seems quite logical that these metrics perform better for the dummy classifier. This is because of the data set being extremely skewed towards the person class, as the number of instances in this class is multiple times larger than any other class present in the data set. When looking at the results of different train and test set sizes for the SVM, RF and NN using iterative stratification, the metrics keep improving when the train and test sets are a bigger subset of the data set. This is to be expected as a bigger training data set ensures that the model is trained with more instances per class, which in turn provides more diverse within-class examples of the different classes. The extra data limits overfitting on the training set and thus yielding better results on the test set.

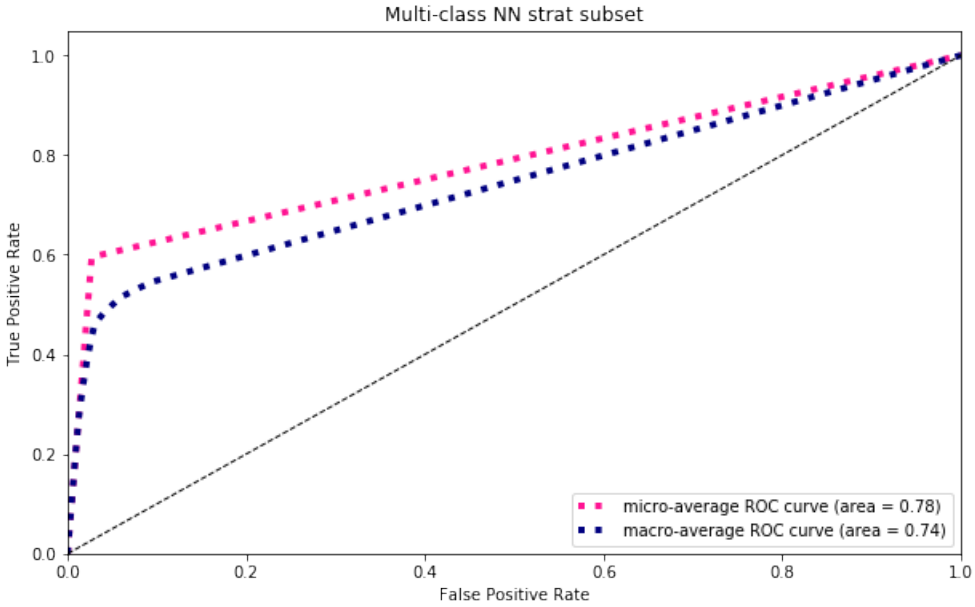


FIGURE 18: Micro and macro ROC curves of the Neural network without sampling methods, which is the best performing classifier on the subset

When comparing the performance of the SVM and NN in tables 16 & 18 against the performance of the random forest using the same iterative stratification on the image sets as can be seen in table 17, the random forest does perform better on the smaller sample sizes where all the metrics score higher than those of the SVM and NN. Whenever the sample size gets sufficiently large, around 5-7.5%, the SVM and NN start outperforming the RF in all the metrics.

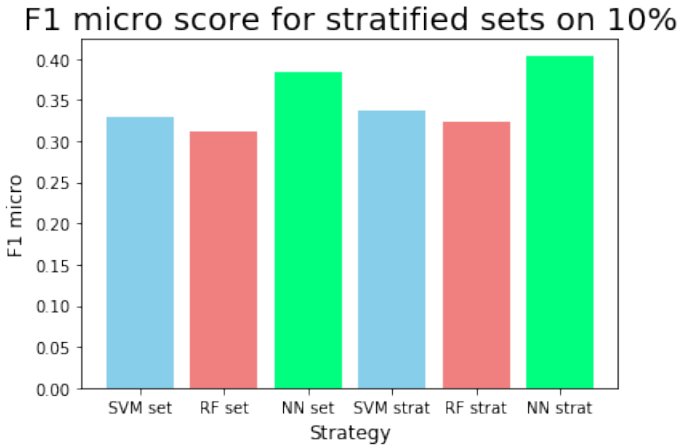


FIGURE 19: F1 micro for stratified methods.

Interestingly enough the same pattern is found when stratifying the data set on labels instead of label sets in an image. For example, the classifiers outperform the dummy classifier on every metric for almost all the strategies. The exception for this is once again the most frequent strategy, also known as the majority class strategy. As explained in the previous section this is to be expected because the dataset is extremely unbalanced which makes picking the majority class, in this case person, quite an effective

strategy. Once again, the accuracy and recall are better for the majority decision strategy compared to the NN, SVM and RF for the smaller data sets (1-2%) and the non-dummy classifiers perform better for all the other metrics. This is interestingly enough with the exception of the SVM using an RBF kernel, this classifier already outperforms the dummy classifier on the 1% data set and is effectively the best classifier for small data sets out of all the tested classifiers as can be seen in table 26. This is likely due to the difference in the kernel, the linear kernel which tries to separate the data linearly to make the hyperplane decision boundaries between classes. The RBF kernel uses a non-linear kernel to make this decision boundary which is more flexible and will most likely fit the data better. In general, the linear kernels are faster than the RBF kernel but the trade-off is that the accuracy is generally lower. When the datasets get sufficiently large enough, all of the classifiers start outperforming the dummy classifier for every metric. The RF outperforms the SVM and NN once again for small data sets (1-2%) on both accuracy and recall but the SVM and NN have better precision and F1 scores.

If bigger sample sizes are used (7.5-10%), the NN outperforms both the SVM and RF on all the metrics. Once again, the SVM and RF tend to overfit too much on the person class where the recall of the SVM and RF is 0.93 and 1 respectively. One of the things that really stands out is that the RF classifier barely improves when the data set is enlarged, the only metric that improves a lot is the precision of the classifier. All the other metrics which are measured only improved with minimal margins when looking at the difference between 1% and 10% of the data set. The SVM improves a lot when looking at the difference between 1% and 10%, all the metrics improved a lot mainly the accuracy, precision and recall. As is to be expected the results of the NN improved a lot when the data set was enlarged, as neural nets need many samples to perform well.

When comparing the single label stratification against the image set stratification it seems that the NN performs better when it is fed data which is stratified without taking the image sets into consideration. On one hand this is quite logical because no biased is introduced by only sampling image sets instead of freely sampling the whole data set, on the other hand there might have been samples which are similar in the same image set which could have helped the classifier to learn the classes. It seems that the model overfits a bit more when image sets are used, this is true for both small data sets (1%) as for bigger data sets (10%).

It is interesting to note that the classification results of the single label stratification were better than the results of the image set classification. This might be because of the bias that is introduced when sampling image sets. When multiple instances of the same class appear on one image, he generalizability of the classifier is decreased as the instances of these classes are most likely not as different as when sampling single labels. To conclude, the best techniques to predict classes in the dataset which was found in this research is the Neural Network with stratification. The Neural Network being the best classifier is not a shocking result as the neural network is more flexible than the SVM and RF especially when feature extractors are used for both RF and SVM while the NN has convolutional layers to extract the features that is uses to learn the different classes.

The results that were obtained from this experiment are interesting as can be seen in tables 23, 22 & 24. The TOMEK links did not improve the results of the models when testing sampling techniques on the class subset. The expected results was that the TOMEK links removing instances from classes which are overlapping according to the nearest neighbor pairs of the same class would improve the classification rate

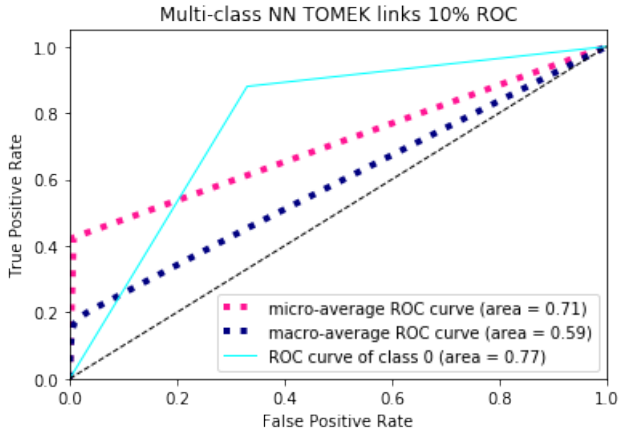


FIGURE 20: NN with Tomek Links 10% of the dataset ROC.

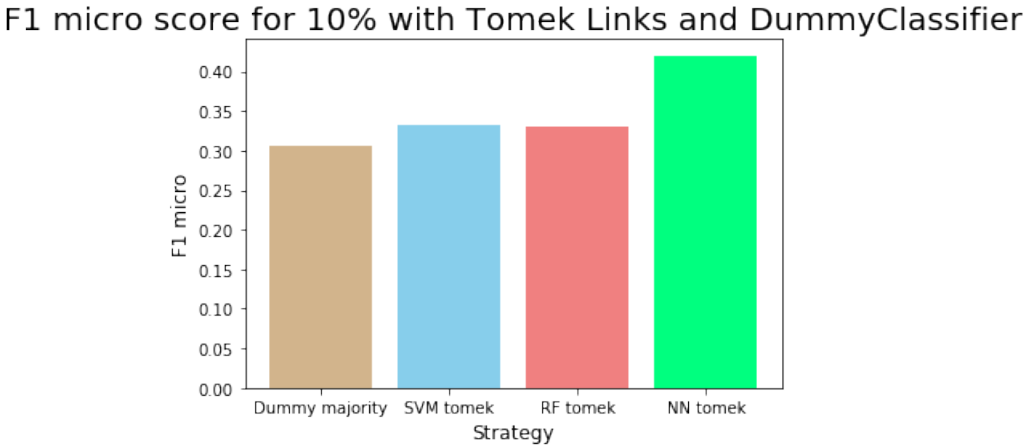


FIGURE 21: F1 micro for Tomek Links and dummy classifier.

significantly. There are also more samples in the subset which might be on a class border or a bad representation of the class, the positive effects would be increased if there are more samples to undersample. After sampling, the train set is left with instances of classes which are not overlapping with other classes thus improving classification results of the models. This effect was also initially expected with TOMEK links on the class subset. Examples of the code that was used to calculate the TOMEK links on the COCO dataset can be found in the appendix in listings 1 & 2.

Undersampling techniques such as EditedNearestNeighbors (ENN) and AllKNN undersampled some of the minority classes to the point where some of these minority classes were omitted from the train set. Because of this the ENN and AllKNN are not included in the results of the complete dataset. Furthermore, most sampling techniques try to balance the dataset to a point where most classes are in the same order of magnitude. This means that all of the classes are oversampled to the point where the number of instances in all classes is almost equal to the number of instances in the person class. As mentioned in earlier section, the person class makes up about 30% of the original dataset which means that all other classes are oversampled until their number of instances is 30% of the size of the original dataset.

While the overall results of the SVM are about equal or sometimes slightly better than the results of the RF when the dataset is big enough (7.5-10%), the RF outperforms by quite a margin the SVM when the dataset is small (1-3%) as can be seen in tables

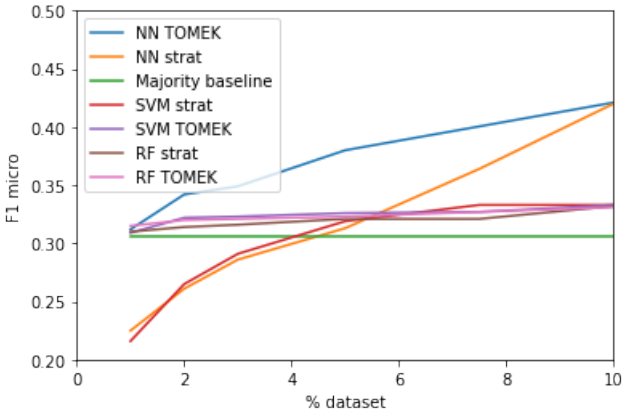


FIGURE 22: Strategies and their micro F1 score for percentages of the COCO dataset used.

16, 17, 19, 20, 23 & 22. This might have multiple reasons: The RF has an extra feature compared to the SVM because the SVM fit converged very slowly or did not converge at all when this feature, MLBP, was used for the SVM. This feature might help classifying the test set when the train set is small in the sense that extra information is provided to the RF classifier. When looking at the results per class it seems that the SVM mainly focused on classifying the majority class correctly whereas the RF tried to optimize the performance on multiple classes.

Looking at larger subsets of the data, it seems that the SVM tried to diversify and tried to learn more classes as almost all the classes have metric scores higher than 0 if the class sizes is not too small (<250). As person is by far the majority class, this class might be a good choice for the classifier to optimize. The random forest scored better on some other classes such as pizza, tv, clock and book. This might again be because of the MLBP feature which was added to the RF classifier. It seems that the RF is once again prioritizing the majority class to get a better classification rate as the recall rate on the person class is once again 1 for the RF whereas it seems to ignore most other classes besides to the traffic light, bowl and dining table classes. It is quite interesting because these classes are not even near as big as the person class, this might mean that the features for these classes are distinct enough from the other classes, even for small data sets. Even more interesting is that the SVM does not score near as good on these classes. According to multiple sources the SVM generally outperforms the RF classifier if an SVM can be used for the domain, this also seems to be the case for these experiments. The RF was faster and easier to use than the SVM, which makes the RF a reliable, easy to use and fast classifier when first inspecting the data and finding out which features are useful for classification.

As expected the neural network's performance is not that good when the sample size is small as can be seen in tables 18,21 & 24 located in the appendix. The performance of the neural network is comparable to the performance of the SVM. The accuracy is almost identical for these two classifiers for small datasets, the precision is a bit worse for the neural network. However, the F1 score and recall are significantly better for the neural network than the SVM. When comparing the results of the Neural Network against the results of the RF on small datasets, the RF is better for both accuracy and recall but the NN is better in both precision and F1 by a large margin already. This is not something that is to be expected as NN's tend to overfit on small sample sizes and might not have enough samples to correctly learn and predict the classes. However, the

overall results of the neural network are to be expected as the metrics scores are not that high, which would be normal for small datasets. If we are looking at the bigger sample sizes however, the neural network outperforms the SVM and RF by a significant margin. This was to be expected as the neural network outperformed the RF and SVM by a large margin in the class subset. All of the metrics, accuracy, precision, recall, F1, are significantly better for the Neural network. This might be because of the nature of neural networks where the neural network tries to learn features which are good for classifying the different classes.

When taking the individual scores into account it seems that the neural network didn't overfit as much because the precision of the person class is 54.1% for the neural network, 36.6% for the SVM and 33.2% for the RF as can be seen in tables 28, 29 & 30. This might show that Neural networks are less prone to picking the majority class too often even if the data set is skewed. The recall is lower for the NN is lower than that of the SVM and RF when looking at the person class, this might show that the neural network tries to diversify the classifications whereas the SVM and RF are still prone to picking the majority class more often. One of the reasons why the RF and SVM have a recall of close to 1 for the person class is that one of the features used is HOG, which is used in many applications to classify or detect persons. Another reason might be that the features that are used work well on objects such as persons, whereas the neural network generates features and does not get the pre-calculated features. This might make the NN better in many scenarios because of the feature extraction and the flexibility of the classifier if the data set is big enough and not too imbalanced.

5.3.1 Validation

The class subset which was used to optimize the models was small enough that sampling was not needed and the whole train and validation/test set could be used to train and test the models. This means that the collected results for the class subset are fairly accurate and unbiased as there is no need to use techniques such as k-fold cross validation. This is because all of the instances belonging to the class subset are used to train and validate the classifiers. The best performing classifiers were trained five times to establish a mean and standard deviation for the metrics of these models. The results which were gathered from the class subset are used to test and validate which sampling techniques and models work well on this particular set. The results gathered from this validation were used to pick an architecture for the neural network and to pick the sampling methods that performed well to use for the entire COCO dataset. It should be taken into account that basing sampling algorithms, hyperparameters and architectures of classifiers on results of the models trained on the class subset is a somewhat biased but relatively good starting point for classifying the entire dataset as can be seen in the results.

To validate the models which were trained, a test set is sampled from the train set. The data set is large enough that both the train and test set can be sampled from the original train set without contaminating the classifier as every classifier uses the same train and test sets for every experiment. When using the same 1-10% of instances in the dataset, the classifiers can at least be compared as they have been trained and tested on the same dataset. This does not mean that there is no need for cross validation on different folds when aiming to extensively validate the classifier. Every sampling method is used on both the train and test set with the exception of the over and undersampling techniques. For these sampling techniques, the standard stratification was used to validate the performance of the model. Generating both the train and test

set by using the same sampling technique will give an insight in techniques which might be good for both training and testing models on a data set.

6 Discussion

The goal of this thesis was to research the impact of different sampling techniques on the performance of widely used classifiers for supervised learning, more specifically object recognition on images. After having tested sampling methods such as random sampling, iterative image set sampling, stratified sampling and several under and over sampling techniques on supervised learning techniques such as Neural Networks, SVM's and Random Forests, results have been gathered and research questions can be answered. There is still a lot to improve on in this research such as further optimizing the models, testing more pre-processing techniques, researching these sampling techniques on other data sets and classifiers.

The results which are obtained during this research are satisfactory and the classifiers outperform the baseline classifiers on most of the metrics, except for maybe accuracy when the baseline is always picking the majority class when the classifiers do not have much data to train on. The baselines give an insight in how well the classifiers actually perform as it makes it easier to interpret the results from the trained models with respect to the result of the baseline classifiers. When looking at the results of the different classes, it is apparent that the person class is one of the easiest classes to learn for the classifier. This is very likely due to the number of instances in the person class in proportion to the entire dataset.

There is a clear distinction between using the entire data set and the class subset in the performance metrics, this at least shows the potential of all the different models and sampling methods which were used in this research. The results do give us an insight in the relations between data set size, sampling technique, classifier and the performance of this particular model. This is exactly the reason why the earlier named benchmark is so important.

Even though a large part of the data set was not used, conclusions can still be drawn because the baseline can always be compared to the actual performance of the model. The metrics which are used to measure the performance of the model are the F1 micro and macro scores as these metrics give an insight in the performance on imbalanced data sets. The micro score is dependent on the total amount of correctly predicted incorrectly predicted classes, whereas the macro score calculates a F1 metric for every class and averages this metric by adding all of the macro scores from other classes and dividing by the number of classes (Z. Zheng, Wu, and Srihari, 2004).

As the COCO set had one class made up about $\frac{1}{3}$ of the data set, the macro scores of classes that do not have many instances in this set have a fairly unstable macro score. This is still important to note because the importance of the classes in the data set is not known. The classification could have been more stable if the classes with too little instances would have been omitted from the train and test set. However, this raises the question whether bias is introduced into the classification process by arbitrarily selecting and omitting data and could be a possible interesting topic for further research. Because the research was focused on classifying imbalanced data sets, the choice was made to keep the data set intact with all of the classes present.

The results also show that the Neural Networks perform the best on this data particular data set and possibly performs better overall on imbalanced data sets. This might be due to the convolutional layers, weight updates and deep learning aspects of the neural network. As the neural network does not need pre-extracted features to be fed to the machine learning algorithm but extracts its own features based on interesting aspects of the class in the images, it might extract features that are performing well on the data set making it more flexible, instead of extracting features that perform well

overall which leads to overfitting. It can not be verified nor falsified whether the Neural Network overfitted on this data set or not as the classifiers were not tested on other data sets, the generalizability of the classifiers trained on this data set is unknown. Some of the results such as the results of the class subset were validated while the results of a percentage of the entire COCO dataset were not cross-validated. This may have an effect on the observations and conclusions which are made in this thesis.

When training the neural network using the ImageDataGenerator in the TensorFlow library with featurewise centering and ZCA whitening, the results were very good (72% validation accuracy). The validation set was passed to the neural network using a special validation_data parameter while fitting on the train set. When the validation set was used to as a set of images that had to be predicted and the results were logged in a classification report, the results were really low. Even when predicting the train set the results in the classification report were quite bad. This was quite confusing and a possible explanation is a bug in the Keras ImageDataGenerator for ZCA and featurewise centering.

Finally, the oversampling techniques might boost the performance of the classifiers which were used in this research a lot. When oversampling on an imbalanced data set where the majority class makes up about 30% of the data set, the amount of oversampling on the other classes would simply be too much to calculate with the limitations in the setup if this thesis. Also, the combination of over and undersampling would be interesting to research as it might yield better results than just undersampling or oversampling. This would make it interesting to research oversampling and oversampling + undersampling for this problem in the future. A short paragraph in the future research will be devoted to this topic as it is likely that these methods will greatly boost the performance of the classification models.

6.1 Further research

While conducting the research there were many questions, solutions, insights and doubts. These questions, insights and doubts will be discussed in this section to give an overview of interesting topics which could not be elaborated on during this research. To further improve upon classifiers which have to predict classes in an imbalanced and to fully understand the dynamics and implications of the different sampling methods and classifiers on the results more research is needed. Unfortunately, some of the experiments which were needed to further elaborate on these implications were not possible to conduct in the scope or time frame of this research.

One of the major points of improvement of this research is that results have to be cross-validated, especially when using the folds of the COCO dataset to train the classifiers. Using stratified cross-validation reduces the bias and variance of the models significantly Kohavi et al., 1995. The results which have been obtained during this research give insight into the domain of image classification on imbalanced data sets but using cross-validation would make these results more robust and would allow for stronger conclusions to be made from the results and data when looking at the results of the 1-10% of the COCO dataset as train set. The results which were obtained during this research can still be used to draw conclusions, but to further elaborate on these conclusions and make stronger arguments more research is needed. Furthermore, the pre-processing and feature extraction might benefit from further research as there are a plethora options available for these steps in the machine learning process. Feature extraction methods such as SIFT, ORB, SURF, shape-based features, edge detectors and texture-based features (Rublee et al., 2011; Nixon and Aguado, 2012).

Another interesting subject would be how well the models generalize from this data set to other data sets. Generalizability is a big problem in the field of machine learning as models often perform well on the data set on which they are trained but when tested on other data sets, the results might not be satisfactory. Data sets such as ImageNet, Google's Open Images or even the CIFAR-10 data set might be used to further train the model in order to generalize the model and use it in multiple scenarios and data sets. Training the classifiers on a data set that is not as imbalanced would most likely benefit the performance and generalizability of the models. However, it is important to note that this data set has occluded and cluttered images instead of unobstructed images which are easier for machine learning algorithms to classify (Lin et al., 2014).

One way to achieve better performance might be to research the feature engineering and pre-processing techniques further so that more meaningful features can be extracted from the images. More information will most likely result in better classification results. Spending more time on feature engineering might also provide a better insight into the problem which will help with the optimization of the models. Feature extraction methods such as SIFT and SURF might yield better results and different ways of combining or selecting the most meaningful features might boost the performance of the classifiers. Another way to boost the performance of the neural networks in particular is to potentially make the network 'deeper'. Deeper neural networks might extract more useful features which cannot be extracted by more shallow neural networks. Another way to achieve better performance is to create a classifier using one-vs-one (OVO) classification strategy to make classifiers which are more precise in prediction a single class. Whether this actually improves the results is unclear, but it might be interesting to research the performance of OVO classifiers instead of one-vs-rest (OVR) classifiers on imbalanced datasets. The downside of using ovo classifiers is that it is quite computationally expensive to compute all of the separate classifiers for classes.

One interesting observation for the SVM implementation that is used in this thesis is that the kernel function which was used is linear. This has the advantage that this kernel function is faster and computationally less expensive to compute than some of the other functions. As can be seen in Figures 19 & 26 the results of the SVM experiments with the RBF kernel seem to outperform the stratified linear SVM by a large margin. It is hard to say that the RBF always outperforms the linear SVM but it seems that the RBF kernel is more flexible in fitting to the data. One downside of using the RBF kernel is that it is computationally very expensive as was mentioned in the experimental setup in the SVM subsection. This is due to the kernel trick that is used in SVMs not scaling well when the number of training samples or features in the feature space is increased. It would be interesting to see how the SVM with an RBF kernel would perform when sufficient computational resources are available to calculate this classifier on a big data set.

The effect of other sampling methods would also be an interesting topic for further research on either the COCO data set or imbalanced data sets in general. Due to the scope and time limitations it was not possible to test all of the available sampling algorithms. Most of the oversampling methods are computationally expensive, this is why only undersampling methods were used in these experiments. When oversampling, the computational cost and the memory that is needed to upsample the minority classes in the data set is such as vast amount. This is because the person class makes up 30% of the data set, this means that all of the other classes have to be upsampled by a large amount which uses a huge amount of memory and processing power which is not feasible on this data set during this research. There might be sampling algorithms or combinations of algorithms such as ADASYN, SMOTE or SMOTETOMEK which

might improve the results of the models on this data set even further (He et al., 2008; Barua, Islam, and Murase, 2011). Furthermore, some of the sampling methods such as Edited Nearest Neighbors (ENN) reduced the data set to approximately $\frac{1}{8}$ of the original subset of the data and might even omit entire classes from the train set. This greatly reduces the data available for the classifiers to train on which naturally yields worse results. This technique might work better when there are not as much classes present in the dataset and when the class imbalance itself is not that extreme.

Finally, there were several assumptions made and concerns when doing this research which might need researched and clarified in the future. Firstly, the amount of data which was used is relatively low compared to the full data set. This has multiple reasons, the first one is that it is interesting to research if there are sampling techniques which can optimize the results when the amount of data is relatively small. The second reason is more practical in the sense that the computational resources to compute all of these models and results is unfeasible in the time for this thesis. As mentioned earlier there are 82783 images in the data set which have a combined size of 12.5GB can be further divided into 604907 sub images. For all of these images' features have to be extracted, pre-processing has to be done and the models have to be trained. This process takes up quite some time, especially when there are many sampling methods for different models and subset sizes.

Other classification algorithms such as KNN or meta models which, can be constructed by combining multiple models, could be used to construct a more optimized model with better results. The SVM, Random Forest and Neural Networks were used as these algorithms seem to be widely used in the field of image classification but that does not mean that these algorithms are always best suited for an image classification problem.

7 Conclusion

The goal of this research was to analyze which sampling and classification methods perform well when working with an imbalanced data set. First of all, pre-processing is an important step in the process of machine learning as it can reduce overfitting on the train set with techniques such as gaussian blurring, data augmentation and one-hot encoding which standardize the data. The performance metric which was chosen to measure the performance of the classifiers is the F1 micro score and in some cases the weighted F1 score. These metrics give an insight in how the precision and recall relate to eachother but the F1 micro also represents the results that the classifiers achieve on imbalanced datasets better than the macro scores on this dataset as there is no single important class. Another reason is that the F1 score gives a more complete understanding of the classifier in contrast to for example the accuracy score.

It is not possible to say which sampling methods perform the best in general and what the optimal sampling rate would be as these choices are likely to be dependent on the domain (G. M. Weiss, 2004). From this research it can be inferred that stratifying data on single labels is most of the time a good way to subset data for training a model. Furthermore, techniques such as TOMEK links, ADASYN and data augmentation specifically for neural networks improve on the performance of the classifiers in some scenarios. The problem with picking a sampling method is that the performance of classifiers on the sets that are sampled with these techniques are very dataset dependent which makes picking an optimal sampling method very hard. Furthermore, this problem makes it impossible to pick a single sampling method that always yields good results. Techniques such as ENN, SMOTETOMEK and random sampling did not yield good results when conducting this research, this does not mean that these sampling methods are bad as these techniques might yield better results on other datasets.

Furthermore, the neural networks seem to outperform the SVMs and RFs when the data set for training is sufficiently large, this may mean that neural networks are the best choice when working with imbalanced, cluttered and occluded data sets such as the COCO data set used for this research. This might have to do with the neural network being more flexible as it can extract its own features, depending on what is important in the dataset instead of having to rely on feature extractors which always look for the same patterns. On the other hand, classifiers such as the RF and SVM are easier to interpret and explain, which might have value when trying to better understand the current domain. The best performing setup is a neural network with only the stratified data for the class subset and the neural network with stratified labels and possibly TOMEK links when using the 1-10% of the entire COCO dataset as the results for these setups where similar. The sampling methods did not always improve the results, which might suggest that the error rate of the minority samples is very high. One explanation for this might be that the learning problem may have another origin, such as the within-class distribution of data for every class. Another explanation might be that other methods such as manually augmenting data may in some cases yield better results than undersampling and oversampling methods due to sampling methods introducing within-class outliers or instances which are close to the border of another class. The final way to improve on the results is to use more data, but in order to use more data the computational resources have to be increasingly powerful.

As can be inferred from the results, all of the classifiers perform better than the baseline even without under or oversampling methods when enough data is available. This shows that these classifiers are able to improve on the baseline classifier which only picked the majority class. Even though the performance of these models is not very

good on the data set with the small subset of data which was used to train the models, these methods do show potential. Understanding the problem of classifying imbalanced data sets and which methods work well in this particular domain have increased due to this research. Further optimization, combining sampling methods, ZCA whitening, using more feature extractors and the ability to feed more data to the models will greatly improve the performance for all of the models. Techniques such as combining classifiers to create a better classifier and creating deeper neural networks might be used to better results.

The bias in this research is limited, as the stratification is a good method to split datasets while keeping bias and variance limited Kohavi et al., 1995. To further diminish the effects of overfitting and reduce the bias, techniques such as sampling, gaussian blurring, adding noise and data augmentation were used. The only bias introduced was when validating on a class subset from the COCO dataset as these classes were a subset of the dataset which does not represent the entire dataset. However, this was only used for optimizing and validating the models. The variance that was introduced could have been reduced by cross-validating the results on the entire COCO dataset using stratified k-folds cross validation Kohavi et al., 1995. Another thing to keep in mind when using folds is that there is variance in the actual samples, this means that the next to cross-validating the folds should be resampled. This thesis shows that machine learning can be used in an effective manner when classifying imbalanced data sets and that machine learning can be used to classify cluttered and occluded objects. It is important to note that the pre-processing, sampling and the classification techniques have a significant effect on the performance of the model and commonly used techniques should be explored before picking and committing to certain techniques.

Finally, to answer the research question: What is the influence of sampling methods for training a classifier to classify images in an imbalanced multi-class data set? The conclusion can be made that the sampling method which yields optimal results is dependent on the classifier, the domain and the dataset. This means that some exploratory research has to be conducted before picking a sampling method. In order to find the best sampling method for the dataset, establish a baseline with performance metrics for the current problem. When the baseline results are collected, implement and optimize the sampling methods and compare these results to the baseline classifier to find the most optimal sampling method for the domain. As all imbalanced datasets are different in terms of data, domain and the proportions of classes, there is not one single technique or a set of techniques which will always yield optimal results. In conclusion, all of the classifiers which were used during this research are sensitive to class imbalance. The effect of class imbalance on classifiers should not be overlooked when working with imbalanced datasets.

8 Reflection

With this thesis I have shown that machine learning on imbalanced data sets can be improved upon by using the right sampling techniques. The models that I constructed during this research were able to classify the objects on images better than the baseline classifier did. During this research I focused on data processing, feature extraction, sampling methods and optimization of the models. When I started my thesis, I had an idea of how data analytics works but I did not have much experience with solving data science problems which were as complex as this project. I had to learn techniques such as TensorFlow, Azure with the machine learning workspace, imblearn and research possible approaches to tackle this problem which did cost me quite some time. I have learned a lot on the subjects of statistics, machine learning, data analytics and conducting research. Furthermore, I have also learned how to run a cloud machine learning service in Azure on which jobs for training models can be scheduled. Another important skill which I learned during this research is learning which metrics to pick and how to interpret these metrics to draw conclusions. I have learned a lot about how and why the classifiers work and in which domains these classifiers perform well.

There is still room for improvement in training and optimizing these models as the domain and number of possibilities in machine learning are quite big. I have shown that skills which I have acquired during the master Artificial Intelligence were sufficient to conduct these experiments and create relatively successful models for the amount of data which was used and the limitations of the research. The most important thing I think I have shown during this thesis is that I'm able to evaluate and analyze my experiments in order to improve upon the results and that I'm not satisfied with only the results but I also need to be able to explanation why the results have a certain value. My research skills and writing skills also improved a lot during the course of this thesis, looking back I'm very grateful for this opportunity from both my supervisors at Utrecht University and Avanade. I hope that this research can have a positive impact on the classification of imbalanced data sets at Avanade. Even though I have learned a lot from conducting this research for my thesis, there is still knowledge to be gained and I hope to expand further on my knowledge regarding this topic in the future.

References

[1] Martın Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: a system for large-scale machine learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[2] Ali Ismail Awad and Mahmoud Hassaballah. "Image feature detectors and descriptors". In: *Studies in Computational Intelligence. Springer International Publishing, Cham* (2016).

[3] Will Badr. *Having an Imbalanced Dataset? Here Is How You Can Fix It.* https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb. Accessed: 2019-05-20.

[4] Sukarna Barua, Md Monirul Islam, and Kazuyuki Murase. "A novel synthetic minority oversampling technique for imbalanced data set learning". In: *International Conference on Neural Information Processing*. Springer. 2011, pp. 735–744.

[5] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. "A study of the behavior of several methods for balancing machine learning training data". In: *ACM SIGKDD explorations newsletter* 6.1 (2004), pp. 20–29.

[6] Indranil Bose and Radha K Mahapatra. "Business data mining—a machine learning perspective". In: *Information & management* 39.3 (2001), pp. 211–225.

[7] Matthew R Boutell, Jiebo Luo, Xipeng Shen, and Christopher M Brown. "Learning multi-label scene classification". In: *Pattern recognition* 37.9 (2004), pp. 1757–1771.

[8] Sing-Tze Bow. *Pattern recognition and image preprocessing*. Marcel Dekker New York, 2002.

[9] Gary Bradski and Adrian Kaehler. "OpenCV". In: *Dr. Dobb's journal of software tools* 3 (2000).

[10] Paula Branco, Luis Torgo, and Rita P Ribeiro. "A survey of predictive modeling on imbalanced domains". In: *ACM Computing Surveys (CSUR)* 49.2 (2016), p. 31.

[11] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[12] Gustavo Camps-Valls and Lorenzo Bruzzone. "Kernel-based methods for hyperspectral image classification". In: *IEEE Transactions on Geoscience and Remote Sensing* 43.6 (2005), pp. 1351–1362.

[13] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[14] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. "Special issue on learning from imbalanced data sets". In: *ACM Sigkdd Explorations Newsletter* 6.1 (2004), pp. 1–6.

[15] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 886–893.

[16] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From data mining to knowledge discovery in databases". In: *AI magazine* 17.3 (1996), p. 37.

[17] Giles M Foody and Ajay Mathur. "A relative evaluation of multiclass image classification by support vector machines". In: *IEEE Transactions on geoscience and remote sensing* 42.6 (2004), pp. 1335–1343.

[18] Mark Andrew Hall. "Correlation-based feature selection for machine learning". In: (1999).

[19] David Hand and Peter Christen. "A note on using the F-measure for evaluating record linkage algorithms". In: *Statistics and Computing* 28.3 (2018), pp. 539–547.

[20] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pp. 1322–1328.

[21] James J Heckman. *Sample selection bias as a specification error (with an application to the estimation of labor supply functions)*. 1977.

[22] Sepp Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.

[23] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[24] Ron Kohavi et al. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *Ijcai*. Vol. 14. 2. Montreal, Canada. 1995, pp. 1137–1145.

[25] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. "Supervised machine learning: A review of classification techniques". In: *Emerging artificial intelligence applications in computer engineering* 160 (2007), pp. 3–24.

[26] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. "Handling imbalanced datasets: A review". In: *GESTS International Transactions on Computer Science and Engineering* 30.1 (2006), pp. 25–36.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[28] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[29] Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 559–563.

[30] Andy Liaw, Matthew Wiener, et al. "Classification and regression by randomForest". In: *R news* 2.3 (2002), pp. 18–22.

[31] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[32] Charles X Ling and Chenghui Li. "Data mining for direct marketing: Problems and solutions." In: *Kdd*. Vol. 98. 1998, pp. 73–79.

[33] Yi Liu and Yuan F Zheng. "One-against-all multi-class SVM classification using reliability measures". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. IEEE. 2005, pp. 849–854.

[34] Marcus A Maloof. "Learning when data sets are imbalanced and when costs are unequal and unknown". In: *ICML-2003 workshop on learning from imbalanced data sets II.* Vol. 2. 2003, pp. 2–1.

[35] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. "Introduction to information retrieval". In: *Natural Language Engineering* 16.1 (2010), pp. 100–103.

[36] Lev Manovich. "Trending: The promises and the challenges of big social data". In: *Debates in the digital humanities* 2 (2011), pp. 460–475.

[37] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. "Stacked convolutional auto-encoders for hierarchical feature extraction". In: *International Conference on Artificial Neural Networks.* Springer. 2011, pp. 52–59.

[38] Mor Naaman. "Social multimedia: highlighting opportunities for search and mining of multimedia data in social media applications". In: *Multimedia Tools and Applications* 56.1 (2012), pp. 9–34.

[39] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10).* 2010, pp. 807–814.

[40] Mark Nixon and Alberto S Aguado. *Feature extraction and image processing for computer vision.* Academic Press, 2012.

[41] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[42] David Martin Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: (2011).

[43] Ronaldo C Prati, Gustavo EAPA Batista, and Maria Carolina Monard. "Data mining with imbalanced class distributions: concepts and methods." In: *IICAI.* 2009, pp. 359–376.

[44] Foster Provost. "Machine learning from imbalanced data sets 101". In: *Proceedings of the AAAI'2000 workshop on imbalanced data sets.* Vol. 68. AAAI Press. 2000, pp. 1–3.

[45] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R Bradski. "ORB: An efficient alternative to SIFT or SURF." In: *ICCV.* Vol. 11. 1. Citeseer. 2011, p. 2.

[46] *scikit-learn.org SVC.* https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Accessed: 2019-03-11.

[47] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. "On the stratification of multi-label data". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.* Springer. 2011, pp. 145–158.

[48] Engin Tola, Vincent Lepetit, and Pascal Fua. "Daisy: An efficient dense descriptor applied to wide-baseline stereo". In: *IEEE transactions on pattern analysis and machine intelligence* 32.5 (2009), pp. 815–830.

[49] Grigorios Tsoumakas and Ioannis Katakis. "Multi-label classification: An overview". In: *International Journal of Data Warehousing and Mining (IJDWM)* 3.3 (2007), pp. 1–13.

[50] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. "scikit-image: image processing in Python". In: *PeerJ* 2 (2014), e453.

[51] Gary M Weiss. "Mining with rarity: a unifying framework". In: *ACM Sigkdd Explorations Newsletter* 6.1 (2004), pp. 7–19.

[52] Gary M Weiss and Foster Provost. "Learning when training data are costly: The effect of class distribution on tree induction". In: *Journal of artificial intelligence research* 19 (2003), pp. 315–354.

[53] Dennis L Wilson. "Asymptotic properties of nearest neighbor rules using edited data". In: *IEEE Transactions on Systems, Man, and Cybernetics* 3 (1972), pp. 408–421.

[54] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[55] Bianca Zadrozny and Charles Elkan. "Learning and making decisions when costs and probabilities are both unknown". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 204–213.

[56] Lun Zhang, Rufeng Chu, Shiming Xiang, Shengcai Liao, and Stan Z Li. "Face detection based on multi-block lbp representation". In: *International Conference on Biometrics*. Springer. 2007, pp. 11–18.

[57] Zhaohui Zheng, Xiaoyun Wu, and Rohini Srihari. "Feature selection for text categorization on imbalanced data". In: *ACM Sigkdd Explorations Newsletter* 6.1 (2004), pp. 80–89.

[58] Zhi-Hua Zhou and Min-Ling Zhang. "Multi-instance multi-label learning with application to scene classification". In: *Advances in neural information processing systems*. 2007, pp. 1609–1616.

[59] Qiang Zhu, Mei-Chen Yeh, Kwang-Ting Cheng, and Shai Avidan. "Fast human detection using a cascade of histograms of oriented gradients". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE. 2006, pp. 1491–1498.

Appendix

The tables which are represented here contain the Accuracy score, which is the same as the precision micro, recall micro and F1 micro in a multi-class setting. Furthermore it contains the macro scores of the earlier mentioned metrics denoted by m. Finally, the columns with a metric name followed by a w represent the weighted score of the respective metrics.

Table 6: class subset classifier results with standard deviation in parenthesis (n=5).

Classifier	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
SVM	0.424 (0.000)	0.340 (0.000)	0.425 (0.000)	0.347 (0.000)	0.390 (0.000)	0.427 (0.000)
RF	0.486 (0.002)	0.396 (0.002)	0.555 (0.003)	0.378 (0.002)	0.464 (0.002)	0.521(0.003)
NN	0.581 (0.013)	0.502 (0.009)	0.532 (0.006)	0.500 (0.010)	0.575 (0.013)	0.584(0.009)

Table 7: class subset random undersampling classifier results.

Classifier	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
SVM	0.289	0.266	0.319	0.267	0.283	0.314
RF	0.384	0.348	0.399	0.399	0.377	0.403
NN	0.072	0.008	0.004	0.062	0.010	0.005

Table 8: class subset random oversampling classifier results.

Classifier	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
SVM	0.378	0.341	0.360	0.421	0.365	0.435
RF	0.506	0.442	0.525	0.427	0.491	0.514
NN	0.502	0.440	0.439	0.457	0.502	0.517

Table 9: class subset ENN classifier results.

Classifier	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
SVM	0.274	0.220	0.273	0.248	0.247	0.307
RF	0.297	0.209	0.431	0.254	0.254	0.464
NN	0.189	0.121	0.154	0.151	0.164	0.199

Table 10: class subset datagen classifier results.

Classifier	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
NN	0.590	0.487	0.505	0.491	0.577	0.579

Table 11: class subset dummy classifier results.

Strategy	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
stratified	0.091	0.066	0.066	0.066	0.091	0.091
most_frequent	0.174	0.019	0.011	0.063	0.052	0.030
uniform	0.068	0.061	0.068	0.068	0.076	0.099

Table 12: class subset SMOTE classifier results.

Classifier	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
SVM	0.380	0.346	0.358	0.424	0.368	0.432
RF	0.496	0.452	0.460	0.467	0.489	0.494
NN	0.524	0.460	0.461	0.470	0.520	0.531

Table 13: class subset TOMEK classifier results.

Classifier	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
SVM	0.423	0.339	0.406	0.346	0.389	0.423
RF	0.488	0.395	0.552	0.377	0.464	0.520
NN	0.575	0.494	0.533	0.494	0.570	0.574

Table 14: class subset SMOTETOMEK classifier results.

Classifier	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
SVM	0.380	0.346	0.359	0.424	0.368	0.433
RF	0.500	0.456	0.464	0.469	0.493	0.498
NN	0.525	0.462	0.466	0.471	0.524	0.536

Table 15: class subset ADASYN classifier results.

Classifier	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
SVM	0.381	0.347	0.358	0.427	0.370	0.432
RF	0.501	0.454	0.455	0.470	0.495	0.502
NN	0.521	0.456	0.448	0.477	0.520	0.534

Table 16: Support Vector Machine iterative imageset stratification results.

%data	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
1	0.198	0.056	0.063	0.055	0.172	0.160
2	0.230	0.059	0.068	0.060	0.188	0.182
3	0.272	0.065	0.075	0.070	0.204	0.188
5	0.303	0.068	0.093	0.075	0.206	0.196
7.5	0.321	0.069	0.104	0.075	0.214	0.203
10	0.330	0.074	0.116	0.079	0.220	0.213

Table 17: Random Forest iterative image set stratification results.

%data	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
1	0.272	0.016	0.048	0.019	0.128	0.117
2	0.282	0.024	0.072	0.025	0.139	0.145
3	0.299	0.038	0.089	0.035	0.157	0.169
7.5	0.312	0.037	0.165	0.034	0.167	0.232
10	0.311	0.039	0.167	0.035	0.169	0.239

Table 18: Neural Network iterative image set stratification results.

%data	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
1	0.198	0.059	0.065	0.058	0.182	0.174
2	0.241	0.089	0.099	0.087	0.224	0.216
3	0.282	0.110	0.125	0.105	0.259	0.246
5	0.304	0.129	0.139	0.130	0.285	0.277
7.5	0.362	0.167	0.190	0.163	0.334	0.324
10	0.383	0.192	0.214	0.182	0.353	0.340

Table 19: Support Vector Machine stratified subset results with standard deviation in parenthesis (n=5).

%data	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
1	0.216	0.053	0.062	0.051	0.195	0.184
2	0.265	0.059	0.065	0.062	0.217	0.203
3	0.291	0.061	0.076	0.064	0.217	0.201
5	0.319	0.069	0.091	0.074	0.221	0.204
7.5	0.333	0.072	0.114	0.077	0.223	0.217
10	0.333(0.000)	0.053(0.000)	0.138(0.000)	0.056(0.000)	0.206(0.000)	0.215(0.000)

Table 20: Random Forest stratified subset results with standard deviation in parenthesis (n=5).

%data	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
1	0.310	0.014	0.049	0.017	0.155	0.157
2	0.314	0.019	0.079	0.020	0.162	0.175
3	0.316	0.027	0.100	0.026	0.166	0.181
5	0.321	0.032	0.118	0.029	0.173	0.202
7.5	0.321	0.035	0.171	0.031	0.174	0.237
10	0.332(0.001)	0.051(0.001)	0.288(0.017)	0.043(0.001)	0.193(0.001)	0.306(0.007)

Table 21: Neural Network stratified subset results with standard deviation in parenthesis (n=5). * represents the usage of a data generator.

%data	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
1	0.225	0.049	0.056	0.050	0.206	0.193
2	0.261	0.077	0.084	0.076	0.240	0.227
3	0.286	0.100	0.105	0.098	0.266	0.251
5	0.313	0.130	0.144	0.123	0.297	0.288
7.5	0.364	0.160	0.178	0.154	0.331	0.314
10	0.420(0.003)	0.200(0.002)	0.253(0.008)	0.186(0.005)	0.366(0.005)	0.358(0.004)
10*	0.399(0.001)	0.146(0.003)	0.234(0.009)	0.148(0.005)	0.314(0.007)	0.333(0.006)

Table 22: Support Vector Machine Tomek Links with standard deviation in parenthesis (n=5).

%data	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
1	0.309	0.045	0.063	0.050	0.196	0.172
2	0.322	0.049	0.081	0.053	0.199	0.183
3	0.323	0.049	0.115	0.052	0.198	0.199
5	0.326	0.048	0.087	0.051	0.199	0.184
7.5	0.327	0.048	0.112	0.051	0.201	0.198
10	0.333(0.000)	0.054(0.000)	0.127(0.000)	0.056(0.000)	0.207(0.000)	0.210(0.000)

Table 23: Random Forest Tomek Links.

%data	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
1	0.315	0.024	0.069	0.026	0.164	0.171
2	0.320	0.032	0.154	0.031	0.171	0.227
3	0.321	0.035	0.151	0.034	0.174	0.231
5	0.323	0.037	0.165	0.034	0.178	0.244
7.5	0.327	0.045	0.203	0.040	0.185	0.258
10	0.331(0.001)	0.051(0.001)	0.260(0.013)	0.043(0.002)	0.192(0.001)	0.293(0.011)

Table 24: Neural Network Tomek Links.

%data	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
1	0.312	0.025	0.039	0.033	0.183	0.153
2	0.342	0.064	0.097	0.069	0.237	0.214
3	0.349	0.096	0.130	0.094	0.265	0.247
5	0.380	0.126	0.184	0.116	0.291	0.285
10	0.421(0.005)	0.193(0.006)	0.264(0.010)	0.180(0.008)	0.362(0.007)	0.362(0.002)

Table 25: dummy classifier results.

strategy	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
stratified	0.104	0.012	0.012	0.012	0.104	0.104
uniform	0.013	0.009	0.013	0.012	0.016	0.108
most_frequent	0.306	0.006	0.004	0.013	0.144	0.094

Table 26: Support Vector Machine with RBF kernel stratified results.

%data	Accuracy/micro	F1 m	Precision m	Recall m	F1 w	Precision w
0.01	0.340	0.057	0.158	0.048	0.208	0.340
0.02	0.348	0.074	0.206	0.06	0.234	0.348
class subset	0.513	0.465	0.464	0.475	0.512	0.517

TABLE 27: Classification report for the neural network on the class subset

Id	precision	recall	f1-score	support
2	0.725	0.745	0.735	2474
11	0.733	0.702	0.717	650
14	0.531	0.508	0.519	512
17	0.467	0.612	0.530	1669
23	0.542	0.348	0.424	462
34	0.694	0.690	0.692	935
36	0.540	0.597	0.567	794
39	0.610	0.649	0.629	1022
58	0.664	0.689	0.676	1021
65	0.494	0.355	0.413	1450
74	0.560	0.617	0.587	851
76	0.603	0.571	0.586	1028
78	0.664	0.672	0.668	539
80	0.375	0.038	0.070	78
89	0.000	0.000	0.000	74
90	0.385	0.363	0.373	634

TABLE 28: Classification report of the RF on 10% of the COCO dataset

Id	precision	recall	f1-score	support
1	0.332	0.993	0.498	18532
2	0.000	0.000	0.000	496
3	0.376	0.064	0.109	3079
4	0.000	0.000	0.000	602
5	0.462	0.078	0.134	383
6	0.000	0.000	0.000	433
7	0.000	0.000	0.000	316
8	0.000	0.000	0.000	705
9	0.500	0.001	0.003	759
10	0.514	0.021	0.040	916
11	0.000	0.000	0.000	131
13	1.000	0.365	0.535	137
14	0.000	0.000	0.000	83
15	0.333	0.001	0.003	675
16	0.143	0.001	0.003	729
17	0.000	0.000	0.000	330
18	0.000	0.000	0.000	377
19	0.000	0.000	0.000	467
20	0.447	0.051	0.092	665
21	0.875	0.012	0.024	569
22	1.000	0.008	0.015	391
23	0.000	0.000	0.000	90
24	1.000	0.087	0.160	369
25	0.000	0.000	0.000	360
27	0.000	0.000	0.000	620
28	0.338	0.029	0.054	787

31	0.000	0.000	0.000	878
32	0.500	0.002	0.004	449
33	0.000	0.000	0.000	425
34	0.429	0.016	0.031	186
35	0.418	0.196	0.267	470
36	0.500	0.005	0.010	196
37	0.438	0.408	0.422	439
38	0.528	0.029	0.055	656
39	0.250	0.004	0.008	240
40	0.000	0.000	0.000	269
41	0.000	0.000	0.000	401
42	0.600	0.007	0.014	416
43	0.000	0.000	0.000	341
44	0.667	0.002	0.005	1698
46	0.000	0.000	0.000	562
47	0.280	0.123	0.171	1451
48	0.333	0.010	0.020	392
49	0.308	0.014	0.028	554
50	0.143	0.002	0.005	429
51	0.210	0.102	0.138	1006
52	0.444	0.006	0.011	691
53	0.500	0.002	0.005	431
54	0.000	0.000	0.000	309
55	0.373	0.054	0.095	460
56	0.000	0.000	0.000	493
57	0.000	0.000	0.000	554
58	0.000	0.000	0.000	202
59	0.698	0.092	0.163	400
60	1.000	0.004	0.008	498
61	0.500	0.002	0.004	455
62	0.209	0.076	0.111	2715
63	0.000	0.000	0.000	411
64	0.000	0.000	0.000	592
65	0.000	0.000	0.000	290
67	0.238	0.009	0.017	1117
70	1.000	0.003	0.007	287
72	0.613	0.248	0.353	404
73	1.000	0.012	0.023	341
74	1.000	0.007	0.013	152
75	0.222	0.005	0.010	412
76	0.000	0.000	0.000	198
77	0.000	0.000	0.000	446
78	1.000	0.008	0.017	119
79	0.000	0.000	0.000	230
80	0.000	0.000	0.000	15
81	0.467	0.018	0.034	393
82	0.000	0.000	0.000	187
84	0.273	0.133	0.179	1732
85	0.674	0.134	0.224	433
86	0.000	0.000	0.000	462

87	0.000	0.000	0.000	107
88	0.000	0.000	0.000	344
89	0.000	0.000	0.000	13
90	0.000	0.000	0.000	138

TABLE 29: Classification report of the SVM on 10% of the COCO dataset

Id	precision	recall	f1-score	support
1	0.366	0.957	0.530	18532
2	0.500	0.002	0.004	496
3	0.192	0.177	0.185	3079
4	0.000	0.000	0.000	602
5	0.248	0.065	0.103	383
6	0.310	0.021	0.039	433
7	0.200	0.003	0.006	316
8	0.143	0.001	0.003	705
9	0.231	0.004	0.008	759
10	0.216	0.023	0.041	916
11	0.000	0.000	0.000	131
13	0.488	0.584	0.532	137
14	0.000	0.000	0.000	83
15	0.000	0.000	0.000	675
16	0.316	0.008	0.016	729
17	0.000	0.000	0.000	330
18	0.000	0.000	0.000	377
19	0.000	0.000	0.000	467
20	0.319	0.035	0.062	665
21	0.130	0.011	0.020	569
22	0.129	0.010	0.019	391
23	0.000	0.000	0.000	90
24	0.343	0.125	0.183	369
25	0.000	0.000	0.000	360
27	0.000	0.000	0.000	620
28	0.214	0.030	0.053	787
31	0.000	0.000	0.000	878
32	0.000	0.000	0.000	449
33	0.000	0.000	0.000	425
34	0.000	0.000	0.000	186
35	0.176	0.302	0.223	470
36	0.000	0.000	0.000	196
37	0.277	0.390	0.324	439
38	0.148	0.014	0.025	656
39	0.043	0.004	0.008	240
40	0.000	0.000	0.000	269
41	0.000	0.000	0.000	401
42	0.000	0.000	0.000	416
43	0.000	0.000	0.000	341
44	0.140	0.014	0.025	1698
46	0.000	0.000	0.000	562

47	0.222	0.145	0.176	1451
48	0.186	0.020	0.037	392
49	0.115	0.020	0.034	554
50	0.000	0.000	0.000	429
51	0.151	0.165	0.158	1006
52	0.083	0.007	0.013	691
53	0.000	0.000	0.000	431
54	0.000	0.000	0.000	309
55	0.212	0.015	0.028	460
56	0.000	0.000	0.000	493
57	0.000	0.000	0.000	554
58	0.000	0.000	0.000	202
59	0.250	0.050	0.083	400
60	0.000	0.000	0.000	498
61	0.000	0.000	0.000	455
62	0.168	0.118	0.139	2715
63	0.000	0.000	0.000	411
64	0.000	0.000	0.000	592
65	0.000	0.000	0.000	290
67	0.182	0.029	0.049	1117
70	0.400	0.007	0.014	287
72	0.277	0.500	0.356	404
73	0.400	0.006	0.012	341
74	0.412	0.046	0.083	152
75	0.000	0.000	0.000	412
76	0.000	0.000	0.000	198
77	0.500	0.002	0.004	446
78	0.264	0.118	0.163	119
79	0.000	0.000	0.000	230
80	0.059	0.133	0.082	15
81	0.222	0.015	0.029	393
82	0.400	0.011	0.021	187
84	0.162	0.105	0.127	1732
85	0.349	0.222	0.271	433
86	0.000	0.000	0.000	462
87	0.000	0.000	0.000	107
88	0.000	0.000	0.000	344
89	0.000	0.000	0.000	13
90	0.000	0.000	0.000	138

TABLE 30: Classification report for the neural network with TOMEK links on 10% of the COCO dataset

Id	precision	recall	f1-score	support
1	0.541	0.880	0.670	18532
2	0.260	0.151	0.191	496
3	0.446	0.478	0.461	3079
4	0.379	0.214	0.274	602
5	0.471	0.256	0.332	383
6	0.360	0.289	0.321	433

7	0.368	0.155	0.218	316
8	0.154	0.033	0.054	705
9	0.216	0.097	0.134	759
10	0.635	0.461	0.534	916
11	0.462	0.046	0.083	131
12	0.857	0.613	0.715	137
13	0.000	0.000	0.000	83
14	0.222	0.024	0.043	675
15	0.259	0.085	0.128	729
16	0.158	0.009	0.017	330
17	0.167	0.005	0.010	377
18	0.322	0.191	0.240	467
19	0.413	0.362	0.386	665
20	0.315	0.158	0.211	569
21	0.412	0.307	0.352	391
22	0.000	0.000	0.000	90
23	0.440	0.320	0.370	369
24	0.454	0.289	0.353	360
25	0.199	0.076	0.110	620
26	0.253	0.178	0.209	787
27	0.088	0.011	0.020	878
28	0.182	0.031	0.053	449
29	0.083	0.002	0.005	425
30	0.249	0.242	0.245	186
31	0.420	0.438	0.429	470
32	0.192	0.026	0.045	196
33	0.520	0.538	0.529	439
34	0.422	0.354	0.385	656
35	0.280	0.242	0.260	240
36	0.167	0.007	0.014	269
37	0.137	0.077	0.099	401
38	0.198	0.079	0.113	416
39	0.205	0.070	0.105	341
40	0.333	0.264	0.295	1698
41	0.343	0.181	0.237	562
42	0.243	0.245	0.244	1451
43	0.110	0.041	0.059	392
44	0.152	0.265	0.193	554
45	0.098	0.028	0.044	429
46	0.233	0.157	0.188	1006
47	0.381	0.436	0.406	691
48	0.248	0.142	0.180	431
49	0.188	0.110	0.139	309
50	0.388	0.559	0.458	460
51	0.395	0.627	0.485	493
52	0.356	0.606	0.448	554
53	0.133	0.050	0.072	202
54	0.335	0.365	0.349	400
55	0.346	0.293	0.317	498
56	0.167	0.015	0.028	455

57	0.196	0.236	0.214	2715
58	0.500	0.002	0.005	411
59	0.413	0.145	0.215	592
60	0.100	0.007	0.013	290
61	0.181	0.098	0.128	1117
62	0.297	0.143	0.193	287
63	0.277	0.450	0.343	404
64	0.217	0.126	0.160	341
65	0.203	0.092	0.127	152
66	0.206	0.180	0.192	412
67	0.053	0.015	0.024	198
68	0.117	0.031	0.049	446
69	0.175	0.252	0.207	119
70	0.093	0.035	0.051	230
71	0.000	0.000	0.000	15
72	0.173	0.099	0.126	393
73	0.050	0.005	0.010	187
74	0.237	0.333	0.277	1732
75	0.615	0.457	0.525	433
76	0.099	0.028	0.044	462
77	0.000	0.000	0.000	107
78	0.071	0.003	0.006	344
79	0.000	0.000	0.000	13
80	0.158	0.022	0.038	138

TABLE 31: Class occurence in the dataset

Id	name	size
1	person	185316
2	bicycle	4955
3	car	30785
4	motorcycle	6021
5	airplane	3833
6	bus	4327
7	train	3159
8	truck	7050
9	boat	7590
10	traffic light	9159
11	fire hydrant	1316
13	stop sign	1372
14	parking meter	833
15	bench	6751
16	bird	7290
17	cat	3301
18	dog	3774
19	horse	4666
20	sheep	6654
21	cow	5686
22	elephant	3905
23	bear	903

24	zebra	3685
25	giraffe	3596
27	backpack	6200
28	umbrella	7865
31	handbag	8778
32	tie	4497
33	suitcase	4251
34	frisbee	1862
35	skis	4698
36	snowboard	1960
37	sports ball	4392
38	kite	6560
39	baseball bat	2400
40	baseball glove	2689
41	skateboard	4012
42	surfboard	4161
43	tennis racket	3411
44	bottle	16983
46	wine glass	5618
47	cup	14513
48	fork	3918
49	knife	5536
50	spoon	4287
51	bowl	10064
52	banana	6912
53	apple	4308
54	sandwich	3089
55	orange	4597
56	broccoli	4927
57	carrot	5539
58	hot dog	2023
59	pizza	4001
60	donut	4977
61	cake	4551
62	chair	27147
63	couch	4113
64	potted plant	5918
65	bed	2905
67	dining table	11167
70	toilet	2873
72	tv	4036
73	laptop	3415
74	mouse	1517
75	remote	4122
76	keyboard	1980
77	cell phone	4460
78	microwave	1189
79	oven	2302
80	toaster	156
81	sink	3933

82	refrigerator	1875
84	book	17315
85	clock	4328
86	vase	4623
87	scissors	1073
88	teddy bear	3442
89	hair drier	135
90	toothbrush	1377

LISTING 1: Example code for neural network with stratified subset and TOMEK links in Python

```
from collections import defaultdict
import itertools
import numpy as np
from numpy import genfromtxt
import os
from PIL import Image, ImageFilter
import keras
import keras.backend as K
from random import shuffle, seed
from skimage.transform import resize
from sklearn.metrics import classification_report, confusion_matrix, roc_curve
from skmultilearn.model_selection import iterative_train_test_split
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import TomekLinks
from imblearn.combine import SMOTETomek
import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dense, Activation, Flatten,
Dropout, GaussianNoise
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical, plot_model
import time

def loadAnnotations(fileLocation):
    with open(fileLocation) as read_file:
        annotations = json.load(read_file)
    return annotations

def getBoundingBox(x,y,w,h,img):
    return img[y:y+h,x:x+w]

def findAnnotationsById(id, property, annotations):
    items = []
    for annotation in annotations:
        if annotation[property] == id:
            items.append(annotation)
    return items

def preProcessImage(img):
    img = np.asarray(img)
    return img

def getBoundingBoxesAnnotations(annotations, path):
    bounded_images = []
    bounded_annotations = []
    try:
        for annotation in annotations:
            image_id = annotation['image_id']
            image_id_string = str(image_id).zfill(12)
            image = Image.open(path + image_id_string +'.jpg').convert('RGB')
            image = preProcessImage(image)
            image_resized = resize(getBoundingBox(int(annotation['bbox'][0]),int(annotation['bbox'][1]),
            int(annotation['bbox'][2]),int(annotation['bbox'][3]),image), (50, 50),anti_aliasing=True)
            bounded_images.append(image_resized)
            bounded_annotations.append(annotation['category_id'])
    except Exception as ex:
            print(ex)
    bounded_images = np.asarray(bounded_images)
    bounded_annotations = np.asarray(bounded_annotations)
    return bounded_images,bounded_annotations

def getBoundingBoxesPicture(image_id, path):
    bounded_images = []
    bounded_annotations = []
    image_id_string = str(image_id).zfill(12)
    image = Image.open(os.path.join(data_folder, path) + image_id_string +'.jpg').convert('RGB')
    image = preProcessImage(image)
    annotations = findAnnotationsById(image_id, 'image_id', train_annotations)
    for annotation in annotations:
        try:
            image_resized = resize(getBoundingBox(int(annotation['bbox'][0]),int(annotation['bbox'][1]),
            int(annotation['bbox'][2]),int(annotation['bbox'][3]),image), (50, 50),anti_aliasing=True)
            bounded_annotations.append(annotation['category_id'])
        except Exception as ex:
            print(ex)
    bounded_images = np.asarray(bounded_images)
    bounded_annotations = np.asarray(bounded_annotations)
    return image_id,bounded_images,bounded_annotations

def constructCNN(conv_layer, layer_size, kernel_size, dense_size, dense_layer, dropout, num_classes, x_train):
    percent_noise = 0.1
    noise = (1.0/255) * percent_noise
    model = Sequential()
    model.add(GaussianNoise(noise, input_shape=x_train.shape[1:]))
    model.add(Conv2D(layer_size, (kernel_size, kernel_size)))

    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    for l in range(conv_layer-1):
        model.add(Conv2D(layer_size, (kernel_size, kernel_size)))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    if(dropout):
        model.add(Dropout(0.25))
    model.add(Flatten())

    for j in range(dense_layer):
        model.add(Dense(dense_size))
        model.add(Activation('relu'))
    if(dropout):
        model.add(Dropout(0.25))
    model.add(Dense(num_classes))
    model.add(Activation('softmax'))

    model.compile(loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

    return model

def probability_mass_split(y, images, folds=7):
    obs, classes = y.shape
    dist = y.sum(axis=0).astype('float')
    dist /= dist.sum()
    index_list = []
    image_list = []
    fold_dist = np.zeros((folds, classes), dtype='float')
    for _ in range(folds):
        index_list.append([])
    for i in range(obs):
        if i < folds:
            target_fold = i
        else:
            normed_folds = fold_dist.T / fold_dist.sum(axis=1)
            how_off = normed_folds.T - dist
            target_fold = np.argmin(np.dot((y[i] - .5).reshape(1, -1), how_off.T))
        fold_dist[target_fold] += y[i]
        index_list[target_fold].append(i)
        image_list[target_fold].append(images[i])
    print("Fold distributions are")
    print(fold_dist)
    return index_list,image_list

def random_sets(y, images, folds):
    index_list = []
    image_list = []
    for _ in range(folds):
        index_list.append([])
        image_list.append([])
    c = list(zip(y, images))
    seed(42)
    shuffle(c)
    y, images = zip(*c)
    foldsize = int(len(y)/folds)
    idx = 0
    for i in range(folds):
        if i < folds-1:
            index_list[i] = y[idx:idx+foldsize]
            image_list[i] = images[idx:idx+foldsize]
        else:
            index_list[i] = y[idx:]
            image_list[i] = images[idx:]
        idx += foldsize
    return index_list,image_list

def getBoundingBoxPreselected(annotations, path):
    bounded_images = []
    bounded_annotations = []
    for annotation in annotations:
        try:
            image_id = annotation[0]
            image_id_string = str(image_id).zfill(12)
            image = Image.open(os.path.join(data_folder, path) + image_id_string +'.jpg').convert('RGB')
            image = preProcessImage(image)
            image_resized = resize(getBoundingBox(int(annotation[1][0]),int(annotation[1][1]),
            int(annotation[1][2]),int(annotation[1][3]),image), (50, 50),anti_aliasing=True)
            bounded_images.append(image_resized)
            bounded_annotations.append(annotation[2])
        except Exception as ex:
                print(ex)
    bounded_images = np.asarray(bounded_images)
    bounded_annotations = np.asarray(bounded_annotations)
    return bounded_images, bounded_annotations

path = "train2014/COCO_train2014_"
category = "category_id"

# read data
with open(os.path.join(data_folder, "annotations/instances_train2014.json")) as read_file:
    train = json.load(read_file)
train_annotations = train["annotations"]
del train

#Load all annotations, when this is done pick a stratified subset and get the bounding boxes
bbox_category_ids = []
bboxes = []
for annotation in train_annotations:
    bbox_category_ids.append(annotation['category_id'])
    bboxes.append((annotation['image_id'],annotation['bbox'],annotation['category_id']))
```

```
train_split_strat_x, test_split_strat_x, bounded_annotations_train, bounded_annotations_val=
train_test_split(bboxes,bbox_category_ids,test_size=0.1,train_size=0.1, stratify=bbox_category_ids)
images_train, bounded_annotations_train = getBoundingBoxPreselected(train_split_strat_x, path)
images_val, bounded_annotations_val = getBoundingBoxPreselected(test_split_strat_x, path)

#data generator example, can be uncommented when using mode.fit_generator
'''
datagen = ImageDataGenerator(
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
datagen.fit(x_train)
print("datagen train")
'''

#flatten the image array and undersample using TOMEK links
dataset_size = images_train.shape[0]
TwoDim_dataset = images_train.reshape(dataset_size,-1)

tl = TomekLinks()
X_res_tm, y_res_tm = tl.fit_resample(TwoDim_dataset, bounded_annotations_train)
x_res_3d = X_res_tm.reshape(X_res_tm.shape[0], 50, 50, 3)

images_train = x_res_3d
bounded_annotations_train = y_res_tm

#Normalize the pixels between 0 and 1 by dividing by 255
x_train = images_train/255
x_test = images_val/255
#Correct the id's for the classification report of the neural network
corrected_ids = [-1,0,1,2,3,4,5,6,7,8,9,10,-1,11,12,13,14,15,16,17,18,19,20,21,22,23,-1,
24,25,-1,-1,26,27,28,29,30,31,32,33,34,35,36,37,38,39,-1,40,41,42,43,44,45,46,47,48,49,50,51,
52,53,54,55,56,57,58,59,-1,60,-1,-1,61,-1,62,63,64,65,66,67,68,69,70,71,72,-1,73,74,75,76,77,78,79]
for index, item in enumerate(bounded_annotations_train):
    bounded_annotations_train[index] = corrected_ids[bounded_annotations_train[index]]
for index, item in enumerate(bounded_annotations_val):
    bounded_annotations_val[index] = corrected_ids[bounded_annotations_val[index]]

#Transform labels to categorical(one-hot encoding)
ylabels_train = to_categorical(bounded_annotations_train)
ylabels_test = to_categorical(bounded_annotations_val)

with tf.Session() as session:
    session.run(tf.global_variables_initializer())
    early_stopping_monitor = EarlyStopping(patience = 10)
    np.set_printoptions(threshold=np.inf)
    #Neural network function that can be used as grid search or just to train one NN
    dense_layers = [3]
    layer_sizes = [128]
    dense_sizes = [128]
    conv_layers = [3]
    kernel_sizes = [3]
    for dense_layer in dense_layers:
        for dense_size in dense_sizes:
            for kernel_size in kernel_sizes:
                for layer_size in layer_sizes:
                    for conv_layer in conv_layers:
                        model = constructCNN(conv_layer, layer_size, kernel_size, dense_size, dense_layer,
                        True, len(ylabels_train[0]), x_train)
                        NAME = "{}-conv-{}-nodes-{}-dense-{}-dense_size-{}-kernel".format(conv_layer,
                        layer_size, dense_layer, dense_size, kernel_size)
                        print(NAME)
                        #log to tensorboard
                        tensorboard = TensorBoard(log_dir="outputs/{}".format(NAME))
                        model.fit(x_train, ylabels_train.astype(np.float32), batch_size=70, epochs=200,
                        validation_data = (x_test, ylabels_test), callbacks = [tensorboard, EarlyStopping],
                        shuffle=True)
                        '''
                        # fit to generator example NN
                        model.fit_generator(datagen.flow(x_train, ylabels_train, batch_size=32,shuffle=True),
                        epochs=10, callbacks = [tensorboard], validation_data = (x_test, ylabels_test))
                        '''
                        #let the neural network predict the classes and transform to human readable format
                        y_pred = model.predict(x_test)
                        predictions = tf.argmax(y_pred,1)
                        true_class = tf.argmax(ylabels_test, 1)
                        true_class_list = true_class.eval()
                        predictions_list = predictions.eval()
                        #show classification report and confusion matrix
                        print(classification_report(true_class_list, predictions_list, digits=3))
                        con_mat = tf.confusion_matrix(true_class, predictions)
                        print('Confusion Matrix: \n\n', tf.Tensor.eval(con_mat,feed_dict=None, session=None))
                        #save the model for later use
                        model.save('outputs/' + NAME +'0.1_SMOTETomek_100epochs.h5')
```

LISTING 2: Example code for SVM and RF with stratified subset and TOMEK links in Python

```
from collections import defaultdict
import numpy as np
import os
from PIL import Image, ImageFilter
from random import shuffle, seed
from skimage import io as io
```

```
from skimage import exposure
from skimage.transform import resize, integral_image
from skimage.color import rgb2gray
from skimage.feature import daisy, hog, multiblock_lbp
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.externals import joblib
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.svm import SVC, LinearSVC
import itertools
from imblearn.under_sampling import TomekLinks
from skmultilearn.model_selection import iterative_train_test_split
from sklearn.dummy import DummyClassifier
import time

def loadAnnotations(fileLocation):
    with open(fileLocation) as read_file:
        annotations = json.load(read_file)
    return annotations

def getBoundingBox(x,y,w,h,img):
    return img[y:y+h,x:x+w]

def preProcessImage(img):
    img = np.asarray(img)
    img = rgb2gray(img)
    img = cv2.GaussianBlur(img,(5,5),0)
    #img = cv2.medianBlur(img,5)
    #img = cv2.bilateralFilter(img,9,75,75)
    #img = cv2.blur(img,(5,5))
    #kernel = np.ones((5,5),np.float32)/25
    #img = cv2.filter2D(img,-1,kernel)
    return img

def getBoundingBoxPreselected(annotations, path):
    bounded_images = []
    bounded_annotations = []
    for annotation in annotations:
        try:
            image_id = annotation[0]
            image_id_string = str(image_id).zfill(12)
            image = Image.open(os.path.join(data_folder, path) + image_id_string +'.jpg').convert('RGB')
            image = preProcessImage(image)
            image_resized = resize(getBoundingBox(int(annotation[1][0]),int(annotation[1][1]),int(annotation[1][2])
            ,int(annotation[1][3]),image), (50, 50), anti_aliasing=True)
            bounded_images.append(image_resized)
            bounded_annotations.append(annotation[2])
        except Exception as ex:
                print(ex)
    bounded_images = np.asarray(bounded_images)
    bounded_annotations = np.asarray(bounded_annotations)
    return bounded_images, bounded_annotations

def calculateHogFeatures(gray_image, o, pixels, cells):
features = hog(gray_image, orientations=o,
                           pixels_per_cell=(pixels, pixels),
                           cells_per_block=(cells, cells),
                           transform_sqrt=True,
                           visualize=False, block_norm = "L2-Hys")
    return features
def calculateDaisyFeatures(gray_image):
    descs = daisy(gray_image, step=180, radius=15, rings=3, histograms=8,
                          orientations=8, visualize=False)
    descs_num = descs.shape[0] * descs.shape[1]
    return descs.reshape(descs.size).tolist()

def calculateFeatures(imgs):
    hog_features = []
    daisy_features = []
    lbp_features = []
    for img in imgs:
        hog_features.append(calculateHogFeatures(img,8,12,1))
        daisy_features.append(calculateDaisyFeatures(img))
        int_img = integral_image(img)
        lbp_features.append(multiblock_lbp(int_img, 0, 0, 30, 30))
    return  hog_features, daisy_features, lbp_features

def svmFit(x_train, y_train):
    print("Fitting the classifier to train")
    clf = LinearSVC()
    clf = clf.fit(x_train, y_train)
    return clf

def svm_rbfFit(x_train, y_train):
    print("Fitting the classifier to train")
    param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
                  'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
    clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'),
                       param_grid, cv=5)
    clf = clf.fit(x_train, y_train)
    print("Best estimator found by grid search:")
    print(clf.best_estimator_)
    return clf

def svmPredict(x_test, y_test, model):
    print("Predicting the test set")
```

```
    y_pred = model.predict(x_test)
    print(classification_report(y_test, y_pred, digits = 3))
    conf_mat = confusion_matrix(y_test, y_pred)
    print(conf_mat)
    return conf_mat

def randomForestFit(x_train, y_train, estimators):
    print("Fitting the classifier to train")
    rf = RandomForestClassifier(n_estimators=estimators)
    rf.fit(x_train, y_train);
    return rf

def rfPredict(x_test, y_test, model):
    print("Predicting the test set")
    y_pred = model.predict(x_test)
    print(classification_report(y_test, y_pred, digits = 3))
    conf_mat = confusion_matrix(y_test, y_pred)
    print(conf_mat)
    return conf_mat

def saveModel(model, filename):
    print("Saving file...")
    joblib.dump(model, open(filename, 'wb'))
    print("File saved")

def loadModel(filename):
    print("loading file...")
    joblib.load(filename)
    print("Model loaded")

np.set_printoptions(threshold=np.inf)

bbox_category_ids = []
bboxes = []
for annotation in train_annotations:
    bbox_category_ids.append(annotation['category_id'])
    bboxes.append((annotation['image_id'],annotation['bbox'],annotation['category_id']))
train_split_strat_x, test_split_strat_x, bounded_annotations_train, bounded_annotations_val=
train_test_split(bboxes,bbox_category_ids,test_size=0.1,train_size=0.1, stratify=bbox_category_ids)
images_train, bounded_annotations_train = getBoundingBoxPreselected(train_split_strat_x, path)
images_val, bounded_annotations_val = getBoundingBoxPreselected(test_split_strat_x, path)


full_picture_features = calculateFeatures(images_train)

full_picture_features_val = calculateFeatures(images_val)

picture_features_svm = [np.concatenate((x,y)) for x,y in zip(full_picture_features[0], full_picture_features[1])]
picture_features_svm_val = [np.concatenate((x,y)) for x,y in zip(full_picture_features_val[0],
                            full_picture_features_val[1])]

svm = svmFit(picture_features_svm, bounded_annotations_train)
svmResults = svmPredict(picture_features_svm_val, bounded_annotations_val, svm)
joblib.dump(value=svm, filename='outputs/svm_strat_tm_0,925.npy')
joblib.dump(value=svmResults, filename='outputs/svm_strat_tm_confusion_matrix_0,925.npy')

#SVM with RBF kernel
#svmrbf = svm_rbfFit(picture_features_svm, bounded_annotations_train)
#svmrbfResults = svmPredict(picture_features_svm_val, bounded_annotations_val, svmrbf)

del picture_features_svm
del picture_features_svm_val

picture_features_rf = [np.concatenate((x,y,[z])) for x,y,z in zip(full_picture_features[0],
                       full_picture_features[1],full_picture_features[2])]
picture_features_rf_val = [np.concatenate((x,y,[z])) for x,y,z in zip(full_picture_features_val[0],
                          full_picture_features_val[1], full_picture_features_val[2])]

del full_picture_features
del full_picture_features_val

rf = randomForestFit(picture_features_rf, bounded_annotations_train, 300)
rf_conf = rfPredict(picture_features_rf_val, bounded_annotations_val, rf)
```