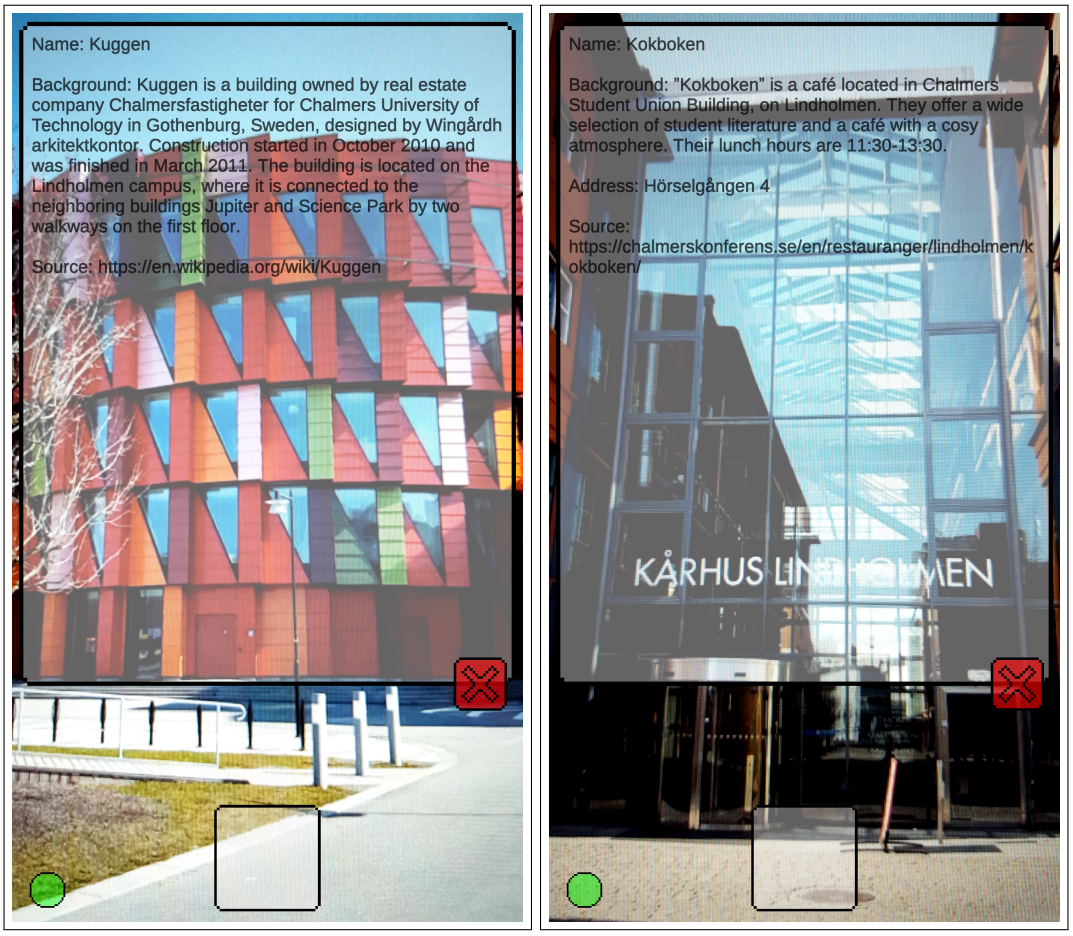
CHALMERS
UNIVERSITY OF TECHNOLOGY



Representation of information of Chalmers buildings with machine learning

Bachelor's thesis in Computer Science and Engineering

ERIK TRAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

BACHELOR'S THESIS 2019:22

Representation of information of Chalmers buildings with machine learning

ERIK TRAN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Representation of information of Chalmers buildings
ERIK TRAN

Supervisor: KARL BÄCKSTRÖM
Examiner: JONAS DUREGÅRD

Bachelor's Thesis 2019:22
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Abstract

This report is a study about machine learning, and is also about a development of a prototype for a mobile application that are able to identify a few chosen buildings with machine learning, retrieve and visualize information about the buildings. The purpose of the mobile application is to improve the experience of a journey through a city, which targets those who are interested in buildings. The target group in this project is tourists and architects. This report consists of brief description of tools and libraries that were used in the development of this project. The main part of this report is about the creation process of an image classifier through four main steps which are collecting training data, conversion and refining data, building the model structure and lastly training and validating data. During this project, an application will be created and the design of the application will be described and visualized detailed. The design consist communication between multiple clients and a server through socket connections. Final result is that the application achieved some parts of the initial aim of this project, but the application still could be improved.

Sammanfattning

Denna rapport är en studie om maskininlärning och handlar också om utvecklandet av en prototyp för en mobilapplikation som kan identifiera några utvalda byggnader med maskininlärning, hämta och visualisera information om byggnaderna. Syftet med mobilapplikationen är att förbättra upplevelsen av en resa genom en stad, som riktar sig till dem som är intresserade av byggnader. Målgruppen i detta projekt är turister och arkitekter. Denna rapport innehåller en kort beskrivning av programmer och bibliotek som användes vid utveckling av detta projekt. Huvuddelen av denna rapport handlar om skapandet av en bildklassificerare genom fyra huvudsteg. Dessa huvudstegen är att samla in träningsdata, konvertering av filmer och förfinandet av bilder, byggandet av modellstrukturen och slutligen träning och validering av datan. Under detta projekt kommer en applikation att skapas och applikationsdesignen kommer att beskrivas och visualiseras detaljerat. Applikationsdesignen innehåller kommunikation mellan flera klienter och en server via socketanslutningar. Slutresultatet är att applikationen uppnådde några delar av det ursprungliga syftet med projektet, men applikationen kunde fortfarande förbättras.

Contents

Abbreviations

AI = Artificial Intelligence
API = Application Program Interface
AR = Augmented Reality
BGR = Blue Green Red
CNN = Convolutional Neural Network
ML = Machine Learning
MLP = Multilayer Perceptron
ReLU = Rectified Linear Unit
RGB = Red Green Blue
TCP = Transmission Control Protocol

1

Introduction

Artificial intelligence (AI) is a type of intelligence demonstrated by machines. One of the goals by creating and using theses machines in industries is to reduce human effort, give accurate and faster results. The traditional goals of AI research is for instance reasoning, learning, planning, communication and the ability to move and manipulate objects [1].

The objective of the research in this project is machine learning (ML), which requires a lot of data. By the easily accessible image data that could be obtained, and the potential of AI, this project will be researching about ML, and attempting to create a machine that is able identify a few chosen buildings.

1.1 Background

There exists tourists and architects who have an interest in buildings, from time to time they will discover interesting buildings that are new to them. In some cases, travelers may need to identify the buildings they have discovered to confirm that they are in the correct location.

Developing a mobile application that identifies buildings, receives information about the identified buildings, could improve the experience of a journey through a city.

1.2 Aim

The aim of this project is to attempt to develop a prototype for a mobile application which through the function described in section 1.1, gives the target group a more interesting experience during a journey through a city. The identification of buildings should be processed quick enough with minimal process delay for a better user experience. The application should mainly be able to retrieve information of the identified building, and display the information directly on to the existing camera view of the mobile application.

1.3 Delimitations

This application will only be created for Android devices, because the creator of this application has access to a Android mobile phone only. Additionally this application will be built only for Gothenburg right now because the creator lives in Gothenburg, and also because the creator requires easily accessible image data. Since Gothenburg is a large city and consists of a lot of buildings, the creator decided to create this application specifically for new students at Chalmers, because the requirement for building this application will be reduced. Requirement is to collect image data of some buildings from Chalmers.

2

Theory

This chapter will cover some key concepts used in this project. Tools and libraries that were used in the development of this project will be mentioned in chapter 3 and therefore will be explained in this chapter. The neural network that were applied for this project, convolutional neural network (CNN) and some of its important core layers will also be explained.

2.1 Machine Learning

Machine learning is the study of algorithms and data that the computer system uses to improve its learning about a specific task [2]. According to Expert System, a company that develops cognitive computing and text analytics software based on artificial intelligence algorithms [3], machine learning was given the following definition [4]:

> "Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.
>
> The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly."

Some real-life examples of advantages by using machine learning is image recognition, speech recognition, medical diagnosis, statistical arbitrage, learning associations, classification, prediction, extraction, regression and financial services [5].

2.2 Convolutional Neural Network

Convolutional neural network (CNN) is a type of deep neural network that commonly are used for image recognition [6]. CNN has also been utilized on other applications, some examples are video analysis, natural language processing, drug discovery and games such as Checkers and Go.

CNN is a type of artificial neural network that consists an input layer, hidden layers and an output layer. The hidden layers includes one or more convolutional layers. Additional useful layers could be pooling, dropout, flatten and dense layers. These layers has its own unique operation and in CNN are mainly used to increase the efficiency for image processing, which results in a more effective system [7]. These layers will be described in subsections 2.2.1-2.2.5.

Mainly, there are four steps in a CNN. First step is convolution, the neural network tries to label input signals by referring to its learning from the past. When a CNN tries to recognize an image, the convolution layer or filter will try to detect patterns such as edges, shapes, textures and objects. Adding another convolutional layer into the neural network will make the neural network able to detect even more shorter edges, and in the end the neural network will be able to classify objects such as dogs, cats or humans depending on what the network has been taught. Second step is subsampling, reducing sensitivity of the image to noise and variations which can be done by reducing size and color contrast of the image, resulting to a lower resolution image. Third step is activation, controls the signal flows from one layer to next. The most common activation function being used in a CNN is ReLU (Rectified Linear Unit), this activation function will be explained in the next section 2.3. Last step is fully connectedness, neurons of preceding layers are connected to every neuron in subsequent layers, which covers all possible pathways from input to output [8].

2.2.1 Conv2D

Conv2D is a python class constructor from the library Keras that constructs a convolutional layer that creates a convolution kernel that is convolved with the input over a single spatial dimension to produces tensors of outputs which could represent edges, shapes, textures or objects. Conv2D constructor has several main arguments that changes the properties of the convolutional layer [9].

- filters - Integer, the number of output filters or neurons in the convolution.
- kernel_size - Tuple, specifying the height and width of the 2D convolution window.
- activation - String, the activation function to use.
- input_shape - Tuple, used on the first layer only in a model, specifying the input shape in width and height, and the color model. For instance the tuple (64, 64, 3) gives input shape with the width and height being 64 pixels with RGB color

specified.

2.2.2 Maxpooling2D

Maxpooling2D is a python class constructor from the library Keras that constructs max-pooling layer, its function is to reduce the amount of parameters and computation in the neural network, which may prevent overfitting [10].

An example for operating in Maxpooling2D with the tuple (3, 3) as its pool size is shown below in figure 2.1 and figure 2.2.

20	54	97	34	178	60	40	30	35
13	25	10	55	130	50	151	110	28
61	20	10	15	120	50	150	120	28
88	56	10	11	50	50	150	161	21
65	53	10	11	31	50	50	60	21
81	27	10	17	58	13	52	69	24
30	61	10	17	52	10	151	67	24
23	28	10	13	37	10	152	66	25
20	21	10	11	25	20	55	61	25

Figure 2.1: The input values before operating in Maxpooling2D layer.

97	178	151
88	58	161
61	52	152

Figure 2.2: The values outputted after operating in Maxpooling2D layer.

Max-pooling takes the maximum value from each cluster of neurons with pool size (3, 3).

2.2.3 Dropout

Dropout is a python class constructor from the library Keras that constructs a dropout layer, which may help prevent overfitting. Its function will be described in the example shown in figure 2.3 [11].

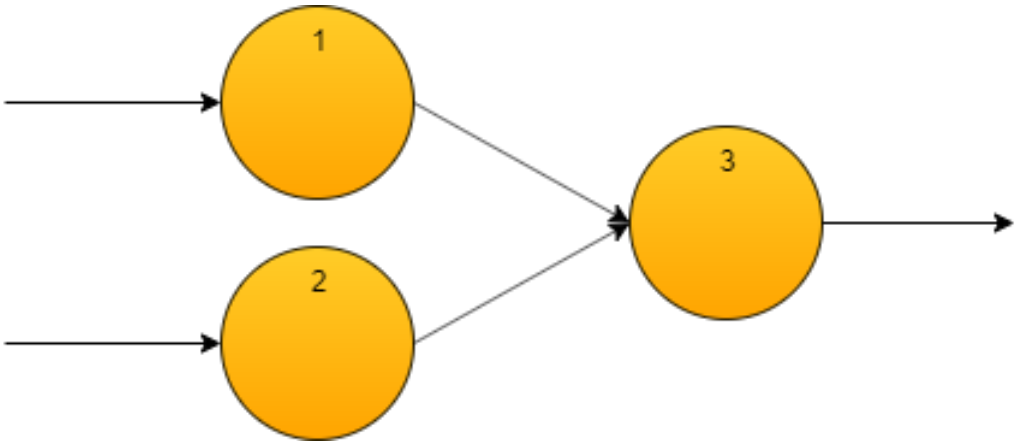


Figure 2.3: An example of subnetwork in a neural network.

Given that the output of first neuron 1 with 90% on correct prediction, 10% on incorrect prediction and the output of second neuron 2 with random predictions, operating in dropout layer with the given values, neuron 3 will output the sum of <multiplication of 1 and output of first neuron 1> and <multiplication of 0 and output of second neuron 2>. Basically, the output of neuron 3 outputs the output of neuron 1, which is redundant work.

Dropout removes neurons with random predictions randomly. It removes either neuron 1 or neuron 2. Gradually, the neural network will give better predictions and less random predictions.

2.2.4 Flatten

Flatten is a python class constructor from the library Keras that constructs a flatten layer, this layer will flatten out any multi-dimensional input to 1D output. It is necessary to operate in flatten layer to prepare operation in the fully connected layer or the dense layer [11].

1	5	9	10
4	9	10	3
20	2	20	1

Figure 2.4: Example of 2D input with the row of size 3 and column of size 4 before operating in Flatten layer.

1	5	9	10	4	9	10	3	20	2	20	1

Figure 2.5: The output in 1D with row of size 1 and column of size 12 after operating in Flatten layer.

2.2.5 Dense

Dense is a python class constructor from the library Keras that constructs dense layer, this layer is a regular neural network layer. Dense constructor has several arguments that changes the properties of its layer. One of the arguments is units which is an integer that represents the number of outputs. For instance if a CNN is trained to classify two unique elements, the value of units should be 2. Another argument is activation which is a string that represents the activation function to be used in the layer [11]. There will be some activation functions presented in the next two sections.

2.3 ReLU

ReLU (Rectified Linear Unit) is a non-linear activation function which has the advantage of not having any backpropagation errors which gives a better gradient propagation and an efficient computation. That is the reason why most of the deep learning applications uses ReLU instead of other logistic activation functions [12].

ReLU is defined according to the equation (2.1),

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.1}$$

2.4 Softmax

Softmax is a activation function that maps outputs to a range between 0 and 1, and the total sum of the outputs is 1, and therefore the outputs of Softmax represents the probabilities of each output. Softmax is usually used in the last layer of any multi-classification CNN model [13].

2.5 Augmented Reality

Augmented reality is a different reality overlayed onto the real-world environment. The objects that reside in the real world are enhanced with computer-generated information, for example images, text or sound [14][15]. The objects from the real world could be detected by using software and algorithms in two stages. The first stage is to detect interest points, fiducial markers or optical flow in the camera images. The second stage is to restore a coordinate system in the real-world environment from the data that were obtained in the first stage [16].

2.6 Unity

Unity is a game engine developed by Unity Technologies that allows the user to create two-dimensional (2D), three-dimensional (3D), virtual reality (VR) and AR applications [17]. Unity is specially popular for creating mobile games within many different platforms such as Android and iOS. Unity also allows the user to create applications for many other platforms such as Windows, Mac, Linux, Nintendo 3DS and PlayStation 4 [18]. To create 3D games withing Unity requires support by external software which produces 3D graphics needed for the game. Unity offers support to the user with software such as DirectX, Metal and OpenGL Core [19].

2.7 JupyterLab

JupyterLab is a web-based user interface that enables you to work with documents and activites. Some examples of those activities could be Jupyter notebooks, text editor, terminals and custom components. Both documents and activities works with each other in a integrated way which enables new workflows for interactive computing [20].

2.7.1 Jupyter Notebook

Jupyter notebooks is a JupyterLab activity, documents that combine runnable code in the programming language Python with Markdown text, LaTeX equations, images and interactive visualizations outputs [21].

Jupyter notebooks mainly contains four different types of cells [22].

- Code cell - a cell which contains of statements in Python, executable code.
- Markdown cell - a cell which contains of Markdown text that after running the cell will render the text to the output.
- Raw cell - a cell which contains of plain text or LaTeX equations. If LaTeX is typed in the raw cell, it will be rendered after the calculation.
- Output cell - a cell that could display different type of outputs such as normal output text from a "print" statement, Markdown text, images and LaTeX equations.

Any of these cell could be run in any desired order, which shows the high flexibility of Jupyter notebooks.

2.8 TensorFlow

TensorFlow is a library for dataflow and differentiable programming across a range of tasks. Mainly, it is used for creating machine learning applications such as different types of artificial neural networks. Google and several other tech companies uses TensorFlow for researching and production. TensorFlow computations are expressed as stateful dataflow graphs. TensorFlow is written in Python, C++ and CUDA, and since it is written in Python, it then also makes it available for Jupyter notebooks. Platforms that could utilize the power of TensorFlow are Linux, macOS, Windows, JavaScript and Android [23].

2.9 Keras

Keras is a high-level neural networks API, a deep learning library that is an abstraction level over TensorFlow, which makes the use of TensorFlow easier, which enables faster experimentation compared with TensorFlow. Keras supports convolutional neural networks. Keras is user friendliness, have modularity and easy extensibility [24].

2.10 NumPy

NumPy (Numerical Python) is a python library that enables handling of large multidimensional data structures that could be represented as matrices and vectors. It contains a large collection of mathematical functions that are used in order to speedily operate these data structures [25].

2.11 OpenCV

OpenCV (Open Source Computer Vision Library) is a highly optimized cross-platform library that is written in C++, Python and Java with focus on real-time applications. OpenCV contains a large collection of functions that are mainly aimed at real-time computer vision and machine learning softwares. OpenCV interfaces support Windows, MacOS, Linux, Android and iOS [26][27].

2.12 Matplotlib

Matplotlib is a 2D plotting python library which produces different types of images with just a few lines of codes. Some examples of types of images are plots, histograms, power spectra and bar charts [28]. This library contains functions, and one of the functions could read an image from a file and store it into an NumPy array, and this NumPy array could be plotted at anytime [29].

2.13 Overfitting

Overfitting occurs when a neural network model is too powerful for the amount of training data. An overfitted model will affect its performance negatively on the validation data. An overfitted CNN model will usually classify the validation data incorrectly. Basically, it means that an overfitted model is bad at generalization [30]. Training data is the data that will be used to train the algorithm of the model. Validation data is the data that will be used for a prediction test of the model.

2.14 Image Augmentation

In many cases, overfitting CNN models will classify validation data incorrectly because it lacks training data of images. Therefore image augmentation is one of the solutions to prevent overfitting. Image augmentation will create training images through many different ways [31]. ImageDataGenerator is a python class constructor from the library Keras. ImageDataGenerator is used to create image augmentations. In an ImageDataGenerator class there are some properties that creates augmented images that differentiates from the original images. Some of the properties of an image that could be changed are rotation degree range, width shift range, height shift range, brightness range, shear range and zoom range. For example, rotating the image might be useful because it is possible that some images taken for validation image data are tilted. Zoom range might be useful too when some of the validation image data is too close compared to other validation image data. Brightness range can be put to good use when some of the validation image data is too bright [32].

It is good to use image augmentation when your image data is not uniquely enough. For example getting the training image data from frames of a recorded video makes each image not uniquely enough, or each image has the same brightness and same rotation range. Training models with image augmentation makes the training image data much more unique by modifying properties of images.

2.15 Accuracy and Loss

Accuracy in a CNN model is the fraction of predictions a classification model got right. The definition of accuracy is the number of correct predictions divided by total number of predictions [33].

Loss in a CNN model is a number that indicates how bad the prediction was on a single example on the model. Total loss of a CNN model is the sum of errors made for each prediction in training or validation set. If the model's prediction is not perfect, then the loss is greater than 0 [34]. For example, a CNN model that classifies cars and non-cars, and a prediction says the probability that it is a car is 70% and the probability that it is not a car is 30%, then the loss is 0.3.

fit_generator

fit_generator is a python function from the library Keras that can be used to train a CNN model. The function takes in some arguments. Some of the arguments are an image data generator for image augmentation for a training set, a validation set, an integer for the total number of steps per epoch, an integer for the total number of epochs. The function returns a history object that contains training accuracy and loss and validation accuracy and loss for each epoch [35].

3

Method

This chapter will cover the most important part of this project, the creation process of an image classifier in section 3.1. Further in this chapter, the application design that uses the created image classifier CNN model will be covered in section 3.2.

3.1 Image Classifier

An image classifier is a program that can classify images. For instance it should be able to classify between different buildings in this case. The classification of images will be done through a CNN model [36]. In this case, the process of creating a image classifier consists of four main steps. First step is data collection which will be covered in subsection 3.1.1. Second step is conversion of videos to images and refining images which will be covered in subsection 3.1.2. Third step will cover the CNN model structure in subsection 3.1.3. Last step, the CNN model will be trained and validated, covered in next chapter about the result, chapter 4, section 4.2.

3.1.1 Data Collection

Collection of data will be done by recording buildings one by one with a Android mobile phone. The recorded video should cover as much as possible only the building in all possible directions while avoiding other objects. The distance between the recorder and the building should be the same as possible from start to end of recording, the reason is because being too close to the building may exclude image data of corners and edges of the building and too far away from the building may make the building smaller compared to the bigger picture of the building in collection of image data. The weather condition during filming are that it should not be raining or snowing, because this application is not intended to be built for those weather conditions, nor should it be too dark, else the training data will be bad. In order to make the application work during rainy weather or snowy weather, image data of those buildings during those weather will be required.

3.1.2 Conversion and Refining

After collecting data by filming, conversion and refining will be the next step. Conversion and refining will be done by running a few lines of python code in a Jupyter notebook. First thing to do is to read every single image frame from every recorded building. Converting a video to image frames will be done by running a few lines of python code with OpenCV, NumPy and Matplotlib imported.

```
1  while i < fc:
2          ret, img = cap.read()
3          plt.imshow(img)
4
5          img = np.rot90(np.rot90(np.rot90(img)))
6          plt.imshow(img)
7
8          b, g, r = cv2.split(img)
9          img = cv2.merge([r, g, b])
10         plt.imshow(img)
11
12         img = cv2.resize(img, dsize=(int(img.shape[1] /
           ↪ 50), int(img.shape[0] / 50)))
13         plt.imshow(img)
14
15         itrs[x] = img
16         ltrs[x] = index
17
18         if not ret:
19             break
20
21         i += 1
```

The task of the block of python code above is converting videos to image frames as NumPy arrays, refining images, storing all images from every recorded video of buildings in a single NumPy array.

- Line 2-3: Reads the next image, stores it in a temporary NumPy array and plots the image.
- Line 5-6: Rotates the image, re-plots the image.
- Line 8-10: Changes color model of the current image frame from BGR to RGB, re-plots the image.
- Line 12-13: Resizes the current image, changes to a lower resolution, for an improved training performance.

The loop of the code above will end after reaching the last image frame of the video. It will be running again but with another video of next building, until all videos are read.

The NumPy array "itrs" in the end will consists of all image frames of all buildings as NumPy arrays.

The refining steps shown above will also be done for the validation data, so both training data and validation data has the same resolution, also for the testing data too for an improved testing performance.

3.1.3 Model Structure

The CNN model structure is built quite simple. It consists of four Conv2D layers, one Maxpooling2D layer, one Dropout layer, one Flatten layer and one Dense layer. All these layers are imported from the library Keras.

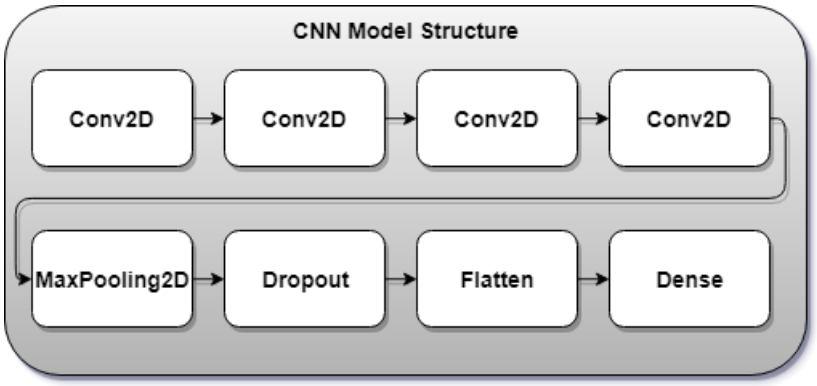


Figure 3.1: Structure of the built CNN model visualized in layers on top of layers.

This structure shown above fits well for my training data. Four Conv2D layers makes the model training accuracy very high, almost 100%. The Maxpooling2D layer helps with overfitting issues by reducing input parameters. The Dropout layer with a fraction rate of 0.4 also helps preventing overfitting. The Flatten layer is used second last to make multi-dimensional inputs to one dimensional, which is essential before going to the output layer. The last layer is an output layer which is a Dense layer will classify validation data with the training data that is being trained on the model.

3.2 Application Design

The application design is simple. The client and server will be communicating with each other through a TCP socket connection. Unity is used to create the application. JupyterLab is used to create the CNN model.

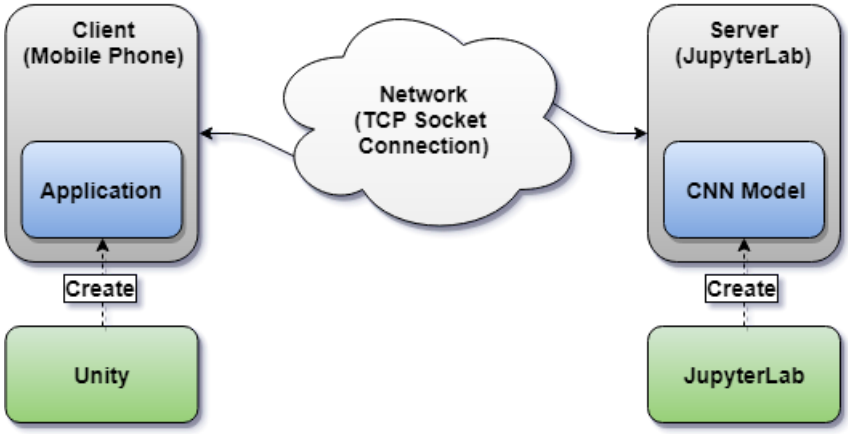


Figure 3.2: Flow of the application, visualization of the application design. Unity is used to create the application, the application sends to the server an image of a building taken from its device through the network. The CNN model is created with JupyterLab. The server will identify all the buildings of the received images with the CNN model, and send back to the client or the corresponding device the information about the building.

When the application is running, the back camera of the mobile phone will always be on until the application terminates. A button is created positioned in middle bottom of the mobile phone screen. The task of the client by using the application is to send an image to the server through a TCP socket connection whenever the button is clicked. In the server, any received image will be refined. Afterwards proceeds the identification of what building the image consists of. After prediction, the information of the predicted building such as name and background will be obtained by a function that returns a string with the information. The string will be sent to the client through the same TCP socket connection. Client receives the string containing information about the building and visualizes it on the mobile phone screen. A more detailed version of the communication between client and server is visualized on figure 3.3.

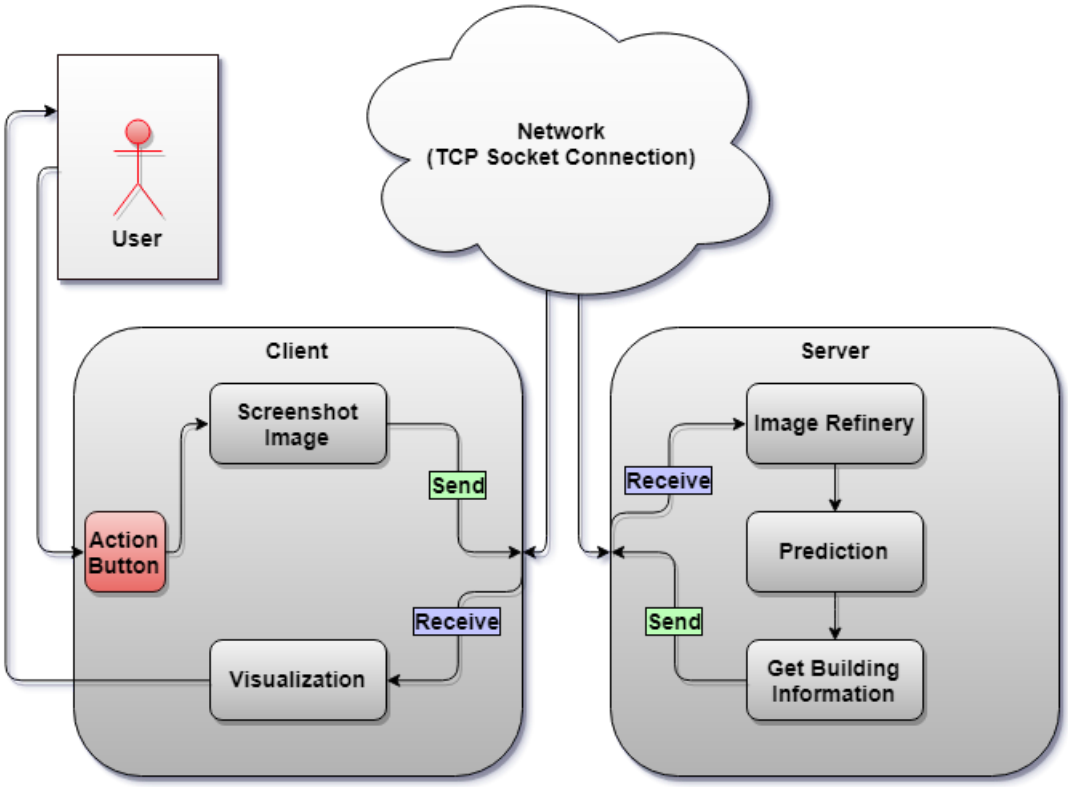


Figure 3.3: Detailed image of communication flow between client and server.

4

Result

This chapter will cover the final results of the whole project. In section 4.1, the results of the mobile application will be described. Section 4.2 is about training and validating CNN model. Section 4.3 will show image results during conversion and refining. Section 4.4 will visualize the results of the CNN model. Last section 4.5 of this chapter will cover the performance of the whole application.

4.1 Mobile Application

The mobile application consists of a button for classifying image of a building, panel for visualizing the information of the predicted building and a signal label that tells the user whether the client is connected to the server or not.

Figure 4.1: Image of the building "Kuggen" with its visualized information on the application.

Figure 4.2: Image of the building "Kokboken" with its visualized information on the application.

4.2 Training and Validation

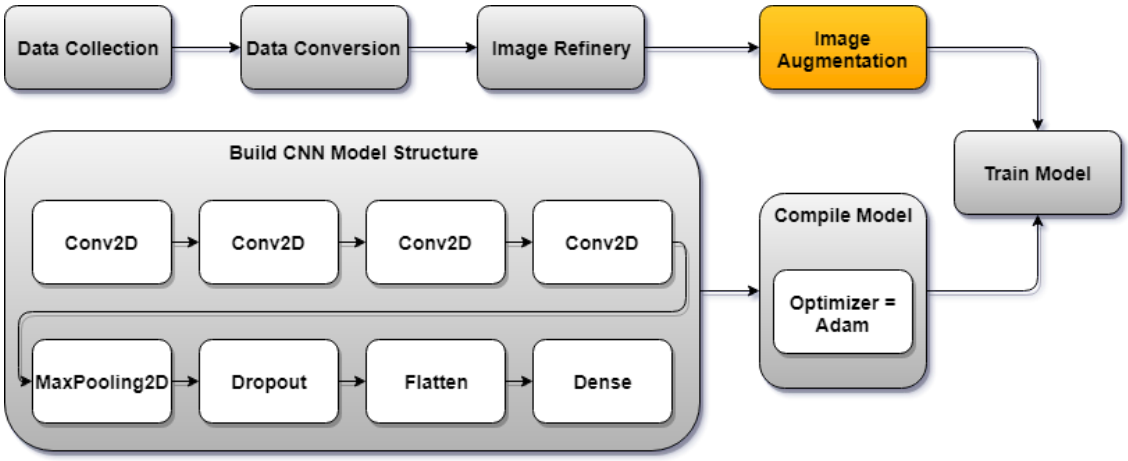


Figure 4.3: Visualization of building steps of the image classifier for this project.

Without image augmentation
The model gets 100% training accuracy and 0% training loss both at 20th epoch. The validation accuracy is 68% and the validation loss is over 100% at 20th epoch, which is bad, and also means the model is overfitting.

With image augmentation
The model gets 100% training accuracy and 0% training loss both at 20th epoch. The validation accuracy is 100% and the validation loss is 0% both at 20th epoch. Image augmentation improved the validation accuracy and is a help to prevent overfitting.

4.3 Conversion and Refining

The result during conversions and refining at the lines of codes in previous chapter 3, section 3.1.2 will be shown in figures 4.4-4.7.

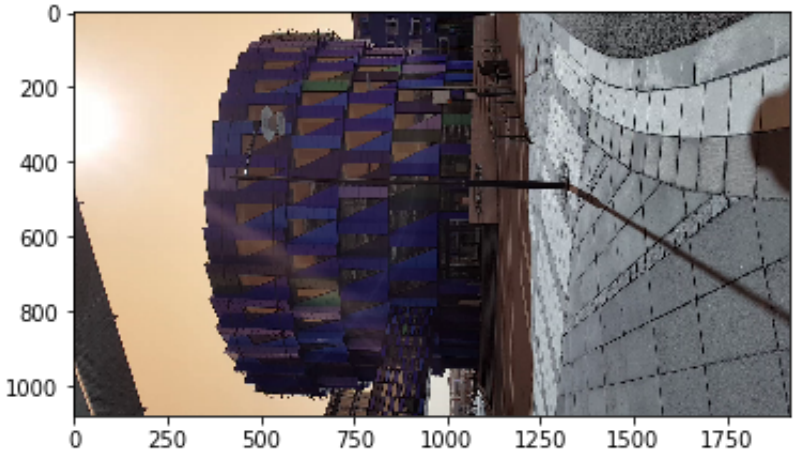


Figure 4.4: An image frame of the building "Kuggen".

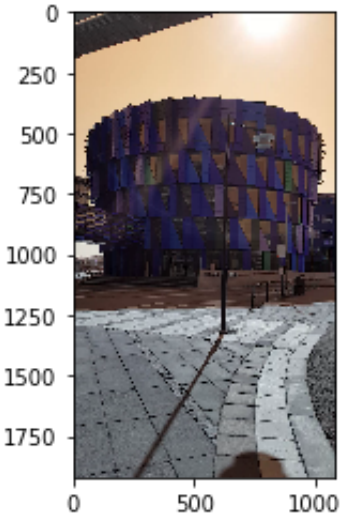


Figure 4.5: A rotated image frame of the building "Kuggen".

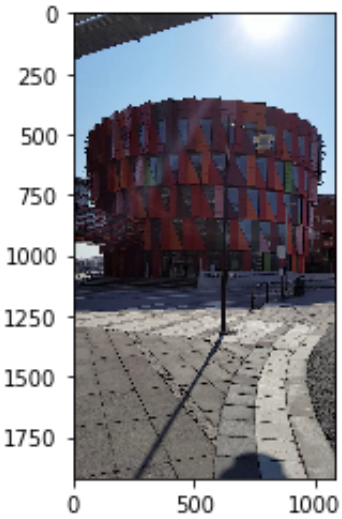


Figure 4.6: Image frame of the building "Kuggen" in RGB color model.

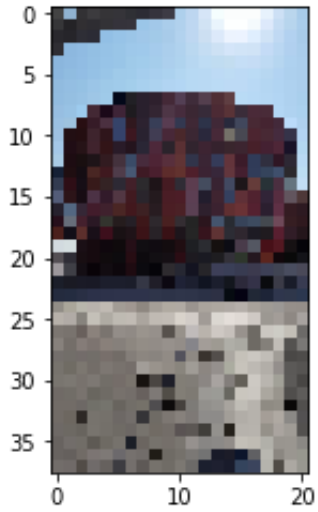


Figure 4.7: Image frame of the building "Kuggen" in a lower resolution.

4.4 Convolutional Neural Network Model

The CNN model gave a high training accuracy, validation accuracy, low training loss and low validation loss. Some images of graphs will be shown in figures 4.8-4.11 to visualize the accuracies and losses with or without image augmentation.

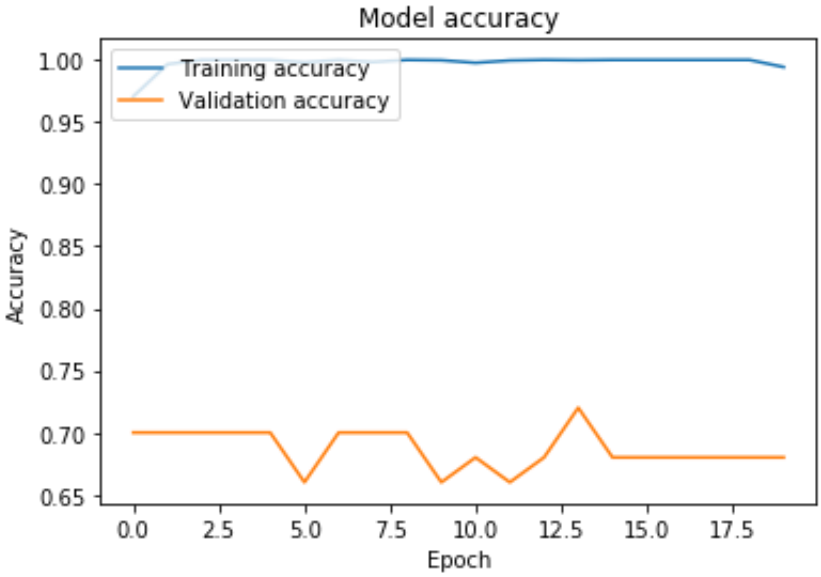


Figure 4.8: Training accuracy and validation accuracy of the CNN model without image augmentation.

Figure 4.9: Training loss and validation loss of the CNN model without image augmentation.

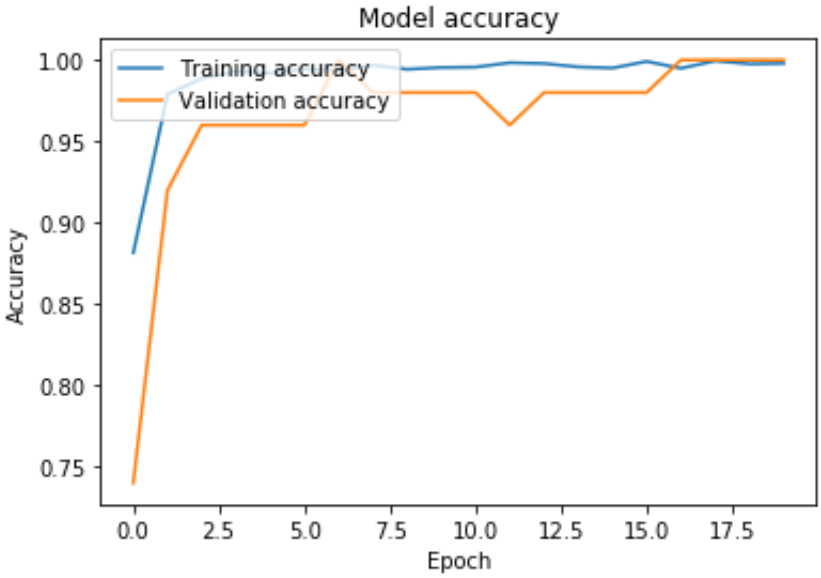


Figure 4.10: Training accuracy and validation accuracy of the CNN model with image augmentation.

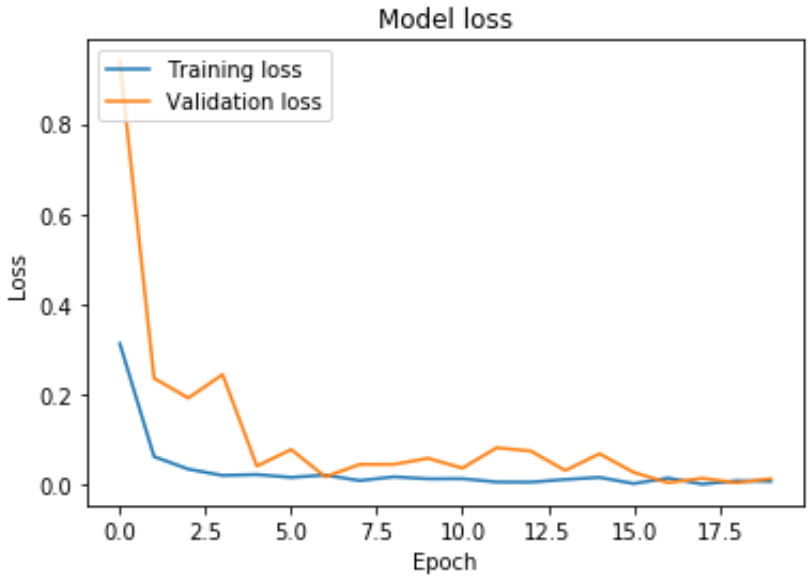


Figure 4.11: Training loss and validation loss of the CNN model with image augmentation.

This CNN model is trained for total of five buildings, with a total of 16678 frames of images of all five buildings combined together.

4.5 Performance

The performance of the whole application is bad, it was slow. It takes 2-3 seconds, from clicking the button of the mobile application, to sending image to the server, to receiving information from the server, to visualizing the received information. When the button of the mobile application is clicked, the application freezes while sending the image to the server, after the image is sent, the application unfreezes. An attempt to solve the freezing problem was by running the sending process parallel by using another thread, which did not solve the problem.

5

Conclusion

This chapter will discuss about the final outcome in section 5.1. Section 5.2 will cover the alternative method that were first used during creation of the application and design comparison between both first method and the current method. In section 5.3 will be discussing about possible continuation of the project.

5.1 Outcome

The final result of the mobile application turned out nicely. It is able to do the main thing which is to retrieve and display the information about a building on to the existing view of a mobile application. But it were not able to do it quick and smoothly, it took approximately two seconds to do all that. Right now the application is only able to classify five chosen buildings which is a bit too few.

5.2 Alternative Method

The first method is creating the CNN model and having it inside the mobile application. It did not work that well, and could not overcome some errors while attempting to load the CNN model into the application.

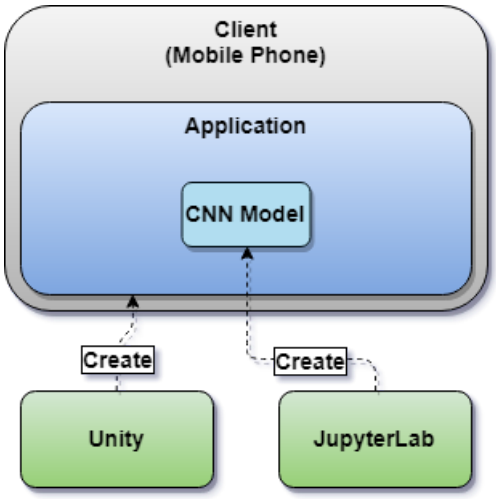


Figure 5.1: The first initial structure of the application design.

There are some differences between first design and current design. First design makes the application skip the whole TCP communication step, which processes much faster. Bad thing is that the application will weight more which takes a lot of space on the mobile phone. Current design is just the opposite from the old design. Current design makes application communicate between client and server which delays time from networking, but the application takes less space from your device compared to the first design. Current design requires a server computer and can not update remotely as easily.

5.3 Continuation

It is a good idea to attempt to build the application with the first design. The first design has more advantages than the current design. It will skip the TCP communication step. It will maybe process faster, depending on what libraries will be used for the CNN model. Examples of future features and functionality for the application will be listed below.

- Add more buildings to CNN model to be classified.
- Obtain more unique training data for an even better accuracy. Requires more training data for each building if more buildings is added to the CNN model.
- Add other objects such as buses and trams to be classified, the application should be able to retrieve information about the bus or tram. Information could be a list of all the stops the vehicle will pass by.
- Attempt to make the application process faster than the current time it takes with identification and network communication.
- Make the application able to classify multiple objects in one button click.
- Add augmented reality to the application, improves visualization of information.
- Extend the application from usable only in Gothenburg to the whole country of Sweden. Training data may not be obtained by recording but by other complex

methods which will not be explained in this project.

References

[1] Peter Norvig Stuart J. Russell. Artificial Intelligence: A Modern Approach, 3rd ed. 2009, pp. 2.

[2] David Andre Martin A. Keane John R. Koza, Forrest H. Bennett III. How can computers learn to solve problems without being explicitly programmed? in Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming. 1996. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.3544&rep=rep1&type=pdf. Accessed on: Oct 06, 2019.

[3] Expert System Company. Who we are. [Online]. Available: https://www.expertsystem.com/company/. Accessed on: Oct 02, 2019.

[4] Expert System Company. What is Machine Learning? A definition, 2017. [Online]. Available: https://www.expertsystem.com/machine-learning-definition/. Accessed on: May 27, 2019.

[5] Mandeep Kaur. Top 10 real-life examples of Machine Learning, 2019. [Online]. Available: https://bigdata-madesimple.com/top-10-real-life-examples-of-machine-learning/. Accessed on: May 27, 2019.

[6] Jürgen Schmidhuber Dan Ciresan, Ueli Meier. Multi-column deep neural networks for image classification. 2012 IEEE Conference on Computer Vision and Pattern Recognition. 2012, pp. 3642–3649. [Online]. Available: https://ieeexplore.ieee.org/document/6248110. Accessed on: Oct 06, 2019.

[7] S. P. Mishrab Pooja Asopaa Sakshi Indoliaa, Anil Kumar Goswamib. Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918308019?via%3Dihub. Accessed on: Oct 06, 2019.

[8] Kenneth Soo Annalyn Ng. ALGOBEANS - CONVOLUTIONAL NEURAL NETWORKS (CNN) INTRODUCTION, 2016. [Online]. Available: https://algobeans.com/2016/01/26/introduction-to-convolutional-

neural-network/. Accessed on: May 27, 2019.

[9] Keras. Convolutional Layers. [Online]. Available: https://keras.io/layers/convolutional/. Accessed on: May 30, 2019.

[10] Pooling Layer in Convolutional Neural Networks (CNNs / ConvNets). [Online]. Available: http://cs231n.github.io/convolutional-networks/#pool. Accessed on: May 30, 2019.

[11] Core Layers. [Online]. Available: https://keras.io/layers/core/. Accessed on: May 30, 2019.

[12] Himanshu Sharma. Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning, 2019. [Online]. Available: https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e. Accessed on: May 31, 2019.

[13] Uniqtech. Understand the Softmax Function in Minutes, 2018. [Online]. Available: https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d. Accessed on: Oct 06, 2019.

[14] Augment Company. Infographic: The History of Augmented Reality, 2016. [Online]. Available: https://www.augment.com/blog/infographic-lengthy-history-augmented-reality/. Accessed on: Oct 06, 2019.

[15] Ariful Islam. The Future of Augmented Reality, 2018. [Online]. Available: https://medium.com/predict/the-future-of-augmented-reality-90143b98f7a3. Accessed on: Oct 06, 2019.

[16] David T. Chen William F. Garrett Mark A. Livingston Andrei State, Gentaro Hirota. Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking. [Online]. Available: https://www.cs.princeton.edu/courses/archive/fall01/cs597d/papers/state96.pdf. Accessed on: Oct 06, 2019.

[17] Samuel Axon. Unity at 10: For better—or worse—game development has never been easier, 2016. [Online]. Available: https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/. Accessed on: Oct 06, 2019.

[18] Unity Company. Unity - Multiplatform. [Online]. Available: https://unity3d.com/unity/features/multiplatform/. Accessed on: May 28, 2019.

[19] Unity Company. Graphics API support. [Online]. Available: https://docs.unity3d.com/Manual/GraphicsAPIs.html. Accessed on: May 28, 2019.

[20] Overview. [Online]. Available: https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html. Accessed on: May 28, 2019.

[21] Notebooks. [Online]. Available: https://jupyterlab.readthedocs.io/en/stable/user/notebook.html. Accessed on: May 28, 2019.

[22] Jupyter Notebook - Types of Cells. [Online]. Available: https://www.tutorialspoint.com/jupyter/jupyter_notebook_types_of_cells.htm. Accessed on: May 28, 2019.

[23] Jeffrey Dean. TensorFlow: Open source machine learning, Youtube. [Video]. Available: https://www.youtube.com/watch?v=oZikw5k_2FM. Accessed on: Oct 06, 2019.

[24] Keras: The Python Deep Learning library. [Online]. Available: https://keras.io. Accessed on: May 29, 2019.

[25] NumPy. [Online]. Available: https://www.numpy.org. Accessed on: May 29, 2019.

[26] OpenCV. [Online]. Available: https://opencv.org. Accessed on: May 29, 2019.

[27] About. [Online]. Available: https://opencv.org/about/. Accessed on: May 29, 2019.

[28] Matplotlib. [Online]. Available: https://matplotlib.org. Accessed on: May 29, 2019.

[29] matplotlib.pyplot.imread. [Online]. Available: https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.imread.html. Accessed on: May 27, 2019.

[30] Jason Brownlee. Overfitting and Underfitting With Machine Learning Algorithms, 2019. [Online]. Available: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/. Accessed on: May 30, 2019.

[31] Suki Lau. Image Augmentation for Deep Learning, 2017. [Online]. Available: https://towardsdatascience.com/image-augmentation-for-

deep-learning-histogram-equalization-a71387f609b2. Accessed on: May 31, 2019.

[32] Image Preprocessing. [Online]. Available: https://keras.io/preprocessing/image/. Accessed on: May 31, 2019.

[33] Classification: Accuracy. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/accuracy. Accessed on: Nov 04, 2019.

[34] Descending into ML: Training and Loss. [Online]. Available: https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss. Accessed on: Nov 04, 2019.

[35] The Sequential model API. [Online]. Available: https://keras.io/models/sequential/. Accessed on: Nov 04, 2019.

[36] Eijaz Allibhai. Building a Convolutional Neural Network (CNN) in Keras, 2018. [Online]. Available: https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5. Accessed on: May 31, 2019.