Roof type classification in aerial imagery using convolutional neural networks
exploring the exploitation of convolutional features using feature coding and random forests

van Driel, B.

Award date:
2019

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Industrial Engineering & Innovation Management
Information Systems Group

Roof Type Classification in Aerial Imagery Using Convolutional Neural Networks

Exploring the exploitation of convolutional features using feature coding and random forests

Bart van Driel

BSc in Economics and Business Economics

In partial fulfilment of the requirements for the degree of
Master of Science in Operations Management and Logistics

Supervisors:
Dr. Anna Wilbik (TU/e)
Dr. Yingqian Zhang (TU/e)
Dr. Laura Genga (TU/e)
Erlijn Linskens (Pipple)
Daniël Kersbergen (Netherlands Red Cross - 510)

Eindhoven, March 2019

Eindhoven University of Technology
School of Industrial Engineering
Series Master Theses Operations Management and Logistics

Keywords: Image Classification, Roof Type, Remote Sensing, Convolutional Neural Networks, Feature Coding, Random Forests

Abstract

Accurate, up-to-date and complete data is essential for efficient and effective humanitarian aid. For example, building characteristics can be an important variable in damage prediction models. This work investigates the classification of roof types (i.e. shape and material) in aerial imagery of Sint Maarten using Convolutional Neural Networks (CNNs). Additionally, the UC-Merced data set is used as benchmark data set. Two strategies are presented trying to improve the classification performance of fine-tuned CNNs. In the first approach, features are extracted from the last convolutional layer and encoded using four coding algorithms and classification is done by a linear Support Vector Machine. The second approach extracts the features from the last fully connected layer and classification is done by Random Forests. The best performing fine-tuned CNN, VGG-16, achieved an accuracy of 88% and 58% on respectively the roof shape and roof material data sets. None of the feature coding schemes significantly improved the performance of the fine-tuned VGG-16. Only on the roof material data set, Random Forests showed a small improvement over the fine-tuned VGG-16.

Executive Summary

Introduction

Accurate, up-to-date and complete data is essential for efficient and effective humanitarian aid. For example, detailed maps, including the outlines and characteristics of buildings, can be important for damage prediction models to enhance the humanitarian aid in disaster response. Currently, such maps are obtained by the efforts of many volunteers, which remotely trace buildings, roads, and additional information in satellite imagery. However, the process of acquiring these maps is time consuming, labour intensive, and the quality depends heavily on the skills of the volunteers. Thus, a more swift, efficient, and reliable solution for this problem is desired.

The Netherlands Red Cross data team, 510, is investigating to (partly) automate the mapping using aerial imagery and machine learning. This investigation is split in two research directions: one project focuses on the automatic detection of buildings, and another on automatic classification of building characteristics in remotely sensed imagery. This research will explore the latter topic. More specifically, it investigates the use of state-of-the-art Convolutional Neural Networks (CNNs) for classification of roof types in aerial imagery of Sint Maarten.

Recent years, CNNs pre-trained on large image data bases have shown outstanding results on other data sets. Fine-tuning the pre-trained networks on the target data set tends to give the best results. In the first approach, features are extracted from the last convolutional layer and encoded using four coding algorithms and classification is done by a linear Support Vector Machine (SVM). The second approach extracts the features from the last fully connected layer and classification is done by Random Forests. These strategies are visualised in figure 1.

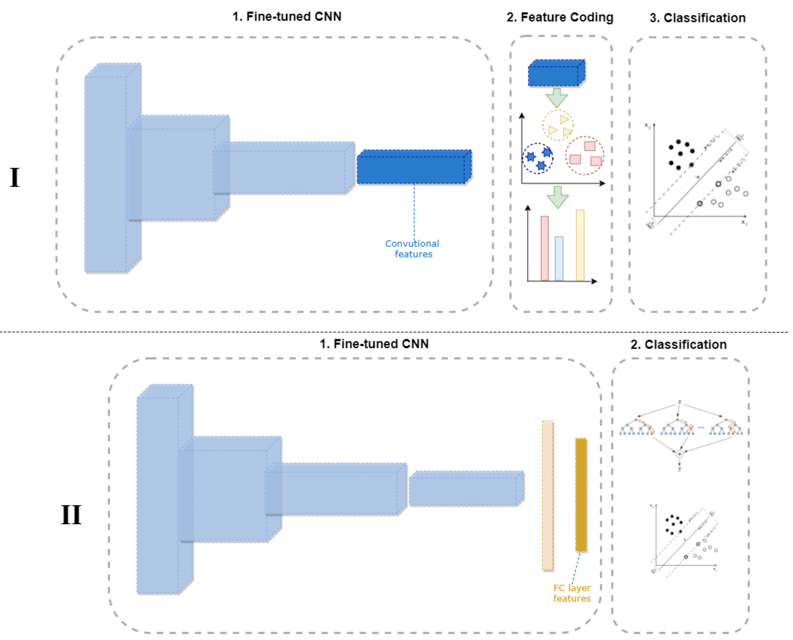


Figure 1: Illustration of the proposed scenarios

Data & Experimental Setup

Three data sets are used to evaluate the models. Two data sets are obtained from aerial imagery of Sint Maarten and detailed building information from OpenStreetMap. Individual buildings are clipped from aerial imagery. The OpenStreetMap data contains characteristics of the roofs of the buildings, i.e. roof shape and roof material. The roof shape is either flat or hipped. The roof material is either concrete, metal or roof tiles. The roof shape and roof material data set consist of respectively 8,349 and 11,661 images, where both data sets exhibit class imbalance. The third data set used is the UC-Merced data set, which serves as a benchmark data set. The data set contains 2100 aerial images evenly distributed over 21 classes.

For the Sint Maarten data sets, only 10% of the total data is stratified sampled as train set. This process is repeated 10 times, resulting in 10 different splits. For the UC-Merced data set, the same split strategy is used as in other works, which is a stratified 5-fold cross validation. First, three pre-trained networks (i.e. VGG-16, InceptionV3, Xception) are fine-tuned on the the train set, where 30% of the train set is used as validation set. Subsequently, the best performing network is used as feature extractor for the proposed approaches.

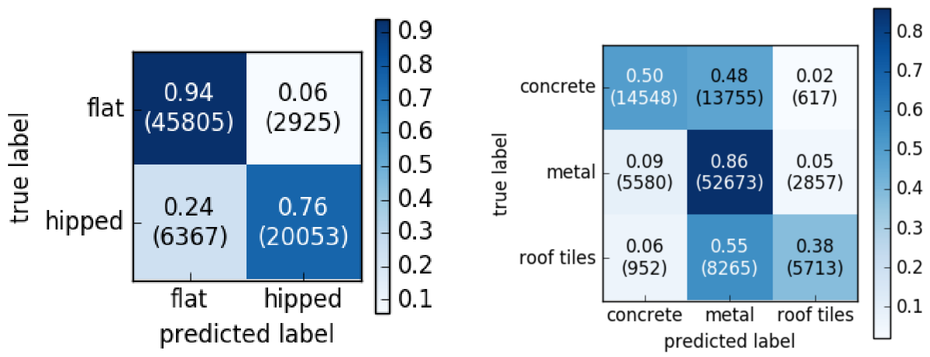
The four coding algorithms used are BOW, VLAD, LLC and IFK. Due to computational limitations, no parameter tuning is done for these algorithms and the hyper-parameters are set based on previous works. The number of clusters in the K-means in BOW, VLAD, and LLC are set to respectively 1000, 100 and 100. The number components in the Gaussian Mixture Model in IFK is set to 100.

The hyper-parameters of Random Forests are tuned via Random Search with 3 folds and 60 draws. No parameter tuning is done for the linear SVM, which parameters are set as described in other works.

Results & Discussion

Overall, VGG-16 achieves the best results on the data sets and is chosen as feature extractor for the proposed methods. None of the feature coding schemes significantly improved the performance of the fine-tuned VGG-16. Only on the roof material data set, Random Forests showed a small improvement over the fine-tuned VGG-16.

The imbalanced classes in the data sets led to even more skewed predictions. Figure 2 shows the confusion matrices of the roof shape and the roof material data sets. The precision is relatively balanced, i.e. 0.88/0.87 and 0.70/0.71/0.63 for respectively the roof shape and roof material data set. The class imbalance is more visible in the recall scores, as indicated by the diagonal values in the confusion matrices. This skewness can be attributed to the optimisation on accuracy, which only takes the amount of correctly predicted samples into account, ignoring per-class performance. This skewness can be corrected, but will cost in terms of accuracy. Increasing the amount of training data has a positive effect on the performance of the CNN. However, this effect diminishes over the amount of data already in the train data set.



(a) Shape (b) Material

Figure 2: Confusion matrices of VGG-16 on the (a) roof shape and (b) roof material data set

The fine-tuned VGG-16 achieved a mean accuracy of 90% and the SVM-CNN combination achieved an mean accuracy of 92%. Similar setups in other works achieved higher mean accuracies. A number of reasons could be the cause of this difference. First, the choice of the deep learning framework might have affected the performance of the models. Supposedly equivalent pre-trained networks in Keras and Caffe result in different classification performance. Secondly, the SVM implementation in the other works could be different. At last, several hyper-parameters during training of the network could be different. However, in the absence of publicly available code, it is difficult to point out the underlying cause of this difference.

Conclusion

Future coding did not improve over the base line, the fine-tuned VGG-16. The other strategy, convolutional features in combination with random forests, showed only a slight improvement on the roof material data set. Both strategies are more complex than conventional Convolutional Neural Network (CNN) classification pipe lines, as they introduce extra steps (i.e. feature coding and classification). Hence, based on the results presented in this work, the potential benefits (i.e. increased classification performance) of experimenting with such pipelines do not outweigh the disadvantages (i.e. more complex pipe line, extra computation time).

The classification of the roof types is partly successful. Classification of the roof shapes is of satisfactory quality, as the accuracy is higher than the desired 85% (i.e. 88%). Classification of the roof material is more problematic, as the achieved accuracy is only 69%.

The key factor in the success of classifying building characteristics is data. Potentially, CNNs can outperform humans in image classification tasks, but this depends very much on the quantity and quality of the data. Additionally, it is important to have a high quality test set for evaluation purposes which is a fair representation of the area of interest.

Therefore, three future research directions are denoted which aim to improve the quality or quantity of the data. First, CNNs can be used to enhance the resolution of imagery. Secondly, research should be conducted in how to efficiently and effectively label large data sets. At last, capturing walls of building in aerial imagery is still problematic and requires more research before it can be implemented in the future.

Preface

This report marks the end of my life as a student. In a period of six and a half years, I studied various subjects at four universities in three different countries and worked with a very diverse group of people. While the research presented in this work is conducted in the past 6 months, it is a product of all experiences over the past years. This process would not have been the same without the support of others.

First, I would like to thank my direct supervisors: Daniël, Erlijn and Anna. Daniël, your technical expertise and domain knowledge helped me a lot during execution of this project. Erlijn, your guidance and our discussions forced me to evaluate the decisions I made and thereby improving the overall quality of my work. Anna, your comments helped me a lot writing and structuring not only this thesis, but also my literature review and research proposal, which was all new to me. I feel lucky to have had three supervisors who always were willing to free up time for regular meetings, which I enjoyed and helped to bring this project to an successful end. Further, I would like to thank Yingqian for her comments on draft versions of my research proposal and thesis.

The last years I tried a lot of new things, which was not always the easiest path. I would like to thank my friends for their help in finding my way. Many of the experiences would not have been so joyful and valuable, or would not even have happened, without the help of you all.

At last, I would like to thank my family. Katja, you were always willing to listen to my frustrations. You helped me put things into perspective whenever things were not working as I wanted. Marieke, I admire your perseverance and resilience in everyday life. You inspire me to push through whenever life gets difficult. Mom and dad, I feel privileged to have had your unconditional support during my studies. I had the freedom to make my own choices and was able to explore many opportunities, for which I am grateful. Thank you.

Bart van Driel

Contents

Glossary

ANN Artificial Neural Network. xiii, 8, 9

BOW Bag Of visual Words. xiii, 2, 8, 11–13, 26, 36, 39, 51

CNN Convolutional Neural Network. vii, xiii, 2, 3, 7–9, 11, 13, 21, 23–27, 31, 39, 41, 46, 48, 51–53

CRISP-DM Cross Industry Standard Process for Data Mining. xiii, 3, 4, 49, 52

FCN FCN. xiii

FN False Negative. xiii, 28, 43

FP False Positive. xiii, 28, 43

FV Fisher Vector. xiii, 12, 13

GMM Gaussian Mixture Model. xiii, 12, 26

IFK Improved Fisher Kernel. xiii, 8, 12, 13, 26, 36, 41, 51

LLC Locality-constrained Linear Coding. xiii, 8, 12, 13, 26, 51

MLP Multi Layer Perceptron. xiii

NLRC Netherlands Red Cross. xiii, 2, 47, 51

RF Random Forests. xiii, 7, 8, 13–15, 21, 27, 37, 41, 51–53

SVM Support Vector Machine. xiii, 2, 7, 8, 13, 14, 21, 23, 26, 27, 35, 37, 41, 44, 46, 48, 51–53

TN True Negative. xiii, 28, 43

TP True Positive. xiii, 28, 43

VLAD Vector of Locally Aggregated Descriptors. xiii, 8, 12, 13, 26, 35, 36, 39, 51

List of Figures

List of Tables

Chapter 1

Introduction

Every year, approximately 90 thousand people are killed and almost 160 million people are affected by natural disasters, such as earthquakes floods, wildfires, and droughts[1]. Besides the direct impact on the local communities of the affected areas, natural disasters have a large economical impact. In 2017, the total economic loss due to natural disasters is estimated to be a staggering 345 billion US dollar [MunichRE, 2018]. Natural disasters occur everywhere and impact people around the world. However, more developed countries, in terms of income, education, economy and governmental efficiency , suffer less from natural disasters than relatively underdeveloped ones [Kousky, 2014]. Figure 1.1 illustrates the upward trend in the amount of natural disasters over the past decades. This upward trend is not expected to change any time soon, as climate change is increasing the odds and the intensity of extreme weather events [ECIU, 2017].

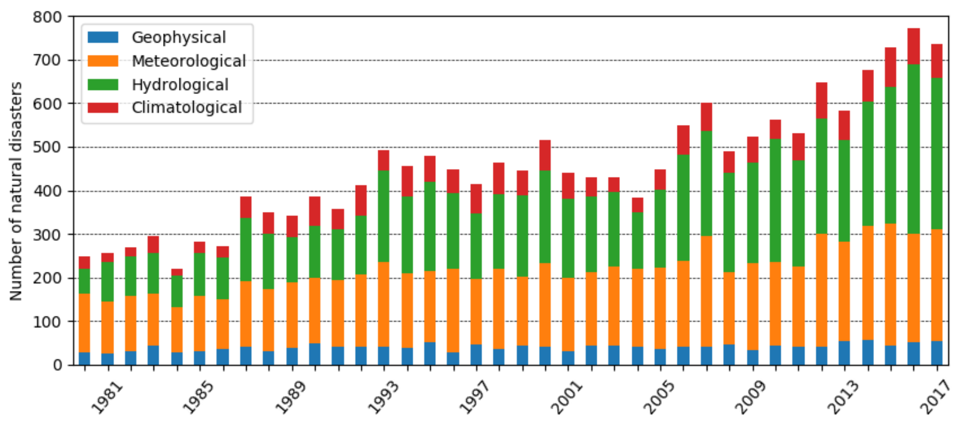


Figure 1.1: Number of natural disasters per category between 1980 and 2017 (Source: Munich RE: NatCatSERVICE - `https://natcatservice.munichre.com`)

Examples of major natural disasters of the past years are the floods in Malawi and hurricane Irma in Sint Maarten. In 2015, heavy rainfall caused the water level of the Shire River, the largest river in Malawi, to rise to the highest level in 30 years. 15 out of 29 districts were impacted by the floods, affecting over one million people. Early September 2017, hurricane Irma struck the island of Sint Maarten, affecting 90% of the buildings and leaving 7000 people (roughly 17% of the total population) without a house. The effectiveness, efficiency, and swiftness of humanitarian aid in such situations depends on the availability of local information, such as maps. However, in

[1]`https://www.who.int/environmental_health_emergencies/natural_events/en/`

practice this information is often not available. For instance, maps for rural areas in developing countries such as Malawi, are generally unavailable or incomplete. Furthermore, in many cases information about the current state of the affected areas is crucial, such as after hurricane Irma in Sint Maarten.

Initiatives like the MissingMaps[2] project help humanitarian aid organisations to obtain the necessary maps. Through MissingMaps, volunteers can remotely trace buildings, roads, and other information in aerial imagery into OpenStreetMap[3], an open source map data base. Subsequently, local volunteers can add more detailed information about the traced objects, e.g. street names and building characteristics. Currently, MissingMaps is the go-to solution for humanitarian aid organisations when up-to-date and complete map data is unavailable. However, the process of acquiring these maps is time consuming, labour intensive, and the quality depends heavily on the skills of the volunteers. Thus, a more swift, efficient, and reliable solution for this problem is desired.

510, the Netherlands Red Cross (NLRC) data team, is investigating the possibility of (partly) automating the mapping process of remote areas using aerial imagery and machine learning. The desired models should be able to detect (the outlines) of buildings within the aerial imagery, and determine additional building characteristics of each building. Both objectives have their own inherent challenges and data needs. Therefore, the project is split into two: one project will focus on the automatic detection of buildings, and another on automatic classification of building characteristics in remotely sensed imagery. This research will explore the latter topic. For example, building characteristics (e.g. roof type and wall type) could indicate the vulnerability of buildings to natural disasters. Therefore, this research will explore the possibility of automatic classification of roof types, i.e. roof shape and roof material, using aerial imagery and detailed building data of Sint Maarten.

Classification of remotely sensed imagery is a widely investigated topic. In the past, image classification relied on image feature extraction algorithms (e.g. SIFT [Lowe, 1999], SURF [Bay et al., 2006], and ORB [Rublee et al., 2011]) to extract features which represent each image. Subsequently, these representations were encoded using coding algorithms (e.g. Bag Of visual Words (BOW) [Sivic and Zisserman, 2003]) and used to train a classifier such as Support Vector Machines (SVMs). Recently, CNNs have risen in popularity and have become the state-of-the-art due to its superior performance [Li et al., 2018]. There has been little research regarding the implementation of CNNs for the classification of roof types. Partovi et al. [2017] implemented a CNN to classify various roof shapes (e.g. flat, hip, pyramid) of buildings in the city of Munich (Germany) using 50cm spatial resolution imagery. Castagno and Atkins [2018a] combined features extracted from aerial imagery using a CNN and LIDAR[4] data. The model is trained on aerial imagery (resolution between 30-70cm) of Witten (Germany) and Manhattan (New York, USA) and evaluated on a independent data set of Ann Arbor (Michigan, USA). At last, Castagno and Atkins [2018b] used CNNs on the aerial imagery of Witten (Germany) to classify roof shapes. This work differs from the others, as only spectral (non-LIDAR) data is used and also extends the classification to roof material.

Using data from Sint Maarten, this research will explore the implementation of state-of-the-art CNN architectures for the classification of the roof shape and material type of buildings. Furthermore, it will explore new strategies to further exploit existing CNN architectures for the classification of remotely sensed imagery.

[2]`https://www.missingmaps.org`
[3]`https://www.openstreetmap.org`
[4]Laser Imaging Detection And Ranging (LIDAR) data is obtained by measuring the distance between the surface and the camera using a laser

1.1 Research Questions

The main objective of this research project is to develop a model which can automatically determine the type of roofs on buildings in aerial imagery. Recently, CNNs have shown promising results in image classification tasks. Therefore, the research will explore new strategies to further exploit CNN architectures achieving state-of-the-art performance on remote sensing bench mark data sets. The main research question is as follows:

- How can convolutional features extracted from pre-trained Convolutional Neural Networks be exploited to classify remotely sensed imagery?

More specifically, the research will propose two different strategies to improve performance. First, it will investigate the use of well known feature coding algorithms, which have shown success in traditional image classification pipelines. The second strategy involves the use of Random forests. These two strategies can be formulated in the following research sub-questions:

1. Can feature encoding of convolutional features improve the classification performance of Convolutional Neural Networks?
2. Can Random Forests improve the classification performance of Convolutional Neural Networks?

Answering these two sub-questions ensures two outcomes. First, by using state-of-the-art CNN architectures as a baseline, the roof type classification problem is solved by the best models currently available. Hence, the obtained results provide a good indication of the current feasibility of classification of roof characteristics in aerial imagery. Secondly, by exploring the aforementioned strategies, this work extends to existing literature regarding the exploitation of CNN in classification of remotely sensed imagery.

1.2 Methodology

This section will describe the methodology used, the Cross Industry Standard Process for Data Mining (CRISP-DM). The CRISP-DM approach consist 6 steps: business understanding, data understanding, data preparation, modelling, evaluation, and deployment. These steps are visualised in figure 1.2.

In the first step, business understanding, the business objectives and success criteria are determined. The business objective describe what the project aims to achieve from a business perspective and the business success criteria specify when the objectives are achieved. The business objective of this work (as described in the introduction) is to develop a model which can successfully automatically determine roof types of buildings in aerial imagery.

Subsequently, the business objectives and success criteria are translated in concrete data mining objectives and success criteria. The business objective is transformed into two concrete data mining objectives: (1) classification of roof shape types and (2) classification of roof material of buildings in aerial imagery. The quality of the predictions is deemed sufficient when an accuracy of at least 85% is achieved.

Data understanding involves the collection, description, exploration, and verification of the available data sets. This work tries to achieve the objectives using data from Sint Maarten. The connection between business understanding and data understanding is clearly visible in stating the data mining objectives. Since the data used in this work only contains information of roof

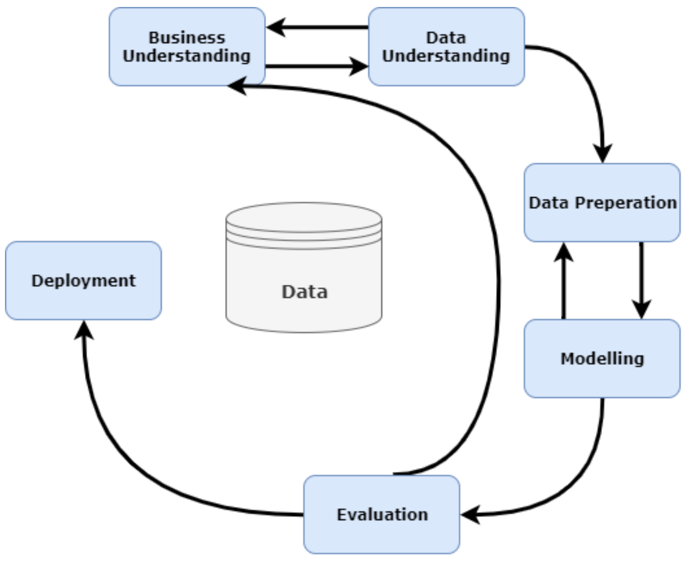


Figure 1.2: Overview of the CRISP-DM framework

shape and roof material, these aspects are the only ones that are incorporated in the data mining objectives.

The main goal of the third step, data preparation, is to prepare the available data in such a way that it can be directly used by the models in the subsequent step, modelling. The data preparation and modelling steps are interconnected, as models require particular input formats for optimal performance. Besides the shared general pre-processing steps, the image classification models used in this work require model-specific pre-processing steps, emphasising the linkage between these two steps.

In the modelling step, various models are selected and tested and the models are evaluated with regard to the data mining objectives and success criteria defined during the business understanding phase.

In the last step performed in this research project, evaluation, the results are discussed with respect to the business objectives and success criteria. If the business objective is not achieved, one can start another iteration in the CRISP-DM cycle, starting again at business understanding. If the business objectives are achieved, the models can be deployed in the business. The exact models in this work will not be deployed, as these are specifically trained on the data set of Sint Maarten. However, it will be evaluated whether similar classification pipe lines can be used in the future, including a set of recommendations.

1.3 Outline

The remainder of this work is structured as follows:

- Chapter 2 provides the theoretical background of the techniques used in this paper.
- Chapter 3 gives an overview of the data used in this work, as well as the data preparation steps executed.
- Chapter 4 describes the experimental setup which is used to answer the research questions.
- Chapter 5 presents the results of the various experiments conducted.

- Chapter 6 assesses the results of the experiments with respect to the research questions and objective.
- Chapter 7 discusses the practical implications of this work for future implementation
- Chapter 8 concludes the paper by summarising the main findings, describing the limitations, and recommending future research directions.

Chapter 2

Background Information

This chapter describes all background related material with respect to this thesis. First, the literature related to this work is summarised in section 2.1, as well as the identified research gaps. Subsequently, all theory behind the models and algorithms are explained in more detail: section 2.2 introduces the concepts behind CNNs, section 2.3 describes the four coding algorithms used in this work, and section 2.4 explains the mechanisms behind SVM and Random Forests (RF).

2.1 Related Work

Traditionally, image classification consisted of three steps: (1) local image feature extraction, (2) feature coding, and (3) classification. The first step involves the use of predefined feature extraction algorithms (e.g. SIFT) which extract local features from several patches within the image. Subsequently, the local image features are encoded by coding algorithms to generate more sparse features suitable for classification. At last, the encoded image features are fed into a classifier. In the past years, image classification problems have been dominated by CNN architectures, which significantly outperform all previous state-of-the-art benchmarks [Krizhevsky et al., 2012]. CNN architectures are typically built up from convolutional layers, pooling layers, and fully connected layers. In these architectures, the convolutional layers and pooling layers are responsible for image feature extraction, and the fully connected layers for classification.

The outstanding performance of CNNs in remote sensing is mainly due to the use of transfer learning: complex deep CNN architectures are pre-trained on enormous image data bases (e.g. ImageNet [Deng et al., 2009]) and used in other image classification problems. Generally, there are three approaches when using existing CNN architectures on another data set: (1) training the CNN from scratch using the target data set, (2) fine-tuning the CNN on the target data set, and (3), using the CNN directly as feature extractor in combination with another classifier. What approach achieves the best results depends on the characteristics of the target data set, and the data set the CNN was initially trained on. Generally, fine-tuning is the better alternative when the target data set is small and the initial data set is similar and fully training the network from scratch is preferable when the target data set is large and different than the initial data set [Castelluccio et al., 2015]. Moreover, no clear winner in terms of performance can be denoted comparing the different pre-trained networks, as the relative performance of pre-trained networks differs per data set [Cheng et al., 2016a, Li et al., 2018].

Despite the use of pre-trained networks, overfitting is still a problem when there is little training data. Several papers investigate so-called regularisation techniques to mitigate this problem. Data augmentation, artificially increasing the number of data samples in the data set, has proven to increase the performance of CNNs in image classification. Relatively simple transformations such as: rotating, flipping, translating, and transposing have shown to improve the classification performance of CNNs [Hu et al., 2015, Cheng et al., 2016b, Scott et al., 2017, Yu et al., 2017a,b]. More complex data augmentation techniques include generation of new samples by adding noise [Slavkovikj et al., 2015], pixel-pairing [Li et al., 2017], and generative-adversarial networks [Zhu et al., 2018].

CNNs have also been used as image feature extractors: by removing one (or more) top layer(s) of the network one can extract features which can be used as input for other classification algorithms. SVMs are commonly used in combination with CNN features. For example, In the work of Nogueira et al. [2017], CNN-SVM combinations outperformed conventional CNN architectures. Besides SVMs, other classification algorithms have been implemented successfully in combination with CNNs as well, such as Recurrent Neural Networks [Wu and Prasad, 2017], Extreme Learning Machines [Yang et al., 2018] and Global Average Pooling [Zhong et al., 2016, Alshehhi et al., 2017].

CNN classification pipelines are different from the traditional image classification pipeline described above, as no feature coding is involved after feature extraction by convolutional layers. Feature coding can, however, also be implemented in combination with CNNs. Features extracted from a convolutional layer of a CNN, so-called convolutional features, describe parts of an image. These local features are similar as the ones obtained by conventional feature extraction algorithms (e.g. SIFT); thus, can be encoded in a similar fashion. Cheng et al. [2017] extracted convolutional features from the last convolutional layers of various pre-trained networks and encoded it by the popular image coding scheme BOW [Sivic and Zisserman, 2003]. BOW maps each local feature to the nearest visual word in a code book, which can be obtained by clustering the local features. Zhou et al. [2017] used BOW and three more complex extensions (i.e. Vector of Locally Aggregated Descriptors (VLAD) [Jégou et al., 2010], Locality-constrained Linear Coding (LLC) [Wang et al., 2010] and Improved Fisher Kernel (IFK) [Perronnin et al., 2010]) to encode convolutional features for the retrieval of remotely sensed imagery. At last, Hu et al. [2015] proposed an architecture where a CNN generated the convolutions features of multiple scales of the same image. Subsequently, these multi-scale convolutional features are encoded by the aforementioned coding schemes. The alternatives including feature coding increased performance over the one without feature coding in all mentioned papers.

The major role of feature coding in the past and its current absence in CNN classification approaches, raises the question whether CNNs rendered this step obsolete. None of the mentioned literature implemented feature coding in the same way as in the conventional image classification pipeline. Furthermore, while research regarding the use of alternative classification algorithms includes more exotic classification algorithms, none of the work investigated the use of RF. This research aims to fill the identified research gaps, which could provide valuable insights in how to improve the classification performance of CNNs.

2.2 Convolutional Neural Network

The human brain consists of billions of neurons which are connected through trillions of connections. Neurons can pass information to other neurons by sending electrical signals through its connections. Together, the neurons and connections enable a human being to process information. An Artificial Neural Network (ANN) is a model which mimics the working of the human brain. ANNs are networks of artificial neurons. Inputs flow through the ANN and are transformed along the way by the weights of the connections and activation functions of the neurons. The simplest type of an ANN is the feed forward neural network, which is illustrated in figure 2.1. A feed forward neural network consists of an input layer, a number of hidden layers, and an output layer. The neurons within a feed forward neural network are connected to all the neurons in the preceding and subsequent layer. ANN are trained in a supervised manner. The network is trained by minimising a loss function using labelled training samples. The loss function indicates the difference between the predicted values by the network and the actual values. Given the loss of the network the weights of the connections are trained iteratively through back propagation and optimisation algorithms such as (stochastic) gradient descent. Back propagation calculates the gradient of the loss function with respect to the weights of the connections in the network, i.e. how the loss changes for small changes of the weights in the network.

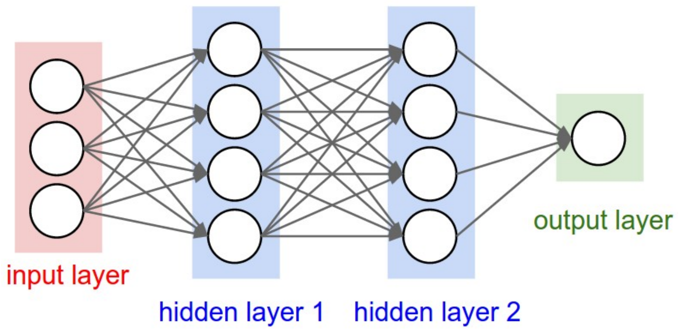


Figure 2.1: Simple feed forward neural network with two hidden layers (source: http://cs231n.github.io/neural-networks-1)

Besides feed forward neural networks, many other types of neural networks exist with each its own advantages and disadvantages. CNN is a type neural network often implemented in computer vision applications. The layers within a CNN are different than conventional neural networks (e.g. feed forward neural network). Typically, the three main components of a CNN architecture are convolutional layers, pooling layers, and fully connected layers. The remaining part of this section will describe these key components in more detail.

2.2.1 Convolutional Layers

Convolutional layers differ from layers in conventional neural networks. Layers within conventional neural networks are fully connected (i.e. each node is connected to all nodes of the preceding and following layer) and each connection has its own trainable weight. Complex classification problems, such as image classification, require several layers of connected neurons to be effective. However, the number of connections increase rapidly for deep networks, requiring a large train data set and excessive amount of computation power [Castelluccio et al., 2015].

CNNs' nodes have a limited receptive field, i.e. a node is not connected to all nodes in the preceding layer. Furthermore, neurons in the same convolutional layer share weights, reducing the amount of weights to be trained. This difference can be seen in figure 2.2. (a) represents feed forward neural network with fully connected layers: there is a connection between each node. (b) shows the nodes in convolutional layers, which have a limited receptive field. (c) illustrates how the connections to nodes in a layer are similar to another, i.e. as they share the same weights. This example demonstrates how the number of trainable weights decreases from 15 to 3 moving from a feed forward neural network to CNN respectively.

Convolutional layers have typically nodes arranged in three dimensions: width, height, and depth. The second layer of the CNN example from figure 2.2 can be seen as a convolutional layer with height, width and depth of 3, 1, and 1, respectively. Filters move over the regions of nodes of the preceding layer and is connected to the nodes in the next layer. This process is illustrated in figure 2.3. Each filter creates a new sub-layer in the depth dimension. Stride determines the step size at which a filter moves over the input. Zero-padding adds additional zeros around the

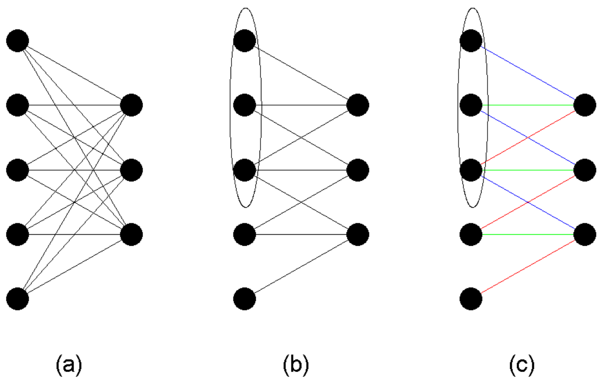


Figure 2.2: Moving from feed forward neural network to CNN: (a) feed forward neural network's fully connected nodes, (b) CNN's nodes with limited receptive field, (c) colours illustrate the weight sharing of nodes in CNN (source: Castelluccio et al. [2015])

boundaries of the input layer of the filter. By applying padding one can influence the size of the output layer.

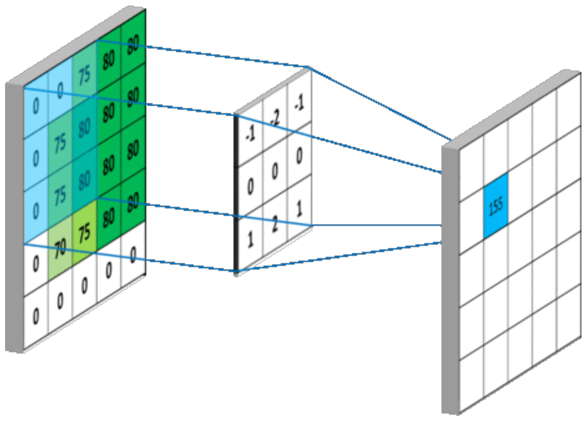


Figure 2.3: Working of filter in CNN (source: `https://mlnotebook.github.io/`)

2.2.2 Pooling Layers

Pooling layers are often placed between convolutional layers to decrease the spatial dimensions of the layers. Pooling layers combine multiple nodes from the preceding layer into a single node. Examples of pooling are max pooling and average pooling. In max and average pooling the output node receives respectively the average and maximum value of the input region (which can be a multi-dimensional region).

2.2.3 Fully Connected Layers

Convolutional and pooling layers are very good in extracting abstract image features from the input images. However, they are not suited for classification tasks. Therefore, fully connected layers are added on top of the convolutional layers. As explained, nodes in fully connected layers are connected to all nodes in the preceding and succeeding layer. The last fully connected layer is often a Softmax layer which outputs the respective normalised probabilities of an input belonging to the classes.

2.3 Feature Encoding

This section will introduce the four feature coding algorithms used in this research. Feature coding will be applied to the local features extracted from the last convolutional layer of the CNN. As described in the previous section, a convolutional layer typically has nodes arranged in three dimension: height (h), width (w), and depth (d). Features maps extracted from a convolutional layer of size $h \times w \times d$ can be flattened out into a set of feature vectors:

$$X = (x_1, ...x_N).$$

x_i denotes a d-dimensional local feature vector which represents a part of an image. The number of local features describing each image, N, is equal to $h \times w$. This conversion is visualised in figure 2.4. Each feature coding scheme will encode the local features of each image, X, into an encoded representation V.

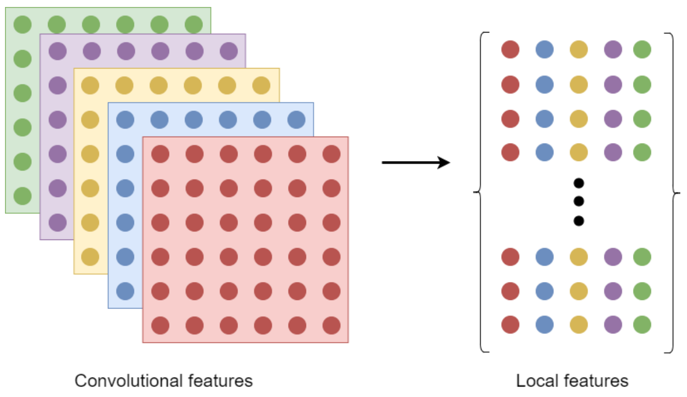


Figure 2.4: Conversion of convolutional features into a set of local features

2.3.1 Bag Of visual Words

The first step in Bag Of visual Words (BOW) [Sivic and Zisserman, 2003] coding scheme is the construction of code book $C = (c_1, ..., c_K)$ with K entries. This code book is generated by k-means clustering all local features X of (a subset of) the imagery. By quantizing each local feature x_i into a code in the code book, a frequency histogram can be constructed for local features X of size K. Finally, the frequency histogram, for local features X is normalised using L2-normalisation[1] resulting in K-dimensional encoded features $V = (v_1, ..., v_K)$.

Using BOW coding scheme has two main benefits. First, it reduces the size of the features to K. Secondly, BOW makes the model more invariant to transformations, since the location of the features are lost when constructing the histogram. On the other hand, the quantization of the local features leads to a loss of (potentially relevant) information.

[1]for a given vector $x = (x_1, ..., x_n)$ the L2 normalised equivalent is equal to: $x/\sqrt{\sum_{i=0}^{n} x_i^2}$

2.3.2 Vector of Locally Aggregated Descriptors

Just as the BOW model, Vector of Locally Aggregated Descriptors (VLAD) [Jégou et al., 2010] constructs a code book $C = (c_1, ..., c_K)$ using k-means clustering and assigning the local features X to the closest code. For each visual word c_i, VLAD accumulates the differences between cluster centre c_i and the local features x closest to that centre. Thus, the VLAD encoded features $V = (v_1, ..., v_K)$ for local features $X = (x_1, ...x_N)$ can be described as follows:

$$v_k = \sum_{i=1}^{N} q_{i,k}(x_i - c_k),$$

where $q_{i,k}$ is 1 when the i^{th} local feature x_i is assigned to cluster k, and 0 otherwise. x_i is the d-dimensional feature vector of the i^{th} local feature, and 0 otherwise. Finally, the encoded features are normalised using signed squared-root normalisation[2] and L2-normalisation resulting in $K \times d$-dimensional VLAD features V.

Just as BOW, VLAD makes the model more invariant to transformations. Although VLAD features can reduce the size of the feature, this depends more on the size of the code book. Also, less information is lost as it takes into account the magnitude of the values of the local features.

2.3.3 Locality-constrained Linear Coding

Similar to BOW and VLAD, Locality-constrained Linear Coding (LLC) [Wang et al., 2010] depends on a code book, $C = (c_1, ..., c_K)$, obtained by K-means clustering. LLC finds for each local feature x_i M-nearest neighbours from code book C, denoted as C_i. C_i has the same size had C, but only the M-nearest neighbours have non-zero values. Next, the LLC encoded features, $V = (v_1, ..., v_K)$, for local feature x_i can be obtained by minimising the following system:

$$\begin{aligned} \min_{V} \quad & \sum_{i=1}^{N} ||x_i - C_i v_i||^2 \\ \text{subject to} \quad & \mathbf{1}^T v_i = 1, i = 1, ...N. \end{aligned}$$

$\mathbf{1}$ denotes a ones vector with the same size as l_i. By minimising this function a K-dimensional LLC feature is obtained for each local feature x_i.

The obtained LLC feature is not sparse in the sense that the spatial dimensions of the feature are decreased, but that it only has m non-zero values. Further, it loses less information compared to BOW and VLAD, as it does not use vector quantization.

2.3.4 Improved Fisher Kernel

Improved Fisher Kernel (IFK) [Perronnin et al., 2010] is a improved version of the original Fisher Vector (FV) coding algorithm [Perronnin and Dance, 2007]. Fisher vector coding assumes that the probability density function of the local features $X = (x_1, ...x_N)$ is described by a Gaussian Mixture Model (GMM), $u_\lambda(X) = \sum_{j=1}^{K} w_j u_j(X)$ with K components. The parameters for u_j are given by $\lambda = (w_j, \mu_j, \Sigma_j, j = 1, ...K)$. w_j, μ_j, Σ_j denote respectively the mixture weight, mean vector, and covariance matrix of the j^{th} component of the mixture model. Assuming the local features X are all generated independently by the GMM, an image can be expressed as the the

[2]for a given vector $x = (x_1, ..., x_n)$ the signed squared-root normalised equivalent is equal to: $sign(x)/\sqrt{|x|}$

log likelihood of all extracted features:

$$G(X|\lambda) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\lambda \log \mu_\lambda(x_i).$$

The assignment of local feature x_i to Gaussian component j is denoted by $\gamma_{i,j}$:

$$\gamma_{i,j} = \frac{w_j u_j(x_i)}{\sum_{k=1}^{K} w_k u_k(x_t)},$$

where the covariance matrix Σ_j is assumed to be diagonal and the variance vector is denoted by σ_i^2 (i.e. the diagonal values of the covariance matrix). FV considers the gradients with respect to the mean μ_j, and standard deviation σ_j, which are denoted as respectively $G_{\mu,j}$ and $G_{\sigma,j}$:

$$G_{\mu,j}(X) = \frac{1}{N\sqrt{w_j}} \sum_{i=1}^{N} \gamma_{i,j} \left(\frac{x_i - \mu_j}{\sigma_j}\right),$$

$$G_{\sigma,j}(X) = \frac{1}{N\sqrt{2w_j}} \sum_{i=1}^{N} \gamma_{i,j} \left(\left(\frac{x_i - \mu_j}{\sigma_j}\right)^2 - 1\right).$$

At last, the FV features $V = (v_1, ..., v_K)$ be obtained by concatenation of $G_{\mu,j}(X)$ and $G_{\sigma,j}(X)$:

$$v_j = [G_{\mu,j}(X); G_{\sigma,j}(X)], j = 1, ..., K.$$

Hence, the final IFK representation V is $2K \times D$-dimensional.

Just as LLC, IFK main trait is that it does not quantize the vectors, and thereby losing less information compared to BOW and VLAD.

2.4 Classifiers

Conventional CNN classification architectures use fully connected connected layer with softmax activation to transform features generated by the convolutional layers into class predictions. However, other classifiers can also be used for classification. This section will describe the two classifiers used in this work: Support Vector Machine (SVM) and Random Forests (RF).

2.4.1 Support Vector Machine

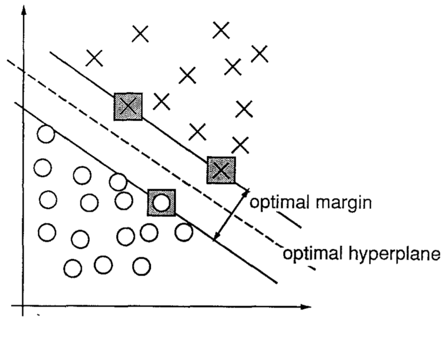


Figure 2.5: Example of a separable 2-dimensional problem (source: Cortes and Vapnik [1995])

One of the classification algorithms used in this work is the SVM algorithm introduced by Cortes and Vapnik [1995]. SVM is a supervised classification algorithm which aims to find the optimal hyper plane to separate two classes. The optimal hyper plane is chosen such that it maximises the distance (i.e. optimal margin) between the data points of both classes. Figure 2.5 illustrates the working of a SVM in a 2-dimensional problem. SVM owns its name to so-called support vectors, marked grey in figure 2.5, which are the observations that define the optimal hyper plane. SVM gained popularity in the field of remote sensing due to its ability to successfully achieve relatively high accuracy with a small training data set, makes no prior assumption on the probability distribution of the data, and is known to find the right balance between accuracy achieved on training data and its ability to generalise to new data [Mountrakis et al., 2011].

Next, a brief description will be given for SVM in its simplest form: as a binary classifier with a linear kernel. Lets define the training data as $(x_1, y_1), ...(x_N, y_N)$, where x_i represents the feature vector for observation i, and $y_i \in (-1, 1)$ the corresponding class label. The data is linearly separable if a vector w and a scalar b exist that satisfies:

$$y_i(w \cdot x_i + b) = 0, i = 1, ..., N.$$

The optimal separating hyper plane is the one which maximises the margin ρ, which can be computed as follows:

$$\rho = \frac{2}{||w||}.$$

The optimal hyper plane can then be found by minimising the following function :

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}||w||^2 \\ \text{subject to} \quad & y_i(w \cdot x_i + b) \geq 1,\ i = 1, ..., N. \end{aligned}$$

However, the above problem assumes that the data is linearly separable. However, this is often not the case or could lead to overfitting. Hence, a slack variable ξ_i and a regularisation constant C are introduced to the optimisation problem described:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(w \cdot x_i + b) \geq 1 - \xi_i,\ i = 1, ..., N, \\ & \xi_i \geq 0;\ i = 1, ..., N, \end{aligned}$$

where the regularisation parameter C denotes the penalty for each incorrect classified observation, and is a parameter to be tuned during training.

In essence, SVM is a binary classifier. However, one can adopt the SVM algorithm in multi-class classification problems. The two most used strategies are the one-vs-rest scheme, and the one-vs-one scheme. The one-vs-rest scheme trains a binary SVM per class, where the observations of the targeted class are seen as the positive observations ($y_i = 1$), and all other classes as the negative observations ($y_i = -1$). Subsequently, the class with highest predicted value (the largest margin) is selected. The one-vs-one scheme trains a SVM per pair of classes, resulting in $K(K - 1)/2$ classifiers for K classes. All SVMs estimate the class label given an observation, and the most predicted class is selected.

2.4.2 Random Forests

The other classification algorithm used in this paper is RF [Breiman, 2001]. Just like SVM, RF is a supervised non-parametric classification algorithm. The RF classifier consist of an ensemble of decision trees, which are trained on subsets of the original train data set. RF has gained traction in the remote sensing community due to its ability to deal with high dimensional data, insensitivity to data quality and easy to use with respect to parameters settings [Belgiu and Drăguţ, 2016].

Lets denote the features of the train data set as $X = (x_1, ..., x_N)$, and the corresponding class labels as $Y = (y_1, ..., y_N)$. Further, each observation x_i contains M variables. The first step in the RF algorithm is setting the number of decision trees, T. Next, T subsets are created by sampling (with replacement) N observations from the train data set (X, Y), denoted by $(X_t, Y_t),\ t = 1, ..., T$. Subsequently, each tree t in the forest is trained separately on sub set (X_t, Y_t).

Training the trees in the forest, also referred to as tree growing, can be done using a slightly adapted version of the CART algorithm [Breiman et al., 1984]. For each tree, the tree growing process starts at the root node containing all samples in subset X_t, Y_t. Next, a selection of $S < M$ variables from all M variables in X_t is randomly drawn (in the original CART algorithm this step is skipped). For all possible binary splits in each of the selected variables, the splitting criterion is computed. The splitting criterion in the CART algorithm is the Gini coefficient. The Gini coefficient of a child node j is defined as follows:

$$Gini_j = 1 - \sum_{i=1}^{K} p_i^2,$$

where K denotes the number of classes in the classification problem. p_i denotes the relative frequency of class i. However, as each split consists of two nodes, the Gini coefficient of the split is the weighted average of the Gini coefficients at both nodes.

$$Gini_{split} = \frac{n_1}{n} Gini_1 + \frac{n_2}{n} Gini_2.$$

n_j $(j = 1, 2)$ refers to the number of observations at child node j and n to the amount of observations at the parent node (note $n_1 + n_2 = n$). Subsequently, the split is chosen which minimises $Gini_{split}$. For the obtained child nodes, this process keeps repeating itself until the nodes (the last child node) are pure, i.e. only observations from one class are left at the node, or another stopping criteria is reached.

Chapter 3

Descriptions of Data Sets

A total of three data sets will be used in this work. Two data sets are obtained from aerial imagery of Sint Maarten for classification of respectively the roof shape and the roof material of buildings. The other data set is the UC-Merced land-use data set. The UC-Merced land-use data set is frequently used in other works, and will serve as a benchmark. Section 3.1 outlines the data sources and necessary steps for the generation of the Sint Maarten data sets. The UC-Merced data set is presented in section 3.2. At last, section 3.3 provides more information regarding data pre-processing and augmentation.

3.1 Sint Maarten

This section summarises all relevant information regarding the data sets of Sint Maarten. First, it will describe the data sources from which the raw data is obtained. Next, a description of the final data sets is presented, including a breakdown with regards to class labels and samples of the imagery.

3.1.1 Data Sources

IGN France obtained aerial imagery covering (almost) all of Sint Maarten in February 2017. This imagery is available as a georeferenced orthoimage through OpenAerialMap[1], a platform for open source aerial imagery. All data on OpenAerialMap can be used under the CC-BY4.0[2] license. Orthoimagery is generated by geometrically correcting the original images for camera tilt, terrain relief, and lens distortion. By doing so, everything in an orthoimage looks like it is viewed from directly above, i.e. with an angle of exactly 90 degrees. The orthoimage has a spatial resolution of 20 cm, i.e. each pixel in the image covers an area of 20 cm by 20 cm on the ground. Additionally, the orthoimage is georeferenced, meaning that the image can be related to actual coordinates. Figure 3.1 shows a part of the orthoimage.

OpenStreetMap[3] is an open source project seeking to create a freely accessibly map data base covering the entire world. The data is obtained by the effort of many volunteers and the contributions of the holders of licensed data sets. The data can be used under the ODbL[4] license. OpenStreetMap includes map data of Sint Maarten such as the location and outlines of buildings

[1]http://openaerialmap.org
[2]https://creativecommons.org/licenses/by/4.0
[3]https://www.openstreetmap.org
[4]https://opendatacommons.org/licenses/odbl

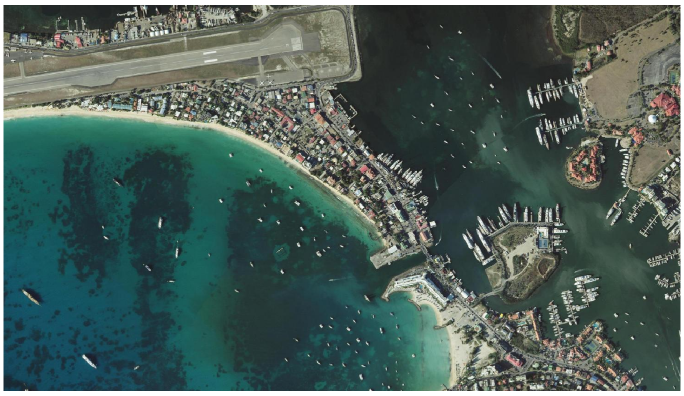


Figure 3.1: Example of aerial imagery of Sint Maarten by IGN France (source: `http://openaerialmap.org`)

and roads. The outline data of Sint Maarten from OpenStreetMap is based on the same aerial imagery used in this paper. Therefore, (almost) no difference exists between the outlines in the OpenStreetMap data and the actual buildings in the aerial imagery. Figure 3.2 shows a part of the OpenStreetMap data.

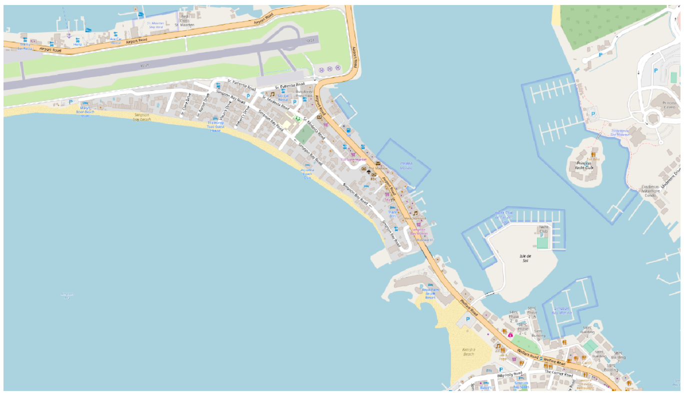


Figure 3.2: Example of OpenStreetMap data (source: `https://www.openstreetmap.org`)

Additionally, OpenStreetMap includes additional attributes of individual buildings. The attributes relevant for this project are the roof shape and roof material of each building. The roof shape is labelled as either flat or hipped. The roof material is denoted as either concrete, metal, or roof tiles.

3.1.2 Description

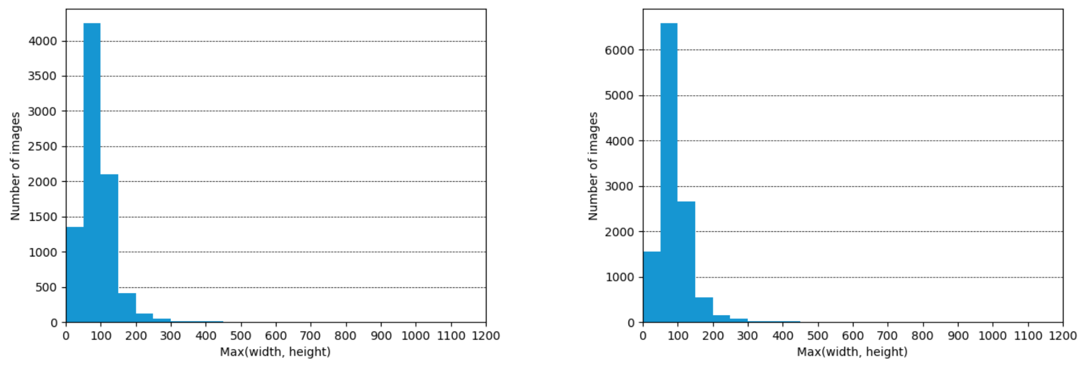
The OpenStreetMap data is used to clip (cut out) the individual buildings from the from the orthoimagery. In total 30,660 buildings are clipped from the orthoimagery. The buildings for which no roof material or roof shape data is available, are excluded from the data set.

The final number of images per data set is 8,349 and 11,661 for respectively the roof shape data set and roof material data set. Table 3.1 shows a break-down of the roof shape and roof material labels of all the buildings. The data sets are moderately imbalanced as the relative frequencies of the classes in the roof shape and roof material data set are respectively 35/65% (hipped/flat), and 58/14/28% (metal/tiles/concrete).

	Material				
Shape	metal	tiles	concrete	NA	total
hipped	1028	1027	90	790	2935
flat	1824	130	2609	851	5414
NA	3938	501	514	17358	22311
total	6790	1658	3213	18999	30660

Table 3.1: Breakdown of roof type labels.

Differences exist in the size of the buildings and therefore also the size of the images differ. Figure 3.3 shows the frequency distribution of the maximum size (i.e. max(width, height)) of the images in the roof shape and roof material data set. A number of images (< 20) are greater than 500 pixels in width or height.

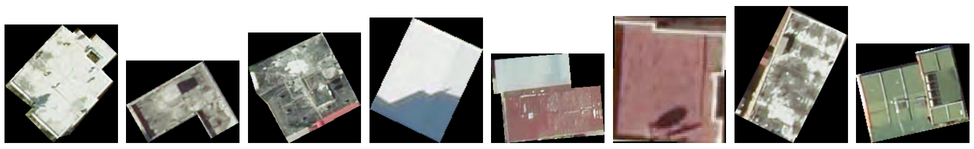


(a) Shape (b) Material

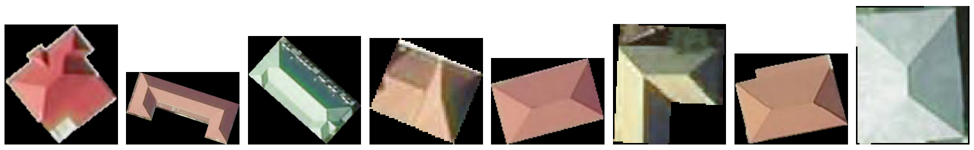
Figure 3.3: Frequency distribution of maximum size of images in (a) the roof shape data, and (b) the roof material data set

Examples of the images are shown in figure 3.4 and figure 3.5 for respectively the roof shapes and roof materials. Taking a closer look at these images, a clear distinction can be made between flat and hipped roofs. However, this is not as easy for the roof materials: the classes metal and roof tiles are sometimes very similar, which makes differentiating between the two more challenging.

Hence, a remark must be made regarding the quality of the data of OpenStreetMap. The quality of the data describes to what level the building labels from OpenStreetMap matches the actual characteristics of the buildings. The labels are obtained by the efforts of many volunteers, who manually labelled the buildings based on the aerial imagery described in the previous section. For untrained volunteers, it might be difficult to determine the correct building characteristics based only on the aerial imagery and some of the buildings might not be correctly labelled. Incorrectly labelled buildings might result in a worse performance of the models and incorrect evaluation. It is highly likely that some buildings are incorrectly labelled, given the similarity in the roof tiles and metal class in the roof material data set. However, the size of the data set and the time constraints of this project make it difficult to validate the labels of OpenStreetMap. Moreover, additional information (e.g. higher quality imagery, local confirmation) is needed to actually validate the quality of the labels. Therefore, the quality of the labels is not checked. This uncertainty with respect to the class labels must be taken into account when discussing the results.

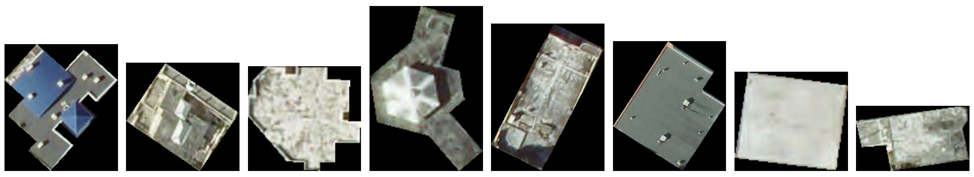


(a) Flat

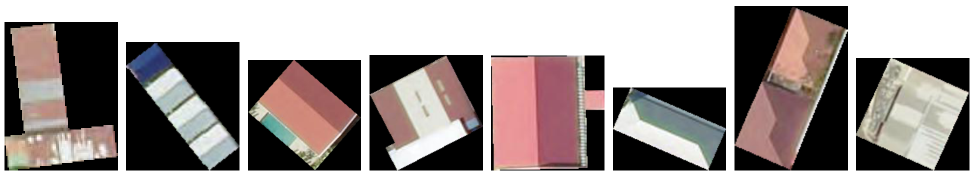


(b) Hipped

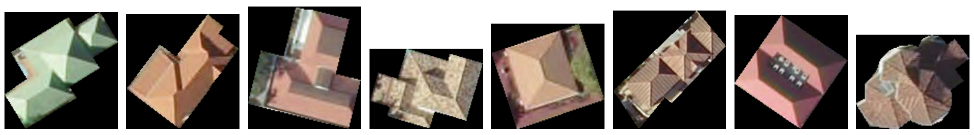
Figure 3.4: Examples of buildings per roof shape type



(a) Concrete



(b) Metal



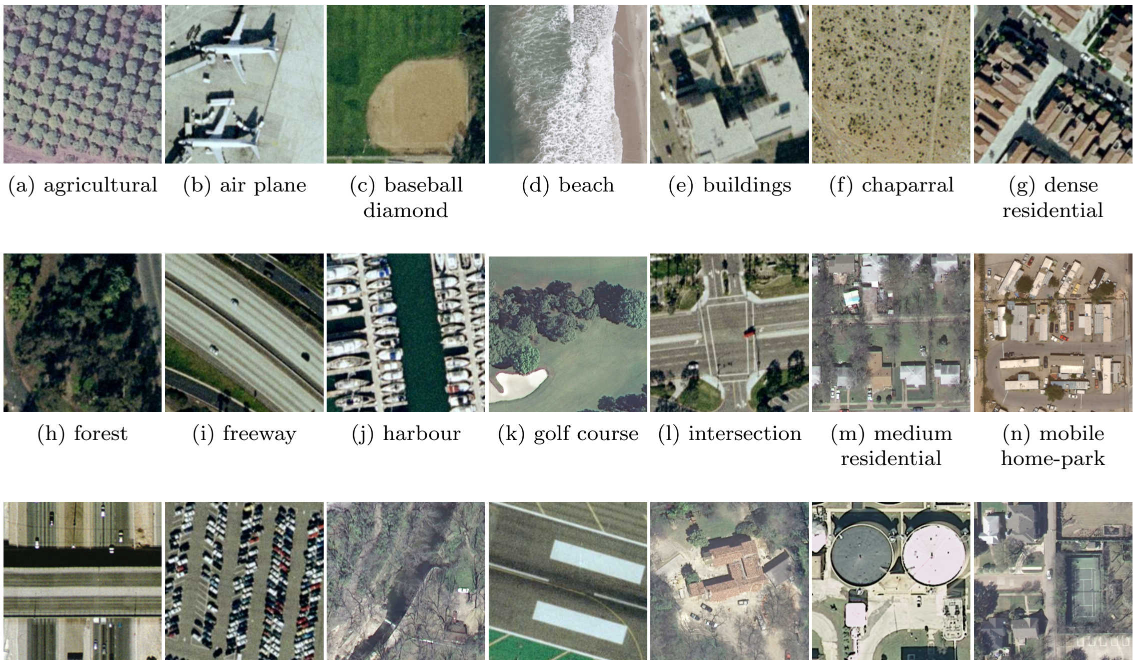
(c) Roof tiles

Figure 3.5: Examples of buildings per roof material type

3.2 UC-Merced

The UC-Merced data set [Yang and Newsam, 2010] contains aerial imagery from 21 classes. The data set consists of 100 images per class, and each image has a size of 256×256 pixels. The spatial resolution of each image is 30cm. Figure 3.6 shows for each of the classes an example image. The UC-Merced is a challenging data set; as there are relatively few images per class, and some of the classes are very similar (i.e. dense residential and medium residential).

In this work, the UC-Merced data set is used as a benchmark data set. There are several benefits in adding a benchmark data set to the study. A comparison can be made with regards to performance, as other works used the exact same data set. Additionally, as the models in this work are similar to the ones in other literature, they should achieve similar results. Mismatching results could indicate incorrect implementation of the models. Furthermore, image classification models tend to behave differently on different data sets. Thus, the addition of another data set improves the quality of the conclusions.



(a) agricultural (b) air plane (c) baseball diamond (d) beach (e) buildings (f) chaparral (g) dense residential

(h) forest (i) freeway (j) harbour (k) golf course (l) intersection (m) medium residential (n) mobile home-park

(o) overpass (p) parking lot (q) river (r) runway (s) sparse-residential (t) storage tanks (u) tennis court

Figure 3.6: Examples of UC-Merced data set

3.3 Data Preparation & Augmentation

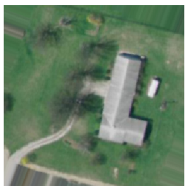
It is important that the images are of the same input format as the images on which the network is trained on. For each network, the images must be re-sized to a pre-defined size (e.g. 224 × 224 pixels) and certain pre-processing steps must be done (chapter 4 will discuss the required image sizes and pre-processing steps for the various models).

Data augmentation will be used to increase the amount of train data. Only basic image transformations are applied to the original images. While many simple augmentation methods have proven to be effective, the augmentation is limited to (horizontal and vertical) flips and rotation. More augmentation techniques would not be beneficial for the alternative classifiers used in this study (i.e. RF and SVM), as the amount of physical memory is not enough to train the classifiers on large sets of augmented images. Further, instead of augmenting the data beforehand in the pre-processing phase, the data is augmented in real-time. Real-time data augmentation transforms batches of the original images just before it is used for training. This increases the amount of augmented samples seen during training CNNs, which is able to be trained with batches of images. For the alternative classifiers however, this advantage is limited, as all images have to be fed into the classifier at once, restricting the amount of images. Examples of the augmented images are shown in figure 3.7.

(a) 56 °, no, yes

(b) 151 °, no, yes

(c) -80 °, no, yes


(d) -160 °, no, no

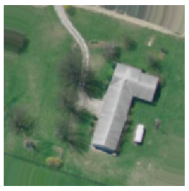

(e) -11 °, yes, no

Figure 3.7: Examples of augmented images. The sub-captions denote respectively the rotation angle, horizontal flip (yes/no), and vertical flip (yes/no).

Chapter 4

Experimental Setup

This chapter describes the implementation details of the experiments in this work. Setting up the experiments is an important step in the process of finding answers to the defined research questions. Two new classification approaches are evaluated which both utilises features extracted from a fine-tuned CNN. The first approach extracts multidimensional features from a convolutional layer of a fine-tuned CNN. Subsequently, these features are encoded using the four encoding algorithms presented in section 2.3. Finally, a linear SVM is used for classification of the images. The second approach extracts image features from the last fully connected layer and uses the classification algorithms presented in section 2.4 as final classifiers. Both approaches make use of the same pre-trained CNN, which is first fine-tuned on the target data.

The two approaches are illustrated in figure 4.1. In the remaining of this chapter, the two approaches will be referred to as scenario I and scenario II.

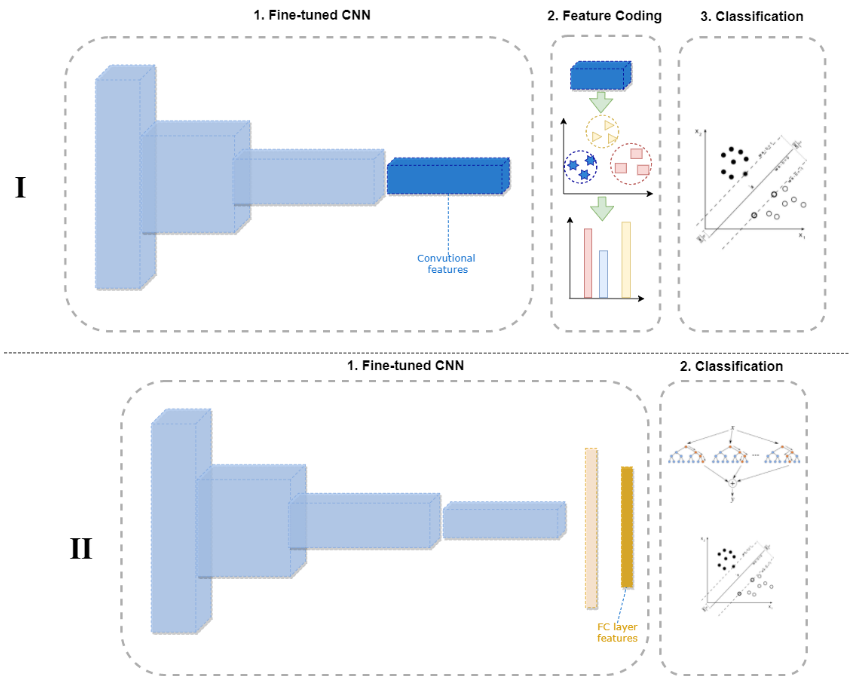


Figure 4.1: Illustration of the proposed scenarios

4.1 Experimental Protocol

The three data sets (i.e. roof shape, roof material, UC-Merced) are split into a training set and a test set. The test set is kept separated from the training process of the models (which includes fine-tuning of parameters). The way the train and test set are obtained differs for the Sint Maarten data sets and the UC-Merced data set. Two reasons for this difference: first, the size of the Sint Maarten data set is greater than the size of the UC-Merced data set. Secondly, by using the same split strategies as other literature for the UC-Merced data, the performance of the models can be evaluated against the results of other works.

The Sint Maarten data sets are stratified random split into 10% training set and 90% test set. A stratified split is used since there is class imbalance, i.e. the classes are not equally represented in the data set. Stratified splitting ensures that the relative frequency of each class is the same among all splits. Further, a relatively small proportion of the data set is taken as training set (i.e. 834 and 1116 images for respectively the roof shape and roof material data set), as in practice, such labels are unavailable and must be generated manually. Therefore, the proposed models should work with a limited amount of training samples. The process of randomly splitting the data sets, training, and testing the models is repeated 10 times. This ensures a more reliable estimate of the performance of the models.

Following the same approach as Hu et al. [2015], Zhou et al. [2017] and Nogueira et al. [2017] a stratified 5-fold cross validation strategy is used for the UC-Merced data set. First, the data is split in 5 stratified sets. Subsequently, the models are trained (and fine-tuned) on 4 out of the 5 folds and evaluated on the one split which was left out. The process of training and testing the models can be repeated 5 times, where every fold is used as test set exactly one time. K-fold cross-validation generates a reliable estimate utilising all data available, which is especially beneficial in scenarios with a small amount of data.

4.2 Model & Parameter Selection

Scenario I and II can be divided into three components: (fine-tuning) pre-trained CNNs, feature coding, and classification. This section will cover the selection of models and parameters for each of these components.

4.2.1 Pre-trained Convolutional Neural Networks

The fine-tuned CNN is a crucial part of the models in this work. Hence, three pre-trained CNN architectures are tested, where after one is chosen as feature extractor for scenario I and scenario II. The three networks used in this work are: VGG-16, InceptionV3, and Xception. The three networks are trained on data from ImageNet [Russakovsky et al., 2015], which contains over 14 million images and over 20,000 different object classes (e.g. chain saw, volcano, folding chair). These networks are chosen as they showed promising results on the ImageNet test data set and on various aerial imagery benchmark data sets. While selecting the networks, only networks were considered which are available directly through the deep learning framework used in this work (see section 4.4). First, VGG-16 is chosen as it is implemented in numerous remote sensing image classification papers. The inclusion of VGG-16 enables cross comparison to other literature and is a safe choice as it has proven to be successful in a wide domain op applications. The other two networks, InceptionV3 and Xception, achieved relatively high accuracy in the ImageNet challenge while keeping their respective network sizes in check. Appendix A summarises the performance and size of the available networks through the deep learning framework used in this work.

Before training the networks, the fully connected layers (at the end of the networks) are replaced by equivalent randomly initialised layers. Only the last fully connected layer is replaced by a layer which size equals the amount of classes in the data set (i.e. 2 for roof shape, 3 for roof material and 21 for UC-Merced). Further, a number of layers are frozen (i.e. weights cannot be trained), which speeds up the training process significantly and reduces the memory consumption. During fine-tuning the networks, 30% of the train data set is used as validation set and only the network is saved which achieves the best accuracy on the validation data set. This measure prevents potential diminished results due to overfitting on the training data. Further, one iteration of training is defined as a full cycle of training over all images in the training data set. The batch size during training (the amount of images fed to the network at once) is set to 16. While a higher batch size might have been preferable, 16 was the highest batch size possible without causing memory shortages. Before feeding the batches of images to the networks, the images are pre-processed in same way as the original images the networks are trained on. Stochastic Gradient Descent is used as optimiser for all models with cross entropy loss as loss function. Preliminary experimenting showed that passing weights to the the loss function during training slightly increased the accuracy on the training set (see appendix B for more detailed results). Hence, in training the Sint Maarten data sets weights are set equal to inverse of the relative frequency of each class. Since the experiments are repeated a number of times, training the CNNs is already a very time consuming process. Hence, due to the computational and time limitations, no further hyper-parameter optimisation is performed. After training the networks, only one pre-trained network is used as feature extractor for the proposed scenarios. Taking all networks in consideration would take too much time as all steps have to be repeated for each model.

VGG-16

The VGG-16 network was introduced in the work of [Simonyan and Zisserman, 2014]. It placed second in the ImageNet challenge 2014 in the classification category and first in the object detection category. The network is 23 layers deep, which mainly are convolutional and pooling layers. The convolutional and pooling layers are structured in 5 blocks, where 2 or 3 convolutional layers are followed by one pooling layer. A summary of the full VGG-16 architecture is shown in appendix C.1. When fine-tuning the network, the first 3 blocks (i.e. the first 10 layers) are frozen. The network is trained for 60 iterations, with a learning rate of 0.001. After 15 train iterations, the network is saved which achieves the highest classification accuracy on the validation data set. Before feeding the images into the network, the images are re-sized to 224×224. Further, VGG-16 requires the input channels to be ordered as Blue Green Red (instead of the conventional order Red Green Blue) and the mean pixel of the images used to train VGG-16 (from ImageNet) is subtracted from all images.

InceptionV3

InceptionV3 [Szegedy et al., 2016] is the third revision of the Inception CNN architectures (also known as GoogLeNet) and came in second in the ImageNet classification challenge in 2015. InceptionV3 is the deepest network used in this work, with a total of 314 layers. Despite the fact that InceptionV3 is much deeper than VGG-16, it has less (trainable) parameters. The network is made up of 11 so-called inception blocks. In a inception block, a single input is connected to several convolutional layers in parallel. After the convolutional layers in the inception block transformed the input, it is concatenated into a single output. A simplified example of an inception block is shown in figure 4.2. The full InceptionV3 architecture is summarised in appendix C.2. During training, the first 9 out of 11 inceptions blocks (i.e. first 249 layers) are frozen. Just as VGG-16, Inception is trained for 60 iterations and after 15 iterations the model is saved which achieves the highest classification accuracy on the validation data set. During preliminary testing the training loss of Inception converged very slowly. Hence, a learning rate of 0.01 is used during the first 15 iterations and 0.001 during the remaining 45 iterations. The required input size of the images for

InceptionV3 is 299 × 299 pixels and all pixel values are normalised between -1 and 1.

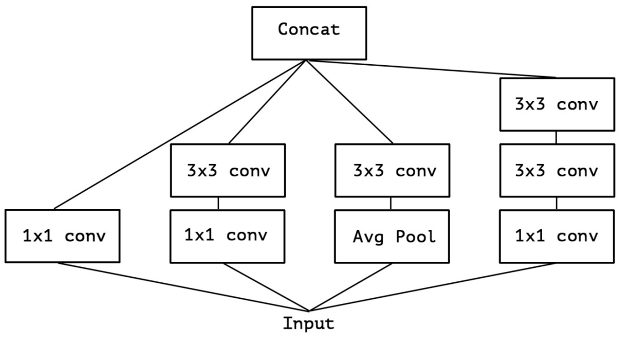


Figure 4.2: Simplified example of an inception block (source: Chollet [2017])

Xception

The third pre-trained network used is the Xception CNN [Chollet, 2017]. Of the three networks used in this paper, Xception achieved the highest classification accuracy on the ImageNet data set. The network consists of 134 layers. Xception makes use of blocks similar to InceptionV3's inception blocks, denotes as extreme inception blocks (hence its name). The filters within normal convolutional layers map the correlation between the pixels across all input channels (also referred to as depth in section 2.2). Xception modules make use of depth-wise separable convolution, i.e. a filter is applied to each channel separately. A summary of the complete Xception network is shown in appendix C.3. During training, the first 10 out of 14 xceptions blocks (i.e. first 95 layers) are frozen. Preliminary tests showed that the model's loss converged relatively slowly compared to the other models. Therefore, Xception is trained for 100 iterations; 30 iterations with training rate of 0.01, and 70 with a training rate of 0.001. Again, after the initial 30 iterations the model with the highest classification accuracy on the validation set is saved to counter overfitting. The required input size and pre-processing steps for Xception are the same as the ones for InceptionV3, respectively 299 × 299 pixels and normalisation between -1 and 1.

4.2.2 Encoding

In scenario I, convolutional features are encoded using the feature coding algorithms presented in section 2.3. BOW, VLAD, and LLC depend on K-means clustering for the construction of a codebook, and IFK uses a Gaussian Mixture Model to encode the features. Due to the computation time needed to encode the features, no parameters are tuned in this step. Thus, the parameters are set based on previous works. Hu et al. [2015] set the number of clusters for BOW, VLAD, and LLC to respectively 1000, 100 and 10000, and the number of components in the GMM for IFK to 100. Moreover, the same parameters are used for BOW, VLAD, IFK by Zhou et al. [2017]. Except for LLC, the same parameters are used in this work. The number of clusters in LLC is restricted by computational limitations, as a high number of clusters leads to very high dimensional LLC features. Hence, the number of clusters for the LLC features is also set to 100. Before encoding, the extracted convolutional features are normalised using L2-normalisation. Due to the limitation in physical memory and to enhance computation time, the K-means clusters and the GMM are fit on the original (non-augmented) training images only. Furthermore, the SVM is trained on a augmented data set which is only two times the size of the original training data (i.e. each original training image is included twice).

4.2.3 Classification

The last step in scenario I and scenario II is classification. Both make use of a SVM with the same parameter settings, despite the fact that both scenarios are structured differently. Scenario II also makes use of a RF classifier. In scenario I, the classifiers are trained on the coded features. In scenario II, the classifiers are trained on the features extracted from the last fully connected layer. Just as the SVM in scenario I, the RF and SVM are only trained on an augmented data set which is two times the size of the data set.

Support Vector Machine

SVM is widely implemented in combination with CNNs. Despite the fact that SVM are able to use various non-linear kernels, mainly simple linear SVMs are used in combination with features extracted from CNNs. In this work the same parameters are used as in Nogueira et al. [2017], which uses a linear SVM with regularisation parameter $C = 1$.

Random Forest

The RF algorithm has several parameters that can be set during training. Parameters of RF include the number of trees in the forest and several stopping criteria parameters, which influence the depth of the trees. In this work 4 parameters are tuned for the RF classifier: number of trees, minimum samples required at split, minimum samples required at each child node, maximum features, and maximum depth of the tree. The distributions for said parameters are shown in table 4.1

Parameter	**Values**
Number of trees	[100, 200, 400, 800, 1600]
Minimum samples at split	[2, 5, 10]
minimum samples at leaf	[1, 2, 4]
maximum features	[sqrt, log2]
Max depth	[20, 40, 60, 80, 100, None]

Table 4.1: Distributions of the selected parameters settings used in the Random Search

Since each experiment must be repeated 10 times for three data sets, a grid search (i.e. testing all possible combinations) over all 540 parameter combinations is unfeasible. Hence, another parameter optimisation strategy is used, called Random Search. Random Search [Bergstra and Bengio, 2012] samples a certain number of parameter settings of a distribution of parameters and evaluates the selected parameters using K-fold cross validation. Bergstra and Bengio showed that Random Search with roughly 60 randomly drawn parameter configurations found models performing as good as using grid search. Furthermore, when giving Random Search as much computational resources as grid search, Random Search is able to achieve better results as it is able to traverse a larger parameter setting space.

In this work, the sampled parameters are evaluated using cross validation with 3 folds, where in each fold one set is used for validation. The parameter configuration achieving the highest average accuracy over the 3 folds is used to train a RF on the complete train set. In the experiments the number of iterations of the random search (the number of configurations drawn) is set to 60. A total of 180 (3 folds multiplied by 60 parameter settings) random forests are trained during parameter tuning, which is three times less than the amount of random forest trained in grid search (without cross validation).

4.3 Background on Evaluation Metrics

This section will discuss the performance metrics and statistical tests used to evaluate the models. Performance metrics are used to measure the performance classification models. Five performance metrics are used in this work: accuracy, precision, recall, f1-score and kappa. To compare the performance metrics, two statistical tests are introduced: the Friedman test and the Wilcoxon test.

4.3.1 Performance Metrics

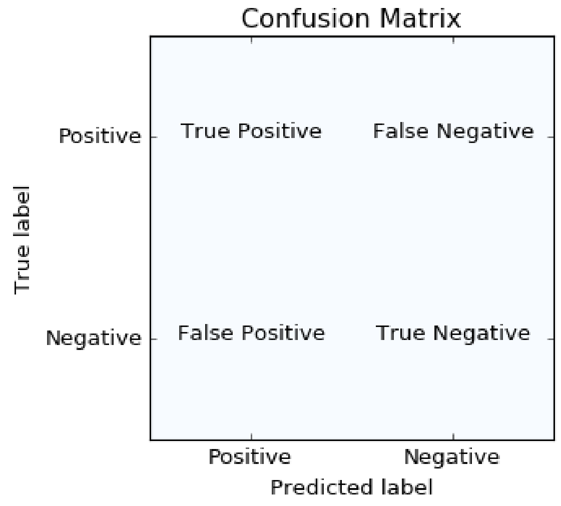


Figure 4.3: Confusion matrix of a binary classification problem

In a binary classification problem, a prediction can be either a True Positive (TP), False Positive (FP), True Negative (TN), or False Negative (FN). Where TP denotes the cases which are positive and predicted as positive, FP the cases which are positive and predicted as negative, TN the cases which are negative and predicted as negative, and FN the cases which are negative and predicted positive. These type of predictions are visualised in figure 4.3.

The simplest classification performance measurement is accuracy, which is defined as the ratio of the number of correctly predicted cases to the total number of cases. Then, using the terminology introduced, accuracy A is given by the equation:

$$A = \frac{TP + TN}{TP + FP + TN + FN}.$$

Accuracy is easy to understand and can be used for both binary and multi-class classification problems. However, in imbalanced data sets accuracy could give an unfair representation of classification performance. For example, in a binary classification problem where 90% of the samples are of the same class, simply assigning all cases to that class would already achieve an accuracy of 90%.

Therefore, we introduce three other metrics, which will be used to asses the per-class classification performance: precision, recall and the F1-score. Precision indicates how many of the positive predicted cases are correctly predicted, and recall expresses the fraction of all positive cases which are correctly predicted. These metrics are captured within the F1 metric, which is the harmonic mean of precision and recall. Precision (P), recall (R) and the F1-score ($F1$) are obtained by respectively:

$$P = \frac{TP}{TP + FP},$$

$$R = \frac{TP}{TP + FN},$$

and

$$F1 = 2 * \frac{P * R}{P + R}.$$

F1, recall, and precision, while valuable metrics for binary classification problems, should not be used for multi-class classification problems [Powers, 2015], as they could be biased (just as accuracy) due to class imbalances.

The third and last evaluation metric we introduce is the Cohen Kappa [Cohen, 1960], hereafter referred to as Kappa. Kappa adjusts the accuracy score for the possibility of agreement (between the predicted and actual scores) when the labels would be randomly assigned. Kappa K and expected agreement A_e can be obtained as follows:

$$K = \frac{A - A_e}{1 - A_e},$$

$$A_e = \frac{1}{n^2} \sum_{i=1}^{C} p_{i,+} p_{+,i},$$

where

$$p_{i,+} = \sum_{j=1}^{C} p_{i,j}$$

,

$$p_{i,+} = \sum_{j=1}^{C} p_{j,i}.$$

In a classification problem with C classes, $p_{i,j}$ denotes the frequency of the number of cases which true label is i and the predicted label is j. Kappa allows evaluation of models using a single metric, while being more robust to class imbalance. However, Kappa is more difficult to interpret than accuracy as it can range from -1 to 1. Arbitrary guidelines exist instructing how to interpret the Kappa scores, such as the ones by Landis and Koch [1977], McHugh [2012] and Fleiss et al. [2013], which are shown in table 4.2. The differences among these interpretations underline the ambiguity in the kappa. Moreover, the interpretation of the Kappa depends on the context of the classification problem. McHugh argues that the aforementioned interpretations might be to lenient for health research, where decisions based on the outcomes might have serious ramifications, and proposes a more strict interpretation for health research.

4.3.2 Statistical Tests

The introduced evaluation metrics are computed for all the models on multiple samples of the data. Hence, statistical tests must indicate whether the metrics among models actually differs. Often, paired t-test or repeated ANOVA are used to compare the performance among multiple classifiers. However, these approaches assume that the difference in the metrics between multiple models is distributed normally and homogeneity of variance, which could hurt the robustness of the comparison. Therefore, Demšar [2006] recommends to use non-parametric (i.e. no assumption on underlying data distribution) statistical tests to compare the performance among classifiers.

Landis and Koch [1977]		McHugh [2012]		Fleiss et al. [2013]	
Kappa	Level of agreement	Kappa	Level of agreement	Kappa	Level of agreement
<0	Poor	<0.20	None	<0.40	Poor
0.00-0.20	Slight	0.21-0.39	Minimal	0.40-0.75	Fair to good
0.21-0.40	Fair	0.40-0.59	Weak	0.75-1.00	Excellent
0.41-0.60	Moderate	0.60-0.79	Moderate		
0.61-0.80	Substantial	0.80-0.90	Strong		
0.81-1.00	Almost Perfect	0.90-1.00	Almost Perfect		

Table 4.2: Interpretations of the Kappa score by Landis and Koch [1977], McHugh [2012] and Fleiss et al. [2013]

Demšar advises to use the Wilcoxon signed-ranked [Wilcoxon, 1945] to compare of pairs of classifiers, and the Friedman test [Friedman, 1937] for three or more classifiers.

Just as paired t-tests, the Wilcoxon test assumes that the data is paired (i.e. the classifiers are tested on the same samples) and sampled randomly. Further, the Wilcoxon tests assume continuous dependent variables and symmetrically distributed differences between the performance of the two classifiers. The Wilcoxon test computes a statistic T_w for a set of differences $d = (d_1, ..., D_N)$, where N is the number of samples on which the two classifiers are tested. d_i is the difference between the scores of the two classifiers on the i^{th} sample. Next, the differences d are ranked (highest difference is assigned 1, second highest 2, etc.) based on their absolute value (i.e. $|d_i|$). In case of ties between differences, the average rank is assigned. Subsequently, the ranks for which d_i is positive (i.e. classifier 1 outperforms classifier 2) are summed, and the ranks for which d_i is negative (i.e. classifier 2 outperforms classifier 1) are summed, denoted by respectively S^+ and S^-, where the ranks for which d_i is 0 are divided equally among S^+ and S^-:

$$S^+ = \sum_{\forall d_i > 0} rank(d_i) + \frac{1}{2} \sum_{\forall d_i = 0} rank(d_i),$$

$$S^- = \sum_{\forall d_i < 0} rank(d_i) + \frac{1}{2} \sum_{\forall d_i = 0} rank(d_i).$$

The test statistic T_w is equal to the minimum of the two sums $T_w = \min(S^+, S^-)$. Given T_w and N ,the $p_{wilcoxon}$-value can be found, indicating the significance of the difference between the two classifiers.

The Friedman test ranks the classifiers for each data set independently. The classifier with the highest score gets rank 1, the second highest rank 2, etc.. Similar to the Wilcoxon test, tied performances will be assigned the average rank. Lets denote the rank of classifier $j \in (1, ..., K)$ on sample $i \in (1, ..., N)$ as r_i^j, and the average rank of classifier j as $\bar{R_j}$:

$$\bar{R_j} = \frac{1}{N} \sum_{i=1}^{N} r_i^j.$$

Then, the Friedman test statistic T_F is obtained as follows:

$$t_F = \frac{12N}{K(K+1)} \Big(\sum_{j=1}^{K} \bar{R_j}^2 - \frac{K(K+1)^2}{4} \Big).$$

Subsequently, for a given T_F, K and N a $p_{friedman}$-value can be computed.

4.4 Implementation Details

Training a CNN is a computationally expensive process and can be quite time-consuming. However, CNN operations (training and prediction) can be significantly accelerated using GPUs. In this work, all experiments are performed on Google Colaboratory[1], which provides a GPU-accelerated Python environment accessible for research and education for up to 12 hours in a single session. Google Colaboratory makes use of a Tesla K80 GPU with 12 GB of memory, a single-core 2.3Ghz Intel Xeon CPU, and 12 GB of RAM. All data pre-processing is performed on a machine with a quad-core 1.8Ghz Intel i7-8550U CPU and 16 GB of RAM.

QGIS [QGIS Development Team, 2009], an open-source geographical information system, is used to pre-process the OpenStreetMap data (data and feature selection). The individual buildings are clipped from the orthoimagery of Sint Maarten using the GDAL [GDAL/OGR contributors, 2018] package for Python.

Using the Google Colaboratory platform restricts the choice of deep learning framework. Hence, for all deep learning performed in this work, the Keras module [Chollet et al., 2015] is used with TensorFlow [Abadi et al., 2015] back-end. Keras provides a high-level interface for several deep learning libraries, such as TensorFlow. The encoding algorithms are implemented using a combination of mathematical and machine learning packages, i.e. SciPy [Jones et al., 2001], NumPy [Oliphant, 2006], and Scikit-learn [Pedregosa et al., 2011]. Scikit-learn also provides the implementations of the classification algorithms used in this work.

All code to run the experiments presented in this work is publicly available[2].

[1] `https://colab.research.google.com/`

[2] `https://github.com/bartvandriel/thesis`

Chapter 5

Results

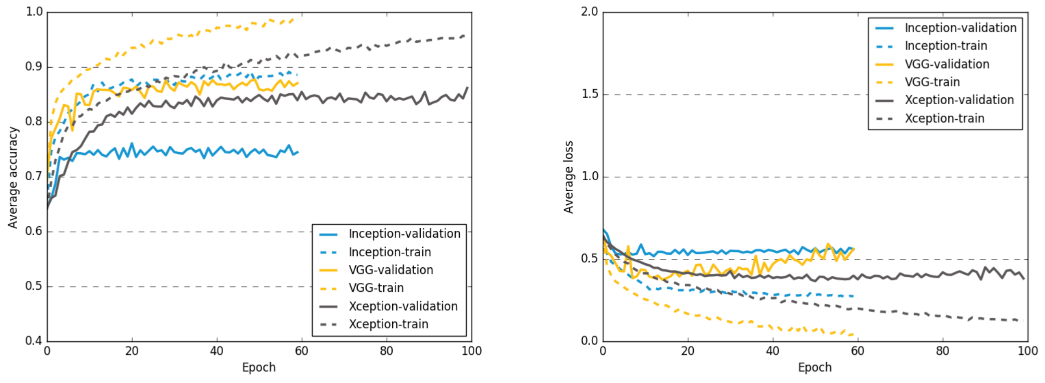
This chapter presents the results obtained by the experiments described in the previous chapter. Section 5.1 summarises the training process of the pre-trained networks and the classification results. Based on these results, one network is chosen as feature extractor in scenario I and scenario II. Section 5.2 and section 5.3 present respectively the results for scenario I and scenario II. A complete overview of all performance metrics of all models for all three data sets is shown in appendix D.

5.1 Pre-trained networks

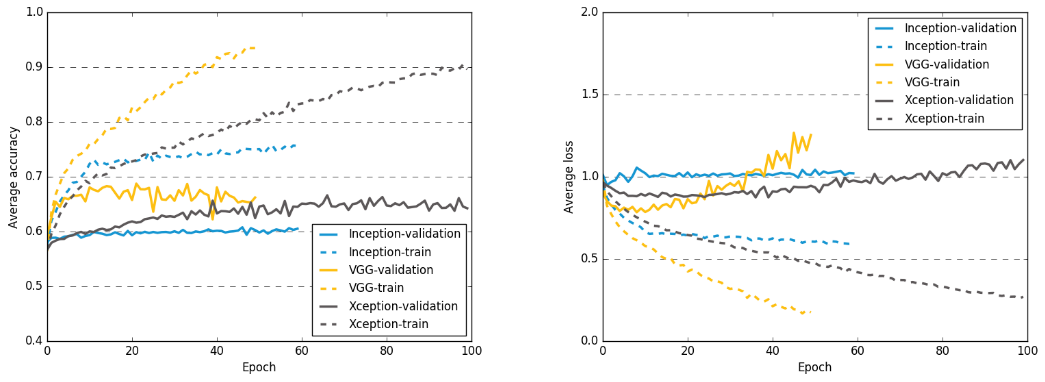
This section presents the results of the three fine-tuned networks. As explained in the previous chapter, the network with the best overall performance will be used as feature extractor for the proposed scenarios. Figure 5.1 shows the mean accuracy and loss on the train (solid lines) and validation set (dashed lines) during the training process for: (a) the roof shape data set, (b) the roof material data, and (c) the UC-Merced data set.

The behaviour of the validation and training loss during training is similar among the data sets. For all data sets, the training loss of VGG-16 converged faster than the other networks, despite the lower learning rate in the first epochs. For the roof shape and roof material data set, the networks tend to overfit on the train data, as the validation loss starts to increase after 20-40 epochs (while training loss is still decreasing). VGG-16 seems to achieve a slightly lower loss over all epochs, closely followed by the Xception network. The mean accuracy of the models during training shows similar patterns to the mean loss. VGG-16 tends to achieve the highest mean accuracy on the validation set on the roof shape and roof material data set. For the UC-Merced data set, Xception achieves similar classification accuracy compared to VGG-16. However, Xception tends to learn the material slower than the VGG-16 network. Further, InceptionV3 converges to the worst loss and accuracy on the training and validation set. On the roof material data set, during training of InceptionV3, the validation loss and accuracy did not converge at all.

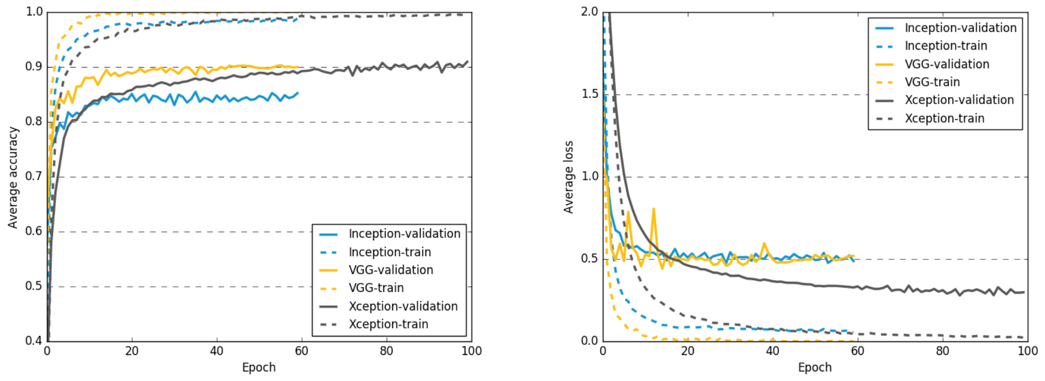
The mean and standard deviations of the accuracy and Kappa score on the data sets are shown in respectively table 5.1 and table 5.2. The difference between the accuracy and Kappa scores scores among the three networks is significant ($p_{friedman} < 0.01$ and all $p_{wilcoxon} < 0.01$) for the roof shape and roof material data set. However, for the UC-Merced data the differences are not significant at a 5% confidence level ($p_{friedman} = 0.056$). At a 10% confidence level the differences between Inception and the other models are significant, but there is still no significant difference between VGG and Xception ($p_{wilcoxon} = 0.593$). A complete overview of the Friedman and Wilcoxon statistical tests for the Kappa and accuracy scores are shown in appendix E.1. VGG-16



(a) Roof shape



(b) Roof material



(c) UC-Merced

Figure 5.1: Mean loss and accuracy during fine-tuning of the pre-trained networks on (a) the roof shape data set, (b) the roof material data set, and (c) the UC-Merced data set.

achieves the highest accuracy and Kappa scores on the roof shape and roof material data, closely followed by Xception. On the UC-Merced data, the accuracy and Kappa scores are similar for VGG-16 and Xception (i.e. no significant difference). On all data sets, InceptionV3 achieves the lowest mean accuracy and Kappa scores. Table 5.3 shows the mean and standard deviation of the per-class F1-scores for respectively the roof shape data and roof material data. VGG-16 tend to have a higher mean per-class F1 than the other networks on all classes, indicating a better balance between precision and recall.

Overall, VGG-16 tends to yield the highest accuracy, Kappa and per-class F1-score among the data sets. Hence, it is chosen as feature extractor for the scenarios as described in chapter 4.

	VGG	**Inception**	**Xception**
shape	0.88 ± 0.01	0.75 ± 0.03	0.84 ± 0.02
material	0.69 ± 0.00	0.61 ± 0.01	0.65 ± 0.01
UC-Merced	0.90 ± 0.03	0.86 ± 0.03	0.91 ± 0.03

Table 5.1: Mean and standard deviation of the accuracy of the fine-tuned networks.

	VGG	**Inception**	**Xception**
shape	0.72 ± 0.02	0.43 ± 0.07	0.63 ± 0.06
material	0.41 ± 0.01	0.10 ± 0.04	0.27 ± 0.04
UC-Merced	0.90 ± 0.03	0.86 ± 0.04	0.90 ± 0.03

Table 5.2: Mean and standard deviation of the Kappa score of the fine-tuned networks.

		VGG	**Inception**	**Xception**
Shape	flat	0.91 ± 0.01	0.82 ± 0.02	0.89 ± 0.01
	hipped	0.81 ± 0.01	0.61 ± 0.06	0.74 ± 0.05
Material	concrete	0.58 ± 0.03	0.22 ± 0.09	0.41 ± 0.10
	metal	0.78 ± 0.01	0.75 ± 0.00	0.76 ± 0.01
	roof tiles	0.47 ± 0.04	0.03 ± 0.03	0.30 ± 0.11

Table 5.3: Mean and standard deviation of the per-class F1-score of the fine-tuned networks.

5.2 Feature coding

In this section, the results of the feature coding algorithms are presented. First, image features are extracted from the last convolutional layer of the fine-tuned VGG-16, whereafter these features are encoded using the four coding schemes presented in chapter 2. Finally, the features are used to train a linear SVM. In addition to the four coding schemes, the extracted features are also fed directly to the SVM, this approach is referred to as NONE.

Table 5.4 and table 5.2 show respectively the accuracy and Kappa scores of the models described above and VGG-16 as a base line. Overall, there is a significant difference in the performance among the models presented in the tables ($p_{friedman} < 0.01$). Further, for the roof shape and roof material data set the performance differences between most of the models' accuracy and Kappa scores are significant ($p_{wilcoxon} < 0.01$), except between VLAD and VGG-16 ($p_{wilcoxon} > 0.10$). For the UC-Merced data set, none of the differences between the coding schemes and VGG-16 is significant at a 5% confidence level ($p_{wilcoxon} > 0.05$). Only the differences between NONE and VGG-16, and, IFK and VGG-16 are significant at a 10% confidence level (respectively $p_{wilcoxon} = 0.08$ and $p_{wilcoxon} = 0.07$). A complete overview of all $p_{friedman}$ and pair-wise $p_{wilcoxon}$ values for the aforementioned models can be found in appendix E.2. On the roof shape and roof material data set, VLAD and VGG-16 achieve the highest accuracy and Kappa scores. On the UC-Merced data set, VLAD and NONE achieve the highest accuracy and Kappa scores, followed by the fine-tuned VGG-16 network and BOW. The per-class F1-scores for the roof shape and roof material data sets are shown in table 5.6. The F1-scores are in line with the corresponding accuracy and Kappa scores, VLAD and VGG-16 tend to achieve the highest per-class accuracy scores.

	NONE			**BOW**			**VLAD**		
Shape	0.86	±	0.01	0.85	±	0.01	0.87	±	0.00
Material	0.66	±	0.01	0.66	±	0.01	0.69	±	0.01
UC-merced	0.92	±	0.01	0.89	±	0.03	0.92	±	0.01

	LLC			**IFK**			**VGG**		
Shape	0.84	±	0.01	0.84	±	0.01	0.88	±	0.01
Material	0.64	±	0.01	0.65	±	0.01	0.69	±	0.00
UC-merced	0.90	±	0.02	0.86	±	0.02	0.90	±	0.03

Table 5.4: Mean and standard deviation of the accuracy of the models of scenario I

	NONE			**BOW**			**VLAD**		
Shape	0.86	±	0.01	0.85	±	0.01	0.87	±	0.00
Material	0.66	±	0.01	0.66	±	0.01	0.69	±	0.01
UC-merced	0.92	±	0.01	0.89	±	0.03	0.92	±	0.01

	LLC			**IFK**			**VGG**		
Shape	0.84	±	0.01	0.84	±	0.01	0.88	±	0.01
Material	0.64	±	0.01	0.65	±	0.01	0.69	±	0.00
UC-merced	0.90	±	0.02	0.86	±	0.02	0.90	±	0.03

Table 5.5: Mean and standard deviation of the Kappa score of the models of scenario I

		NONE			BOW			VLAD		
Shape	flat	0.90	±	0.01	0.89	±	0.01	0.91	±	0.00
	hipped	0.79	±	0.02	0.78	±	0.02	0.81	±	0.01
		LLC			**IFK**			**VGG**		
	flat	0.88	±	0.01	0.88	±	0.01	0.91	±	0.01
	hipped	0.77	±	0.01	0.73	±	0.02	0.81	±	0.01
		NONE			**BOW**			**VLAD**		
Material	concrete	0.56	±	0.03	0.55	±	0.02	0.59	±	0.02
	metal	0.74	±	0.01	0.74	±	0.01	0.77	±	0.01
	roof tiles	0.44	±	0.02	0.44	±	0.03	0.48	±	0.03
		LLC			**IFK**			**VGG**		
	concrete	0.54	±	0.03	0.47	±	0.04	0.58	±	0.03
	metal	0.73	±	0.01	0.76	±	0.00	0.78	±	0.01
	roof tiles	0.40	±	0.02	0.30	±	0.05	0.47	±	0.04

Table 5.6: Mean and standard deviation of the per-class F1-score of the models of scenario I

5.3 Classifiers

This section presents the results of the models from scenario II. The image features are extracted from the last fully connected layer of the fine-tuned VGG-16 network, and final classification is done by means of a linear SVM and a RF. The parameter tuning for the RF is done via a random search over a set of parameter settings, the final parameter settings for each fold and each data set can be found in the appendix (table F.1).

Table 5.7 and table 5.8 show the accuracy and Kappa metrics of the classifiers (and VGG as a base line) over the three data sets. On all three data sets, the differences between the mean accuracies and Kappa scores are significant when comparing RF, SVM and VGG-16 ($p_{friedman} < 0.05$). For each data set, the differences between SVM' accuracy and Kappa score, and the scores of the other models (RF and VGG-16) are significant (all $p_{wilcoxon} < 0.05$, except for the UC-Merced data set with $p_{wilcoxon} < 0.10$), and VGG-16 and RF tend to achieve equivalent results ($p_{wilcoxon} > 0.10$). A complete overview of the statistical tests is given in appendix E.3.

On the roof shape and roof material data, RF outperforms SVM. On the UC-Merced data however, SVM outperforms RF. The per-class F1-scores are shown in table 5.9 for the roof shape and roof material data. Generally, all F1-scores lie very close to each other across the different classifiers. Only for the metal class in the roof material data set, the F1-scores are considerably lower using SVM compared to the other classifiers.

	RF			SVM			VGG		
Shape	0.87	±	0.01	0.86	±	0.02	0.88	±	0.01
Material	0.70	±	0.01	0.64	±	0.02	0.69	±	0.00
UC-Merced	0.89	±	0.02	0.92	±	0.02	0.90	±	0.03

Table 5.7: Mean and standard deviation of the accuracy of the models of scenario II

	RF			SVM			VGG		
Shape	0.87	±	0.01	0.86	±	0.02	0.88	±	0.01
Material	0.70	±	0.01	0.64	±	0.02	0.69	±	0.00
UC-Merced	0.89	±	0.02	0.92	±	0.02	0.90	±	0.03

Table 5.8: Mean and standard deviation of the Kappa score of the models of scenario II

		RF			SVM			VGG		
Shape	flat	0.91	±	0.01	0.89	±	0.01	0.91	±	0.01
	hipped	0.81	±	0.01	0.79	±	0.02	0.81	±	0.01
Material	concrete	0.56	±	0.02	0.56	±	0.03	0.58	±	0.03
	metal	0.79	±	0.00	0.73	±	0.02	0.78	±	0.01
	roof tiles	0.45	±	0.03	0.45	±	0.02	0.47	±	0.04

Table 5.9: Mean and standard deviation of the per-class F1-score of the models of scenario II

Chapter 6

Discussion

In this chapter the results will be discussed with respect to the research questions introduced in chapter 1. The main research question of this work is as follows:

- How can convolutional features extracted from pre-trained Convolutional Neural Networks be exploited to classify remotely sensed imagery?

This question is very broad and other literature explored many ways how to improve the classification performance of CNNs. Hence, two gaps in the existing literature are identified, which opens a window of opportunity for improving the state-of-the-art. This work aims to fill these gaps by answering the following two research questions:

1. Can feature encoding of convolutional features improve the classification performance of Convolutional Neural Networks?
2. Can Random Forests improve the classification performance of Convolutional Neural Networks?

Besides discussing the results with respect to these research questions, this chapter takes a closer look at the best classification results of the Sint Maarten and UC-Merced data sets and provides additional insights with respect to the implementation of such models in practice.

6.1 Exploitation of CNN features

This section discusses whether the two proposed scenarios improve the classification performance compared to the base line, the fine-tuned VGG-16. Table 6.1 and table 6.2 show the difference (in %) between the various coding schemes, classifiers, and the base line. Each cell indicates the difference in mean accuracy between the model indicated by the row value and the model indicated by the column value. For instance, BOW's mean accuracy is 2.4% lower than the mean accuracy by VGG-16 for the roof shape data set. The asterisks denote respectively whether the $p_{wilcoxon}$-value of the difference between the two models is lower than respectively 0.01 (***), 0.05 (**), and 0.10 (*).

None of the feature coding schemes increase the classification accuracy significantly. Only VLAD achieves equivalent performance compared to the VGG-16 base line on all three data sets. Relatively simple feature coding algorithms, BOW and VLAD achieve the best results, while

	NONE	BOW	VLAD	LLC	IFK	VGG
NONE	-	1.0***	-1.1***	1.8***	2.6***	-1.4***
BOW	-1.0***	-	-2.2***	0.8**	1.5***	-2.4***
VLAD	1.1***	2.2***	-	3.0***	3.7***	-0.2
LLC	-1.8***	-0.8**	-3.0***	-	0.7**	-3.2***
IFK	-2.6***	-1.5***	-3.7***	-0.7**	-	-3.9***
VGG	1.4***	2.4***	0.2	3.2***	3.9***	-

(a) Shape

	NONE	BOW	VLAD	LLC	IFK	VGG
NONE	-	-0.1	-3.8***	1.9***	0.1	-4.0***
BOW	0.1	-	-3.6***	2.1***	0.3	-3.8***
VLAD	3.8***	3.6***	-	5.7***	3.9***	-0.2
LLC	-1.9***	-2.1***	-5.7***	-	-1.8***	-5.9***
IFK	-0.1	-0.3	-3.9***	1.8***	-	-4.1***
VGG	4.0***	3.8***	0.2	5.9***	4.1***	-

(b) Material

	NONE	BOW	VLAD	LLC	IFK	VGG
NONE	-	1.3**	0.1	1.2**	2.8**	0.8*
BOW	-1.3**	-	-1.2	-0.1	1.5**	-0.5
VLAD	-0.1	1.2	-	1.1**	2.7**	0.7
LLC	-1.2**	0.1	-1.1**	-	1.6*	-0.4
IFK	-2.8**	-1.5**	-2.7**	-1.6*	-	-2.0*
VGG	-0.8*	0.5	-0.7	0.4	2.0*	-

(c) UC-Merced

Table 6.1: Differences between the accuracy of the coding algorithms and the VGG-16 base line (in %). The asterisks denote whether the $p_{wilcoxon}$-value is lower than 0.01 (***), 0.05 (**), and 0.10 (*).

	RF	SVM	VGG
RF	-	1.8***	-0.1
SVM	-1.8***	-	-2.0***
VGG	0.1	2.0***	-

(a) Shape

	RF	SVM	VGG
RF	-	6.1***	0.9**
SVM	-6.1***	-	-5.2***
VGG	-0.9**	5.2***	-

(b) Material

	RF	SVM	VGG
RF	-	-1.3**	-0.4
SVM	1.3**	-	0.8*
VGG	0.4	-0.8*	-

(c) UC-Merced

Table 6.2: Differences between the accuracy of the SVM, RF and the VGG-16 base line (in %). The asterisks denote whether the $p_{wilcoxon}$-value is lower than 0.01 (***), 0.05 (**), and 0.10 (*).

the most complex coding scheme, IFK, achieves the lowest result. The results of RF and SVM trained on the features extracted from the last fully connected layer, neither show any considerable increases in classification accuracy on the three data sets. RF scores equivalent accuracy on the roof shape and UC-Merced data set, and achieves slightly higher accuracy on the roof material data set. SVM performs significantly worse on the two Sint Maarten data sets, while achieving similar accuracy on the UC-Merced data set. Based on these results, one could conclude that feature coding does not improve the classification performance of CNNs and RF could potentially improve the classification performance of CNNs.

However, one could argue that the comparison between the VGG-16 and the alternative approaches is unfair, as the VGG-16 model is trained on more augmented data due to the possibility of batch-wise training. While there exist adaptations of SVM [Diehl and Cauwenberghs, 2003, Nikitidis et al., 2010] and RF [Saffari et al., 2009] which are able to be trained on batches of data, these are not (yet) available in mainstream machine- and deep-learning packages. Thus, the inability of batch-wise training can be seen as a limitation inherent to the alternative approaches. Due to this shortcoming, the extracted image features of all training images have to be able to fit in the machine's memory.

The alternative approaches introduce new steps into the classification process. This makes classification more complex and time consuming, which may be less important in a research environment, but very unpractical in real world applications. The base line VGG-16 can be trained using batches of images and predicts new imagery in a single step. The proposed scenarios requires similar time to extract the features, but also requires training additional feature coding schemes and classifiers, making it more computational and time expensive then conventional CNN classification strategies. The extra complexity and time-consumption of using these alternative strategy do not outweigh the potential (small) improvements in classification performance. This more practical look at image classification pipelines is something that is often missing in the related literature.

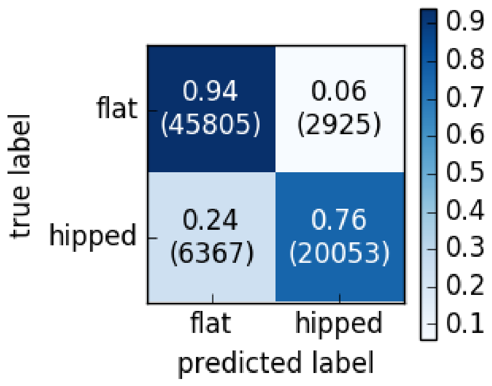
6.2 Per data set evaluation

While the research questions in this work are more focused on exploring new strategies to improve classification performance, the main objective is to produce a model which predicts the roof shape and roof material of the buildings as good as possible. Therefore, this section will take a closer look at the best performing model. Overall, none of the models beats the fine-tuned VGG-16 performance by a considerable margin. Hence, this section only focuses on these results. The remainder of this section is structured as follows: first, the results of the Sint Maarten data sets will be discussed and we will provide some additional insights. Secondly, the results of the model on the UC-Merced data set will be discussed with respect to the results in other papers.

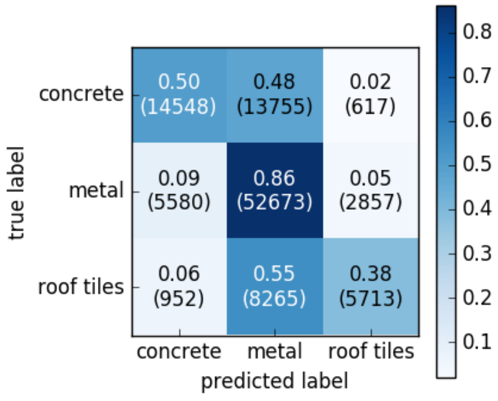
6.2.1 Sint Maarten

The data set of Sint Maarten contains two type of classes describing buildings: roof shape and roof material. The labels for roof shape and roof material are respectively flat and hip; and concrete, metal and roof tiles. The roof shape is more easily to distinguish then the roof material of a building, as illustrated in chapter 3. Furthermore, both data sets are moderately imbalanced. The relative frequencies of flat and hipped are respectively 65/35%, and of concrete, metal and roof tiles are respectively 28/58/14%.

The F1-scores shown in table 5.3 already indicate that the class imbalance might affect the results of the model. On the roof shape data set, the F1-score of the majority class (i.e. flat, 0.91) is considerably higher than the minority class (i.e. hipped, 0.81). The F1-scores on the roof



(a) Shape



(b) Material

Figure 6.1: Confusion matrices of VGG-16 on the (a) roof shape and (b) roof material data set.

material data set follow the same trend: the F1-score of the majority class (i.e. metal, 0.78) is highest one of the three, followed by the second largest class (i.e. concrete, 0.58) and the smallest class (i.e. roof tiles, 0.47). This trend can also be seen in the the confusion matrices shown in figure 6.1. The x- and y-axis denote respectively the true and predicted label. On the roof shape data set, the relative frequency of the flat and hipped predicted samples is 69/31%. For the roof material data set, these relative frequencies for concrete, metal and roof tiles are 20/71/9%. Thus, the predictions are more skewed than the actual class distribution of the data sets, meaning that this model overestimates the amount of buildings belonging to the majority class. This skewness is particularly visible in the per-class recall scores, which are 0.94/0.76 and 0.50/0.86/0.38 for respectively the roof shape and roof material data set, while the per-class precision scores are relatively balanced, i.e. respectively 0.88/0.87 and 0.70/0.71/0.63.

This skewness due to imbalance can be explained by the way the models are optimised. During training, the accuracy of the validation set is monitored and only the best performing network is saved. Therefore, the models are optimised with regard to accuracy, which disadvantages the minority class(es). Accuracy only measures whether a sample is predicted correct and does not care about per class performance. Correcting for this imbalance, would hurt the overall accuracy. Since predicting a sample as the majority class already has a higher probability of success than predicting the samples as the minority class, a model optimised for accuracy tends to predict the majority class more easily.

The effects of the class imbalance in the data sets are clearly visible in the results. The next question is whether it actually matters to this specific problem. Some classification problems require a very good precision or recall on a specific class, while the other class is less important. On the other hand, in other classification problems one only cares about the overall accuracy. As the experiments in this work only serve as a proof of concept and the data quality requirements for future follow-up projects are not yet defined, it is unknown whether this trait is undesirable. However, future projects should define the purpose and requirements of the predictions before training the model.

The fine-tuned VGG-16 achieves an overall accuracy score of 88%, which exceeds the desired result expressed prior to this work (achieving an accuracy of at least 85%). Furthermore, the Kappa score, which can be seen as a better indicator of classification quality, is 72%. Following the guidelines of Landis and Koch [1977], McHugh [2012] and Fleiss et al. [2013], the achieved Kappa score can be interpreted as respectively *Substantial*, *Moderate* and *Fair to good*.

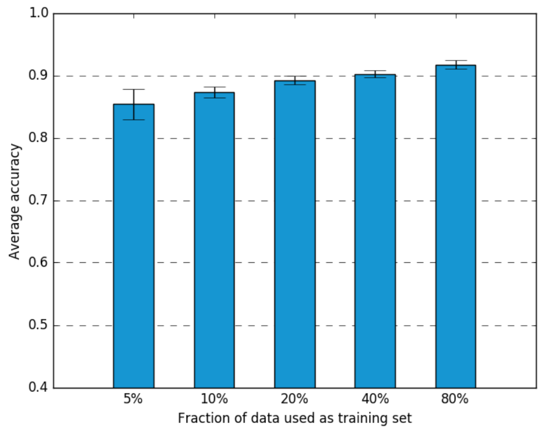
The roof material data set seems to be more difficult to classify compared to the roof shape data set. The mean accuracy is 69%, which is lower than the desired 85%. The poor classification performance is also reflected in the Kappa score, which is only 41%. The Kappa score just falls in the *Moderate* category by Landis and Koch [1977], *Weak* category by McHugh [2012], and *Fair to good* category by Fleiss et al. [2013].

Performance indicator	**Shape**	**Material**
Kappa	0.72	0.41
Landis and Koch [1977]	Substantial	Moderate
McHugh [2012]	Moderate	Weak
Fleiss et al. [2013]	Fair to good	Fair to good

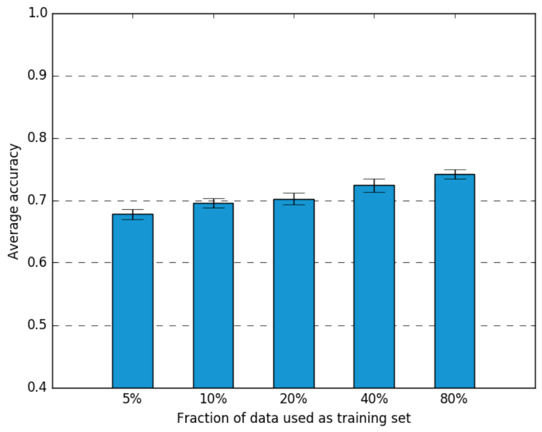
Table 6.3: Interpretations of the kappa score for the roof shape and roof material data set.

The Kappa scores and interpretations by the three guidelines are shown in table 6.3. Note again that these guidelines are entirely arbitrary, although they might help the reader to better understand the kappas score. The results also uncover the weakness of the interpretation by Fleiss et al. [2013], which denotes both classifications as *Fair to good*, while the difference in their respective Kappa scores is over 30%.

As mentioned in chapter 3, there is some uncertainty with respect to the quality of the class labels. The buildings are labelled by many (untrained) volunteers. While the roof shape classes are quite distinct, this is not always the case for the roof material classes, especially metal and roof tiles. Mislabelled images have two consequences on the results. First, mislabelled images in the train set deteriorate the performance of the classification model, as the model will learn the wrong connections between imagery and labels. Secondly, mislabelled images in the test set result in an incorrectness in the evaluation metrics, as predictions are wrongfully denoted as TP, FP, TN and FN.



(a) Shape



(b) Material

Figure 6.2: Performance of VGG-16 using various fractions of the total data set

In the experiments presented so far, only 10% of the data is used as train set. Training the network on more data should always improve the performance of the model. However, labelling new data is a time consuming and labour intensive process. For that reason, the VGG-16 is fine-tuned using various fractions of the total data set for training. Figure 6.2 shows the mean accuracy of the fine-tuned VGG-16 models trained on 5%, 10%, 20%, 40%, and 80% of total data set. For equal comparison, the models are tested on the same images (which constitute 20% of the total data set). As expected, the accuracy increases over the data set. However, the marginal effect of adding more train data decreases over the amount of train data. For example, the marginal affect of increasing the fraction train data by 1% is 0.3845% per percent between 5% and 10%, which diminishes to a mere 0.0378% increase in accuracy between 40% and 80%. A similar trend is also visible for the roof material data (except between 10% and 20%). A complete overview of the sensitivity of the accuracy to the amount of train data is shown in table 6.4.

From (%)	To (%)	Shape (%)	Material (%)
0	5	21.3452	16.9510
5	10	0.3845	0.3527
10	20	0.1940	0.0672
20	40	0.0482	0.1087
40	80	0.0378	0.0435

Table 6.4: Increase in accuracy for a 1% increase in the fraction of data used for training

6.2.2 UC-Merced

Unlike the Sint Maarten data sets, the UC-Merced data set does not suffer from class imbalance. It contains 2100 images equally divided into 21 classes. The confusion matrix of the predictions for the UC-Merced data set is shown in figure 6.3. Overall, the fine-tuned VGG-16 network is good in distinguishing between the different classes, where for 17 out of 21 classes more than 85% of the samples are correctly predicted. The model has more difficulty with similar classes, such as: medium-residential, dense-residential and mobile-home park; and: runway and freeway. Moreover, the fine-tuned VGG-16 achieves a mean accuracy of 90%. While this the performance is pretty impressive, Nogueira et al. [2017] reported a higher mean accuracy (98%) with a similar experimental setup. Equal to the experiment in this work, Nogueira et al. split the data set in 5 sets, where in each fold 3 sets are used as training, 1 for validation, and 1 for testing. Additional, they also use Stochastic Gradient Descent with learning rate 0.001 for fine-tuning. They do not specify the batch size during training. Final classification is done by a linear SVM with C = 1. The implementation differs in the deep learning frame work used, they use the Caffe deep learning framework instead of Keras.

The VGG-16 network is originally trained on ImageNet using the Caffe framework[1]. Keras ported the original weights to be compatible to their own framework. In essence, if the weights are ported perfectly, the Caffe and Keras version of the pre-trained VGG-16 should achieve equal results. Hence, a small experiment is done using the Caffe framework for MATLAB. The pre-trained VGG-16 networks (not fine-tuned) in Keras and Caffe are used to extract image features from the UC-Merced data set. In both setups the features are extracted from the last fully connected layer and final classification is done by the same linear SVM as used in the rest of this work. The experiments are carried out using a stratified 5-fold cross-validation protocol, where 4 sets are used for training and 1 for testing. For fair comparison, the splits in the 5-fold cross-validation are exactly the same for the Keras as the Caffe framework. In this experiment, the Keras model and Caffe model achieve a mean accuracy of respectively 87.6 and 90.0, indicating a performance difference between the Keras and Caffe models.

The reported accuracy by Nogueira et al. [2017], 98%, is achieved by extracting the features using a fine-tuned VGG-16 and final classification by a linear SVM. An equivalent model in this work achieved a mean accuracy of 92%, which denotes a 6% performance gap. The features extracted by the VGG-16 used by Nogueira et al. are available on the authors website[2], creating

[1]https://github.com/BVLC/caffe
[2]https://github.com/keillernogueira/exploit-cnn-rs

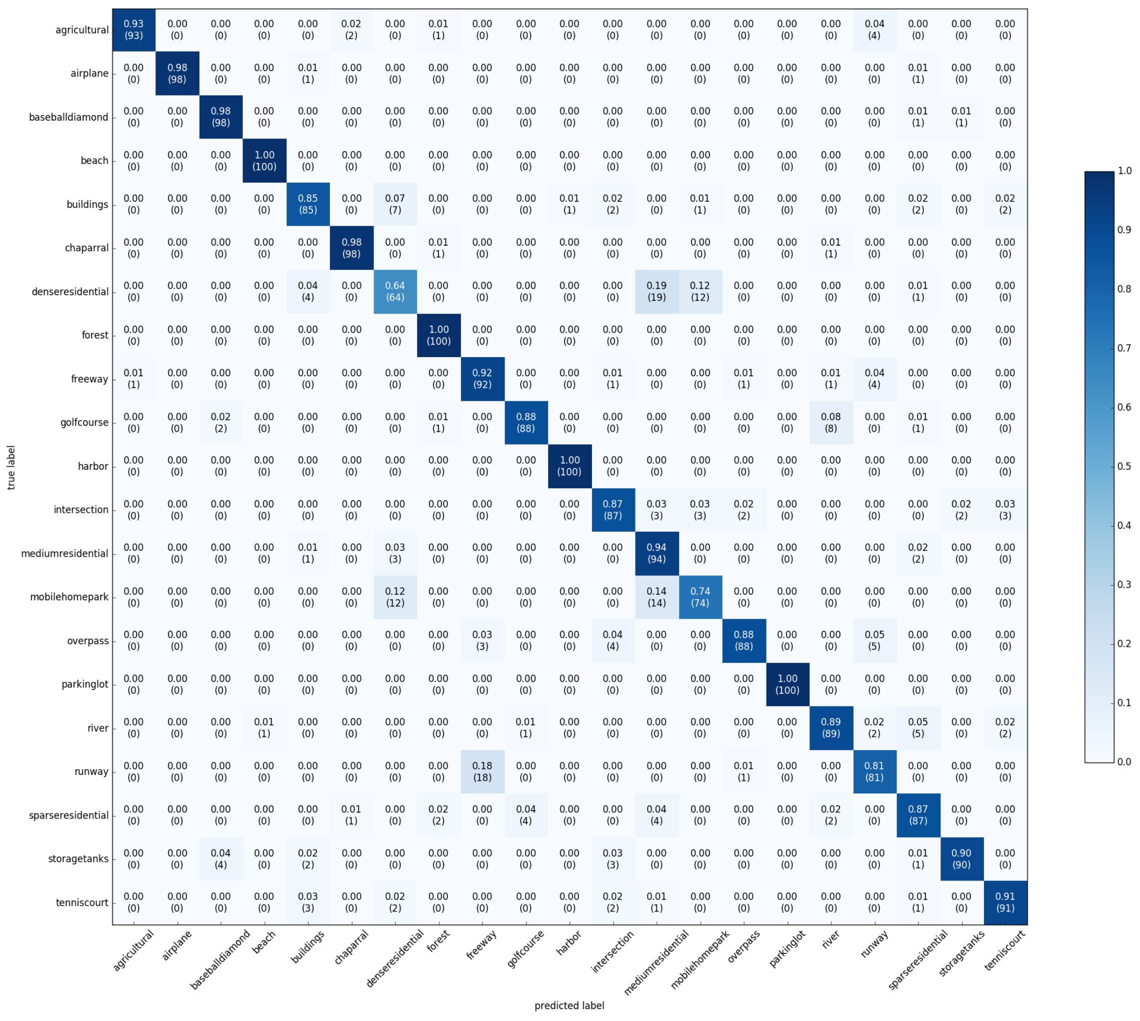


Figure 6.3: Confusion matrix of VGG-16 on the UC-Merced data set

the possibility for further evaluation. A new experiment is set up in a similar fashion as described before (stratified 5-fold cross-validation), the image features by Nogueira et al. are used to train the same linear SVM as before. Using these image features, an accuracy is achieved of 95%. The difference between the accuracy by Nogueira et al. (98%) and in the accuracy of this experiment (95%) could be explained by a difference in the implementation of the SVM. However, due to the absence of detailed implementation details or code, no definitive conclusion can be made.

At last, various hyper-parameters settings during training could be the source of the difference. Training a CNN involves setting a lot of hyper-parameters. While the same learning rate is used, other settings could have been different (e.g. batch-size, momentum). One notable difference is that a number of layers are frozen during training in this work due to computational limitations, which is not done in the referenced paper. Further, Nogueira et al. denote that more importance is given to the final fully connected layer during training, a functionality currently unavailable in Keras.

Hence, the performance difference could be explained by the choice of framework, the hyper-parameter settings of the final classifier, and other hyper-parameter settings during training. The uncertainty with respect to the underlying cause underlines the importance of openness concerning the experiments to enable reproducible results. One major step in the right direction would be making the code used for the experiments publicly available.

Chapter 7

Practical Implications

While chapter 6 discusses the results with respect to the specified research questions, this work must be placed in the grand scheme of things. The NLRC data team, 510, investigates the possibility of automating the mapping process currently done by many volunteers. This investigation has two research directions; the first one focuses on automatically detecting and tracing buildings in aerial imagery, the second one examines the possibility of automatically determining building characteristics in imagery of buildings. This work falls within the second direction of the overarching research.

Although the models in this work only classify roof material and roof shape, other building characteristics could be valuable as well. For example, the construction material of walls could be an indicator of building robustness against natural disasters and could be used as a variable in damage predictions models. These predictions models could be used to enhance the efficiency and effectiveness of humanitarian aid. Models classifying other characteristics can be constructed in a similar fashion as the models presented in this work, where the key differences are the image characteristics and the class labels. Classifying wall types is impossible using the imagery of Sint Maarten currently available. The height from which the imagery is taken is too high, which makes oblique viewpoints of buildings impossible. Furthermore, the roof shape labels used in this work can be either hipped or flat. However, Partovi et al. [2017] breaks up the hipped class in more specific classes (e.g. gable, half-hip, hip, pyramid). Shooting imagery from lower heights (resulting in higher quality imagery) and always breaking up classes in more specific classes (as labels can always be aggregated) may seem like a good solution, this is not necessarily the case; shooting images from lower levels means that more images in total are needed to cover similar areas of interest, and; breaking up classes could lead to an increase in time needed to label the images. However, these examples emphasise the necessity of proper preparation prior to gathering the data. Hence, the process of generating the data (i.e. imagery and labels) should be in close collaboration with the intended end-user(s). Involving these stake holders late or insufficiently could lead to a mismatch in expectations regarding the data, resulting in delays, increased costs, or less effective models.

Using models to classify building types automatically drastically reduces the amount of manual labelling to be done for large areas. On the other hand, using models increases the importance of the quality of the labels. The problem of incorrectly labelled images is twofold: it could hurt the classification performance of the models and the performance metrics could give a faulty impression of the classification capability of the models. When a model, which is wrongfully deemed of sufficient quality, is deployed, the model could generate labels with unknown characteristics. Such unknown characteristics could have unintended side-effects when used for humanitarian aid purposes, thereby potentially affecting the life of vulnerable people. Therefore, evaluation on a fair representation of the images with corresponding class labels is very important.

CNNs have proven to be able to outperform human classification performance in both general object classification [He et al., 2015] as in more specialised medical image classification [Haenssle et al., 2018]. However, the potential of CNNs can only be fully utilised given sufficient data in terms of quality and quantity. The labels of roof shape and roof material used in this work are obtained manually through the efforts of (untrained) volunteers. To our knowledge, each image is only labelled once by one volunteer. Hence, possible differences among volunteers are not observed. To improve label quality, images can be labelled by multiple persons, creating the opportunity to compare the labelling efforts of the volunteers. Another approach would be the involvement of experts or locals. Experts can label a part of the data set, which can serve as a benchmark for the labelling quality of the volunteers. Even better would be to let locals assess individual buildings in person. However, this would be very time consuming and expensive for larger areas. Future implementations of CNN based classification models could benefit from semi-supervised learning strategies, such as pseudo-labelling [Lee, 2013]. In this strategy the model is first trained on a (small) set of labelled images. Subsequently, the model predicts the labels of a set of test images, whereafter this will be added to the train data. This enlarged train set is then used to re-train the model. Furthermore, one can use the trained model to assist the labelling process: the trained model suggests labels of new imagery. This approach can especially be useful in data sets with many distinct classes. Another interesting possibility is to use CNNs to enhance the quality of the data. Dong et al. [2016] proposed so-called super-resolution CNNs which can learn the mapping between low- and high-resolution imagery.

After finalising the data set, one can start building classification models. In practice, training CNNs from scratch for image classification is unfeasible due to limited amount of data and computational limitations. Hence, fine-tuning pre-trained networks on the target data is the better alternative. There are many pre-trained networks available and past research have showed that the relative performance differs per data set. Unfortunately, the performance of pre-trained networks as feature extractor (no fine-tuning) in combination with a classifier, is not a reliable indicator of the performance of the fine-tuned equivalent. Appendix G shows results of the pre-trained networks used in this work as feature extractors, where a linear SVM is implemented as final classifier. While using the pre-trained CNNs as feature extractor does not indicate its relative performance among other networks, it is a good starting point in assessing the feasibility of the classification problem.

One factor to consider is the required input size of the images, as one needs to transform the imagery to a network-specific format. Downsizing images could result in a loss of relevant information deteriorating the classification performance. In this study most images are smaller than 200×200 (before re-sizing). Thus, not much information is lost due to re-sizing for most imagery. However, for larger images this might affect the classification performance. Currently, most pre-trained CNNs support input size of 224×224 or 299×299.

Another factor to consider is the support of the deep learning framework used. While ports of networks across frameworks is (often) possible, porting introduces extra complexity in setting up the models. In this work the Keras framework is used, which provides a high-level interface to other (more) low-level deep learning frameworks. Its simplicity increases the usability, especially for non-specialised people. However, for certain highly specialised computer vision scientists, Keras could not provide the flexibility needed. A tutorial on how to use Keras on Google Colaboratory is available on Github[1]. For new classification problems, we suggest to start with relatively simple CNNs which have proven their effectiveness in other works (e.g. VGG-16) using similar training parameters as used in this work. Next, if desirable, one can experiment with more advanced training strategies.

Hardware is an important aspect when using (deep) CNN structures. Parallelism of computations in GPUs severely speeds up the training and prediction phase of CNNs. For example, training the VGG-16 network on the roof material data set (approximately 800 images) takes around 18

[1]https://github.com/bartvandriel/keras_tutorial_image_classification

seconds per epoch when using the GPU. If only the CPU is enabled, the same epoch takes over 17 minutes, which is 58 times the GPU enabled processing time. Generally, two options can be considered when choosing hardware and both alternatives have their own respective advantages and disadvantages. First, one can set up their models using cloud computing services, which are very flexible in scale and performance. However, cloud computing might get expensive when using for extended periods of time, and requires the data to be transferred to external parties, which might be undesirable with respect to data protection guidelines. The second alternative is set up your own dedicated GPU server. This might be cheaper in the long run and gives full control with respect to the data. On the other hand, setting up your own server has relatively high start-up costs and is less flexible in terms of scale and performance.

As discussed in the previous chapter, optimising the models on accuracy could lead to skewed predictions when the data set is imbalanced. Whether this skewness is problematic or not depends on the problem domain and should be within the requirements of the end-user(s). Furthermore, correcting the skewness leads to a decrease in overall accuracy. The easiest approach would be passing weights to the loss function and optimising on loss (instead of accuracy). The weights should be set equal to the inverse of the relative proportions of the classes in the data set. Further, balancing the training and validation set by under- or over-sampling respectively the majority and minority class(es) has similar effects to the weighted loss function alternative.

Before deploying the models, the classification results should be evaluated with respect to the requirements of the end-user and adapted if needed. The evaluation metrics used in this work are especially good for classification problems. Image segmentation and retrieval problems require other evaluation metrics. Most important is that the requirements are constructed before collecting the data, as the characteristics of the images define the classification possibilities. This underlines the importance of a structured way of working. The CRISP-DM model captures most of the steps discussed in this chapter and is a good guideline for future implementation.

Chapter 8

Conclusion

Finally, this chapter concludes this thesis. First, section 8.1 summarises the main conclusions. Subsequently, section 8.2 describes the limitations, which should be taken into account interpreting the conclusions. At last, section 8.3 recommends a number of promising new research directions.

8.1 Conclusions

Accurate, complete and up-to-date maps of areas are crucial for effective and efficient humanitarian aid. These maps are often unavailable for the most vulnerable areas. Currently, these maps are obtained trough the work of many volunteers, who manually trace and classify buildings and roads in OpenStreetMap using aerial imagery. NLRC 510 investigates the possibility of automating this process. This investigation takes two directions: tracing the buildings and classifying characteristics of the buildings in aerial imagery. This work focuses on the latter, using aerial imagery from Sint Maarten combined with detailed building information from OpenStreetMap. The Sint Maarten data includes two roof type classes, roof shape and roof material. The labels for roof shape is either flat or hipped, and for roof material either concrete, metal or roof tiles. In order to validate the models, the UC-Merced data set is included for comparison.

Recent years, CNN based classification models have shown outstanding results on benchmark data sets. Especially, the use of deep CNNs pre-trained on large image data bases (e.g. ImageNet) has improved the usability of these kind of networks. Generally, fine-tuning these pre-trained networks achieve the best results. In this work, two new approaches are proposed which further exploit features generated by a fine-tuned CNN. The first approach is inspired by traditional image classification pipe lines, which encodes image features before classification. First, features are extracted from the last convolutional layer. Subsequently, the features are encoded using four feature coding algorithms (i.e. BOW, VLAD, LLC and IFK). Finally, classification is conducted by a linear SVM. The second approach investigates the use of RF with CNNs. Features are extracted from the last fully connected layer of a fine-tuned CNN and classification is done by a RF and SVM. Three pre-trained networks are fine-tuned on the aforementioned data sets. Overall, VGG-16 tends to achieve the best classification performance and is chosen as feature extractor.

None of the feature coding approaches outperforms the base line network, the fine-tuned VGG-16 network. Only VLAD, one of the relatively simple coding schemes, achieves equivalent results on all three data sets. The alternative classifiers, using the features from the fully connected layers, did not show consistent improvements compared to the VGG-16 model over all data sets. On the roof material data set, RF showed a small improvement and SVM achieved a slightly higher accuracy on the UC-Merced data set. The proposed approaches introduce new steps in the classification process, making the implementation of such models more complex and time consuming compared to simply using a fine-tuned network. Moreover, the positive effects of data augmentation is limited in the alternative approaches, since SVM and RF are (in essence) not able to be trained on batches of data. Based on the results presented in this work, feature coding does not improve classification performance of CNNs on the three data sets used. RF showed a small improvement ($< 1\%$) for the material data set and equivalent performance on the other two data sets compared to the VGG-16 base line. To conclude, RF can improve classification accuracy of the CNNs base classification models. However, the potential benefits (i.e. increased classification performance) of experimenting with the proposed approaches do not outweigh the disadvantages (i.e. more complex classification pipeline, extra computation time), as the potential improvement is quite small.

With a mean accuracy and Kappa score of respectively 88% and 72%, the model classifying the roof shapes (i.e. flat or hipped) is of a satisfactory quality. The classification of the roof material (i.e concrete, metal and roof tiles) is more problematic, with a mean accuracy and Kappa of respectively 69% and 41%. Due to class imbalance, the predictions of both models are skewed to the majority class(es). As this work is purely a proof of concept it is unknown whether this skewness is undesirable. Correcting this skewness will decrease overall accuracy. This trade-off must be taken into account in future implementations. A potential reason for the unsatisfactory classification of the roof material data set could be the quality of the building labels. As mentioned in chapter 3, the buildings in the metal and roof tiles class look very much alike. Mislabelled buildings could decrease the performance of the models and cause a misrepresentation of the classification performance by the evaluation metrics. The classification models can be improved by using more train data. However, the marginal effect of adding more train data decreases over the amount of train data already used.

While the models are conducted on similar kind of setups, the results of other works have not been replicated. Various reasons could contribute to this performance gap. First, the deep learning framework used in this study achieves different results using, what is supposed to be, the same model. Furthermore, the linear SVM implementation used in this work could differ from other works. Moreover, training CNNs involves setting many hyper-parameters and most studies only report a fraction of those. Hence, without additional information regarding these parameters uncertainty remains with respect to the underlying reason of this performance gap.

By providing new insights in using CNN based classification approaches this work contributes to the existing literature. Previous works experimented with alternative classifiers and feature coding, but not as in this work. Furthermore, other works ignore the practical limitations to computational resources. In practice, such expensive resources are not always available. Furthermore, this work extends the little existing research to classification of building characteristics using CNNs.

The key factor in the success of classifying building characteristics is data. Potentially, CNNs can outperform humans in image classification tasks, although this depends very much on the quantity and quality of the data. Additionally, a high quality test set, which is a fair representation of the area of interest, is important for evaluation purposes. What kind of buildings characteristics can be classified is related to what is visible in the imagery. For example, if one aims to determine wall types the images should be taken from a lower angle. Training a CNN requires many parameter choices, which not only affect the performance but also the characteristics of the classification. Therefore, the models should be set up in alignment with the requirements of the end-user(s). A structured approach (such as CRISP-DM) could help to enhance the process of creating these classification models.

8.2 Limitations

The results presented in this work provide valuable insights in the application of CNNs for the automatic classification of roof types in aerial imagery. However, several factors limit the extend of the conclusions.

First, the conclusions made in this work are only based on the data sets used, i.e. the roof shape, roof material and the UC-Merced data set. As can be seen in the results, the relative performance of the models differ per data set. This indicates that the same experiments on other data sets could lead to other conclusions.

Secondly, the uncertainty regarding the quality of the labels is another limitation. The absence of additional ground truth information makes it impossible to determine the quality of the labels.

At last, limits to the computational resources constrained the design choices of the experiment. For example, the amount of augmented data used to train the SVM and RF is bound due to the available RAM in Google Colaboratory. Therefore, utilising more data augmentation could improve the feature coding and RF approach. Additionally, several training hyper-parameters are set such that the computational burden is reduced, e.g. freezing layers. More computational resources would enable to chose between more hyper-parameters, and potentially, result to different results. However, these kind of resources are expensive, which makes it less appealing to use such classification pipelines in practice.

8.3 Future Research

This work provides a basis for future automatic classification of building characteristics. Potentially, CNNs can outperform humans in image classification tasks. Hence, future research direction are identified currently holding back the potential of CNNs in classification of building characteristics: data quality and quantity.

The quality of the data influences what a classification models can classify. Higher resolution imagery will reveal more refined details, which enables to classification of more distinct classes. Obtaining high resolution imagery is difficult and buying it from external parties can be very expensive. One interesting solution would be to enhance image resolution using CNNs as discussed by Dong et al. [2016]. In theory, such models can transform low resolution satellite imagery into higher resolution imagery.

Secondly, more research should be done to efficiently and effectively label large image data sets. As the quality and quantity of the labels affect the classification performance and evaluation reliability, this is an important direction for future implementation. One approach could be some sort of model assisted labelling.

At last, capturing walls within remotely sensed imagery is still very difficult. Often, oblique perspectives are missing due to the height the image is taken. Moreover, if walls are visible, only a part of the wall or one side of the building is visible. More research should be done in how to capture walls effectively and efficiently in remotely sensed imagery, whereafter can be used to train CNN classification models.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org. 31

Rasha Alshehhi, Prashanth Reddy Marpu, Wei Lee Woon, and Mauro Dalla Mura. Simultaneous extraction of roads and buildings in remote sensing imagery with convolutional neural networks. *ISPRS Journal of Photogrammetry and Remote Sensing*, 130:139–149, 2017. 8

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006. 2

Mariana Belgiu and Lucian Drăguţ. Random forest in remote sensing: A review of applications and future directions. *ISPRS Journal of Photogrammetry and Remote Sensing*, 114:24–31, 2016. 15

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012. 27

L Breiman, JH Friedman, RA Olshen, and CJ Stone. Classification and regression trees. monterey, ca: Wadsworth. wadsworth statistics/probability series, 1984. 15

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 14

Jeremy Castagno and Ella Atkins. Roof shape classification from lidar and satellite image data fusion using supervised learning. *Sensors*, 18(11):3960, 2018a. 2

Jeremy D Castagno and Ella M Atkins. Automatic classification of roof shapes for multicopter emergency landing site selection. *arXiv preprint arXiv:1802.06274*, 2018b. 2

Marco Castelluccio, Giovanni Poggi, Carlo Sansone, and Luisa Verdoliva. Land use classification in remote sensing images by convolutional neural networks. *arXiv preprint arXiv:1508.00092*, 2015. 7, 9, 10

Gong Cheng, Chengcheng Ma, Peicheng Zhou, Xiwen Yao, and Junwei Han. Scene classification of high resolution remote sensing images using convolutional neural networks. In *Geoscience and Remote Sensing Symposium (IGARSS), 2016 IEEE International*, pages 767–770. IEEE, 2016a. 7

Gong Cheng, Peicheng Zhou, and Junwei Han. Learning rotation-invariant convolutional neural networks for object detection in vhr optical remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 54(12):7405–7415, 2016b. 8

Gong Cheng, Zhenpeng Li, Xiwen Yao, Lei Guo, and Zhongliang Wei. Remote sensing image scene classification using bag of convolutional features. *IEEE Geoscience and Remote Sensing Letters*, 14(10):1735–1739, 2017. 8

François Chollet et al. Keras. `https://keras.io`, 2015. 31

François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, pages 1610–02357, 2017. 26

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960. 29

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 13, 14

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006. 29, 30

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009. 7

Christopher P Diehl and Gert Cauwenberghs. Svm incremental learning, adaptation and optimization. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 4, pages 2685–2690. IEEE, 2003. 41

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2): 295–307, 2016. 48, 53

ECIU. Heavy weather: Tracking the fingerprints of climate change, two years after the paris summit. Technical report, Energy & Climate Intelligence Unit, December 2017. URL `https://eciu.net/assets/Reports/ECIU_Climate_Attribution-report-Dec-2017.pdf`. 1

Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. *Statistical methods for rates and proportions*. John Wiley & Sons, 2013. 29, 30, 42, 43

Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937. 30

GDAL/OGR contributors. *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation, 2018. URL `http://gdal.org`. 31

HA Haenssle, C Fink, R Schneiderbauer, F Toberer, T Buhl, A Blum, A Kalloo, A Ben Hadj Hassen, L Thomas, A Enk, et al. Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. *Annals of Oncology*, 29(8):1836–1842, 2018. 48

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 48

Fan Hu, Gui-Song Xia, Jingwen Hu, and Liangpei Zhang. Transferring deep convolutional neural networks for the scene classification of high-resolution remote sensing imagery. *Remote Sensing*, 7(11):14680–14707, 2015. 8, 24, 26

Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3304–3311. IEEE, 2010. 8, 12

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. URL `http://www.scipy.org/`. 31

Carolyn Kousky. Informing climate adaptation: A review of the economic costs of natural disasters. *Energy Economics*, 46:576–592, 2014. 1

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 7

J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977. 29, 30, 42, 43

Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, page 2, 2013. 48

Wei Li, Guodong Wu, Fan Zhang, and Qian Du. Hyperspectral image classification using deep pixel-pair features. *IEEE Transactions on Geoscience and Remote Sensing*, 55(2):844–853, 2017. 8

Ying Li, Haokui Zhang, Xizhe Xue, Yenan Jiang, and Qiang Shen. Deep learning for remote sensing image classification: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1264, 2018. 2, 7

David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999. 2

Mary L McHugh. Interrater reliability: the kappa statistic. *Biochemia medica: Biochemia medica*, 22(3):276–282, 2012. 29, 30, 42, 43

Giorgos Mountrakis, Jungho Im, and Caesar Ogole. Support vector machines in remote sensing: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(3):247–259, 2011. 14

MunichRE. A stormy year: Natural catastrophes 2017. Technical report, Munich RE, December 2018. URL `https://www.munichre.com/content/dam/websites/munichre/mrwebsites/topics-online/2018/topics-geo-2017/further-information/302-09092_en.pdf`. 1

Symeon Nikitidis, Nikos Nikolaidis, and Ioannis Pitas. Incremental training of multiclass support vector machines. In *2010 international conference on pattern recognition*, pages 4267–4270. IEEE, 2010. 41

Keiller Nogueira, Otávio AB Penatti, and Jefersson A dos Santos. Towards better exploiting convolutional neural networks for remote sensing scene classification. *Pattern Recognition*, 61: 539–556, 2017. 8, 24, 27, 44, 46

Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006. 31

T Partovi, F Fraundorfer, S Azimi, D Marmanis, and P Reinartz. Roof type selection based on patch-based classification using deep learning for high resolution satellite imagery. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 42, 2017. 2, 47

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 31

Florent Perronnin and Christopher Dance. Fisher kernels on visual vocabularies for image categorization. In *2007 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2007. 12

Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In *European conference on computer vision*, pages 143–156. Springer, 2010. 8, 12

David MW Powers. What the f-measure doesn't measure: Features, flaws, fallacies and fixes. *arXiv preprint arXiv:1503.06410*, 2015. 29

QGIS Development Team. *QGIS Geographic Information System*. Open Source Geospatial Foundation, 2009. URL http://qgis.org. 31

Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011. 2

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y. 24

Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. On-line random forests. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1393–1400. IEEE, 2009. 41

Grant J Scott, Matthew R England, William A Starms, Richard A Marcum, and Curt H Davis. Training deep convolutional neural networks for land–cover classification of high-resolution imagery. *IEEE Geoscience and Remote Sensing Letters*, 14(4):549–553, 2017. 8

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 25

Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *null*, page 1470. IEEE, 2003. 2, 8, 11

Viktor Slavkovikj, Steven Verstockt, Wesley De Neve, Sofie Van Hoecke, and Rik Van de Walle. Hyperspectral image classification with convolutional neural networks. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1159–1162. ACM, 2015. 8

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 25

Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3360–3367. IEEE, 2010. 8, 12

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945. 30

Hao Wu and Saurabh Prasad. Convolutional recurrent neural networks forhyperspectral data classification. *Remote Sensing*, 9(3):298, 2017. 8

Yi Yang and Shawn Newsam. Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*, pages 270–279. ACM, 2010. 20

Zhijing Yang, Faxian Cao, Jinchang Ren, and Wing-Kuen Ling. Convolutional neural network extreme learning machine (cnn-elm) for effective classification of hyperspectral images. *Journal of Applied Remote Sensing*, 2018. 8

Shiqi Yu, Sen Jia, and Chunyan Xu. Convolutional neural networks for hyperspectral image classification. *Neurocomputing*, 219:88–98, 2017a. 8

Xingrui Yu, Xiaomin Wu, Chunbo Luo, and Peng Ren. Deep learning in remote sensing scene classification: a data augmentation enhanced convolutional neural network framework. *GIScience & Remote Sensing*, 54(5):741–758, 2017b. 8

Yanfei Zhong, Feng Fei, and Liangpei Zhang. Large patch convolutional neural networks for the scene classification of high spatial resolution imagery. *Journal of Applied Remote Sensing*, 10 (2):025006, 2016. 8

Weixun Zhou, Shawn Newsam, Congmin Li, and Zhenfeng Shao. Learning low dimensional convolutional neural networks for high-resolution remote sensing image retrieval. *Remote Sensing*, 9(5):489, 2017. 8, 24, 26

L. Zhu, Y. Chen, P. Ghamisi, and J. A. Benediktsson. Generative adversarial networks for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 56(9): 5046–5063, Sept 2018. ISSN 0196-2892. doi: 10.1109/TGRS.2018.2805286. 8

Appendix A

Pre-trained networks available in Keras

Rank	Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters
1	NASNetLarge	343 MB	0.825	0.96	88949818
2	InceptionResNetV2	215 MB	0.803	0.953	55873736
3	*Xception*	*88 MB*	*0.79*	*0.945*	*22910480*
4	ResNeXt101	170 MB	0.787	0.943	44315560
5	ResNet152V2	232 MB	0.78	0.942	60380648
6	*InceptionV3*	*92 MB*	*0.779*	*0.937*	*23851784*
7	ResNeXt50	96 MB	0.777	0.938	25097128
8	DenseNet201	80 MB	0.773	0.936	20242984
9	ResNet101V2	171 MB	0.772	0.938	44675560
10	ResNet152	232 MB	0.766	0.931	60419944
11	ResNet101	171 MB	0.764	0.928	44707176
12	DenseNet169	57 MB	0.762	0.932	14307880
13	ResNet50V2	98 MB	0.76	0.93	25613800
14	DenseNet121	33 MB	0.75	0.923	8062504
15	ResNet50	98 MB	0.749	0.921	25636712
16	NASNetMobile	23 MB	0.744	0.919	5326716
17	*VGG16*	*528 MB*	*0.713*	*0.901*	*138357544*
18	MobileNetV2	14 MB	0.713	0.901	3538984
19	VGG19	549 MB	0.713	0.9	143667240
20	MobileNet	16 MB	0.704	0.895	4253864

Table A.1: Overview of pre-trained networks available through Keras

Appendix B

Weighted vs. Non-weighted training

			weighted			**non weighted**			**Weighted >non weighted**
Shape	precision	flat	0.89	±	0.02	0.88	±	0.01	TRUE
		hipped	0.88	±	0.04	0.89	±	0.02	FALSE
	recall	flat	0.94	±	0.02	0.95	±	0.01	FALSE
		hipped	0.78	±	0.04	0.76	±	0.03	TRUE
	fscore	flat	0.91	±	0.01	0.91	±	0.01	TRUE
		hipped	0.83	±	0.03	0.82	±	0.02	TRUE
	accuracy	overall	0.89	±	0.02	0.88	±	0.01	TRUE
	kappa	overall	0.74	±	0.04	0.73	±	0.03	TRUE
Material	precision	concrete	0.69	±	0.07	0.64	±	0.05	TRUE
		metal	0.71	±	0.03	0.72	±	0.04	FALSE
		roof tiles	0.65	±	0.09	0.66	±	0.09	FALSE
	recall	concrete	0.51	±	0.10	0.54	±	0.12	FALSE
		metal	0.85	±	0.04	0.83	±	0.05	TRUE
		roof tiles	0.40	±	0.09	0.39	±	0.07	TRUE
	fscore	concrete	0.58	±	0.05	0.58	±	0.07	FALSE
		metal	0.77	±	0.02	0.77	±	0.02	TRUE
		roof tiles	0.49	±	0.07	0.48	±	0.06	TRUE
	accuracy	overall	0.69	±	0.02	0.69	±	0.02	TRUE
	kappa	overall	0.41	±	0.05	0.41	±	0.06	TRUE

Table B.1: Results of fine-tuned VGG-16 on the validation data set using a weighted vs. a non-weighted loss function

Appendix C

Architectures of Convolutional Neural Networks

C.1 VGG-16

Number	Name	Type	Output shape	Parameters	Trainable
1	input_1	InputLayer	(None, 224, 224, 3)	0	FALSE
2	block1_conv1	Conv2D	(None, 224, 224, 64)	1792	FALSE
3	block1_conv2	Conv2D	(None, 224, 224, 64)	36928	FALSE
4	block1_pool	MaxPooling2D	(None, 112, 112, 64)	0	FALSE
5	block2_conv1	Conv2D	(None, 112, 112, 128)	73856	FALSE
6	block2_conv2	Conv2D	(None, 112, 112, 128)	147584	FALSE
7	block2_pool	MaxPooling2D	(None, 56, 56, 128)	0	FALSE
8	block3_conv1	Conv2D	(None, 56, 56, 256)	295168	FALSE
9	block3_conv2	Conv2D	(None, 56, 56, 256)	590080	FALSE
10	block3_conv3	Conv2D	(None, 56, 56, 256)	590080	FALSE
11	block3_pool	MaxPooling2D	(None, 28, 28, 256)	0	TRUE
12	block4_conv1	Conv2D	(None, 28, 28, 512)	1180160	TRUE
13	block4_conv2	Conv2D	(None, 28, 28, 512)	2359808	TRUE
14	block4_conv3	Conv2D	(None, 28, 28, 512)	2359808	TRUE
15	block4_pool	MaxPooling2D	(None, 14, 14, 512)	0	TRUE
16	block5_conv1	Conv2D	(None, 14, 14, 512)	2359808	TRUE
17	block5_conv2	Conv2D	(None, 14, 14, 512)	2359808	TRUE
18	block5_conv3	Conv2D	(None, 14, 14, 512)	2359808	TRUE
19	block5_pool	MaxPooling2D	(None, 7, 7, 512)	0	TRUE
20	flatten_1	Flatten	(None, 25088)	0	TRUE
21	fc1	Dense	(None, 4096)	102764544	TRUE
22	fc2	Dense	(None, 4096)	16781312	TRUE
23	predictions	Dense	(None, 1000)	4097000	TRUE

Table C.1: Summary of the VGG-16 network

C.2 InceptionV3

Number	Name	Type	Output shape	Parameters	Trainable
1	input_2	InputLayer	(None, 299, 299, 3)	0	FALSE
2	conv2d_1	Conv2D	(None, 149, 149, 32)	864	FALSE
3	batch_normalization_1	BatchNormalization	(None, 149, 149, 32)	96	FALSE
4	activation_1	Activation	(None, 149, 149, 32)	0	FALSE
5	conv2d_2	Conv2D	(None, 147, 147, 32)	9216	FALSE
6	batch_normalization_2	BatchNormalization	(None, 147, 147, 32)	96	FALSE
7	activation_2	Activation	(None, 147, 147, 32)	0	FALSE
8	conv2d_3	Conv2D	(None, 147, 147, 64)	18432	FALSE
9	batch_normalization_3	BatchNormalization	(None, 147, 147, 64)	192	FALSE
10	activation_3	Activation	(None, 147, 147, 64)	0	FALSE
11	max_pooling2d_1	MaxPooling2D	(None, 73, 73, 64)	0	FALSE
12	conv2d_4	Conv2D	(None, 73, 73, 80)	5120	FALSE
13	batch_normalization_4	BatchNormalization	(None, 73, 73, 80)	240	FALSE
14	activation_4	Activation	(None, 73, 73, 80)	0	FALSE
15	conv2d_5	Conv2D	(None, 71, 71, 192)	138240	FALSE
16	batch_normalization_5	BatchNormalization	(None, 71, 71, 192)	576	FALSE
17	activation_5	Activation	(None, 71, 71, 192)	0	FALSE
18	max_pooling2d_2	MaxPooling2D	(None, 35, 35, 192)	0	FALSE
19	conv2d_9	Conv2D	(None, 35, 35, 64)	12288	FALSE
20	batch_normalization_9	BatchNormalization	(None, 35, 35, 64)	192	FALSE
21	activation_9	Activation	(None, 35, 35, 64)	0	FALSE
22	conv2d_7	Conv2D	(None, 35, 35, 48)	9216	FALSE
23	conv2d_10	Conv2D	(None, 35, 35, 96)	55296	FALSE
24	batch_normalization_7	BatchNormalization	(None, 35, 35, 48)	144	FALSE
25	batch_normalization_10	BatchNormalization	(None, 35, 35, 96)	288	FALSE
26	activation_7	Activation	(None, 35, 35, 48)	0	FALSE
27	activation_10	Activation	(None, 35, 35, 96)	0	FALSE
28	average_pooling2d_1	AveragePooling2D	(None, 35, 35, 192)	0	FALSE
29	conv2d_6	Conv2D	(None, 35, 35, 64)	12288	FALSE
30	conv2d_8	Conv2D	(None, 35, 35, 64)	76800	FALSE
31	conv2d_11	Conv2D	(None, 35, 35, 96)	82944	FALSE
32	conv2d_12	Conv2D	(None, 35, 35, 32)	6144	FALSE
33	batch_normalization_6	BatchNormalization	(None, 35, 35, 64)	192	FALSE
34	batch_normalization_8	BatchNormalization	(None, 35, 35, 64)	192	FALSE
35	batch_normalization_11	BatchNormalization	(None, 35, 35, 96)	288	FALSE
36	batch_normalization_12	BatchNormalization	(None, 35, 35, 32)	96	FALSE
37	activation_6	Activation	(None, 35, 35, 64)	0	FALSE
38	activation_8	Activation	(None, 35, 35, 64)	0	FALSE
39	activation_11	Activation	(None, 35, 35, 96)	0	FALSE
40	activation_12	Activation	(None, 35, 35, 32)	0	FALSE
41	mixed0	Concatenate	(None, 35, 35, 256)	0	FALSE
42	conv2d_16	Conv2D	(None, 35, 35, 64)	16384	FALSE
43	batch_normalization_16	BatchNormalization	(None, 35, 35, 64)	192	FALSE
44	activation_16	Activation	(None, 35, 35, 64)	0	FALSE
45	conv2d_14	Conv2D	(None, 35, 35, 48)	12288	FALSE
46	conv2d_17	Conv2D	(None, 35, 35, 96)	55296	FALSE
47	batch_normalization_14	BatchNormalization	(None, 35, 35, 48)	144	FALSE
48	batch_normalization_17	BatchNormalization	(None, 35, 35, 96)	288	FALSE
49	activation_14	Activation	(None, 35, 35, 48)	0	FALSE
50	activation_17	Activation	(None, 35, 35, 96)	0	FALSE
51	average_pooling2d_2	AveragePooling2D	(None, 35, 35, 256)	0	FALSE
52	conv2d_13	Conv2D	(None, 35, 35, 64)	16384	FALSE
53	conv2d_15	Conv2D	(None, 35, 35, 64)	76800	FALSE
54	conv2d_18	Conv2D	(None, 35, 35, 96)	82944	FALSE
55	conv2d_19	Conv2D	(None, 35, 35, 64)	16384	FALSE
56	batch_normalization_13	BatchNormalization	(None, 35, 35, 64)	192	FALSE
57	batch_normalization_15	BatchNormalization	(None, 35, 35, 64)	192	FALSE
58	batch_normalization_18	BatchNormalization	(None, 35, 35, 96)	288	FALSE
59	batch_normalization_19	BatchNormalization	(None, 35, 35, 64)	192	FALSE
60	activation_13	Activation	(None, 35, 35, 64)	0	FALSE
61	activation_15	Activation	(None, 35, 35, 64)	0	FALSE
62	activation_18	Activation	(None, 35, 35, 96)	0	FALSE
63	activation_19	Activation	(None, 35, 35, 64)	0	FALSE
64	mixed1	Concatenate	(None, 35, 35, 288)	0	FALSE
65	conv2d_23	Conv2D	(None, 35, 35, 64)	18432	FALSE

66	batch_normalization_23	BatchNormalization	(None, 35, 35, 64)	192	FALSE
67	activation_23	Activation	(None, 35, 35, 64)	0	FALSE
68	conv2d_21	Conv2D	(None, 35, 35, 48)	13824	FALSE
69	conv2d_24	Conv2D	(None, 35, 35, 96)	55296	FALSE
70	batch_normalization_21	BatchNormalization	(None, 35, 35, 48)	144	FALSE
71	batch_normalization_24	BatchNormalization	(None, 35, 35, 96)	288	FALSE
72	activation_21	Activation	(None, 35, 35, 48)	0	FALSE
73	activation_24	Activation	(None, 35, 35, 96)	0	FALSE
74	average_pooling2d_3	AveragePooling2D	(None, 35, 35, 288)	0	FALSE
75	conv2d_20	Conv2D	(None, 35, 35, 64)	18432	FALSE
76	conv2d_22	Conv2D	(None, 35, 35, 64)	76800	FALSE
77	conv2d_25	Conv2D	(None, 35, 35, 96)	82944	FALSE
78	conv2d_26	Conv2D	(None, 35, 35, 64)	18432	FALSE
79	batch_normalization_20	BatchNormalization	(None, 35, 35, 64)	192	FALSE
80	batch_normalization_22	BatchNormalization	(None, 35, 35, 64)	192	FALSE
81	batch_normalization_25	BatchNormalization	(None, 35, 35, 96)	288	FALSE
82	batch_normalization_26	BatchNormalization	(None, 35, 35, 64)	192	FALSE
83	activation_20	Activation	(None, 35, 35, 64)	0	FALSE
84	activation_22	Activation	(None, 35, 35, 64)	0	FALSE
85	activation_25	Activation	(None, 35, 35, 96)	0	FALSE
86	activation_26	Activation	(None, 35, 35, 64)	0	FALSE
87	mixed2	Concatenate	(None, 35, 35, 288)	0	FALSE
88	conv2d_28	Conv2D	(None, 35, 35, 64)	18432	FALSE
89	batch_normalization_28	BatchNormalization	(None, 35, 35, 64)	192	FALSE
90	activation_28	Activation	(None, 35, 35, 64)	0	FALSE
91	conv2d_29	Conv2D	(None, 35, 35, 96)	55296	FALSE
92	batch_normalization_29	BatchNormalization	(None, 35, 35, 96)	288	FALSE
93	activation_29	Activation	(None, 35, 35, 96)	0	FALSE
94	conv2d_27	Conv2D	(None, 17, 17, 384)	995328	FALSE
95	conv2d_30	Conv2D	(None, 17, 17, 96)	82944	FALSE
96	batch_normalization_27	BatchNormalization	(None, 17, 17, 384)	1152	FALSE
97	batch_normalization_30	BatchNormalization	(None, 17, 17, 96)	288	FALSE
98	activation_27	Activation	(None, 17, 17, 384)	0	FALSE
99	activation_30	Activation	(None, 17, 17, 96)	0	FALSE
100	max_pooling2d_3	MaxPooling2D	(None, 17, 17, 288)	0	FALSE
101	mixed3	Concatenate	(None, 17, 17, 768)	0	FALSE
102	conv2d_35	Conv2D	(None, 17, 17, 128)	98304	FALSE
103	batch_normalization_35	BatchNormalization	(None, 17, 17, 128)	384	FALSE
104	activation_35	Activation	(None, 17, 17, 128)	0	FALSE
105	conv2d_36	Conv2D	(None, 17, 17, 128)	114688	FALSE
106	batch_normalization_36	BatchNormalization	(None, 17, 17, 128)	384	FALSE
107	activation_36	Activation	(None, 17, 17, 128)	0	FALSE
108	conv2d_32	Conv2D	(None, 17, 17, 128)	98304	FALSE
109	conv2d_37	Conv2D	(None, 17, 17, 128)	114688	FALSE
110	batch_normalization_32	BatchNormalization	(None, 17, 17, 128)	384	FALSE
111	batch_normalization_37	BatchNormalization	(None, 17, 17, 128)	384	FALSE
112	activation_32	Activation	(None, 17, 17, 128)	0	FALSE
113	activation_37	Activation	(None, 17, 17, 128)	0	FALSE
114	conv2d_33	Conv2D	(None, 17, 17, 128)	114688	FALSE
115	conv2d_38	Conv2D	(None, 17, 17, 128)	114688	FALSE
116	batch_normalization_33	BatchNormalization	(None, 17, 17, 128)	384	FALSE
117	batch_normalization_38	BatchNormalization	(None, 17, 17, 128)	384	FALSE
118	activation_33	Activation	(None, 17, 17, 128)	0	FALSE
119	activation_38	Activation	(None, 17, 17, 128)	0	FALSE
120	average_pooling2d_4	AveragePooling2D	(None, 17, 17, 768)	0	FALSE
121	conv2d_31	Conv2D	(None, 17, 17, 192)	147456	FALSE
122	conv2d_34	Conv2D	(None, 17, 17, 192)	172032	FALSE
123	conv2d_39	Conv2D	(None, 17, 17, 192)	172032	FALSE
124	conv2d_40	Conv2D	(None, 17, 17, 192)	147456	FALSE
125	batch_normalization_31	BatchNormalization	(None, 17, 17, 192)	576	FALSE
126	batch_normalization_34	BatchNormalization	(None, 17, 17, 192)	576	FALSE
127	batch_normalization_39	BatchNormalization	(None, 17, 17, 192)	576	FALSE
128	batch_normalization_40	BatchNormalization	(None, 17, 17, 192)	576	FALSE
129	activation_31	Activation	(None, 17, 17, 192)	0	FALSE
130	activation_34	Activation	(None, 17, 17, 192)	0	FALSE
131	activation_39	Activation	(None, 17, 17, 192)	0	FALSE

132	activation_40	Activation	(None, 17, 17, 192)	0	FALSE
133	mixed4	Concatenate	(None, 17, 17, 768)	0	FALSE
134	conv2d_45	Conv2D	(None, 17, 17, 160)	122880	FALSE
135	batch_normalization_45	BatchNormalization	(None, 17, 17, 160)	480	FALSE
136	activation_45	Activation	(None, 17, 17, 160)	0	FALSE
137	conv2d_46	Conv2D	(None, 17, 17, 160)	179200	FALSE
138	batch_normalization_46	BatchNormalization	(None, 17, 17, 160)	480	FALSE
139	activation_46	Activation	(None, 17, 17, 160)	0	FALSE
140	conv2d_42	Conv2D	(None, 17, 17, 160)	122880	FALSE
141	conv2d_47	Conv2D	(None, 17, 17, 160)	179200	FALSE
142	batch_normalization_42	BatchNormalization	(None, 17, 17, 160)	480	FALSE
143	batch_normalization_47	BatchNormalization	(None, 17, 17, 160)	480	FALSE
144	activation_42	Activation	(None, 17, 17, 160)	0	FALSE
145	activation_47	Activation	(None, 17, 17, 160)	0	FALSE
146	conv2d_43	Conv2D	(None, 17, 17, 160)	179200	FALSE
147	conv2d_48	Conv2D	(None, 17, 17, 160)	179200	FALSE
148	batch_normalization_43	BatchNormalization	(None, 17, 17, 160)	480	FALSE
149	batch_normalization_48	BatchNormalization	(None, 17, 17, 160)	480	FALSE
150	activation_43	Activation	(None, 17, 17, 160)	0	FALSE
151	activation_48	Activation	(None, 17, 17, 160)	0	FALSE
152	average_pooling2d_5	AveragePooling2D	(None, 17, 17, 768)	0	FALSE
153	conv2d_41	Conv2D	(None, 17, 17, 192)	147456	FALSE
154	conv2d_44	Conv2D	(None, 17, 17, 192)	215040	FALSE
155	conv2d_49	Conv2D	(None, 17, 17, 192)	215040	FALSE
156	conv2d_50	Conv2D	(None, 17, 17, 192)	147456	FALSE
157	batch_normalization_41	BatchNormalization	(None, 17, 17, 192)	576	FALSE
158	batch_normalization_44	BatchNormalization	(None, 17, 17, 192)	576	FALSE
159	batch_normalization_49	BatchNormalization	(None, 17, 17, 192)	576	FALSE
160	batch_normalization_50	BatchNormalization	(None, 17, 17, 192)	576	FALSE
161	activation_41	Activation	(None, 17, 17, 192)	0	FALSE
162	activation_44	Activation	(None, 17, 17, 192)	0	FALSE
163	activation_49	Activation	(None, 17, 17, 192)	0	FALSE
164	activation_50	Activation	(None, 17, 17, 192)	0	FALSE
165	mixed5	Concatenate	(None, 17, 17, 768)	0	FALSE
166	conv2d_55	Conv2D	(None, 17, 17, 160)	122880	FALSE
167	batch_normalization_55	BatchNormalization	(None, 17, 17, 160)	480	FALSE
168	activation_55	Activation	(None, 17, 17, 160)	0	FALSE
169	conv2d_56	Conv2D	(None, 17, 17, 160)	179200	FALSE
170	batch_normalization_56	BatchNormalization	(None, 17, 17, 160)	480	FALSE
171	activation_56	Activation	(None, 17, 17, 160)	0	FALSE
172	conv2d_52	Conv2D	(None, 17, 17, 160)	122880	FALSE
173	conv2d_57	Conv2D	(None, 17, 17, 160)	179200	FALSE
174	batch_normalization_52	BatchNormalization	(None, 17, 17, 160)	480	FALSE
175	batch_normalization_57	BatchNormalization	(None, 17, 17, 160)	480	FALSE
176	activation_52	Activation	(None, 17, 17, 160)	0	FALSE
177	activation_57	Activation	(None, 17, 17, 160)	0	FALSE
178	conv2d_53	Conv2D	(None, 17, 17, 160)	179200	FALSE
179	conv2d_58	Conv2D	(None, 17, 17, 160)	179200	FALSE
180	batch_normalization_53	BatchNormalization	(None, 17, 17, 160)	480	FALSE
181	batch_normalization_58	BatchNormalization	(None, 17, 17, 160)	480	FALSE
182	activation_53	Activation	(None, 17, 17, 160)	0	FALSE
183	activation_58	Activation	(None, 17, 17, 160)	0	FALSE
184	average_pooling2d_6	AveragePooling2D	(None, 17, 17, 768)	0	FALSE
185	conv2d_51	Conv2D	(None, 17, 17, 192)	147456	FALSE
186	conv2d_54	Conv2D	(None, 17, 17, 192)	215040	FALSE
187	conv2d_59	Conv2D	(None, 17, 17, 192)	215040	FALSE
188	conv2d_60	Conv2D	(None, 17, 17, 192)	147456	FALSE
189	batch_normalization_51	BatchNormalization	(None, 17, 17, 192)	576	FALSE
190	batch_normalization_54	BatchNormalization	(None, 17, 17, 192)	576	FALSE
191	batch_normalization_59	BatchNormalization	(None, 17, 17, 192)	576	FALSE
192	batch_normalization_60	BatchNormalization	(None, 17, 17, 192)	576	FALSE
193	activation_51	Activation	(None, 17, 17, 192)	0	FALSE
194	activation_54	Activation	(None, 17, 17, 192)	0	FALSE
195	activation_59	Activation	(None, 17, 17, 192)	0	FALSE
196	activation_60	Activation	(None, 17, 17, 192)	0	FALSE
197	mixed6	Concatenate	(None, 17, 17, 768)	0	FALSE

198	conv2d_65	Conv2D	(None, 17, 17, 192)	147456	FALSE
199	batch_normalization_65	BatchNormalization	(None, 17, 17, 192)	576	FALSE
200	activation_65	Activation	(None, 17, 17, 192)	0	FALSE
201	conv2d_66	Conv2D	(None, 17, 17, 192)	258048	FALSE
202	batch_normalization_66	BatchNormalization	(None, 17, 17, 192)	576	FALSE
203	activation_66	Activation	(None, 17, 17, 192)	0	FALSE
204	conv2d_62	Conv2D	(None, 17, 17, 192)	147456	FALSE
205	conv2d_67	Conv2D	(None, 17, 17, 192)	258048	FALSE
206	batch_normalization_62	BatchNormalization	(None, 17, 17, 192)	576	FALSE
207	batch_normalization_67	BatchNormalization	(None, 17, 17, 192)	576	FALSE
208	activation_62	Activation	(None, 17, 17, 192)	0	FALSE
209	activation_67	Activation	(None, 17, 17, 192)	0	FALSE
210	conv2d_63	Conv2D	(None, 17, 17, 192)	258048	FALSE
211	conv2d_68	Conv2D	(None, 17, 17, 192)	258048	FALSE
212	batch_normalization_63	BatchNormalization	(None, 17, 17, 192)	576	FALSE
213	batch_normalization_68	BatchNormalization	(None, 17, 17, 192)	576	FALSE
214	activation_63	Activation	(None, 17, 17, 192)	0	FALSE
215	activation_68	Activation	(None, 17, 17, 192)	0	FALSE
216	average_pooling2d_7	AveragePooling2D	(None, 17, 17, 768)	0	FALSE
217	conv2d_61	Conv2D	(None, 17, 17, 192)	147456	FALSE
218	conv2d_64	Conv2D	(None, 17, 17, 192)	258048	FALSE
219	conv2d_69	Conv2D	(None, 17, 17, 192)	258048	FALSE
220	conv2d_70	Conv2D	(None, 17, 17, 192)	147456	FALSE
221	batch_normalization_61	BatchNormalization	(None, 17, 17, 192)	576	FALSE
222	batch_normalization_64	BatchNormalization	(None, 17, 17, 192)	576	FALSE
223	batch_normalization_69	BatchNormalization	(None, 17, 17, 192)	576	FALSE
224	batch_normalization_70	BatchNormalization	(None, 17, 17, 192)	576	FALSE
225	activation_61	Activation	(None, 17, 17, 192)	0	FALSE
226	activation_64	Activation	(None, 17, 17, 192)	0	FALSE
227	activation_69	Activation	(None, 17, 17, 192)	0	FALSE
228	activation_70	Activation	(None, 17, 17, 192)	0	FALSE
229	mixed7	Concatenate	(None, 17, 17, 768)	0	FALSE
230	conv2d_73	Conv2D	(None, 17, 17, 192)	147456	FALSE
231	batch_normalization_73	BatchNormalization	(None, 17, 17, 192)	576	FALSE
232	activation_73	Activation	(None, 17, 17, 192)	0	FALSE
233	conv2d_74	Conv2D	(None, 17, 17, 192)	258048	FALSE
234	batch_normalization_74	BatchNormalization	(None, 17, 17, 192)	576	FALSE
235	activation_74	Activation	(None, 17, 17, 192)	0	FALSE
236	conv2d_71	Conv2D	(None, 17, 17, 192)	147456	FALSE
237	conv2d_75	Conv2D	(None, 17, 17, 192)	258048	FALSE
238	batch_normalization_71	BatchNormalization	(None, 17, 17, 192)	576	FALSE
239	batch_normalization_75	BatchNormalization	(None, 17, 17, 192)	576	FALSE
240	activation_71	Activation	(None, 17, 17, 192)	0	FALSE
241	activation_75	Activation	(None, 17, 17, 192)	0	FALSE
242	conv2d_72	Conv2D	(None, 8, 8, 320)	552960	FALSE
243	conv2d_76	Conv2D	(None, 8, 8, 192)	331776	FALSE
244	batch_normalization_72	BatchNormalization	(None, 8, 8, 320)	960	FALSE
245	batch_normalization_76	BatchNormalization	(None, 8, 8, 192)	576	FALSE
246	activation_72	Activation	(None, 8, 8, 320)	0	FALSE
247	activation_76	Activation	(None, 8, 8, 192)	0	FALSE
248	max_pooling2d_4	MaxPooling2D	(None, 8, 8, 768)	0	FALSE
249	mixed8	Concatenate	(None, 8, 8, 1280)	0	FALSE
250	conv2d_81	Conv2D	(None, 8, 8, 448)	573440	TRUE
251	batch_normalization_81	BatchNormalization	(None, 8, 8, 448)	1344	TRUE
252	activation_81	Activation	(None, 8, 8, 448)	0	TRUE
253	conv2d_78	Conv2D	(None, 8, 8, 384)	491520	TRUE
254	conv2d_82	Conv2D	(None, 8, 8, 384)	1548288	TRUE
255	batch_normalization_78	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
256	batch_normalization_82	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
257	activation_78	Activation	(None, 8, 8, 384)	0	TRUE
258	activation_82	Activation	(None, 8, 8, 384)	0	TRUE
259	conv2d_79	Conv2D	(None, 8, 8, 384)	442368	TRUE
260	conv2d_80	Conv2D	(None, 8, 8, 384)	442368	TRUE
261	conv2d_83	Conv2D	(None, 8, 8, 384)	442368	TRUE
262	conv2d_84	Conv2D	(None, 8, 8, 384)	442368	TRUE
263	average_pooling2d_8	AveragePooling2D	(None, 8, 8, 1280)	0	TRUE

264	conv2d_77	Conv2D	(None, 8, 8, 320)	409600	TRUE
265	batch_normalization_79	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
266	batch_normalization_80	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
267	batch_normalization_83	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
268	batch_normalization_84	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
269	conv2d_85	Conv2D	(None, 8, 8, 192)	245760	TRUE
270	batch_normalization_77	BatchNormalization	(None, 8, 8, 320)	960	TRUE
271	activation_79	Activation	(None, 8, 8, 384)	0	TRUE
272	activation_80	Activation	(None, 8, 8, 384)	0	TRUE
273	activation_83	Activation	(None, 8, 8, 384)	0	TRUE
274	activation_84	Activation	(None, 8, 8, 384)	0	TRUE
275	batch_normalization_85	BatchNormalization	(None, 8, 8, 192)	576	TRUE
276	activation_77	Activation	(None, 8, 8, 320)	0	TRUE
277	mixed9_0	Concatenate	(None, 8, 8, 768)	0	TRUE
278	concatenate_1	Concatenate	(None, 8, 8, 768)	0	TRUE
279	activation_85	Activation	(None, 8, 8, 192)	0	TRUE
280	mixed9	Concatenate	(None, 8, 8, 2048)	0	TRUE
281	conv2d_90	Conv2D	(None, 8, 8, 448)	917504	TRUE
282	batch_normalization_90	BatchNormalization	(None, 8, 8, 448)	1344	TRUE
283	activation_90	Activation	(None, 8, 8, 448)	0	TRUE
284	conv2d_87	Conv2D	(None, 8, 8, 384)	786432	TRUE
285	conv2d_91	Conv2D	(None, 8, 8, 384)	1548288	TRUE
286	batch_normalization_87	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
287	batch_normalization_91	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
288	activation_87	Activation	(None, 8, 8, 384)	0	TRUE
289	activation_91	Activation	(None, 8, 8, 384)	0	TRUE
290	conv2d_88	Conv2D	(None, 8, 8, 384)	442368	TRUE
291	conv2d_89	Conv2D	(None, 8, 8, 384)	442368	TRUE
292	conv2d_92	Conv2D	(None, 8, 8, 384)	442368	TRUE
293	conv2d_93	Conv2D	(None, 8, 8, 384)	442368	TRUE
294	average_pooling2d_9	AveragePooling2D	(None, 8, 8, 2048)	0	TRUE
295	conv2d_86	Conv2D	(None, 8, 8, 320)	655360	TRUE
296	batch_normalization_88	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
297	batch_normalization_89	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
298	batch_normalization_92	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
299	batch_normalization_93	BatchNormalization	(None, 8, 8, 384)	1152	TRUE
300	conv2d_94	Conv2D	(None, 8, 8, 192)	393216	TRUE
301	batch_normalization_86	BatchNormalization	(None, 8, 8, 320)	960	TRUE
302	activation_88	Activation	(None, 8, 8, 384)	0	TRUE
303	activation_89	Activation	(None, 8, 8, 384)	0	TRUE
304	activation_92	Activation	(None, 8, 8, 384)	0	TRUE
305	activation_93	Activation	(None, 8, 8, 384)	0	TRUE
306	batch_normalization_94	BatchNormalization	(None, 8, 8, 192)	576	TRUE
307	activation_86	Activation	(None, 8, 8, 320)	0	TRUE
308	mixed9_1	Concatenate	(None, 8, 8, 768)	0	TRUE
309	concatenate_2	Concatenate	(None, 8, 8, 768)	0	TRUE
310	activation_94	Activation	(None, 8, 8, 192)	0	TRUE
311	mixed10	Concatenate	(None, 8, 8, 2048)	0	TRUE
312	global_average_pooling2d_1	GlobalAveragePooling2D	(None, 2048)	0	TRUE
313	dense_1	Dense	(None, 1024)	2098176	TRUE
314	dense_2	Dense	(None, 1000)	1025000	TRUE

Table C.2: Summary of the InceptionV3 network

C.3 Xception

Number	Name	Type	Output shape	Parameters	Trainable
1	input_3	InputLayer	(None, 299, 299, 3)	0	FALSE
2	block1_conv1	Conv2D	(None, 149, 149, 32)	864	FALSE
3	block1_conv1_bn	BatchNormalization	(None, 149, 149, 32)	128	FALSE
4	block1_conv1_act	Activation	(None, 149, 149, 32)	0	FALSE
5	block1_conv2	Conv2D	(None, 147, 147, 64)	18432	FALSE

6	block1_conv2_bn	BatchNormalization	(None, 147, 147, 64)	256	FALSE
7	block1_conv2_act	Activation	(None, 147, 147, 64)	0	FALSE
8	block2_sepconv1	SeparableConv2D	(None, 147, 147, 128)	8768	FALSE
9	block2_sepconv1_bn	BatchNormalization	(None, 147, 147, 128)	512	FALSE
10	block2_sepconv2_act	Activation	(None, 147, 147, 128)	0	FALSE
11	block2_sepconv2	SeparableConv2D	(None, 147, 147, 128)	17536	FALSE
12	block2_sepconv2_bn	BatchNormalization	(None, 147, 147, 128)	512	FALSE
13	conv2d_95	Conv2D	(None, 74, 74, 128)	8192	FALSE
14	block2_pool	MaxPooling2D	(None, 74, 74, 128)	0	FALSE
15	batch_normalization_95	BatchNormalization	(None, 74, 74, 128)	512	FALSE
16	add_1	Add	(None, 74, 74, 128)	0	FALSE
17	block3_sepconv1_act	Activation	(None, 74, 74, 128)	0	FALSE
18	block3_sepconv1	SeparableConv2D	(None, 74, 74, 256)	33920	FALSE
19	block3_sepconv1_bn	BatchNormalization	(None, 74, 74, 256)	1024	FALSE
20	block3_sepconv2_act	Activation	(None, 74, 74, 256)	0	FALSE
21	block3_sepconv2	SeparableConv2D	(None, 74, 74, 256)	67840	FALSE
22	block3_sepconv2_bn	BatchNormalization	(None, 74, 74, 256)	1024	FALSE
23	conv2d_96	Conv2D	(None, 37, 37, 256)	32768	FALSE
24	block3_pool	MaxPooling2D	(None, 37, 37, 256)	0	FALSE
25	batch_normalization_96	BatchNormalization	(None, 37, 37, 256)	1024	FALSE
26	add_2	Add	(None, 37, 37, 256)	0	FALSE
27	block4_sepconv1_act	Activation	(None, 37, 37, 256)	0	FALSE
28	block4_sepconv1	SeparableConv2D	(None, 37, 37, 728)	188672	FALSE
29	block4_sepconv1_bn	BatchNormalization	(None, 37, 37, 728)	2912	FALSE
30	block4_sepconv2_act	Activation	(None, 37, 37, 728)	0	FALSE
31	block4_sepconv2	SeparableConv2D	(None, 37, 37, 728)	536536	FALSE
32	block4_sepconv2_bn	BatchNormalization	(None, 37, 37, 728)	2912	FALSE
33	conv2d_97	Conv2D	(None, 19, 19, 728)	186368	FALSE
34	block4_pool	MaxPooling2D	(None, 19, 19, 728)	0	FALSE
35	batch_normalization_97	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
36	add_3	Add	(None, 19, 19, 728)	0	FALSE
37	block5_sepconv1_act	Activation	(None, 19, 19, 728)	0	FALSE
38	block5_sepconv1	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
39	block5_sepconv1_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
40	block5_sepconv2_act	Activation	(None, 19, 19, 728)	0	FALSE
41	block5_sepconv2	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
42	block5_sepconv2_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
43	block5_sepconv3_act	Activation	(None, 19, 19, 728)	0	FALSE
44	block5_sepconv3	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
45	block5_sepconv3_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
46	add_4	Add	(None, 19, 19, 728)	0	FALSE
47	block6_sepconv1_act	Activation	(None, 19, 19, 728)	0	FALSE
48	block6_sepconv1	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
49	block6_sepconv1_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
50	block6_sepconv2_act	Activation	(None, 19, 19, 728)	0	FALSE
51	block6_sepconv2	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
52	block6_sepconv2_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
53	block6_sepconv3_act	Activation	(None, 19, 19, 728)	0	FALSE
54	block6_sepconv3	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
55	block6_sepconv3_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
56	add_5	Add	(None, 19, 19, 728)	0	FALSE
57	block7_sepconv1_act	Activation	(None, 19, 19, 728)	0	FALSE
58	block7_sepconv1	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
59	block7_sepconv1_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
60	block7_sepconv2_act	Activation	(None, 19, 19, 728)	0	FALSE
61	block7_sepconv2	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
62	block7_sepconv2_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
63	block7_sepconv3_act	Activation	(None, 19, 19, 728)	0	FALSE
64	block7_sepconv3	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
65	block7_sepconv3_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
66	add_6	Add	(None, 19, 19, 728)	0	FALSE
67	block8_sepconv1_act	Activation	(None, 19, 19, 728)	0	FALSE
68	block8_sepconv1	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
69	block8_sepconv1_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
70	block8_sepconv2_act	Activation	(None, 19, 19, 728)	0	FALSE
71	block8_sepconv2	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE

72	block8_sepconv2_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
73	block8_sepconv3_act	Activation	(None, 19, 19, 728)	0	FALSE
74	block8_sepconv3	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
75	block8_sepconv3_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
76	add_7	Add	(None, 19, 19, 728)	0	FALSE
77	block9_sepconv1_act	Activation	(None, 19, 19, 728)	0	FALSE
78	block9_sepconv1	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
79	block9_sepconv1_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
80	block9_sepconv2_act	Activation	(None, 19, 19, 728)	0	FALSE
81	block9_sepconv2	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
82	block9_sepconv2_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
83	block9_sepconv3_act	Activation	(None, 19, 19, 728)	0	FALSE
84	block9_sepconv3	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
85	block9_sepconv3_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
86	add_8	Add	(None, 19, 19, 728)	0	FALSE
87	block10_sepconv1_act	Activation	(None, 19, 19, 728)	0	FALSE
88	block10_sepconv1	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
89	block10_sepconv1_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
90	block10_sepconv2_act	Activation	(None, 19, 19, 728)	0	FALSE
91	block10_sepconv2	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
92	block10_sepconv2_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
93	block10_sepconv3_act	Activation	(None, 19, 19, 728)	0	FALSE
94	block10_sepconv3	SeparableConv2D	(None, 19, 19, 728)	536536	FALSE
95	block10_sepconv3_bn	BatchNormalization	(None, 19, 19, 728)	2912	FALSE
96	add_9	Add	(None, 19, 19, 728)	0	TRUE
97	block11_sepconv1_act	Activation	(None, 19, 19, 728)	0	TRUE
98	block11_sepconv1	SeparableConv2D	(None, 19, 19, 728)	536536	TRUE
99	block11_sepconv1_bn	BatchNormalization	(None, 19, 19, 728)	2912	TRUE
100	block11_sepconv2_act	Activation	(None, 19, 19, 728)	0	TRUE
101	block11_sepconv2	SeparableConv2D	(None, 19, 19, 728)	536536	TRUE
102	block11_sepconv2_bn	BatchNormalization	(None, 19, 19, 728)	2912	TRUE
103	block11_sepconv3_act	Activation	(None, 19, 19, 728)	0	TRUE
104	block11_sepconv3	SeparableConv2D	(None, 19, 19, 728)	536536	TRUE
105	block11_sepconv3_bn	BatchNormalization	(None, 19, 19, 728)	2912	TRUE
106	add_10	Add	(None, 19, 19, 728)	0	TRUE
107	block12_sepconv1_act	Activation	(None, 19, 19, 728)	0	TRUE
108	block12_sepconv1	SeparableConv2D	(None, 19, 19, 728)	536536	TRUE
109	block12_sepconv1_bn	BatchNormalization	(None, 19, 19, 728)	2912	TRUE
110	block12_sepconv2_act	Activation	(None, 19, 19, 728)	0	TRUE
111	block12_sepconv2	SeparableConv2D	(None, 19, 19, 728)	536536	TRUE
112	block12_sepconv2_bn	BatchNormalization	(None, 19, 19, 728)	2912	TRUE
113	block12_sepconv3_act	Activation	(None, 19, 19, 728)	0	TRUE
114	block12_sepconv3	SeparableConv2D	(None, 19, 19, 728)	536536	TRUE
115	block12_sepconv3_bn	BatchNormalization	(None, 19, 19, 728)	2912	TRUE
116	add_11	Add	(None, 19, 19, 728)	0	TRUE
117	block13_sepconv1_act	Activation	(None, 19, 19, 728)	0	TRUE
118	block13_sepconv1	SeparableConv2D	(None, 19, 19, 728)	536536	TRUE
119	block13_sepconv1_bn	BatchNormalization	(None, 19, 19, 728)	2912	TRUE
120	block13_sepconv2_act	Activation	(None, 19, 19, 728)	0	TRUE
121	block13_sepconv2	SeparableConv2D	(None, 19, 19, 1024)	752024	TRUE
122	block13_sepconv2_bn	BatchNormalization	(None, 19, 19, 1024)	4096	TRUE
123	conv2d_98	Conv2D	(None, 10, 10, 1024)	745472	TRUE
124	block13_pool	MaxPooling2D	(None, 10, 10, 1024)	0	TRUE
125	batch_normalization_98	BatchNormalization	(None, 10, 10, 1024)	4096	TRUE
126	add_12	Add	(None, 10, 10, 1024)	0	TRUE
127	block14_sepconv1	SeparableConv2D	(None, 10, 10, 1536)	1582080	TRUE
128	block14_sepconv1_bn	BatchNormalization	(None, 10, 10, 1536)	6144	TRUE
129	block14_sepconv1_act	Activation	(None, 10, 10, 1536)	0	TRUE
130	block14_sepconv2	SeparableConv2D	(None, 10, 10, 2048)	3159552	TRUE
131	block14_sepconv2_bn	BatchNormalization	(None, 10, 10, 2048)	8192	TRUE
132	block14_sepconv2_act	Activation	(None, 10, 10, 2048)	0	TRUE
133	global_average_pooling2d_2	GlobalAveragePooling2D	(None, 2048)	0	TRUE
134	dense_3	Dense	(None, 1000)	2049000	TRUE

Table C.3: Summary of the Xception network

Appendix D

Complete results per data set

D.1 Roof shape

		VGG			**Inception**			**Xception**		
precision	flat	0.88	±	0.01	0.78	±	0.03	0.83	±	0.03
	hipped	0.87	±	0.02	0.69	±	0.06	0.87	±	0.04
recall	flat	0.94	±	0.01	0.87	±	0.04	0.95	±	0.02
	hipped	0.76	±	0.02	0.55	±	0.09	0.65	±	0.08
fscore	flat	0.91	±	0.01	0.82	±	0.02	0.89	±	0.01
	hipped	0.81	±	0.01	0.61	±	0.06	0.74	±	0.05
accuracy	overall	0.88	±	0.01	0.75	±	0.03	0.84	±	0.02
kappa	overall	0.72	±	0.02	0.43	±	0.07	0.63	±	0.06

		None			**BOW**			**VLAD**		
precision	flat	0.87	±	0.01	0.86	±	0.01	0.88	±	0.01
	hipped	0.84	±	0.01	0.83	±	0.02	0.87	±	0.02
recall	flat	0.92	±	0.01	0.92	±	0.01	0.94	±	0.01
	hipped	0.75	±	0.02	0.74	±	0.02	0.76	±	0.02
fscore	flat	0.90	±	0.01	0.89	±	0.01	0.91	±	0.00
	hipped	0.79	±	0.02	0.78	±	0.02	0.81	±	0.01
accuracy	overall	0.86	±	0.01	0.85	±	0.01	0.87	±	0.00
kappa	overall	0.69	±	0.02	0.67	±	0.02	0.72	±	0.01

		LLC			**IFK**		
precision	flat	0.86	±	0.01	0.82	±	0.01
	hipped	0.81	±	0.03	0.89	±	0.02

recall	flat	0.91	±	0.02	0.96	±	0.01
	hipped	0.73	±	0.03	0.61	±	0.03
fscore	flat	0.88	±	0.01	0.88	±	0.01
	hipped	0.77	±	0.01	0.73	±	0.02
accuracy	overall	0.84	±	0.01	0.84	±	0.01
kappa	overall	0.65	±	0.02	0.62	±	0.02

		RF			**SVM**		
precision	flat	0.87	±	0.01	0.88	±	0.01
	hipped	0.88	±	0.02	0.81	±	0.04
recall	flat	0.94	±	0.01	0.90	±	0.03
	hipped	0.75	±	0.03	0.78	±	0.03
fscore	flat	0.91	±	0.01	0.89	±	0.01
	hipped	0.81	±	0.01	0.79	±	0.02
accuracy	overall	0.87	±	0.01	0.86	±	0.02
kappa	overall	0.72	±	0.02	0.68	±	0.03

Table D.1: Overview of all performance metrics for the roof shape data set

D.2 Roof Material

		VGG			Inception			Xception		
precision	concrete	0.70	±	0.05	0.78	±	0.06	0.72	±	0.07
	metal	0.71	±	0.01	0.60	±	0.01	0.65	±	0.02
	roof tiles	0.63	±	0.05	0.69	±	0.14	0.66	±	0.08
recall	concrete	0.50	±	0.07	0.14	±	0.07	0.31	±	0.11
	metal	0.86	±	0.04	0.98	±	0.02	0.92	±	0.03
	roof tiles	0.38	±	0.07	0.02	±	0.02	0.21	±	0.09
fscore	concrete	0.58	±	0.03	0.22	±	0.09	0.41	±	0.10
	metal	0.78	±	0.01	0.75	±	0.00	0.76	±	0.01
	roof tiles	0.47	±	0.04	0.03	±	0.03	0.30	±	0.11
accuracy	overall	0.69	±	0.00	0.61	±	0.01	0.65	±	0.01
kappa	overall	0.41	±	0.01	0.10	±	0.04	0.27	±	0.04

		None			BOW			VLAD		
precision	concrete	0.60	±	0.01	0.60	±	0.03	0.67	±	0.04
	metal	0.71	±	0.01	0.70	±	0.01	0.71	±	0.01
	roof tiles	0.50	±	0.05	0.53	±	0.03	0.61	±	0.03
recall	concrete	0.53	±	0.05	0.51	±	0.03	0.53	±	0.04
	metal	0.78	±	0.03	0.80	±	0.03	0.84	±	0.04
	roof tiles	0.40	±	0.03	0.37	±	0.04	0.40	±	0.05
fscore	concrete	0.56	±	0.03	0.55	±	0.02	0.59	±	0.02
	metal	0.74	±	0.01	0.74	±	0.01	0.77	±	0.01
	roof tiles	0.44	±	0.02	0.44	±	0.03	0.48	±	0.03
accuracy	overall	0.66	±	0.01	0.66	±	0.01	0.69	±	0.01
kappa	overall	0.37	±	0.02	0.36	±	0.02	0.42	±	0.01

		LLC			IFK		
precision	concrete	0.56	±	0.03	0.67	±	0.02
	metal	0.69	±	0.01	0.65	±	0.01
	roof tiles	0.47	±	0.03	0.62	±	0.02
recall	concrete	0.52	±	0.03	0.36	±	0.04
	metal	0.76	±	0.02	0.90	±	0.02
	roof tiles	0.35	±	0.03	0.20	±	0.04
fscore	concrete	0.54	±	0.03	0.47	±	0.04
	metal	0.73	±	0.01	0.76	±	0.00
	roof tiles	0.40	±	0.02	0.30	±	0.05
accuracy	overall	0.64	±	0.01	0.65	±	0.01
kappa	overall	0.33	±	0.02	0.28	±	0.03

		RF			SVM		
precision	concrete	0.77	±	0.04	0.58	±	0.03
	metal	0.69	±	0.01	0.71	±	0.01
	roof tiles	0.69	±	0.03	0.47	±	0.05
recall	concrete	0.45	±	0.04	0.54	±	0.04
	metal	0.91	±	0.02	0.74	±	0.03
	roof tiles	0.34	±	0.04	0.43	±	0.02
fscore	concrete	0.56	±	0.02	0.56	±	0.03
	metal	0.79	±	0.00	0.73	±	0.02
	roof tiles	0.45	±	0.03	0.45	±	0.02
accuracy	overall	0.70	±	0.01	0.64	±	0.02
kappa	overall	0.40	±	0.02	0.36	±	0.03

Table D.2: Overview of all performance metrics for the roof material data set

D.3 UC-Merced

		VGG			Inception			Xception		
precision	agricultural	0.99	±	0.03	0.98	±	0.04	0.91	±	0.13
	airplane	1.00	±	0.00	1.00	±	0.00	0.98	±	0.02
	baseballdiamond	0.95	±	0.05	1.00	±	0.00	1.00	±	0.00
	beach	0.99	±	0.02	0.93	±	0.05	0.87	±	0.06
	buildings	0.88	±	0.02	0.89	±	0.03	0.84	±	0.04
	chaparral	0.97	±	0.04	0.97	±	0.04	1.00	±	0.00
	denseresidential	0.72	±	0.17	0.54	±	0.05	0.68	±	0.09
	forest	0.95	±	0.04	0.97	±	0.04	0.99	±	0.02
	freeway	0.85	±	0.14	0.73	±	0.16	0.85	±	0.13
	golfcourse	0.96	±	0.06	0.92	±	0.03	0.97	±	0.04
	harbor	0.99	±	0.02	0.95	±	0.04	0.98	±	0.04
	intersection	0.89	±	0.06	0.79	±	0.07	0.97	±	0.05
	mediumresidential	0.71	±	0.11	0.92	±	0.11	0.78	±	0.13
	mobilehomepark	0.84	±	0.15	0.90	±	0.08	0.91	±	0.07
	overpass	0.96	±	0.02	0.96	±	0.05	0.99	±	0.02
	parkinglot	1.00	±	0.00	0.78	±	0.11	0.94	±	0.06
	river	0.89	±	0.08	0.82	±	0.17	0.91	±	0.12
	runway	0.86	±	0.13	0.93	±	0.09	0.92	±	0.07
	sparseresidential	0.85	±	0.01	0.86	±	0.11	0.93	±	0.05
	storagetanks	0.97	±	0.04	0.87	±	0.08	0.85	±	0.06
	tenniscourt	0.93	±	0.07	0.97	±	0.04	0.99	±	0.02
recall	agricultural	0.93	±	0.12	0.98	±	0.04	0.99	±	0.02
	airplane	0.98	±	0.02	0.94	±	0.05	0.96	±	0.04
	baseballdiamond	0.98	±	0.02	0.70	±	0.03	0.89	±	0.05
	beach	1.00	±	0.00	0.99	±	0.02	0.99	±	0.02
	buildings	0.85	±	0.09	0.89	±	0.11	0.88	±	0.09
	chaparral	0.98	±	0.02	0.98	±	0.04	0.92	±	0.14
	denseresidential	0.64	±	0.21	0.80	±	0.13	0.73	±	0.16
	forest	1.00	±	0.00	0.97	±	0.06	0.98	±	0.04
	freeway	0.92	±	0.14	0.98	±	0.02	1.00	±	0.00
	golfcourse	0.88	±	0.14	0.81	±	0.22	0.80	±	0.17
	harbor	1.00	±	0.00	1.00	±	0.00	1.00	±	0.00
	intersection	0.87	±	0.12	0.89	±	0.15	0.93	±	0.09
	mediumresidential	0.94	±	0.04	0.41	±	0.32	0.71	±	0.19
	mobilehomepark	0.74	±	0.18	0.83	±	0.10	0.92	±	0.07
	overpass	0.88	±	0.07	0.78	±	0.11	0.91	±	0.06
	parkinglot	1.00	±	0.00	1.00	±	0.00	1.00	±	0.00
	river	0.89	±	0.05	0.85	±	0.14	0.74	±	0.18
	runway	0.81	±	0.26	0.76	±	0.30	0.90	±	0.14
	sparseresidential	0.87	±	0.07	0.83	±	0.10	0.85	±	0.08
	storagetanks	0.90	±	0.04	0.83	±	0.15	0.95	±	0.06
	tenniscourt	0.91	±	0.09	0.92	±	0.09	0.96	±	0.04
fscore	agricultural	0.95	±	0.08	0.98	±	0.02	0.95	±	0.07
	airplane	0.99	±	0.01	0.97	±	0.03	0.97	±	0.02
	baseballdiamond	0.96	±	0.02	0.82	±	0.02	0.94	±	0.03
	beach	1.00	±	0.01	0.96	±	0.03	0.92	±	0.03
	buildings	0.86	±	0.06	0.89	±	0.06	0.86	±	0.06
	chaparral	0.98	±	0.02	0.98	±	0.02	0.95	±	0.08

	denseresidential	0.67	±	0.19	0.64	±	0.06	0.70	±	0.11
	forest	0.98	±	0.02	0.97	±	0.05	0.98	±	0.02
	freeway	0.87	±	0.09	0.83	±	0.11	0.91	±	0.08
	golfcourse	0.91	±	0.08	0.84	±	0.15	0.86	±	0.11
	harbor	1.00	±	0.01	0.98	±	0.02	0.99	±	0.02
	intersection	0.87	±	0.05	0.83	±	0.07	0.95	±	0.06
	mediumresidential	0.81	±	0.07	0.48	±	0.29	0.72	±	0.11
	mobilehomepark	0.77	±	0.14	0.85	±	0.05	0.91	±	0.05
	overpass	0.92	±	0.04	0.86	±	0.08	0.95	±	0.04
	parkinglot	1.00	±	0.00	0.87	±	0.07	0.97	±	0.03
	river	0.89	±	0.04	0.83	±	0.14	0.80	±	0.13
	runway	0.81	±	0.19	0.80	±	0.22	0.90	±	0.08
	sparseresidential	0.86	±	0.04	0.84	±	0.07	0.88	±	0.05
	storagetanks	0.93	±	0.03	0.85	±	0.11	0.89	±	0.04
	tenniscourt	0.92	±	0.05	0.94	±	0.05	0.97	±	0.02
accuracy	overall	0.90	±	0.03	0.86	±	0.03	0.91	±	0.03
kappa	overall	0.90	±	0.03	0.86	±	0.04	0.90	±	0.03

		None			**BOW**			**VLAD**		
precision	agricultural	0.92	±	0.07	0.93	±	0.10	0.98	±	0.04
	airplane	0.98	±	0.02	0.97	±	0.04	0.97	±	0.02
	baseballdiamond	0.99	±	0.02	0.92	±	0.03	0.94	±	0.04
	beach	0.95	±	0.04	0.99	±	0.02	0.99	±	0.02
	buildings	0.91	±	0.08	0.83	±	0.04	0.88	±	0.08
	chaparral	0.99	±	0.02	0.95	±	0.04	0.96	±	0.03
	denseresidential	0.74	±	0.17	0.70	±	0.16	0.76	±	0.14
	forest	0.94	±	0.09	0.93	±	0.04	0.98	±	0.02
	freeway	0.95	±	0.03	0.90	±	0.09	0.89	±	0.08
	golfcourse	0.89	±	0.10	0.90	±	0.10	0.92	±	0.15
	harbor	0.98	±	0.04	0.97	±	0.04	1.00	±	0.00
	intersection	0.98	±	0.02	0.94	±	0.08	0.92	±	0.08
	mediumresidential	0.70	±	0.11	0.70	±	0.10	0.76	±	0.09
	mobilehomepark	0.90	±	0.08	0.88	±	0.14	0.90	±	0.09
	overpass	0.95	±	0.06	0.94	±	0.05	0.92	±	0.05
	parkinglot	1.00	±	0.00	0.97	±	0.04	0.97	±	0.02
	river	0.92	±	0.07	0.89	±	0.04	0.92	±	0.07
	runway	0.93	±	0.09	0.92	±	0.09	0.95	±	0.04
	sparseresidential	0.92	±	0.05	0.78	±	0.09	0.86	±	0.04
	storagetanks	0.96	±	0.02	0.95	±	0.05	0.97	±	0.04
	tenniscourt	0.97	±	0.03	0.96	±	0.06	0.93	±	0.06
recall	agricultural	0.98	±	0.02	0.92	±	0.09	0.94	±	0.07
	airplane	0.99	±	0.02	0.99	±	0.02	1.00	±	0.00
	baseballdiamond	0.96	±	0.04	0.99	±	0.02	0.99	±	0.02
	beach	1.00	±	0.00	0.96	±	0.04	1.00	±	0.00
	buildings	0.82	±	0.10	0.86	±	0.07	0.79	±	0.10
	chaparral	0.95	±	0.10	0.95	±	0.10	0.98	±	0.04
	denseresidential	0.64	±	0.15	0.61	±	0.15	0.61	±	0.14
	forest	1.00	±	0.00	1.00	±	0.00	1.00	±	0.00
	freeway	0.91	±	0.10	0.93	±	0.10	0.95	±	0.06

	golfcourse	0.93	±	0.10	0.91	±	0.07	0.91	±	0.07
	harbor	1.00	±	0.00	1.00	±	0.00	1.00	±	0.00
	intersection	0.98	±	0.02	0.88	±	0.10	0.89	±	0.06
	mediumresidential	0.91	±	0.08	0.82	±	0.08	0.90	±	0.05
	mobilehomepark	0.87	±	0.09	0.84	±	0.17	0.91	±	0.10
	overpass	0.98	±	0.02	0.93	±	0.07	0.89	±	0.11
	parkinglot	0.99	±	0.02	0.99	±	0.02	1.00	±	0.00
	river	0.83	±	0.10	0.88	±	0.14	0.91	±	0.07
	runway	0.96	±	0.04	0.88	±	0.17	0.96	±	0.04
	sparseresidential	0.84	±	0.09	0.71	±	0.14	0.79	±	0.13
	storagetanks	0.90	±	0.05	0.82	±	0.09	0.92	±	0.09
	tenniscourt	0.87	±	0.05	0.91	±	0.07	0.94	±	0.06
fscore	agricultural	0.95	±	0.04	0.92	±	0.06	0.96	±	0.04
	airplane	0.99	±	0.01	0.98	±	0.02	0.99	±	0.01
	baseballdiamond	0.97	±	0.02	0.95	±	0.02	0.96	±	0.02
	beach	0.97	±	0.02	0.97	±	0.02	1.00	±	0.01
	buildings	0.86	±	0.07	0.84	±	0.03	0.83	±	0.08
	chaparral	0.97	±	0.06	0.95	±	0.05	0.97	±	0.02
	denseresidential	0.68	±	0.14	0.65	±	0.15	0.67	±	0.14
	forest	0.97	±	0.05	0.96	±	0.02	0.99	±	0.01
	freeway	0.93	±	0.05	0.91	±	0.07	0.92	±	0.06
	golfcourse	0.90	±	0.06	0.90	±	0.03	0.90	±	0.07
	harbor	0.99	±	0.02	0.99	±	0.02	1.00	±	0.00
	intersection	0.98	±	0.02	0.90	±	0.06	0.90	±	0.06
	mediumresidential	0.79	±	0.07	0.75	±	0.09	0.82	±	0.06
	mobilehomepark	0.88	±	0.02	0.84	±	0.11	0.90	±	0.04
	overpass	0.97	±	0.04	0.93	±	0.04	0.90	±	0.05
	parkinglot	0.99	±	0.01	0.98	±	0.02	0.99	±	0.01
	river	0.87	±	0.06	0.88	±	0.09	0.91	±	0.05
	runway	0.94	±	0.05	0.89	±	0.10	0.96	±	0.02
	sparseresidential	0.87	±	0.06	0.73	±	0.09	0.82	±	0.09
	storagetanks	0.93	±	0.03	0.88	±	0.07	0.94	±	0.05
	tenniscourt	0.92	±	0.04	0.93	±	0.06	0.94	±	0.05
accuracy	overall	0.92	±	0.01	0.89	±	0.03	0.92	±	0.01
kappa	overall	0.92	±	0.01	0.89	±	0.03	0.91	±	0.01

		LLC			**IFK**		
precision	agricultural	0.98	±	0.04	0.94	±	0.04
	airplane	0.96	±	0.03	0.96	±	0.04
	baseballdiamond	0.94	±	0.04	0.90	±	0.06
	beach	0.95	±	0.03	0.94	±	0.04
	buildings	0.86	±	0.03	0.75	±	0.04
	chaparral	0.97	±	0.05	0.95	±	0.04
	denseresidential	0.71	±	0.12	0.62	±	0.17
	forest	0.95	±	0.04	0.89	±	0.08
	freeway	0.88	±	0.05	0.86	±	0.06
	golfcourse	0.89	±	0.10	0.86	±	0.14
	harbor	0.96	±	0.06	0.99	±	0.02
	intersection	0.90	±	0.08	0.90	±	0.06

	mediumresidential	0.68	±	0.08	0.66	±	0.14
	mobilehomepark	0.92	±	0.10	0.80	±	0.13
	overpass	0.89	±	0.08	0.92	±	0.07
	parkinglot	1.00	±	0.00	0.94	±	0.06
	river	0.90	±	0.10	0.86	±	0.05
	runway	0.86	±	0.09	0.89	±	0.08
	sparseresidential	0.88	±	0.03	0.76	±	0.10
	storagetanks	0.95	±	0.03	0.95	±	0.05
	tenniscourt	0.92	±	0.05	0.89	±	0.04
recall	agricultural	0.89	±	0.15	0.94	±	0.07
	airplane	0.99	±	0.02	0.98	±	0.02
	baseballdiamond	0.96	±	0.05	0.98	±	0.02
	beach	1.00	±	0.00	0.99	±	0.02
	buildings	0.78	±	0.07	0.74	±	0.09
	chaparral	0.97	±	0.06	0.97	±	0.06
	denseresidential	0.60	±	0.10	0.48	±	0.21
	forest	1.00	±	0.00	1.00	±	0.00
	freeway	0.91	±	0.10	0.92	±	0.07
	golfcourse	0.90	±	0.10	0.88	±	0.07
	harbor	1.00	±	0.00	0.99	±	0.02
	intersection	0.88	±	0.08	0.75	±	0.08
	mediumresidential	0.84	±	0.14	0.73	±	0.09
	mobilehomepark	0.83	±	0.16	0.87	±	0.12
	overpass	0.90	±	0.06	0.85	±	0.11
	parkinglot	0.99	±	0.02	0.99	±	0.02
	river	0.82	±	0.11	0.78	±	0.09
	runway	0.90	±	0.06	0.91	±	0.06
	sparseresidential	0.84	±	0.13	0.65	±	0.20
	storagetanks	0.88	±	0.05	0.88	±	0.13
	tenniscourt	0.93	±	0.04	0.86	±	0.16
fscore	agricultural	0.92	±	0.09	0.94	±	0.04
	airplane	0.98	±	0.02	0.97	±	0.03
	baseballdiamond	0.95	±	0.02	0.94	±	0.03
	beach	0.98	±	0.02	0.97	±	0.03
	buildings	0.82	±	0.06	0.74	±	0.04
	chaparral	0.97	±	0.04	0.96	±	0.03
	denseresidential	0.64	±	0.10	0.53	±	0.19
	forest	0.98	±	0.02	0.94	±	0.05
	freeway	0.89	±	0.06	0.89	±	0.04
	golfcourse	0.89	±	0.08	0.86	±	0.09
	harbor	0.98	±	0.04	0.99	±	0.01
	intersection	0.89	±	0.06	0.81	±	0.06
	mediumresidential	0.74	±	0.09	0.68	±	0.09
	mobilehomepark	0.86	±	0.09	0.82	±	0.09
	overpass	0.89	±	0.06	0.88	±	0.07
	parkinglot	0.99	±	0.01	0.96	±	0.04
	river	0.85	±	0.08	0.81	±	0.06
	runway	0.87	±	0.04	0.90	±	0.05
	sparseresidential	0.85	±	0.07	0.69	±	0.15
	storagetanks	0.91	±	0.02	0.91	±	0.09

	tenniscourt	0.93	±	0.03	0.86	±	0.10
accuracy	overall	0.90	±	0.02	0.86	±	0.02
kappa	overall	0.89	±	0.02	0.86	±	0.02

			RF			**SVM**	
precision	agricultural	0.99	±	0.02	0.98	±	0.02
	airplane	0.98	±	0.02	0.98	±	0.02
	baseballdiamond	0.91	±	0.04	0.93	±	0.05
	beach	0.96	±	0.03	0.98	±	0.02
	buildings	0.79	±	0.04	0.89	±	0.04
	chaparral	0.97	±	0.04	1.00	±	0.00
	denseresidential	0.68	±	0.17	0.60	±	0.19
	forest	0.96	±	0.04	0.97	±	0.04
	freeway	0.92	±	0.07	0.97	±	0.04
	golfcourse	0.90	±	0.10	0.94	±	0.03
	harbor	0.98	±	0.02	0.99	±	0.02
	intersection	0.92	±	0.05	0.92	±	0.05
	mediumresidential	0.72	±	0.10	0.70	±	0.13
	mobilehomepark	0.82	±	0.15	0.86	±	0.09
	overpass	0.91	±	0.10	0.96	±	0.04
	parkinglot	0.98	±	0.02	0.99	±	0.02
	river	0.85	±	0.13	0.92	±	0.08
	runway	0.91	±	0.09	0.97	±	0.05
	sparseresidential	0.82	±	0.08	0.89	±	0.05
	storagetanks	0.99	±	0.02	1.00	±	0.00
	tenniscourt	0.93	±	0.08	0.94	±	0.07
recall	agricultural	0.92	±	0.14	0.98	±	0.02
	airplane	0.98	±	0.02	0.98	±	0.02
	baseballdiamond	0.99	±	0.02	0.98	±	0.04
	beach	0.99	±	0.02	1.00	±	0.00
	buildings	0.80	±	0.12	0.87	±	0.07
	chaparral	0.99	±	0.02	0.99	±	0.02
	denseresidential	0.57	±	0.19	0.62	±	0.27
	forest	0.99	±	0.02	1.00	±	0.00
	freeway	0.96	±	0.06	0.98	±	0.04
	golfcourse	0.83	±	0.17	0.92	±	0.09
	harbor	1.00	±	0.00	0.99	±	0.02
	intersection	0.83	±	0.18	0.84	±	0.16
	mediumresidential	0.87	±	0.06	0.81	±	0.14
	mobilehomepark	0.80	±	0.11	0.80	±	0.20
	overpass	0.94	±	0.04	0.93	±	0.02
	parkinglot	1.00	±	0.00	1.00	±	0.00
	river	0.90	±	0.04	0.94	±	0.04
	runway	0.87	±	0.10	0.97	±	0.04
	sparseresidential	0.80	±	0.10	0.92	±	0.07
	storagetanks	0.89	±	0.08	0.90	±	0.13
	tenniscourt	0.87	±	0.09	0.90	±	0.11
fscore	agricultural	0.95	±	0.08	0.98	±	0.02

	airplane	0.98	±	0.02	0.98	±	0.02
	baseballdiamond	0.95	±	0.02	0.95	±	0.03
	beach	0.98	±	0.02	0.99	±	0.01
	buildings	0.79	±	0.07	0.88	±	0.05
	chaparral	0.98	±	0.02	0.99	±	0.01
	denseresidential	0.62	±	0.19	0.60	±	0.24
	forest	0.98	±	0.02	0.99	±	0.02
	freeway	0.94	±	0.05	0.98	±	0.02
	golfcourse	0.85	±	0.11	0.93	±	0.04
	harbor	0.99	±	0.01	0.99	±	0.01
	intersection	0.86	±	0.12	0.87	±	0.11
	mediumresidential	0.78	±	0.05	0.73	±	0.06
	mobilehomepark	0.80	±	0.10	0.82	±	0.12
	overpass	0.92	±	0.07	0.94	±	0.01
	parkinglot	0.99	±	0.01	1.00	±	0.01
	river	0.87	±	0.06	0.93	±	0.04
	runway	0.89	±	0.09	0.97	±	0.04
	sparseresidential	0.81	±	0.08	0.91	±	0.05
	storagetanks	0.94	±	0.06	0.94	±	0.08
	tenniscourt	0.90	±	0.09	0.92	±	0.07
accuracy	overall	0.89	±	0.02	0.92	±	0.02
kappa	overall	0.89	±	0.02	0.92	±	0.02

Table D.3: Overview of all performance metrics for the UC-Merced data set

Appendix E

Complete results significance tests

E.1 Networks

Shape		**Accuracy**				**Kappa**		
	Wilcoxon				Wilcoxon			
		VGG	Inception	Xception		VGG	Inception	Xception
	VGG	nan	0.005	0.005	VGG	nan	0.005	0.005
	Inception	0.005	nan	0.005	Inception	0.005	nan	0.005
	Xception	0.005	0.005	nan	Xception	0.005	0.005	nan
		Tw	P			Tw	P	
	FRIEDMAN	20.000	0.000		FRIEDMAN	20.000	0.000	
Material		**Accuracy**				**Kappa**		
	Wilcoxon				Wilcoxon			
		VGG	Inception	Xception		VGG	Inception	Xception
	VGG	nan	0.005	0.005	VGG	nan	0.005	0.005
	Inception	0.005	nan	0.005	Inception	0.005	nan	0.005
	Xception	0.005	0.005	nan	Xception	0.005	0.005	nan
		Tw	P			Tw	P	
	FRIEDMAN	20.000	0.000		FRIEDMAN	20.000	0.000	
UC-Merced		**Accuracy**				**Kappa**		
	Wilcoxon				Wilcoxon			
		VGG	Inception	Xception		VGG	Inception	Xception
	VGG	nan	0.080	0.593	VGG	nan	0.080	0.593
	Inception	0.080	nan	0.043	Inception	0.080	nan	0.043
	Xception	0.593	0.043	nan	Xception	0.593	0.043	nan
		Tw	P			Tw	P	
	FRIEDMAN	5.778	0.056		FRIEDMAN	5.778	0.056	

Table E.1: Overview of Wilcoxon and Friedman tests for the fine-tuned VGG-16, InceptionV3 and Xception network

E.2 Coding

Shape	**Accuracy**							**Kappa**					
	Wilcoxon							Wilcoxon					
		None	BOW	VLAD	LLC	IFK	cnn		None	BOW	VLAD	LLC	IFK	cnn
	None	nan	0.01	0.01	0.01	0.01	0.01	None	nan	0.01	0.01	0.01	0.01	0.01
	BOW	0.01	nan	0.01	0.01	0.01	0.01	BOW	0.01	nan	0.01	0.01	0.01	0.01
	VLAD	0.01	0.01	nan	0.01	0.01	0.28	VLAD	0.01	0.01	nan	0.01	0.01	0.28
	LLC	0.01	0.01	0.01	nan	0.02	0.01	LLC	0.01	0.01	0.01	nan	0.01	0.01
	IFK	0.01	0.01	0.01	0.02	nan	0.01	IFK	0.01	0.01	0.01	0.01	nan	0.01
	cnn	0.01	0.01	0.28	0.01	0.01	nan	cnn	0.01	0.01	0.28	0.01	0.01	nan
		Tw	P						Tw	P				
	FRIEDMAN	43.66	0.00					FRIEDMAN	44.51	0.00				
Material	**Accuracy**							**Kappa**					
	Wilcoxon							Wilcoxon					
		None	BOW	VLAD	LLC	IFK	cnn		None	BOW	VLAD	LLC	IFK	cnn
	None	nan	0.65	0.01	0.01	0.72	0.01	None	nan	0.11	0.01	0.01	0.01	0.01
	BOW	0.65	nan	0.01	0.01	0.33	0.01	BOW	0.11	nan	0.01	0.01	0.01	0.01
	VLAD	0.01	0.01	nan	0.01	0.01	0.88	VLAD	0.01	0.01	nan	0.01	0.01	0.14
	LLC	0.01	0.01	0.01	nan	0.01	0.01	LLC	0.01	0.01	0.01	nan	0.01	0.01
	IFK	0.72	0.33	0.01	0.01	nan	0.01	IFK	0.01	0.01	0.01	0.01	nan	0.01
	cnn	0.01	0.01	0.88	0.01	0.01	nan	cnn	0.01	0.01	0.14	0.01	0.01	nan
		Tw	P						Tw	P				
	FRIEDMAN	43.37	0.00					FRIEDMAN	46.91	0.00				
UC-Merced	**Accuracy**							**Kappa**					
	Wilcoxon							Wilcoxon					
		None	VLAD	LLC	IFK	BOW	cnn		None	VLAD	LLC	IFK	BOW	cnn
	None	nan	0.85	0.04	0.04	0.04	0.08	None	nan	0.72	0.04	0.04	0.04	0.08
	VLAD	0.85	nan	0.04	0.04	0.14	0.35	VLAD	0.72	nan	0.04	0.04	0.14	0.35
	LLC	0.04	0.04	nan	0.08	1.00	0.50	LLC	0.04	0.04	nan	0.08	1.00	0.50
	IFK	0.04	0.04	0.08	nan	0.04	0.07	IFK	0.04	0.04	0.08	nan	0.04	0.07
	BOW	0.04	0.14	1.00	0.04	nan	0.50	BOW	0.04	0.14	1.00	0.04	nan	0.50
	cnn	0.08	0.35	0.50	0.07	0.50	nan	cnn	0.08	0.35	0.50	0.07	0.50	nan
		Tw	P						Tw	P				
	FRIEDMAN	16.63	0.01					FRIEDMAN	16.63	0.01				

Table E.2: Overview of Wilcoxon and Friedman tests for the feature coding models and the VGG-16 base line

E.3 Classifiers

Shape	**Accuracy**				**Kappa**			
	Wilcoxon				Wilcoxon			
		RF	SVM	VGG		RF	SVM	VGG
	RF	nan	0.01	0.26	RF	nan	0.01	0.20
	SVM	0.01	nan	0.01	SVM	0.01	nan	0.01
	VGG	0.26	0.01	nan	VGG	0.20	0.01	nan
		Tw	P			Tw	P	
	FRIEDMAN	15.85	0.00		FRIEDMAN	15.80	0.00	
Material	**Accuracy**				**Kappa**			
	Wilcoxon				Wilcoxon			
		RF	SVM	VGG		RF	SVM	VGG
	RF	nan	0.01	0.01	RF	nan	0.01	0.11
	SVM	0.01	nan	0.01	SVM	0.01	nan	0.01
	VGG	0.01	0.01	nan	VGG	0.11	0.01	nan
		Tw	P			Tw	P	
	FRIEDMAN	18.20	0.00		FRIEDMAN	15.80	0.00	
UC-Merced	**Accuracy**				**Kappa**			
	Wilcoxon				Wilcoxon			
		RF	SVM	VGG		RF	SVM	VGG
	RF	nan	0.04	0.14	RF	nan	0.04	0.20
	SVM	0.04	nan	0.08	SVM	0.04	nan	0.08
	VGG	0.14	0.08	nan	VGG	0.20	0.08	nan
		Tw	P			Tw	P	
	FRIEDMAN	6.00	0.05		FRIEDMAN	6.00	0.05	

Table E.3: Overview of Wilcoxon and Friedman tests for the RF model, the SVM model and the VGG-16 base line

Appendix F

Random Search Optimal Parameters

	Fold	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**
Shape	1	200	5	1	sqrt	60
	2	100	5	2	sqrt	60
	3	800	2	1	sqrt	100
	4	400	5	1	sqrt	80
	5	200	2	1	sqrt	40
	6	800	2	1	sqrt	100
	7	1600	5	1	sqrt	80
	8	200	5	1	sqrt	None
	9	200	2	1	sqrt	100
	10	200	2	1	sqrt	40

	Fold	[1]	[2]	[3]	[4]	[5]
Material	1	800	2	1	sqrt	80
	2	1600	2	1	sqrt	20
	3	100	2	1	sqrt	60
	4	200	2	1	sqrt	80
	5	1600	2	1	sqrt	100
	6	800	2	1	sqrt	40
	7	200	2	1	sqrt	80
	8	1600	2	1	sqrt	80
	9	800	2	1	sqrt	80
	10	400	2	1	sqrt	40

	Fold	[1]	[2]	[3]	[4]	[5]
UC-Merced	1	200	5	1	sqrt	80
	2	1600	2	2	sqrt	40
	3	1600	2	2	sqrt	20
	4	1600	2	1	sqrt	60
	5	400	2	1	sqrt	60

[1]	Number of trees
[2]	Minimum samples at split
[3]	minimum samples at leaf
[4]	maximum features
[5]	Max depth

Table F.1: Optimal parameters found for RF by the Random Search

Appendix G

Pre-trained CNNs as Feature Extractor

		VGG			**InceptionV3**			**Xception**		
precision	flat	0.80	±	0.01	0.84	±	0.02	0.86	±	0.01
	hipped	0.70	±	0.05	0.74	±	0.04	0.76	±	0.04
recall	flat	0.86	±	0.04	0.87	±	0.03	0.88	±	0.03
	hipped	0.60	±	0.03	0.69	±	0.04	0.73	±	0.03
fscore	flat	0.83	±	0.02	0.85	±	0.01	0.87	±	0.01
	hipped	0.65	±	0.01	0.72	±	0.02	0.74	±	0.02
accuracy	overall	0.77	±	0.02	0.81	±	0.02	0.82	±	0.02
kappa	overall	0.47	±	0.03	0.57	±	0.03	0.61	±	0.03

Table G.1: Results of pre-trained networks (no fine-tuning) as feature extractors on the roof shape data set

		VGG			**InceptionV3**			**Xception**		
precision	concrete	0.43	±	0.02	0.49	±	0.04	0.50	±	0.03
	metal	0.66	±	0.01	0.68	±	0.01	0.68	±	0.01
	roof tiles	0.36	±	0.04	0.40	±	0.06	0.44	±	0.05
recall	concrete	0.47	±	0.05	0.49	±	0.06	0.53	±	0.04
	metal	0.66	±	0.06	0.69	±	0.06	0.69	±	0.04
	roof tiles	0.30	±	0.04	0.37	±	0.05	0.35	±	0.06
fscore	concrete	0.45	±	0.02	0.49	±	0.03	0.51	±	0.01
	metal	0.66	±	0.03	0.68	±	0.03	0.69	±	0.02
	roof tiles	0.32	±	0.02	0.38	±	0.03	0.39	±	0.05
accuracy	overall	0.55	±	0.02	0.59	±	0.02	0.60	±	0.02
kappa	overall	0.21	±	0.01	0.27	±	0.02	0.28	±	0.02

Table G.2: Results of pre-trained networks (no fine-tuning) as feature extractors on the roof material data set