Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelor Thesis

Alexios Sidiropoulos

Application of deep neural networks for bicycle detection and classification

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Alexios Sidiropoulos

Application of deep neural networks for bicycle detection and classification

Bachelor Thesis based on the examination and study regulations
for the Bachelor of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Klaus Jünemann
Second examiner: Prof. Dr. Heike Neumann

Day of delivery: 11th December 2019

Alexios Sidiropoulos

Title of Bachelor Thesis

Application of deep neural networks for bicycle detection and classification

Keywords

Image Classification, Object Detection, Machine Learning, Deep Learning, Convolutional Neural Networks, Residual Networks, TensorFlow, Keras

Abstract

This thesis focuses on the implementation and design of convolutional neural networks for image classification and object detection. It describes the process starting from data collection and manipulation, to creation of simple CNNs with a few layers, as well as a more complex implementation of a Residual Network which uses 50 layers for classifying images into predefined categories. The object detection algorithms use pre-trained models and transfer learning to create models for bicycle localization and detection.

Alexios Sidiropoulos

Titel der Bachelorarbeit

Anwendung von tiefen neuronalen Netzwerken zur Fahrraderkennung und -klassifizierung

Stichworte

Bildklassifizierung, Objekterkennung, Maschinelles Lernen, Deep Learning, Faltungsneuronale Netze, Residual Networks, TensorFlow, Keras

Kurzzusammenfassung

Gegenstand dieser Arbeit ist das Design und die Implementierung von faltungsneuronale Netze zur Klassifizierung von Bildmaterial und dem Erkennen von Objekten. Angefangen mit Ausführungen über das richtige Zusammenstellen und Bearbeiten der Daten, werden alle Schritte erläutert, welche diese Aufgabenstellung mit sich bringt. Dies beinhaltet Erläuterungen über den Aufbau kleinerer CNNs mit nur wenigen Schichten, sowie Ausführungen über den Bau eines aus 50 Schichten bestehenden Residual Network. Die angewandten Algorithmen zur Objekterkennung verwenden bereits trainierte Modelle und das Konzept des Transfer-Learnings, um Modelle zur Lokalisierung und Detektion von Fahrrädern zu erstellen.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

1 INTRODUCTION

In this era of technology, an enormous amount of data in the form of images, sound or text is available on the internet, with its size increasing with each passing day. It is impossible however for humans to process and use all this data. For this reason, the field of machine learning has had an exponential increase in popularity. It allows people to create programs that are processing all this data for them and minimizing the human effort needed. Machine learning models are continuously integrated into industry-level software, allowing for an automated decision making and problem solving.

The thesis described in this document, experiments with convolutional neural networks of different architectures used for image classification and object detection, specifically for images containing bicycles. It starts with describing in theory what a machine learning model is and how it works, the data collection and its curation that needs to be performed and how the architectures of the networks are. Additionally, it explains the training process of the models and reports on their results.

This thesis is part of a bigger project being developed in the University of Applied Sciences Hamburg, which attempts to create a cheap solution for a bicycle counting system. The hardware part of this project will include a Raspberry Pi for computations and storage, attached with a camera. On the software part of the device, a program will detect passing bicycles through a live-feed from the camera. The software will consist of a trained neural network able to accurately detect bicycles and counting their number depending on their direction. The work that has been performed in this thesis is partly based on the work of Kaloyan Dimitrov [1].

2 MACHINE LEARNING

Machine learning is an application of artificial intelligence, which focuses on reducing the human intervention required to make a system learn to identify patterns and make decisions, depending on streams of data that it analyses beforehand.

Using the data provided by the user, also called the “training data”, a machine learning algorithm builds a mathematical model which is capable of making predictions or decisions on its own. The accuracy of the predictions or decisions that the model performs, depends on several factors the most important of which is the training data. The training data contains examples of the objects that need to be identified and used to fit the parameters of the models. [2]

Machine learning models are nowadays used for a wide variety of applications, including computer vision, speech recognition, social network filtering and even for medical diagnoses.

2.1 **Types of machine learning algorithms**

Machine learning algorithms are divided into categories depending on the human intervention needed, the way that data is received and processed and how a program can adjust its own parameters in order to achieve better performance. The main types of machine learning algorithms are supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning.

2.1.1 Supervised learning

A supervised learning is a type of machine learning, where the data used for training the network is already labeled or classified. When the system begins to train, it immediately has an answer on how both the input and the output should look like. The data is prepared by the user and then fed into the system. The system then starts to recognize patterns that exist in the input data and correlate to the desired output. By doing this, the system with react accordingly to new input given to it, based on the training data that it has received. Supervised learning algorithms are useful in use-cases like object recognition, spam detection or pattern recognition. [3]

2.1.2 Unsupervised learning

Unlike supervised learning, unsupervised machine learning models receive data and understand patterns without the help of the user. The data fed to the network is neither classified nor labeled, and the output values are unknown. The algorithm analyses the

information received from the data and groups them depending on similarities or differences that may exist between them. Unsupervised learning algorithms are used for clustering data into categories, applications of which can be performed in the fields of human behavior or visual recognition. [3]

2.1.3 Semi-supervised learning

Semi-supervised learning algorithms use the ideas of both supervised and unsupervised machine learning. Semi-supervised learning algorithms are using both labeled and unlabeled data for the training process. Such algorithms have the advantage that they do not require an expensive and time-consuming process of acquiring data and labeling them and often result in higher accuracy results on the output, because it removes the human biases that may be imposed on the model. Well-known applications of semi-supervised machine learning algorithms include webpage classification and speech analysis, among others. [4]

2.1.4 Reinforcement learning

In reinforcement learning algorithms, the system interacts with its environment and receives a reward or a penalty. When the algorithm performs an action correctly, it is awarded with a reward, otherwise it receives a penalty. The algorithm is not explicitly told how to perform actions by the user, but it rather works by analyzing and working through the problem on its own, trying to achieve its maximum success. Application of reinforcement learning algorithms can be seen in the field of robotics or advertisement. [5]

2.2 **Deep learning**

Deep learning is a subset of machine learning which mimics the way the human brain works for decision making. A deep neural network processes data and creates patterns which sometimes outperforms the capabilities of humans. It utilizes a hierarchical level which is built on simplifications based on the human brain with neuron nodes that connect and communicate with each other.

A deep learning model takes raw data in the form of images, sound or text which have already been labeled by the user and learns to classify them in specific categories.

2.2.1 Neural network architecture

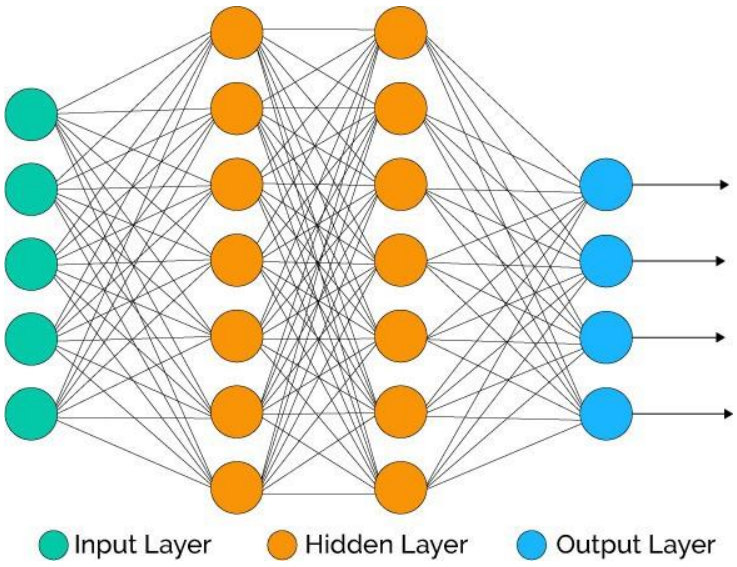

Figure 1: Structure of a neural network [6]

2.2.1.1 Neurons, Synaptic Weights and Biases

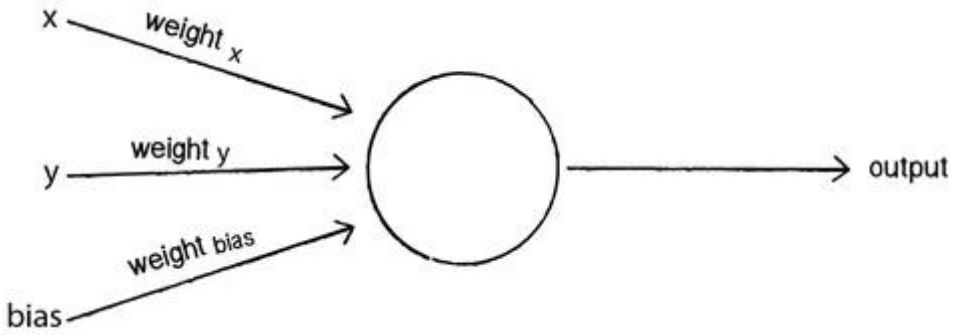

Figure 2: Inputs received by neuron and calculated output

A neuron (sometimes also referred to as unit or node) is a computational unit of a neural network, which receives several inputs and outputs a single value. Each node trains on a distinct set of features depending on the values received from the previous layer.

For an artificial neuron to calculate its output value, each of its inputs is multiplied with a weight and then they are summed together. The summation of the values is then passed through an activation function. [7]

$$f(x, w) = \varphi\left(b + \sum_{i=0}^{p}(x_i * w_i)\right) \qquad \textit{(Eq. 1)}$$

The equation above shows the calculations that take place in each node in order to produce its output. The ***x*** represents the input passed to the node and ***w*** the synaptic weight (usually referred to as weight) that it is multiplied with for a ***p*** number of inputs. The activation function is denoted as ***φ*** and the bias as ***b***.

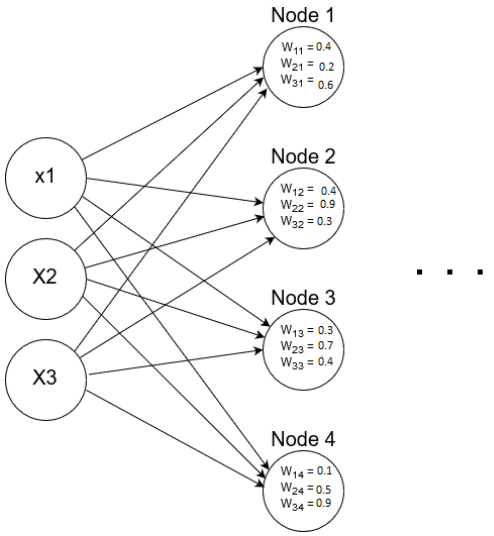

Figure 3: Synaptic weights for each input (values are used as an example)

The number of synaptic weights that exist for each node is proportionate to the number of its input vectors. The weights are the connections that exist between neurons and they highlight the importance of each input connection. The higher the value of the weight for an input, the higher the importance of that input, which forms the "decision" of the node for that particular feature of the model. The weights are selected randomly at the initial layer and they are calibrated after each output node.

The bias is a special neuron called bias neuron which stores a constant number, usually the value of 1, and it also carries a weight. The bias neurons are not connected to any previous layer and they are added to all neurons in all layer, except the output layer. This allows us to shift the activation function to the left or right (see 2.2.1.3).

2.2.1.2 Layers

A set of different nodes is called a "layer" of the network. Depending on the implementation of the model, a deep neural network can contain from two to hundreds of layers. Each layer of the network receives information about different features of the input data from its previous layer and deconstructs the received information into more detailed information and outputs it to the next layer. Every layer between the input layer and the output layer is called a "hidden layer". The input layer is the first contact of the model with the data to be analyzed and feeds information to the rest of the network, while the output layer collects and gives the information that was calculated.

2.2.1.3 Activation functions

In neural networks, activation functions are used to calculate the weighted sum of input and biases and decide if a neuron should be activated or not. If we consider the part of the equation (Eq. 1) without the activation function, it is as follows:

$$Y = b + \sum_{i=0}^{p} (x_i * w_i) \qquad \textit{(Eq. 2)}$$

We can see that the equation (Eq. 2) can contain any number between negative infinity to positive infinity. The neuron on its own does not have any limitations as to the value that it holds and for that reason, the use of activation functions is needed in order to set a threshold where its decided if the neurons containing each value should be activated or not and pass their value as input to the next layer neurons. The activation function also provides non-linearity to the model. If we do not use an activation function the values would simply be a polynomial, which are easy equations to solve for the system, but they are not able to describe complex operations and thus the system is limited to the power that it has and cannot learn complex functional mappings from complex input data such as images, videos or audio. [8]

Neural networks are considered Universal Function Approximators, which means that they should be able to calculate and learn any function and that is why they contain an activation function. The activation functions are residing within neurons for all layers, except for the input layer. The input layer simply passes the information received from the raw data and does not perform any computations, so an activation function is not needed.

Any non-linear function can be used as an activation function, with the most widely used to be the ReLU activation function and the graphs of some functions are shown below. [9]

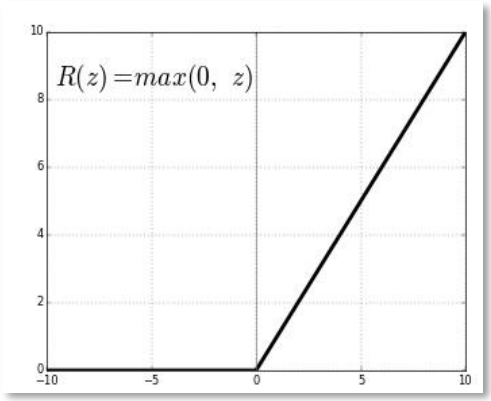

Figure 4: ReLu Activation function

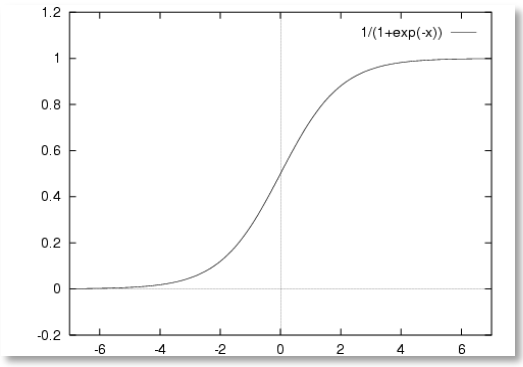

Figure 5: Sigmoid Activation function

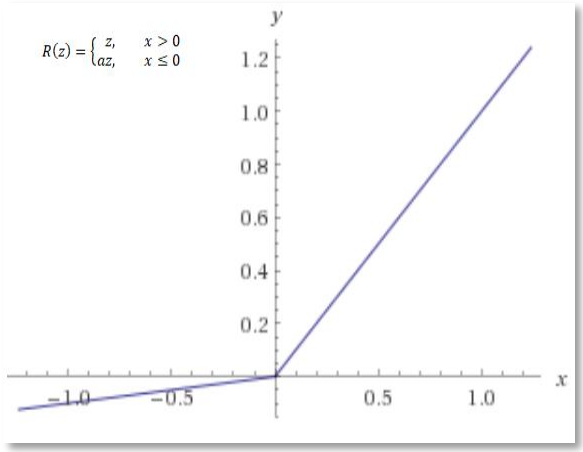

Figure 6: Leaky ReLU Activation function

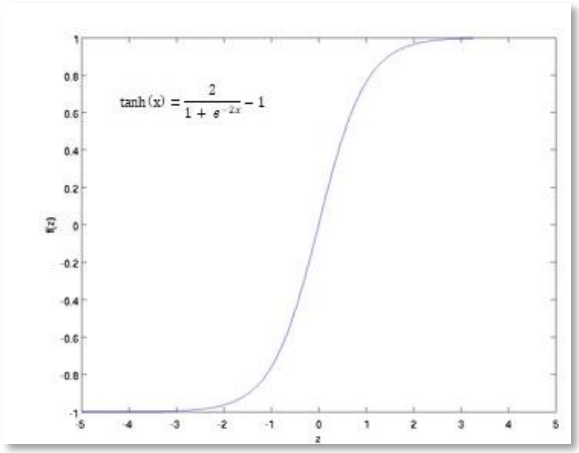

Figure 7: Tanh Activation function

2.2.1.4 Loss functions and cost functions

With the help of loss functions (sometimes referred to as error function), a model has a way of evaluating how accurate its predictions are for a single training sample. The bigger the output value of the loss function, the worse the predictions of the model are and vice versa.

A cost function on the other hand, is a generalization of the cost functions and it holds the value for the average loss of the model for the entire training dataset. By minimizing the value of the cost function, a model is learning better from the information provided by the raw data. [10]

2.2.1.5 Backpropagation and gradient descent

A gradient is a vector-valued derivative, which indicates the slope of the cost function. In a machine learning model, the optimization algorithm used to minimize the value of the cost function is called gradient descent. Gradient descent enables a model to learn the path it should follow to reduce its cost function, which means that it better modifies its parameters and variables. [10]

Backpropagation uses the technique of gradient descent to optimize the overall model. Backpropagation informs the network after each iteration though it, if its prediction was correct or not. By doing this, backpropagation enables the model to fine-tune the weights of its neurons based on the error rate obtained from the previous iteration. When this process is iterated and with proper tuning of the weights, the error rates are reduced and the model is increasing its generalization, making the performance of the network better. [11]

2.3 **Convolutional Neural Network**

A convolutional neural network (CNN) is a deep learning network similar to the neural networks described above. The name convolutional neural network takes its name from the operations that are performed in its hidden layers. Typically, CNNs have multiple hidden layers, which include convolutional layers, pooling layers, fully connected layers and normalization layers. What distinguishes a CNN however from other neural networks is its ability to analyze visual imagery. A CNN receives an image as input and assigns weights and biases to different features of the image, making it capable of effectively recognizing patterns. [12]

2.3.1 How does a CNN see an image?

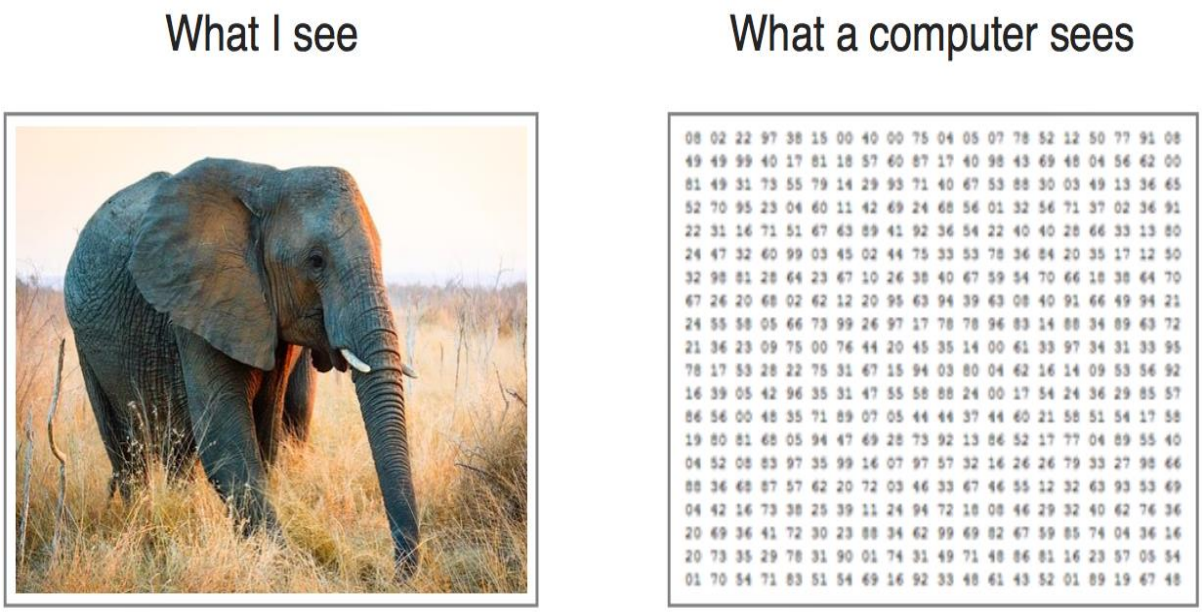

Figure 8: How a CNN sees an image [13]

An image, when captured from a digital camera contains a wide number of pixel values which represent the RGB values that are present in the image. Instead of the actual image, a computer reads the information of the pixels in the form of arrays, containing the values for the image's width, height and depth [13]. When an image is grayscale, it does not contain any color other that white and black, along with different mixtures of the two colors. Then it is said that the depth (or channel) of the image is 1. This allows the computer to know in advance the processing of channels that will take place. If an image contains colors however, the channel of the image is set to 3, containing RGB array values in the scale of 0 to 255. [14]

2.3.2 The Convolution operation

The convolution layer is where the extraction of features from an input image is performed. A convolution is a linear operation that multiplies an array of input data (as explained in 2.3.1) with a 2D array of weights, also referred to as filter. The input matrix is read from the top left of the image and the filter is moving along the input image, performing the multiplications, resulting in a matrix called "Feature Map" [13]. When the filter is applied systematically on the input image and is designed to recognize a specific pattern, then the filter can effectively find the pattern anywhere in the image. [15]

2.3.3 What is Max-pooling?

After the convolutional layer has performed its calculations, its output is passed to the pooling layer. The limitation of the output of the convolution layer is that it is very precise to features of the input image. This means that with distortions to the image, the model would not be as robust as to the patterns it recognizes. In order to optimize the output of the convolution layer, a pooling process is used, which is a sample-based discretization process, aiming to down-sample the input image, reducing its dimensionality and allow the mode to generalize more on patterns [16].

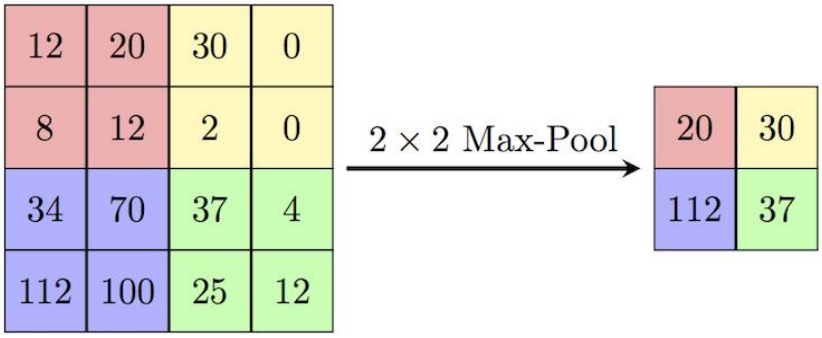

Figure 9: Example of the Max-pooling operation [17]

A Max-pooling layer allows the model to take the maximum value of each cluster of neurons from the previous layer. By performing this process, the spatial size is reduced in regard to the number of parameters and calculations in the network. Max-pooling also helps to avoid overfitting in the model, by providing an abstracted form of the representation.

2.3.4 Dropout

Dropout is a regularization technique in which some neurons of each layer are "disconnected" temporarily and randomly from the neural network. The neurons that are

dropped-out of the network keep changing for each training example, thus giving a different model structure for each iteration. This helps the model prevent overfitting and becoming too dependent on the current dataset that is used, and it helps the model to generalize. [18]

2.3.5 Batch Normalization

For a neural network to learn patterns and features, its weights must be calculated at each layer, which implies that their activation functions will also change continuously in the training process. The distributions of the input data are also changed at each layer, since the output of the activation functions becomes the input of the next layer. This is referred to as internal covariate shift [19]. The convept of batch normalization is to limit the covariance shit by normalizing the activations of each layer [20]. This enables a better performance, improvements in speed and more stability in the model.

2.3.6 Learning Rate

The learning rate of a network is a hyperparameter used for fine-tuning a model. The learning rate informs the model how fast or slow the adjustments of the weights in the network should take place for moving towards a minimum value of the loss function. [21]

2.3.7 Fully connected layers

In a convolutional neural network, we have the convolutional layers which are responsible for extracting features and patterns from an image, or other raw data. This however does not correspond to the output result of the overall network that we can have. In order to receive a prediction dependent on the number of classes that are available, a fully connected layer (which has exactly the same number of neurons as the available classes in the network) is added to the end of the network, which takes the output of the convolutional layers and converts it into a prediction that can be translated into an output class. All the neurons of the last convolutional layer are directly connected to all the neurons of the fully connected layer.

2.3.8 Transfer Learning

Transfer learning is a machine learning method based on the idea that an already trained network can be reused to classify or detect objects in a new network. This gives the new model a head start and speeds up the process of training and developing models, by using the knowledge acquired from one task and applying it on the other. It enables the user to also train deep neural networks with less data, since typically a network requires a large

amount of data for training. Transfer learning is mostly used for computer vision and natural language processing. [22]

A CNN consists of multiple layers, where each layer contains its own filters. These filters contain information regarding the input image received by the network. The filters depending on the stage they are placed in, read different information and learn features about the current image. The first few layers of a CNN usually learn to recognize parts of the image such as colors, edges or certain lines. The following layers then receive the information learned from previous layers, such as colors and lines, and are able to learn to recognize trivial shapes, forming bigger objects as the network goes deeper. When performing transfer learning, the model keeps the information learned in layers regarding general features intact and fine-tunes the deeper layers with more specific information about features regarding the task at hand. [23]

2.4 **Over- and Underfitting**

Overfitting appears when a model trains itself on the training data too well. The model captures the noise and inaccurate data entries of the training data, causing it to have a negative effect on its performance. Overfitting can occur when too much training data is fed into a shallow network, making the model memorize the data points, instead of trying to recognize and predict patterns in new data points.

In contrast to overfitting, an underfit model is unable to model the training data and generalize on new data. Underfitting occurs when the training data is not sufficient for the machine learning algorithm to build an accurate model, or when the training process is interrupted too early. This can cause the model to have poor performance on predicting classes for previously unseen data.

2.5 **Residual Network**

2.5.1 Vanishing Gradient Problem

In artificial neural networks with gradient based learning methods and backpropagation a problem occurs when the gradients of loss with respect to weights are moving from the last layers of the networks to the first layers of the networks. The problem is that when performing backpropagation in a network with many layers, the gradients tend to get much smaller with each iteration, which means that the neurons learn very slowly in the initial layers compared to the later layers. When the initial layers of the model learn slowly, the model is having difficulties recognizing simple patterns which is the building block of the neural network, causing the training process to be very time-consuming and have a very low accuracy of its predictions. [24] This problem is referred to as the vanishing gradient problem.

2.5.2 Structure of a residual network

It is widely observed, that the deeper a convolutional network is, the better its performance. When a user however designs a network and keeps adding layers to it, eventually the problem of the vanishing gradient appears, and the accuracy of the model becomes saturated. This occurs because the network keeps getting bigger, and the gradients that are used for the loss function calculation are reaching towards zero, and therefore the weights of the network do not update, causing the network to not learn [25]. Residual networks solve this problem by using skip connections to connect layers at different depths of the model and allow the gradients to flow through the network without passing through non-linear activation functions.

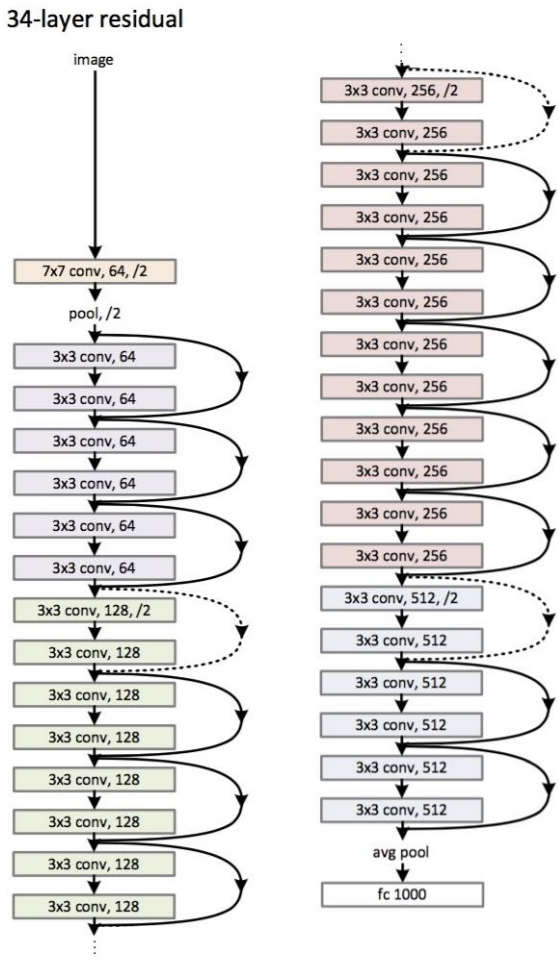

Figure 10: Example of a Residual Network architecture with 34 layers [26]

In Figure 10, the structure of a residual network is displayed. Typically, residual networks consist of residual blocks stacked up, which are implemented with skip connection over 2 or 3 layers that contain convolutional layers, poolings and batch normalizations.

A residual network has the capacity to include hundreds of layers into its architecture, without having the problems of performance degradation or increased difficulty in training.

2.6 **Data augmentation**

Training complex machine learning algorithms requires the user to collect data that can be used from the system to learn to recognize the patterns its intended to. However, the collection of data can be very expensive and time-consuming. The technique of data augmentation gives a limited solution to this problem by increasing the number of data points that can be used for training.

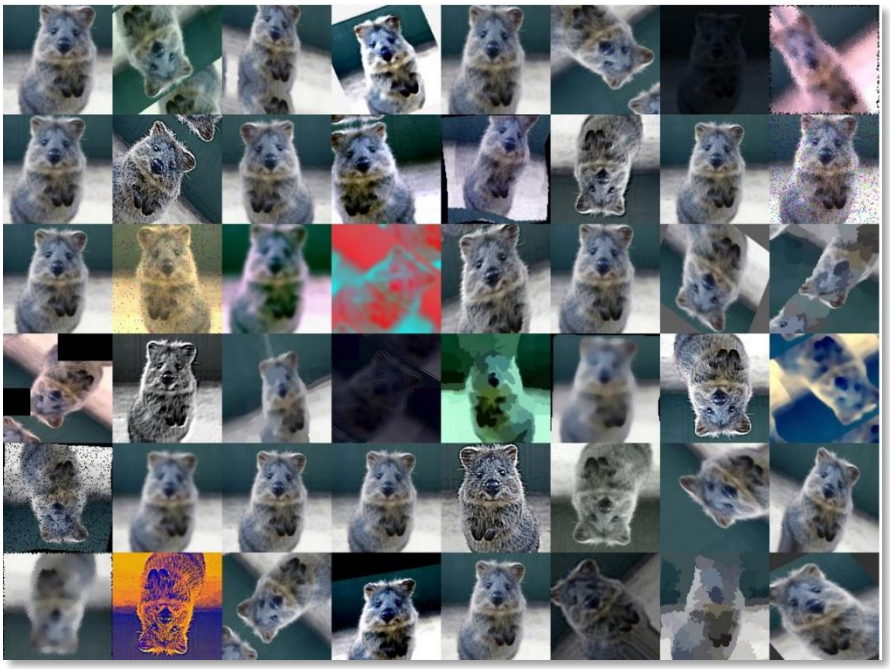

Figure 11: Image augmentation examples [27]

In terms of images, data augmentation takes the original image and performs modifications on it, thus significantly increasing the number of samples and enabling diversity of images that can be used for training. Data augmentation on images can have many forms, such as rotation of image, modification on the lightning conditions, different cropping of the images, translation on x- or y-axis, adding noise in the images, etc. [27]

3 REQUIREMENTS AND THESIS WORK

The requirements of this thesis work can be split into two sections: implementation of machine learning models and administration of the workstation (located in the University of Applied Sciences Hamburg) where the models should be trained.

Machine learning:

1. Collection of images with and without bicycles
2. Application of data augmentation of the overall dataset to expand its size
3. Curation and labeling of images to be ready for training
4. Implementation of convolutional neural networks for classification with different depths and evaluation
5. Reporting of classification networks using TensorBoard
6. Implementation of a residual network, testing and comparison to simple classification networks
7. Implementation and evaluation of object detection models for bicycle detection

System Administration:

1. Configuration of Secure Shell connection for remote access
2. Setup of remote power-on of workstation

4 COMPUTATIONAL ENVIRONMENT

This chapter introduces in detail the computing machine specifications, hardware setup as well as used software tools and applications that were utilized for the implementation and evaluation of deep neural networks for image classification and bicycle detection.

4.1 Main Libraries

For the implementation of the object detection and image classification, many libraries and frameworks have been used. However, the two main libraries are explained in the subsections below, which are TensorFlow [4.1.1] and Keras [4.1.2].

4.1.1 TensorFlow

TensorFlow is an open source artificial intelligence suite of software created by the Google Brain team [28] for numerical computations and large-scale machine learning applications. With the use of TensorFlow, developers are able to create deep learning models which express arbitrary computations as graphs of dataflows. TensorFlow is an ecosystem which delivers the tools needed for a full-scale deployment [29]. The three main components of TensorFlow are:

TensorFlow API

The TensorFlow API contains all the toolkits that are used to define models and train them. The core of the API has been written in the programming language C++, in combination with Nvidia's CUDA, which enables very high-performant and efficient code available both for CPU as well as GPU for faster computations. Most operations that are performed with C++ are however done on the back-end. The programming part of the code, which is developed by the user, is available in several programming languages, with the most used one being Python. This allows the user to write code in a user-friendly and easy language such as Python, while simultaneously taking advantage of the speed of C++. [30]

TensorFlow Serving

TensorFlow Serving is a high-performance and flexible serving system which has been created for an easier deployment of the machine learning models created. It allows the trained models to be available for prediction requests directly in the production environments. [31] [32]

TensorBoard

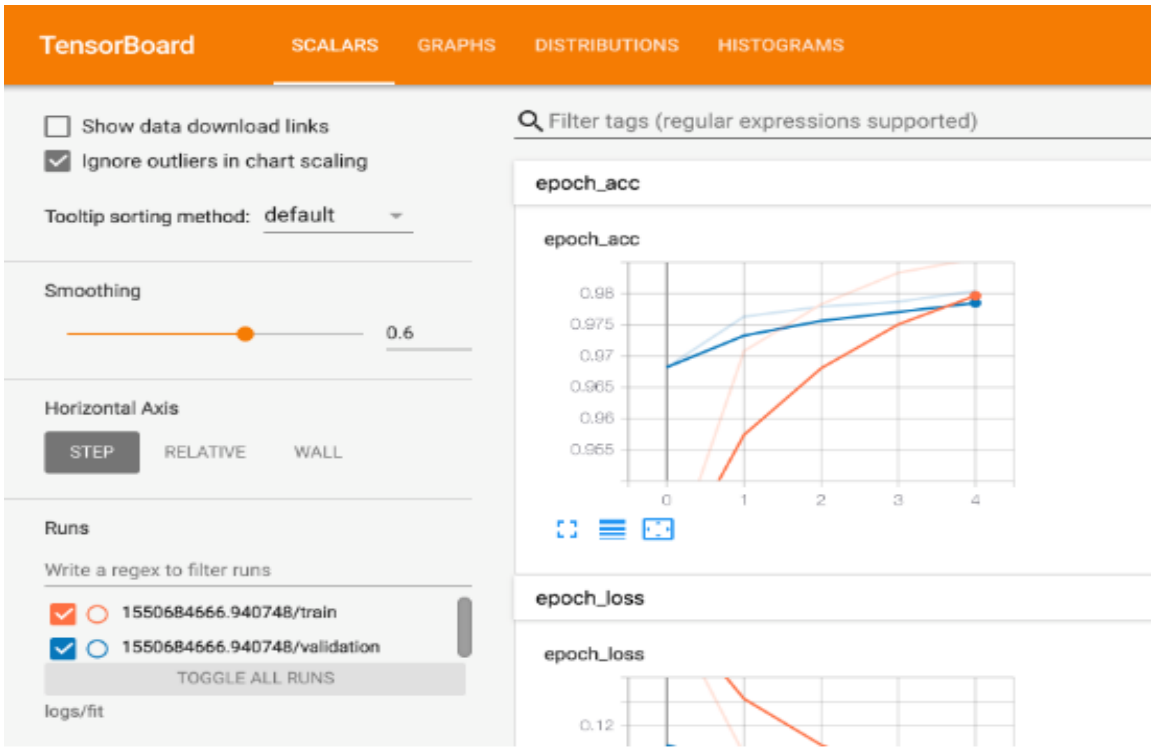

Figure 12: Example of a TensorBoard visualization of a model

TensorBoard is a tool used for the visualization of the attributes of the machine learning models. It allows the user to track the information of the model and to visualize them in the form of graphs in order to understand, debug and optimize the model. [33] [34]

The TensorBoard interface allows the user to display the graphs while simultaneously training the model. The information that it displays are separated into tabs, some of which can be seen in Figure 12 and are as follows:

Scalars

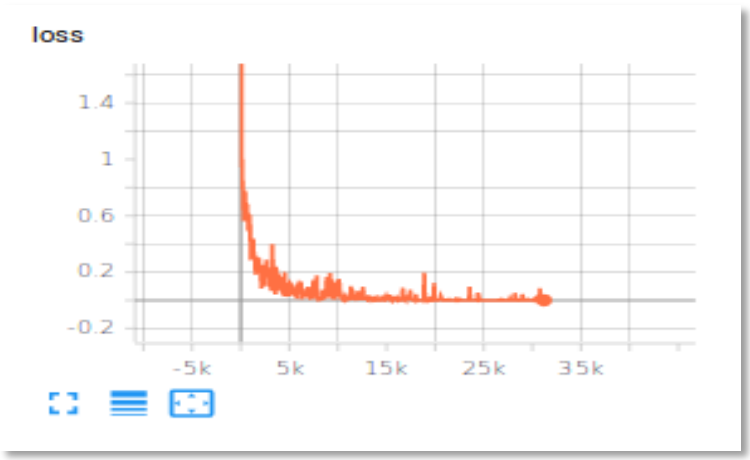

Figure 13: Display of information about the training loss of a model

The Scalars tab shows graphs related to the training information received from the model. It displays graphs about the accuracy and loss of the model, training speed, learning rate, etc. on the y-axis, and the global step (i.e. iteration of the model) on the x-axis.

Graphs

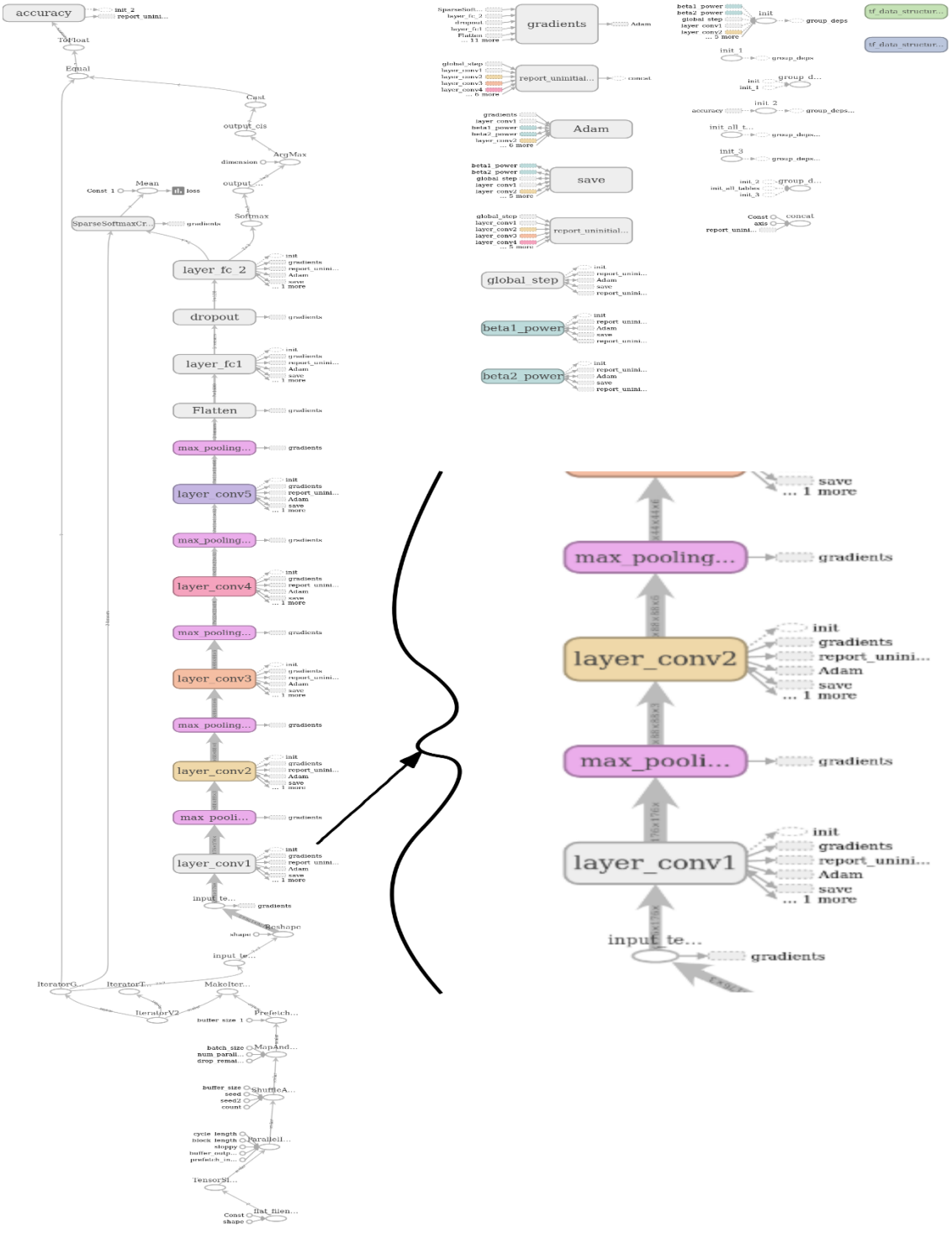

Figure 14: Structure of a 5-layer classification model

The Graphs dashboard maps the relationships among the layers and their function in the model on a conceptual level. By reviewing the structure of the model, the developer can inspect any unwanted connections inside the model and change them for better performance.

Distributions and Histograms

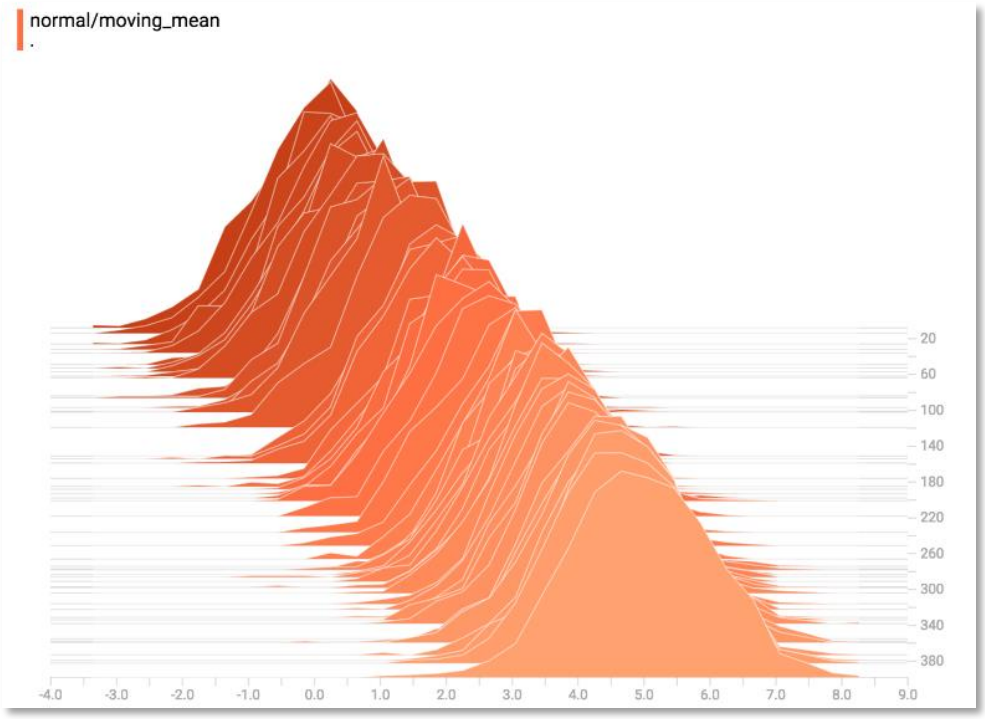

Figure 15: Example of a histogram of a model from TensorBoard

The Histogram and Distribution dashboards display the statistical distribution of a Tensor and how it has changed over time. The difference between the two is that Histograms display the same information as a 3D representation of the data across iterations. [35]

4.1.2 Keras

Keras is an open-source neural network library that runs on top of Tensorflow, among others, and is written in the programming language Python. Keras has been designed to be user-friendly, modular and fast. Keras is a high-level API wrapper, which uses low-level APIs to handle computations. This allows for a faster definition and training of models, with a minimalistic style of code required from the user and can run on both CPUs and GPUs. To build a model with Keras, one must decide between one of two ways of implementation:

Sequential Model API

Figure 16: Structure of the Sequential model in the form of a path graph [36]

The Sequential model is the simplest way to create neural networks. It works by having layers one after another in a linear manner. The Sequential model has the restrictions that it does not allow the user to have multiple inputs or outputs, which can sometimes be needed depending on the problem at hand. [37]

Functional Model API

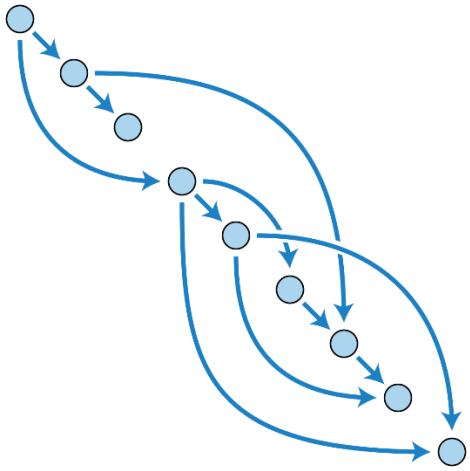

Figure 17: Structure of a Functional model in the form of a directed acyclic graph [38]

The Functional API allows for a more complex definition of models, which provides more flexibility as for the implementation of the algorithms at the cost of simplicity and readability. In contrast to the Sequential model, the functional model can handle multiple inputs and outputs, as well as models that share layers. In addition, the Functional model is based on the idea that a deep learning model is often a directed acyclic graph, while the Sequential model follows the idea of a path graph. [39]

4.2 **Hardware**

The training and implementation of deep neural networks requires a powerful computer due to the complex and long calculations that need to be performed. If the computer is not powerful enough, these calculations can take hours or even days to be done, thus being very time-consuming.

All the implementation that has been performed for this thesis, has been done on a CADnetwork Deep Learning desktop computer specifically designed for deep learning applications. The desktop computer runs on an Ubuntu OS and has 2 very powerful Nvidia GPUs (GeForce RTX 2080 Ti), 64 GB RAM memory and 28 processors (Intel Core (TM) i9-7940X CPU), among other components. This enables for very fast computations for deep learning on the computer, especially when TensorFlow has been configured to use the GPUs for calculations, instead of the CPUs. GPUs are designed to process graphics extremely fast in modern technology and they are capable to use their resources to perform operations on multiple vector data in parallel. The architecture of CPUs on the other hand allows them to process one operation at a time, which reduces their effectiveness for deep learning in comparison to GPUs.

4.3 **Docker**

The development of software often results in execution problems when transferred from one computer environment to another, due to different configurations, operating systems, etc. Docker is a tool developed for an easier creation, deployment and execution of applications with the use of containers. A container allows the developer to package up an application and all its parts (libraries, configuration files, dependencies) and have it ready for deployment. Docker containers have the advantages that are lightweight, standalone and executable packages that make the containerized software run the same in every environment, regardless of the infrastructure or the operating system used. [40]

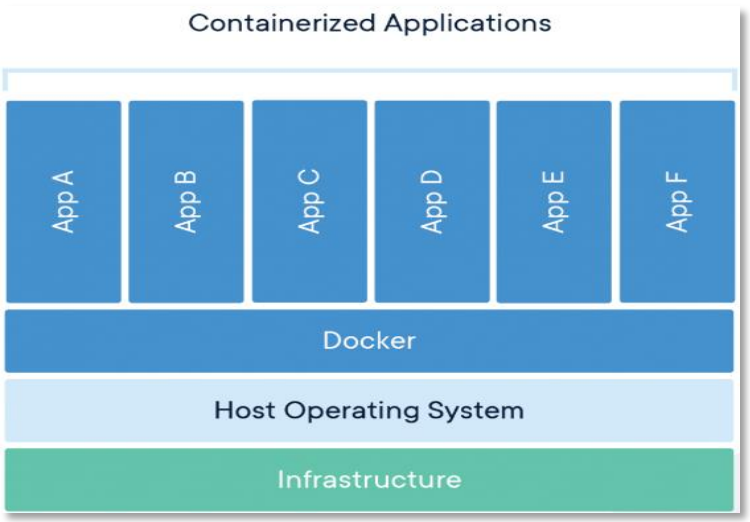

Figure 18: Structure of Docker containers [40]

A container can be thought of as a virtual machine (VM). Virtual machines are created and run on hypervisor environments, which are computer software, firmware or hardware that manage the execution of the VM. Each VM requires its own guest operating system, along with its related binaries, libraries and application files, which consumes a large amount of processing power and resources from the host machine. Each Docker container however, shares the operating system with its host machine, which significantly reduces its size and boosts performance, while maintaining their own executables, code, libraries, tools and configuration files. [41]

As part of the implementation phase of this thesis, a Docker environment was used in order to have a containerized environment, which includes all the necessary files and libraries. The container included an installation of the TensorFlow library, which was required to train models and test their accuracies, alongside with required dependencies from TensorFlow such as Python, NumPy and more. This allowed for a centralized source of information, where all the computations and experimentations were taking place.

4.4 **Remote access with Secure Shell**

The computer used for this thesis is placed in a secure location at the University of Applied Sciences, due to its high cost and the operations that it performs. Physical access to this computer is not always possible, so a method needed to be found to be able to access the computer at anytime from anywhere. For this reason, the employment of the Secure Shell (SSH) was used.

SSH is a cryptographic network protocol which provides a secure channel over an unsecured network in a client-server architecture. The SSH provides a secure way of executing commands, making changes and configuring services remotely using a text-based interface by spawning a remote shell.

The SSH authentication that has been implemented is using an RSA cryptographic algorithm and is performed with an SSH key pair consisting of a public and a private key, which are secure keys used to authenticate a client to an SSH server. [42]

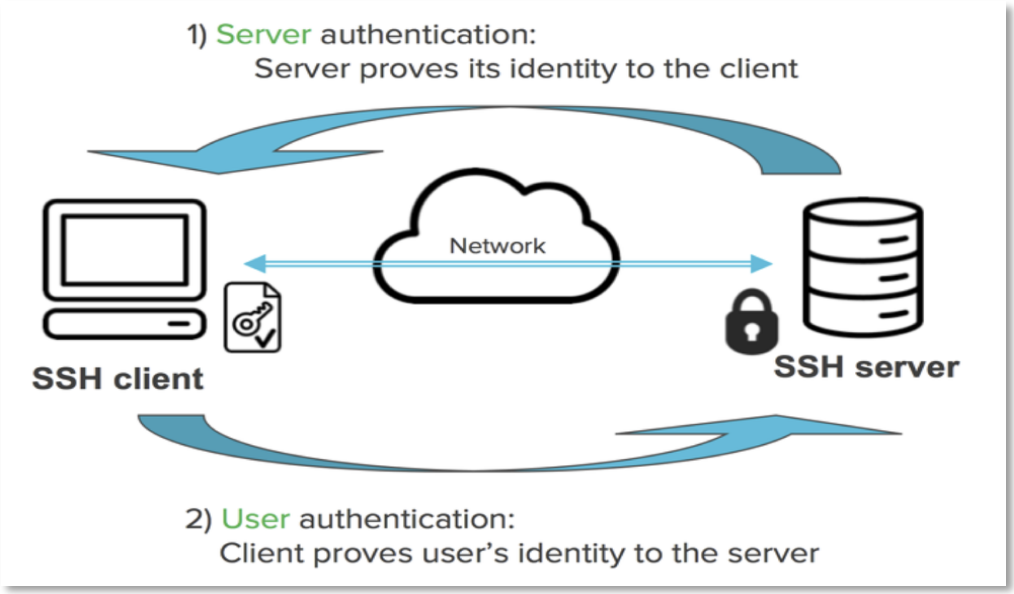

Figure 19: SSH operations [42]

The public key needs to be uploaded to the SSH server and can be shared without any risks. In contrast, the private key is used and retained by the client and it should not be shared. The private key is the only key that can decrypt any messages encrypted by the public key and any compromise to it can allow a third-party access to the server. The authentication of the SSH connection will be successful only if the client can prove to the server that it is in possession of the private key. [42]

4.5 **Remote power-on of workstation**

For the SSH connection to be successful as described in chapter [4.4], the computer would need to be constantly turned on. This would cause the computer to consume a large amount of power, even while it is not being used. To avoid such a case, two methods for powering on the workstation have been researched and implemented, which are explained below.

4.5.1 Wake-on-Lan

WoL (also known as "Power on by PCI/PCIe") is an Ethernet networking standard that allows a user to send a network message to power up a computer from a low power mode remotely [43]. A computer even when shut down, while still connected to a power source, uses a very low amount of power. When the BIOS or the network adapter settings (depending on the type and configuration of the computer) for WoL are turned on, the computer provides the network card with a sufficient amount of power to remain in standby mode and actively listens to the predefined ports for a "magic packet". A packet can be considered "magic" when it contains 6 bytes of the largest possible byte value (*FF FF FF FF FF FF*), also known as the

synchronization stream, followed by 16 repetitions of the target computer's MAC address. The last 6 bytes of the data are reserved for an optional password set on the configuration phase which can be used to increase the security of the system. [44]

When all the configurations needed have been performed, the user is able to use any website or application developed for sending a "magic packet" to power on the computer.

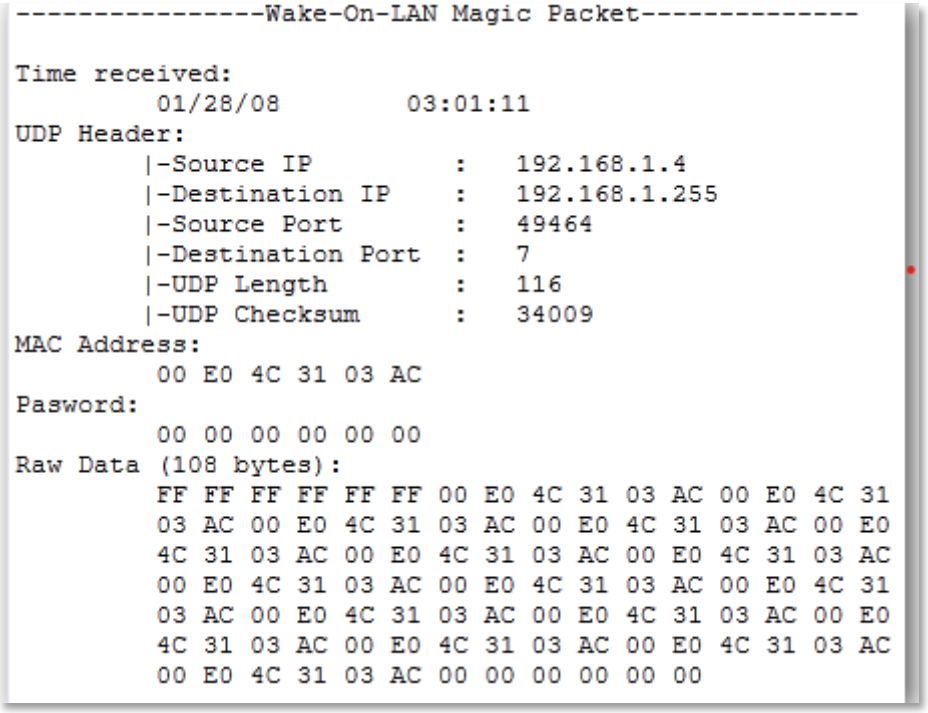

Figure 20: Magic packet example [45]

The WoL feature does not pose a security vulnerability to the system. It allows the user to only turn on the computer, but it does not allow any other activity to take place through it.

4.5.2 Smart-Plug

Another way to power on a computer while it is turned off is by modifying the BIOS settings. Most computer contain a setting under the "ACPI Configuration" tab called "Restore on AC/Power Loss" (sometimes it may be named "AC Power Recovery" or "After Power Loss"). By default, this setting is turned off, however when enabled and after loss of power it allows the computer to automatically turn on when the power is restored.

This configuration can be combined with the technology of the Smart-Plugs. A Smart-Plug is a plug that connects to a traditional electrical outlet. Contrary to the traditional electrical outlet, a Smart-Plug can be configured to connect to a Wi-Fi connection and allow access to it from a smartphone or a computer. It enables the user controlling it to allow or

block alternating current passing through it and accordingly turn on or off the device connected to it. Depending on each model and type of Smart-Plug, it can have a wide range of analytics and function in the respective mobile application such as measurements of power consumption, set schedules for powering on or off the device, voice control of the device and more. The model of the Smart-Plug used for this implementation is the D-Link DSP-W 215/E Home.

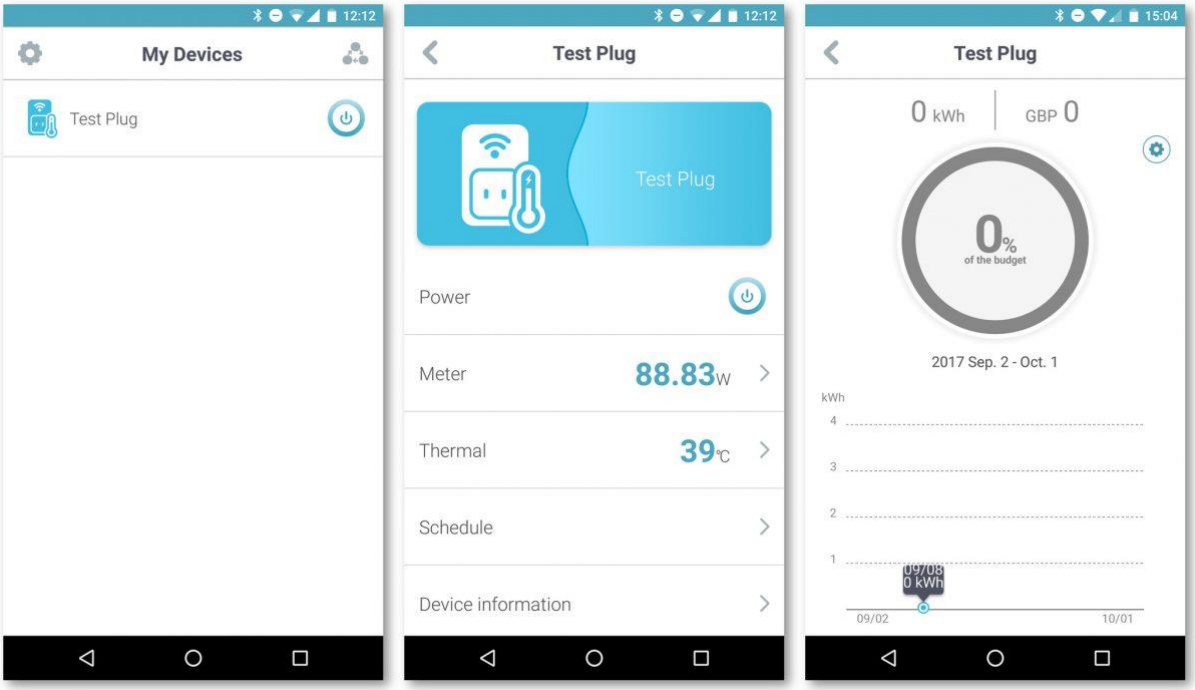

Figure 21: User Interface of smart-plug application [46]

By combining the “Restore on AC/Power Loss” setting and the Smart-Plug, we can turn on the computer simply by first blocking the AC passing through the Smart-Plug and then allowing it which will automatically trigger the aforementioned BIOS setting to turn on the computer. This allows us to turn off the computer when it is not in use, and remotely power it on when access is required, thus eliminating the amount of power that the computer would need to be constantly in standby mode.

5 IMPLEMENTATION AND RESULTS

5.1 **Data Curation**

Before training the network, there is a process of gathering and manipulating images, which would be used as a blueprint for the model to recognize bicycles. These images contained bicycles moving either on the left or the right direction, always with a rider on the bicycle. Images were provided by Prof. Dr. Klaus Jünemann, some of which were captured from students during projects of similar manner, and the dataset was further developed during this thesis. Previous students that contributed to the creation of the dataset include: Jeonghyun Son, Sami Ullah, Raihan Maksud, Akibur Rahman and Kaloyan Dimitrov.

5.1.1 Data gathering and separation

For capturing images of bicycles while in motion, videos were recorded from different perspectives and different backgrounds. The videos were then being split into frames and saved as images. The images that were captured in this thesis work make up for 1001 images from the complete dataset.

When the network is being trained, the search window always includes some part of the background for the detection of the bicycles. For this reason, the background of the images had to vary so that the model does not index parts of the background as features of the desired outcome. Examples of images with different backgrounds can be seen in Figure 22.

Figure 22: Bicycles with different backgrounds

The images that were gathered, also needed to be split into three categories. The classification that the network was required to perform should be able to recognize and classify a picture that was fed into the network into one of the following categories:

- “bike_left” - bicycle moving from right to left

Figure 23: Bicycle of class "bike_left"

- “bike_right“ - bicycle moving from left to right

Figure 24: Bicycle of class "bike_right"

- “no_bike“ - picture without any bicycle in it

Figure 25: Bicycle of class "no_bike"

For the classification network to be able to assign labels to each training image that was used, the training dataset was split into three folders, each containing the respective images.

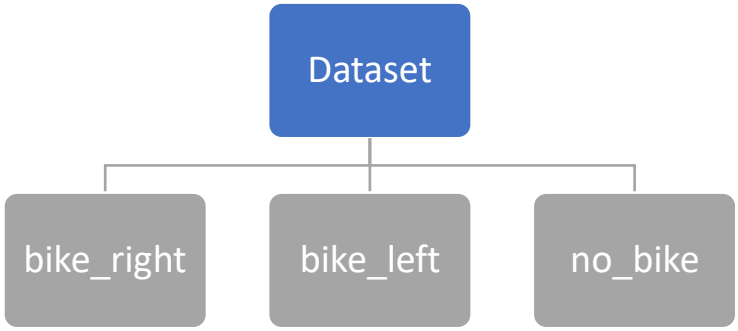

Figure 26: Separation of images depending on their content

5.1.2 Data Clean-Up

The images that were extracted from the videos, contained snapshots that would not have been useful to the model and had to be removed from the overall dataset. These images could potentially diverge the model from its intended purpose and cause it to malfunction by feeding it false data. Such images can be seen in Figure 27.

Figure 27: Ineffectual images

For a better structure and for an easier manipulation of the dataset, different Python scripts were created to perform actions on the images. The following script is a very simple tool which helps in the creation of a unified image type in the dataset. It simply selects all the images that have a ".png" extension and converts them to an image file of the ".jpg" type.

```
1.  from PIL import Image
2.  import glob, os
3.
4.  directory = "bike_right/"
5.
6.  for infile in glob.glob('*.png'):
7.      file, ext = os.path.splitext(infile)
8.      im = Image.open(infile)
9.      rgb_im = im.convert('RGB')
10.     rgb_im.save(directory + file + ".jpg")
11.
```

Several scripts were created, each one with its own functionality, which include:

- The renaming of the images
- Renaming of the tags in the respective XML files (see (5.1.3))
- Resizing the images

5.1.3 Labeling

The task of object detection requires some additional input from the user. For an object detection model to use the images for training, the user first must label them with the category that each image belongs to. This has been done with using the LabelImg tool which can be seen in Figure 28.

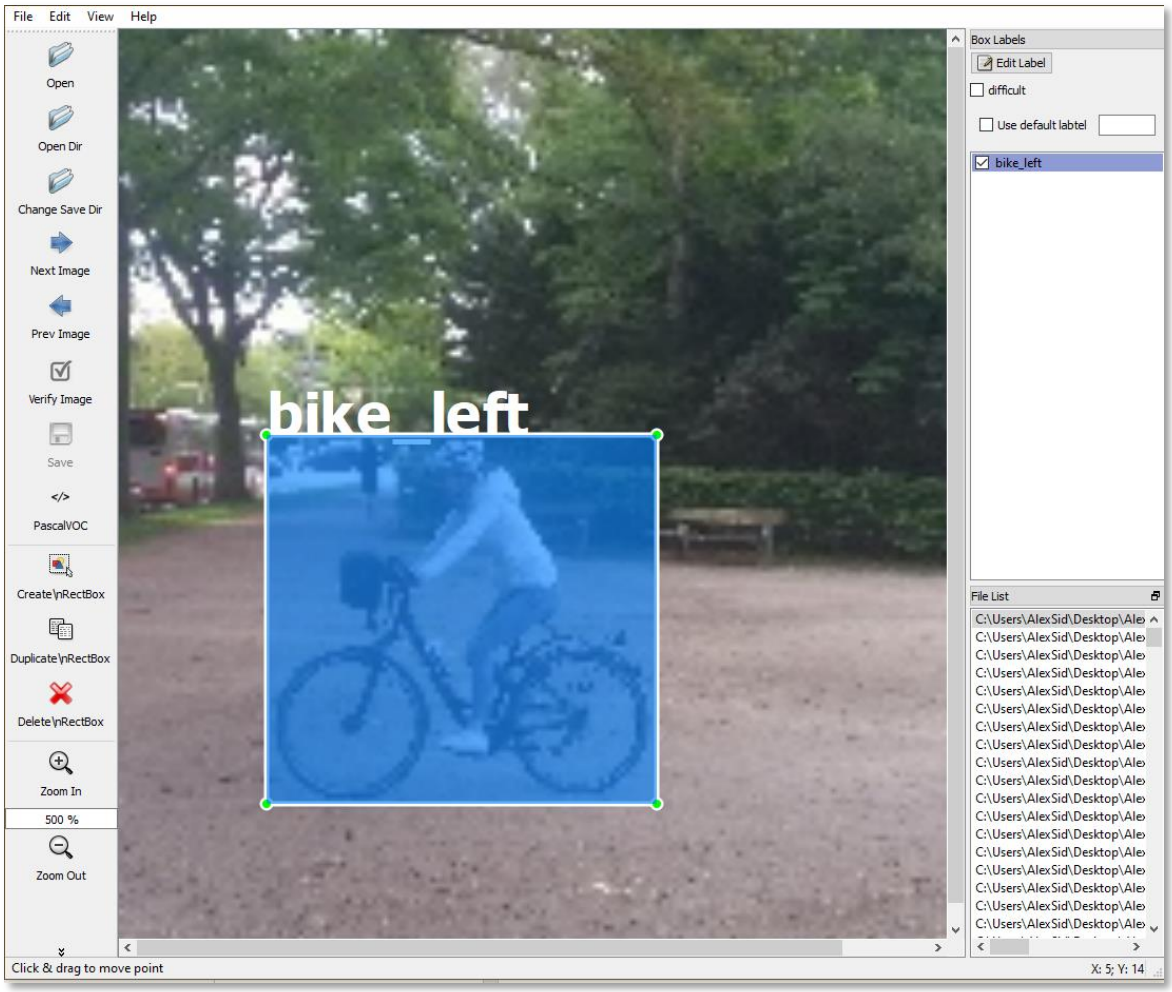

Figure 28: Labeling of an image in the LabelImg tool [47]

LabelImg makes it easy for the user to go through the folder containing the images that need to be labeled and manually specify the position of the particular object that will later be recognized (in this thesis, the bicycles). When the user positions the bounding box (blue box seen in Figure 28) and clicks on the "Save" button, LabelImg automatically generates an XML file containing information about the image.

```
1.  <annotation>
2.      <folder>bike_left</folder>
3.      <filename>left (1).jpg</filename>
4.      <path>C:\Users\AlexSid\Desktop\Alex\Studies\Bachelor
            Thesis\bike_left\left (1).jpg</path>
5.      <source>
6.          <database>Unknown</database>
7.      </source>
8.      <size>
9.          <width>176</width>
10.         <height>176</height>
```

```
11.                     <depth>3</depth>
12.             </size>
13.             <segmented>0</segmented>
14.             <object>
15.                     <name>bike_left</name>
16.                     <pose>Unspecified</pose>
17.                     <truncated>0</truncated>
18.                     <difficult>0</difficult>
19.                     <bndbox>
20.                             <xmin>28</xmin>
21.                             <ymin>77</ymin>
22.                             <xmax>102</xmax>
23.                             <ymax>147</ymax>
24.                     </bndbox>
25.             </object>
26.     </annotation>
```

The XML files create a hierarchical structure with the use of tag elements, which makes it readable and easy to manipulate. The most important tags are explained below:

Tag	Description
<folder>	The folder name where the image and the XML file is stored
<filename>	States the name of the image that the XML file should be linked to
<size>	Describes the size of the image in width, height and depth
<name>	Annotates the label given to the object while labeling
<bndbox>	Contains the start and end point of the bounding box. It is described by using the minimum and maximum values for the x and y axis

Table 1: XML tags and their information

5.1.4 Dataset expansion

For our model to be able to predict the class of the image fed to it or the position of a bicycle in an image more accurately, the number of images in the dataset had to be increased. For this reason, the concept of data augmentation was employed. The data augmentation that was performed in this thesis, was to simply convert the images in the class folder “bike_left” in order to fit the images in the class folder “bike_right” and vice versa. The operation includes the conversion of the image by flipping it horizontally across its middle vertical axis and then changing accordingly the information contained in the respective XML file. This is performed by a Python Script which can be seen in Appendix B.

The code in Appendix B, iterates through the folder (in this case the “images” folder) and first selects the files with the “.jpg” extension. It continues by flipping the image horizontally

and then modifying its name to a predefined image name convention set by the user. The modification of the name has been created in order to have consistency in the dataset, with a unified name variable for each class followed by an ascending index. The script then selects the XML file related to the currently processed image and performs modifications on it. These modifications are:

- Renaming the XML file
- Changing the <filename> tag to include the name of the flipped image
- Alters the <name> tag which states the label of the image
- Refactors the <xmin> and <xmax> tags of the file to change the position of the bounding box

Since the images are only flipped horizontally, the y-axis values of the bounding box remain the same as the original image. Also, some additional tags can be omitted, as they have no effect in the training process. The final result can be seen and compared in Figure 29.

Figure 29: Comparison of original (left) and flipped (right) images

The same action is performed also for the classification task, excluding the manipulation of the XML files, since the XML files are not used for classification. By performing this data augmentation on the dataset, the number of images that can be used for training a network are automatically doubled. The complete dataset at the time of writing this thesis has the following number of images:

Class	Number of images
bike_left	2746
bike_right	2746
no_bike	1548
Overall Dataset	7040

Table 2: Number of images in the dataset

5.2 **Classification**

The convolutional neural networks for classification have been done with the help of TensorFlow and Keras and the work described in (5.2.1) is based on the work of Kaloyan Dimitrov [1]. Multiple CNNs have been designed, each with a different number of layers and filters, as to be able to see the improvements or deteriorations when the network is expanded. For the classifications the dataset described in (5.1) has been used with an overall number of images being 7040. From the dataset, some random images were extracted from each class in order to use them as test data and these images were not used in the training process. The following networks also restrict a number of images from the remaining dataset to use for validating the accuracy of the predictions from the network. The proportion of the dataset used for training the network is 70%, while the 30% left is used for validation.

5.2.1 Simple convolutional neural networks

The process of training the CNN starts with converting the raw images into a format that can be used by TensorFlow. The Python file "*create_dataset.py*" (see Appendix D) is performing this task by taking images from the subfolders as seen in Figure 26, retrieving its information and assigning a label to it depending on its parent folder. All this information is then converted into a TensorFlow specific file system called "TFRecords", which is a binary representation of the images and their labels. Two different TFRecords files are created from the Python script, one containing information about the images to be used for training and another one for validating. The TFRecords files are then ready to be used from the TensorFlow problem for training and evaluating the network.

The graphs displayed in the figures below, display the accuracies and losses of each model. The lines on the graphs displayed in orange color show information regarding the training losses, while the lines in blue color are displaying the information of the evaluations regarding losses and accuracies. As it can be seen from the figures below, the training accuracies are not displayed on the graphs. This is because we are interested in seeing how well the network actually performs on data not available for training (from the validation dataset). Also, for each model below there is a description of its training time, as well as the number of filters used for each layer. Each network was trained for a global step between 30000 and 35000 and contains a dropout regularization of 0.5. While training the models, an average GPU utilization value of 85% was observed. The TensorFlow models were not enabled to use 2 GPUs that were available in the desktop computer.

The following code shows how a layer is defined in TensorFlow. The number of filters of each layer can be changed by changing the number of the "filters" parameter in line 2. A convolutional neural network can be expanded by stacking the code below one after the other.

```
net = tf.layers.conv2d(inputs=net, name='layer_conv2',
                       filters=64, kernel_size=3,
                       padding='same', activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2,
                       strides=2)
```

After the training of the network was finished, the networks were tested on 74 images that were not included in either the training or the validation dataset. The script that tests these images, first loads the model that has been trained, and then passes the images through it and makes a prediction about each image. The predictions are split into 3 parts, storing values about how accurate each class prediction is. After the predictions for all the images is finished, the script sums up the percentage of the accurate predictions for each class and divides it by the number of classes (there are 3 classes for this implementation) to form an overall accuracy percentage of the network.

The images that were used to test the network are of increased difficulty to test how the network performs when the images are not exactly like the images that were used for training or validation. These images have either a bicycle in a different angle towards the camera, distortion on the images, contain only small parts of a bicycle in them or have multiple bicycles included in one capture. Some examples of the test images can be seen in Figure 30. The same images were used to test all networks and the results of the prediction can also be seen for each developed network.

Figure 30: Examples of images used for testing

3-Layer network

Figure 31 and Figure 32 show the information retrieved in the training process of a model containing 3 layers with a sequence of 32/64/64 filters. The training process reached the global step of 32000 in 15 minutes with a maximum accuracy of 88%.

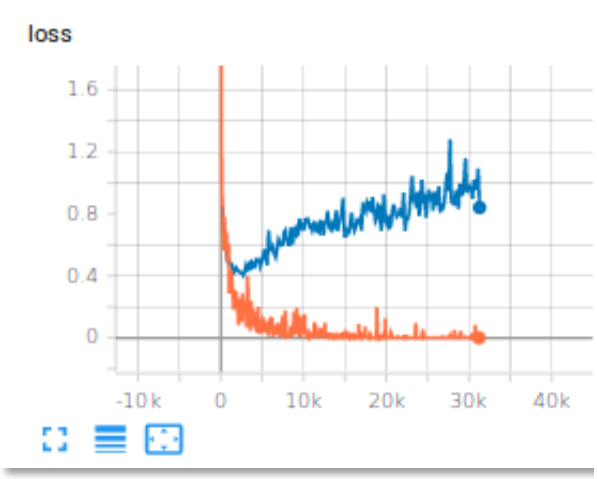

Figure 31: Training and evaluation loss of a 3-layer network

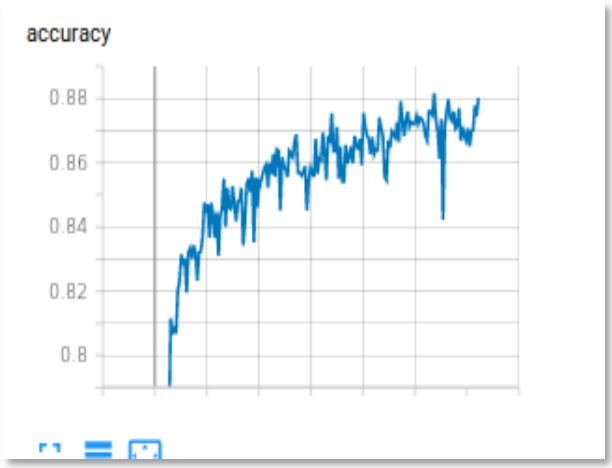

Figure 32: Evaluation accuracy of a 3-layer network

Testing this network on new images resulted to an overall accuracy of 56% across all classes and images as it can be seen in Figure 33.

```
Number of 'left' test images:  25
 Accuracy Left =  0.68
Number of 'right' test images:  25
 Accuracy Right =  0.48
Number of 'neg' test images:  24
 Accuracy Neg =  0.5416666666666666
Overall Accuracy=  0.5672222222222222
```

Figure 33: Testing accuracy of 3-layer network

4-Layer Network

The 4-layer model from Figure 34 and Figure 35 contains a filter sequence of 64/128/128/64. The time needed to reach the step 32000 was 21 minutes and the maximum accuracy reached in the validation was over 92.5%.

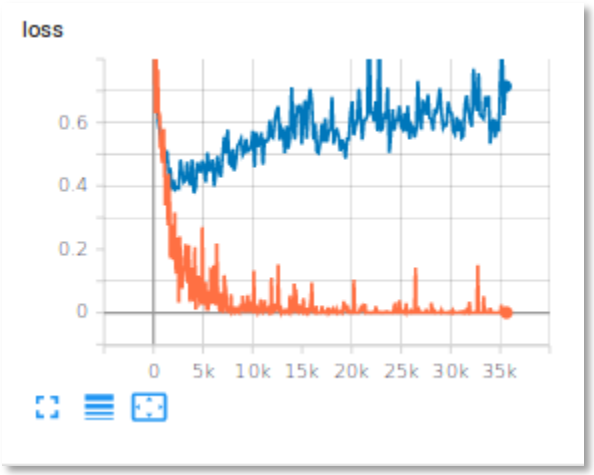

Figure 34: Training and evaluation loss of a 4-layer network

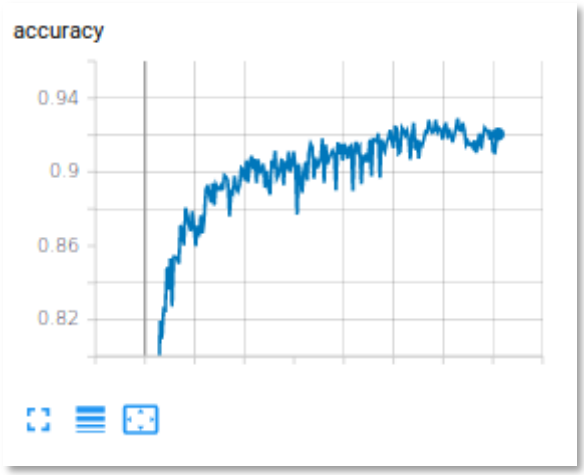

Figure 35: Evaluation accuracy of a 4-layer network

If Figure 33 and Figure 36 are compared, one can already see that the accuracy increased to 71% by adding 1 additional layer to the network.

```
Number of 'left' test images:  25
 Accuracy Left =  0.68
Number of 'right' test images:  25
 Accuracy Right =  0.76
Number of 'neg' test images:  24
 Accuracy Neg =  0.7083333333333334
Overall Accuracy=  0.7161111111111111
```

Figure 36: Testing accuracy of a 4-layer CNN

5-Layer Network

The 5-layer network reached the global step of 33000 in 17 minutes with a filter sequence of 32/64/64/32/64. The maximum validation accuracy reached was just below 92%.

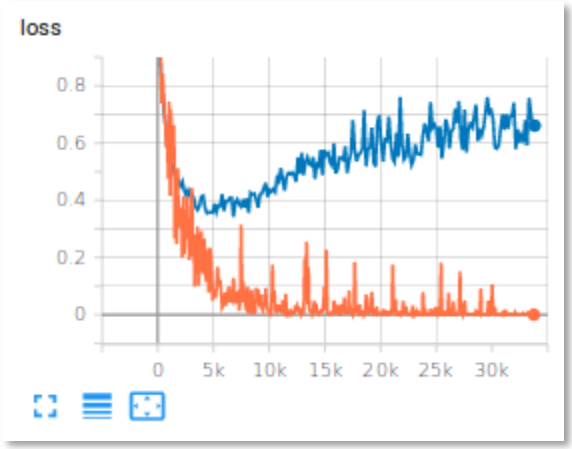

Figure 37: Training and evaluation loss of a 5-layer network

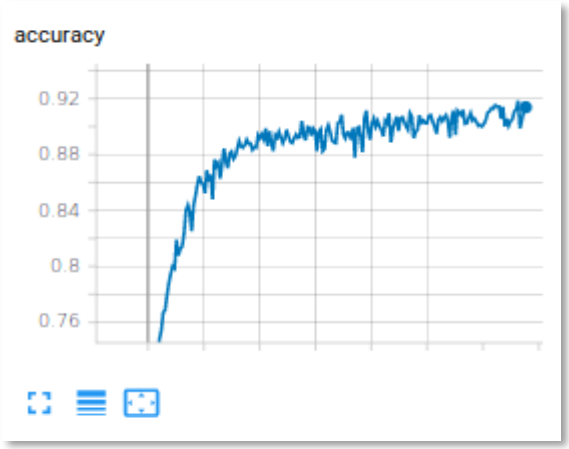

Figure 38: Evaluation accuracy of a 5-layer network

The test images applied on this network, result in the accuracy of the model being 68%.

```
Number of 'left' test images:  25
 Accuracy Left =  0.76
Number of 'right' test images:  25
 Accuracy Right =  0.76
Number of 'neg' test images:  24
 Accuracy Neg =  0.5416666666666666
Overall Accuracy=  0.6872222222222222
```

Figure 39: Testing accuracy of a 5-layer CNN

7-Layer Network

Lastly, the 7-layer network was designed with the filter sequence 32/64/64/128/128/64/64 and it was able to reach a validation accuracy of over 92%. The training time for reaching the global step 33000 was 18 minutes.

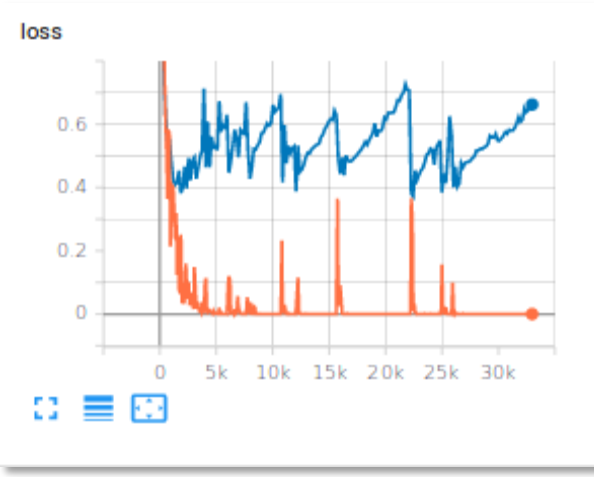

Figure 40: Training and evaluation loss of a 7-layer network

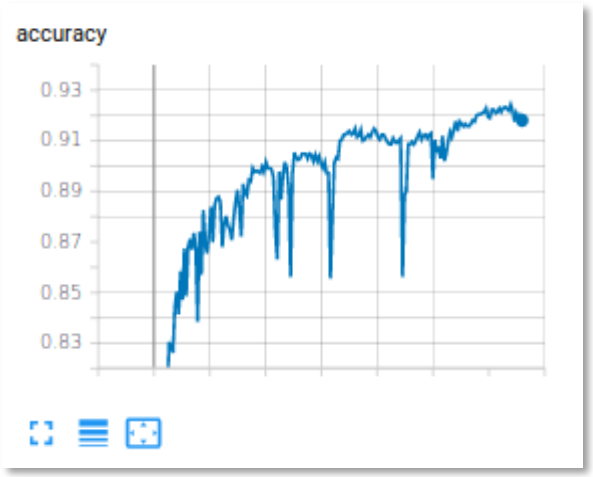

Figure 41: Evaluation accuracy of a 7-layer network

Figure 36, Figure 39 and Figure 42 show that the accuracy of the network does not increase drastically as more layers are added. This occurs because some information is lost from the input images, while being processed by the network.

```
Number of 'left' test images:  25
 Accuracy Left =  0.76
Number of 'right' test images:  25
 Accuracy Right =  0.76
Number of 'neg' test images:  24
 Accuracy Neg =  0.5833333333333334
Overall Accuracy=  0.7011111111111111
```

Figure 42: Testing accuracy of a 7-layer network

5.2.2 Residual Network

The implemented residual network is a much deeper network than the ones described in (5.2.1). This network consists of 50 layers, in a combination of convolutional layers, pooling layers, fully connected layers and normalization layers. The structure of this network follows the architecture in Figure 43.

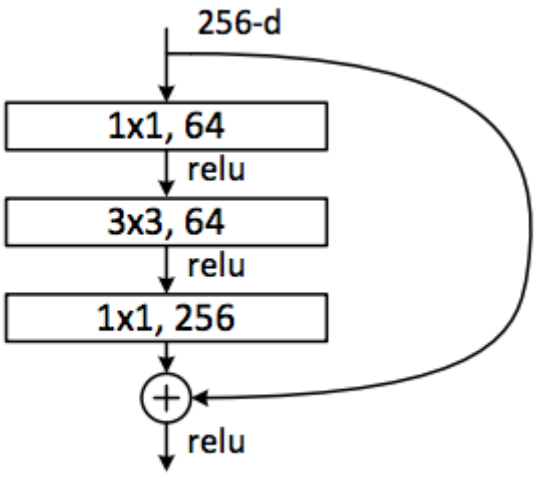

Figure 43: Structure of the residual network [48]

The information from the images passes first through a convolutional layer, then batch normalization and then through an activation function. However, every three consecutive layers, the output information of the third layer is added with a shortcut connection to the information used as input in the first of the three layers. The implementation of a residual block in code can be seen in below:

```
1.      # Save the input value
2.      X_shortcut = X
3.
4.
5.      ##### MAIN PATH #####
6.      # First component of main path
7.      X = Conv2D(filters = F1, kernel_size= (1, 1), strides = (s,s),
8.                 padding="valid", name = conv_name_base + '2a',
                   kernel_initializer = glorot_uniform(seed=0))(X)
9.      X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
10.      X = Activation('relu')(X)
11.
12.      # Second component of main path
13.      X = Conv2D(filters = F2, kernel_size=(f,f), strides=(1,1),
                   name = conv_name_base + '2b', padding="same",
                   kernel_initializer = glorot_uniform(seed=0))(X)
14.      X = BatchNormalization(axis = 3, name= bn_name_base + '2b')(X)
15.      X = Activation('relu')(X)
16.
17.      # Third component of main path
18.      X = Conv2D(filters = F3, kernel_size=(1,1), strides = (1,1),
                   name= conv_name_base + '2c',padding="valid",
                   kernel_initializer=glorot_uniform(seed=0))(X)
```

```
19.       X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)
20.
21.       ##### SHORTCUT PATH ####
22.       X_shortcut = Conv2D(filters = F3, kernel_size= (1,1),
                      strides=(s,s), name=conv_name_base + '1',
                      padding="valid",kernel_initializer=
                      glorot_uniform(seed=0))(X_shortcut)
23.       X_shortcut = BatchNormalization(axis=3,
                      name=bn_name_base+'1')(X_shortcut)
24.
25.       # Final step: Add shortcut value to main path, and pass it
          #through a RELU activation
26.       X = Add()([X_shortcut,X])
27.       X = Activation("relu")(X)
28.
```

As it can be seen from the code above, the last layer of the residual block (named “third component”) as well as the shortcut path do have have any activation function applied to them. The activation function is later applied to the ouput of the concatenation the two layers.

Figure 44 shows a sample of information retrieved from the network regarding its structure and layer connections.

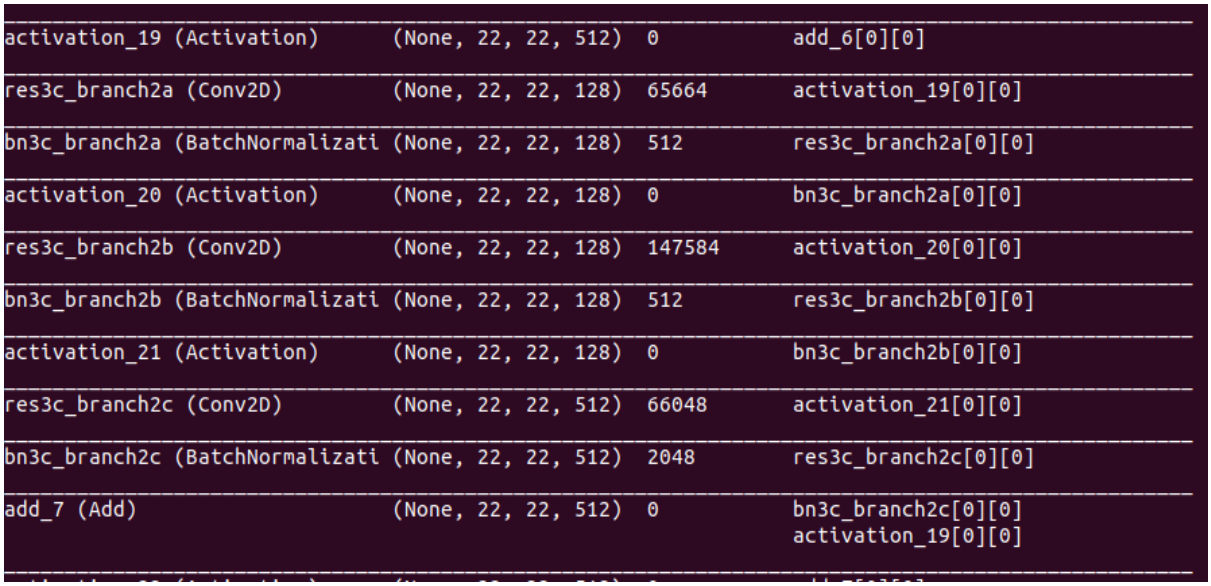

Figure 44: Network information received after training

The model is training for 50 epochs. An epoch is the number of times the network goes through the entire dataset to learn and recognize patterns that exist in the images. The

training process lasted 57 minutes, which is a relatively fast procedure, considering the number of layers and images that needed to be processed.

```
500/500 [==============================] - 106s 212ms/step - loss: 1.6517 - acc: 0.4260 - val_loss: 2.5090 - val_acc: 0.3330
Epoch 2/50
500/500 [==============================] - 90s 180ms/step - loss: 1.6082 - acc: 0.4264 - val_loss: 1.5822 - val_acc: 0.3215
Epoch 3/50
500/500 [==============================] - 90s 181ms/step - loss: 1.4625 - acc: 0.4276 - val_loss: 8.8344 - val_acc: 0.3778
Epoch 4/50
500/500 [==============================] - 90s 180ms/step - loss: 1.3952 - acc: 0.4942 - val_loss: 1.2920 - val_acc: 0.3775
Epoch 5/50
500/500 [==============================] - 91s 181ms/step - loss: 1.3987 - acc: 0.5105 - val_loss: 8.2468 - val_acc: 0.3007
```

Figure 45: Start of training process for residual network

Figure 45 shows the beginning of the training process. The network records and prints information about the training and loss accuracy, as well as evaluation loss and accuracy reached. The information in Figure 45 shows a very high number of validation losses and a very low number of validation accuracy. This appears because the network is just starting to calibrate and adjust its weights depending on features that it learns and correct predictions that it makes.

```
Epoch 45/50
500/500 [==============================] - 86s 172ms/step - loss: 0.0256 - acc: 0.9918 - val_loss: 0.0084 - val_acc: 1.0000
Epoch 46/50
500/500 [==============================] - 86s 172ms/step - loss: 0.0263 - acc: 0.9908 - val_loss: 0.0895 - val_acc: 0.9445
Epoch 47/50
500/500 [==============================] - 87s 174ms/step - loss: 0.0268 - acc: 0.9922 - val_loss: 0.0039 - val_acc: 1.0000
Epoch 48/50
500/500 [==============================] - 86s 172ms/step - loss: 0.0206 - acc: 0.9930 - val_loss: 0.0117 - val_acc: 1.0000
Epoch 49/50
500/500 [==============================] - 86s 171ms/step - loss: 0.0243 - acc: 0.9919 - val_loss: 2.2732 - val_acc: 0.5997
Epoch 50/50
500/500 [==============================] - 86s 172ms/step - loss: 0.0275 - acc: 0.9904 - val_loss: 0.1600 - val_acc: 0.9553
```

Figure 46: Finishing steps of training and evaluation

As it can be seen from Figure 46, the evaluation of the network on the test data reaches an astonishingly high number, getting up to 100% on validating test images, while finishing the training process. This shows that the model performs very effectively on classifying images that are not used in training.

When the training is finished, the program saves a version of the current status of the model as ".h5" extension file. This file can be used for further training of the network or for predicting the classes of images.

In the end of the program, the network reports as to how many parameters were used for training through the whole process. Figure 47 shows the results of this implementation.

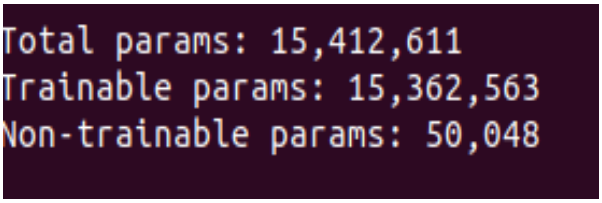

Figure 47: Parameters of Residual Network

As it can be from Figure 47, the network used over 15 million parameters for the training. The non-trainable parameters of the network mean that there are 50048 weights that were not systematically updated and did not contribute to the overall training of the model.

The comparison between the simple convolutional neural networks and the residual network created shows that the residual network has a much higher prediction rate. Its structure allows it to be a stable network and have a large number of layers (in this case 50 layers), without degradation of the output accuracy.

5.3 **Object Detection**

To train models to detect bicycles when images are passed through them, pre-trained models and the topic of transfer learning has been used. The models [49] are pre-trained on multiple categories and objects, and they are further trained to detect bicycles in this implementation. The models used are the *"ssd_mobilenet_v1_coco"* and the *"faster_rcnn_resnet101_coco"*. The difference in speed on these two models can be seen in the following table.

Model	Speed (ms)
ssd_mobilenet_v1_coco	30
faster_rcnn_resnet101_coco	90

Table 3: Speed comparison of pre-trained models

As it can be seen from Table 3], the *"ssd_mobilenet_v1_coco"* is the faster option. In fact, as of the time of writing this thesis, the *"ssd_mobilenet_v1_coco"* is one of the fastest models for TensorFlow. The reason why the *"faster_rcnn_resnet101_coco"* was chosen is to be able to compare the actual time needed for both models to reach the same training global step and to test their accuracy.

For the models to be able to learn from the images that are fed to them, the images need to be converted into a format that is easy to process from TensorFlow. This requires some manual execution from the user. First the XML files of the images need to be converted into a CSV file format file. This is performed by running the script *"xml_to_csv.py"* (see Appendix D), which takes the image name, width and height of the image, the class it belongs to and the maximum and minimum values of the drawn bounding box in the x- and y-axis and converts them into the CSV format. When the script is finished, a file will automatically be created with the name (in this example) *"bike_labels.csv"*. After the file is created, the script *"generate_tfrecord.py"* (see Appendix D) needs to be executed. This script receives as input the previously CSV file created and the images and creates a *".record"* file, which is a TensorFlow specific file used for object detection models. The same process has to be performed twice, one to create a *".record"* file for training and one for evaluation. The training process can then be started by using the *"train.py"* script and consequently the *"eval.py"* script for evaluating the model (see Appendix D).

Faster-RCNN

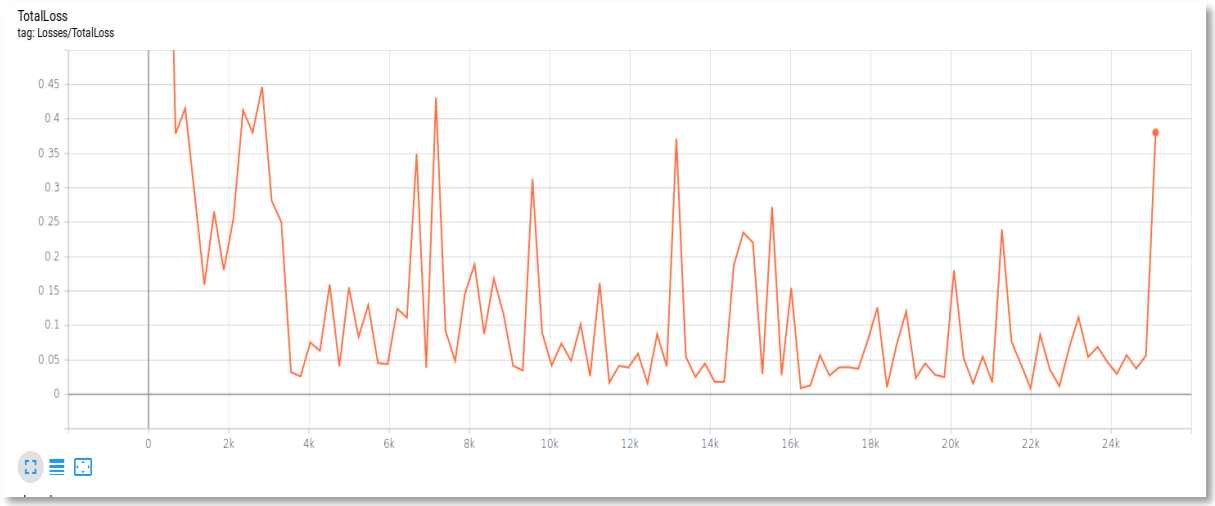

Figure 48: Total loss of faster_rcnn_resnet101_coco

The *"faster_rcnn_resnet101_coco"* was trained for 25000 steps, with a learning rate of 0.0003 and a batch size of 1. The batch size states how many images should be passed through the network for each iteration. The time needed for the training to reach 25000 steps was approximately 3.5 hours.

Figure 49: Evaluation example of Faster R-CNN model

SSD

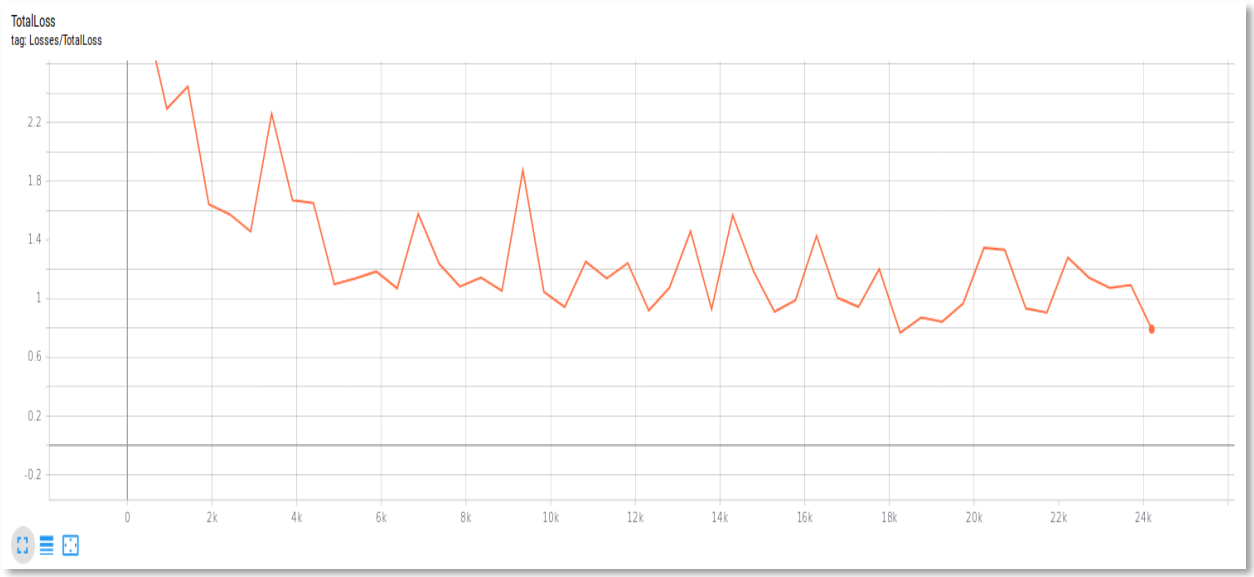

Figure 50: Total loss of ssd_mobilenet_v1_coco

The SSD model is a way faster solution compared to the Faster R-CNN. The time required for training the model up to the global step of 24000 was 1.40 hours, which means it was approximately 2.5 times less. The learning rate used was 0.004 and the batch size 24.

Figure 51: Evaluation example of SSD model

Both models have an excellent accuracy on detecting bicycles in images and are able to place the bounding box correctly, even when the bicycles have an angle towards the camera or are tilted (as seen in the left image of Figure 51). The SSD is trained in a much faster way than the Faster R-CNN. The Faster R-CNN has however a much less loss compared to the SSD, as it can be seen from Figure 50 and Figure 51. This is related to the parameters of the network, since the learning rates and batch size for both models are different.

6 CONCLUSION AND FUTURE WORK

The work explained in this thesis includes both theory as well as practical experimentation on image classification and detection specifically for bicycles. Multiple convolutional neural networks are explained and implemented for bicycle classification, resulting in an increase of over 15% accuracy on the test data with the addition of only 1 extra layer in the network. The implementation of a residual network shows that it is a very powerful and interesting network that should be considered, and it is capable of reaching over 95% accuracy on test data, compared to simpler convolutional neural networks, with the advantage of receiving very low losses for image classification due to its architecture with skip connections.

The pre-trained object detection models used in this thesis, result in an easy implementation of effective models that can recognize bicycles even when the conditions of the test images fed into the network are not of the exact same conditions as the training images. Some of the test images used contained noise or tilted bicycles, but the models were still able to reach accuracies on average above 90%. The experimentation with the SSD and the Faster R-CNN models, showed a 2.5 times faster training in favor of the SSD.

The bicycle dataset used has been developed during other projects of the University of Applied Sciences Hamburg and was further developed in this thesis. The concept of data augmentation was introduced and applied on the overall dataset, enabling an increased number of images.

The further development of this work might include the increase of layers and experimentation on residual networks, with an increased number of images included in the dataset. The dataset should also be further developed with data augmentation to introduce the images to color variations, images of bicycles with rotations and different perspectives in respect to the camera. Additionally, the networks should be configured to test data retrieved from a live-feed of a camera. This will provide an approximation of the end result of the project.

7 GLOSSARY

SSH	–	Secure Shell
VM	–	Virtual Machine
RSA	–	Rivest–Shamir–Adleman
WoL	–	Wake-on-LAN
LAN	–	Local Area Network
BIOS	–	Basic Input/Output System
MAC	–	Media Access Control
ACPI	–	Advanced Configuration and Power Interface
AC	–	Alternating Current
Wi-Fi	–	Wireless Fidelity
API	–	Application Program Interface
CUDA	–	Computer Unified Device Architecture
GPU	–	Graphics Processing Unit
XML	–	Extensible Markup Language
CPU	–	Central Processing Unit
CNN	–	Convolutional Neural Network
ReLU	–	Rectified Linear Unit
SSD	–	Single-Shot Detector
RGB	–	Red-Green-Blue
CSV	–	Comma-Separated Values
RAM	–	Random Access Memory

8 REFERENCES

[1] K. Dimitrov, "Image based detection and location of bicycles using deep learning with convolutional neural networks," Bachelor-Thesis at Department of Information and Electrical Engineering of HAW Hamburg, 2019.

[2] D. Moore, "Machine Learning," [Online]. Available: https://towardsdatascience.com/machine-learning-d352adb85e5e. [Accessed 06 09 2019].

[3] "Supervised vs Unsupervised Learning: Key Differences," [Online]. Available: https://www.guru99.com/supervised-vs-unsupervised-learning.html. [Accessed 18 11 2019].

[4] N. Castle, "What is Semi-Supervised Learning?," [Online]. Available: https://blogs.oracle.com/datascience/what-is-semi-supervised-learning. [Accessed 12 11 2019].

[5] . B. Osiński and K. Budek, "What is reinforcement learning? The complete guide," [Online]. Available: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/. [Accessed 18 11 2019].

[6] R. Bhatia, "When not to use Neural Networks," [Online]. Available: https://medium.com/datadriveninvestor/when-not-to-use-neural-networks-89fb50622429. [Accessed 08 12 2019].

[7] T. Babs, "The Mathematics of Neural Networks," [Online]. Available: https://medium.com/coinmonks/the-mathematics-of-neural-network-60a112dd3e05. [Accessed 15 11 2019].

[8] A. S. V, "Understanding Activation Functions in Neural Networks," [Online]. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0. [Accessed 23 11 2019].

[9] S. Sharma, "Activation Functions in Neural Networks," [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed 24 11 2019].

[10] P. Pandey, "Understanding the Mathematics behind Gradient Descent.," [Online]. Available: https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e. [Accessed 05 11 2019].

[11] S. Kostadinov, "Understanding Backpropagation Algorithm," [Online]. Available: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd. [Accessed 25 10 2019].

[12] V. Nigam, "Understanding Neural Networks. From neuron to RNN, CNN, and Deep Learning," [Online]. Available: https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90. [Accessed 06 12 2019].

[13] K. Sorokina, "Image Classification with Convolutional Neural Networks," [Online]. Available: https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8. [Accessed 08 12 2019].

[14] Ujjwalkarn, "An Intuitive Explanation of Convolutional Neural Networks," [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. [Accessed 08 11 2019].

[15] J. Brownlee, "How Do Convolutional Layers Work in Deep Learning Neural Networks?," [Online]. Available: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/. [Accessed 23 11 2019].

[16] V. Nigam, "Understanding Neural Networks. From neuron to RNN, CNN, and Deep Learning," [Online]. Available: https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90. [Accessed 06 12 2019].

[17] "Max-pooling / Pooling," [Online]. Available: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling. [Accessed 06 12 2019].

[18] Edpresso, "What is dropout in neural networks?," [Online]. Available: https://www.educative.io/edpresso/what-is-dropout-in-neural-networks. [Accessed 07 12 2019].

[19] A. Ag, "Batch Normalization in Deep Learning," [Online]. Available: https://medium.com/ai%C2%B3-theory-practice-business/batch-normalization-in-deep-learning-ca215a7a7a5d. [Accessed 01 12 2019].

[20] keitakurita, "An Intuitive Explanation of Why Batch Normalization Really Works," [Online]. Available: https://mlexplained.com/2018/01/10/an-intuitive-explanation-of-why-batch-normalization-really-works-normalization-in-deep-learning-part-1/. [Accessed 01 12 2019].

[21] A. Rakhecha, "Understanding Learning Rate," [Online]. Available: https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de. [Accessed 12 11 2019].

[22] N. Donges, "What is Transfer Learning? Exploring the popular deep learning approach," [Online]. Available: https://builtin.com/data-science/transfer-learning. [Accessed 17 09 2019].

[23] A. Ananthram, "Deep Learning for beginners using Transfer Learning in Keras," [Online]. Available: https://towardsdatascience.com/keras-transfer-learning-for-beginners-6c9b8b7143e. [Accessed 06 11 2019].

[24] A. S. Walia, "The Vanishing Gradient Problem," [Online]. Available: https://medium.com/@anishsingh20/the-vanishing-gradient-problem-48ae7f501257. [Accessed 01 11 2019].

[25] P. Ruiz, "Understanding and visualizing ResNets," [Online]. Available: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8. [Accessed 29 11 2019].

[26] A. Dabydeen, "Transfer Learning Using ResNet50 and CIFAR-10," [Online]. Available: https://medium.com/@andrew.dabydeen/transfer-learning-using-resnet50-and-cifar-10-6242ed4b4245. [Accessed 07 12 2019].

[27] Algorithmia, "Introduction to Dataset Augmentation and Expansion," [Online]. Available: https://algorithmia.com/blog/introduction-to-dataset-augmentation-and-expansion. [Accessed 19 10 2019].

[28] "Google Brain Team," [Online]. Available: https://ai.google/research/teams/brain/. [Accessed 26 08 2019].

[29] "An end-to-end open source machine learning platform," [Online]. Available: https://www.tensorflow.org/. [Accessed 24 08 2019].

[30] S. Yegulalp, "What is TensorFlow? The machine learning library explained," [Online]. Available: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html. [Accessed 19 08 2019].

[31] "Serving Models," [Online]. Available: https://www.tensorflow.org/tfx/guide/serving. [Accessed 15 11 2019].

[32] "TensorFlow Serving," [Online]. Available: https://github.com/tensorflow/serving. [Accessed 25 11 2019].

[33] "TensorBoard: TensorFlow's visualization toolkit," [Online]. Available: https://www.tensorflow.org/tensorboard. [Accessed 10 09 2019].

[34] "Get started with TensorBoard," [Online]. Available: https://www.tensorflow.org/tensorboard/get_started#using_tensorboard_with_other_methods. [Accessed 10 09 2019].

[35] "TensorBoard Histogram Dashboard," [Online]. Available: https://github.com/tensorflow/tensorboard/blob/master/docs/r1/histograms.md. [Accessed 10 09 2019].

[36] "Path graph," [Online]. Available: https://en.wikipedia.org/wiki/Path_graph. [Accessed 25 11 2019].

[37] "Getting started with the Keras Sequential," [Online]. Available: https://keras.io/getting-started/sequential-model-guide/. [Accessed 10 09 2019].

[38] "Directed acyclic graph," [Online]. Available: https://en.wikipedia.org/wiki/Directed_acyclic_graph. [Accessed 25 11 2019].

[39] "The Keras functional API in TensorFlow," [Online]. Available: https://www.tensorflow.org/guide/keras/functional. [Accessed 25 11 2019].

[40] Docker, "What is a Container?," [Online]. Available: https://www.docker.com/resources/what-container. [Accessed 10 06 2019].

[41] OpenSource, "What is Docker?," [Online]. Available: https://opensource.com/resources/what-docker. [Accessed 15 11 2019].

[42] "SSH Key," [Online]. Available: https://www.ssh.com/ssh/key/. [Accessed 15 08 2019].

[43] D. Reid, "How To: Wake on LAN / Wake on WAN," [Online]. Available: https://www.smallnetbuilder.com/lanwan/lanwan-howto/29941-how-to-wake-on-lan-wake-on-wan?start=1. [Accessed 24 08 2019].

[44] AMD, "Magic Packet Technology," [Online]. Available: https://www.amd.com/system/files/TechDocs/20213.pdf. [Accessed 14 11 2019].

[45] M. Crider/Yatritrivedi, "What Is Wake-on-LAN, and How Do I Enable It?," [Online]. Available: https://www.howtogeek.com/70374/how-to-geek-explains-what-is-wake-on-lan-and-how-do-i-enable-it/. [Accessed 25 08 2019].

[46] Z. Seng, "D-Link DSP-W215 WiFi Smart Plug Review," [Online]. Available: https://zitseng.com/archives/14373. [Accessed 08 12 2019].

[47] "LabelImg," [Online]. Available: https://github.com/tzutalin/labelImg. [Accessed 30 09 2019].

[48] Ö. Şahin, "How to fine-tune ResNet in Keras and use it in an iOS App via Core ML," [Online]. Available: https://heartbeat.fritz.ai/how-to-fine-tune-resnet-in-keras-and-use-it-in-an-ios-app-via-core-ml-ee7fd84c1b26. [Accessed 09 12 2019].

[49] "Tensorflow detection model zoo," [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. [Accessed 20 09 2019].

[50] M. Peixeiro, "Hitchhiker's Guide to Residual Networks (ResNet) in Keras," [Online]. Available: https://towardsdatascience.com/hitchhikers-guide-to-residual-networks-resnet-in-keras-385ec01ec8ff. [Accessed 02 10 2019].

9 APPENDICES

9.1 **Appendix A**

The implementation of the SSH requires access to the server computer and the client computer. To allow access to the from a client to the server, the following procedure must be followed:

- Generation of keys. The client must generate a private and a public key which will be used to authenticate the access to the computer. This is done by using an SSH command in the command prompt: "*ssh-keygen*". This automatically generates a new combination of public and private key. The user is also able to select where this file containing the private key should be stored. It is recommended however to leave the path to its default settings.
- Enter a password. This is an optional step which adds an extra layer of security to the connection by encrypting the private key on the disk.
- Store the public key on the server. On the client computer with a Windows OS, the file containing the public key can be sent to the server by using the following command:

```
scp <PATH_TO_FILE>/id_rsa.pub username@<IP_Address>:~/.ssh/authorized_keys
```

If everything is setup in the correct way, then the user is able to access the remote server by using the following command:

```
ssh -X username@<IP_Address> -p <PORT_NUMBER>
```

The default port number for SSH is 22, however when the configuration of the server has been changed and a specific port has been set for SSH connections, then it needs to be included in the command for a successful connection.

9.2 **Appendix B**

```
1.  import cv2
2.  import glob
3.  import xml.etree.ElementTree as ET
4.  import os
5.
6.  folder_path = "images/"
7.  xml_ext = "xml"
8.  jpg_ext = "jpg"
9.  asterisk = "*"
10. dot = "."
11. image_size = 176
12. category = 'bike_right'  # Change this depending on what the label in the XML file should be
13. direction = 'right'  # Name of the files
14. last_number = 1289  # Number on the name of the last picture in final image dataset (in order to have
    consistency)
15.
16.
17. def xml_file(img):
18.     xml = folder_path + img.split('\\')[1].split(dot + jpg_ext)[0]  + dot + xml_ext
19.
20.     tree = ET.parse(xml)
21.     root = tree.getroot()
22.
23.     for f in root.findall('object'):
24.         name = f.find('name')
25.         bndbox = f.find('bndbox')
26.
27.         xmin = bndbox.find('xmin')
28.         xmax = bndbox.find('xmax')
29.
30.         name.text = category
31.
32.         delta_x = int(xmax.text) - int(xmin.text)
33.
34.         xmax.text = str(image_size - int(xmin.text))
35.         xmin.text = str(image_size - int(xmin.text) - delta_x)
36.
37.     tree.write(xml)
38.
39.     return img.split('\\')[1].split(dot + jpg_ext)[0]
40.
41.
42. def xml_file_change_name(img, new_name):
43.     xml = folder_path + img.split('\\')[1].split(dot + jpg_ext)[0] + dot + xml_ext
44.
45.     tree = ET.parse(xml)
46.     root = tree.getroot()
47.
48.     for filename in root.findall('filename'):
49.         filename.text = new_name
50.
51.     tree.write(xml)
52.
```

```
53.
54.  def get_name(filename):
55.      xml = folder_path + filename
56.      tree = ET.parse(xml)
57.      root = tree.getroot()
58.
59.      for filename in root.findall('filename'):
60.          text = filename.text
61.
62.      text = text.split(dot + jpg_ext)[0] + dot + xml_ext
63.
64.      tree.write(xml)
65.      return text
66.
67.
68.  def convert_name():
69.
70.      images_path = folder_path + asterisk + dot + jpg_ext
71.      image_files = glob.glob(images_path)
72.      i = last_number
73.      for img in image_files:
74.          i = i + 1
75.          new_name = direction + " (" + str(i) + ").jpg"
76.          os.rename(img, folder_path + "/" + new_name)
77.          xml_file_change_name(img, new_name)
78.
79.      for filename in os.listdir(folder_path):
80.          if filename.split(dot)[1] == xml_ext:
81.              name = get_name(filename)
82.              os.rename(folder_path + "/" + filename, folder_path + "/" + name)
83.
84.
85.  def convert():
86.
87.      images_path = folder_path + asterisk + dot + jpg_ext
88.      image_files = glob.glob(images_path)
89.      for img in image_files:
90.          image = cv2.imread(img)
91.          image = cv2.flip(image, 1)  # Flip the image horizontally
92.          image_name = xml_file(img)
93.          cv2.imwrite(os.path.join(folder_path, dot.join([image_name, jpg_ext])), image)
94.
95.      convert_name()
96.
97.
98.  convert()
```

9.3 **Appendix C**

The user can record the utilization of the GPUs by executing the following command:

```
while true; do nvidia-smi --query-gpu=utilization.gpu --format=csv >>
        gpu_utillization.log; sleep 1; done
```

This command will automatically create a file and continuously write the percentage of the GPUs used until it is stopped. The following is an example of how the output looks like:

```
utilization.gpu [%]
56 %
32 %
utilization.gpu [%]
84 %
37 %
```

9.4 **Appendix D**

Description of software on CD-ROM

The CD-ROM attached to this thesis contains software that related to the implementation and development of this thesis.

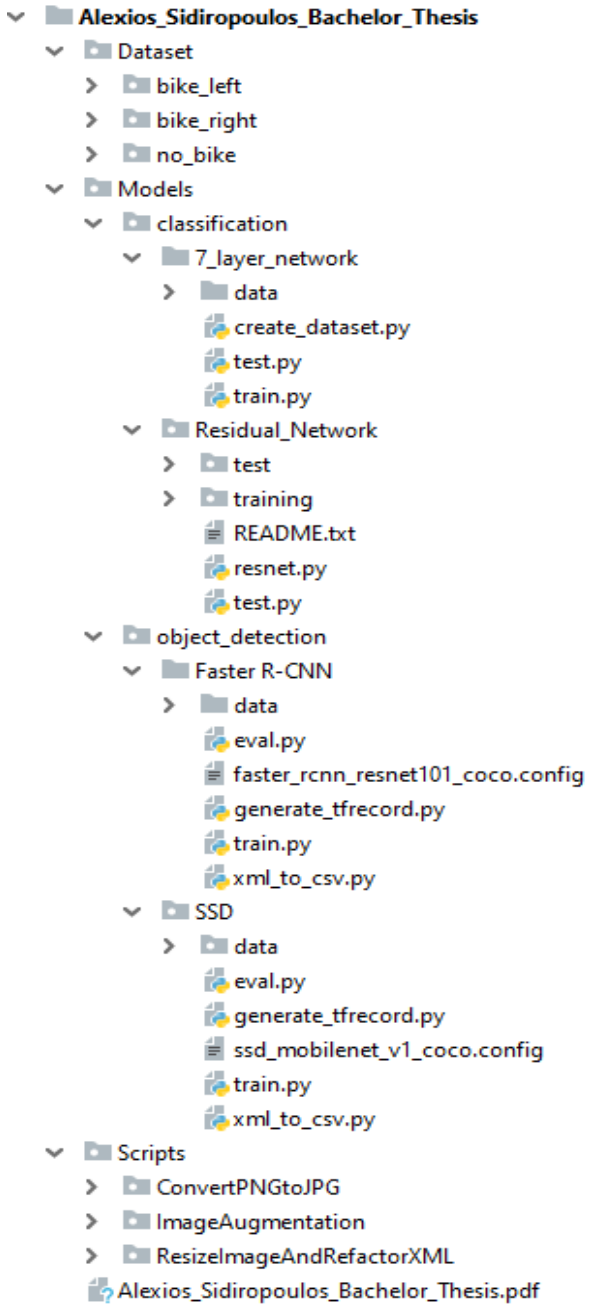

Figure 52: Folder structure on CD-ROM

The folder *"Alexios_Sidiropoulos_Bachelor_Thesis"* contains three folders, *"Dataset"*, *"Models"* and *"Scripts"*, and an PDF file named *"Alexios_Sidiropoulos_Bachelor_Thesis.pdf"*.

The file *"Alexios_Sidiropoulos_Bachelor_Thesis.pdf"* is a digital copy of this bachelor thesis.

The *"Dataset"* folder contains all the images that were curated and used during this thesis. It contains 3 sub-folders, where each contains a class of the images that can be used for classification or bicycle detection, namely *"bike_left"*, *"bike_right"* and *"no_bike"*. The folders contain also the related XML files of the images.

The *"Models"* folder contains the files needed for training networks for classification or object detection. In the *"classification"* folder, there exist 2 sub-folders, one containing a program for training a simple CNN and the other the program for training a residual network for image classification. In each folder, one can find the scripts for training and testing the neural networks, along with some examples of images and TFRecords files that need to be used. In the folder *"7_layer_network"*, there exists also a file named *"create_dataset.py"*, which converts the images into TFRecords files to be used by TensorFlow. The *"object_detection"* folder also contains 2 sub-folders, one for an implementation of the SSD model and one for the Faster R-CNN model. Each of these folders contains its configuration file, training script, script to convert the image and XML files into the required formats and the evaluation script for testing the networks.

The *"Scripts"* folder contains scripts that can be used to manipulate the dataset. The *"ConvertPNGtoJPG"* contains a simple script that allows the user to convert images from the ".png" file extension into the ".jpg" file extension. The *"ImageAugmentation"* contains a script that allows the user to flip images in order to expand the dataset. The script requires the user for some input regarding the naming convention of the images that will be created, and it automatically refactors the bounding box of the XML files of the images. It additionally performs all the necessary modification of the XML files, like changing the class of the image, filename, etc. Lastly, the folder *"ResizeImageAndRefactorXML"* contains a script which, as the name implies, resizes the images into the specified dimensions and refactors the bounding box of the XML files to fit in the new dimensions after the conversion.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach§16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfs-mittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellenhabe ich unter Angabe der Quellen kenntlich gemacht.

_____ _____

Ort, Datum Unterschrift