TALLINN UNIVERSITY OF TECHNOLOGY
SCHOOL OF ENGINEERING
Department of Electrical power Engineering and Mechatronics

FACE IDENTIFICATION USING CAPSULE NETWORK WITH SMALL DATA SET

NÄOTUVASTUS VÄIKESE ANDMEHULGAGA KAPSELVÕRGU ABIL

MASTER THESIS

Üliõpilane:	Ahmed Amin Osman
Üliõpilaskood:	165596 MAHM
Juhendaja:	Saleh Ragheb Saleh Alsaleh, Eng,

Tallinn 2020

AUTHOR’S DECLARATION

Hereby I declare, that I have written this thesis independently.
No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

“.......” .... 20…..

Author: ....
/signature /

Thesis is in accordance with terms and requirements

“.......” .... 20….

Supervisor: ….........................
/signature/

Accepted for defence

“.......”....................20… .

Chairman of theses defence commission: ....
/name and signature/

[1] *Non-exclusive Licence for Publication and Reproduction of Graduation Thesis is not valid during the validity period of restriction on access, except the university`s right to reproduce the thesis only for preservation purposes.*

_____________ (*signature*)

_____________ (*date*)

Department of Electrical Power Engineering and Mechatronics

FACE IDENTIFICATION USING CAPSULE NETWORK WITH SMALL DATA-SET

Student: Ahmed Amin Osman, 165596

Study programme, Mechatronics

Supervisor(s): Eng, Saleh Ragheb Saleh Alsaleh, +372 5875 8076

(in English) Face Identification Using Capsule Network With Small Data-Set

(in Estonian) NÄOTUVASTUS VÄIKESE ANDMEHULGAGA KAPSELVÕRGU ABIL

Thesis main objectives:

1. Exploring the potential of Capsule Network on Face Identification
2. Comparative analysis against proven models and networks

Thesis tasks and time schedule:

No	Task description	Deadline
1.	Literature Review	11/25/2019
2.	Building the Capsule model	11/27/2019
3.	Building The CNNs Model and Face-Net Model	11/30/2019
4.	Writing	1/2/2020

Language: English **Deadline for submission of thesis:** 1/3/2020

Student: Ahmed Amin Osman .... "........"...................................2020

/signature/

Supervisor: Eng, Saleh Ragheb Saleh Alsaleh…"................."..........................2020

/signature/

Head of the study programme: Prof.Mart Tamre ……… "......."............2020

/signature/

Terms of thesis closed defence and/or restricted access conditions to be formulated on the reverse side

CONTENTS

PREFACE

The Face identification using Capsul networks work was done at the Robotics Control Laboratory of the mechatronics and autonomous systems centre at the school of engineering at Tallinn university of technology in Tallinn, Estonia.

The research had two academic supervisors, Researcher Saleh Alsaleh And Professor Mart Tamre of Tallinn University of Technology.

The author would like thanks Taltech University for giving him the opportunity to pursue his dream. He also would like to thanks his supervisor Engineer Saleh for being helpful and full of motivation during the time of the thesis. He also wants to thanks professor Mart Tamre for his guidance during the time of this thesis as well as during his studies. Finally, the author would like to thank his family and friends who have been by his side during hard and challenging times.

This thesis explored the potential of using Capsule Network for face identification. Capsule Networks have shown that it could be a potentially efficient method in image classification as it had outperformed CNNs in the classification of the MNIST digit dataset. The aim of this work is to apply this method to a more complex problem of face identification and to benchmark the results to other proven methods.

The author benchmarked the performance of the Capsule Network algorithm in comparison with two other methods (FACE-NET and CNN). The classification accuracy was tested on a small dataset sampled from the (Labelled Faces in the wild and Faces94) datasets.

Caps-net has shown better results in comparison with the CNNs model when both models were trained using a small data-set, however, still using a pre-trained model out-performed both models in the case of FaceNet.

Keywords: Capsule Networks, Convolutional Neural Networks, FACE-NET, Image Classification, face identification, master thesis

List of abbreviations and symbols

CNNs	Convolutional Neural Network
SVM	Support Vector Machines
ReLU	Rectified Linear Unit
nan	Not a Number

INTRODUCTION

In recent years many advancements in face identification methods have been made by researchers due to the vast amount of commercial applications especially in critical security-related areas such as Law enforcement and security access control, with the combination of the available big data and low-cost computational power machine learning methods have matured to a degree where we see their use in face identification in our daily lives.

While the state of the art machine learning classification methods such as convolutional neural networks (CNN) are being widely used for face identification they still inherit some limitations and researchers are trying to address, One of the main limation of CNN is that it doesn't capture the relation between the features of a mage what this translate to that if a CNN model was trained on a specific image and later we test the model on a rotated copy of that image CNN will fail to correctly classify the image, this will require a training the CNN on different rotate it copy of the same image which means the requirement of huge amount of data to get a well-trained model.

in 2017 Hinton et al published Dynamic routing between capsules [1] which suggest a new method to overcome CNN's limitations where "A capsule is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an object part " this method is called CapsNet.

The CapsNet doesn't just extract and learn information about the features of an image it also learns the relationship between these features which in theory should result in a more accurate model, This was shown to be true when running on the model on the MNIST digit dataset where the CapsNet model outperformed the state of the art CNN models.

this promising result is encouraging to try and apply CapsNet on more complex problems such as image identification, Although the method is recent and has not matured enough the aim of this work is to try and answer the following questions:

- Can CapsNet outperform CNN in the face identification application when trained on both models are trained using a small data-set?
- How does CapsNet perform compared to a more mature and pre-trained model?

While in [2] there CapsNet on has been applied on face identification it didn't take into consideration the different levels of maturity of the methods being compared. If using a large data set we can see the advantage of using CaspNet is not so clear, however, the author argues that CapsNet could be benefited from in cases where only a small dataset is available for training which is one of the limitations of CNN.

So in this work, we will train and evaluate the performance of CapsNet compared to CNN using a small dataset, which should give us some insight to answer the first question and finally we will evaluate the CapsNet performance compared to a pre-trained model to see if it still will outperform a more mature pre-trained model.

The structure is divided into seven chapters where chapter 1 will provide a literature review on related work. Subsequently, chapter 2 will describe the data-set and libraries used in the implantation of the work.

Chapters 3,4 and 5 cover the implantation part of the work where the three different methods, CapsNet, CNN and Facenet are detailed. The theory, architecture, and results of these methods are described with the use of graphs and tables to summarize the results.

In chapter 6 we will analyze and compare the results of the three methods and try to give answers to the main questions behind this work. Finally, in chapter 7 a summary of the work is represented.

Problem Statement

Convolutional Neural Networks (CNNs) dominate computer vision-related problems, and are the most effective in solving and performing, image classification, image identification and image recognition, from simple objects to complex objects. However, CNNs fail in some important areas, they don't store the orientation and position and only recognize features rather than taking into account the spatial relationship between the features.

So far CapsNet was tested on small datasets such as the MNIST (Mixed National of Standards and Technology) dataset, this dataset is a handwritten digit used for training image processing system, some call it the 'Hello World' of computer vision algorithms.

By small dataset we mean, the number of classes and the pixel size. This thesis on Face identification using CapsNet with small dataset was motivated by the weaknesses of CNNs and that this algorithm had not been tested on larger datasets. Therefore we decided to test the CapsNet on Face identification problem with small (Limited Labels per Class) dataset and to train traditional CNNs and FaceNet model as a benchmark for analysing the performance and the potential of CapsNet.

Contribution

To run an unsupported dataset on the Capsule Network, we used [3] as an example.
As we have expected, machine learning platforms don't support the capsNet algorithm yet. To get our model running we had to take the following steps.

- Writing a TensorFlow input pipeline:
- Making communication channels for the dataset
- Writing a squashing function to ensure the length of vectors to remain between 0 and 1
- Changing the architecture to suit our tasks, such as the number of capsules in the primary layer and the CapFace layer.

1 LITERATURE REVIEW

1.1 Human vision

For a human being to recognizing and perceiving faces is a vital must have the ability to coexist society. By perceiving faces we can distinguish and tell identity, mood, sex, and the race of an individual and thereby take a suitable decision. Our visual system can recognize and identify thousands of faces learned during our lifetime especially known and familiar faces by just a glance, even after a long time of separation. Human visual system mastered this skill even if there are big changes caused by viewing conditions, ageing, hairstyle and distractions caused by wearing eye-glasses, sun-glasses, hats or even wearing make-up [4].

Humans are exceptional at perceiving faces, part of the reason for this is a specialized brain region called the Fusiform Face Area (FFA). The FFA is tiny and is located in the brain’s temporal cortex, which is generally responsible for object recognition.

1.2 Computer vision

Generally, we make and invent all types of technology to ease and speed up our daily tasks and needs, mostly, in such a way that it serves us as another human would. Taking that into account computer vision is the branch of science that deals with vision and mimics human visual system. We see things through the eyes with the help of light and then the brain processes the object and gives us a clue about what the object is based on previous experience.
Larry Roberts from MIT whom a lot of people regard him as the father of computer vision proposed [5] the possibility of deducing 3-D geometrical information from a 2-D perspective view of blocks in his PhD.
The first digital image was taken in 1959 when Russel kirsch and his colleagues had devised an instrument that can transform images into grids of binary numbers, a language computer could understand. Thanks to Russel and his team what we now can process, manipulate images to meet our needs so easily. Figure 1.1 shows the first-ever digitally scanned image [6].

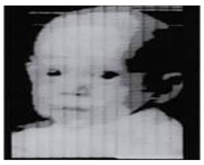


Figure 1.1 Kirsch's three months old song

1.3 Face detection

Face detection is the first step of face recognition and there are different methods and each of these methods has its own advantages and disadvantages. Alignment and detection of faces are crucial to many face applications, such as face recognition and facial expression. Some of the popular methods are.

1.3.1 Robust real-time object detection

In 2001, Viola and Jones published [7] a breakthrough robust real-time object detection algorithm. This paper presents a visual object detection algorithm that can process images extremely fast while maintaining high detection rates. In this algorithm, there are three key important elements. The first is the introduction of "Integral Images" which computes the detected features super-fast. The second one is based on Adaboost and it selects the key features and the third method combines classifiers in a "cascade" which spends more time on promising features of the object.

1.3.1.1 features

Objects are classified based on the value of simple features. The authors have used [7] three kinds of features and they are as follows.

- Two rectangle features

The value of the two rectangle features is the difference between the sum of pixels within two rectangle regions, note that the regions have the same shape and size and are vertically or horizontally adjacent as shown in figure 1.2 [7]

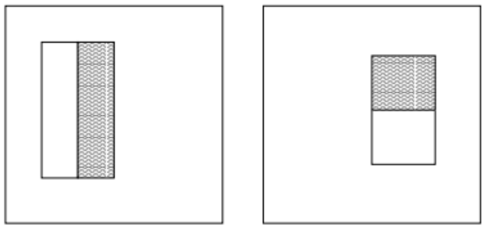


Figure 1.2 Two rectangular feature

- Three rectangle features

The three rectangle feature computes the sum within the two outside rectangles subtracted from the sum in the centre rectangle [7]

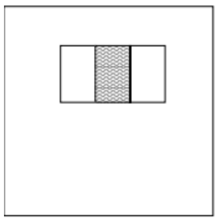


Figure 1.3 3 three rectangle feature

- Four rectangle features

The four rectangle feature computes the difference between diagonal pairs of rectangles [7]

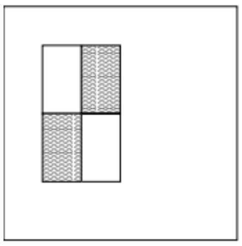


Figure 1.4 four rectangle feature

1.3.1.2 Integral images

Features of the targeted object are analysed very rapidly by using an intermediate rendition for the targeted object which the authors call it the integral image. The integral image located at x, y is the addition of pixels above and to the left of x, y.

$$ii(x,y) = \sum_{x' \leq y' \leq y} i(x',y') \tag{1.1}$$

Where $ii(x,y)$ is the output features (Integral image) and $i(x,y)$ is the input image (Original image)

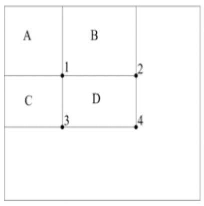
$$s(x,y) = \mathrm{s(x,y-1)} + i(x,y) \tag{2.2}$$

$$ii(x,y) = ii(\mathrm{x-1,y}) + s(x,y) \tag{3.3}$$

s (x, y) is the accumulative raw sum, note that s (x, -1) = 0, and ii (-1, y) = 0. We can calculate the integral image in one pass over the original image [7].

Any rectangular sum can be calculated in four array references using the integral imag e, see Figure 1-5.

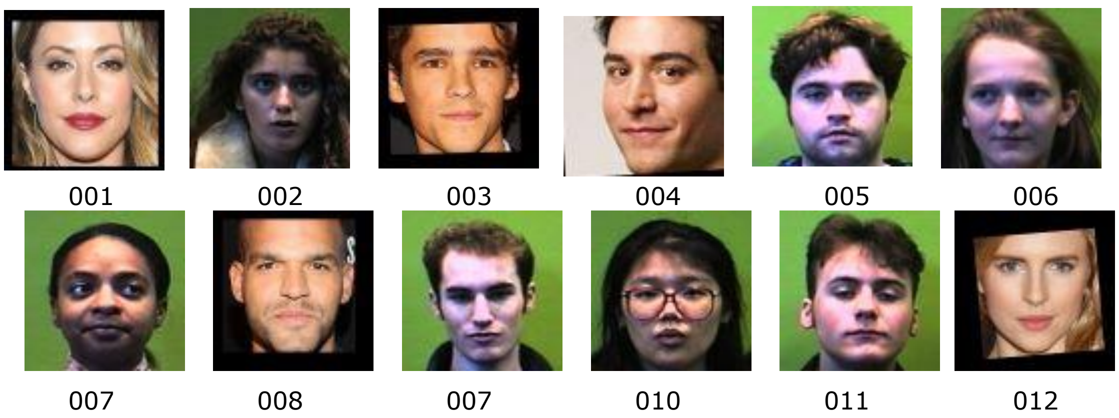
Four array references can be used to calculate the number of pixels within rectangle D. The integral image value at location 1 is the sum of rectangle A pixels. Location 2 value is A+B, location 3 value is A+C, location 4 value is A+B+C+D.



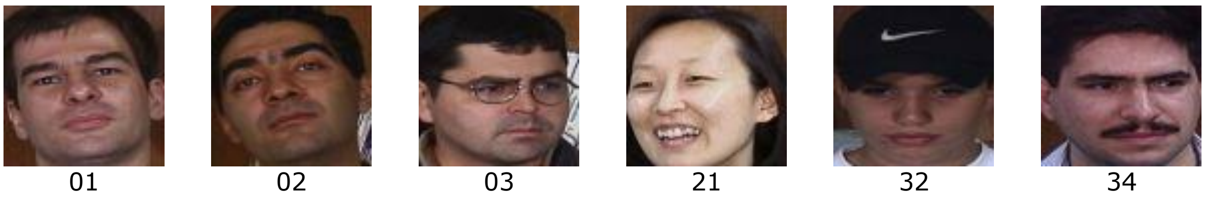
2 DATA-SET AND LIBRARIES AND HARDWARE

A mixture of LFW (labelled faces in the wild) a public data set for face verification created by professor Erik learned miller et al and Faces94 [8] were used for training the networks namely, capsule network, convolutional neural network and face-Net. Labelled Faces in the wild [9].

Table 2.1 Classes of the first training data-set



After the first training was done, we wanted to experiment how well the capsNet will perform on a larger data-set.



For the second training, we used a data set called Georgia Tech Face Database [10].

Table 2.2 Data-set Description

Method	**Data-set**			**Number of Classes**		**Images per class**	**Total**
1.	LFW	+	Faces94	5	7	10	120
2.	Georgia Tech Face Database			50		15	750

2.1 Platforms and libraries

Anaconda is a platform which comes with different machine learning libraries like python, Jupyter notebook to develop programs quickly. It takes out the complex and time taking processes to easy and quick solutions. Anaconda is used in this project to spend more time on solving the problems rather than figuring out about libraries versions and their compatibility. Following libraries are mainly used in this thesis project.

1. OpenCV: OpenCV is an open-source computer vision library which contains useful functions for computer vision applications in a simple language for image processing. In this thesis, OpenCV is used for reading input dataset from the path and resizing images to the required dimensions and visualizing samples of the dataset to make sure that the dataset is correctly fed into the network.
2. Python Imaging Library: PIL or later known as pillow
3. Matplotlib: Matplotlib is a plotting library that is included python packages and it's used for plotting graphs.
4. Numeric python: NumPy packages are used for scientific computing and storing data in the required format for such a way that computational becomes more efficient.
5. OS library: This package is used for communicating with the different directories in the computer
6. TensorFlow: Developed by Google and it's mainly used to processes multidimensional arrays which are also known as tensors.
7. Keras: is an API neural network that is written in python and it runs on top of TensorFlow, CNTK (Microsoft Cognitive Toolkit) or Theano.

2.1.1 Hardware and Software

2.1.1.1 Hardware

For the training, we utilised two computers at the Robotics Control Laboratory of the mechatronics and autonomous systems centre at the school of engineering at Tallinn university of technology in Tallinn, Estonia.

For training the CNNs and the Face-Net, we used desktop with the following properties.

Table 2.3 Specifications for Desktop 1

Operating System	Processor	Installed Memory	Graphics
Windows 10 64-bit	Intel, i5, CPU 3.30GH$_z$	8GB	Nvidia Quadro 600

For training the Capsul Network, we used a desktop with the following properties

Operating System	Processor	Installed Memory	Graphics
Windows 10 64-bit	Intel, i5, CPU 3.30GH$_z$	32GB	Nvidia Quadro 600

2.2 Data-set pre-processing for CAPS-Net

As CapsNet is still a recent method it requires pre-processing of the data which unlike the FaceNet and CNN which have matured enough that most machine learning libraries are able to automatically pre-process the data, Every machine learning project starts with gathering training examples and then pre-processing that dataset. The following steps have been taken prior to building the network

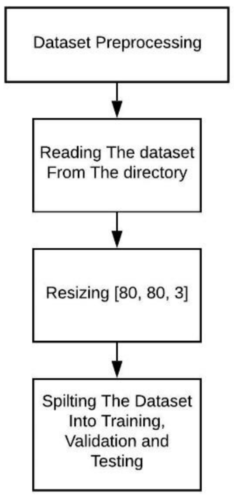

Dataset Preprocessing
Reading The dataset From The directory
Resizing [80, 80, 3]
Spilting The Dataset Into Training, Validation and Testing

Figure 2.1 Data-set pre-processing flow chart

So far CAPSNET was tested on small data-set images, such as CIFAR and MNIST. Due to the fact that this network requires a high computational power machine even for small data-set in terms of pixel size. Different pixel sizes were tested for the network, some of which are smaller than 80*80 and some of them greater than 80*80, after testing we concluded that the optimal size was 80*80.

3 CAPS-NET

3.1 Capsule neural network

3.1.1 Applications

Capsule Network is one of the newest additions in the field of machine learning. The Capsule Network is still in its infant, research and development phase, as a result, there are no commercial applications that are based on the Capsule Network yet. Although this algorithm is in its testing phase, however, it has shown some advantages over the traditional neural networks [1].

Because of high sensitivity in real life and data limitations, self-driving cars is one of the possible application areas for capsule neural network. The capsule network model could be handy for application where the dataset is limited [11].

In 2018 a group of Canadian researchers used capsule network for classifying brain tumours [12] and they have compared the results of three models. And their results show that capsNet is truly becoming a trend in the field of image classification. Table 1.1 shows the results of the experiment in comparison with CNN. One big advantage of the CapsNet in this experiment is the elimination of human-annotated images. In this experiment, they have used small dataset images generated from MRI (Magnetic Resonance Imaging) scan. This experiment supports the strength of CapsNet in a small dataset.

Table 3.1 Comparison betweenCapsNet approach and CNN results

	Approach	**Accuracy**
1.	CapsNet given brain image as input	78%
2.	CapsNet given segmented tumour as input as input	86.56%
3.	Proposed CapNet Architecture	**90.89%**
4.	CNN given brain image as input	61.97%
5.	CNN given segmented tumour as input as input	72.13%
6.	Modified CNN with brain image and tumour Boundry box as input	88.33%

3.2 Theory

"A capsule is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an object part " [1]

In 2011 Geoffrey E.Hinton published Transforming auto-encoders [13] that introduced many of the key ideas of capsule networks, however, the author had a hard time making them work properly until 2017 when Hinton et al published Dynamic routing between capsules [1]. the authors managed to reach the state-of-the-art performance on the MNIST dataset and demonstrated considerably better results than the traditional convolutional neural networks on highly overlapping digits.

In computer graphics, we start with an abstract of representation of an object and we call some rendering function and then we get an image. Each object or object part has some instantiation parameters and we call some functions then we get an image as illustrated in figure 3.1

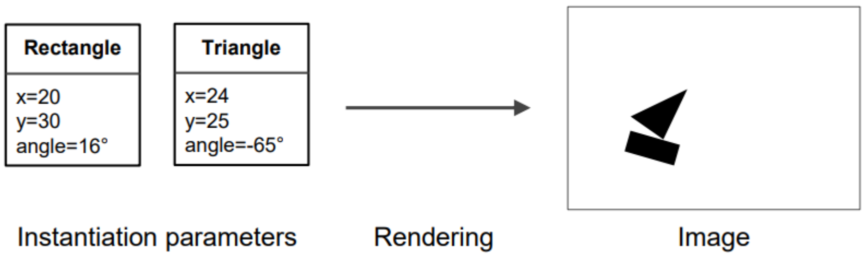


Figure 3.1 Computer graphics illustration

In capsule networks, we start with an image and it finds what objects it contains and what their instantiation parameters are. Basically, a capsule network is a neural network that performs inverse graphics.

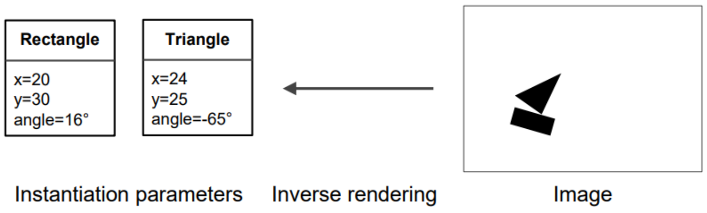


Figure 3.2 Inverse graphics

Capsules try to predict the presence and the instantiation parameters of an object at a given location.

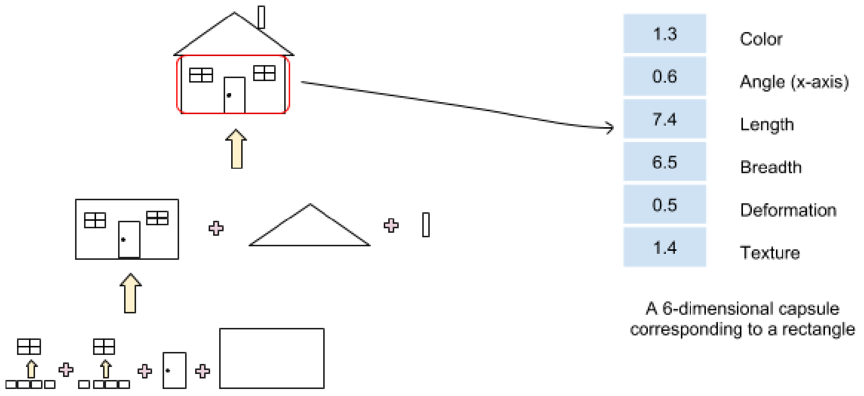


Figure 3.3 Example of capsule network

Capsule networks look for whether specific objects are present in the target image and accordingly there is a capsule which returns.

- The probability an entity exists
- The instantiation parameters of the detected entities.

In figure 3.3 our imaginary capsule is made-up of 6 neurons each corresponding to certain area/part of the rectangle. The probability of the presence of the rectangle is determined by the length of the vector. So, in figure 3.3 the probability of a rectangle being present will be:

$$\sqrt{1 \cdot 3^2 + 0.6^2 + 7 \cdot 4^2 + 6 \cdot 5^2 + 0.5^2 + 1 \cdot 4^2} = 10.06$$

Something is not right, if the output vector represents the probability of an entity exists, it should be less than or equal to 1 ($0 \leq P \geq 1$) and this is where the squashing function comes alive.

The squashing function is used to make sure that short vectors get shortened to nearly zero length and long vectors get shortened to a length below 1.

$$V_j = \frac{\|s_j\|^2}{1+\|s_j\|^2} \frac{s_j}{\|s_j\|} \qquad 3.0$$

Where V_j is the resulting vector of capsule j

S_j is its total input. Except for the initial layer of the capsules,

3.3 The intuition behind CapsNet

The implementation of capsNet is divided into three main parts.

First, we will construct the primary capsules layer which consists of three subdivisions.

- Convolution
- Reshape
- Squash

Secondly, we will build the Higher layer capsules which primarily incorporates the.

- Routing by agreement
-

Lastly, we will perform loss calculations.

- Margin Loss
- Reconstructions

3.3.1 Primary capsules

To understand the theory behind CapsNet we will show an illustrative example. In the first layer, we start the process of inverse graphics. Suppose we want to classify boat image and a house image shown in figure 3.4[14]. in the first layer, these images are broken down into their subparts.

Figure 3.4 Left house, Right boat

These images consist of triangular shape and rectangular shape as shown in figure 3.5[15]. In the primary capsule layer, the representing capsules of the triangle and the rectangle are constructed. Let' say that we assign 100 capsules, where 50 capsules are representing the triangle and 50 representing the rectangle.

In figure 3.5[16] black arrows are representing the output capsule for the rectangle and the blue arrow is representing the output capsule for the triangle. These arrows are placed every location of the image and they are indicating the presence of an object in a specific location.

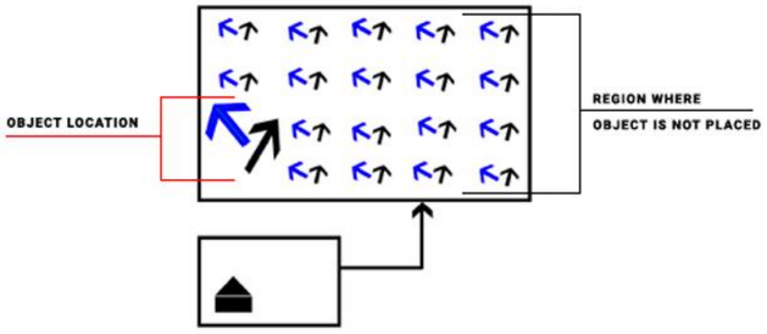


Figure 3.5 representation output capsules

The length of the arrow is shorter where the object is not placed, and longer where it is placed. The orientation of the arrow shows the position and the scale in the given image.

To achieve the representation output capsules, we feed the image two convolutional layers, and this will output some array of feature maps, suppose this outputs an array of 18 feature maps, and then we reshape these feature maps into two vectors of 9x9 for all the locations in the image.

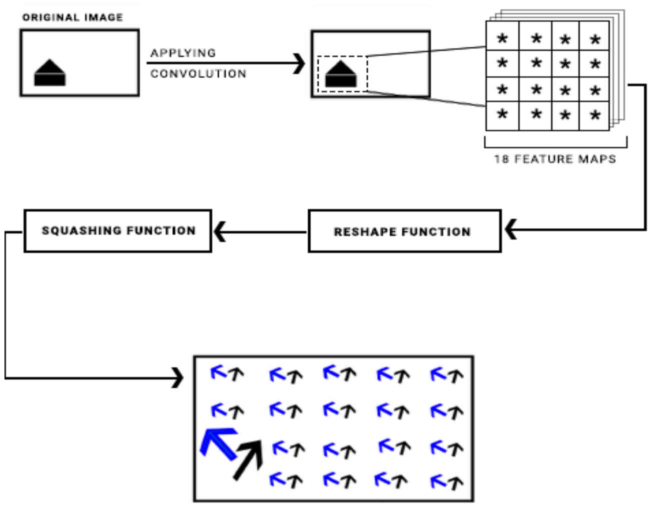


Figure 3.6 Processing steps of the input image

The length of the arrows represent the probability of an object about a specific location, therefore we ensure that the length is always between 0 and 1. To achieve this we use the squashing function shown in equation 3.1.

$$V_j = \frac{\|s_j\|^2}{1+\|s_j\|^2} \frac{s_j}{\|s_j\|} \qquad 3.1$$

Now we need to figure out how these subparts are related to our example of the boat and the house. And which parts do belong to the triangle and the house

3.3.2 Higher layer capsules

Right after the squashing function in the primary capsule layer and before the higher layer capsule, every capsule in the primary layer will make predictions for every capsule in the higher layer capsules. In our example, we have two predictions to make, the boat and the house 100 capsules each. These capsules will make predictions depending upon the orientation of both classes. Figure 3.7 [17] illustrates this scenario.

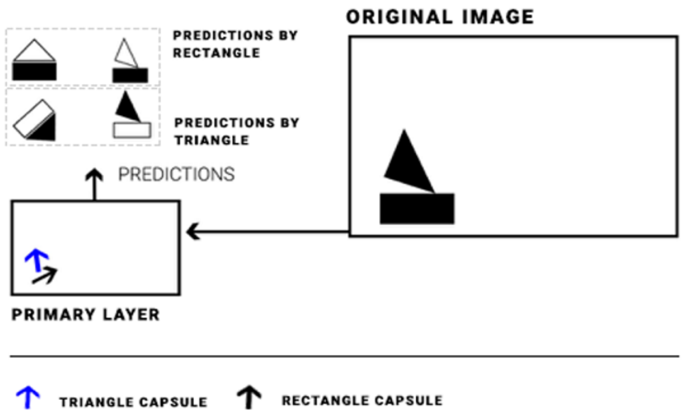


Figure 3.7 Predictions made by the primary layer

As figure 3.7 shows, both the triangle capsule and the rectangle predicted the presence of a boat in the given image.

In figure 3.8 [18] both the triangle and rectangle are democratic and have agreed that the input image is more likely a boat. This process of agreeing is called routing by agreement.

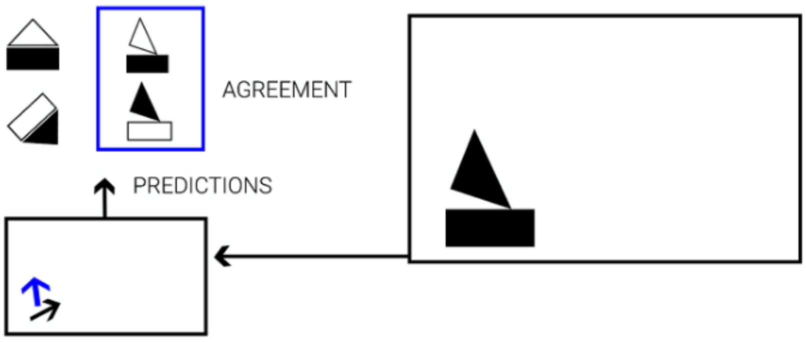


Figure 3.8 Routing By agreement illustration

So far the higher layer has done nothing much, all the work was done by the primary layer. Now, the higher layer will need to compute its outputs and to check it's predictions with its computations.

For the higher layer to come up with its own outputs, it performs something called routing weights. At the moment the higher layer has the predictions made by the primary layer, for the first iteration it sets its routing weights to be zero for all and then we feed these weights into a softmax function. This process is shown in figure 3.9 [19]

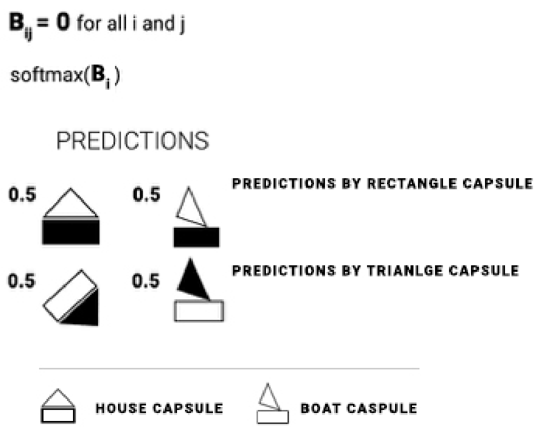

B_{ij} = 0 for all i and j
softmax(B_i)
PREDICTIONS
0.5
0.5
PREDICTIONS BY RECTANGLE CAPSULE
0.5
0.5
PREDICTIONS BY TRIANLGE CAPSULE
HOUSE CAPSULE
BOAT CASPULE

Figure 3.9 Weight initialization

First, we defined two regular convolutional layers and then we fed the dataset. Every image in the dataset will be broken into small subparts in this layer. Capsules in this layer are placed in every location of the face images and the output vector of this capsules carries features that are located in the faces, these features could be a specific feature and the pose of the arrow represents the orientation of that specific feature in relation to other features. These arrows locate features in the face with its precise location. This is achieved by implementing three, Convolution layers, Reshaping the output function and then squashing by using equation 3.0 with small Epsilon value to avoid zero division, the shaped function to ensure that the length of each vector lies between 1 and 0, where 1 is indicating the presence of features in the given face image/images, and the 0 indicates that no detected features in that specific of the images.
Now vectors carry features locations but we don’t know which features belong to which class, and label. To know the relation between features and labels, we will build the caps -face(Higher Layer Capsules).

3.3.3 Caps-face

Before the caps-face becomes functional, every function in the primary layer will output predictions for every capsule in the caps-face layer. The number of capsules in the caps-face layer is determined by the number of classes that are needed to be predicted. Capsules in this layer will then make predictions about the classes based on their features orientation, but it has to agree with the predictions made by the primary capsules, and this method is where the name routing by agreement comes from. Basically, the predictions in the caps-face should match the predictions of the primary capsules layer.

This method has several advantages, there is no need to send a signal to another capsule if the primary layer agreed to select a capsule in the caps-face layer and therefore will be made stronger and clearer for the prediction. We will need to set up routing weights for the caps-face to calculate its own output. For this, all the predictions given by the primary layer will be set to be zero. These weights are fed into a softmax function and the output is assigned to each prediction.

For each face, in the batch, the primary layer will output 32 maps each containing, 32x32 grid of 8-dimensional vectors.

Capsule 1 Number of Maps	32
Capsule 1 Number of Caps	Capsule 1 Number of Maps * 32 * 32 = 32768
Capsule 1 Dims	8

The model structure mainly consists of six main parts as shown in figure 2-1. CNNs use deduced copies of learned feature detectors and this allows them to carry knowledge about good weights extracted from one position in an image to other [1]. This has proven to be very useful in image processing.
Although the authors are replacing max-pooling with routing by agreement and replacing scalar output feature detectors of CNNs with vector output capsule. However, authors still wanted to keep a copy of the learned features across space and to achieve this, all the layers except the capsule layer are CNNs layers. For this case, two convolutional layers are used to extract features.

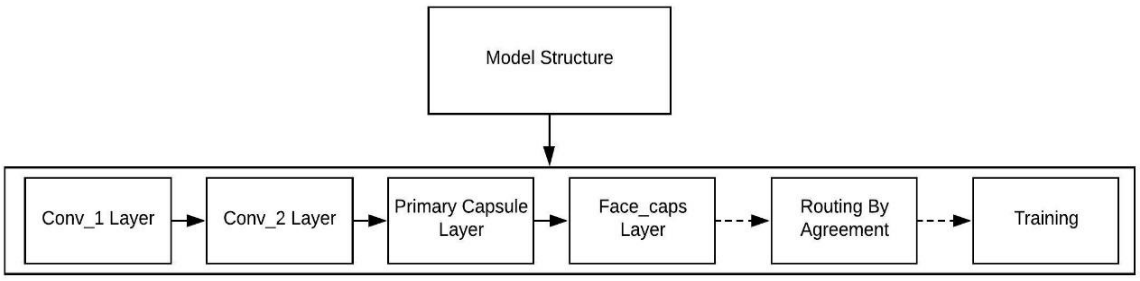


Figure 3.10 Network flow chart

3.3.4 Activation for CNNs layer

In neural networks, activation functions are used for mapping the output of nodes, usually, the resulting value is either 1 or 0, depending on the type of the activation function. Mainly activation functions are divided into two categories:

- Linear Activation function

  The output of f(x) = x function is unlimited can be any range between [-∞ to ∞], figure 3.5 [20] illustrates. This is not a very useful function for complex and real data that is needed to feed to the neural network.

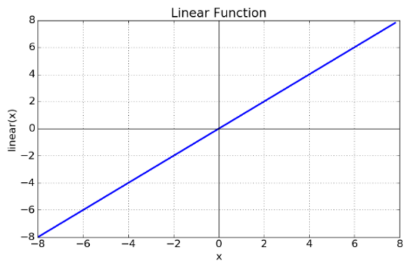


Figure 3.11 linear function

- Non-linear activation Function

  Non-linear [21] activation functions are the most widely used activation functions due to the fact that they make for the model easily to generalize and adapt different datatypes and differentiating between outputs.

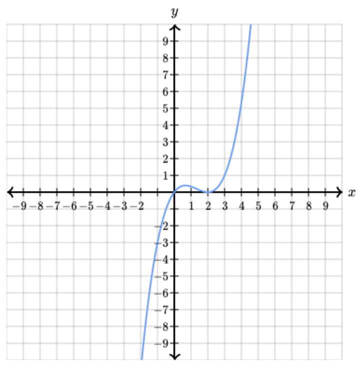


Figure 3.12 Non-linear function

In neural networks, we use activation functions to add our model non-linearity so the network can map more complex patterns and relationships in the given data. The most common non-linear activation functions are ReLU, Sigmoid and tanh.

3.3.5 ReLU activation function

The rectified linear unit, in recent years, has become one of the most popular if not the most popular activations function.

Equation 2,0 returns 0 for all the values that are smaller than 0, and it returns x for the values equal to or greater than 0. Figure 3.7 shows the graphical representation of ReLU activation function

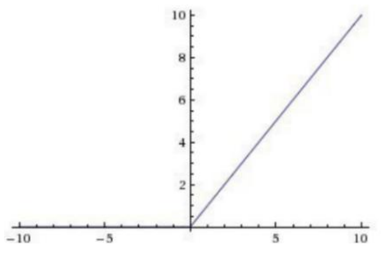


Figure 3.13 ReLU activation function

$$F(x)\begin{cases} 0\, for\; x < 0 \\ x\, for\; x => 0 \end{cases} \qquad 3.1$$

3.3.6 Loading data set

Every machine learning project starts with gathering training examples and then pre-processing that dataset. The following steps have been taken prior to building the network

The first step was getting a dataset to work with and then reading it from the working directory. Due to the limitations of capsule networks, it was decided that to use reasonable size for the network, which has been so far tested on small datasets in terms of the pixel size.

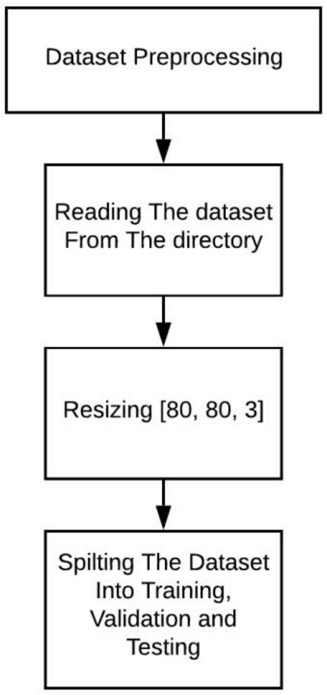

Dataset Preprocessing
Reading The dataset From The directory
Resizing [80, 80, 3]
Spilting The Dataset Into Training, Validation and Testing

Figure 3.14 Data set loading steps

Since capsule networks can be a substitute for conventional CNNs, it was decided to use quit limited dataset. One drawback of CNNs is that it requires very large input data in order to get reasonable results. However

3.4 Primary capsule layer

The primary layer which is the first layer is composed of 16 maps of 32x32 capsules each, therefore, there are 32768 primary capsules.

Table 3.2 Parameter of the primary capsule layer

Capsule 1 Number of Maps	32
Capsule 1 Number of Caps	Capsule 1 Number of Maps * 32 * 32 = 32768
Capsule 1 Dims	8

To compute the output of these primary capsules in the first layer, two conventional convolutional layers were used with the following parameters in table 3.2

Table 3.3 CNNs Parameters

Conv_1 Parameters		Conv_2 Parameters	
Filters	256	Filters	256
Kernal Size	9	Kernal Size	9
Strides	1	Strides	2
Padding	Valid	Padding	Valid
Activation	ReLU	Activation	ReLU

Since we used kernel size of 9 and padding was Valid (no padding) and stride of 2, our input images got shrunk to 32x32 pixels that are because after each convolutional layer the input images loses 8 pixels (Kernal size-1=8) and stride of 2 in the second convolutional layer so the image size was divided by 2. Therefore, we end up with 32x32 feature maps.

The next step is to reshape the output of the primary capsules to get a bunch of 8D vectors. Conv2 outputs an array of 16x8 feature maps for each instance with each feature maps of 32x32, so we end up shape of (batch size, 32,32,80), let's not forget that the first layer is fully connected to the primary capsule layer, so we need to reshape it to (batch size, 32x32x16, 8).

We need to squash these vectors based on the squash function in equation 3.0 to ensure that the length of vectors is always between 0 and 1.

Equation 3.0 will squash every vector in the given array along the given axis. We know that the derivative of $\|S_j\|^2$ is undefined if $\|S_j\|^2 = 0$, so we can't directly get norms of the vectors. Therefore, we need to get the norm manually (Custom made function: in the notebook) by calculating the square root of the sum squares plus a tiny number (0.00001) so we don't get nan error.

Now we are ready to get every output of the primary capsule by applying the squash function in equation 3.0

3.5 Face capsule

In order to compute the output of the Face capsules, we need to get the predicted output vectors one for each primary capsule, then we can implement routing by agreement.

Table 3.4 Face capsule parameters

Capsule 2 Number of Caps	12
Capsule 1 Dims	16

The number of caps of capsule 2 indicates the total number of labels we want to predict as shown in table 3.3

We need to predict the output of each capsule in the second layer for each capsule in the first layer.

$$s_j = \sum_i c_{ij}\, \hat{u}_i|_j \quad \hat{u}_i|_j \ = W_{i\ j} u_i \tag{3.2}$$

To achieve this we need a transformation matrix Wi, j for the primary capsules and the face capsules, we can calculate the predicted output as equation 3.2 shows. We want to transform Capsule_1 Dims (8D vector) into 16D, therefore, we need each transformation matrix to have a shape of (16, 8).

To calculate $\hat{u}_i|_j$ for every pair of the primary capsule and the face capsule, we need to multiply their arrays, and TensorFlow function [22] is used which lets us multiply two matrices and as well higher-dimensional array.

the primary capsule layer has a shape of (?, 32768, 12, 8, 1) and capsule face layer has a shape of (?, 32768, 12, 16, 1). For the second layer, we need to create 12 copies of the first layer (32768), to achieve this a handy TensorFlow function [23] is used which allows us to copy arrays and then create a new array in any shape we want. Now we only need to multiply these two arrays.

3.6 Routing by agreement

Activation vectors of the Face cap send feedback signals to the capsules at the primary capsule layer. If the prediction vectors of the capsule of the primary layer and the capsules of the face cap layer are in agreement, their dot product should be high.

First, we initialize weights of the routing b_{ij} to zero. We need to multiply routing weights and caps_2 predictions in an elementwise matrix multiplication way and they should have the same rank. Therefore we add two extra dimensions of size 1 to the routing weights. To get the outputs of the second layer capsules and we matched the rank of routing weights and caps_2 predictions, we can apply the squashing function

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} \quad 3.3$$

b_{ij} indicates whether capsules in the primary layer and capsules in the face cap have strong coupling.

3.7 Results

3.7.1 Routing iteration (1)

Figure 3.15 shows three predicted classes label 001, label 002, label 001 was misclassified for label 012, and label 004 got a high prediction rate than the correct label. Classifier got right all the examles of the label 002.

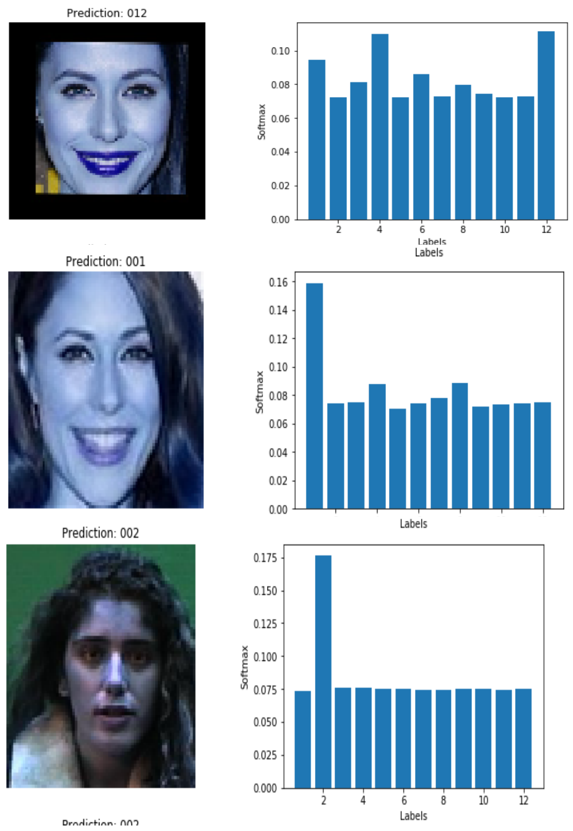


Figure 3.15 Example of predicted classes

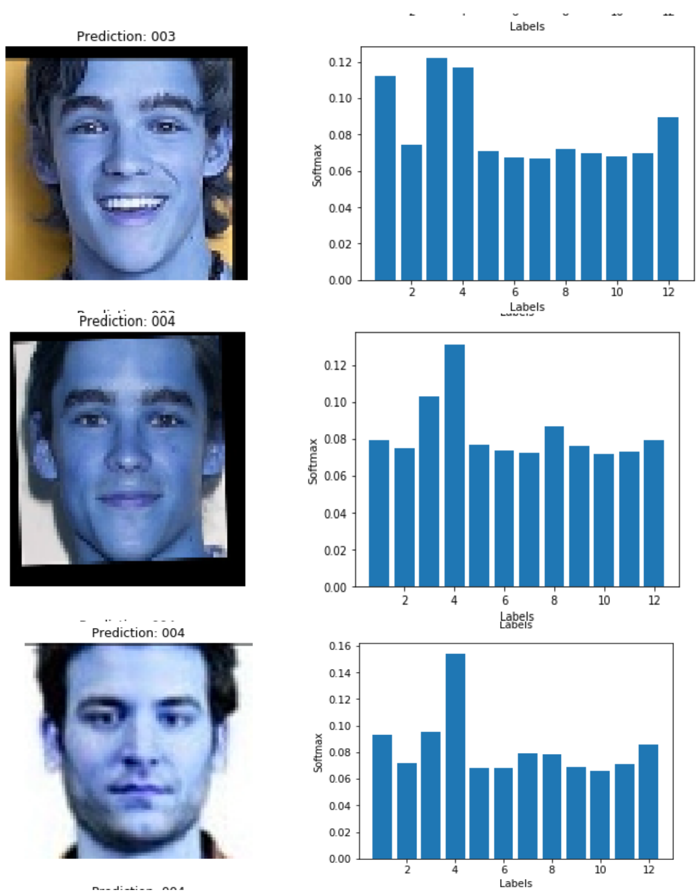


Figure 3.16 Mixed Label predictions

Table 3.5 Results

Final training Accuracy	Final validation Accuracy	Test Accuracy	Test Loss
100%	90%	70%	0.259

Table 3.6 Summary of test with 1 iteration

Label	Correctly Classified	Miss-Classified	Success percentage
001	5/6	1	83.300 %
002	6/6	0	100 %
003	3/6	3	50 %
004	6/6	0	100 %
005	6/6	0	100 %
006	6/6	0	100 %
007	6/6	0	100 %
008	3/6	3	50 %
009	6/6	0	100 %
010	6/6	0	100 %
011	6/6	0	100 %
012	5/6	1	83.300 %
Total	64	8	88.900 %

3.7.2 Routing iteration (2)

For the final test 72 images were used for testing, 6 images per class. The following are the results of each class. Due to the scale of the test set, only a few of them will be presented here.

Table 3.7 Results

Final Training Accuracy	Final Validation Accuracy	Test Accuracy	Test Loss
100%	87.500%	90%	0.083

Table 3.8 Summary of test with 2 iterations

Label	Correctly Classified	Miss-Classified	Success Percentage
001	6/6	0	100 %
002	6/6	0	100 %
003	4/6	2	66.670 %
004	3/6	3	50 %
005	6/6	0	100 %
006	6/6	0	100 %
007	6/6	0	100 %
008	3/6	3	50 %
009	6/6	0	100 %
010	6/6	0	100 %
011	6/6	0	100 %
012	5/6	1	83.300 %
Total	63	9	87.500 %

3.8 Experimenting with larger data-set

In this section, we will train the CapsNet with a larger data-set. Taking account of the results of the previous section, we decided to train the network with just one routing iteration. Because our model performed well on the training part with two routing iterations but underperform the test phase than the model with just one routing iteration. Also, the training time was longer with the 2 routing iterations.

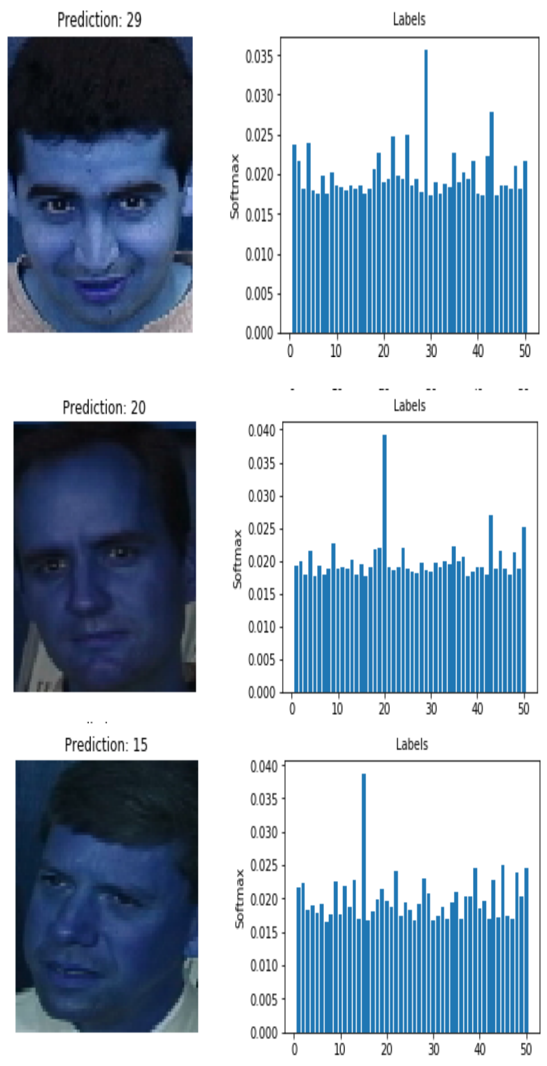


Figure 3.17 Three correctly Classified Classes

Table 3.9 Training results

Final Training Accuracy	Final Validation Accuracy		Test Accuracy	Test Loss
100%	100%		0.9380	0.115

Table 3.10 Results

Label	Correctly Classified	Miss-Classified	Success percentage
01	1/2	1	50%
02	2/2	0	100%
03	2/2	0	100%
04	2/2	0	100%
05	2/2	0	100%
06	2/2	0	100%
07	2/2	0	100%
08	0	2	0%
09	1/2	1	50%
10	2/2	0	100%
11	1/2	1	50%
12	2/2	0	100%
13	1/2	1	50%
14	0	2	0%
15	2/2	0	100%
16	0	2	0%
17	1/2	1	50%
18	1/2	1	50%
19	2/2	0	100%
20	2/2	0	100%
21	2/2	0	100%
22	2/2	0	100%
23	2/2	0	100%
24	2/2	0	100%
25	2/2	0	100%
26	2/2	0	100%
27	2/2	0	100%
28	2/2	0	100%
29	2/2	0	100%
30	2/2	0	100%
31	2/2	0	100%
32	2/2	0	100%
33	2/2	0	100%
34	0	2	0%
35	2/2	0	100%
36	2/2	0	100%
37	2/2	0	100%
38	2/2	0	100%
39	2/2	0	100%
40	2/2	0	100%
41	0	2	0%
42	2/2	0	100%
43	2/2	0	100%
44	2/2	0	100%
45	2/2	0	100%
46	0	2	0%
47	2/2	0	100%
48	2/2	0	100%
49	2/2	0	100%
50	2/2	0	100%
Total	82	18	82%

Conclusion

For the first training, we have observed that the network was overfitting right from the beginning, the training accuracy relatively stayed at 100%, while the validation accuracy was fluctuating. For the final independent test, we have recorded a test accuracy of 70%. Of course with so little dataset per class, our number one concern was overfitting. In the next section, we will see the results of two iteration steps.

With two routing iterations, the network didn’t improve on the validation accuracy and went down 2.5%, while the test accuracy has improved by 20%, however, when we tested the network with a totally new dataset, the results were not as good as with the one routing iteration model and it went down 1.4% on the final test.

With more classes, our models have improved on the validation accuracy, where it was overfitting in the first two models. Also, we have seen a 4% increase in test accuracy.

We have learned that the test accuracy could be improved by using more classes as in the case of the last model, where the examples per class in the last models were smaller than the examples per class for the other two models.

The reason is when the model learns some enough features about a class, it doesn’t improve the weights of the network, even if we have thousands of images of that class.

With more classes, the network will be pushed to learn how to distinguish, human words, it will have more experience.

4 CONVOLUTIONAL NEURAL NETWORK

4.1 Theory

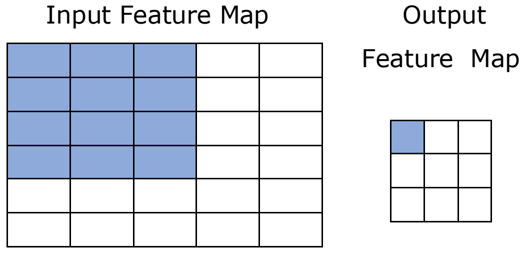
Convolutional neural networks have seen unprecedented improvements due to the advancement of computational machines and the availability of data information and in some areas, it has already surpassed human accuracy.

One weakness of convolutional neural networks is, it needs a lot of datasets or the need to use advanced techniques such as transfer learning and data augmentation if the dataset is too small. To have an affair comparison with the capsule network, which is yet to be standardized and many of the advanced techniques are not applicable. This model will almost have the same techniques with a larger dataset than the one the capsule network is trained; thus, it’s not expected this model to achieve or reach a state of the art accuracy. In addition to that, the aim of this thesis is not to improve the accuracy of CNNs in the domain of face recognition therefore, the results don’t reflect what CNNs can achieve when advanced techniques are used, such as using pre-trained weights.

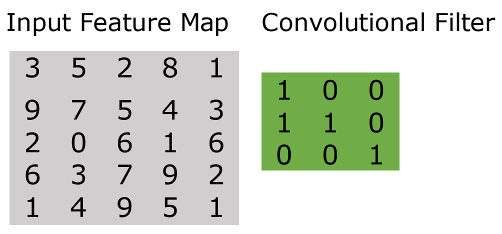
A convolution extracts tiles from the input feature map and uses filters to calculate new features, generating an output feature map, or transformed features which may have a size and depth different from the input feature map. Two parameters describe convolutions.

- Tile size that is extracted (usually 3x3 or 5x5 pixels)
- The depth (channels) of the map of the output feature (this corresponds to the number of filters)

The filters effectively move over the grid of the input horizontally and vertically, one pixel at a time for extracting each corresponding tile.



In figure 4.1 there are nine possible locations to extract tiles from the input feature map, this convolution produces a 3x3 output feature map.



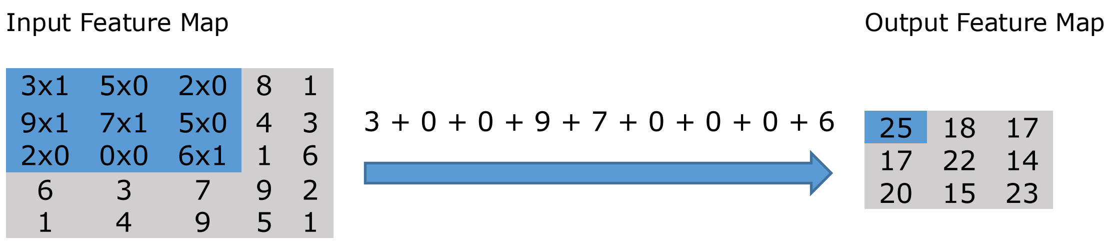
Input Feature Map

3	5	2	8	1
9	7	5	4	3
2	0	6	1	6
6	3	7	9	2
1	4	9	5	1

Convolutional Filter

1	0	0
1	1	0
0	0	1

In CNN we apply element-wise multiplication of the convolution filter over the input feature map as figure 4.3 illustrates. The convolution filter slides on one pixel at a time to the right and then one pixel down starting from the most left pixel over the input feature map.



Input Feature Map

3x1	5x0	2x0	8	1
9x1	7x1	5x0	4	3
2x0	0x0	6x1	1	6
6	3	7	9	2
1	4	9	5	1

Output Feature Map

25	18	17
17	22	14
20	15	23

Figure 4.1 the convolution of the 3x3 filter is performed over the 5x5 input feature map, on the right is the resulting convolved feature

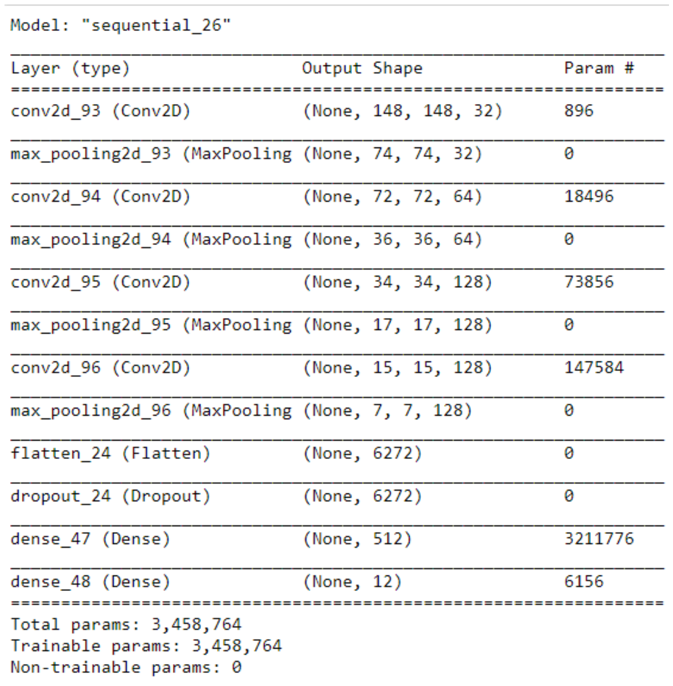
The CNN “learns” during training the optimal values for the convolution filter matrices that allow it to extract meaningful and useful features such as textures, edges, shapes etc., from the input feature map. The number of features that CNN can extract increases with the number of filters (Output filter map depth) applied to the input. However, the trade-off is that filters make up most of CNN’s resources, thereby the training also becomes longer as more and more filters are introduced or added. In addition, each filter that is applied to the network outputs less incremental value than the previous one. So, we aim to build networks by using the minimum needed number of filters to extract the features that are important for accurate image classification.

4.2 Architecture

To have a fair comparison, a conventional CNNs was designed without using advanced techniques such as transferring transfer learning and the same dataset is used for training the network. It may prove difficult to get good results with a shallow CNNs architecture. Therefore, the aim of this network is not to get good results but rather to see how it performs with a little dataset and without advanced techniques against the capsule network. This convnet will be a stack of alternating ReLU activation functions and MaxPooling layers.

Table 4.1 CNNs Parameters

Layers	**Filters**	**stride**	**Max Pooling**	**activation**
1	32	3x3		ReLU
2			2x2	
3	64	3x3		ReLU
4			2x2	
5	128	3x3		ReLU
6			2x2	
7	128	3x3		ReLU
8			2x2	
Flatten				
Dropout	0.2			
9	Dense512			ReLU
10	Dense12			Sigmoid



```
Model: "sequential_26"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_93 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_93 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_94 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_94 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_95 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_95 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_96 (Conv2D)           (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_96 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
flatten_24 (Flatten)         (None, 6272)              0
_________________________________________________________________
dropout_24 (Dropout)         (None, 6272)              0
_________________________________________________________________
dense_47 (Dense)             (None, 512)               3211776
_________________________________________________________________
dense_48 (Dense)             (None, 12)                6156
=================================================================
Total params: 3,458,764
Trainable params: 3,458,764
Non-trainable params: 0
```

Figure 4.2 Model summary

4.3 Results

The training accuracy until it reaches nearly 100%, while the validation accuracy stall at 75-77%. These are the characteristics of overfitting. This is because our training examples are few (120).

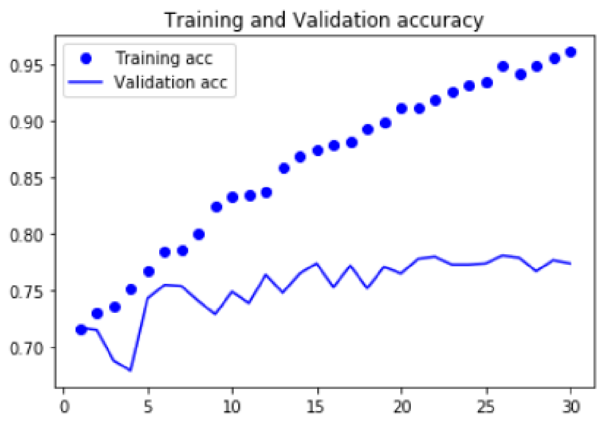


Figure 4.3 Training and Validation accuracy

After only 5 epochs the validation loss reaches its minimum and then it stalls and fluctuates, while the training loss reaches almost zero.

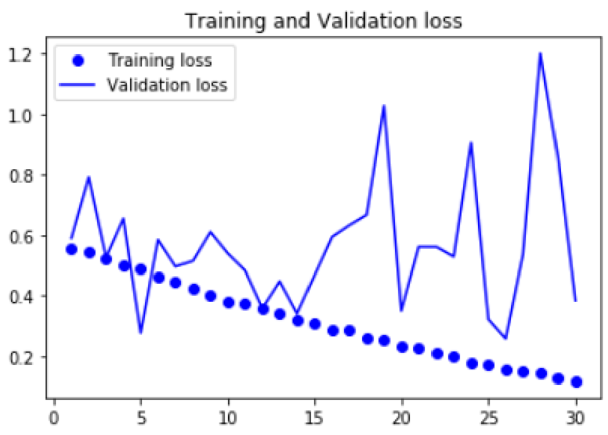


Figure 4.4 Training and Validation loss

5 FACE-NET

5.1 Theory

In 2015 Florian Schroff et al published [24] a paper that was titled FaceNet: A Unified Embedding for Face Recognition and Clustering. Given an input face, the algorithm will extract high-quality features from the face and will predict 128 element vectors representation, these features are known as face embeddings.
To measure the similarity of faces, FaceNet maps face images to a compact Euclidean Space, and the distance corresponds to the similarity of faces.

5.1.1 Triplet loss

In triplet loss, the squared distance between all faces, regardless of imaging conditions, of the same person is small, whereas the squared distance between two different persons is large [24].

$$\left\| f(x_i^a) - f\left(x_i^p\right) \right\|_2^2 + \propto < \| f(x_i^a) - f(x_i^n) \|_2^2 \qquad 5.0$$

Where x_i^a is an anchor input image,

x_i^p positive image for the anchor,

x_i^n is an image that doesn't belong to the anchor image

$\propto$ is a margin that is enforced between positive and negative pairs.

$$\forall \left(f\,(x_i^a), f\left(x_i^p\right), f(x_i^n) \right) \in T \qquad 5.1$$

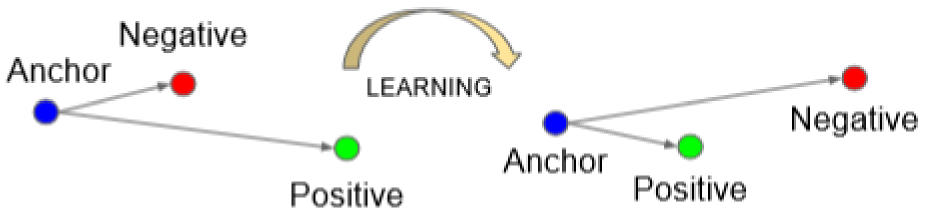


Figure 5.1 Before and after applying triplet loss

In equation 5.1 for every function in it, belongs (T) to all possible triplets in the training set and has cardinality N.
The minimized triplet loss can be calculated as shown in equation 5.0

5.1.2 Triplet selection

If we choose A, P, N (Anchor, Positive, Negative) randomly equation 5.0 can be easily satisfied. To prevent this from happening, the authors chose triplets that are hard to train on. This also increases computational efficiency. It’s time taking to compute the argmin and argmax for all the training set, and it’s possible that it might lead to poor training [24].

5.2 Architecture

FaceNet extracts high-quality features from the given image and these features are called face embeddings, the extracted face embeddings are used to train a face identification system. In this thesis project SVM (Support Vector Machines) classifier is used to do the classification.

5.2.1 Support vector machines

SVM (Support vector machines) is a linear model for classification. SVM can solve many classification problems, linear and non-linear.
The SVM algorithm creates a hyperplane which separates the data into classes.
Separating two or more classes there are a number of possible hyperplanes as shown in figure 5.2[25] that could be chosen. The objective is to find an optimal hyperplane, where the margin is maximum, in other words, the maximum distance between classes. The number of planes is directly proportional to the number of classes we want to predict.

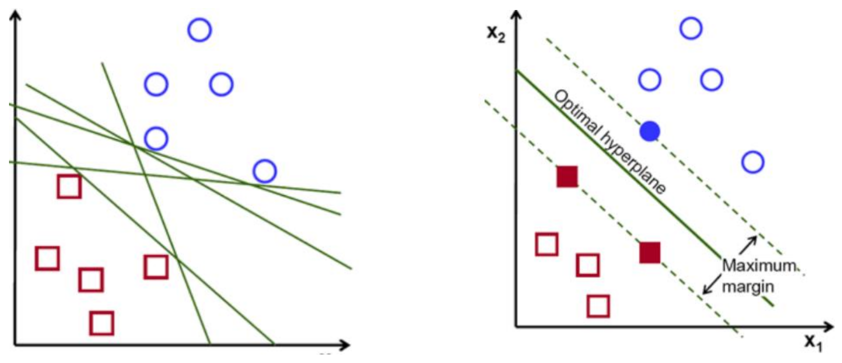


Figure 5.2 Possible hyperplanes

5.2.2 Face vector calculation

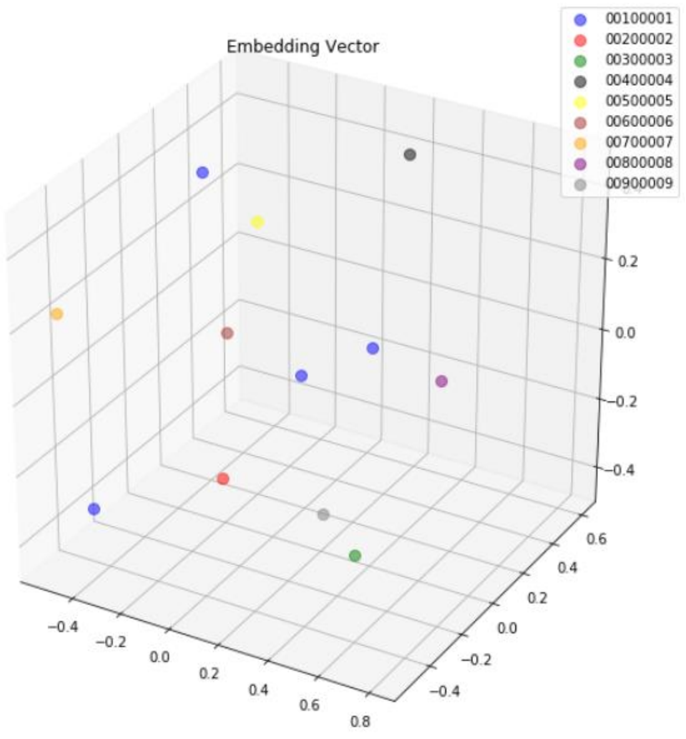


Figure 5.3 Face embeddings in Vector space

Figure 5.4 shows labels in the Euclidean space where each class has a unique embeddings vector which distinguishes from other classes. By knowing the average distance of all labels we can set a logical threshold, such that any input image where the distance is greater than the decided threshold is unknown. Figure 5.5 illustrates the distance between class 001 and 002. The top left number is the distance between the two compared classes.

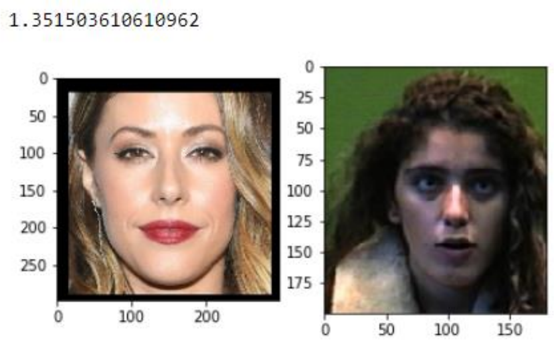


Figure 5.4 Distance between label 001 and label 002

In figure 5.5 two different examples belonging to the same class were compared and the distance is smaller than that one in figure 5.4.

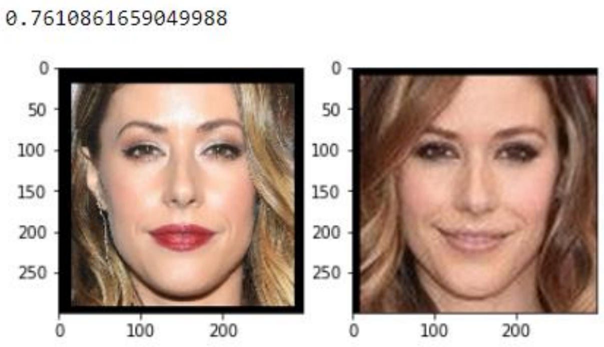


Figure 5.5 Label 001 vs label 001

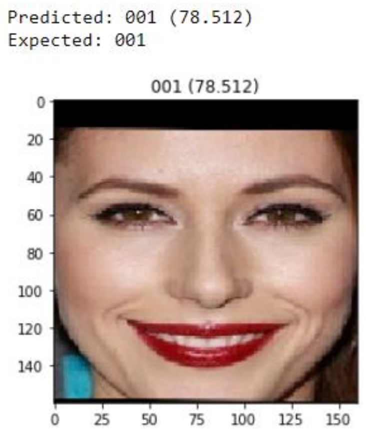
In table 5.1 we can observe that the Euclidean distance between different classes is 1.42 on average, and is almost 50% more than that of the same class.

Table 5.1 Euclidean distance between classes

Class vs Class	**Distance**
001 vs 002	1.35
003 vs 004	1.36
005 vs 006	1.50
007 vs 008	1.55
009 vs 010	1.31
011 vs 012	1.44
001 vs 001	0.76

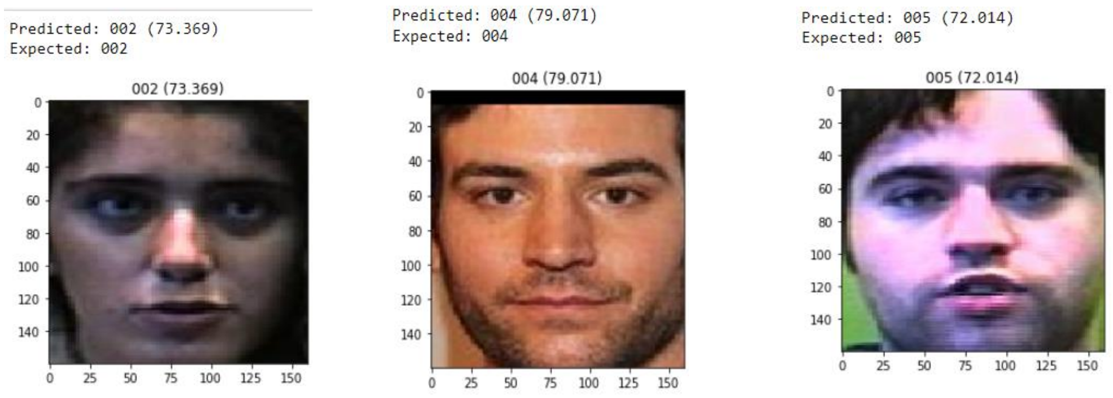
5.3 Results

These results are not state-of-the-art results. But it is satisfactory when compared to the training time of the Caps-Net and CNNs. We can easily increase the accuracy by using more examples of each class for training.
Five images per class were used for the testing and all of them were predicted correctly



```
Predicted: 001 (78.512)
Expected: 001
```

Figure 5.6



```
Predicted: 002 (73.369)
Expected: 002
```

```
Predicted: 004 (79.071)
Expected: 004
```

```
Predicted: 005 (72.014)
Expected: 005
```

Figure 5.7 Prediction examples

Conclusion

Face-Net is yet by far one of the best solutions for face identification and recognition. This method is capable of learning all the features in the face from a single image.
It doesn't require a lot of training time, as there are pre-trained models available.

This solution is more suitable for large companies which use face recognition access gates because you don't need to train your model for every new employee that joins your company.

6 ANALYSIS

First, we observed the CNN The training accuracy increased linearly until it reached nearly 100%, while the validation accuracy reached its maximum after only 5 epochs and then it stalled at 75%, whereas the training loss decreased linearly to almost zero and the validation loss reached its lowest point after just 5 epochs. These characteristics are known as overfitting. Overfitting is mainly caused by a small data-set. This is exactly the main reason why we used a small dataset in the first place, the idea was to see if CAPSNET can overcome this performance using the same small data set.

However, when using a more mature algorithm with a pre-trained model that allows us to overcome the challenge of overfitting we got a much better results, in the case of FACE-NET we used a pre-trained model and SVM for the classification, as the model is pre-trained we just had to get the data in the correct format, the network have correctly classified all the classes with an average confidence 76% and an average of 1.37 Euclidean distance between each two consecutive classes. Whereas the average Euclidean distance between the same class was 0.76, this distance can be used for setting a threshold.
these results were, as we have expected as such a well mature and pre-trained model should give.

Finally, for the CAPS-NET first, we used a one-step routing iteration, the network managed to reach training accuracy of 100%, validation accuracy of 90% and test accuracy of 70%. Also, the network was tested on a total of 76 new faces belonging to the 12 classes, 6 faces per class. Of the 76, 64 faces were correctly classified while 8 faces were miss-classified. On the new test data, we have achieved an accuracy of 88.9%. we also tested two routing iteration, the training accuracy stayed the same while the validation accuracy decreased 2.5%, and we have recorded a 20% increase in the test accuracy and when testing with new data set there was a 1.4% decrease in the test accuracy.

We also trained the Caps-Net model with a large data-set, it became evident the more training examples you have for training the better might results be. Training accuracy and the Validation accuracy were not overfitting,

Table 6.1 Test accuracy of the models

Method	Test Accuracy
Caps-Net (1 step)	70%
Caps-Net (2 steps)	90%
Face-Net	100%
CNNs	75%
Caps-Net	0.9380

As we can see in Table 11 CAPSNET has shown better results in comparison with the CNNs model, although the two models overfitted, CAPS-NET managed to close the overfitting by 15%, this gives results help us to answer the first question we asked, Can CAPS-NET outperform CNN in the face identification application when trained on both models are trained using a small data-set? The answer is yes, based on the results we obtained we can clearly see the CAPS-NET outperformed CNN when using a small data-set, however using a pre-trained model out-performed both models in the case of facet.

SUMMARY

The current thesis explored the potential of using Capsule Network for face identification using a minimal dataset for training.

The goal of the work was to answer two questions:
The current thesis explored the potential of using Capsule Network for face identification using a minimal dataset for training.

- Can CapsNet outperform CNN in the face identification application when trained on both models are trained using a small data-set?
- How does CapsNet perform compared to a more mature and pre-trained model?

We have implemented three models for face identification, CNNs model, Face-Net model, and Capsule network model, where the main focus was the capsule network and the two other models as a benchmark.
For the training, we used 120 face images belonging to 12 different classes, 7 classes coming from the Faces94 data-set and 5 classes from the LFW data-set.
The CNN model reached an accuracy of 75% after only 5 epochs and then it stalled. overfitting was observed as expected due to the small data size. the idea was to see if CAPS-NET can overcome this performance using the same small data set.

The CAPS-NET model was tested on two configurations first, we used a one-step routing iteration, the network managed to reach a test accuracy of 70%. The second time we used two-step routing and we were able to get 90% test accuracy.

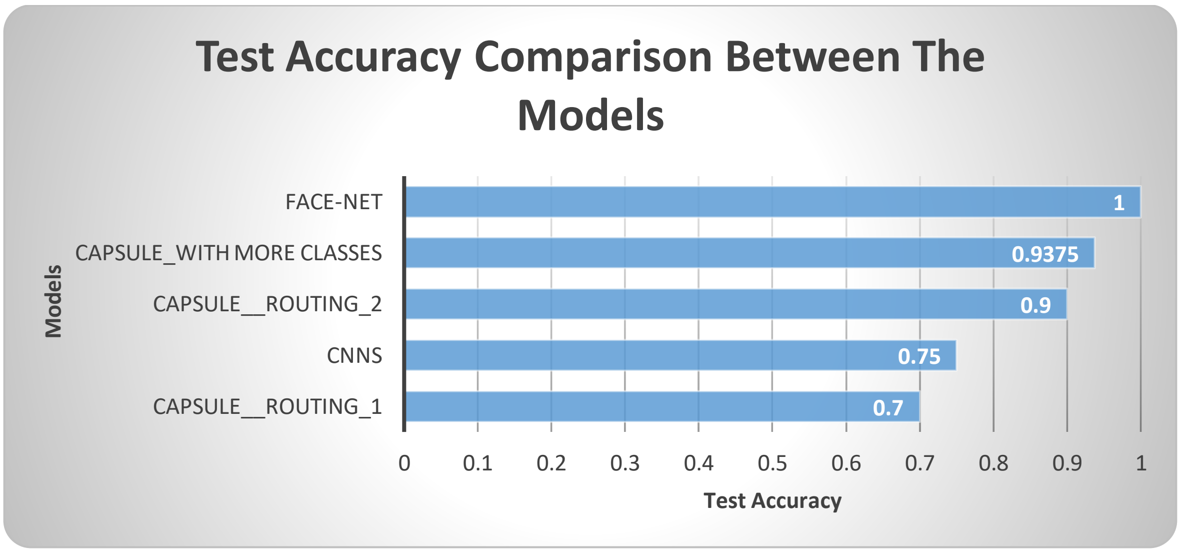
We also experimented Caps-Net with a larger data set. The results we got were promising and the overfitting almost diminished. I have achieved a training accuracy of 100%, validation accuracy of 100% and test accuracy of 0.9375.

Finally the FACE-NET we used a pre-trained model and SVM for the classification, as the model is pre-trained, the network has correctly classified all the classes %100 test accuracy with average confidence 76%.

CAPS-NET has shown better results in comparison with the CNNs model when both models were trained using a small data-set, however, still using a pre-trained model out-performed both models in the case of facet.

Some of the challenges during the building of the caps-net was getting the data into the right format and matching output vector shapes from the primary capsule layer
In concluding the Caps-net I believe that the CAPS-NET has great potential in the use of face identification when using a small data set and as it matures with time it could be the state of the art methods when it comes to image classification.

In the future more work should be done to get a better measure of the optional of using CAPS-NET, this can be done by training CAPS-NET on big available datasets and developing the method further to include features such as transfer learning.



See lõputöö uurib Capsule Network kasutamist näotuvastuses, kasutades treenimiseks minimaalset andmehulka.

Töö eesmärk oli vastata kahele küsimusele:
See lõputöö uurib Capsule Network kasutamist näotuvastuses, kasutades treenimiseks minimaalset koolitust.

• Kas CapsNet võib näotuvastuses edestada CNN-i, kui mõlemat mudelit treenitakse väikeses andmehulgaga?
• Kuidas CapsNet töötab küpsema ja eelkoolitatud mudeliga võrreldes?

Oleme kasutnud kolm näotuvastuse mudelit: CNN-i mudeli, Face-Net-mudeli ja Capsule'i võrgumudeli, põhirõhk on kapselvõrgul ja veel kahel mudelil võrdluseks.
Koolituseks kasutasime 120 näopilti 12 erinevast klassist, 7 Faces94 andmestikust ja 5 LFW andmestikust.
CNN-mudeli täpsus saavutas 75% täpsusega ainult 5 iteratsiooni järel ja siis see takerdus. Ootuspäraselt eeldati andmete puudumise tõttu ülesobitust. idee oli näha, kas CAPS-NET saaks sama andmehulgaga abil sellest ületada tulemuse.

CAPS-NET mudelit testiti kõigepealt kahes konfiguratsioonis, me kasutasime marsruutide üheastmelist iteratsiooni ja võrgul õnnestus saavutada 70% -line katse täpsus. Teisel korral kasutasime kaheastmelist marsruutimist ja suutsime saada 90% testi täpsuse.

Katsetasime Caps-Netiga ka suurema andmebaasi. Saadud tulemused olid paljulubavad ja ülesobitumine vähenes. Olen saavutanud treenimise täpsuse 100%, valideerimise täpsuse 100% ja testi täpsuse 0,9375.

Lõpuks, FACE-NET, mida me kasutasime, on klassifitseerimise ja SVM-i jaoks eelkoolitatud mudel, kuna mudel on eelnevalt koolitatud, võrk on õigesti klassifitseeritud kõigi klasside jaoks 100% -lise testitäpsusega, keskmise usaldusega 76%.

CAPS-NET on näidanud paremaid tulemusi kui CNN-mudel, kui mõlemat mudelit koolitati väikese andmemahuga, kuid kasutades eelkoolitatud mudelit, mõlemad tulemused ületati.

Mõned caps-net'i loomise väljakutsed olid andmete korrektne vormindamine ja esmase kapsli kihi väljundvektori kuju sobitamine

Caps-net'i kokkuvõtteks usun, et CAPS-NETil on suur potentsiaal näotuvastuse kasutamiseks väikese andmekogumiga ja kuna see aja jooksul küpseb, võib see olla uusim kujunduse klassifitseerimise meetod.

CAPS-NET-i selektiivsuse paremaks mõõtmiseks on tulevikus vaja veel rohkem ära teha, seda saab teha CAPS-NET-i väljaõppe kaudu olemasolevatest suurtest andmekogumitest ja edasi arendada selliste funktsioonide lisamise meetodit nagu ülekandekoolitus

LIST OF REFERENCES

[1] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Nips, pp. 3857–3867, 2017.

[2] R. Mukhometzianov and J. Carrillo, “CapsNet comparative performance evaluation for image classification,” pp. 1–14, 2018.

[3] “Example tutorial.” [Online]. Available: https://github.com/naturomics/CapsLayer/blob/master/docs/tutorials.md.

[4] M. A. Turk and A. P. Pentland, “Face recognition using eigenfaces.” pp. 586–591, 1991.

[5] T. S. Huang, “Computer Vision: Evolution and Promise,” *Report*, 1997.

[6] R. Kirsch, “The First Digital Image,” 1959. [Online]. Available: https://www.nist.gov/news-events/news/2007/05/fiftieth-anniversary-first-digital-image-marked.

[7] M. J. J. Paul Viola, “Robust object detection with real-time fusion of multiview foreground silhouettes,” *Opt. Eng.*, vol. 51, no. 4, p. 047202, 2001.

[8] Faces94, “Faces94.” [Online]. Available: https://cswww.essex.ac.uk/mv/allfaces/.

[9] E. L.-M. et Al, “LFW.” [Online]. Available: http://vis-www.cs.umass.edu/lfw/.

[10] “Georgia Tech Face Database.” [Online]. Available: http://www.anefian.com/research/face_reco.htm?fbclid=IwAR3bpQc0hJYhv4HWN-qMZNhEErifxexxe_iRUy44n74vz-eCtha00nx70uk.

[11] “Data limitation.”

[12] P. Afshar, A. Mohammadi, and K. N. Plataniotis, “Brain Tumor Type Classification via Capsule Networks,” *Proc. - Int. Conf. Image Process. ICIP*, pp. 3129–3133, 2018.

[13] G. E. Hinton, A. Krizhevsky, and S. D. Wang, “Transforming Auto-Encoders,” pp. 44–51, 2011.

[14] "Boat and house Example." [Online]. Available: https://software.intel.com/en-us/articles/understanding-capsule-network-architecture?fbclid=IwAR2U_aBGsCTMBu3EVPINonqvypqu0BL2Ze-YwvX4sW3BZfa7Fo_kJEqb_uY.

[15] "Subparts." [Online]. Available: https://software.intel.com/en-us/articles/understanding-capsule-network-architecture?fbclid=IwAR2U_aBGsCTMBu3EVPINonqvypqu0BL2Ze-YwvX4sW3BZfa7Fo_kJEqb_uY.

[16] "Arrow representation." [Online]. Available: https://software.intel.com/en-us/articles/understanding-capsule-network-architecture?fbclid=IwAR2U_aBGsCTMBu3EVPINonqvypqu0BL2Ze-YwvX4sW3BZfa7Fo_kJEqb_uY.

[17] "Predictions." [Online]. Available: https://software.intel.com/en-us/articles/understanding-capsule-network-architecture?fbclid=IwAR3Nkh1Sa9np6SCRzZSz2AmbGd0XSI_70yrpygXdGXR716GnQKW91J45SAc.

[18] "Agreement." .

[19] "Weights to be zero." [Online]. Available: https://software.intel.com/en-us/articles/understanding-capsule-network-architecture?fbclid=IwAR3Nkh1Sa9np6SCRzZSz2AmbGd0XSI_70yrpygXdGXR716GnQKW91J45SAc.

[20] W. Vogt, "Linear Function," *Dictionary of Statistics & Methodology*, 2015. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[21] R. Chapman, "Nonlinear Function," *Strange Attractors*, 2008. [Online]. Available: https://www.khanacademy.org/math/cc-eighth-grade-math/cc-8th-linear-equations-functions/linear-nonlinear-functions-tut/e/linear-non-linear-functions.

[22] Tensorflow, "tensorflow Matmul Function." .

[23] Tensorflow, “tf.tile().” [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/tile?version=stable.

[24] F. Schroff and J. Philbin, “FaceNet: A Unified Embedding for Face Recognition and Clustering.”

[25] “SVM.” [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

APPENDICES

```
    data.append(image)

    l = imagePath[-7:-4]
    labels.append(l)
                                                                      In [ ]:
  labels = np.asarray(labels)
  labels = labels.flatten()
  labels = labels.tolist()
  labels = [str(r) for r in labels]
  labels = [int(r) for r in labels]
  labels[:] = [x-1 for x in labels]
  print(len(set(labels)))
  print(len(labels))
                                                                      In [ ]:
  train_data, val_data, train_labels, val_labels = train_test_split(data, l
  abels, test_size=0.2, random_state=42)
  train_data, test_data, train_labels, test_labels = train_test_split(train
  _data, train_labels, test_size=0.2, random_state=42)

  train_data = np.array(train_data)
  train_labels = np.array(train_labels)

  val_data = np.array(val_data)
  val_labels = np.array(val_labels)

  test_data = np.array(test_data)
  test_labels = np.array(test_labels)
                                                                      In [ ]:
  X_train = train_data
  X_valid = val_data

  y_train = train_labels
  y_valid = val_labels

  X_test = test_data
  y_test = test_labels

  print(len(X_train),len(X_valid),len(X_test))

  print(len(y_train),len(y_valid),len(y_test))
                                                                      In [ ]:
  X_train = X_train / 255
  X_valid = X_valid / 255
  X_test = X_test / 255

  from PIL import ImageEnhance
  def preprocessing_function(img):

def preprocessing_function(img):
    img = img * 255
    img = Image.fromarray(img.astype('uint8'), 'GREY')

    return np.array(img) / 255

train_datagen = ImageDataGenerator()
train_datagen_augmented = ImageDataGenerator(
    rotation_range=20,
    shear_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    preprocessing_function=preprocessing_function)
inference_datagen = ImageDataGenerator()
train_datagen.fit(X_train)
train_datagen_augmented.fit(X_train)
inference_datagen.fit(X_valid)
inference_datagen.fit(X_test)
```

In [ ]:

```
def conv_caps_layer(input_layer, capsules_size, nb_filters, kernel, strid
e=2):

    capsules = tf.contrib.layers.conv2d(
        input_layer, nb_filters * capsules_size, kernel, stride, padding=
"VALID")
    shape = capsules.get_shape().as_list()
    capsules = tf.reshape(capsules, shape=(-1, np.prod(shape[1:3]) * nb_f
ilters, capsules_size, 1))
    return squash(capsules)

def routing(u_hat, b_ij, nb_capsules, nb_capsules_p, iterations=4):


    for it in range(iterations):
        with tf.variable_scope('routing_' + str(it)):
            c_ij = tf.nn.softmax(b_ij, axis=2)

            s_j = tf.multiply(c_ij, u_hat)
            s_j = tf.reduce_sum(s_j, axis=1, keepdims=True)

            v_j = squash(s_j)
            v_j_tiled = tf.tile(v_j, [1, nb_capsules_p, 1, 1, 1])
            u_dot_v = tf.matmul(u_hat, v_j_tiled, transpose_a=True)
            b_ij += tf.reduce_sum(u_dot_v, axis=0, keepdims=True)

    return tf.squeeze(v_j, axis=1)
```

```
def fully_connected_caps_layer(input_layer, capsules_size, nb_capsules, i
terations=4):

    shape = input_layer.get_shape().as_list()

    len_u_i = np.prod(shape[2])
    len_v_j = capsules_size

    nb_capsules_p = np.prod(shape[1])

    _init = tf.random_normal_initializer(stddev=0.01, seed=0)
    _shape = (nb_capsules_p, nb_capsules, len_v_j, len_u_i)
    w_ij = tf.get_variable('weight', shape=_shape, dtype=tf.float32, init
ializer=_init)

    input_layer = tf.reshape(input_layer, shape=(-1, nb_capsules_p, 1, le
n_u_i, 1))
    input_layer = tf.tile(input_layer, [1, 1, nb_capsules, 1, 1])


    u_hat = tf.einsum('abdc,iabcf->iabdf', w_ij, input_layer)


    b_ij = tf.zeros(shape=[nb_capsules_p, nb_capsules, 1, 1], dtype=np.fl
oat32)

    return routing(u_hat, b_ij, nb_capsules, nb_capsules_p, iterations=it
erations)

def squash(vector):

    vector += 0.00001
    vec_squared_norm = tf.reduce_sum(tf.square(vector), -2, keepdims=True
)
    scalar_factor = vec_squared_norm / (1 + vec_squared_norm) / tf.sqrt(v
ec_squared_norm)
    vec_squashed = scalar_factor * vector
    return(vec_squashed)
```

In [ ]:

```
from model_base import ModelBase


class ModelSignature(ModelBase):


    NB_LABELS = 50

    def __init__(self, model_name, output_folder):
        ModelBase.__init__(self, model_name, output_folder=output_folder)

    def _build_inputs(self):

        tf_images = tf.placeholder(tf.float32, [None, 80, 80, 3], name='i
mages')
        tf_labels = tf.placeholder(tf.int64, [None], name='labels')
        return tf_images, tf_labels

    def _build_main_network(self, images, conv_2_dropout):


        shape = (self.h.conv_1_size, self.h.conv_1_size, 3, self.h.conv_1
_nb)
        conv1 = self._create_conv(self.tf_images, shape, relu=True, max_p
ooling=False, padding='VALID')

        conv1 = tf.nn.dropout(conv1, keep_prob=conv_2_dropout)


        caps1 = conv_caps_layer(
            input_layer=conv1,
            capsules_size=self.h.caps_1_vec_len,
            nb_filters=self.h.caps_1_nb_filter,
            kernel=self.h.caps_1_size)

        caps2 = fully_connected_caps_layer(
            input_layer=caps1,
            capsules_size=self.h.caps_2_vec_len,
            nb_capsules=self.NB_LABELS,
            iterations=self.h.routing_steps)

        return caps1, caps2

    def _build_decoder(self, caps2, one_hot_labels, batch_size):

        labels = tf.reshape(one_hot_labels, (-1, self.NB_LABELS, 1))
        mask = tf.matmul(tf.squeeze(caps2), labels, transpose_a=True)

        capsule_vector = tf.reshape(mask, shape=(batch_size, self.h.caps_
2_vec_len))

        fc1 = tf.contrib.layers.fully_connected(capsule_vector, num_outpu
ts=400)
        fc1 = tf.reshape(fc1, shape=(batch_size, 5, 5, 16))
        upsample1 = tf.image.resize_nearest_neighbor(fc1, (8, 8))
        conv1 = tf.layers.conv2d(upsample1, 4, (3,3), padding='same', act
ivation=tf.nn.relu)
```

```
        upsample2 = tf.image.resize_nearest_neighbor(conv1, (16, 16))
        conv2 = tf.layers.conv2d(upsample2, 8, (3,3), padding='same', act
ivation=tf.nn.relu)

        upsample3 = tf.image.resize_nearest_neighbor(conv2, (32, 32))
        conv6 = tf.layers.conv2d(upsample3, 16, (3,3), padding='same', ac
tivation=tf.nn.relu)

        upsample4 = tf.image.resize_nearest_neighbor(conv2, (80, 80))
        conv7 = tf.layers.conv2d(upsample4, 32, (3,3), padding='same', ac
tivation=tf.nn.relu)


        logits = tf.layers.conv2d(conv7, 3, (3,3), padding='same', activa
tion=None)
        decoded = tf.nn.sigmoid(logits, name='decoded')
        tf.summary.image('reconstruction_img', decoded)

        return decoded

    def init(self):


        self.tf_images, self.tf_labels = self._build_inputs()

        self.tf_conv_2_dropout = tf.placeholder(tf.float32, shape=(), nam
e='conv_2_dropout')

        batch_size = tf.shape(self.tf_images)[0]

        one_hot_labels = tf.one_hot(self.tf_labels, depth=self.NB_LABELS)

        self.tf_caps1, self.tf_caps2 = self._build_main_network(self.tf_i
mages, self.tf_conv_2_dropout)

        # Faces reconstruction
        self.tf_decoded = self._build_decoder(self.tf_caps2, one_hot_labe
ls, batch_size)

        # Loss function
        _loss = self._build_loss(
            self.tf_caps2, one_hot_labels, self.tf_labels, self.tf_decode
d, self.tf_images)
        (self.tf_loss_squared_rec, self.tf_margin_loss_sum, self.tf_predi
cted_class,
         self.tf_correct_prediction, self.tf_accuracy, self.tf_loss, self
.tf_margin_loss,
         self.tf_reconstruction_loss) = _loss

        # optimizer
        optimizer = tf.train.AdamOptimizer(learning_rate=self.h.learning_
rate)
        self.tf_optimizer = optimizer.minimize(self.tf_loss, global_step=
tf.Variable(0, trainable=False))


        tf.summary.scalar('margin_loss', self.tf_margin_loss)
        tf.summary.scalar('accuracy', self.tf_accuracy)
        tf.summary.scalar('total_loss', self.tf_loss)
        tf.summary.scalar('reconstruction_loss', self.tf_reconstruction_l
oss)

        self.tf_test = tf.random_uniform([2], minval=0, maxval=None, dtyp
e=tf.float32, seed=None, name="tf_test")

        self.init_session()


    def _build_loss(self, caps2, one_hot_labels, labels, decoded, images)
:



        capsules_length = tf.sqrt(tf.reduce_sum(tf.square(caps2), axis=2,
keepdims=True))

        max_l = tf.square(tf.maximum(0., 0.9 - capsules_length))
        max_l = tf.reshape(max_l, shape=(-1, self.NB_LABELS))
        max_r = tf.square(tf.maximum(0., capsules_length - 0.1))
        max_r = tf.reshape(max_r, shape=(-1, self.NB_LABELS))
        t_c = one_hot_labels
        m_loss = t_c * max_l + 0.5 * (1 - t_c) * max_r
        margin_loss_sum = tf.reduce_sum(m_loss, axis=1)
        margin_loss = tf.reduce_mean(margin_loss_sum)


        loss_squared_rec = tf.square(decoded - images)
        reconstruction_loss = tf.reduce_mean(loss_squared_rec)


        loss = margin_loss + (0.0005 * reconstruction_loss)


        predicted_class = tf.argmax(capsules_length, axis=1)
        predicted_class = tf.reshape(predicted_class, [tf.shape(capsules_
length)[0]])
        correct_prediction = tf.equal(predicted_class, labels)
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32)
)

        return (loss_squared_rec, margin_loss_sum, predicted_class, corre
ct_prediction, accuracy,
                loss, margin_loss, reconstruction_loss)

    def optimize(self, images, labels, tb_save=True):

        tensors = [self.tf_optimizer, self.tf_margin_loss, self.tf_accura
cy, self.tf_tensorboard]
        _, loss, acc, summary = self.sess.run(tensors,
            feed_dict={
            self.tf_images: images,
            self.tf_labels: labels,
            self.tf_conv_2_dropout: self.h.conv_2_dropout
        })

        if tb_save:

            self.train_writer.add_summary(summary, self.train_writer_it)
            self.train_writer_it += 1

        return loss, acc

    def evaluate(self, images, labels, tb_train_save=False, tb_test_save=
False):

        tensors = [self.tf_margin_loss, self.tf_accuracy, self.tf_tensorb
oard]
        loss, acc, summary = self.sess.run(tensors,
                feed_dict={
                self.tf_images: images,
                self.tf_labels: labels,
                self.tf_conv_2_dropout: 1.
            })

        if tb_test_save:

            self.test_writer.add_summary(summary, self.test_writer_it)
            self.test_writer_it += 1

        if tb_train_save:

            self.train_writer.add_summary(summary, self.train_writer_it)
            self.train_writer_it += 1

        return loss, acc

    def predict(self, images):

        tensors = [self.tf_caps2]

        caps2 = self.sess.run(tensors,
            feed_dict={
            self.tf_images: images,
            self.tf_conv_2_dropout: 1.
        })[0]

        caps2 = np.sqrt(np.sum(np.square(caps2), axis=2, keepdims=True))
        caps2 = np.reshape(caps2, (len(images), self.NB_LABELS))

        softmax = np.exp(caps2) / np.sum(np.exp(caps2), axis=1, keepdims=
True)

        return softmax

    def reconstruction(self, images, labels):

        tensors = [self.tf_decoded]

        decoded = self.sess.run(tensors,
            feed_dict={
            self.tf_images: images,
            self.tf_labels: labels,
            self.tf_conv_2_dropout: 1.
        })[0]

        return decoded

    def evaluate_dataset(self, images, labels, batch_size=10):

        tensors = [self.tf_loss_squared_rec, self.tf_margin_loss_sum, sel
f.tf_correct_prediction,
                   self.tf_predicted_class]

        loss_squared_rec_list = None
        margin_loss_sum_list = None
        correct_prediction_list = None
        predicted_class = None

        b = 0
        for batch in self.get_batches([images, labels], batch_size, shuff
le=False):
            images_batch, labels_batch = batch
            loss_squared_rec, margin_loss_sum, correct_prediction, classe
s = self.sess.run(tensors,
                feed_dict={
                self.tf_images: images_batch,
                self.tf_labels: labels_batch,
                self.tf_conv_2_dropout: 1.
            })
            if loss_squared_rec_list is not None:
                predicted_class = np.concatenate((predicted_class, classe
s))
                loss_squared_rec_list = np.concatenate((loss_squared_rec_
list, loss_squared_rec))
                margin_loss_sum_list = np.concatenate((margin_loss_sum_li
st, margin_loss_sum))
                correct_prediction_list = np.concatenate((correct_predict
ion_list, correct_prediction))
            else:
                predicted_class = classes
                loss_squared_rec_list = loss_squared_rec
                margin_loss_sum_list = margin_loss_sum
                correct_prediction_list = correct_prediction
            b += batch_size

        margin_loss = np.mean(margin_loss_sum_list)
        reconstruction_loss = np.mean(loss_squared_rec_list)
        accuracy = np.mean(correct_prediction_list)

        loss = margin_loss

        return loss, accuracy, predicted_class
```

In [ ]:

```
model = ModelSignature("Signature", output_folder="outputs")
model.init()
```

In [ ]:

```
BATCH_SIZE = 50


def plot_progression(b, cost, acc, label): print(
    "[%s] Batch ID = %s, loss = %s, acc = %s" % (label, b, cost, acc))


b = 0
valid_batch = inference_datagen.flow(X_valid, y_valid, batch_size=BATCH_S
IZE)
best_validation_loss = None
```

```
augmented_factor = 0.99
decrease_factor = 0.90
train_batches = train_datagen.flow(X_train, y_train, batch_size=BATCH_SIZ
E)
augmented_train_batches = train_datagen_augmented.flow(X_train, y_train,
batch_size=BATCH_SIZE)

while True:
    next_batch = next(
        augmented_train_batches if random.uniform(0, 1) < augmented_facto
r else train_batches)
    x_batch, y_batch = next_batch


    cost, acc = model.optimize(x_batch, y_batch)

    x_batch, y_batch = next(valid_batch, None)

    cost_val, acc_val = model.evaluate(x_batch, y_batch, tb_test_save=Tru
e)

    if b % 10 == 0:
        plot_progression(b, cost, acc, "Train")
        plot_progression(b, cost_val, acc_val, "Validation")
    if b % 100 == 0:
        print("Evaluate full validation dataset ...")
        loss, acc, _ = model.evaluate_dataset(X_valid, y_valid)
        print("Current loss: %s Best loss: %s" % (loss, best_validation_l
oss))
        plot_progression(b, loss, acc, "TOTAL Validation")
        if best_validation_loss is None or loss < best_validation_loss:
            best_validation_loss = loss
            model.save()
        augmented_factor = augmented_factor * decrease_factor
        print("Augmented Factor = %s" % augmented_factor)
    if b==50:
      model.save()
      break
    b += 1
```

In [ ]:

```
loss, acc, predicted_class = model.evaluate_dataset(X_test, y_test)

print("Test Accuracy = ", acc)
print("Test Loss = ", loss)
print(predicted_class)
print(len(predicted_class))
```

In [ ]:

```
import array as arr
```

```
a = arr.array('i',[1,2,3,4,5,6,7,8,9,10,11,12])
predicted_id_to_user = ["001","002","003","004","005","006","007","008","
009","010","011","012"]

images1 = []
path = sorted(list(paths.list_images("TestDriveData1")))

for imagePath in path:
  image = cv2.imread(imagePath.replace("\\","/"))
  image = cv2.resize(image,(80, 80),3)
  plt.imshow(image)
  plt.show()
  image = np.array(image) / 255
  images1.append(image)
                                                                    In [ ]:
model = "c1s_9_c1n_256_c2s_6_c2n_64_c2d_0.7_c1vl_16_c1s_5_c1nf_16_c2vl_32
_lr_0.0001_rs_1--Signature--1576988451.036376.data-00000-of-00001"
                                                                    In [ ]:
predictions_for_faces = model.predict(images1)
                                                                    In [ ]:
fig, axs = plt.subplots(15, 2, figsize=(10, 72))
axs = axs.ravel()
for i in range(30):|
    if i%2 == 0:
        axs[i].axis('off')
        axs[i].imshow(images1[i // 2])
        axs[i].set_title("Prediction: %s" % predicted_id_to_user[np.argma
x(predictions_for_real[i//2])])
    else:
        axs[i].bar(a, predictions_for_real[i // 2])
        axs[i].set_ylabel("Softmax")
        axs[i].set_xlabel("Labels")

plt.show()
                                                                    In [ ]:
Faces3 = []
path = sorted(list(paths.list_images("Test/DriveData2")))
for imagePath in path:
  image = cv2.imread(imagePath.replace("\\","/"))
  image = cv2.resize(image,(80, 80),3)
  plt.imshow(image)
  plt.show()
  image = np.array(image) / 255
  Faces3.append(image)
```