DEEP LEARNING APPROACHES IN FACE RECOGNITION USING TENSORFLOW

A THESIS SUBMITTED TO

THE FACULTY OF ARCHITECTURE AND ENGINEERING

OF

EPOKA UNIVERSITY

BY

EMILIAN CIKALLESHI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR

THE BACHELOR DEGREE

IN

COMPUTER ENGINEERING

JULY, 2020

**Approval sheet of the Thesis**

This is to certify that we have read this thesis entitled “**DEEP LEARNING APPROACHES IN FACE RECOGNITION USING TENSORFLOW**” and that in our opinion it is fully adequate, in scope and quality, as a thesis for the Bachelor Degree of Computer Engineering.

Head of Department:

Dr. Ali Osman Topal

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Student Name: Emilian Cikalleshi

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ACKNOWLEDGMENTS**

I would like to express my gratitude to my supervisor Ali Osman Topal, for his unwavering support, guidance and insights throughout this research project.

I would also like to thank all my friends and family who supported me with love and understanding,

**TABLE OF CONTENT**

Contents

[List of Figures 5](#_Toc46737089)

[ABSTRACT 7](#_Toc46737090)

[ABSTRAKT 8](#_Toc46737091)

[CHAPTER 1: INTRODUCTION 9](#_Toc46737092)

[1.1 Artificial intelligence 9](#_Toc46737093)

[1.2 Machine learning 10](#_Toc46737094)

[1.3 Deep learning 11](#_Toc46737095)

[1.4 Deep learning and face recognition [3] 13](#_Toc46737096)

[CHAPTER 2: LITERATURE REVIEW 14](#_Toc46737097)

[2.1 Face Recognition 14](#_Toc46737098)

[2.2 Face Recognition Processing Flow 14](#_Toc46737099)

[2.3 Artificial Neural Networks 15](#_Toc46737100)

[2.4 Convolution neural networks (CNN) 18](#_Toc46737101)

[2.4.1 Convolution Layer 18](#_Toc46737102)

[2.4.2 Pooling Layer 21](#_Toc46737103)

[2.4.3 Full connection 24](#_Toc46737104)

[CHAPTER 3: METHODOLOGY 25](#_Toc46737105)

[3.1 Tools: 25](#_Toc46737106)

[3.2 Dependencies 25](#_Toc46737107)

[3.4 Workplace 26](#_Toc46737108)

[3.5 Dataset 27](#_Toc46737109)

[3.6 Shifting and rotating images 28](#_Toc46737110)

[3.7 Convolution Neural Network Model Layers 29](#_Toc46737111)

[3.8 Training the model 31](#_Toc46737112)

[3.9 Testing the model 32](#_Toc46737113)

[Chapter 4: Results and discussion 34](#_Toc46737114)

[4.1 Face Detection 34](#_Toc46737115)

[4.2 Accuracy graph for the trained model 36](#_Toc46737116)

[CHAPTER 5: CONCLUSIONS 41](#_Toc46737117)

[REFERENCES 42](#_Toc46737118)

[APPENDIX 43](#_Toc46737119)

# List of Figures

Figure 1.1 – Subsets of AI

Figure 1.2 – ML steps

Figure 1.3 – Artificial neural network

Figure 1.4 – Machine learning vs deep learning

Figure 2.1 - Face recognition processing flow

Figure 2.2 – ANN Model

Figure 2.3 - Artificial Neuron Model

Figure 2.4 - Convolution layer

Figure 2.5 - Activation layer

Figure 2.6 - Pooling layer

Figure 2.7 - FC layer

Figure 2.8 - Regularization

Figure 2.9 - Dropout

Figure 2.10 Mini-batch values

Figure 3.1 The Tools used

Figure 3.2 Imported Libraries

Figure 3.3 Jupyter workplace

Figure 3.4 Python Code Files

Figure 3.5 Leonardo DiCaprio

Figure 3.6 Edited Pictures

Figure 3.7 CNN representation

Figure 3.8 How photos look after layers have been applied

Figure 3.9 Training and validation accuracy

Figure 3.10 Training amd validation loss

Figure 3.11 George W. Bush

Figure 4.1 Random pictures from dataset

Figure 4.2 Face extraction function

Figure 4.3 New dataset with extracted faces

Figure 4.4 Training and validation accuracy graph for DS1

Figure 4.5 Training and validation accuracy graph for DS2

Figure 4.6 Training and validation loss for DS1

Figure 4.7 Training and validation loss for DS2

Figure 4.8 accuracy function after changing the learning rate

Figure 4.9 loss function after changing the learning rate

Figure 4.10 Results for testing model with learning rate 0.00005

# ABSTRACT

Cikalleshi, Emilian

Bsc. In Computer Engineering

July, 2020

Epoka University, Tirane

With the increasing demand for faster and more secure ways to keep our sensitive data safe we as engineers are always seeking for better and more reliable algorithms that can provide that. Facial Recognition has changed from an advanced security tool in movies to everybody’s primary security system in their smartphone. It has found use in securing our personal data with a very sophisticated 3D face recognition assisted with depth sensors. Many Intelligence agencies use face recognition to analyze suspects but with such a huge database it takes too long to get a hit in face recognition algorithms and it also uses to much resources for processing. The purpose of this research is to test the effect that different CNN layers will have in the accuracy and efficiency of a face recognition algorithm by training different models and using a better classified dataset by using face detection algorithm and extracting the face from pictures.

***Keywords****: face recognition, algorithm, dataset, database, layers, CNN, face detection, face extracting, model, efficiency, accuracy.*

# ABSTRAKT

Cikalleshi, Emilian

Bsc. në Inxhinieri Kompjuterike

Korrik, 2020

Universiteti Epoka, Tirane

Me rritjen e kërkesës per mënyra më të shpejta e të sigurta për të mbrojtur të dhenat tona, ne si inxhinerë jemi gjithmonë në kërkim të algoritmeve më të mirë e më të besueshëm të cilët mund ta mundësojnë këtë gjë. Njohja e fytyrës ka ndryshuar nga një mjet i avancuar sigurie në filma, tek mënyra primare e sigurisë për secilin prej nesh nga celulari ynë. Ka gjetur përdorim në sigurimin e të dhënave tona personale me një mënyrë shumë të sofistikuar me njofje fytyre 3D me ndihmën e sensorëve të thellsisë. Shumë agjensi inteligjence e perdorin njohjen e fytyrës për të analizuar të dyshuarit, por me një bazë të dhënash shumë të madhe algoritmit do ti duhej shumë gjatë për të gjetur nje lidhje dhe gjithashtu kerkon shumë energji proçesimi. Qëllimi i këtij kerkimi është të testoj ndryshimin që sjellin perdorimi i shtresave të ndryshme të CNN-së në saktësinë dhe efiçencën e algoritmit duke trajnuar modele të ndryshme dhe duke përdorur një mledhje të dhënash më të klasifikuar me anë të algoritmit të detektimit të fytyrës për të ekstraktuar fytyrat nga fotot.

***Fjalë kyçe***: *njohja e fytyres, algorit*ë*m, mbledhej e t*ë dhënave*, baz*ë të dhënash*, shtresa, CNN, detektim fytyre, ekstraktimi i fytyr*ës*, model, efiçence, saktësi.*

# CHAPTER 1: INTRODUCTION

As technology advances many activities that previously were performed by people are now done by computers. Most complex activities are possible due to artificial intelligence. Applications for artificial intelligence are endless in several industries such as banking, marketing and entertainment.

## 1.1 Artificial intelligence

In 2020 people have great benefits from the usage of artificial intelligence in many fields of life such as in applications like Pinterest for classifying images, in Google Maps, in music recommender systems, making a simple Google search, getting recommendations for a movie on Netflix, buying online on Amazon, self-driving cars etc. This term was firstly introduced in 1950s in a computer science conference in Dartmouth [1]. The scientists who participated tried to model the way human brain works and to create advanced computers to imitate humans. Also Alan Turing proposed the Turing Test [2] to check whether a machine can think like a human or not. It stated that a machine to be intelligent would need to have the following characteristics:

* natural language processing - to enable communication
* knowledge representation - to store information
* automated reasoning - to use the stored information for giving conclusions
* machine learning - to adapt to new circumstances
* computer vision - to perceive objects
* robotics - to move objects

AI defines the behavior of an application or a computer that completes a set of tasks and solves complex algorithms. AI is used for building intelligent machines in order to mimic human behaviors. Two main subsets of AI are machine learning and deep learning.

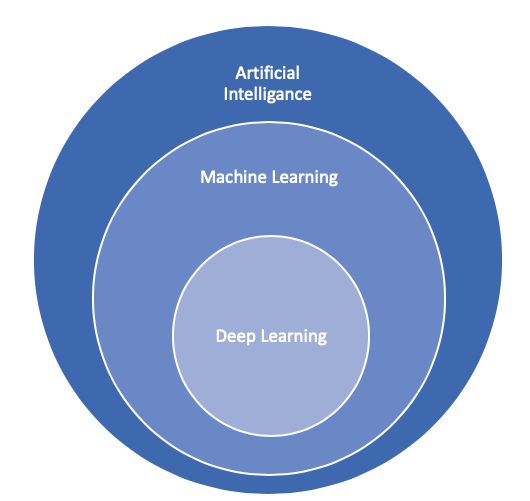


Figure 1.1 – Subsets of AI

## 1.2 Machine learning

Machine learning is a subset of AI that gives systems the ability to figure things out from experience without the need to be programmed for unique tasks. The intention of ML is to create specific algorithms that are able to make valuable predictions on given data.

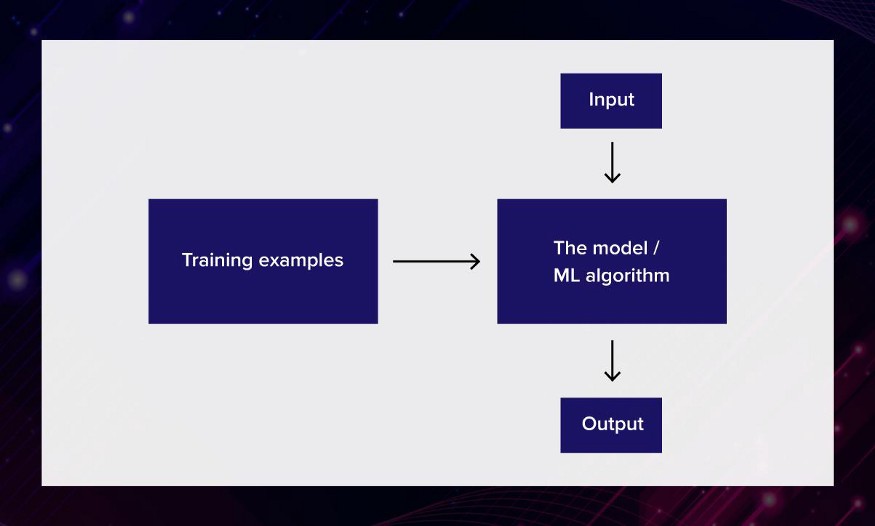


Figure 1.2 – ML steps

Today's main machine learning models used for classification tasks are:

* Linear regression
* Logistic regression
* Decision tree
* K-Nearest Neighbors
* Naive Bayes
* SVM
* Ensembles
* Neural networks

## 1.3 Deep learning

Nowadays computers in machine learning are able to learn without being told what to do exactly. Computers can learn based on data where a huge complex dataset is given and processed by making a prediction on the outcome from the given input. Deep learning is a famous machine-learning technique based on the concept of human brains, artificial neural networks. Deep learning systems resemble the human brain where in humans a neuron is a cell which transmits electrical data from one neuron to another forming the neural network, whereas in machines virtual neurons are bits of code running statistical regressions.

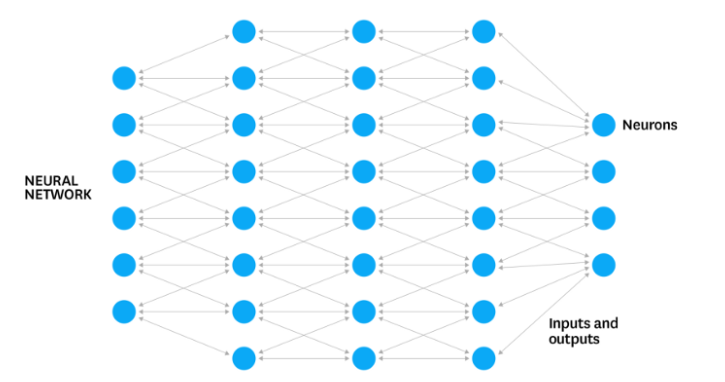


Figure 1.3 – Artificial neural network

Most of machine learning algorithms are linear while deep learning algorithms apply nonlinear transformation to the input and create a statistical model as output. The iterations continue hierarchically until reaching an accurate result. Challenges of Deep Learning include the necessity of having large datasets in order to produce a precise outcome, having enough input data prevents false readings that may occur due to poor lighting conditions and different poses.

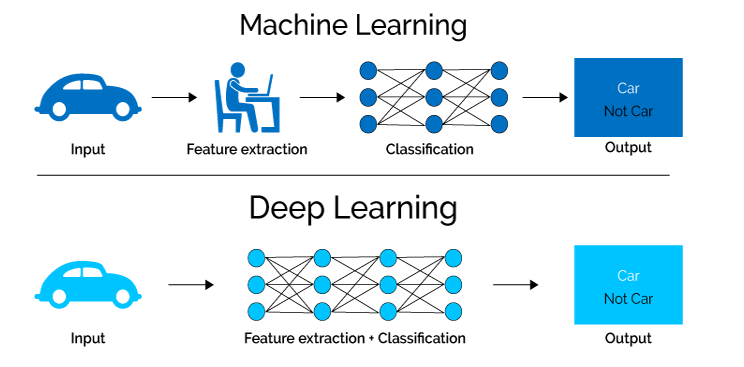


Figure 1.4 – Machine learning vs deep learning

## 1.4 Deep learning and face recognition [3]

People learn how to recognize faces since birth and at the age of four months they may distinguish one person from another. Organs that we pay more attention are: eyes, cheekbones, nose, mouth, eyebrows. Face perceptions are very complex as facial recognition involves different brain areas. Our brain identifies a person by comparing the resulting picture with other internal patterns and finding typical distinctions.

Deep learning methods [4] use large datasets of faces and build an analogy between the real-time data and the stored database to identify an individual. These methods try to imitate the capability of human performance for face recognition tasks. Face recognition contains detection, alignment, feature extraction, face classification and recognition. It will be discussed in more details in the next chapter.

# CHAPTER 2: LITERATURE REVIEW

## 2.1 Face Recognition

Face recognition [5] is becoming a significant research topic as the demands for security arise day by day. It has a wide application in many directions such as law enforcement in police investigation, mobile phones such as Apple replacing passwords and fingerprint scanners, social media such as Facebook for tagging individuals in photos, business security, sport events, airports and border crossings, marketing for targeting audience etc. It is used to identify two-dimensional or three-dimensional images of individual’s face based on distinctive face details such as distance between the eyes, size of the eyes, the elongation of the face , shape of the chin, texture or color of skin. It converts all these details into a mathematical representation and then compares and distinguishes the collected data from other faces in the given database.

The face recognition systems may use a variety of algorithms such as Viola-Jones method [7]. FaceNet [11] is another face recognition system developed by Google which uses a convolutional neural network. It relies on image pixels as features achieving an accuracy of 99.63%.

## 2.2 Face Recognition Processing Flow

1. Firstly, we need to find all the faces in the given image.
2. Secondly, we need to analyze the facial features despite image quality or poor lighting conditions.
3. Thirdly, we need to detect unique details of the face and be able to differ against other known faces.
4. Finally, we need to compare the characteristics between different faces and make a prediction to determine the name of the person.

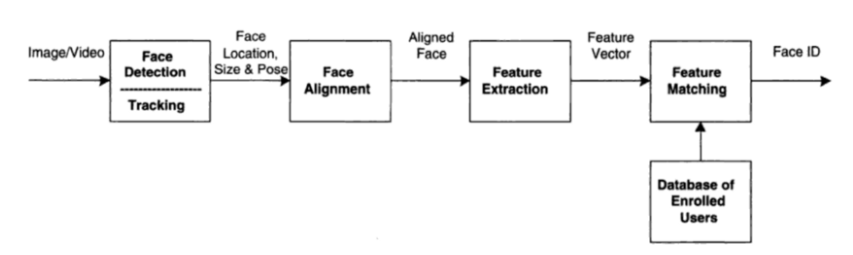


Figure 2.1 - Face recognition processing flow

Our brain does all these four steps automatically and we are able to recognize faces even if we do not see them too often. Whereas computers are not able of such a high level of generalization as the details that are very obvious to people do not make sense for a computer because what computers actually do is that they analyze an image pixel by pixel. So what we have to do is to teach computers the face recognition process step by step by combining different algorithms in one single chain.

## 2.3 Artificial Neural Networks

Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons. Neuron in ANNs tend to have fewer connections than biological neurons. Each neuron in ANN receives a number of inputs. An activation function is applied to these inputs which results in activation level of neuron.

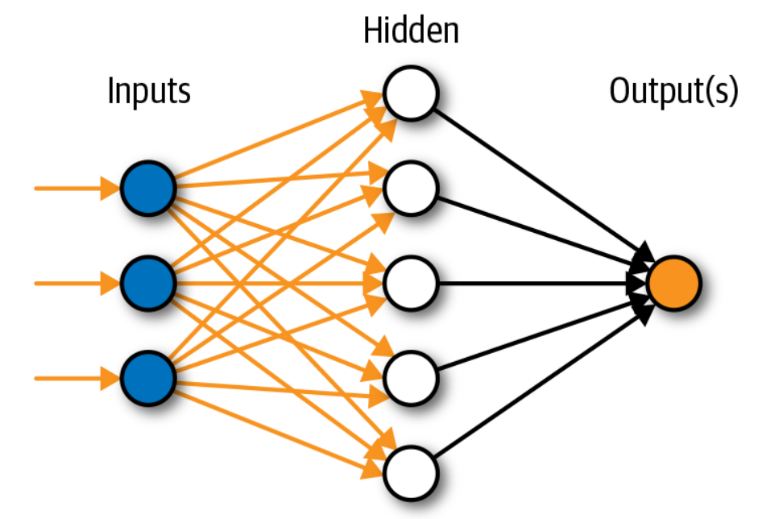


Figure 2.2 – ANN Model

An Artificial Neural Network is specified by: neuron model: the information processing unit of the NN, an architecture: a set of neurons and links connecting neurons. Each link has a weight, a learning algorithm: used for training the NN by modifying the weights in order to model a particular learning task correctly on the training examples. The aim is to obtain a NN that is trained and generalizes well. It should behaves correctly on new instances of the learning task.

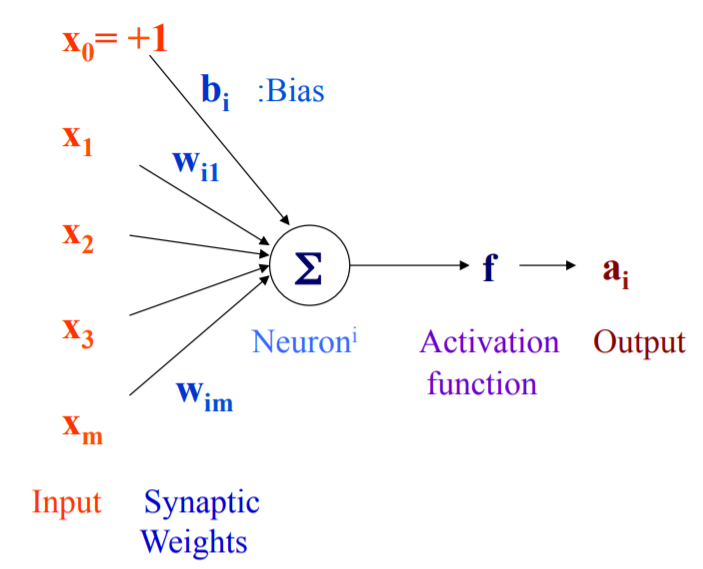
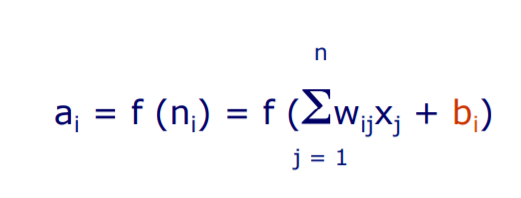
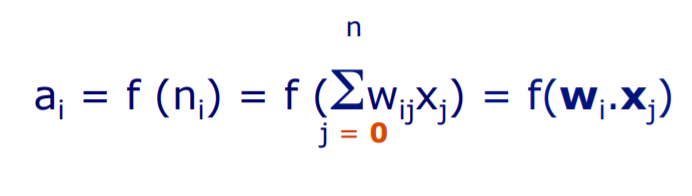


Figure 2.3 - Artificial Neuron Model



An artificial neuron computes the weighted sum of its input (called its net input), adds its bias and passes this value through an activation function. We say that the neuron “fires” (i.e. becomes active) if its output is above zero. Bias can be incorporated as another weight clamped to a fixed input of +1.0. This extra free variable (bias) makes the neuron more powerful.

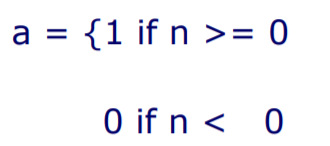


Activation functions also called the squashing function as it limits the amplitude of the output of the neuron. Many types of activations functions are used:

* Linear

C:\Users\erjola\Desktop\14.PNG

* Threshold



* Sigmoid



## 2.4 Convolution neural networks (CNN)

Convolution neural networks (CNN) are artificial neural network that are mostly used for analyzing images. Although image analyses is the most widespread use of CNN it can also be used for other data analysis or classification problems as well. Most generally we can think of a CNN as an artificial neural network that has some type of specialization for being able to pick out or detect patterns and make sense of them.

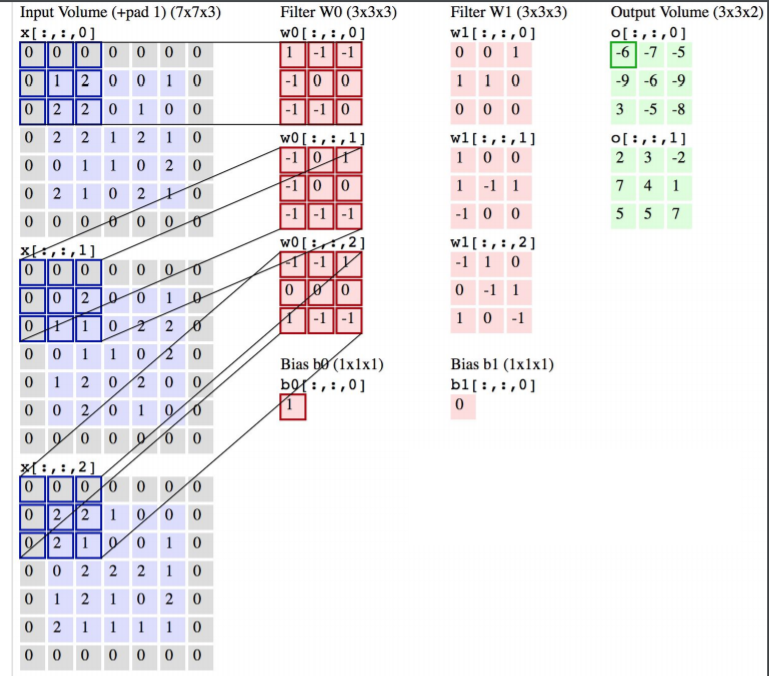
CNN is based on hidden layers called convolutional layers. Now CNNs usually have other non-convolutional layers as well, but the basis of a CNN is the convolutional layers. Just like any other layer a convolutional layer receives input, then transforms the input in some way and then outputs the transformed input to the next layer. With a convolutional layer this transformation is a convolution operation. As mentioned earlier convolutional neural networks are able to tech patterns and images. More precisely the convolutional layers are able to detect patterns. With each convolutional layer we need to specify the number of filters the layers should have and will speak technically about what a filter is in just a few moments.

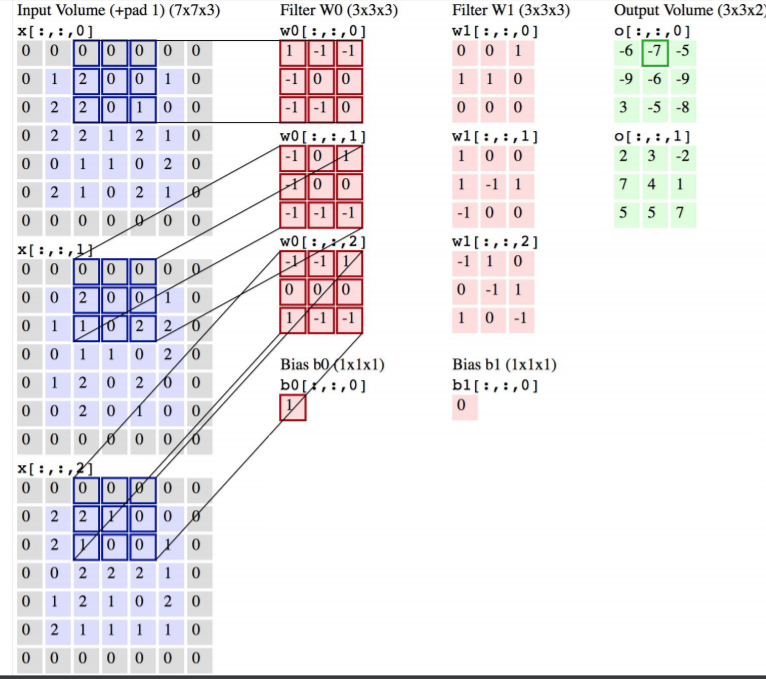
Convolutional Neural Networks have several types of layers:

* Convolutional layer
* Pooling layer
* Full connection

### 2.4.1 Convolution Layer

The process is a 2D convolution on the inputs. The “dot products” between weights and inputs are “integrated” across “channels”. Filter weights are shared across receptive fields. The filter has same number of layers as input volume channels, and output volume has same “depth” as the number of filters.





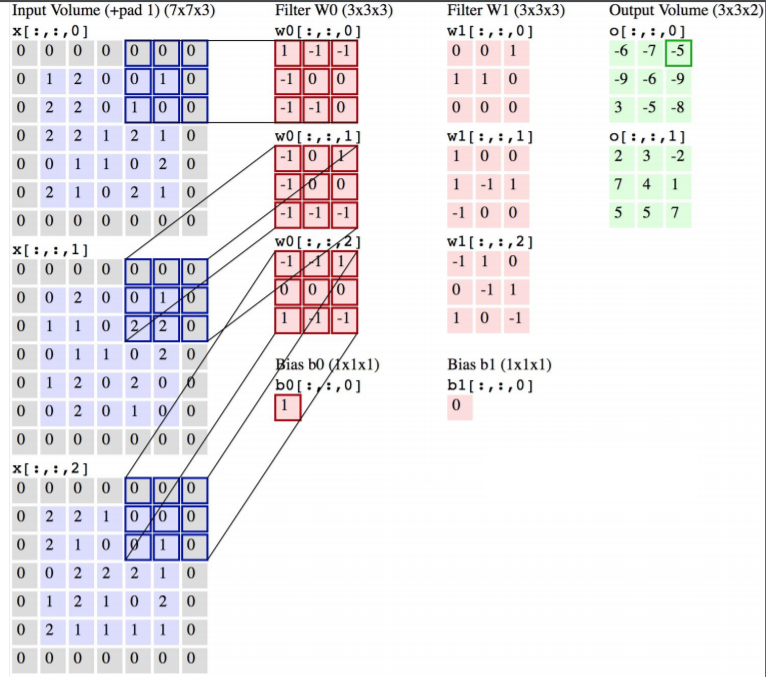


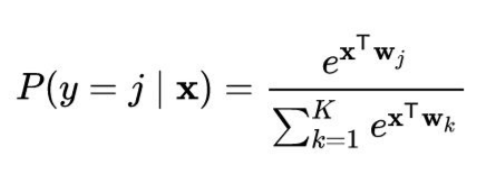
Figure 2.4 - Convolution layer

Activation layer is used to increase non-linearity of the network without affecting receptive fields of convolution layers. Prefer ReLU, results in faster training. LeakyReLU addresses the vanishing gradient problem.



Figure 2.5 - Activation layer

Softmax is a special kind of activation layer, usually at the end of FC layer outputs. It can be viewed as a fancy normalizer (a.k.a. Normalized exponential function). Produce a discrete probability distribution vector. Very convenient when combined with cross-entropy loss.



### 2.4.2 Pooling Layer

Convolutional layers provide activation maps whereas pooling layer applies non-linear downsampling on activation maps. Pooling is aggressive (discard info); the trend is to use smaller filter size and abandon pooling

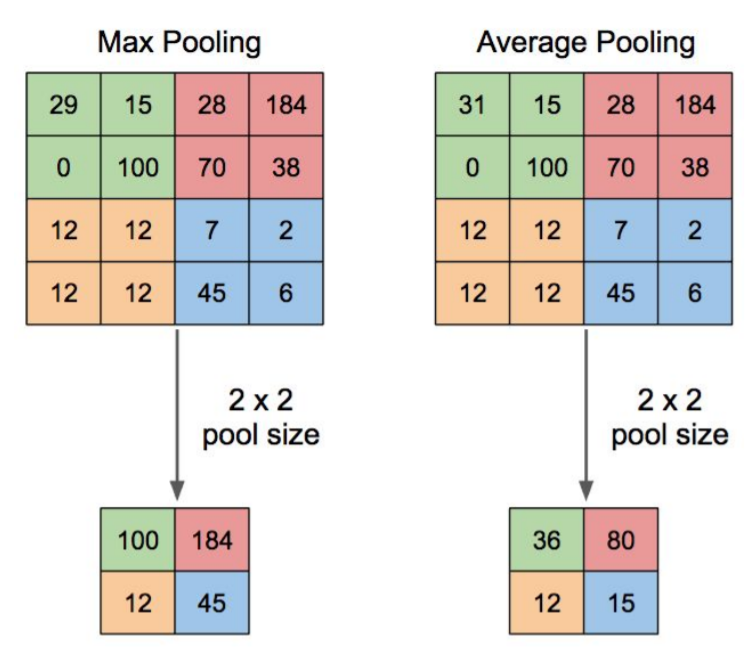


Figure 2.6 - Pooling layer

FC Layer is regular neural network that can be viewed as the final learning phase, which maps extracted visual features to desired outputs. It is usually adaptive to classification/encoding tasks. Common output is a vector, which is then passed through softmax to represent confidence of classification. The outputs can also be used as “bottleneck”.

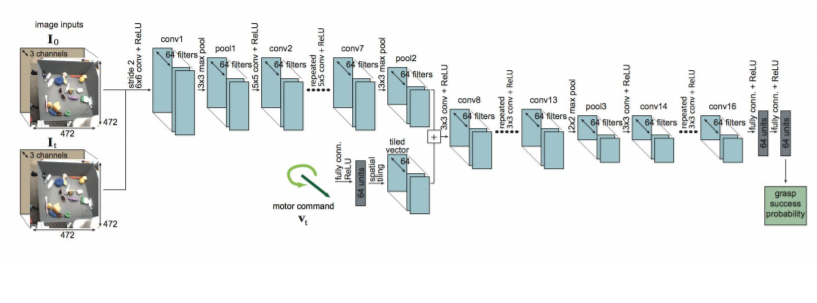
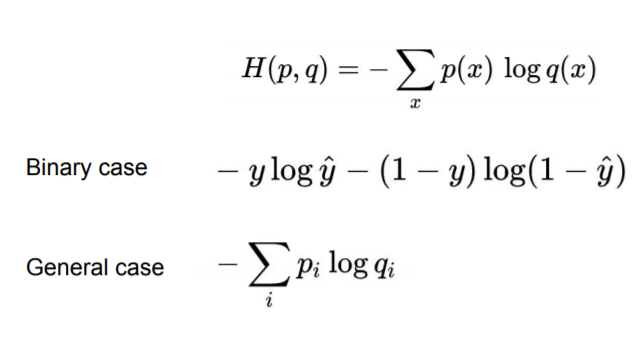


Figure 2.7 - FC layer

Loss Layer - L1, L2 loss. Cross-Entropy loss (works well for classification, e.g., image classification), Hinge Loss, Huber Loss, more resilient to outliers with smooth gradient. Minimum Squared Error (works well for regression task, e.g., Behavioral Cloning)



Regularization - L1 / L2. Dropout, batch norm, gradient clipping, max norm constraint to prevent overfitting with huge amount of training data.



Figure 2.8 - Regularization

Dropout - During training, randomly ignore activations by probability p. During testing, use all activations but scale them by p. Effectively prevent overfitting by reducing correlation between neurons.

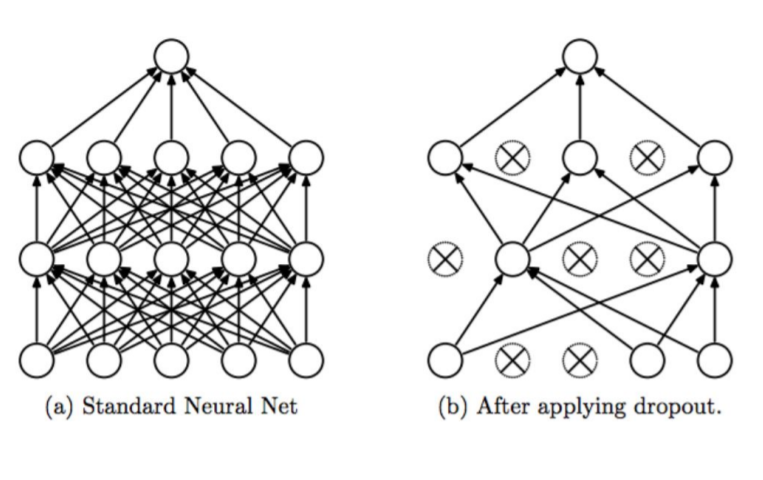


Figure 2.9 - Dropout

Batch Normalization - Makes networks robust to bad initialization of weights. It is usually inserted right before activation layers. Reduce covariance shift by normalizing and scaling inputs. The scale and shift parameters are trainable to avoid losing stability of the network.

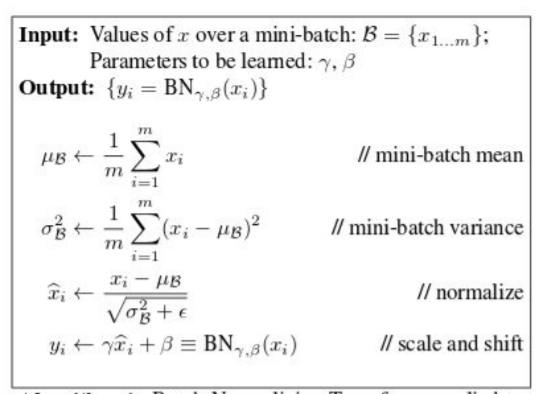


Figure 2.10 Mini-batch values

### 2.4.3 Full connection

Fully connected layers in a CNN are not to be confused with fully connected neural networks – the classic neural network architecture, in which all neurons connect to all neurons in the next layer. Convolutional neural networks enable deep learning for computer vision.

# CHAPTER 3: METHODOLOGY

## 3.1 Tools:

Python 3.7.7

Tensorflow 2.0.0 (cpu)

conda 4.8.3

jupyter-notebook 6.0.3

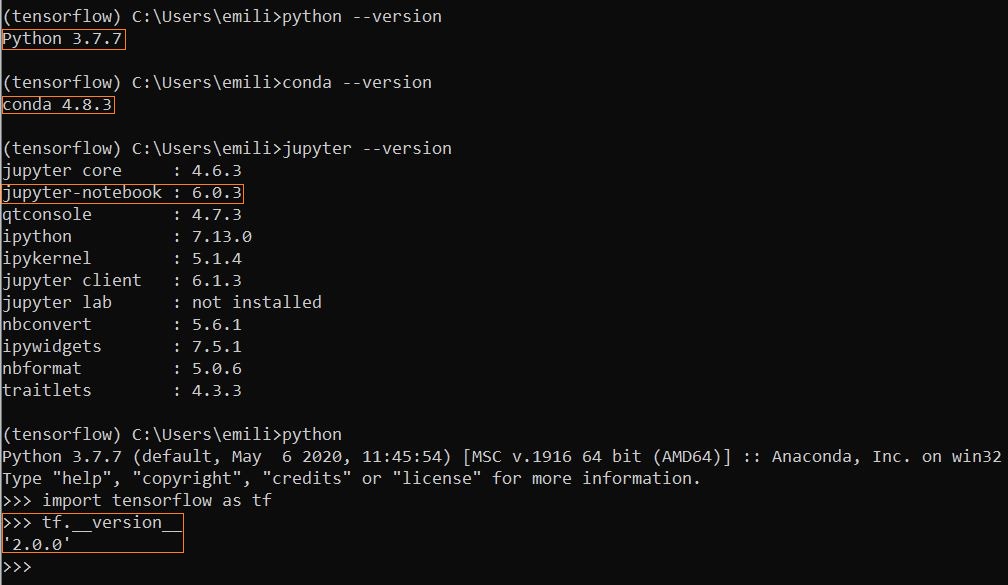


Figure 3.1 The Tools used

## 3.2 Dependencies

For the code to run successfully many python libraries had to be installed like matplotlib or tensorflow and also I installed conda3 and “jupyter-notebook” to use it as the workplace. The following are the main imported libraries that will be needed in further operations.

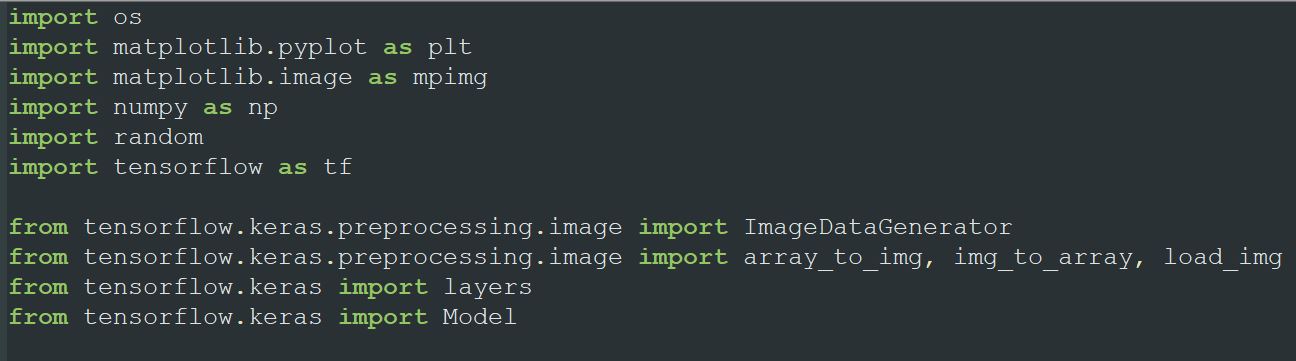


Figure 3.2 Imported Libraries

## 3.3 Workplace

Jupyter-Notebook 6.0.3 is the IDE I used to run the code and Python 3.7.7 (tensorflow) is the language.

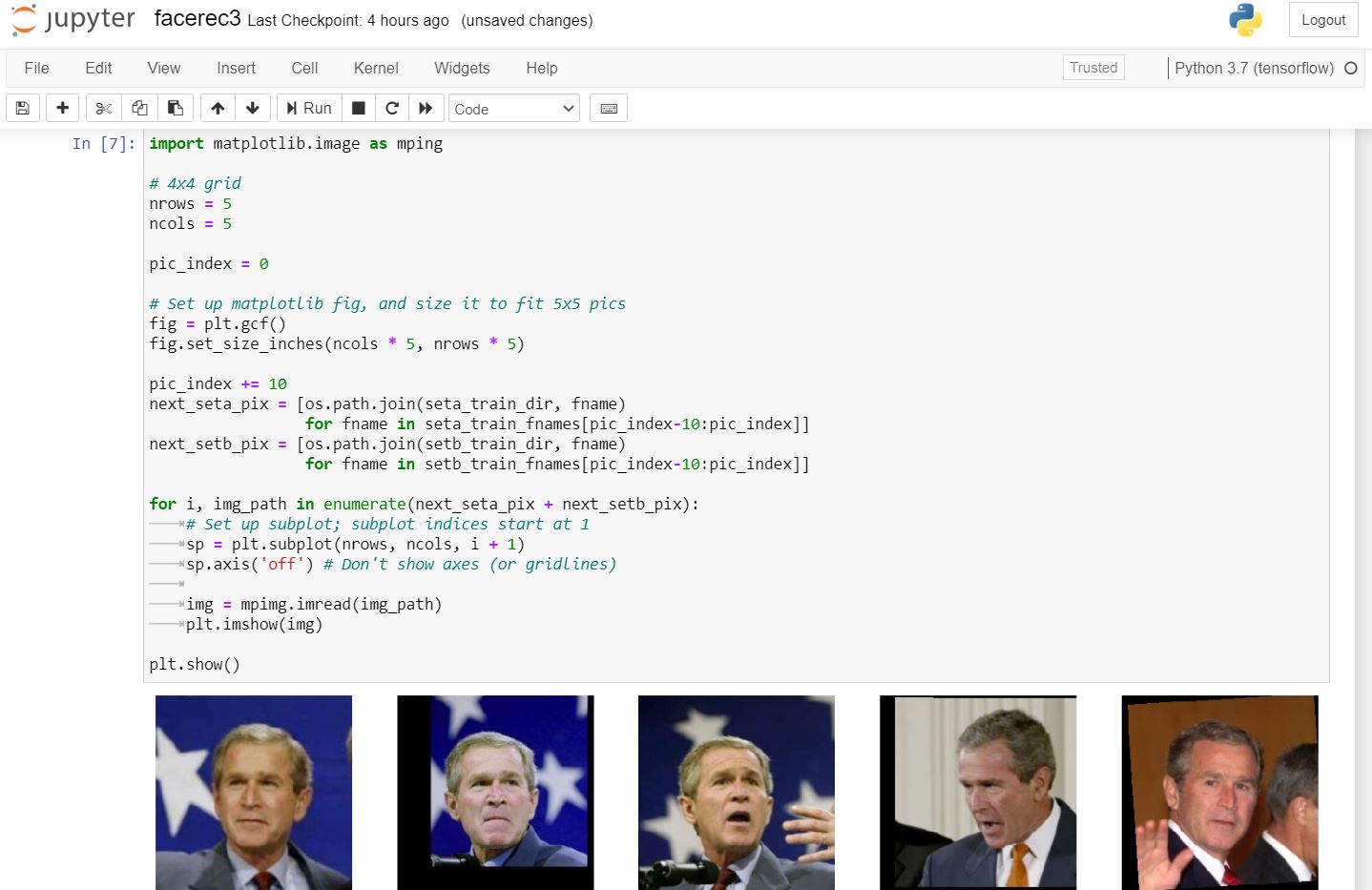


Figure 3.3 Jupyter workplace

The files shown in fig 3.4 are the code files for the whole Face Recognition training, validating and testing. So basically we first prepare the paths to the dataset in order to use them later. Than we take each picture and shift it rotate it to train better with different angles. The model is later on prepared with different layers like Conv2D, MaxPooling, Flattern etc. After we compile the model the training start which will be the longest process of them all. When it is done we print some validation graphs based on how good the model is doing and then start testing.

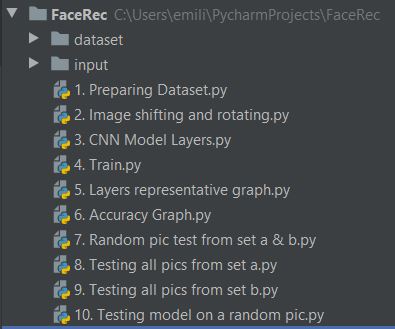


Figure 3.4 Python Code Files

## 3.4 Dataset

In the dataset there have been used three folders of data for each person. These folders contain train, validation and test sets. It is important to use validation set in order to see how the model is doing after each epoch so that we can cut the training process when the accuracy gets good enough without having to train the whole dataset. This way we can save time as the model training normally takes a lot time especially with the cpu version of tensorflow.

## 3.5 Shifting and rotating images

After the images path have been set we do some editing to the images in order to help and improve training the model. So we are going to rotate the photos with a 40 degree angle shift its width and height with a 0.2 range and also zoom and shear with a 0.2 range. After this implementation we can see how this picture of Leonardo DiCaprio has changed.



Figure 3.5 Leonardo DiCaprio

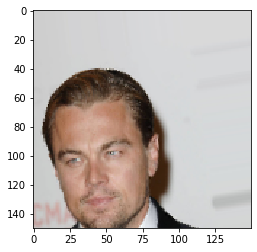






Figure 3.6 Edited Pictures

## 3.6 Convolution Neural Network Model Layers

CNN layers are used to form a huge network with many nodes or parameters. We feed the network some pictures and they form the input layer. The output layer is what we expect from the algorithm to guess correctly. In the middle of these two layers are placed many other layers called hidden layers. These layers are not individually important but the net the form with define particular patterns for certain features. Layers are like filters so they can highlight certain features of the face like the edges, eyes, mouth, nose etc. In the end of the training the patterns formed by the CNN layers will be responsible for guessing the right face. The total parameter set by the layers in my model are 77,222,993.

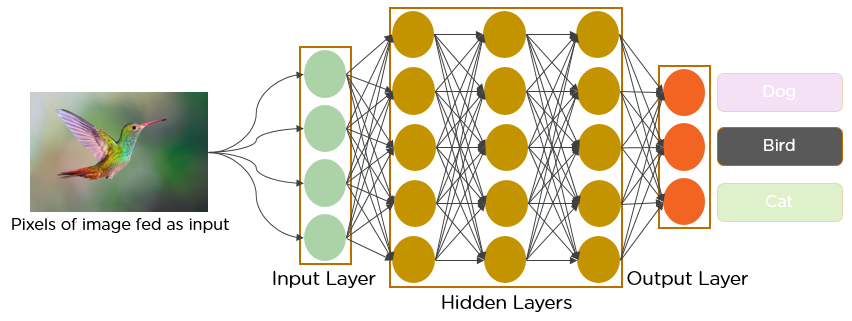


Figure 3.7 CNN representation

In the picture 3.8 below it is a representation of how the pictures look after the layers have been applied. We can see that the layers filter the face emphasizing certain parts based on the filter type.



Figure 3.8 How photos look after layers have been applied

## 3.7 Training the model

After setting the layers we need to compile the model. I have used binary-cross-entropy as the loss function and Adam Optimizer as the optimizing function for model compilation. Than we start the training of the model. I have set the epoch number to 80, steps per epoch to 10 and validation steps to 7. Training the model is the most time consuming task in thus program. For the model to be good enough it need a lot of data but training a lot of data is time and space consuming. My machine was running this task in i7 8th Gen Intel cpu dual core processor and I used a dataset with a total of 770 photos and the training time was about 40 min. When the training is done the graphs show the training accuracy and loss of the training.

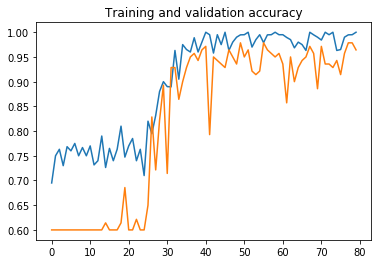


Figure 3.9 Training and validation accuracy

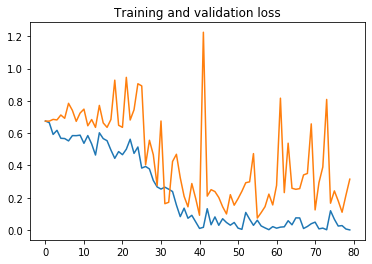


Figure 3.10 Training amd validation loss

## 3.8 Testing the model

In this final step we can test the model created with never seen pictures in the test data folder. Validation graphs are important to tell if the model is improving but with the test data we can now really see how it is performing. We can do tests that take a random image in the test data folder and see if the algorithm is going to guess the right person. Just like we said about CNN layers the output layer is going to be composed of an array of values corresponding to each category. So in the following fig we can see that George W. Bush was correctly guess by the model predictor, because the number (1.) in the output layer is assigned to George W. Bush.



Figure 3.11 George W. Bush

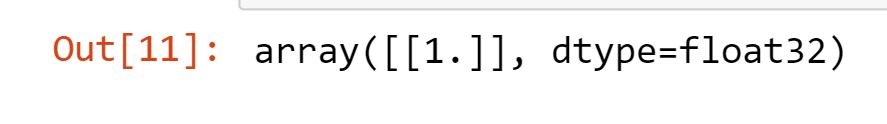


Figure 3.12 Output representing George W. Bush

So it will print George W. Bush name when the output is close to 1.

# Chapter 4: Results and discussion

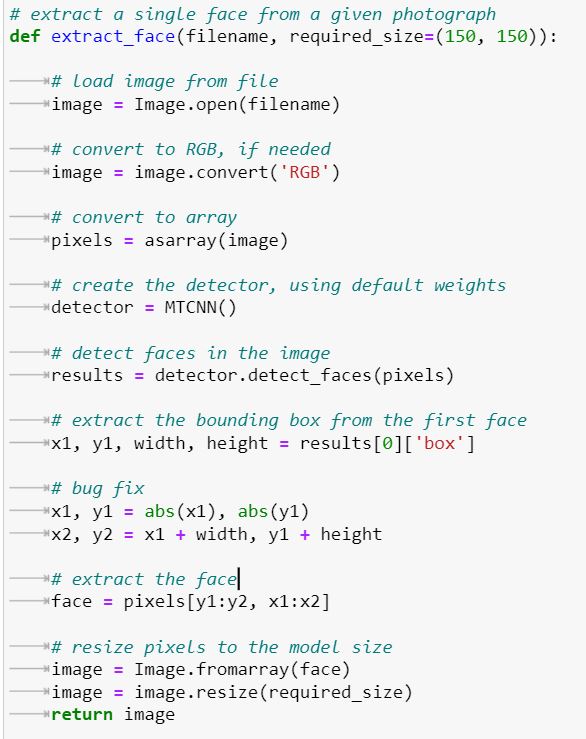
## 4.1 Face Detection

The normal dataset is composed of pictures of persons with more details than their face. So in the plot below we can see hands flags and more which don’t have anything to do with face features so they do not help in face recognition. But they can actually have a negative effect in the algorithm accuracy because after filters are applied some details outside the face can be included in a person’s features and be part of its pattern in the model. So in order to eliminate this we need to extract the faces from the pictures and basically create a new dataset with the faces only.



Figure 4.1 Random pictures from dataset

The face detection function loads the image of the given path. It uses the MTCNN face detector algorithm to draw a square on the face of the person. And then extract the face only resizing the photo to a 150x150 pixels and returning the image. This way we have formed a new dataset with the same people but with their faces only.



4.2 Face extraction function

If we put this function in a loop we can create the new dataset with the extracted faces of people that are in the dataset. This is how the dataset will look like after the extract\_face() function have been executed.



Figure 4.3 New dataset with extracted faces

## 4.2 Accuracy graph for the trained model

So after running the original face recognition through 800 steps of training we can see how the graph rises towards 1 indicating improvement in accuracy. But if we check the training and validation accuracy graph when algorithm was run with the extracted faces dataset we can see that it hits the ceiling way earlier. So with first dataset we get an accuracy close to 1 at about 35th epoch and for the second dataset this happens before 18th epoch. This is about half the time.

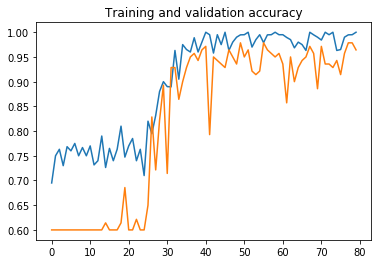


Figure 4.4 Training and validation accuracy graph for DS1

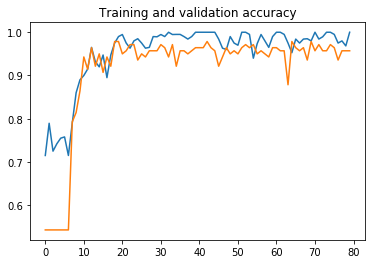
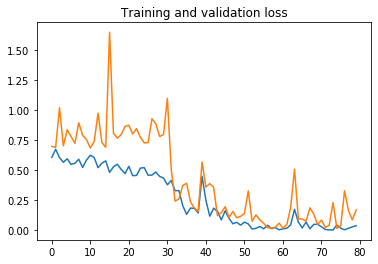
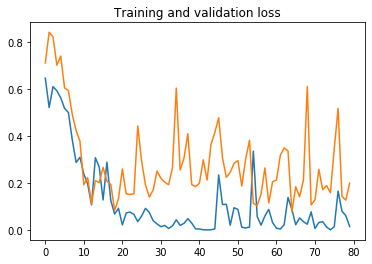


Figure 4.5 Training and validation accuracy graph for DS2

This second dataset seems to perform better based on accuracy but accuracy is not everything we need for the model to perform better. We need to look at the loss function as well.



4.6 Training and validation loss for DS1



4.7 Training and validation loss for DS2

So from the loss function we can see that the validation loss function start to increase a little after 20th epochs that is the same spot when accuracy tops its values. So what happens here is called model overfitting and it occurs when the model becomes too specific and it starts to perform worse when validated on the testing.

After numerous runs in trying to fix the model overfitting I tried to change the number of epochs or steps per epoch, the number of validation steps but what seemed to fix it was decreasing the learning rate. I switched learning rate from 0.0005 to 0.00005 and it made up the best model yet. We can see that the graph tops the values at about 40 epochs and loss function again increases a little after 40 epochs but this is the best model when testing. It guessed correct almost every picture in the test folder with accuracy value about 93.3%, while the first model with the original dataset had testing accuracy value at 89% and the over-fitted model was lower with 85%

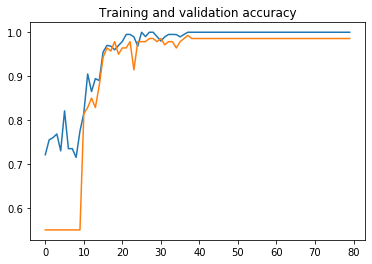


Figure 4.8 accuracy function after changing the learning rate

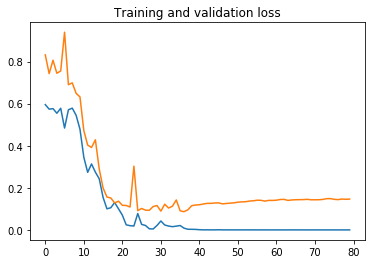


Figure 4.9 loss function after changing the learning rate

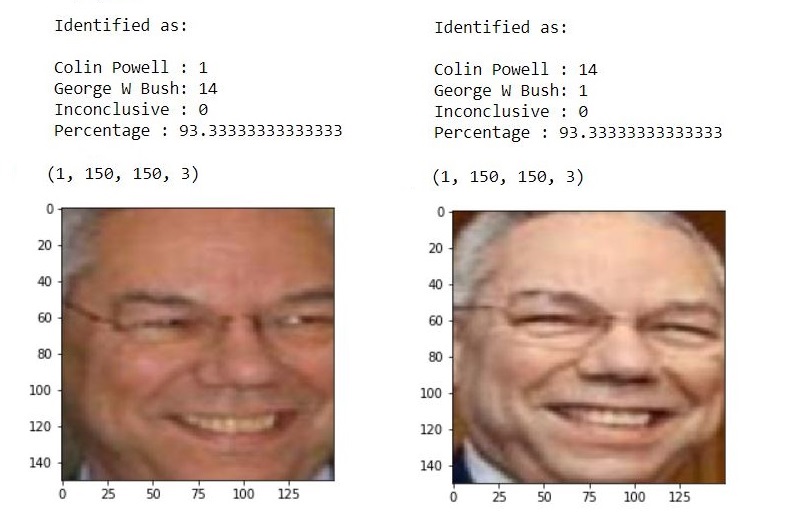


Figure 4.10 Results for testing model with learning rate 0.00005

# CHAPTER 5: CONCLUSIONS

Face Recognition is a very powerful tool in keeping our data secured and classifying people by their face. CNN comes in handy when taking about efficiency and speed. It is important to have a proper refined dataset in order to get good results by our model.

Engaging with layers combination, steps size of training as well as changing the learning rate and the size of the dataset are indeed necessary experiments into further development and improvement of our model. Trying to find the right combination of these properties for the Face Recognition algorithm is very time consuming but very important into improving the existing algorithms.

Also it is very helpful to use face detectors before proceeding to face recognition because it really eases the job for the algorithm when it is messing with the right part of the images in order to form Convolutional Neural Network patterns for the faces from the dataset.

# REFERENCES

[1] Darthmouth workshop - <https://en.wikipedia.org/wiki/Dartmouth_workshop>

[2] “Learn About The Turing Test: Who’s a Bot?”, Brenda Barron, July 2019

[3] Deep Face Recognition: A Survey, [Mei Wang](https://arxiv.org/search/cs?searchtype=author&query=Wang%2C+M), [Weihong Deng](https://arxiv.org/search/cs?searchtype=author&query=Deng%2C+W), February 2019

[4] Deep Face Recognition, Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman

[5] Face Recognition using Machine Learning, Arun Alvappillai, Peter Neal Barrina

[6] “Face Recognition Based on Convolution Neural Network (CNN) Applications in Image Processing: A Survey”, Sudha Sharma, Alpesh Soni, Vijay Malviya, April 2019

[7] “Faster face detection using Convolutional Neural Networks & the Viola-Jones algorithm”, Karina Enriquez, May 2018

[8] “How to Perform Face Detection with Deep Learning”, Jason Brownlee, November 2019

[9] “Face Recognition Using Convolutional Neural Networks”, V.K.N Kamlesh Pai, Manoi Balrai , Sachinkumar Mogaveera, Deepak Aeloor, December 2018

[10] “Facial Recognition using Convolutional Neural Networks and Implementation on Smart Glasses”, Suleman Khan, M. Hammad Javed, Ehtasham Ahmed, Syed A A Shah, Syed Umaid Ali, July 2019

[11] “FaceNet: A Unified Embedding for Face Recognition and Clustering”, Florian Schroff, Dmitry Kalenichenko, James Philbin, June 2015

# APPENDIX

# Code credit goes to <https://www.youtube.com/watch?v=-53SA35nPR8>

import os

from matplotlib import pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from tensorflow.keras.preprocessing.image import array\_to\_img, img\_to\_array, load\_img

#Setting names of the directories for both sets.

base\_dir = 'dataset'

seta = 'George\_W\_Bush'

setb = 'Colin\_Powell'

#Each of the sets has three sub directories train, validation and test

train\_dir = os.path.join(base\_dir, 'train')

validation\_dir = os.path.join(base\_dir, 'validation')

test\_dir = os.path.join(base\_dir, 'test')

def prepare\_data(base\_dir, seta, setb):

# Takes the directory names for the base directory, and both the sets

# Returns the paths for train, validation for each of the sets.

seta\_train\_dir = os.path.join(train\_dir, seta)

setb\_train\_dir = os.path.join(train\_dir, setb)

seta\_valid\_dir = os.path.join(validation\_dir, seta)

setb\_valid\_dir = os.path.join(validation\_dir, setb)

seta\_train\_fnames = os.listdir(seta\_train\_dir)

setb\_train\_fnames = os.listdir(setb\_train\_dir)

return seta\_train\_dir, setb\_train\_dir, seta\_valid\_dir, setb\_valid\_dir, seta\_train\_fnames, setb\_train\_fnames

seta\_train\_dir, setb\_train\_dir, seta\_valid\_dir, setb\_valid\_dir, seta\_train\_fnames, setb\_train\_fnames = prepare\_data(base\_dir, seta, setb)

seta\_test\_dir = os.path.join(test\_dir, seta)

setb\_test\_dir = os.path.join(test\_dir, setb)

test\_fnames\_seta = os.listdir(seta\_test\_dir)

test\_fnames\_setb = os.listdir(setb\_test\_dir)

datagen = ImageDataGenerator(

rotation\_range = 40,

width\_shift\_range = 0.2,

height\_shift\_range = 0.2,

shear\_range = 0.2,

zoom\_range = 0.2,

horizontal\_flip = True,

fill\_mode = 'nearest')

img\_path = os.path.join(seta\_train\_dir, seta\_train\_fnames[3])

img = load\_img(img\_path, target\_size=(150, 150))

x = img\_to\_array(img)

x = x.reshape((1,) + x.shape)

i = 0

for batch in datagen.flow(x, batch\_size = 1):

plt.figure(i)

imgplot = plt.imshow(array\_to\_img(batch[0]))

i += 1

if i % 5 == 0:

break

# Import Tensorflow Libraries

from tensorflow.keras import layers

from tensorflow.keras import Model

import matplotlib.pyplot as plt

img\_input = layers.Input(shape=(150, 150, 3))

# 2D Convolution Layer with 64 filters of dimension 3x3 and ReLU activation algorithm

x = layers.Conv2D(64, 3, activation='relu')(img\_input)

# 2D Max Pooling Layer

x = layers.MaxPooling2D(2)(x)

# 2D Convolution Layer with 128 filters of dimension 3x3 and ReLU activation algorithm

x = layers.Conv2D(128, 3, activation='relu')(x)

# 2D Max Pooling Layer

x = layers.MaxPooling2D(2)(x)

# 2D Convolution Layer with 256 filters of dimension 3x3 and ReLU activation algorithm

x = layers.Conv2D(256, 3, activation='relu')(x)

# 2D Max Pooling Layer

x = layers.MaxPooling2D(2)(x)

# 2D Convolution Layer with 512 filters of dimension 3x3 and ReLU activation algorithm

x = layers.Conv2D(512, 3, activation='relu')(x)

# 2D Max Pooling Layer

x = layers.MaxPooling2D(2)(x)

# 2D Convolution Layer with 512 filters of dimension 3x3 and ReLU activation algorithm

x = layers.Conv2D(512, 3, activation='relu')(x)

# 2D Max Pooling Layer

x = layers.Flatten()(x)

# Fully Connected Layers and ReLU activation algorithm

x = layers.Dense(4096, activation='relu')(x)

x = layers.Dense(4096, activation='relu')(x)

x = layers.Dense(1000, activation='relu')(x)

# Dropout Layer for optimization

x = layers.Dropout(0.5)(x)

# Fully Connected Layers and sigmoid activation algorithm

output = layers.Dense(1, activation='sigmoid')(x)

model = Model(img\_input, output)

model.summary()

import tensorflow as tf

# Using binnary\_cossentropy as the loss function

# and Adam Optimizer as the optimizing function when training

model.compile(loss='binary\_crossentropy',

optimizer = tf.keras.optimizers.Adam(learning\_rate=0.0005),

metrics = ['acc'])

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255

train\_datagen =ImageDataGenerator(rescale=1./255)

val\_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train\_datagen generator

train\_generator = train\_datagen.flow\_directory(

train\_dir, # Training directory

target\_size = (150,150),

batch\_size = 20,

class\_mode = 'binary')

validation\_generator = val\_datagen.flow\_from\_from\_directory(

validation\_dir, #Validationdirectory

target\_size = (150,150),

batch\_size = 20,

class\_mode = 'binary')

import matplotlib.image as mping

# 4x4 grid

nrows = 5

ncols = 5

pic\_index = 0

# Set up matplotlib fig, and size it to fit 5x5 pics

fig = plt.gcf()

fig.set\_size\_inches(ncols \* 5, nrows \* 5)

pic\_index += 10

next\_seta\_pix = [os.path.join(seta\_train\_dir, fname)

for fname in seta\_train\_fnames[pic\_index-10:pic\_index]]

next\_setb\_pix = [os.path.join(setb\_train\_dir, fname)

for fname in setb\_train\_fnames[pic\_index-10:pic\_index]]

for i, img\_path in enumerate(next\_seta\_pix + next\_setb\_pix):

# Set up subplot; subplot indices start at 1

sp = plt.subplot(nrows, ncols, i + 1)

sp.axis('off') # Don't show axes (or gridlines)

img = mpimg.imread(img\_path)

plt.imshow(img)

plt.show()

#train the model

mymodel = model.fit\_generator(

train\_generator,

steps\_per\_epoch = 10,

epochs = 80,

validation\_data = validation\_generator,

validation\_steps = 7,

verbose = 2)

import numpy as np

import random

from tensorflow.keras.preprocessing.image import img\_to\_array, load\_img

successive\_outputs = [layers.output for layers in model.layers[1:]]

visualization\_model = Model(img\_input, successive\_outputs)

a\_img\_files = [os.path.join(seta\_train\_dir, f) for f in seta\_train\_fnames]

b\_img\_files = [os.path.join(setb\_train\_dir, f) for f in setb\_train\_fnames]

img\_path = random.choice(a\_img\_files + b\_img\_files)

img = load\_img(img\_path, target\_size = (150, 150))

x = img\_to\_array(img)

x = x.reshape((1,) + x.shape)

x /= 255

successive\_feature\_maps = visualization\_model.predict(x)

layer\_names = [layer.name for layer in model.layers]

# Now let's display our representations

for layer\_name, feature\_map in zip(layer\_names, successive\_feature\_maps):

if len(feature\_map.shape) == 4:

# Just do this for the conv / maxpool layers, not the fully-connected layers

n\_features = feature\_map.shape[-1] # number of features in feature map

# The feature map has shape (1, size, size, n\_features)

size = feature\_map.shape[1]

# We will tile our images in this matrix

display\_grid = np.zeros((size, size \* n\_features))

for i in range(n\_features):

# Postprocess the feature to make it bisually platable

x = feature\_map[0, :, :, i]

x -= x.mean()

x /= x.std()

x \*= 64

x += 128

x = np.clip(x, 0, 255).astype('uint8')

# We'll tile each filter into this big horizontal grid

display\_grid[:, i \* size : (i + 1) \* size] = x

# Display the grid

scale = 20. / n\_features

plt.figure(figsize = (scale \* n\_features, scale))

plt.title(layer\_name)

plt.grid(False)

plt.imshow(display\_grid, aspect = 'auto', cmap = 'viridis')

# Accurracy results for each training and validation epoch

acc = mymodel.history['acc']

val\_acc = mymodel.history['val\_acc']

# Loss Results for each training and validation epoch

loss = mymodel.history['loss']

val\_loss = mymodel.history['val\_loss']

epochs = range(len(acc))

# Plot accuracy for each training and validation epoch

plt.plot(epochs, acc)

plt.plot(epochs, val\_acc)

plt.title('Training and validation accuracy')

plt.figure()

# Plot loss for each training and validation epoch

plt.plot(epochs, loss)

plt.plot(apochs, val\_loss)

plt.title('Training and validation loss')

train\_img = random.choice(seta\_train\_fnames)

train\_image\_path = os.path.join(seta\_train\_dir, train\_img)

train\_img = load\_img(train\_image\_path, target\_size = (150, 150))

plt.imshow(train\_img)

train\_img = (np.expand\_dims(train\_img, 0))

print(train\_img.shape)

train\_img = tf.cast(train\_img, tf.float32)

model.predict(train\_img)

train\_img = random.choice(setb\_train\_fnames)

train\_image\_path = os.path.join(setb\_train\_dir, train\_img)

train\_img = load\_img(train\_image\_path, target\_size = (150, 150))

plt.imshow(train\_img)

train\_img = (np.expand\_dims(train\_img, 0))

print(train\_img.shape)

train\_img = tf.cast(train\_img, tf.float32)

model.predict(train\_img)

cal\_cp = 0

cal\_gwb = 0

cal\_unconclusive = 0

alist = []

for fname in test\_fnames\_seta:

if fname.startswith('.'):

continue

file\_path = os.path.join(seta\_test\_dir, fname)

load\_file = load\_img(file\_path, target\_size = (150,150))

load\_file = (np.expand\_dims(load\_file, 0))

load\_file = tf.cast(load\_file, tf.float32)

pred\_img = model.predict(load\_file)

if(pred\_img[0] < 0.5):

cal\_cp += 1

elif(pred\_img[0] > 0.5):

cal\_gwb += 1

else:

print(pred\_img[0], "\n")

cal\_unconclusive += 1

alist.append(file\_path)

print(alist)

print("Identified as: \n")

print("Colin Powell :", cal\_cp)

print("George W Bush:", cal\_gwb)

print("Inconclusive :", cal\_unconclusive)

print("Percentage :",(cal\_gwb/(cal\_gwb + cal\_unconclusive + cal\_cp))\*100)

a = (cal\_gwb/(cal\_gwb + cal\_unconclusive + cal\_cp)) \* 100

cal\_cp = 0

cal\_gwb = 0

cal\_unconclusive = 0

blist = []

for fname in test\_fnames\_setb:

if fname.startswith('.'):

continue

file\_path = os.path.join(setb\_test\_dir, fname)

load\_file = load\_img(file\_path, target\_size = (150,150))

load\_file = (np.expand\_dims(load\_file, 0))

load\_file = tf.cast(load\_file, tf.float32)

pred\_img = model.predict(load\_file)

if(pred\_img[0] < 0.5):

cal\_cp += 1

elif(pred\_img[0] > 0.5):

cal\_gwb += 1

else:

print(pred\_img[0], "\n")

cal\_unconclusive += 1

blist.append(file\_path)

print(blist)

print("Identified as: \n")

print("Colin Powell :", cal\_cp)

print("George W Bush:", cal\_gwb)

print("Inconclusive :", cal\_unconclusive)

print("Percentage :",(cal\_gwb/(cal\_gwb + cal\_unconclusive + cal\_cp))\*100)

b = (cal\_cp/(cal\_gwb + cal\_unconclusive + cal\_cp)) \* 100

avg = (a + b)/2

print("Average Percentage :", avg)

rand\_test\_img = random.choice(test\_fnames\_setb)

rand\_test\_image\_path = os.path.join(setb\_test\_dir, rand\_test\_img)

rand\_test\_img = load\_img(rand\_test\_image\_path, target\_size = (150, 150))

plt.imshow(rand\_test\_img)

rand\_test\_img = (np.expand\_dims(tand\_test\_img, 0))

print(tand\_test\_img.shape)

print("Idetified as:\n")

if(model.predict(train\_img) < 0.5):

print("Collin Powell")

elif(model.predict(train\_img) > 0.5):

print("George W Bush")

else:

print("Inconclusive")

Cikalleshi, Emilian Deep Learning approaches in Face Recognition with Tensorflow 2020