Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelor Thesis

Jeonghyun Son

## Evaluation of Convolutional Neural Network performance using synthetic data for training

*Fakultät Technik und Informatik*
*Department Informations- und Elektrotechnik*

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

Jeonghyun Son

# Evaluation of Convolutional Neural Network performance using synthetic data for training

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.Ing. Lutz Leutelt

Day of delivery 8. October 2019

**Jeonghyun Son**

## Title of the paper

Evaluation of Convolutional Neural Network performance using synthetic data for training

## Keywords

Computer graphics, Machine learning, Deep Learning, Convolutional neural networks, Object Classification, Synthetic data, Three.js

## Abstract

One of the limitations of supervised learning in deep learning algorithm is to gather and label a large set of data. In this document, the approach to solve this limitation is presented by using synthetic data. A scene of a real-like traffic situation with bicycles is created with 3D framework, THREE.js. The synthetic data is automatically generated with labels by taking a screenshot of rendering scene. The data is used to train on convolutional nerual network for image classification. At the end, the performance of convolutional neural network model is evaluated on real image dataset.

**Jeonghyun Son**

## Thema der Bachelor Thesis

Bewertung der Leistung von faltenden neuronalen Netzwerken bei Verwendung synthetischer Daten für das Training

## Stichworte

Computergrafik. Machinelles Lernen, Deep Learning, Faltungsneuronale Netze, Objektklassifizierung, Synthetische Daten, Three.js

## Kurzzusammenfassung

Eine große Herausforderung während des Trainings von faltenden neuronalen Netzwerken besteht darin, eine große Menge gekennzeichneter Daten zu Verfügung zu stellen. In dieser Arbeit wird ein Ansatz zur Lösung dieser Einschränkung durch die Verwendung synthetischer Trainingsdaten vorgestellt. Mithilfe des 3D Framework THREE.js wird eine Szene einer realitätsnahen Verkehrssituation mit Fahrrädern erzeugt. Anschließend werden synthetische Trainingsdaten mit Kennzeichnungen generiert, indem eine Vielzahl von Momentaufnahmen der Szene erstellt werden. Diese Daten werden anschließend verwendet, um ein faltendes neuronales Netzwerk für eine Bildklassifizierungsaufgabe zu trainieren. Abschließend wird die Leistung des Modells des Netzwerks mit Hilfe von realen Bilddaten bewertet.

# Contents

**List of Figures** **7**

**List of Tables** **9**

**1. Introduction** **10**

1.1. Previous work . . . . . 10

1.2. Objectives . . . . . 11

**2. Theory** **12**

2.1. Machine Learning . . . . . 12

2.1.1. Supervised and unsupervised learning . . . . . 13

2.1.2. Deep Learning and neural network . . . . . 13

2.1.3. Structure of neural network . . . . . 14

2.1.4. Convolutional neural network . . . . . 15

2.1.5. Convolutional Neural Network for Image Classification . . . . . 17

2.1.6. Overfitting and underfitting . . . . . 17

2.1.7. Data augmentation . . . . . 17

2.1.8. Training data, validation data and test data . . . . . 18

2.2. Computer graphics . . . . . 18

2.2.1. Three.js framework . . . . . 19

2.2.2. View Frustum in Three.js . . . . . 19

2.2.3. 3D world to 2D screen coordinate system . . . . . 21

2.2.4. Blob Web API . . . . . 22
2.2.5. Skeletal animation . . . . . 23

**3. Requirements and Design** **24**
3.1. Requirements . . . . . 24
3.2. Synthetic data design . . . . . 24
3.3. Convolutional neural network setup . . . . . 26

**4. Implementation** **27**
4.1. Synthetic data generation . . . . . 27
4.1.1. Data creation . . . . . 27
4.1.2. Rigging with Blender . . . . . 28
4.1.3. Determination of object coordinates . . . . . 29
4.1.4. Automatic data extraction . . . . . 30
4.1.5. Synthetic data augmentation . . . . . 31
4.2. Convolutional neural network . . . . . 32
4.2.1. Train CNN for classification . . . . . 32
4.2.2. Evaluation process . . . . . 33

**5. Results** **36**
5.1. Data result . . . . . 36
5.2. Training result . . . . . 38
5.3. Classification result . . . . . 39

**6. Conclusion and Future work** **41**

**List of Abbreviations** **43**

**References** **44**

**Appendices** **48**

**A. Description of programs on a WD hard drive** **49**

# List of Figures

2.1. Artificial neural network [13] . . . . . 14
2.2. CNN structure [15] . . . . . 15
2.3. Field of view(fov) [24] . . . . . 20
2.4. Camera view [24] . . . . . 20
2.5. The axis of screen and cartesian coordinates. Left figure represents screen coordinates and right figure represents cartesian coordinates [26] . . . . . 21

3.1. Texture files for bicycle 1 and 2 . . . . . 25
3.2. Texture files for background . . . . . 26

4.1. 3D model before and after rigging in Blender . . . . . 28
4.2. Determination of points. Max. (yellow) and min. (orange) point from the side view. Note that the box and points are drawn only for this figure. . . . . 29
4.3. Adjustment of maximum and minimum points. Max. (yellow) and min. (orange) point from side camera angle view. Note that the box and points are drawn only for this figure. . . . . 30
4.4. Synthetic image before and after adding noise . . . . . 31
4.5. Synthetic image before and after adding blurred effect . . . . . 31
4.6. Synthetic image before and after adding 10 degree angle . . . . . 32
4.7. Real images used in previous work from Kaloyan Dimitrov [29] . . . . . 34
4.8. Real images taken on the street . . . . . 35

5.1. Synthetic images generated by Three.js . . . . . 37

5.2. Image with annotation file . . . . . 38

5.3. Validation accuracy and loss graph from tensorboard . . . . . 39

# List of Tables

3.1. Variation in rendering scene . . . . . 25

4.1. Number and type of synthetic images . . . . . 32
4.2. Real image datasets for evaluation . . . . . 33

5.1. Test accuracy on first model . . . . . 39
5.2. Test accuracy on second model . . . . . 40
5.3. Test accuracy on final model . . . . . 40

# 1 Introduction

In modern society, the concept of machine learning is used in a wide range of fields. Due to the accessibility of computing power, the vast number of data generated each day and technical evolution, the field of machine learning has made revolutionary advances last five years.[1]

Among all machine learning techniques, deep learning has recently risen in popularity in certain fields, notably in computer vision.[1] However, training the neural network in deep learning often requires a large amount of data to ensure high accuracy. Especially for supervised learning, collecting and labeling big datasets can be tedious and time-consuming work.

To solve these issues, various approaches have been taken over time. One of the recent approaches is to use synthetic data using computer graphics techniques. After the success of deep convolutional neural networks in various vision tasks concerning object detection or classification, generation and use of synthetic dataset has been frequently considered. With the latest advances in 3D technology, the variation of data is much wider and the creation of labels can be automated by machines.[2]

## 1.1. Previous work

As using synthetic data has become one of the more attractive alternatives and effective methods to replace time-consuming work such as capturing and annotating training data, several attempts have been made to train a neural network with synthetic data.

In the field of optical character recognition, synthetic images have long been applied for training by adding noise and deformation.[3] Besides, the synthetic images are generated to be fed to a convolutional neural network to learn how to detect company logos in the absence of a large training set.[2] Although these two researches have been performed on less complex images, the result of neural network performance has proven to be effective.

Furthermore, the other research shows that this approach has been used on different pedestrian detectors. In [4], the attempt was made to train a pedestrian detector by generating synthetic images of pedestrians in various poses and environments. Similarly, another research was focused on counting the number of pedestrians using Deep models trained on synthetically generated images.[5] The results on both researches are encouraging and successful for generating a large set of data without a huge effort, yet there is still a room for improvement by applying complex artifacts on images.

Furthermore, an attempt to increase the performance of neural network on image detection was made in [6], where synthetic data is applied with different post-processing effects such as object boundary blurring and Motion blurring to make it as close to real life images. Interestingly, the result of [6] shows that the average precision of deep learning model using synthetic data for training was higher than using only real life data in certain cases.

## 1.2. Objectives

In this paper, a deep learning technique using convolution neural network is performed on classification of data. The convolutional neural network is selected because of its strong ability in image processing. The network is trained with synthetic data generated by Three.js framework, which renders the scene of a traffic situation with bicycles in different settings. Furthermore, the approach on automatic image creation from the scene including label and annotation file is discussed hereafter. The goal of this project is to observe and evaluate the performance of a convolutional neural network on image datasets from the real world while using synthetic data for training. The detail requirements for this work are presented in Chapter 3.

# 2 Theory

## 2.1. Machine Learning

In his well-known quote, Tom Michell[8] stated that

> "the computer program is said to learn from experience $E$ with respect to some class of tasks T and performance measure $P$, if its performance at tasks in $T$."

Based on this quote, it can be interpreted that the computer learns from data $(E)$, and performs tasks and makes decisions $(T)$ by itself. Its performance $(P)$ can be measured through evaluation.

In scientific terms, machine learning is a subset of artificial intelligence that enables a system to learn from data and perform tasks without explicit programming. It uses a variety of algorithms that can learn from data to improve, describe data and predict outcomes. The algorithm produces a model based on training with a large set of data. In machine learning, the model is the result of algorithms based on training, and provides an output to the task.[9]

In general, machine learning is used for two purposes. First, tasks which are too complex to be performed by humans. Machine learning algorithms can analyze very large and complex datasets in a short time, such as performing weather predictions and analyzing Web search engines and astronomical data. Detecting patterns in large and complex datasets is the most promising domain since the program can learn with almost unlimited memory and the increase of speed of processors has been contributing to this work.

The second purpose of machine learning is adaptivity. In contrast to conventionally programmed tools which cannot be changed once they have been written, machine learning programs introduce flexibility and adaptivity. As this kind of program is based on input, it is adaptive to changes in the environment and offers a solution. Typical application examples are the programs that require to adapt to variations of different users, such as spam detection and speech recognition.[10]

### 2.1.1. Supervised and unsupervised learning

There are different approaches of learning in machine learning. In this chapter, two main approaches are discussed, supervised and unsupervised learning. Supervised learning generally means that the program is given with input data and desired output data, intended to find the pattern in data. In supervised learning, the program is trained with data which has already been labeled with the attributes and meaning of data. Therefore, the behaviour of the program can be predicted for a new dataset. The main two algorithms are regression and classification. Supervised training models have been used in various areas, including fraud detection, recommendation solutions, speech recognition, or risk analysis. [11][12]

In contrast to the supervised learning, unsupervised learning is not provided with the desired output. The program finds the structure and interference between dataset without knowing the meaning of the data. The most common unsupervised learning method is cluster analysis, which can be used for exploratory data analysis to find the hidden pattern or structures in data.[11]

### 2.1.2. Deep Learning and neural network

Deep learning is a sub-field of machine learning which puts an emphasis on learning successive layers of representations from data. The number of layers contributing to a model is called depth of the model. The modern deep learning often involves numerous successive layers of representations which is exposed to learn automatically from training data. These successive layered representations learned via models are called neural

networks, structured in literal layers on top of each other.[12]

A neural network is a network of neurons consisting of three or more layers, including an input layer, hidden layers and an output layer. The term neural network comes from neurobiology, inspired by the understanding of the human brain. It is designed to emulate how the human brain works so computers can be trained to deal with abstractions and problems that are poorly defined.[12]

### 2.1.3. Structure of neural network

The detailed structure of a neural network is shown in figure 2.1 which describes a brief representation of artificial neural network, a part of neural network family. [13]

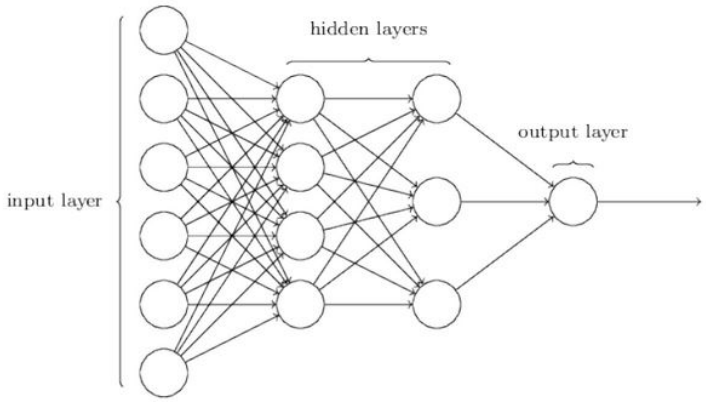


Figure 2.1.: Artificial neural network [13]

The neural network consists of neurons and layers. The fundamental element of neural networks are neurons, the round circles in figure 2.1, which receive information and send the result to other cells in the layers. The artificial neuron takes in some input values $x$ = [ $x_1$, $x_2$, ... $x_n$] and each of which is multiplied by a specific weight $w$ = [ $w_1$, $w_2$, ... $w_n$]. These weighted inputs are summed together and produce logits of neurons $z$, defined as

$$z = \sum_{i=0}^{n} w_i x_i \tag{2.1}$$

The logit of neuron is then passed through a function $f$, also known as an activation function. The activation function is an indication of active neurons and produces the output

$y = f(z+b)$, where $b$ is bias term. This output $y$ from the function can be transmitted to another neuron. There are different types of activation functions such as Sigmoid, Tanh, ReLU and softmax.[14]

The first layer in the neural network is called the input layer, and the neurons within this layer are called input neurons. The right output layer contains the output neurons or a single output neuron. The middle layer is called a hidden layer since the neurons in this layer are neither inputs or outputs. The original input from the input data is ingested through the input layer and stored in input neurons. Then the data is modified in the hidden layer and the output layers based on the weights are applied to the nodes. The typical neural network may consist of thousands or even millions of simple processing nodes that are closely interconnected. For deep learning, it contains more hidden layers than a traditional neural network. The number of hidden layers in the model increases depending on the complexity of the problem. [12]

### 2.1.4. Convolutional neural network

In deep learning, convolutional neural networks, CNNs, are a type of neural network commonly used for processing data that has a grid-like topology, such as a time-series or images. The term convolutional neural network derived from the mathematical operation called convolution, which is the integration of two given functions and expresses how the shape of one is modified by the other. CNN takes the image's raw pixel data as input and learns to extract these features, and ultimately decides what objects it infers to. [16][15]

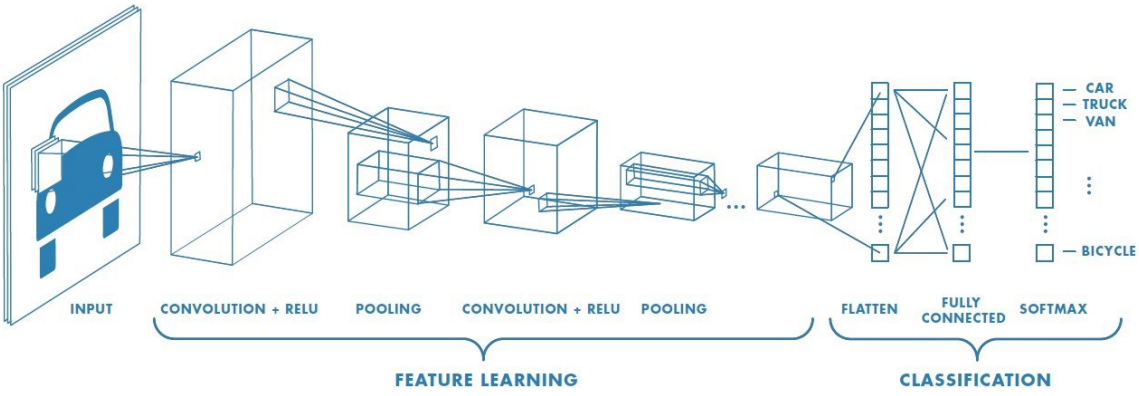


Figure 2.2.: CNN structure [15]

Convolutional neural networks for classification are usually composed of a set of layers that can be grouped by their functionalities and followed by one to the other. To start, the CNN receives an input feature map from the input image. The feature map is derived from an image's raw pixel data which is a three-dimensional matrix where the first two are width and length, and the third dimension is 3, 3 channels of color. [16]

n the convolution layer, convolution extracts tiles of the input feature map and applies filters to them to compute new features, producing an output feature map. Convolutions are defined by two parameters based on the size of the input feature tiles, typically 3x3 or 5x5. The depth of the output feature map corresponds to the number of filters that are applied. During a convolution, the filters effectively slide over the input feature map's grid horizontally and vertically, one pixel at a time, extracting each corresponding tile. For each filter-tile pair, the CNN performs element-wise multiplication of the filter matrix and the tile matrix and then sums all the elements of the resulting matrix to get a single value. Each of these resulting values for every filter-tile pair is then output in the convolved feature matrix.[16]

After each convolution operation, the CNN applies a Rectified Linear Unit layer(ReLU), an activation function which transformation to the convolved feature in order to introduce non linearity into the model. The ReLU function is defined as $F(x) = max(0, x)$ that returns $x$ for all values of $x > 0$, and returns 0 for all values of $x \leq 0$. [16]

Followed by the ReLU layer, Max-pooling Layer is applied to down-sample the convolved feature and reduce the number of dimensions of the feature map. It is typically used to reduce the cost and time for computing time. It divides down the feature map and extracts tiles of a specified size. For each tile, the maximum value is output to a new feature map, and all other values are discarded. Max pooling operations take two parameters, size of max-pooling filter and stride.[16]

Finally, fully connected layers perform the classification using neurons based on the features extracted by convolution computation. Typically, the final fully connected layer contains a softmax activation function, which outputs a probability value from 0 to 1 for each of the classification labels the model is trying to predict. [16]

### 2.1.5. Convolutional Neural Network for Image Classification

Image classification has become one of the most important topics in computer vision and machine learning. A neural network has been applied to an image classification problem in various areas, for example, google photos have used the classifier to categorize the user's image content. Due to the rapid development of computing platforms like GPU, increasingly more researchers start to apply this algorithm in complex image classification. Normally it takes more than dozens of hours to train a good performance model, even with high-performance GPU and some other parallel computing techniques.[17][16]

Image classification implements a supervised learning technique. It defines a set of target classes and trains a model to recognize them using labeled example photos. Early models relied on raw pixel data as the input, however, it cannot encompass the variety of an object, for example, different camera angles and focus, lighting, the position of an object and its background. These differences are significant enough that they cannot be corrected by taking weighted averages of pixel RGB values. To model objects more flexibly, classic computer vision models added new features derived from pixel data, such as color histograms, textures, and shapes.[16]

### 2.1.6. Overfitting and underfitting

Overfitting refers to a machine learning model that is trained too well by training data. This happens when a model learns all details and noise in training data and thus has negative impacts on performance on new data. For example, training with one single data on a model would result in a negative performance on new data. Overfitting can be prevented using data augmentation (see 2.1.7) or using dropout regularization. Underfitting refers to a model that cannot model the training data and generalize to new data. It is likely caused by too much difference gap between training data.[18] [16]

### 2.1.7. Data augmentation

Deep learning models for image classification or object detection often requires a large set of data. Data augmentation is a strategy to expand the diversity of data available

for training without acquiring new data. It is especially useful for situations where the available data is limited to training a network, or the data is user generated content. Data augmentation techniques include cropping, padding, flipping, color-based transformation and adding noise and distortion.[20]

### 2.1.8. Training data, validation data and test data

In machine learning, a model is built to learn and make predictions based on data. In general, multiple datasets are used to build models, namely Training, validation, and test dataset. The training dataset is used to fit the model. The model is trained based on this dataset. The validation dataset is the data used to provide an unbiased evaluation of a model fit on the training datasets while tuning model hyperparameters. The test dataset is a sample of new data that is used to test the final model trained by the training dataset. It is important to keep test data separate from training and validation data to make an accurate prediction. [19]

## 2.2. Computer graphics

The term “computer graphics” refers to anything involved in the creation or manipulation of images on the computer, including animated images. It is a subset of computer science which includes the study of digitally synthesized visual content, such as two and three-dimensional graphics and image processing. Computer graphics offer effective ways to produce a picture of the real-world as well as synthetic objects. It also provides the ability to create moving pictures using time frame and rendering scenes In the modern world, computer graphics have been applied widely such as user interface, simulation and animation, scientific visualization, and virtual reality. The fundamentals of computer graphics include mathematical structure, modeling, user interfaces, graphics software and hardware, viewing, rendering and image processing.[21][22]

### 2.2.1. Three.js framework

In the last few years, web browsers became more powerful and capable to display and deliver complex applications and graphics. Currently, most browsers have adopted WebGL, which allows to not only create 2D applications and graphics in the browser but also create good performing 3D applications, using the capabilities of the GPU. However, due to the complex inner details and shader languages, it is difficult to create scenes directly on WebGL.[23]

To solve this difficulty, Three.js was introduced in 2010. Three.js is an open-source JavaScript library that allows us to create and render 3D scenes directly in a web browser easily and quickly. An extensive API for this with a large set of functions, creating Three.js provides an easy-to-use API to create and manipulate 3D objects and scenes without having to know too much about WebGL or complex math formulas.[23]

### 2.2.2. View Frustum in Three.js

When looking into the 3D scenes, the objects seem to be in the $3D$ world. This concept is derived from view frustum. In computer graphics, the term view frustum refers to the pyramid-shaped volume in front of the viewer within the field of view. It is a part of $3D$ world space which is visible to the eye, or camera. Object outside the view frustum is not visible and therefore it is not necessary to render. The field of view angle, also known as angle of view in terms of optical sensors or instrument, is a solid angle of the camera with respect to the visible area. The field of view angle, $fov$, is defined as

$$\theta_{fov} = 2\arctan(h/2d) \tag{2.2}$$

where $h$ is height of field of view and $d$ is the distance from camera to the field of view.[24]

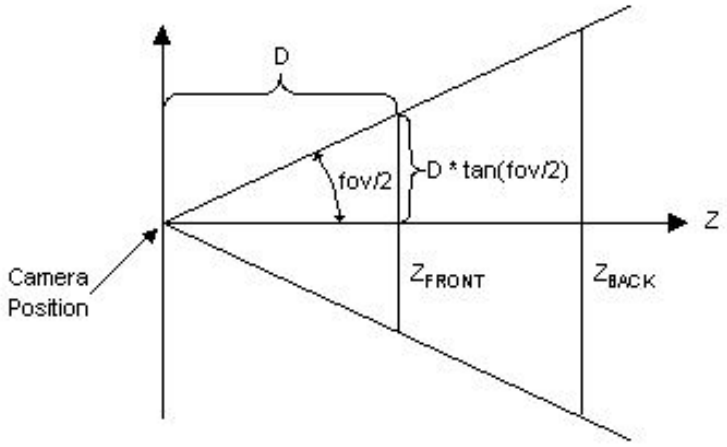


Figure 2.3.: Field of view(fov) [24]

In Three.js, the most common camera and one used in this project is the perspective camera. It gives a realistic 3D view where objects in far distance appear smaller than objects in close distance. The perspective camera defines a frustum that is based on 4 properties of view frustum.[25]

```
1 const camera = new THREE.PerspectiveCamera(fov, aspect, near, far)
```

Listing 2.1: Perspective Camera in Three.js

`near` defines where the front of the frustum starts and `far` defines where it ends. The field of view, `fov`, defines how tall the front and back of the frustum are by computing the correct height to get the specified field of view at near units from the camera. The `aspect` defines the camera frustum ratio of canvas width and height.

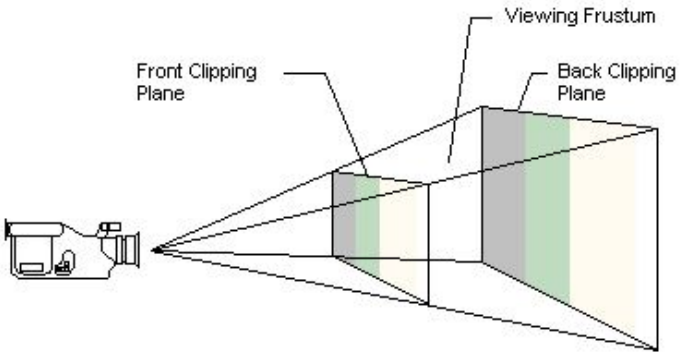


Figure 2.4.: Camera view [24]

### 2.2.3. 3D world to 2D screen coordinate system

Imagine having an object at the point $(0, 0, 0)$ in a 3D world. The coordinate of the two-dimensional world of this object is different, as only x and y plane exist. Also, depending on the angle of the camera, the position of the object changes in the image.

The position of a 3D world can be defined by a simple notation. The object is located at world coordinate $(x, y, z)$. Based on the equation 2.2, the frustum height $h$ can be derived as

$$h = 2d \cdot (\tan(\frac{\theta_{fov}}{2})) \tag{2.3}$$

where $d$ is a distance between camera and $z$-coordinate, and $\theta_{fov}$ is field of view angle. Using the result from frustum height and aspect ratio $r$, the frustum width $w$ can be calculated as

$$w = r \cdot h \tag{2.4}$$

In order to determine the position of the object in a 2D world, the device-independent coordinate is determined. Since the screen coordinates start at $(0, 0)$ and the maximum is at $(1, 1)$ and the Cartesian coordinates have $(0, 0)$ in the middle. Hence, the coordinates are translated into screen coordinates. [26]

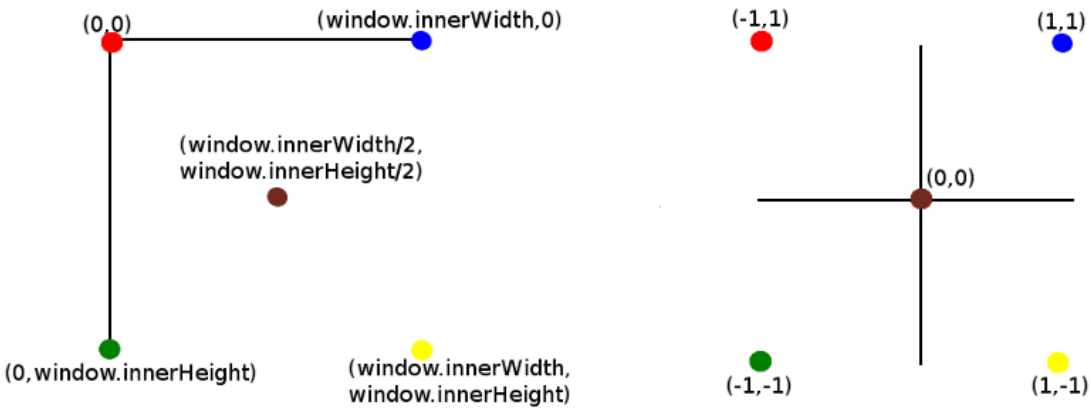


Figure 2.5.: The axis of screen and cartesian coordinates. Left figure represents screen coordinates and right figure represents cartesian coordinates [26]

The device-independent coordinates $x_d$ and $y_d$ are calculated as

$$x_d = 2 \cdot \frac{x}{w} \tag{2.5}$$

$$y_d = 2 \cdot \frac{y}{h} \tag{2.6}$$

where $x$ and $y$ is position of the object in the $3D$ world. Finally, the pixel $x_s$ and pixel $y_s$ in screen coordinates are calculated based on the screen width $w_s$ and height $w_h$.

$$x_s = \frac{w_s}{2} \cdot (x_d + 1) \tag{2.7}$$

$$y_s = \frac{h_s}{2}(1 - y_d) \tag{2.8}$$

This mathematical derivation can be programmed with javaScript as seen below.

```
function worldPosToScreenPixel(pos)
{
    pos.applyMatrix4(camera.matrixWorldInverse);
    const frustHeight = 2*Math.abs(pos.z) *Math.tan((camera.fov/2)*Math.PI/180);
    const frustWidth = camera.aspect * frustHeight;
    const dicx = 2*pos.x / frustWidth;
    const dicy = 2* (pos.y) / frustHeight;
    const pixelx = canvas.width/2 * (dicx + 1);
    const pixely = canvas.height/2 * (1 - dicy);

    return [pixelx, pixely];
};
```

Listing 2.2: World position to screen pixel

### 2.2.4. Blob Web API

When writing code for the Web, there are many web application programming interfaces, APIs, available. Web APIs are typically used with JavaScript and one of the most popular API is called Blob. Blob means a binary large object which is a collection of binary data stored as a single entity in the system. It is represented as a file-like object of immutable, raw data. Blob is used to hold multimedia objects such as images, videos, and sound. [27]

In JavaScript, the `HTMLCanvasElement.toBlob()` method creates a Blob object representing the image contained in the canvas; this file may be cached on the disk or

stored in memory at the discretion of the user agent. If type is not specified, the image type is .png and the created image is in a resolution of 96dpi. [27] The example code below takes the image in the <canvas> element whose ID is "canvas", obtains a copy of it as a PNG image, then appends a new <img> element to the document, whose source image is the one created using the canvas. [27]

```
var canvas = document.getElementById('canvas');
canvas.toBlob(function(blob) {
  var newImg = document.createElement('img'), url = URL.createObjectURL(blob);
  newImg.src = url;
  document.body.appendChild(newImg);
});
```

Listing 2.3: screenshot of canvas using Blob API

### 2.2.5. Skeletal animation

Skeletal animation is a technique used in computer animation in which character is represented with surface and a set of interconnected bones. The skeleton serves as an intuitive handle for the animation process, and it can be used to control the deformation of any object. Skeletal animation is typically used for animating characters such as Humans, mammals, insects, and even invertebrates. The skeletal animation process includes skeletal, rigging, key-frames and output. The skeletal represents a set of bones that can be hierarchically organized and applied to an object that has important joints. In the rigging process, the skeletal is placed in the exact location of the bones of the object. This way, the bones are controlled by the skeletal. The key-frame is a drawing that defines the starting and ending point of transition. The output phase shows the animation of the object. [28]

# 3 Requirements and Design

## 3.1. Requirements

This paper includes the following requirements:

1) Creation of graphics scene containing traffic situations with bikes

2) Automatic generation of synthetic images from a rendering scene

3) Classification of real life images using convolutional neural network trained with synthetic data

4) Evaluation of convolutional neural network performance

## 3.2. Synthetic data design

The synthetic data was generated using JavaScript with Three.js framework in rendering scenes. This work uses bicycles as observable objects in traffic situations as a starting point and can be adapted to other traffic objects in future research on machine learning. The simulation represents a real-looking traffic situation as it is applied to detect and classify bicycles in real life images. The scene includes two bicycles and one car driving on a crossroads and background with trees and houses.

It is also important to add variation to the scene to avoid over-fitting of the CNN model. Hence, different colors of bicycles and bikers, and different textures on the background and ground are added. Every bicycle and biker has five color variations that change after finishing a bidirectional run. At the same time, the position of tree and house changes

either left or right and the ground and background textures randomly change to one of 10 different textures. Lastly, the angle of the camera has five different variations, which show the side of the bike, slightly front side, and backside.

| Object | No. of variation |
|---|---|
| bicycle 1 | 5 |
| bicycle 2 | 5 |
| man on the bicycle 1 | 5 |
| woman on the bicycle 2 | 5 |
| background | 10 |
| ground | 10 |
| position of tree | 2 |
| position of house | 2 |
| camera angle | 5 |

Table 3.1.: Variation in rendering scene

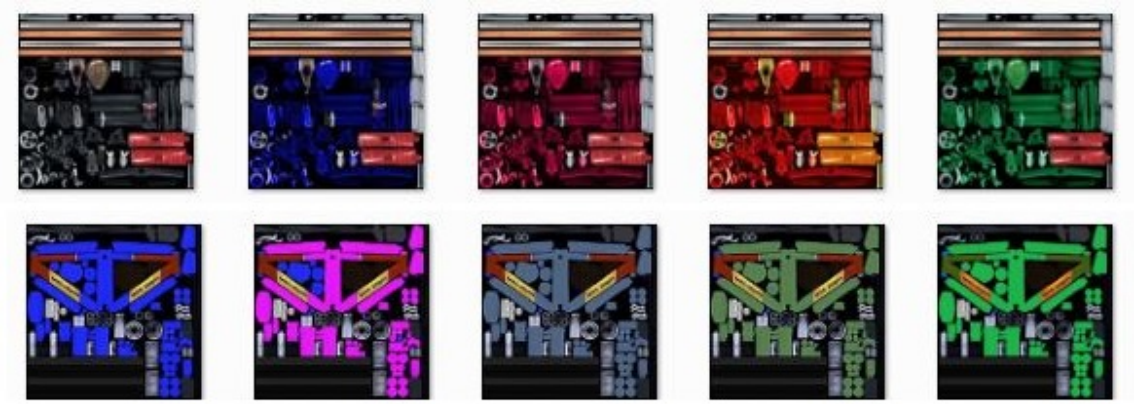

Figure 3.1.: Texture files for bicycle 1 and 2


Figure 3.2.: Texture files for background

## 3.3. Convolutional neural network setup

To start with image classification, the convolutional neural network train setup was configured based on work from the previous student, Kaloyan Dimitrov.[29] The training and evaluation of the convolutional neural network are implemented using TensorFlow. The network is trained for 30,000 steps with a learning rate of 0.0001 and a batch size of 32. Every convolutional layer is followed by a max-pooling layer. The kernel size for the convolution layer is set to 3. The max-pooling layers have a size and a stride of 2 and all layers use ReLU as an activation function. Based on the previous project, the best performing convolutional neural network was the network with 4 layers with dropout regularization turned on, namely the networks with 32/64/64/32 and 16/32/16/32 with an accuracy rate of 96 percent. The network ends with a 3-neuron output layer with a softmax activation function which predicts whether the image is of a bicycle going left, right or no bike (negative).[29]

Based on his project, the best performing model, the convolutional network with 4 layers with 32/64/64/32 filters with dropout, is chosen for this project. The output consists of bicycles going left, right and no bike, labeled as left, right and neg.

# 4 Implementation

## 4.1. Synthetic data generation

As mentioned in the chapter 3.1, the data was generated on Firefox web browser with Three.js framework using JavaScript.

### 4.1.1. Data creation

As a first step, the canvas was created for the size which is appropriate for training. The size of the canvas was chosen for a 500x500 pixel dimension. 3D object models such as a tree, a car, bikes, and human model can be downloaded in online websites. However, finding a model with a person on a bike is challenging, therefore it is modeled manually with a software called Blender. This will be discussed in the following chapter 4.1.2. The 3D object files in `.obj` and `.mtl` format are imported to the canvas via `objLoader` and `mtlLoader` from Three.js library. `.obj` file contains mesh of the model, and `.mtl` has material setting files.

For the background, there are several approaches to make the scene realistic. The first approach is to use an environment map, also known as a skybox, which creates a realistic background scene by using images in the cube. However, due to the nature of environment map, the position and the size of the objects has to be manually adjusted for every setting. As the automatic generation and variation of data is a core part of this project, the skybox approach is not realized in this project.

The second approach is to create a box and putting texture images on the box plane.

Although It creates a less realistic view compared to the skybox, it can generate a wide diversity of the scene.

### 4.1.2. Rigging with Blender

Finding a 3D object model that contains both a biker and bike is very challenging. The online websites usually offer only a bike model and creating own 3D model takes a lot of knowledge and techniques. Blender has an embedded function called Rigging, a skeleton animation technique, which can change the pose of a 3D object by putting a skeleton(armatures) around the object.

The original model of a human object is imported in Blender, then the armatures are added on the object that match the object skeleton. Scaling to fit into the model is also necessary for this step. Finally, once the armatures are set, the model can be rigged in any position.

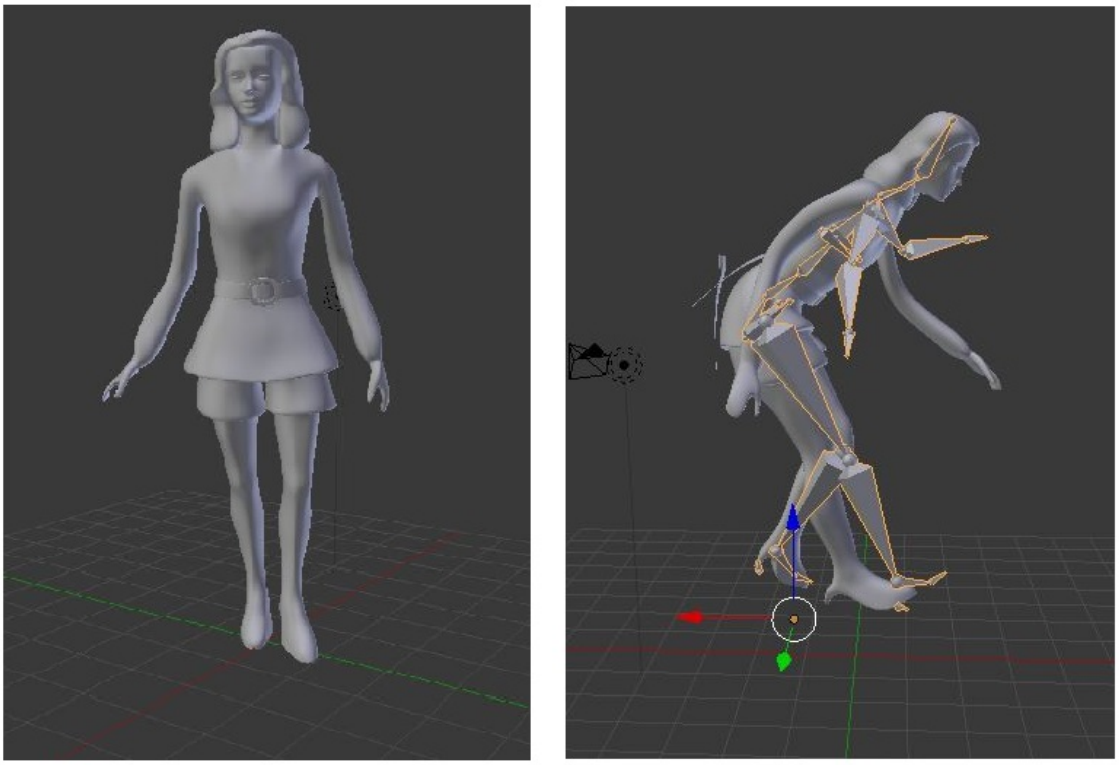

Figure 4.1.: 3D model before and after rigging in Blender

Once the rigging is finished, the model is exported in `.obj` format and imported to the

canvas in Three.js

### 4.1.3. Determination of object coordinates

Before extracting data, it is critical to determine whether the screenshot of the canvas contains a bike or no bike. In order to check this, it is necessary to transform the position of bikes into a 2D screen coordinate system. However, since `position.bike` only returns the middle point of the object, the program should ensure that the front and backside of the bike are in the screenshot images. Thus, maximum and minimum points are defined to ensure this.

To identify the maximum and minimum point of the object, a Three.js function `box3.THREE` is made to draw an invisible bounding box around the object. Using its method `box3.THREE`, the size of the object is defined and therefore the maximum and minimum point can be calculated based on the width and length of an object.

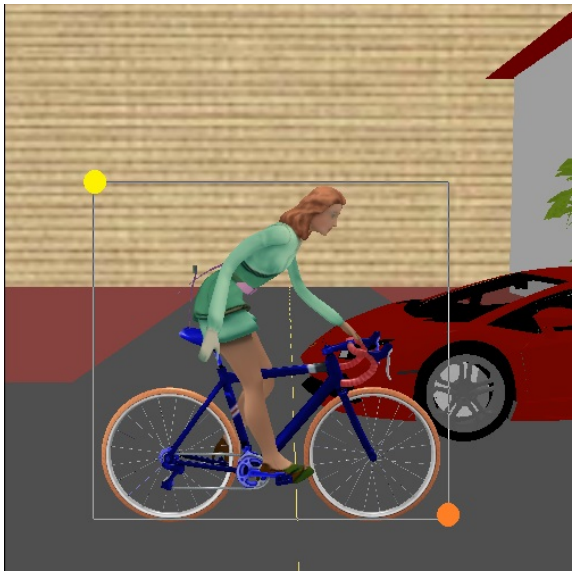

Figure 4.2.: Determination of points. Max. (yellow) and min. (orange) point from the side view. Note that the box and points are drawn only for this figure.

As the camera angle is moving with five different angles, the points have to cover and fit the bicycles in every angle without having too much free space. Hence, two points have different depths, putting maximum points at the front and minimum points on the back.

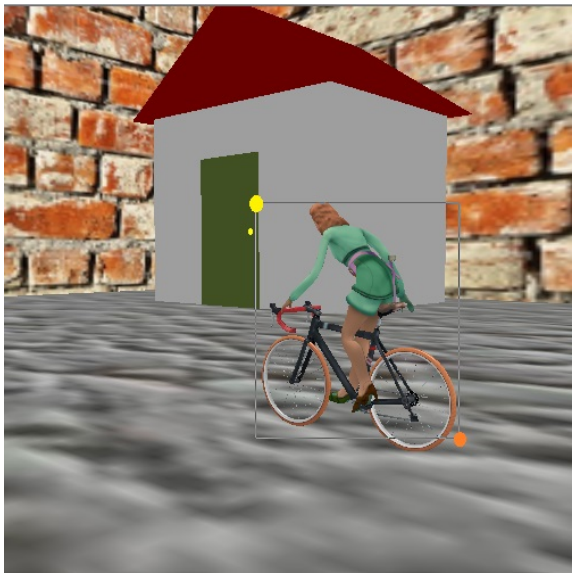

Figure 4.3.: Adjustment of maximum and minimum points. Max. (yellow) and min. (orange) point from side camera angle view. Note that the box and points are drawn only for this figure.

The following points are passed to the function `worldPosToScreenPixel(pos)` as described in 4.1.3, and translated into 2D screen coordinates. With these two screen coordinates, the program checks whether these points are within the range of the size of the canvas, between 0 to 500 in this case.

### 4.1.4. Automatic data extraction

After the minimum and maximum points are determined, they are passed to the function `setinterval`. This function runs every one-second interval, and every interval position of a bicycle in the 3D world is determined as well as minimum and the maximum points. Using blob function in `setinterval`, the screenshot of the canvas is created automatically every second. To extend this for future research, an XML file including maximum and minimum points can be also generated via Blob API.

### 4.1.5. Synthetic data augmentation

The 3D generated data has an unrealistic look as it contains sharp edges. This can cause a misleading of the neural network performance during the training phase since the real images rarely have sharp edges due to the low resolution. To avoid this, data is post-processed in order to have a more realistic look.

Matlab has a function `imnoise`, which adds noise to the images. Using this function, all images are added with Gaussian white noise. Furthermore, in order to speed up the training time, the size of pictures is adjusted from 500x500 to 176x176 pixels using Matlab `imresize` function. In addition to the noise, the Gaussian filter with $\sigma = 1.5$ is applied to the images in order to blur the images. `imgaussfilt`. Since some real images contain a tilted angle of bicycles, some parts of the synthetic images was tilted by 10-degree angle using a function `imrotate`.

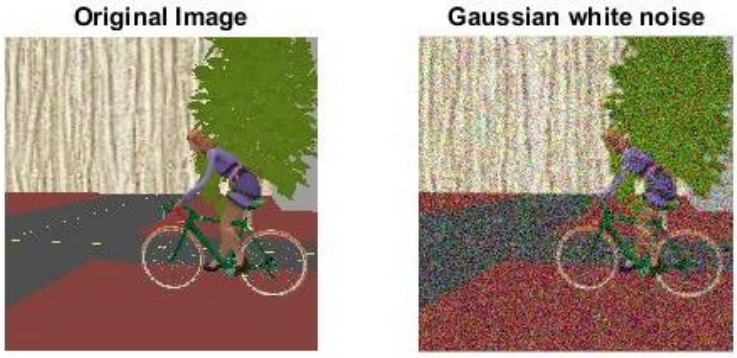


Figure 4.4.: Synthetic image before and after adding noise

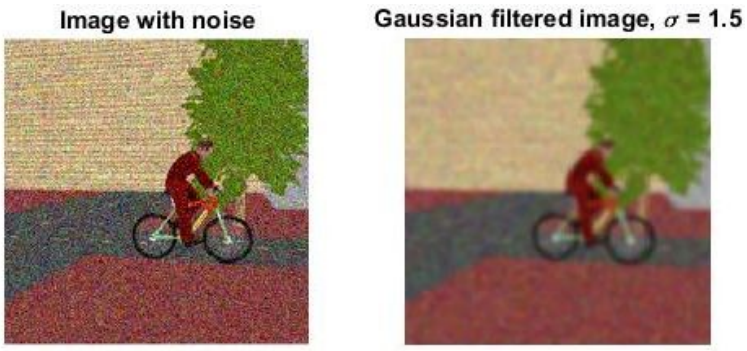


Figure 4.5.: Synthetic image before and after adding blurred effect


Figure 4.6.: Synthetic image before and after adding 10 degree angle

## 4.2. Convolutional neural network

### 4.2.1. Train CNN for classification

Using the setup from chapter 3.3, the training on the neural network was made with datasets created from Three.js scene. The datasets are divided into 3 types, noise images, noise and blurred images, and noise, blurred and tilted images. The datasets are created in a format of tfrecord file, a readable file format for Tensorflow. 70 percent of images are used for training and 30 percent are used for validation in order to check the training result. The network is trained until it reaches a stable validation accuracy rate. The intermediate step of the training is recorded using Tensorboard.

| Image type | left | right | neg |
|---|---|---|---|
| with noise effect | 1595 | 1579 | 1422 |
| with noise and blurred effect | 1468 | 1627 | 1666 |
| with noise, blurred, and tilted effect | 1468 | 1627 | 1666 |

Table 4.1.: Number and type of synthetic images

### 4.2.2. Evaluation process

The performance of the trained model is tested on two datasets, the dataset from Kaloyan Dimitrov [29] and dataset of real images with a low resolution.

| Source | right | left | neg |
|---|---|---|---|
| Kaloyan Dimitrov | 486 | 608 | 1154 |
| Jeonghyun Son | 218 | 185 | 331 |

Table 4.2.: Real image datasets for evaluation

The images from Kaloyan Dimitrov[29] seen in figure 4.7 have clear backgrounds and the bicycles are observable very clearly. The image has a full-size bike covering nearly an entire space of the picture.

Figure 4.7.: Real images used in previous work from Kaloyan Dimitrov [29]

On the other hand, the real images taken on the street contain a lot of noises in the background. The bike is located with a tilted angle, the size of the bike is small in the image and not clearly visible as seen in figure 4.8.

Figure 4.8.: Real images taken on the street

The evaluation process is divided into several steps. First, synthetic data with noise effect is trained on a network. The test is made on real image datasets which are divided into left, right, and no bike images using the corresponding network model. The total test accuracy is calculated by averaging each test accuracy on left, right, and no bike datasets. The process is repeated for all training datasets.

# 5 Results

## 5.1. Data result

The data was created successfully with the direction label defined on the file name. The results of data extraction using Three.js are shown in figure 5.1.

Figure 5.1.: Synthetic images generated by Three.js

Some errors were observed in the course of data extraction. Sometimes the web browser becomes frozen during the image extraction process and the scene does not render, it thus creates multiple identical images that could potentially lead to overfitting of the network. Also, due to the delay in the browser, a few negative labeled images contain a part of bicycles. Moreover, when the browser becomes slower due to too much rendering in the scene, the checking bike function has stopped and created the pictures with the bike with no bike label. These images were removed manually.

As an extension to this work, the xml file for bike location annotation generated by Three.js represents a promising result as seen in 5.2. This xml file contains bounding box information of a bicycle and can be generated with images automatically. This can

be later used for future research on object detection with a convolutional neural network trained on synthetic data.

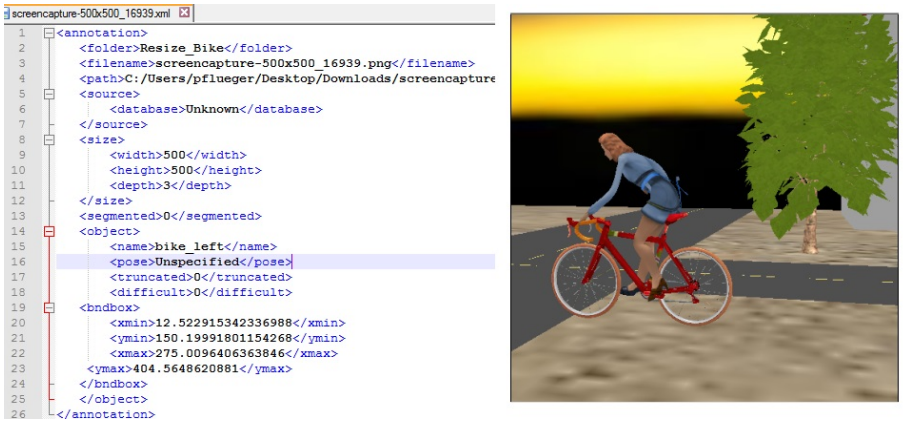


Figure 5.2.: Image with annotation file

## 5.2. Training result

The training was made with synthetic data for 31807 steps and the best validation accuracy resulted in 98 percent. The training takes about 15 minutes using a computer with RTX 2080 ti GPU. In figure 5.3, the validation accuracy rate and the loss of the validation and training data are shown. From training steps 0 to 11000, the noise images are fed to the network. At training step 11000, the process is aborted and the model is additionally fed with new blurred images. Lastly, at step 22462, the tilted images are fed to the network.

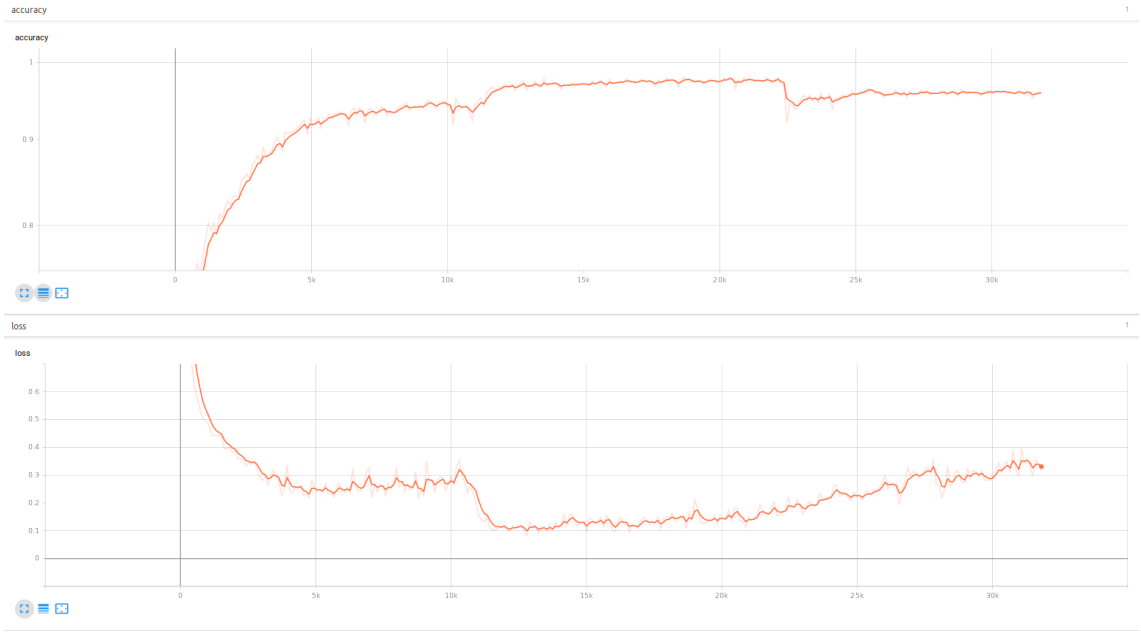


Figure 5.3.: Validation accuracy and loss graph from tensorboard

With first dataset, it reached at 94% validation accuracy. After second dataset was added, the accuracy has risen to 97% and the loss dropped at 0.1. When the tilted dataset was added, the accuracy decreased to 94%.

## 5.3. Classification result

The trained models are tested on real image datasets and test accuracy is measured based on left, right, and neg(no bike) accuracy. The result of the first model trained only with noise image is shown in figure 5.1.

| Dataset | left | right | neg | Average test accuracy |
|---|---|---|---|---|
| Kaloyan Dimitrov | 86.67 % | 18.1% | 33.01% | 45.92% |
| Jeonghyun Son | 47.31% | 55.15% | 4.24% | 33.56% |

Table 5.1.: Test accuracy on first model

As expected, the accuracy of data from Kaloyan Dimitrov has a slightly better than the other data. However, the performance on both datasets remain very poor, under 50%.

Additionally, the previous model is trained with noise and blurred images. The result of this model is shown in figure 5.2. It shows a slight improvement in both real image datasets. Interestingly, the accuracy on the second dataset has improved on left bike data, with 94.14% accuracy. However, the model can barely recognize whether the image contains a bike or no bike.

| Dataset | left | right | neg | Average test accuracy |
|---|---|---|---|---|
| Kaloyan Dimitrov | 53.94% | 68.51% | 46.7% | 56.38% |
| Jeonghyun Son | 94.14% | 29.09% | 4.24% | 42.49% |

Table 5.2.: Test accuracy on second model

Lastly, the previous model is trained with all datasets, noise, blurred and tilted images. In figure 5.3, the average test accuracy is increased on the first real image dataset, while it drops on the second dataset. The model has a major difficulty classifying no bike images, as it appears to be as low as 7.87%.

| Dataset | left | right | neg | Average test accuracy |
|---|---|---|---|---|
| Kaloyan Dimitrov | 62.33% | 67.69% | 46.27% | 58.75% |
| Jeonghyun Son | 86.34% | 24.84% | 7.87% | 39.68% |

Table 5.3.: Test accuracy on final model

Overall, the performance of the network is rather poor, with the best test accuracy of 58% on the first dataset and 42.49% on the second dataset. The possible reason for these accuracy is that synthetic data has a lack of variation and realistic settings, as it carries no artifact and very homogeneous training datasets. Furthermore, the poor accuracy on neg images(no bike) could be due to a lack of background objects in the neg images.

# 6 Conclusion and Future work

In this thesis, the convolutional network and machine learning were explained and trained with synthetically generated image data. The scene with bicycles in traffic situations is created with Three.js framework. Using Blob API, the screenshots of the scene were created with labels. Due to the slow web browser performance, the screenshot was produced correctly in average 80% of the time. Around 14,111 synthetic images were created and 70% was used for training and 30% was for validation. In addition, the approach to automatic creation of an annotation file with a bounding box of a bicycle was implemented as a contribution to future research on object detection. The synthetic data was post-processed by adding noise, blurred and tilted effect.

The CNN configurations were investigated based on the previous work from Kaloyan Dimitrov.[29] The network is fed with three types of synthetic data, images with noise, blurred and tilted effect, and each dataset was trained on the CNN model respectively. After each training step, the model was tested on two real image datasets, and evaluated test accuracy on left, right and no bike images.

The result of classification shows that using synthetic data for training CNN has a limitation in performance. The test accuracy on both images was below 60%, and has difficulties to classify no bike images on certain datasets. Especially, the model performed poorly on images with a person or objects. The possible reason can be that synthetic data includes no realistic artefact effects, and has very homogeneous backgrounds.

Further research can be made to improve the convolutional network by adding more realistic artefacts on synthetic data, such as shades and light adjustment. Moreover, creating

a background with a more realistic view by using an environment map can be implemented to improve test accuracy.

# List of Abbreviations

| | |
|---|---|
| 2D | Two-dimensional |
| 3D | Three-dimensional |
| API | Application programming interface |
| CNN | Convolutional neural network |
| GPU | Graphics processing unit |
| ReLU | Rectified linear unit |

# References


[1] The Royal Society, *Machine learning: the power and promise of computers that learn by example*, The Royal Society, (2017), ISBN 978 1 78252 259 1

[2] C. Eggert, A. Winschel, R. Lienhart, *On the Benefit of Synthetic Data for Company Logo Detection*, In Proceedings of the 23rd ACM international conference on Multimedia, 23 (2015), 1283-1286

[3] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *Gradient-Based Learning Applied to Document Recognition*, Intelligent Signal Processing, (2001), 306–351

[4] J. Marin, D. Vazquez, D. Geronimo, A. M. Lopez, *Learning Appearance in Virtual Scenarios for Pedestrian Detection*, Conference on Computer Vision and Pattern Recognition, (2010), 137–144.

[5] S. Ghosh, P. Amon, A. Hutter, A. Kaup, *Pedestrian Counting Using Deep Models Trained on Synthetically Generated Images*, International Conference on Computer Vision Theory and Applications(VISAPP), (2017), 86-97

[6] A. Rozantsev, V. Lepetit and P. Fua, *On rendering synthetic images for training an object detector*, Computer Vision and Image Understanding, 137 (2015)

[7] A. Agrawal, *Application of Machine Learning to Computer Graphics*, IEEE Computer Graphics and Applications, 38 (2018), 93-96

[8] T. Mitchell , *Machine Learning*, McGraw-Hill Education, (1997), ISBN 978 0070428072

[9] N,Deusebrith, A. Faisal, C.Ong, *Mathematics for Machine Learning*, Cambridge University Press, (2019), ISBN 978 1108470049

[10] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, (2014), ISBN 978 1107057135

[11] M. Mohammed, M. Badruddin Khan, E. Bashier, *Machine Learning: Algorithms and Applications*, CRC Press, (2016), ISBN 978 1315354415

[12] J. Mueller and L.Massaron, *Machine Learning For Dummies*, John Wiley & Sons, (2016), ISBN 978 1119245513

[13] M. Nielsen, *Neural Networks and Deep Learning*, [Online]. Available: http://neuralnetworksanddeeplearning.com. [Accessed: 02- Oct- 2019]

[14] J. Ahire, *The Artificial Neural Networks handbook*, [Online]. Available: https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4. [Accessed: 11- Sep- 2019]

[15] M. Mishra, *Convolutional Neural Networks, Explained*, [Online]. Available: https://www.datascience.com/blog/convolutional-neural-network. [Accessed: 11- Sep- 2019]

[16] Google, *ML Practicum: Image Classification*, [Online]. Available: https://developers.google.com/machine-learning/practica/image-classification/. [Accessed: 09- Sep- 2019]

[17] C. Wang and Y. Xi, *Convolutional Neural Network for Image Classification*, Johns Hopkins University Baltimore, MD, vol. 21218

[18] J. Brownlee, *Overfitting and Underfitting With Machine Learning Algorithms*, [Online]. Available: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/. [Accessed: 09- Sep- 2019].

[19] J. Brownlee,*What is the Difference Between Test and Validation Datasets?*. [Online]. Available: https://machinelearningmastery.com/difference-test-validation-datasets/. [Accessed: 05- Oct- 2019].

[20] DeepDetect, *Training with Data Augmentation*, [Online]. Available: https://www.deepdetect.com/tutorials/data-augmentation/. [Accessed: 09- Sep- 2019].

[21] LMU Computer Science, *Computer Graphics Basics*, [Online]. Available: https://cs.lmu.edu/ ray/notes/graphicsintro/. [Accessed: 09- Sep- 2019].

[22] D. Eck, *Introduction to Computer Graphics*, Hobart and William Smith Colleges, (2018), Product No. 23477605

[23] J. Dirksen, *Three.js Essentials*, Packt Publishing , (2014), ISBN 978 1783980864

[24] Microsoft Corporation, *The Viewing Frustum*, [Online]. Available: http://doc.51windows.net/Directx9_SDK/graphics/programmingguide/fixedfunction/viewportsclipping/viewingfrustum.htm. [Accessed: 11- Sep- 2019]

[25] Three.js, *Three.js projecting mouse clicks to a 3D scene - how to do it and how it works*, [Online]. Available: https://barkofthebyte.azurewebsites.net/post/2014/05/05/three-js-projecting-mouse-clicks-to-a-3d-//scene-how-to-do-it-and-how-it-works. [Accessed: 02- Oct- 2019]

[26] acarlon, *The Artificial Neural Networks handbook*, [Online]. Available: https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4. [Accessed: 11- Sep- 2019]

[27] MDN web docs, *HTMLCanvasElement.toBlob()*, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/toBlob. [Accessed: 11- Sep- 2019]

[28] Ahmedabad, *What is Skeletal Animation Technique?*, [Online]. Available: https://www.slideshare.net/animationcoursesahmedabad/what-is-skeletal-animation-technique. [Accessed: 09- Sep- 2019]

[29] 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)

# Appendices

# A

# Description of programs on a WD hard drive

WD hard drive contains all the files and programs used in this thesis. In the drive, there are two main folders, namely bike_simulation, and classification.

The bike_simulation folder contains all 3D objects and texture images for Three.js scene. bike_1, bike_2, bike_man, bike_woman, car, and tree folder contains 3D objects including 3D object texture image files, mtl file for definitions of materials and obj file for geometry definition. three.js-master folders contain all Three.js modules, and the texture folder contains background and ground texture images. The file `test.js` is JavaScript file and runs with `index.html` file. The JavaScript code runs on the Firefox web browser and needs to be run on the local server as it uses external sources.

To run a local server with Python version 3.x:

```
python -m http.server
```

This serve files from the current directory at localhost under port 8000. In Firefox broswer, type the following command on the address bar:

```
http://localhost:8000/
```

In order to generate images with annotation file(xml), comment lines from 588 to 600, and uncomment lines from 602 to 617 in `test.js`

The classification folder contains images and convolutional neural network models. Images folder contains blurred, noise, tilted synthetic images for training, and real images for testing. Matlab folder has scripts for post-processing images. test_dataset has real image files in a tfrecords format. train_dataset contains synthetic image files in tfrecords format.

The trained_model has convolutional neural network models as results of training.

The file `create_dataset.py` generates a labeled dataset in tfrecords format from synthetic images. It creates eval.tfrecords and train.tfrecords files. eval.records is used for validation, and train.tfrecords is for training. `create_testdataset.py` creates a test dataset in a tfrecords format. `train.py` can train the network or can be used for testing.

To run tensorboard for visualizing training result:

```
python -m tensorboard.main --logdir=eval/
```

For acquiring the program, please address to Professor Jünemann from HAW Hamburg for the WD hard drive.

# 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 Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Ort, Datum

Unterschrift