Tampere University

Topi Hintikka

REAL-TIME SINGLE-SHOT FACE RECOGNITION USING MACHINE LEARNING

Faculty of Information Technology and Communication Sciences
Master's Thesis
April 2019

ABSTRACT

Topi Hintikka: Real-time Single-shot Face Recognition using Machine Learning
Master's Thesis
Tampere University
Information Technology
April 2019

Face recognition is one of the earliest applications in the field of computer vision. In general, face recognition refers to recognizing the identity of a person from the facial image, by comparing the image to the set of facial images from known identities. Numerous approaches for solving this face recognition problem utilizing traditional *machine learning* methods have been proposed over the years. However, these methods have not been capable of solving the problem in an unconstrained environment. Due to increase of available data and computational power over the past 5 years, *convolutional neural network* (CNN) based *deep learning* approaches have became a *state-of-the-art* solution for face recognition problem. While some of these methods are capable of even beating a human in recognition accuracy, also the computational complexity of deep learning models used in solutions, have increased significantly and usually a *graphics processing unit* (GPU) is required to run face recognition in real-time.

A pipeline for solving the face recognition problem in an unconstrained environment on real-time from the video stream, is presented in this thesis. For solving the problem, the pipeline is divided into 4 steps where each step solves one smaller more specific problem. For each of the steps, multiple approaches are tested to find an optimal solution to the problem solved in the step. After all, a solution for each of the steps in the pipeline are proposed and the solutions are collected for demonstrating the complete system. The system is designed to run on embedded NVIDIA Jetson TX2 hardware.

Face detection is the first step in the pipeline, where faces are detected from the video stream and extracted. Then, the affection of applying different image alignments for the extracted facial croppings before the following steps of the pipeline, are tested. Alignment is performed by training machine learning models for facial landmark detection and applying an image transformation based on the detections. The third step, feature extraction from the facial images, is solved by training CNN's with different alignments and loss functions to find an optimal solution for extraction. In the last step, an embedding of extracted features, is classified using nearest neighbor search from the set of known embeddings extracted earlier. Different clustering approaches are tested for accelerating the search speed by finding the nearest neighbor only from the subset of known embeddings.

For face detection, all the tested lightweight CNN object detectors achieved similar accuracy and each of them could be used. For extracted facial images, trained *Multi-task Cascaded Convolutional Networks* (MTCNN) framework's *ONet* achieved superior accuracy in facial landmark detection and it was utilized in implementing different alignments for facial images. Different alignments and loss functions had significant affection to nearest neighbor search. The highest accuracy was achieved by applying an affine transformation based on detected facial landmarks before feature extraction performed by CNN trained with *Additive Angular Margin* (ArcFace) loss. For accelerating nearest neighbor search, a hierarchical correlation clustering algorithm CHUNX achieved very accurate approximation for the nearest neighbor with significantly smaller slope compared to the exact nearest neighbor search. After all, the implemented demo application runs in real-time on Jetson TX2 hardware with over 10 frames per seconds (FPS).

Keywords: Face Recognition, Machine Learning, Deep Learning, Convolutional Neural Networks, Nearest Neighbor Search, Clustering

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Topi Hintikka: Reaaliaikainen kasvontunnistus käyttäen koneoppimisen menetelmiä
Diplomityö
Tampereen yliopisto
Tietotekniikka
Huhtikuu 2019

Kasvontunnistus on yksi konenäön ensimmäisistä sovelluskohteista. Yleisesti sillä viitataan henkilöllisyyden tunnistamiseen kasvokuvan perusteella vertailemalla kasvoja joukkoon tunnettuja kasvoja ja etsimällä parasta vastaavuutta. Vuosien varrella on ehdotettu lukuisia vaihtoehtoja ongelman ratkaisemiseksi, jotka hyödyntävät perinteisiä koneoppimisen menetelmiä. Näiden menetelmien avulla ei kuitenkaan ole kyetty ratkaisemaan ympäristössä, jota ei ole standardoitu. Saatavilla olevan datan määrän ja laskentatehon kasvettua merkittävästi viimeisen viiden vuoden aikana, konvoluutioneuroverkkoihin (*engl. convolutional neural networks* (CNN)) pohjautuvat menetelmät ovat osoittautuneet toimivimmaksi ongelman ratkaisemiseksi. Osa näistä menetelmistä on onnistunut lyömään ihmisen tunnistustarkkuudessa, mutta samanaikaisesti ne ovat muuttuneet laskennallisesti huomattavasti raskaammiksi. Käytännössä menetelmiä käyttöön vaaditaan grafiikkaprosessori, jotta niillä voitaisiin tehdä kasvontunnistusta reaaliajassa.

Tässä työssä esitellään ratkaisu reaaliaikaiseen kasvontunnistukseen liittyvän ongelman ratkaisuun ympäristössä, jota ei ole standardoitu. Ongelma jaetaan ratkaisua varten neljään eri osaan, joista kukin osista muodostaa pienemmän tarkemmin rajatun ongelman. Jokaisen osaongelman ratkaisemiseksi testataan useampia eri lähestymisiä, ja lopulta paras ratkaisu jokaiseen osaongelmaan kootaan yhteen. Tämän pohjalta toteutetaan demo sulautetulle NVIDIA Jetson TX2 laitteelle.

Kasvojen paikallistaminen ja irrottaminen videokuvasta muodostaa ensimmäisen osaongelman. Tämän jälkeen tutkitaan videosta irrotettujen kasvokuvien esikäsittelyn vaikutusta seuraavien osaongelmien ratkaisuun. Esikäsittely tehdään tunnistamalla kasvoista tiettyjen pisteiden sijainti (muun muassa silmät) ja tekemällä tämän perusteella muunnos kuvalle, jolla pyritään asettamaan esimerkiksi silmät tiettyyn kohtaan kuvassa. Kolmas osaongelma liittyy piirrevektorin irrottamiseen esikäsitellystä kasvokuvasta. Parhaan piirreirrottajan löytämistä varten koulutetaan konvoluutioneuroverkkoja käyttäen erilaisia esikäsittelyitä ja sakkofunktioita (*engl. loss functions*). Lopulta irrotetut piirrevektorit luokitellaan etsimällä vektorille lähin naapuri joukosta piirrevektoreita, jotka on aikaisemmin irrotettu samalla piirreirrotusmenetelmällä. Lisäksi tutkitaan eri klusterointimenetelmiä, jotta etsimistä voitaisiin nopeuttaa rajaamalla joukkoa pienemmäksi ennen tarkempaa lähimmän naapurin etsintää.

Kasvojen paikallistamiseen videosta kaikki kolme testattua kevyttä konvoluutioneuroverkkoa saavuttivat lähes toisiaan vastaavan tarkkuuden ja niitä kaikkia oltaisiin voitu hyödyntää toteutetussa demossa. Tiettyjen pisteiden tunnistamiseen kasvokuvasta koulutettu *Multi-task Cascaded Convolutional Networks* (MTCNN) järjestelmään kuuluva *ONet* -konvoluutioneuroverkko saavutti selvästi parhaan tarkkuuden ja sitä hyödynnettiin eri kasvokuvien esikäsittelyiden toteuttamiseen. Eri esikäsittelyillä ja sakkofunktioilla oli suuri vaikutus piirrevektorien luokittelun tarkkuuteen. Paras tarkkuus saavutettiin tekemällä affiinimunnos (*engl. affine transformation*) ennen piirrevektorin irrotusta ja käyttämällä *Additive Angular Margin* (ArcFace) -sakkofunktiota piirreirroitukseen käytetyn konvuluutioneuroverkon koulutuksessa. Parhaan vastaavuuden etsinnässä hierarkinen korrelaatioklusterointialgoritmi CHUNX saavutti erittäin hyvän tarkkuuden hakemalla lähintä naapuria ainoastaan algoritmin osoittamasta alijoukosta verrattuna tarkkaan lähimmän naapurin etsintään. Lisäksi CHUNX:lla tehty lähimmän naapurin etsintä oli suhteessa selvästi nopeampi kuin tarkka lähimmän naapurin etsintä etenkin hakuavaruuden ollessa suurempi. Lopulta osaongelmien parhaiden ratkaisujen avulla toteutettiin demo, joka toimii reaaliajassa Jetson TX2 laitteella yli kymmenen hertsin päivitysnopeudella.

Avainsanat: Kasvontunnistus, Koneoppiminen, Syväoppiminen, Kovoluutioneuroverkot, Lähimmän naapurin etsintä, Klusterointi

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

PREFACE

The project related to this thesis was implemented at Wapice during the spring 2019. First, I would like to thank Wapice for providing an opportunity to work with very interesting projects related to data science including this thesis. Especially fruitful discussions and useful tips from Ilari Kampman and my workmates in data science team have helped me not only with this thesis but also in constant personal development in the field of data science.

In addition, I would like to thank Associate Professor Heikki Huttunen for great tips and feedback for improving this thesis, and steering the writing process to correct direction.

Finally, I am very grateful to Mira for her support and understanding of long days and nights spent with my studies over the years. Also, the support from my family and friends have helped me carrying on throughout my studies.

In Tampere, Finland, on 12 April 2019

Topi Hintikka

CONTENTS

1. INTRODUCTION 1
2. THEORY 4
   - 2.1 Machine learning 4
   - 2.2 Deep learning 5
   - 2.3 Neural networks 6
     - 2.3.1 Activation 7
     - 2.3.2 Loss 8
     - 2.3.3 Optimization 9
     - 2.3.4 Regularization 11
   - 2.4 Convolutional neural networks 12
     - 2.4.1 Convolutional layer 13
     - 2.4.2 Pooling layer 14
     - 2.4.3 Dense layer 16
     - 2.4.4 Commonly used architectures 16
3. PIPELINE 20
   - 3.1 Face detection 21
     - 3.1.1 Viola-Jones object detector 22
     - 3.1.2 Single Shot MultiBox Detector 23
     - 3.1.3 "YOLO: You Only Look Once" object detector 25
     - 3.1.4 Detector evaluation 25
   - 3.2 Facial image preprocessing 26
     - 3.2.1 Landmark detection 26
     - 3.2.2 Landmark detection evaluation 28
     - 3.2.3 Image transformations 28
   - 3.3 Face recognition 30
     - 3.3.1 Triplet loss 30
     - 3.3.2 Additive Angular Margin Loss 31
     - 3.3.3 Feature extraction evaluation 32
   - 3.4 Nearest neighbour search 33
     - 3.4.1 Vector similarity 34
     - 3.4.2 *k*-means clustering 35
     - 3.4.3 Hierarchical clustering 35
4. IMPLEMENTATION 37
   - 4.1 Data 37
   - 4.2 Hardware 38
   - 4.3 Detecting faces from image 39
   - 4.4 Aligning facial image 42
   - 4.5 Extracting face embedding 45
   - 4.6 Classifying embedding 46

4.7 System implementation 51

5. RESULTS AND DISCUSSION 53

5.1 Face detection 53

5.2 Facial image alignment 55

5.3 Face recognition 56

6. CONCLUSION 59

REFERENCES 62

APPENDIX A: VISUALIZATIONS OF *K*-MEANS AND CHUNX CLUSTERING RESULTS 67

LIST OF FIGURES

Figure 2.1. *Structure of a simple neural network with one hidden layer. Symbols x_n are n inputs to the network, y_n are the outputs and h_n describes the stage on hidden layer between the input and output.* 6
Figure 2.2. *ReLU and logistic sigmoid activation functions.* 8
Figure 2.3. *The affect of learning rate in optimization as a function between the neural network parameters $\mathbf{\Theta}$ and the loss. a) well chosen (blue) b) too big (green) c) too small (orange) learning rate.* 10
Figure 2.4. *Different scenarios related to optimization. a) the minima is hidden "behind obstacle", b) only local minima is found, and c) with good initial parameters the minima can be found.* 10
Figure 2.5. *VGG16 network architecture [1].* 13
Figure 2.6. *Convolutional layer with 4x4x1 input and 2x2 kernel with stride 1.* 14
Figure 2.7. *Demonstration of a single filter's ability to extract features (edges) from grayscale image (image taken from VGGFace2 dataset [2]).* 15
Figure 2.8. *Max pooling, a) using 2x2 kernel and stride=1, b) 3x3 kernel with stride=2.* 15
Figure 2.9. *Inception module.* 17
Figure 2.10. *Residual block.* 18
Figure 2.11. *Building blocks of a) MobileNetV1 and b) MobileNetV2 with two different strides.* 18

Figure 3.1. *Pipeline of the system* 20
Figure 3.2. *Desired face detector output (image taken from VGGFace2 dataset [2]).* 21
Figure 3.3. *Computation of integral image.* 22
Figure 3.4. *Visualization of integral image (image taken from VGGFace2 dataset [2]).* 22
Figure 3.5. *Example of different Haar features (A, B, C and D) within the relative detection window (redrawn from [3]).* 23
Figure 3.6. *Architecture of the SSD network (modified from [4]).* 24
Figure 3.7. *An illustration of SSD feature maps. a) groundtruth image with predicted boxes for two classes, person (green) and face (red), b) four bounding boxes per cell with 5×5 feature map, where only one of four boxes (marked with red) detects the face cell, c) four bounding boxes per cell with 3×3 feature map, where one of four boxes (marked with green) detects the person in cell and one of four boxes in another cell detects a face (marked with red) (groundtruth image taken from VGGFace2 dataset [2]).* 24
Figure 3.8. *Intersection over union (IoU). The closer the predicted bounding box B is to groundtruth A, the higher the IoU is.* 25

Figure 3.9. *Face alignment process. Detect the five landmarks and align image so that the landmark can be found from same coordinates in the image. Locations marked with green point are considered as groundtruth locations and red are the detected landmarks from the image that is aligned. (images taken from VGGFace2 dataset [2])* 27

Figure 3.10. *Architecture of the ONet in MTCNN cascaded CNNs with the sizes of the outputs for each of the blocks.* 27

Figure 3.11. *Components of isometries, similarity transforms and affine transforms in group of perspective transforms: a) translation, b) rotation, c) scale and d) shear. The anchor point is set to upper left corner.* 29

Figure 3.12. *Visualization of triplet selection problem by locating positive and negative samples of the triplet based on distance to the anchor (a): b) easy positive is close to the anchor (a), c) hard positive is further away from the anchor (a), d) hard negative is closer to anchor (a) than hard positive (c), e) semi-hard negative is little further from anchor (a) compared to hard positive (c), f) easy negative is far away from the anchor.* 31

Figure 3.13. *Confusion matrix in binary classification.* 32

Figure 3.14. *Example of hierarchy of the CHUNX. Nodes including the samples from data* ***X*** *have gray background. Values in the nodes refer to the* i*:th angle related to the corresponding eigenvector which is the most significant component that defines the cluster.* 36

Figure 4.1. *Bounding boxes detected with different frameworks. (images taken from VGGFace2 dataset [2])* 40

Figure 4.2. *Differences in detection results related to framework. On upper row, Darknet framework with YOLOv3 Tiny CNN and Tensorflow with MobileNetV1+SSD are used. On bottom row, OpenCV library is used with both detectors. (image taken from VGGFace2 dataset [2]).* 41

Figure 4.3. *Examples of implemented unified face cropping (images taken from VGGFace2 dataset [2]).* 41

Figure 4.4. *Examples of facial landmark detector output. a)Dlib detector, b) MTCNN ONet detector (images taken from VGGFace2 dataset [2])* 43

Figure 4.5. *Affect of the cropping in landmark detection with different detectors. (images taken from VGGFace2 dataset [2])* 44

Figure 4.6. *Differences between the alignments. a) unaligned, b) similarity aligned and c) affine aligned. (images taken from VGGFace2 dataset [2])* 45

Figure 4.7. *ROC curves for LFW evaluation with different alignments for triplet and ArcFace losses.* 47

Figure 4.8. *Search speed as a function of search space size with different maximum cluster sizes (s) with dynamically adjustable* k*-means clustering classifier.* 48

Figure 4.9. *Search speed as a function of search space size with different number of clusters (s) using static* k*-means clustering classifier.* 49

Figure 4.10. *Search speed as a function of search space size with different maximum cluster sizes (s) utilizing CHUNX based clustering classifier.* 50

Figure 5.1. *Average images with different alignments utilizing test partition of VGGFace2 dataset.* 56

Figure 5.2. *Distribution of the pairs with same identity and different identities with LFW (left) and VGGFace2 (right) datasets with magnitude on y-axis.* 57

Figure 5.3. *Search speed as a function of search space size with different classifiers with small number of groundtruth embeddings.* 58

Figure 5.4. *Search speed as a function of search space size with different classifiers with high number of groundtruth embeddings.* 58

Figure A.1. *Visualization of* k*-means clustering with 96 clusters utilizing 9035 identities in VGGFace2 dataset, where one image from each cluster was picked (images taken from VGGFace2 dataset [2]).* 67

Figure A.2. *Visualization of CHUNX clustering, where maximum number of images per cluster was set to 50 and only clusters with more than 16 facial images were chosen, utilizing 9035 identities in VGGFace2 dataset, where one image from each cluster was picked (images taken from VGGFace2 dataset [2]).* 68

Figure A.3. *Visualization of CHUNX clustering tree, where maximum number of images per cluster was set to 100 and only clusters with more than 16 facial images were chosen, utilizing one image from 9035 identities in VGGFace2 dataset. The nodes in the leaf include the facial images. For the nodes that have only one child in the visualization, the other children are due to the defined minimum cluster size.* 69

LIST OF TABLES

Table 4.1. *Optimal thresholds and corresponding accuracies with different alignments utilizing triplet and ArcFace losses in training.* 46

Table 4.2. *Accuracy of the nearest neighbour search with different methods for finding correct match within one, five and ten nearest neighbours using two different alignments. Base classifier accuracies from nearest neighbour search and the rest are approximations of it utilizing cluster structure. For dynamic* k*-means and CHUNX, maximum cluster size was set to 60 and for static* k*-means, number of clusters was 100.* 50

Table 5.1. *Detector's performance.* 53

Table 5.2. *Face detector inference times in milliseconds (ms).* 54

Table 5.3. *Landmark detector performance for each of the five detected landmarks. The scores are percentages of the width of the image.* 55

Table 5.4. *Landmark detector inference times in milliseconds (ms).* 55

Table 5.5. *Accuracy of the feature extractor of this thesis compared to other methods.* 56

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
ArcFace	Additive Angular Margin Loss
AUC	Area Under Curve
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
DoF	Degrees of Freedom
FDDB	Face Detection Data Set and Benchmark
FLOPS	Floating Point Operations Per Second
FPR	False Positive Rate
FPS	Frames Per Second
GPU	Graphics Processing Unit
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
IoU	Intersection over Union
LFW	Labeled Faces in the Wild
mAP	Mean Average Precision
MTCNN	Multi-task Cascaded Convolutional Networks
NMS	Non-Maximum Suppression
OpenCV	Open Source Computer Vision Library
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristics
SGD	Stochastic Gradient Descent
SSD	Single Shot MultiBox Detector
TPR	True Positive Rate
YOLO	You Only Look Once, an object detection framework

1. INTRODUCTION

Over the past decade, numerous *machine learning* applications have become familiar to many people. For the end user, these applications are usually advertised as "smart" solutions that utilize *artificial intelligence* (AI) to solve different problems. Although the concepts behind the neural networks (1943) [5] and machine learning (1959) [6] that are the key elements of today's state-of-the-art AI applications, have been introduced over 50 years ago, the performance of the AI solutions have improved drastically within the past five years.

The main reasons behind this rapid development include the growth of data collection and an increase in computational power. With bigger amount of data, it is possible to create more complex machine learning models which provide increased performance in terms of accuracy, since the data most likely covers bigger amount of possible scenarios in a real life environment. The term *deep learning*, refers to these complex machine learning models that can be trained to find the key features from the data and are capable of classifying the data in greatly varying environment.

In the field of computer vision, the current state-of-the-art deep learning models can solve numerous problems that are too challenging for more simple machine learning models. However, the most of these models are computationally too heavy for today's mobile devices and embedded systems. They often require a *graphics processing unit* (GPU) in order to be fast enough to run in real-time with video streams.

Face recognition is one of the earliest applications in the field of computer vision. The topic has been researched widely in the past and numerous solutions to this problem have been proposed. However, especially more simple methods from the 2000s were not able to solve the problem in an unconstrained environment. After all, in the last few years, the deep learning methods utilizing *convolutional neural networks* (CNN) such as the DeepFace (2014) [7] and FaceNet (2015) [8] have been capable of solving the problem and outperforming the human accuracy in recognition benchmark. [9]

Face recognition is considered as a difficult machine learning problem since the appearance of the faces vary, for example skin tone, makeup, eye glasses and facial hair cause challenges for the recognition. Also, the facial expressions and aging might break a face recognition system [10]. In an unconstrained environment, pose, illumination and occlusions caused by obstacles must be considered in face recognition system development [10]. In addition, there are vulnerabilities in face recognition systems related to presentation attacks [11]. For example it has been reported multiple times lately, that the face recognition systems of mobile devices might for example accept a printed image of a person while accessing their

personal mobile devices.

There are applications for face recognition in many fields. Some of the most important applications for face recognition are related to safety. Face recognition can be used for person identification, for example in access control and video surveillance systems. [10]

This thesis covers a development process of a complete face recognition system, which is capable of recognizing reliably previously known people based on frames captured from video stream. The main focus in this thesis is in classification of the facial images. However, also other parts of the system are covered, since they must also be implemented in order to estimate the performance of the complete system.

The goal of the thesis is to develop a robust face recognition system that is capable of classifying facial images that are found from video stream, with high confidence. It should run on NVIDIA Jetson TX2 hardware or similar embedded platform, in real-time with reasonable frame rate (at least 20 frames per second (FPS)). The complete system includes multiple components. First, a reliable face detector that is capable of finding faces from the frames of the video stream, which are at least partially visible (20% minimum), is needed. These faces should be classified by comparing them to the set of known facial images with at least 98% accuracy. In addition, the number of false positives, where a sample is recognized to incorrect identity, should be minimized although it might weaken the overall accuracy. The search speed for the nearest neighbour search should be logarithmically proportional to the number of the known facial images.

Chapter 2 explains the theory behind the methods that are used. The theory starts from the basics of machine learning which forms the fundamentals to the used methods. After that, the term deep learning is explained, which leads to an introduction of the neural networks. After all, the building blocks of the CNN and the concepts that are required for training a CNN are presented based on principles related to the neural networks.

The different tasks related to the face recognition problem are presented in Chapter 3. First, the pipeline that will be followed in the implementation, is defined. Based on the pipeline, the problem is divided to smaller steps that can be solved individually. Then, the different tasks are introduced in detail and methods for solving them are covered. Also, the evaluation of the methods that are introduced for solving the tasks, are studied. Chapter 4 describes the implementation of pipeline steps. It focuses on describing, how the methods are utilized for solving the steps and visualizing the output of the implemented methods. Also, the datasets and the environment for the implementation are covered. Based on the implementation, the solution for the face recognition system is described by collecting a solution for each of the steps.

The results of implementation are evaluated and discussed in Chapter 5. The performance of the methods for each step of the pipeline are evaluated with commonly used performance metrics. The conclusion in Chapter 6 sums up the achieved results and reviews the thesis.

The solution for the problem stated in the introduction is presented based on the results and development ideas are discussed. Also, the requirements for the system presented in this chapter are revisited and compared to the results.

2. THEORY

In order to develop system for face recognition, the main concepts behind the used methods are covered. First, the term machine learning is defined. Then, the difference between the traditional machine learning methods and deep learning is explained. Finally, the techniques and methods for developing deep learning models for solving the presented problems are introduced.

2.1 Machine learning

A person collects huge amount of data every day using all of the available senses. These include smell, taste, vision, sensory, hearing and balance. Based on this collected data, numerous decisions are made on a daily basis. Some of these decisions require a lot of collected knowledge, but on the other hand some decisions are made without prior knowledge by trying to adopt the knowledge to an unknown situation.

Data can also be collected using different sensors, that presents the collected data in generally used format such as numbers, text or images. This kind of data can also be analysed and used by a human, but it is highly subjective and limited by ones ability to collect and understand only small amount of data. In addition, the simple well formatted abstract problems that are the most difficult to solve for a human, are the easiest to apply machine learning methods [12, p. 2].

In machine learning, decisions are made based on available data using computer. The available data is used to create a model, that can for example classify or detect patterns from the data. It is also possible to predict unseen data to do estimations for the future. [13, p. 1] Compared to the human, machine learning methods can utilize much more data to form the model. In general, the more data is available, the easier it is to develop good models. On the other hand, complex problems related to vision or hearing, which are intuitive for humans, are very difficult for machine learning models due to the high amount of possible scenarios [12, p. 1–2]. The solution for these complex problems is discussed in Section 2.2.

The main goal in machine learning model development is to train a model that performs well with the data that has not been used in the development of the model. Training the model can also be called *fitting* which refers to adjusting the model parameters to fit to available data. In training, the variables in the machine learning model are adjusted to minimize the *cost function*. This minimization process is discussed in Section 2.3.3. Trained machine learning model can later be used for example predicting output or classifying new data, that has not been available while training the model. The better the model performs with

previously unseen data, the more generalized it is. The complex models are capable of learning very detailed features from the data. However, if the model learns these details, its overall performance may weaken. In this case, the model *overfits* to the data. On the other hand, if the model is too simple for the problem, it *underfits* and is not capable of learning all the key features from the data. [12, p. 96–113]

The overfitting problem can be solved by carefully estimating the requirements for machine learning model complexity and the data used in model development. For the data, *cross-validation* can be used to group data so that the machine learning model performance can be objectively estimated while developing the model. In cross-validation, the goal is to divide data to training and validation sets. The validation set is only used to estimate the model performance, not in training. Both the training and the validation datasets should include samples from original dataset so that they both represents the whole dataset, but the two parts cannot include same samples. [14, p. 241–249]

Machine learning methods can be divided into three main categories. One of them, *supervised learning*, outputs for the data are known and the relations from specific inputs to outputs should be learned by the model. Supervised learning suits well for example in classification of the input samples and in object detection. [13, p. 2–9]

However, annotating the known outputs for the data is time consuming and therefore expensive. The annotation is mostly done by humans, and there might also be some differences between the annotated output depending on the person that annotates the data. If the outputs for the data are unknown, another main category, *unsupervised learning* finds the relations from data. It can be used for example in dimensionality reduction and knowledge mining, where hidden patterns, relations or associations from the data that could classify the samples, are searched using for example clustering or decision trees. [12, p. 142–148] Supervised learning can also support unsupervised learning by steering and verifying the unsupervised learning process. This is called *semi-supervised learning*. [12, p. 240]

2.2 Deep learning

The simple machine learning algorithms such as k-means, use k nearest neighbours from data to classify samples, perform well with well formatted problems and simple data. Well formatted problems refer, for example, to simple classification problems where the data is classified to few available classes. In these cases, the dimensionality of the data and the number of the outputs in the system should be relatively low. While the data becomes more complex (for example dimensionality is higher and there are noise in the data), they are not suitable. [12, p. 137–151] These cases include for example object detection and feature extraction from the images.

Machine learning models that are capable of solving these complex problems, require more representational capability, which can come from the depth of the model where multiple

processing steps are performed in a series. Each of the steps learn new more abstract features based on the previous step. [13, p. 995–1000] The deep learning models can have millions of parameters on numerous layers. The more parameters and layers there exist, the more challenging it is to train the network and understand its strategy for decision making. Therefore, deep learning methods are also called as *black box* methods.

Development and usage of deep learning models is similar to traditional more simple machine learning models discussed in Section 2.1. The main difference is related to model training. While the deep learning model is more complex, the minima of the loss is harder to find. In addition, a complex deep learning model can learn very detailed representation (fine details) from the data, which causes overfitting. The most simple way to avoid overfitting is to use *early stopping* where the number of *epochs* (the whole set of training data is shown once to the model [14, p. 397]) run during the training is limited so that the common features in the data are learnt, but the unique details are left out from the model. [12, p. 224–225, 241–243]

2.3 Neural networks

Deep learning refers to the neural network models that are highly influenced by the modelling of the human brains. Single unit in neural network tries to replicate a brain cell which structure was first introduced by McCulloch in 1943 [5] and later modeled as a perceptron by Rosenblatt in 1958 [15]. In neural networks, perceptrons are placed on layers from input to output layer. There can also one be or more layers of perseptrons between the input and output, these layers are called *hidden layers* [14, p. 392-393]. The structure of a simple neural network is illustrated in Figure 2.1.

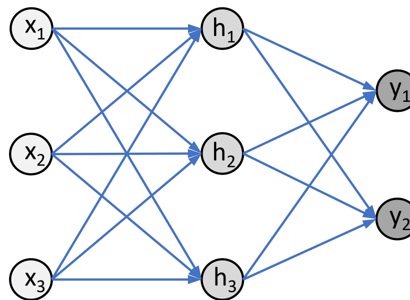

Figure 2.1. *Structure of a simple neural network with one hidden layer. Symbols x_n are n inputs to the network, y_n are the outputs and h_n describes the stage on hidden layer between the input and output.*

The neural network in Figure 2.1 is called *feedforward neural network*, which refers to its structure that does not include any feedback connections to previous layers. The whole network can be modelled as a function f that maps the input values to outputs as

$$\mathbf{y} = f_{\mathbf{\Theta}}(\mathbf{x}), \tag{2.1}$$

where $\mathbf{x}$ is an input vector to the network, $\mathbf{y}$ is an output and $\mathbf{\Theta}$ models the neural network parameters. While training the neural network, these parameters are optimized to learn the mapping from input to output. With three layer neural network as presented in Figure 2.1, the mapping function f consists of multiple functions where each function represents a layer as

$$f_{\mathbf{\Theta}}(\mathbf{x}) = f_{\mathbf{\Theta}_2}(f_{\mathbf{\Theta}_1}(\mathbf{x})), \tag{2.2}$$

where $f_{\mathbf{\Theta}_1}$ is the mapping from input to hidden layer, $f_{\mathbf{\Theta}_2}$ is mapping from the hidden layer to output layer. [12, p. 164–165]

In addition to the neural network architecture (number of the layers and the width of the each layer), also *activation function*, optimizer and cost function (loss) must be designed for fitting a neural network. While fitting the network, also regularization (Section 2.3.4) must be considered in order not to overfit. [12, p. 172–173]

2.3.1 Activation

If the neural network were using only the weights of the perceptrons in order to calculate the output of the network, it would be a linear function, which cannot be used in more advanced classification tasks that include non-linearity. In neural networks, non-linearity is implemented using a non-linear activation function for the outputs of the perceptrons. One of the most used activation functions is *rectified linear unit* (ReLU) which is defined as

$$g(z) = \max(0, z), \tag{2.3}$$

where $z \in \mathbf{R}$ is the output of a perceptron. [12, p. 168–172][16] There are also different specializations such as ReLU6, which limits the maximum of the output to 6.

However, in a classification task, wanted output from the neural network is a probability in range [0,1] for each of the possible classes. Therefore, ReLU cannot be used as an activation function on output layer of the neural network in classification task. For binary output (classification probability), a *logistic sigmoid* function can be used for instance. It is defined as

$$g(z) = \frac{1}{1 + \exp(-z)}. \tag{2.4}$$

In a multiclass case, a probability for each class i is presented as a separate output node. For that purpose *softmax function*

$$g(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{2.5}$$

can be used. The sum of the probabilities for classes equals to 1. [12, p. 65–67, 180–181] ReLU and logistic sigmoid functions are presented in Figure 2.2.

The most of the current state-of-the-art neural networks utilize ReLU or softmax activation functions. Still, there are also numerous other activation functions and different variations

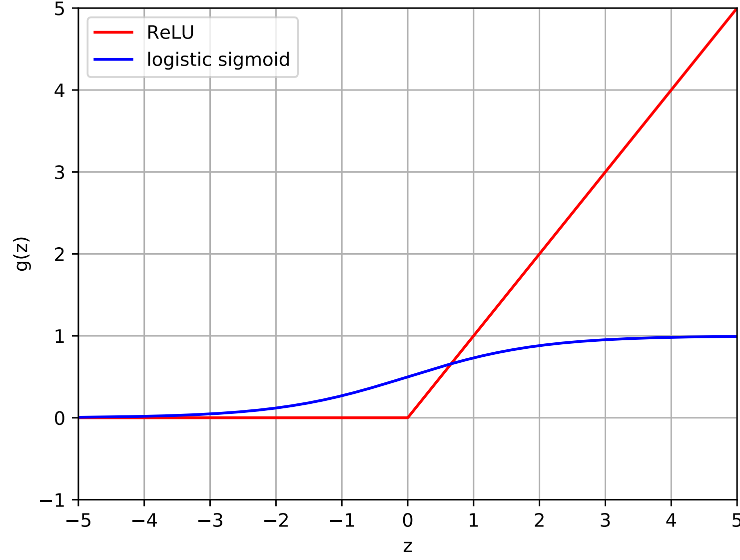

Figure 2.2. *ReLU and logistic sigmoid activation functions.*

to them available that have been developed for different purposes. [17] Activation function adds hyperparameters that must be tuned while developing neural network based deep learning model. The optimal function can be found only via trial and error, but the most used functions are a good starting point.

2.3.2 Loss

In order to estimate the performance of the neural network during the training, and also to steer it, the loss L must be defined. It measures, how far the estimated output of the neural network with input $\mathbf{x} \in \mathbf{R}^D$, where D defines dimensionality of the data, is from the desired output. One simple measure for measuring the output $\mathbf{y} \in \mathbf{R}^N$ with length N is an *euclidean loss* (ℓ_2 loss). It is formed as

$$L_{euclidean}(f_{\mathbf{\Theta}}(\mathbf{x}), \mathbf{y}) = \|\mathbf{y} - f_{\mathbf{\Theta}}(\mathbf{x})\|_2^2 = \sum_{i=1}^{N} (y_i - f_{\mathbf{\Theta}_i}(\mathbf{x}))^2, \tag{2.6}$$

where $f_{\mathbf{\Theta}}(\mathbf{x})$ is presented in Equation 2.1. In classification tasks, the output of the network is a vector of probabilities for a sample being in class c among all the possible classes C. Therefore, a neural network can be modelled as a probability function $p_{\mathbf{\Theta}}$ with neural network weights $\mathbf{\Theta}$. In this case, *categorical cross-entropy* loss is described as

$$L_{categorical\ cross-entropy}(p_{\mathbf{\Theta}}(\mathbf{x}), \mathbf{y}) = -\sum_{c=1}^{C} y_c \log p_{\mathbf{\Theta},c}(\mathbf{x}), \tag{2.7}$$

which computes the loss as a sum of probabilities $p_{\mathbf{\Theta},c}$ for $\mathbf{x}$ to belong in class c, can be used. [14, p. 395–396]

The choice of a loss function is respect to the problem that is solved. It should present the differences related to it. After all, the goal is to minimize the loss that describes the problems related to the output of the model that is trained.

2.3.3 Optimization

In traditional optimization problem, the goal is to find optimal solution for the problem. In machine learning, a known set of data is utilized for optimizing model parameters $\mathbf{\Theta}$ to map input samples $\mathbf{X} \in \mathbf{R}^{D \times M}$, where M is the number of training data samples, to corresponding outputs $\mathbf{Y} \in \mathbf{R}^{N \times M}$ that minimize the cost function $J(\mathbf{\Theta}; \mathbf{X}, \mathbf{Y})$. However, the machine learning model is optimized with limited amount of data while the model should work well also with unseen data. Therefore, the model optimization is only a guess based on available data. [12, p. 271]

While optimizing a deep learning model, cost function can be considered as an average of the loss over the training dataset. With carefully evaluated data and generalization, the cost function of the model trained with available training dataset with empirical distribution, represents closely the model's cost function with true distribution of the data. With this assumption, the cost function can be modelled as an average of the loss over M samples of data as, [12, p. 272–273],

$$J(\mathbf{\Theta}; \mathbf{X}, \mathbf{Y}) = \frac{1}{M} \sum_{i=1}^{M} L(f_{\mathbf{\Theta}}(\mathbf{x}_i), \mathbf{y}_i). \tag{2.8}$$

In neural network training, based on cost function value, the parameters of the network $\mathbf{\Theta}$ are adjusted. The adjustment (training the network) is performed by iterating over the training data and constantly updating the parameters. The parameter update utilizes an *optimizer*. One of the most used optimizer is *stochastic gradient descent* (SGD). In SGD, the parameters are updated based on the *minibatch*, which is a subset m of the training data samples. For minibatch, the gradient of the cost function

$$\boldsymbol{g} = \nabla_{\mathbf{\Theta}} J(\mathbf{\Theta}; \mathbf{X}, \mathbf{Y}) = \frac{1}{m} \nabla_{\mathbf{\Theta}} \sum_{i=1}^{m} L(f_{\mathbf{\Theta}}(\mathbf{x}_i), \mathbf{y}_i) \tag{2.9}$$

is computed. After processing a minibatch, the computed gradient can be used to update neural network parameters

$$\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \epsilon \boldsymbol{g}, \tag{2.10}$$

where ϵ is a *learning rate*, which defines the size of the change in parameters based on the computed gradient. [12, p. 274–278, 290–291]

Learning rate is chosen to be big enough to reach the minimum of the cost function yet small enough to hit it. It can also be changed during training, for example to get close to the minima first, and then decreasing the learning rate to reach the minima. The affect of the

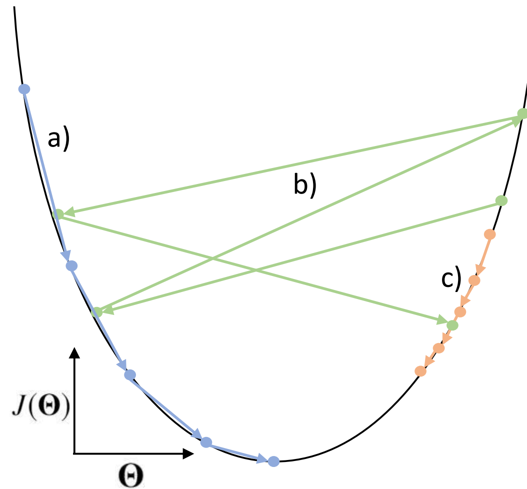

Figure 2.3. *The affect of learning rate in optimization as a function between the neural network parameters* $\mathbf{\Theta}$ *and the loss. a) well chosen (blue) b) too big (green) c) too small (orange) learning rate.*

learning rate is shown in Figure 2.3. In the figure, the optimization problem is simplified into two dimensions, where the cost function $J(\mathbf{\Theta})$ is modelled in two dimensional space as a function of parameters $\mathbf{\Theta}$.

In deep learning, the optimization of the parameters is a very difficult task since their dependency on each other is complex. While the gradient is used in finding the optimal parameters, the minima of the cost function might be behind an "obstacle" that gradient may be unable to handle. In addition, neural network training can "get stuck" to the local minima of the cost function although more optimal solution is available. Also the initial parameters can define, where the minima of the gradient is eventually found. [12, p. 274–289] Figure 2.4 illustrates these situations.

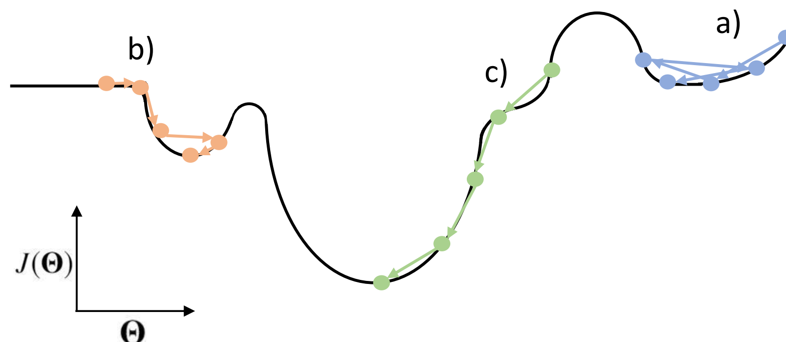

Figure 2.4. *Different scenarios related to optimization. a) the minima is hidden "behind obstacle", b) only local minima is found, and c) with good initial parameters the minima can be found.*

After all, optimization is a difficult task where numerous components, and their hyperparameters, related to neural network such as network architecture, activation functions, used losses and optimizer, affect the performance of the trained model. The performance is very difficult or impossible to predict before training.

2.3.4 Regularization

In optimization, the goal is to minimize the loss. However, the loss minimization can lead to overfitting especially if the used machine learning model is complex and it is capable of representing fine details. If the model overfits, the model does not work well with the data that has not been used in training the model. As stated in Section 2.1, the goal in machine learning model development is to create a model, that performs well with the data that model has not seen before. Regularization is used to meet that goal. In addition, controlling the training process based on metrics or carefully choosing the training data that were discussed in Section 2.1, regularization can also be implemented by adding penalties based on model parameters during the optimization [12, p. 226].

For optimization, the norm penalty $\Omega(\Theta)$ can be added to the total cost function $\tilde{J}$ as

$$\tilde{J}(\mathbf{\Theta};\mathbf{X},\mathbf{Y}) = J(\mathbf{\Theta};\mathbf{X},\mathbf{Y}) + \alpha\Omega(\mathbf{\Theta}), \tag{2.11}$$

where hyperparameter $\alpha > 0$ defines the weight of the norm penalty. The most used norm penalties are l_2 and l_1 penalties. The l_2 is defined as

$$\Omega_{l_2}(\mathbf{\Theta}) = \|\mathbf{\Theta}\|_2 = \sqrt{\sum_i \Theta_i^2} \tag{2.12}$$

and l_1 is defined as

$$\Omega_{l_1}(\mathbf{\Theta}) = \|\mathbf{\Theta}\|_1 = \sum_i |\Theta_i|. \tag{2.13}$$

The difference between l_1 and l_2 losses is that l_1 loss penalises for single high absolute value weight and l_2 for norm of the weights. [12, p. 226–231]

Penalty can be added to every component in neural network. For example *Keras* deep learning library [18] provides an *application programming interface* (API), where penalty can be added to network weights, bias and output of the activation function for each layer. Bias is an additional constant that is added to each neuron or other unit in the network [14, p. 392–393].

Generalization can also be added to the neural networks by randomly disabling temporarily some neurons or other neural network units with specific probability during training which decreases network's representational power. This method, *dropout*, was presented by Srivastava *et al.* in 2014 [19]. In dropout, a configured amount of units are temporally disconnected from the network while training so that they are not used in evaluating network output on that minibatch. By randomly changing the disconnected units between

the epochs, the training uses different sub-nets of the original neural network in training. After training these sub-nets can be used together as a one complete network, where the trained weights are scaled from training with unit's probability to be present in a sub-net.

Batch normalization can also be used in accelerating the convergence during the neural network training and regularizing the output. It normalizes the inputs within a training batch for the layer in order to avoid *covariate shift* which means that the distribution of the input for specific network part changes while the parameters in preceding layers change. Because of the distribution change in preceding layer's output while training, the following layers must also learn the changes on the preceding layers, which amplifies the effect on parameter update throughout the network and slows the training process. [20]

2.4 Convolutional neural networks

Traditional neural networks presented in Section 2.3 suit well for one dimensional data. However, with multi-dimensional data such as images, traditional networks are impractical. CNNs provide an effective way for extracting features from multi-dimensional data and they are used in numerous machine learning applications for that purpose. [12, p. 326] CNNs are capable of achieving similar performance compared to traditional fully connected networks, similar to one in Figure 2.1, with significantly lower amount of parameters because the training is easier due to smaller amount of parameters and fewer connections between the parameters. In addition, current GPUs are well optimized for calculation of two dimensional convolution which is the cornerstone for CNNs. [21]

CNN includes one or more convolutional layers (Section 2.4.1) in the network architecture. They are capable of reducing the amount of needed parameters drastically compared to the traditional networks. CNNs are based on filtering the image with numerous filters at each layer in order to calculate the output of the layer. Therefore convolutional layers share parameters within a layer, which can also be used as a regularization method. In addition, the convolution operation of the filtering is computationally lighter than matrix multiplication that would be needed with traditional neural networks. [12, p. 326–335]

Since the AlexNet in 2012 [21], CNNs have significantly improved the state-of-the-art classification accuracy in annual ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) which uses a subset of ImageNet dataset [22]. While the AlexNet included five convolutional (Section 2.4.1) and three dense (Section 2.4.3) layers, later the depth and the complexity of the architecture has increased. Some of these architectures are presented in Section 2.4.4.

The architecture of the commonly used CNNs consist of components that can be considered as building blocks of the network. Each block contains usually convolutional layers, an activation function for the output of the convolutional layer which is similar to the traditional neural networks, and a pooling layer which performs sub-sampling. [12, p. 336] For example in classification problems or feature vector extraction, the desired output of

the network is one dimensional. In these cases one or more dense layers are used to map the outputs from convolutional layers to desired outputs. These basic layers are used to form building blocks for various CNN architectures that have been developed for different deep learning applications. For this thesis, focus is on computationally light CNNs for image processing. This architecture is visualized in Figure 2.5 utilizing an architecture of the VGG16 [23] network.

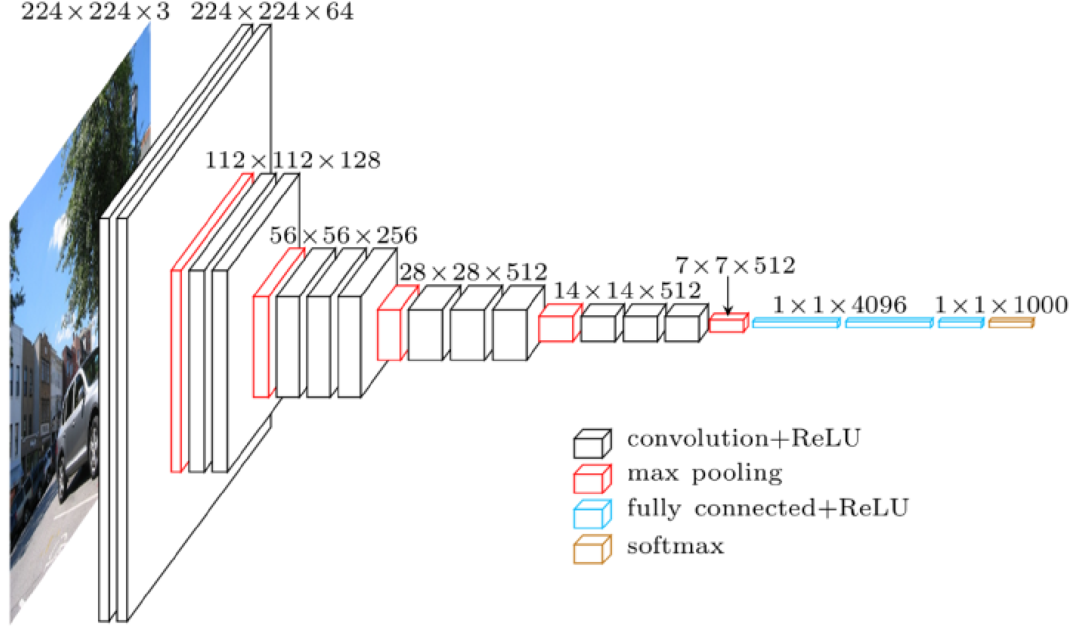

Figure 2.5. *VGG16 network architecture [1].*

2.4.1 Convolutional layer

The task of the convolutional layer in most of the image related problem is to filter the $W \times H \times D$ (*width*×*height*×*depth*) sized input with the set of N filters. Filtering is a convolutional operation between the input image I and filtering kernel K, which can be represented as a discrete case with images. For each location (i, j) in image the result of convolution is

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{2.14}$$

where n and m define the kernel size. The convolution operation is commutative, which is possible because the kernel relative to the image I is flipped (kernel index decrease while the index of image increases). By utilizing commutation $(K * I)(i,j) = (I * K)(i,j)$ for Equation 2.14 and using *cross correlation* function instead of convolution, where the kernel is flipped, the equation is modified to

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n), \tag{2.15}$$

which is used by the deep learning frameworks instead of the convolution. [12, p. 327–329]

An example of a convolutional layer is presented in Figure 2.6. The convolution operation is run through each pixel location in image matrix. The depth of the filter is same that the depth of the image matrix. The image kernel size $m \times n$ can be chosen freely. Usually 1×1, 3×3 or 5×5 kernel sizes are used in convolutional layers. In addition to kernel size, also *stride* must be chosen. It defines a step size for filtering (with stride=1 each pixel is used, stride=2 every second pixel etc.). The edges of the image require a special processing in order to fit the kernel also to the border pixels of the image. Two main approaches can be used. First, image can be padded with zeros to increase the image size in order to get output which width and height are same than in input. The other approach is run kernel only so that it fits completely inside the image. In this case, width and height of the output of the layer is smaller than the input.

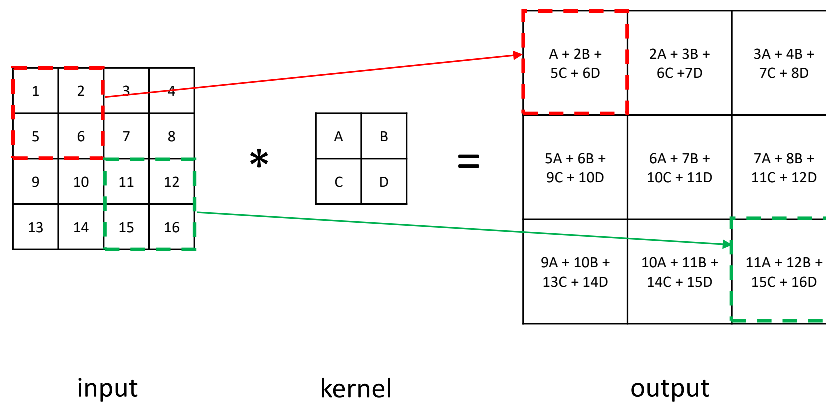

Figure 2.6. *Convolutional layer with 4x4x1 input and 2x2 kernel with stride 1.*

Filtering of the image with simple kernel is a powerful tool for feature extraction. For example, in Figure 2.7 edges are extracted form the image using kernel that subtracts surrounding pixel's values from each pixel in the image in order find the changes in pixel values.

As shown in Figure 2.7, a single filter is capable of extracting features from the image using only nine parameters. By filtering the image with multiple filters on a convolutional layer, it is possible to extract very detailed features with very few parameters (kernel weights). In addition, the number of the input channels for each of the convolutional layers can be increased to extract more information from the previous layer of the network. The convolution operation can also be applied depthwise to the input channels with three-dimensional filtering kernels that are capable of extracting features not only from each channel but also combining information from multiple channels.

2.4.2 Pooling layer

Pooling is used in CNN for reducing the size of the output by sub-sampling the output from the convolutional layer after the activation function is applied. The goal in pooling is to

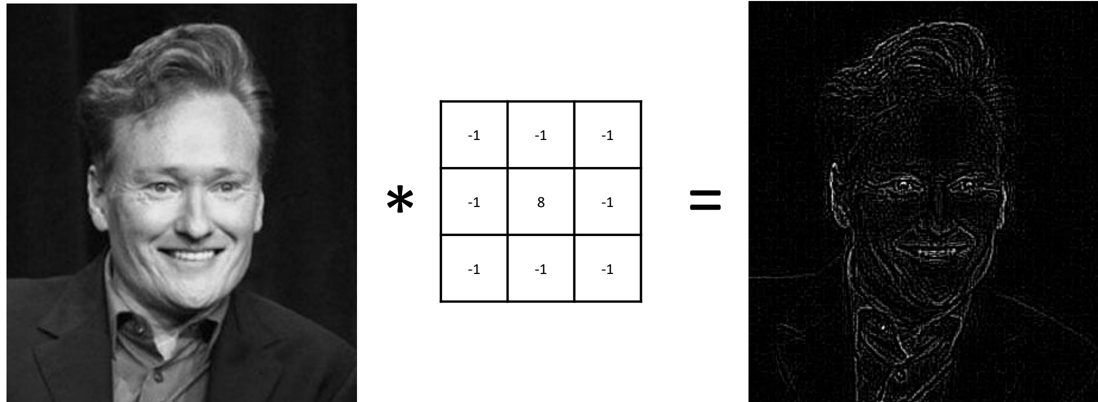

Figure 2.7. *Demonstration of a single filter's ability to extract features (edges) from grayscale image (image taken from VGGFace2 dataset [2]).*

remain the important features from the input and remove others. With CNN, *max pooling* is the most used pooling, which finds the maximum value within the pooling kernel and outputs that. Similar to convolutional layer, pooling kernel size and stride must be chosen. Figure 2.8 illustrates max pooling operation with different kernel sizes and stride. [12, p. 335–339]

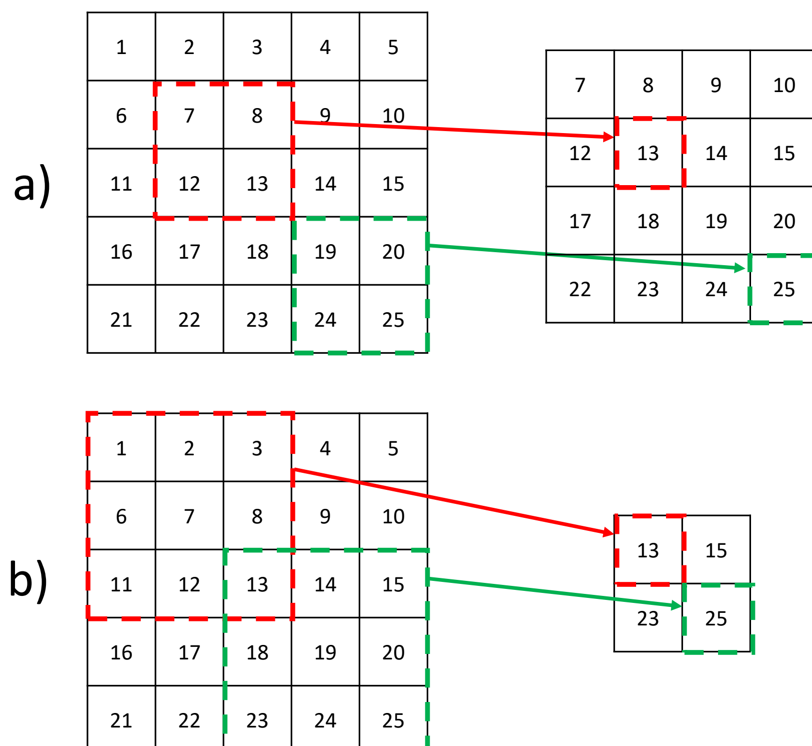

Figure 2.8. *Max pooling, a) using 2x2 kernel and stride=1, b) 3x3 kernel with stride=2.*

Pooling layer reduces the size of the input for the following layer in CNN. This reduces the total inference time of the CNN which means that the network is faster to process the

image. The inference time is one of the main problems related to neural network usage especially in real-time applications on hardware with limited computational capacity.

2.4.3 Dense layer

Dense layer refers to a fully connected layer of perceptrons similar to three-layer network presented in Figure 2.1, which performs linear transformation for input. Dense layer inputs a n-length vector of features and connects each of the elements in the input vector to every perceptron of the dense layer. The width of the layer m is defined by the number of perseptrons. For the output of each perceptron, an activation function is used for non-linearity. [12, p. 164–167]

In CNN, dense layers are usually used between the last convolutional layer and output to map the features extracted by the convolutional layers to output. Compared to the convolutional layers that share parameters efficiently, a dense layer has $n \times m$ parameters and they are not shared. In order to limit the network size (number of parameters in network), the size of the dense layer should be limited. It can be achieved by reducing the number of features in convolutional layers before flattening the network to dense layers. Flattening is an operation that creates a one dimensional vector from multidimensional output of the convolutional layer.

2.4.4 Commonly used architectures

During the last five years, numerous CNN architectures have been proposed for different purposes. Some architectures that have proven to perform well in common object recognition tasks for classifying objects in commonly used datasets such as COCO (Common Objects in Context) [24] or ImageNet [22]. Since the classification of images in these datasets can be considered to cover well any general task related to feature extraction from images, also the used network architectures suit well for various tasks. For this thesis, some commonly used CNN architectures are used as a complete network or some components from them are integrated to custom architecture.

Followed by AlexNet [21], the VGG16 network[23] with similar simple architecture was presented in 2014. Compared to AlexNet, VGG16 has more convolutional layers (16) and the network size in general was bigger. VGG16 architecture consists of five blocks of convolutional layers, max pooling after each block and three dense layers. Due to very big number of parameters (138 millions) and many computationally relatively heavy convolutional layers with big number of filters, the VGG16 network is not suitable for real-time applications.

In order to increase the models representational capacity, and therefore increase the performance, but at the same time limit the size of the network and computational complexity, the base of the CNN architecture must be re-designed. In 2014, the GoogLeNet

network [25], which has approximately 20 times less parameters (6.8 million) and ten times less needed FLOPS (floating point operations per second) compared to VGG16 while the result in ILSVRC competition is similar. The architecture of the GoogLeNet consists of *Inception* modules that replace traditional convolutional layers compared to VGG16. The *inception* module, which is presented in Figure 2.9, reduces the dimensionality of the data before expensive operations, which reduces the number of parameters and reduces computational complexity although the number of layers (22) is higher than in VGG16.

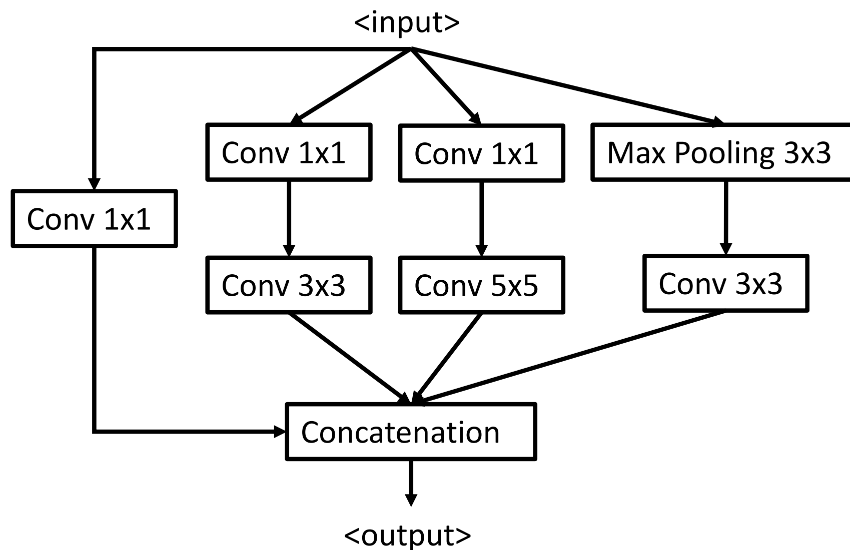

Figure 2.9. *Inception module.*

Another approach to replace traditional convolutional layers, similarly to inception modules, is residual blocks that was introduced in 2015 [26] by He *et al.*, which is named in CNN architectures as ResNet. The residual block, which is shown in Figure 2.10, forms the output as a sum of block input and convolutional layers' output. This approach allows deeper networks where the depth increases complexity instead adding parameters to each convolutional layer as in VGG16. In the paper He *et al.* reports that the depth of the network does not improve accuracy if traditional convolutional layers are used, because the problem becomes too difficult to optimize. Therefore, structures such as residual block are needed to increase the complexity of the model without adding parameters or computational complexity.

For utilizing CNNs in real time applications such as video processing, on mobile and embedded devices, these custom building blocks must be optimized more. Howard *et al.* proposed in 2017 MobileNet models for this purpose [27]. The idea in MobileNet is similar to GoogLeNet [25], computationally heavy convolution operations are split to two separable convolution operations that are lighter to compute. In MobileNet, first, depthwise convolution is applied to each channel with separate filter and then outputs of the depthwise convolutions are combined using convolutions with 1x1 kernel filters.

In 2018, the next generation of MobileNet, named as MobileNetV2, added residual blocks

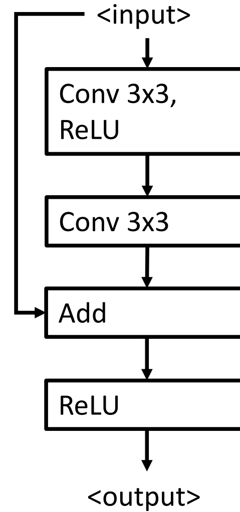

Figure 2.10. *Residual block.*

from ResNet [26] to the block structure [28]. These *bottleneck* blocks consist of two convolutions with 1x1 kernel which task is to scale down and up sample size, around the computational heavy convolution similar to layer in VGG16 [23] architecture. The blocks in both MobileNetV1 (the original MobileNet) and MobileNetV2 are illustrated in Figure 2.11. Both MobileNet versions use ReLU6 activation function.

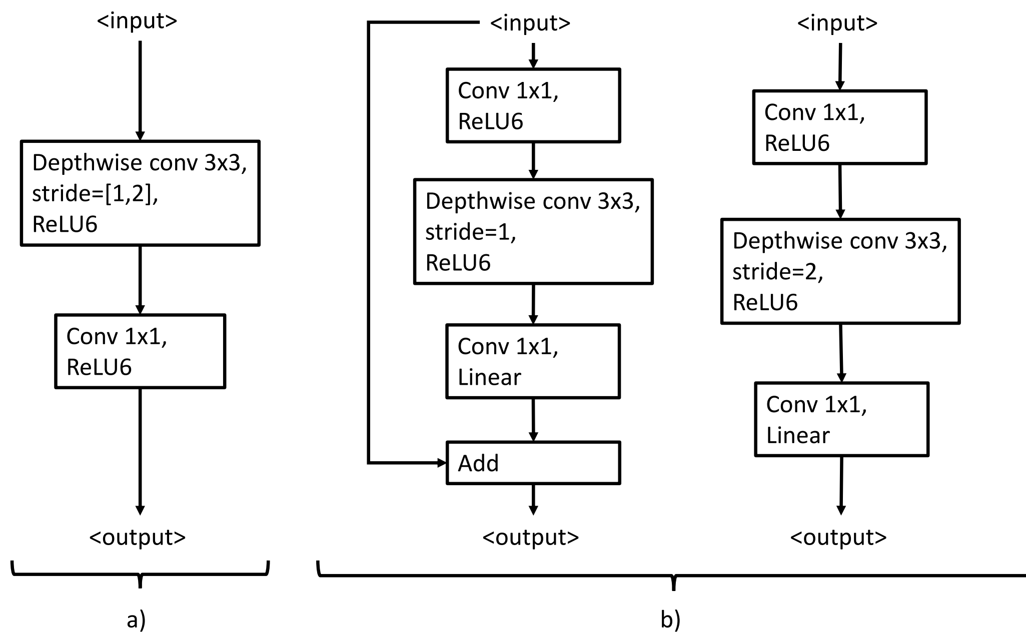

Figure 2.11. *Building blocks of a) MobileNetV1 and b) MobileNetV2 with two different strides.*

While training the presented architectures, also different regularization methods discussed in Section 2.3.4 are applied. For example ResNet, MobileNetV1 and MobileNetV2 utilize batch normalization [20]. In general, the deeper the network is, the more difficult it is to train. However, if fewer parameters and faster inference times are required, increasing the depth of the CNN is the only possibility to improve the representational capability of the model.

Transfer learning refers to using available machine learning models to solve new problems. The machine learning models may not work in new domain with different data than has originally been used in training. However, training models from scratch is time consuming and more challenging. By using a pre-trained model and applying transfer learning, the training of the new model can be eased. [29] There are numerous sources for open-source pre-trained models that include these commonly used architectures to solve different problems such as ILSVRC competition and object detection with COCO dataset.

3. PIPELINE

Face recognition from the video stream can be divided into three main steps. First, the faces are detected like any other object from the frames of the video stream. Based on detection, the detected faces can be extracted from the frames and classified. Before classification, the extracted facial images can be pre-processed. Recognition includes two steps, first the features are extracted from the facial image and then the resulting embedding, which is a feature vector including features extracted from the facial image, is classified by comparing it to the known set of embeddings in order to find the best match. Figure 3.1 presents this recognition pipeline.

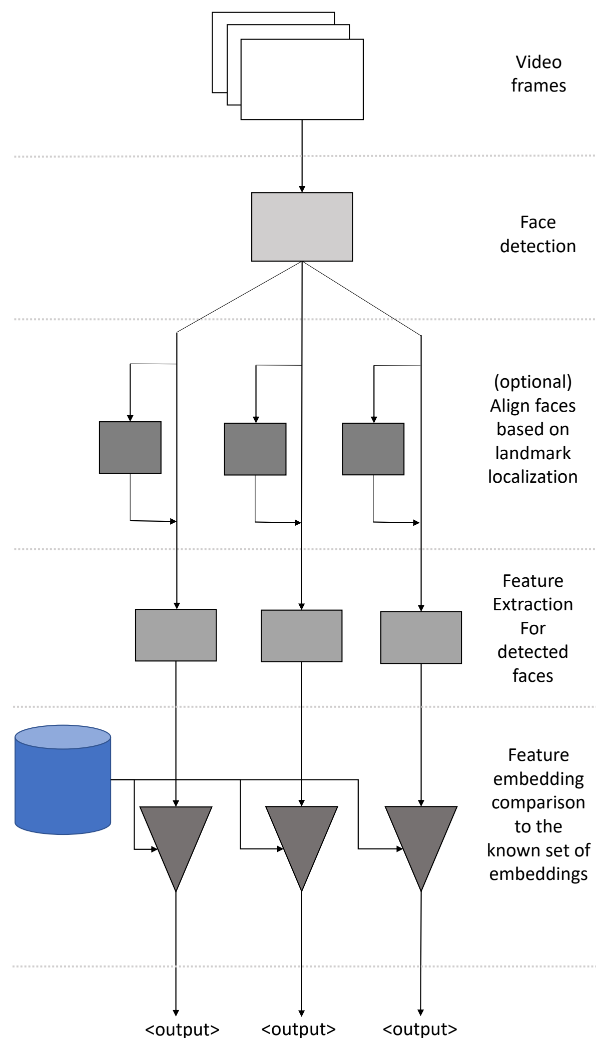

Figure 3.1. *Pipeline of the system*

As shown in the figure, classification is made based on set of embeddings that are extracted from facial images. The extraction of these embeddings is made using the detection and feature extraction steps of the pipeline. Each face that is detected from the input frames, is processed through feature extraction and feature embedding comparison step individually. Each face in the facial images obtains a unique identifier which is later used in classification of new samples, which in this case are the extracted embeddings of the faces that are detected from the frames of the video stream.

3.1 Face detection

Face detection is a similar problem to any other object detection problem. In general, object detection system is capable of detecting multiple object classes. In this case, a face is the only object class that is detected. The goal in face detection is to find and locate one or more faces from images. In this thesis, the desired output of the face detector is illustrated in Figure 3.2.

Figure 3.2. *Desired face detector output (image taken from VGGFace2 dataset [2]).*

A bounding box can be drawn around each face based on the coordinates that are obtained from the output of the face detection algorithm. Based on the coordinates, facial images are cut from the image and passed to the next step of the pipeline.

As mentioned in the introduction (Chapter 1), face detection is widely researched problem. For this thesis, only few approaches are covered. First, Viola-Jones object detection framework [3] based face detector, which has been referred also in the surveys, is tested. After that, two neural network based approaches, YOLO (You Only Look Once) [30] and

SSD (Single Shot MultiBox Detector) [4] are went through. The third widely used neural network based approach, *Region Proposal Networks* is left out, because their performance in terms of speed do not meet the requirements for the real time system with hardware described in Section 4.2 [31].

3.1.1 Viola-Jones object detector

Viola-Jones object detection framework is a three-step procedure to detect objects from the images. It was published in 2001 and it was originally developed for face detection. The first step in the procedure include a generation of *integral image*, which is formed as a sum of pixel values on top and left side of specific pixel as shown in Figure 3.3. In addition, the computation of an integral image is visualized in Figure 3.4. [3]

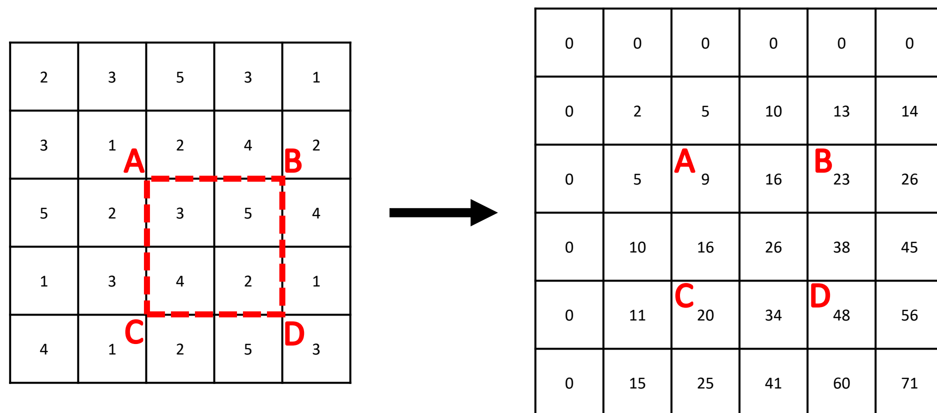

Figure 3.3. *Computation of integral image.*

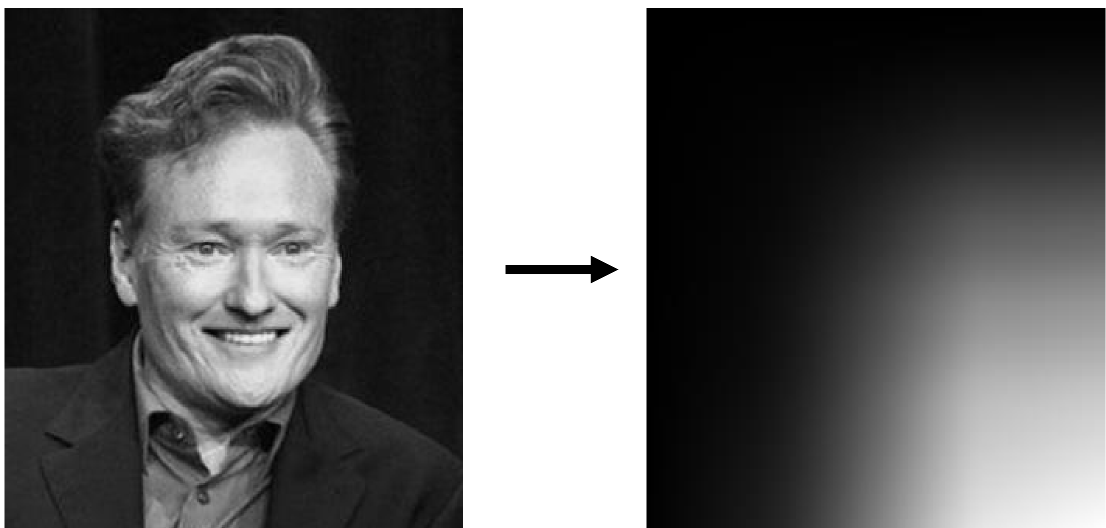

Figure 3.4. *Visualization of integral image (image taken from VGGFace2 dataset [2]).*

Based on integral image, *Haar features*, which describe the difference of pixel values on specific areas as shown in in Figure 3.5, can be computed efficiently based on pixel area sums that lay under specified rectangles. Sum of the pixel values on white areas are subtracted from the sum of values on black areas to compute single feature within the rectangle inside detection window. For example, a sum of pixel values within the red square in the Figure 3.3, can be calculated as $D + A - (C + B)$ which is $48 + 9 - (20 + 23) = 14$. [3]

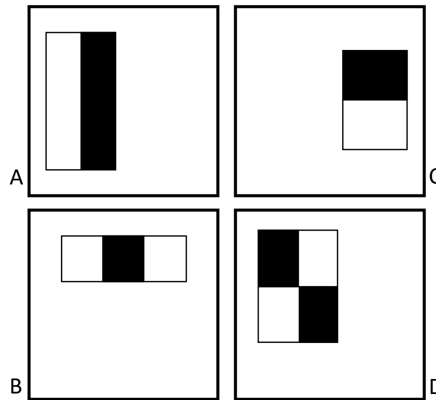

Figure 3.5. *Example of different Haar features (A, B, C and D) within the relative detection window (redrawn from [3]).*

Cascaded structure of AdaBoost classifiers is used for classification based on extracted Haar features. Each of the AdaBoost classifier consist of weighted weak classifiers that threshold single extracted features. In cascade on classifiers, each classifier classifies window to either include the object or not. If no object is detected, the sample is rejected immediately without running it through the rest of cascaded structure of classifiers, because only very few of the extracted features are significant. This way, classifiers in the cascade can detect object starting from abstract features and focus on interesting areas by continuing more detailed features on those areas, which is very efficient in rejecting false positives. [3]

3.1.2 Single Shot MultiBox Detector

The idea of SSD framework is to locate and classify the objects in the image with single forward pass to the neural network. For the image that is fed to the network, the output consists of bounding boxes around detected objects and a class for each of the detected objects respectively. The SSD framework consists of a base network, which output is truncated, and additional feature layers that replace the output layers of the base network. SSD architecture is presented in Figure 3.6.

The feature layers are used to detect objects at multiple scales by producing a discrete set of bounding boxes on each location of the feature map and a probability for different object classes occurrence within the box. Feature maps, which are illustrated in Figure 3.7 are formed as a $N \times N$ grid on the input image, where different scales output varying number

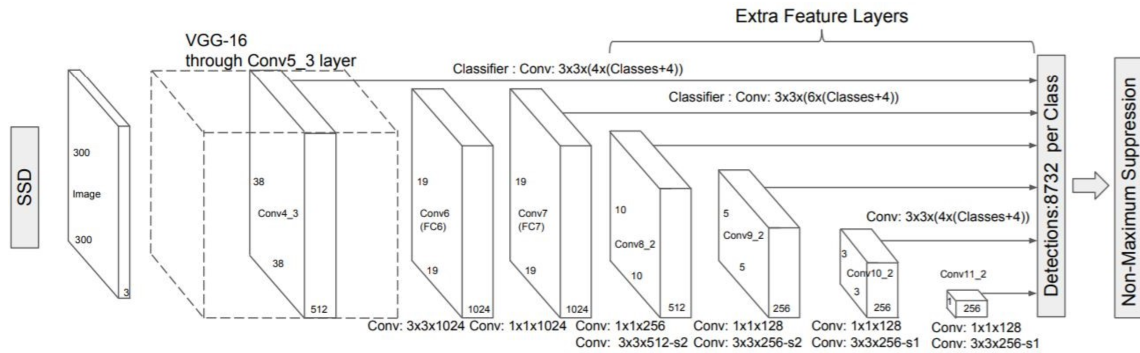

Figure 3.6. *Architecture of the SSD network (modified from [4]).*

of cells. In the architecture presented in Figure 3.6, the grid size varies from 38×38 to 1×1 in six different sizes and four or six bounding boxes per cell. For Figure 3.7, number of boxes per cell is limited to four to improve readability. Boxes are also adjusted to object shape within the box. Based on the bounding boxes, non-maximum suppression (NMS) [32] is used to output final detections. NMS combines overlapping bounding boxes of same object class into one bounding box. [4]

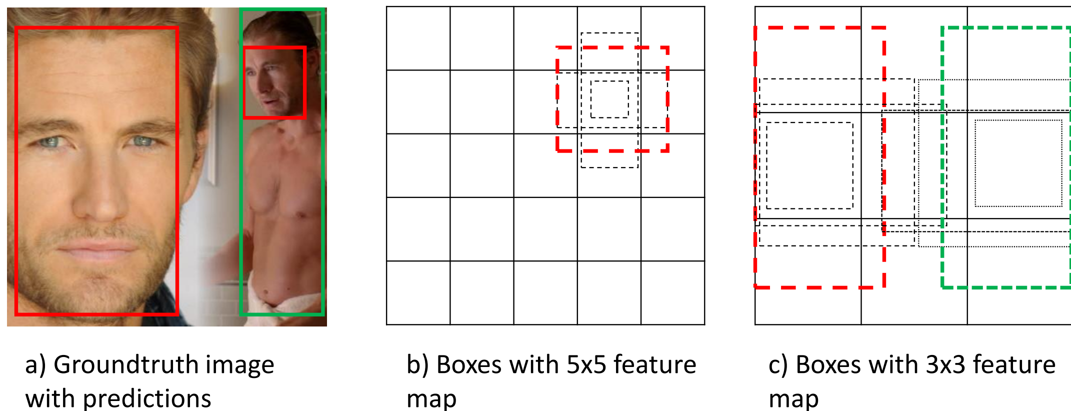

Figure 3.7. *An illustration of SSD feature maps. a) groundtruth image with predicted boxes for two classes, person (green) and face (red), b) four bounding boxes per cell with* 5×5 *feature map, where only one of four boxes (marked with red) detects the face cell, c) four bounding boxes per cell with* 3×3 *feature map, where one of four boxes (marked with green) detects the person in cell and one of four boxes in another cell detects a face (marked with red) (groundtruth image taken from VGGFace2 dataset [2]).*

The base network can be chosen based on the requirements of the application. For example, VGG16 network is too heavy to be used on the hardware (section 4.2) that is used in this thesis. Therefore, a relatively light network such as MobileNetV1 or V2 can be used although the maximum performance might weaken slightly.

3.1.3 "YOLO: You Only Look Once" object detector

Similarly to SSD (presented in Section 3.1.2), YOLO is capable of locating and classifying objects in multiple classes on single forward pass. In addition, the base architecture constructs of blocks of convolutional layers that are responsible for feature extraction, similarly to base networks in SSD. However, the structure of the additional layers that do the object localization and classification, in YOLO, is different compared to the SSD.

The latest versions of YOLO, named as YOLOv3 [30], uses anchor boxes in prediction similarly to SSD. Instead of fixing the anchors by hand, k-means clustering is used for finding the most suitable priors (boxes) based on training data annotations. To stabilize the training, the coordinates relative to grid cells for the predicted boxes are used as an output instead of image coordinates. While the SSD framework scales the grid in order to find different size objects, YOLOv3 increase the resolution of the features by running it through a layer that increases number of channels where the higher resolution features are.

3.1.4 Detector evaluation

For face detection, the similarity of the correctly predicted bounding boxes is evaluated. In order to determine whether the accuracy of the prediction, *Intersection over Union* (IoU) [33] metric is used. IoU is defined as

$$\mathrm{IoU} = \frac{A \cap B}{A \cup B}, \tag{3.1}$$

where A is the groundtruth and B predicted bounding box. IoU is visualized in Figure 3.8.

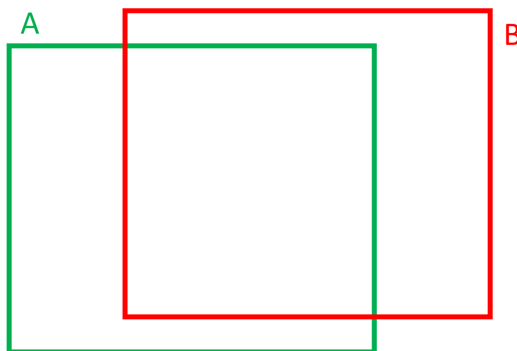

Figure 3.8. *Intersection over union (IoU). The closer the predicted bounding box B is to groundtruth A, the higher the IoU is.*

Based on chosen IoU threshold, the detection can be considered as a positive or negative. With specific IoU threshold, number of correct detections can be compared to the number of groundtruth bounding boxes in the image. Mean average precision (mAP) is defined as an average of correctly detected objects per class over the test dataset. The positive prediction has an overlapping bounding box with the groundtruth bounding box with IoU,

which is the same or higher than the defined threshold. On the other hand, the prediction is negative, if there is no prediction for the groundtruth bounding bounding box with specified IoU threshold.

In COCO evaluation metrics [34], thresholds from 0.5 to 0.95 with interval 0.05 for IoU are used and the final mAP is computed as an average over the possible thresholds. COCO evaluation metrics rewards for well located bounding boxes since high IoU thresholds are used. On the other hand, a threshold of 0.5 can be used for estimating how well the objects are found in general without penalizing the small margins to groundtruth coordinates.

3.2 Facial image preprocessing

Facial image alignment step can be added between the face detection and feature extraction. Adding this step might improve classification results, since the pose is standardized in the images. However, feature extraction can also be performed with unaligned facial images, where the entire alignment step is skipped. In general, the alignment step increases the computational complexity of the system by adding an extra machine learning model and an image transformation to the pipeline.

Different preprocessing methods are used before feature extraction in different existing deep learning based extraction methods. For example, DeepFace [7] performs 3D alignment of the face in order to transform facial images as similar as possible before extraction. In addition, FaceNet [8] performs translation and scale transformations, MobileFaceNets [35] a similarity transformation and OpenFace project [36] an affine transformation, that are more simple 2D transformations explained in Section 3.2.3. The transformations require a specific number of detected facial landmarks, such as eyes, from the facial images to create point correspondences to the fixed groundtruth locations, for forming the mapping between the current and desired locations in the images [37, p. 37–44].

In this thesis, five facial landmarks are detected. They are left and right eye, nose tip and left and right of the mouth. First, the landmarks are detected from face, and then the facial image is aligned so that the landmark locations match to the fixed locations. This process is shown in Figure 3.9.

3.2.1 Landmark detection

For facial landmark detection, the wanted output of the machine learning model is a vector of coordinates, where specific facial landmarks are located in the facial image. Two computationally light solutions, an ensemble of regression trees [38], and *Multi-task Cascaded Convolutional Networks* (MTCNN) [39] are introduced for landmark localization. For training the model, both aforementioned methods utilize an euclidean loss defined in Equation 2.6, where the squared sum of coordinate differences between the predicted and groundtruth landmark locations is computed.

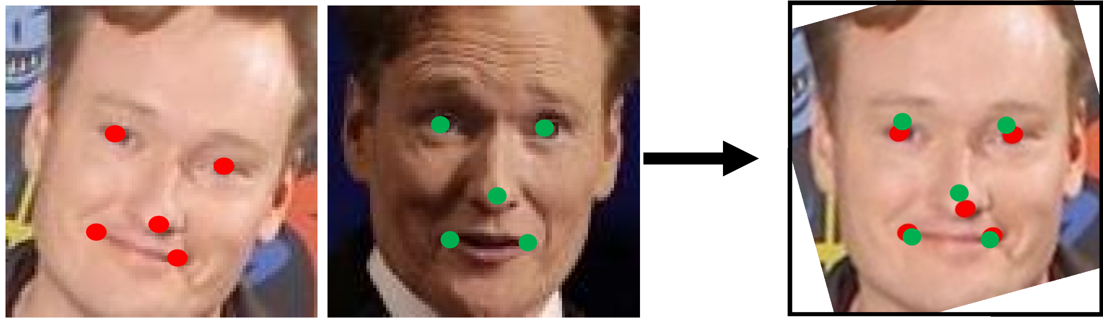

Figure 3.9. *Face alignment process. Detect the five landmarks and align image so that the landmark can be found from same coordinates in the image. Locations marked with green point are considered as groundtruth locations and red are the detected landmarks from the image that is aligned. (images taken from VGGFace2 dataset [2])*

Ensemble of regression trees by Kazemi and Sullivan in 2014 [38], predicts landmarks with a cascade of regression functions. Each regressor in the cascade updates the prediction by adding an update vector to predictions from the previous cascade based on output of the previous cascade and the input image. For the first regressor in the cascade, an initial shape estimate defines the basis for the update. Regression functions are learnt with gradient tree boosting. For each regressor, the decisions are made in regression tree based on difference between two pixel values. Before the difference computations at each cascade level, the image is warped using the current estimation of the landmarks in order to compare pixel values from the same location in the image relative to facial landmarks.

MTCNN is a model for both predicting the bounding box location similarly to face detection and facial landmarks within the predicted bounding box. It is a cascade of three CNNs where the first CNN, named *PNet*, is responsible for obtaining multiple candidates for detected face. The second CNN, *RNet*, removes most of the candidate bounding boxes that are false and outputs the detected faces. The final CNN, *ONet* is responsible for finding the final bounding boxes for the existing faces and also locating the facial landmarks. The architectures of the cascaded CNNs are presented in Figure 3.10. [39]

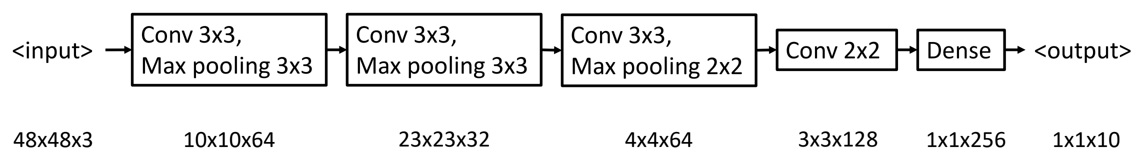

Figure 3.10. *Architecture of the ONet in MTCNN cascaded CNNs with the sizes of the outputs for each of the blocks.*

In face detection, the idea of the MTCNN is similar to the Viola-Jones detector (Section

3.1.1), where the interesting areas are located first and the detection confidence is increased by rejecting false positives from those areas. This way, it is possible to simplify the structure of the CNNs since single network has simple task to be solved.

3.2.2 Landmark detection evaluation

Landmark localization evaluation for the facial landmarks presented in Figure 3.9 uses a measure, where the distances between the groundtruth and estimated landmark locations are computed separately for each of the landmarks. For each of the detected landmarks, the error is computed as

$$e_{landmark} = \frac{1}{N}\sum_{i=1}^{N}\frac{\sqrt{(y'_i - y_i)^2 + (x'_i - x_i)^2}}{l}, \tag{3.2}$$

where l is the scale for the distance, N number of images in evaluation dataset, (y', x') are the groundtruth coordinates for the landmark, and (y, x) define the predicted location of the landmark. The overall error for the landmark detection is computed as an average of errors for each of the landmarks.

For scale l, the inter-ocular distance or the shape (for example width or height) of the image can be used in normalizing the results. For some poses, the distance between the eyes can be impossible to measure (for example, both eyes are not visible). In these cases, image dimensions output more reliable results. By scaling the results, the size of the image does not affect the results. For inter-ocular distance, also the scaling of the face in the image is normalized.

3.2.3 Image transformations

A group of projective transformations consists of subgroups that are specializations of the projective transformation. These subgroups include for example translation and affine transformations. The subgroups form a hierarchical structure, where for example translation is one element in an affine transformation. The first class in the hierarchy, *isometries*, include translation and rotation transformations. For the second class, *similarity transformation*, also the scale transformation is added. The third class, *affine transformation*, adds shearing to the transformation. In shearing, the lines that are parallel preserves, but the perpendicular lines are not perpendicular after the transformation. The transformations included in first three subgroups of projective transform group, are shown in Figure 3.11. [37, p. 37–44]

For transformations, *degrees of freedom* (DoF) define the number of parameters in the transformation. Based on the parameters, the number of point correspondences that are needed for implementing the transformation, can be defined. For isometries, a

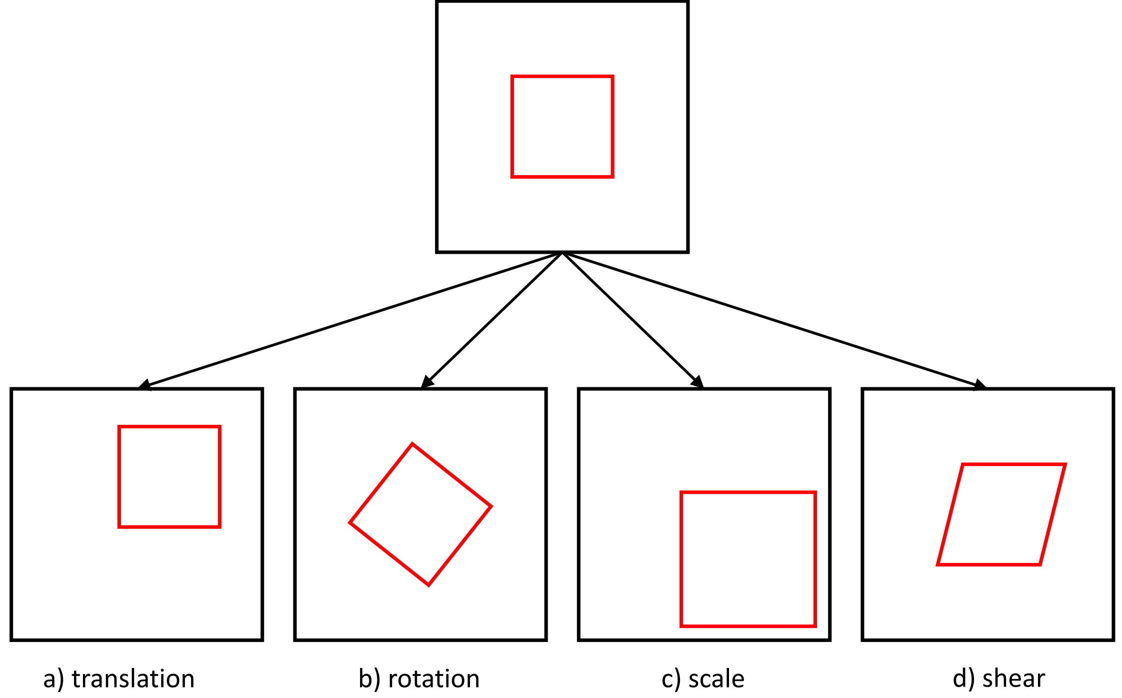

Figure 3.11. *Components of isometries, similarity transforms and affine transforms in group of perspective transforms: a) translation, b) rotation, c) scale and d) shear. The anchor point is set to upper left corner.*

transformation from point (x, y) to (x', y') is defined in homogeneous coordinates as

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \beta \cos\phi & -\sin\phi & t_x \\ \beta \sin\phi & \cos\phi & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3.3}$$

where t_x and t_y define translation, θ is the rotation angle and $\beta = \pm 1$, which defines the orientation. With $\beta = 1$, orientation preserves, and $\beta = -1$ reverses it. Isometry requires three parameters (3 DoF): components of translation (t_x, t_y) and the rotation ϕ that can be solved with two point correspondences. For similarity transform, defined with

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} s \cos\phi & -s \sin\phi & t_x \\ s \sin\phi & s \cos\phi & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3.4}$$

where s is the scale factor, the parameter for the scale is added. Still, the transformation can be solved with two corresponding points. For affine transformation, the formula for the transformation is

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3.5}$$

where transformation matrix $\mathbf{A}$ defined as

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \tag{3.6}$$

includes the parameters for affine transformation. Shear adds two parameters to the transformation matrix compared to similarity transformation (6 DoF in total), and three point correspondences are required for solving the transformation. [37, p. 37–44]

3.3 Face recognition

In feature extraction, a cropped facial image, which might also be pre-processed in the previous step of the pipeline, is processed to an embedding which is an n-dimensional vector of features, that represents the face in cropped facial image. The embedding should be as similar as possible for each unique face (for each person) regardless the factors related to for example pose and environment that is captured in the image.

For this thesis, the problem is limited to only optimizing the loss function for training. Two different losses, *Triplet Loss* [8] and *Additive Angular Margin Loss* (ArcFace) [40] are presented for describing the optimization problem related to face recognition, where the similarity of the extracted embeddings are optimized.

3.3.1 Triplet loss

The triplet loss, which was presented by Schroff*et al.* in FaceNet paper [8], optimizes the CNN based on triplets where the distances within the samples of specific class are minimized (positive match) and the difference between the different classes (negative match) are maximized. It is formed based on the difference of euclidean losses (Equation 2.6) of the positive ($\mathbf{x}^p$) and negative ($\mathbf{x}^n$) match to the anchor ($\mathbf{x}^a$) which can be expressed based on Equation 2.6 for triplet i as

$$||f(\mathbf{x}_i^a) - f(\mathbf{x}_i^p)||_2^2 + \alpha < ||f(\mathbf{x}_i^a) - f(\mathbf{x}_i^n)||_2^2, \tag{3.7}$$

where α defines the margin between the positive and negative samples in optimization. Based, on this expression, the triplet loss for N triplets can be defined similarly to Section 2.3.2 as

$$L_{triplet} = \sum_i^N \max([||f(\mathbf{x}_i^a) - f(\mathbf{x}_i^p)||_2^2 - ||f(\mathbf{x}_i^a) - f(\mathbf{x}_i^n)||_2^2 + \alpha], 0), \tag{3.8}$$

where CNN is modelled with function f. By utilizing three samples to form the triplet, both negative and positive match, and the anchor, where the matches are compared, the embedding of the face can directly be optimized to separate identities.

While training the model, choosing the triplets from the training data, is a key element. The FaceNet finds the optimal triplets from minibatches with *semi-hard negative mining*. It leaves the most difficult negatives (distance to the anchor is smaller than with some of the positives) out from the training in order to stabilize it. Figure 3.12 visualizes the problem related to the triplet selection. [8]

The structure of the FaceNet model while training consists of CNN that is responsible for feature extraction, L_2 normalization of the output and triplet loss. The architecture of

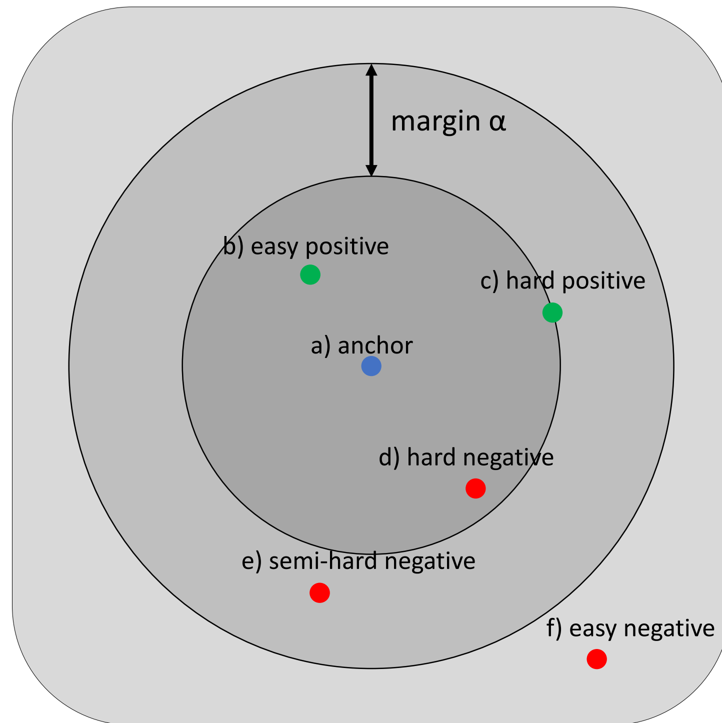

Figure 3.12. *Visualization of triplet selection problem by locating positive and negative samples of the triplet based on distance to the anchor (a): b) easy positive is close to the anchor (a), c) hard positive is further away from the anchor (a), d) hard negative is closer to anchor (a) than hard positive (c), e) semi-hard negative is little further from anchor (a) compared to hard positive (c), f) easy negative is far away from the anchor.*

the CNN and the dimensionality (length) of the output feature embedding can be chosen based on requirements of the application. They affect for example in the computational complexity of the model. [8]

3.3.2 Additive Angular Margin Loss

While the triplet loss aims to minimize the distance within a class so that the hardest positive is closer than the hard negatives from the anchor, *additive angular margin* (ArcFace) loss minimizes samples' distances to their class means and maximizes the distances between the means of the different classes. In Arcface loss, these distances are interpreted as angles θ_j, where j represents sample, between the feature embedding $\mathbf{x}_i$, where i refers to the sample index, and $\mathbf{W}_j$ which can be considered as a mean of the class for sample j. [40]

By combining the softmax activation function (Equation 2.5), that maps the embedding vector to class probabilities, and the categorical cross-entropy loss (Equation 2.7) that penalizes incorrect classifications, *softmax loss* is defined as

$$L_{softmax} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{\exp(\mathbf{W}_{y_i}^T \mathbf{x}_i)}{\sum_{j=1}^{n} \exp(\mathbf{W}_j^T \mathbf{x}_i)}, \tag{3.9}$$

where index y_i refers to weights corresponding $\mathbf{x}_i$. While the logit $\mathbf{W}_j^T\mathbf{x}_i$ is transformed to the product of parameter lengths and the angle as $\|\mathbf{W}_j^T\|\|\mathbf{x}_i\|\cos\theta_j$ and, in addition, by normalizing the weights $\mathbf{W}_j$ individually as $\|\mathbf{W}_j^T\| = 1$, and feature vector as $\|\mathbf{x}_i\| = 1$, applying additional scale s and adding the angular margin penalty m to the angle θ_j, ArcFace loss can be written based on Equation 3.9 as, [40],

$$L_{arcface} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp(s\cos(\theta_{y_i}+m))}{\exp(s\cos(\theta_{y_i}+m))+\sum_{j=1,j\neq y_i}^{n}\exp(s\cos\theta_j)}. \tag{3.10}$$

Similarly to the triplet loss, the architecture of the feature extraction CNN and the dimensionality of the output can be chosen based on the application. For example, *MobileFaceNet* has utilized successfully ArcFace loss in applications that require lower computational complexity [35]. The *MobileFaceNet* architecture is based on MobileNetV2 blocks followed by layers that are responsible for mapping the extracted features to output of the CNN [35].

3.3.3 Feature extraction evaluation

In binary classification case, the sample is classified as true or false. Each classification is either positive or negative based on the groundtruth of the sample. Based on this information, a *confusion matrix* can be drawn that covers all the possible combinations. It is presented in Figure 3.13. [41]

Predicted class \ True class	P	N
T	True Positives (TP)	False Positives (FP)
F	False Negatives (FN)	True Negatives (TN)

Figure 3.13. *Confusion matrix in binary classification.*

Based on the cells in Figure 3.13, accuracy metrics can be defined. *True positive rate* (TPR) evaluates, how many of the positive samples are classified to "true" as

$$TPR = \frac{TP}{P}, \tag{3.11}$$

where TP are positive samples classified as "true" and P are positive samples. *False positive rate* (FPR) measures how many of the samples that are negative are classified to "true" as

$$FPR = \frac{FP}{N}, \tag{3.12}$$

where N are negative samples and FP are negative samples classified as "true. In addition to true and false positive rates, accuracy for binary case can be defined as

$$accuracy = \frac{TP + TN}{P + N}, \tag{3.13}$$

where TN are negative samples classified as "false". All the accuracy parameters are visualized in Figure 3.13. [41]

Based on TPR and FPR, *receiver operating characteristics* (ROC) curve can be drawn by changing the threshold for classification (which similarity defines whether the sample is classified as "true" or "false"). Different thresholds define the classifier sensitivity to the false positives (how many classified as "true" are negative). *Area under curve* (AUC) metric measures the area under ROC curve. The closer the AUC is from the maximum value 1, the better the classifier is in separating the positive and negative samples from each other. [41]

The threshold can be adjusted to define, how sensitive the similarity search is to false positives. The threshold defines whether the nearest neighbour from the set of groundtruth embeddings is close enough from the embedding that is classified. Generally, the lower the threshold is, the less sensitive the classification is to false positives, but simultaneously the risk of false negatives (negative match classified as "true") increases.

For evaluating the feature extraction, *labeled faces in the wild* (LFW) evaluation protocol [42] is used in forming a binary classification problem and benchmarking feature extractors. LFW dataset splits [43] divide dataset to ten folds where images are paired to positive and negative pairs. Positive pairs include two images with same identity and negative pairs have different identities in images.

3.4 Nearest neighbour search

In classification of the embeddings extracted in the previous step of the pipeline, nearest neighbour search refers to finding the most similar embedding (vector) from the set of embeddings. The most similar embedding defines the class for the sample. By comparing the number of correct nearest neighbour searches to the total number of searches, the top-1 accuracy is obtained as $\frac{correctly\ classified}{number\ of\ samples\ in\ search\ space}$. In addition, top-5 and top-10

accuracy can be used to find out, whether the correct match is within the five or ten nearest neighbours respectively.

Finding the nearest neighbour from the random set of embeddings is a linear operation (sample is compared to each embedding in the set and each comparison is a constant time operation). While the size of the set of embeddings increase, the search space can be reduced to subset of the original set. Clustering methods are capable of dividing the embeddings to clusters (subsets), where each subset contains embeddings that are similar to each other. In this case, the search estimates the nearest neighbour by searching it from the subspace rather than from full search space.

Different types of clustering methods can be used, two of them are introduced in this thesis. Partitioning methods divide the samples to desired number of clusters that are as far from each other as possible. The other type of methods introduced in this thesis, hierarchical clustering, forms a hierarchy between the clusters which reminds a tree structure. [44, p. 448–449]

For high dimensional data, the clustering becomes challenging. One problem is that the noise may dominate in distance measures between the samples since all the dimensions may not be significant in clusters. This could be solved for example searching clusters in subspace of the original dataset (feature selection) or reducing the dimensionality of the data. However, these approaches lead to the loss of data that is considered to be non-meaningful. [44, p. 508–512]

3.4.1 Vector similarity

The embeddings can be modelled as arrays with fixed dimensionality that represent the features that are extracted from the face. In this case, these arrays $\mathbf{x}_i$ and $\mathbf{x}_j$ that are the two samples to be compared, simple vector similarity metrics are used. In order to limit the distances between the samples, each sample $\mathbf{x}_n$ is normalized before the comparison ($||\mathbf{x}_n|| = 1$).

The angle between the samples $\mathbf{x}_i$ and $\mathbf{x}_j$ can be used for representing the similarity between the samples. For that purpose, *cosine similarity*

$$d_{cosine}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{||\mathbf{x}_i||\,||\mathbf{x}_j||}, \tag{3.14}$$

can be used.

For normalized samples, the denominator of the equation equals to 1. The possible values of d_{cosine} are on interval $[-1, 1]$ similarly to cosine of the angle. The value 1 corresponds to 0° angle, 0 to 90° angle and -1 corresponds 180° angle. The smaller the angle is, the closer the samples are from each other based on cosine similarity metric. [44, p. 77–78]

For this thesis, cosine similarity metric is modified so that value 0 corresponds to 0° angle and value 2 corresponds 180° angle (based on Equation 3.14 as $|d_{cosine} - 1|$). This scaling is similar to other distance metrics, which helps in implementing different distance metrics.

3.4.2 *k*-means clustering

k-means clustering is a partition clustering method that groups the data to the clusters, where each sample $\mathbf{x}_j$ in data $\mathbf{X}$ belongs to cluster C_i, which central point $\mathbf{c}_i$ is closest to the sample. For example the cosine distance defined in Equation 3.14 can be used to determine the closest distance. In addition, the number of the clusters k can be defined based on the data that is fitted to the clusters.

Data is fitted with k-means clustering by first defining the initial cluster centroids and then iterating them. First, each sample in the data is located to the cluster which centroid is closest to the sample. Then, new cluster centroids are computed to be the mean of the samples in the cluster. The iteration is continued until the stopping condition, where the location of the cluster centroids do not change between the iterations, is met. [44, p. 452–453]

3.4.3 Hierarchical clustering

For utilizing hierarchical clustering for high dimensional data, the most significant components of the data are proven to work in forming the hierarchical model. This way, while the root node describes all the data that is clustered, the following level of hierarchy is determined based on the most significant components, and this hierarchy is continued to the less significant components in descending order until the stopping condition of the clustering algorithm is met. [45]

Principal component analysis (PCA) can be used in finding the most significant components. PCA is a technique that extracts the important features from the data and represents that information in principal components that are a set of orthogonal vectors. [46]

First, the PCA computes the covariance matrix $\Sigma_{\mathbf{X}}$ for data $\mathbf{X}$, where the rows represent samples and columns represent features. For covariance matrix $\Sigma_{\mathbf{X}}$, a sample covariance for the columns i and j of the the data X as $\mathbf{y}_i$ and $\mathbf{y}_j$ respectively, is formed as

$$\sigma_{i,j} = \mathrm{cov}(\mathbf{y}_i, \mathbf{y}_j) = \frac{(\mathbf{y}_i - \mu_i \mathbf{l}_n) \cdot (\mathbf{y}_j - \mu_j \mathbf{l}_n)}{n-1}, \tag{3.15}$$

where n is the number of rows in X and $\mathbf{l}_n$ is a vector of ones with length n. Based on Equation 3.15, the covariance matrix can be defined as

$$\Sigma_{\mathbf{X}} = \mathrm{cov}(\mathbf{X}) = \frac{1}{n-1} \mathbf{X}_C^T \mathbf{X}_C, \tag{3.16}$$

where centered data $\mathbf{X}_C$ is formed similarly to 3.15 by subtracting the means $\boldsymbol{\mu}$ of each column of $\mathbf{X}$ as $\mathbf{X}_C = \mathbf{X} - \mathbf{1}_n \boldsymbol{\mu}^T$. For PCA, the *eigenvalues* λ are solved so that there exists an *eigenvector* $\mathbf{v}$ that fulfils the equation

$$\Sigma_{\mathbf{X}} \mathbf{v} = \lambda \mathbf{v}, \tag{3.17}$$

for each eigenvalue λ. Based on Equation 3.17, the matrix $\Sigma_{\mathbf{X}}$ can be decomposed to

$$\Sigma_{\mathbf{X}} = \mathbf{V} \boldsymbol{\Lambda} \mathbf{V}^{-1}, \tag{3.18}$$

where $\boldsymbol{\Lambda}$ holds the eigenvalues λ in diagonal elements and *eigenvector matrix* $\mathbf{V}$ includes eigenvectors that are *mutually orthogonal*. [45]

The eigenvector matrix $\mathbf{V}$ is utilized in hierarchical correlation clustering algorithm CHUNX [45]. It computes the angles $\pm\gamma_i$ between the samples in data $\mathbf{X}$ and the eigenvectors $\mathbf{v}_i$ in $\mathbf{V}$, where the sign in $\pm\gamma_i$ refers to the angles between the positive and negative directions of the eigenvectors. In order to compute the eigenvectors, there must be at least as many embeddings in the set of groundtruth embeddings as the dimensionality of the embedding is. The angles to the principal components in $\mathbf{V}$ are computed with Equation 3.14. The most significant component for sample $\mathbf{x}$ is defined by the smallest angle between it and the eigenvectors $\mathbf{v}_i$ in $\mathbf{V}$.

Based on the smallest angle with specific eigenvector, the data is divided to the clusters that are children of the node in tree structure. For each of the cluster, the next most significant component recursively splits clusters until the defined stopping condition, where a hyperparameter is used in this case, is met. For CHUNX, the hyperparameter is the maximum cluster size. The recursion is continued as long as the cluster size is smaller than the set limit. An example of clustering hierarchy is shown in Figure 3.14. While the formation of the hierarchical structure in CHUNX is respect to the cluster size, clusters can appear only in the leafs of the tree. [45]

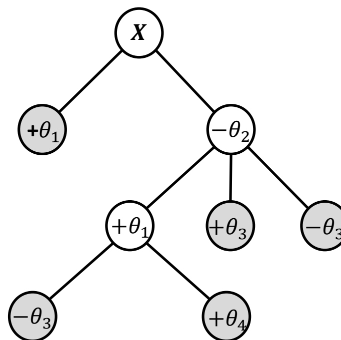

Figure 3.14. *Example of hierarchy of the CHUNX. Nodes including the samples from data* $\boldsymbol{X}$ *have gray background. Values in the nodes refer to the* i*:th angle related to the corresponding eigenvector which is the most significant component that defines the cluster.*

4. IMPLEMENTATION

The implementation of the face recognition system was made by collecting a solution for each of the steps in the pipeline presented in Figure 3.1 to a complete pipeline based on the experiments. The methods that were used in the implementation are presented in Chapter 3. Multiple separate solutions for each pipeline step were experimented and compared in order to find the best solution for each step.

In order to develop and test solutions for each step, few datasets are required. They are presented in this chapter. In addition, computational complexity of the implemented methods is evaluated by timing the inference times using hardware, which is also introduced. The system and methods were implemented with Python programming language. The implementations of the pipeline steps utilize multiple open-source libraries and machine learning model sources.

4.1 Data

There are numerous datasets available for face detection, facial landmark localization and face recognition. However, most of them are only for research purposes and commercial use is prohibited. Due to restriction of commercial usage, the amount of available datasets was relatively limited. After all, two datasets for face detection and face recognition were chosen. For both of these cases, one dataset is larger and used in training of the machine learning models. In general, larger dataset provides better material for training. The other set in both of the cases was reserved for evaluation purposes only, since it improves the reliability of obtained results. The evaluation datasets were chosen because they are widely used in benchmarking the performance of the models.

For face detection, the datasets are labeled so that for each image the locations of the faces are given as explained in Section 3.1. In these datasets, each face that fill the specification of the face that is defined in dataset report. In this thesis WIDER FACE [47] dataset with 32,203 images that include 393,703 faces with high variance in factors described in introduction (Chapter 1), was used in face detector training. It is divided into three parts, training, validation and testing. The training and validation parts were used. WIDER FACE dataset is very challenging dataset for any machine learning model due to its high variance. It suits well in face detector model training since the model can be considered to work well in most of the environments if the performance with validation part of the dataset is good.

In estimation of face detector performance, the FDDB (Face Detection Data Set and Benchmark) [48] dataset was used. It includes 2845 images with 5171 faces labeled.

Compared to the WIDER FACE, the FDDB dataset has smaller variance, yet it is high enough for obtaining reliable results. After all, the FDDB dataset covers most of the possible scenarios that are faced in this application.

Face recognition datasets include only a label for each image. The label corresponds to one identity in a dataset. There might also be some other people on the background of the images, but the classified identity can always be easily recognized. For training the face recognizer, VGGFace2 dataset [2] was used. In total, the VGGFace2 dataset has 3.31 million images of 9131 persons. The dataset has a high variance for example in ethnicity and profession.

In addition, VGGFace2 dataset include annotations for bounding boxes for face detection and five facial landmarks (eyes, nose, left and right of the mouth) within the box. Based on the bounding boxes, the whole dataset was pre-processed to 96×96 pixel images where the annotated face was cropped to cover most of the image. The faces were cropped with modified bounding box, which was extended from the annotated box. Also facial landmarks were recalculated based on the cropping. For facial landmark localization, only pre-processed VGGFace2 dataset was used. For training, a subset of the dataset training partition was randomly picked. The validation used the test partition of the dataset.

For benchmarking face recognition performance, LFW [49] dataset was utilized. There are images from 5749 individuals in the dataset (13,233 images in total). 1680 identities in the dataset have more than one image. In addition, the search speed among the faces in the dataset was measured. For that purpose, an image from each of the 5749 individuals were picked. LFW dataset was pre-processed similarly to VGGFace2 dataset. Since there are no groundtruth bounding boxes available, developed face detector (Section 4.3) was used to create them first in order to crop the face from the images.

There are 522 overlapping identities between the LFW and VGGFace2 dataset, 244 in identities that has more than 1 image in LFW dataset, but they were not removed. There might also be same images, but it was not tested. In LFW benchmark, 410 overlapping identities appear 4281 identities that are involved in the benchmark. This has an impact on accuracy scores achieved with LFW dataset, but the affection is expected to be small, because of the big amount of images in VGGFace2 dataset, and it does not affect the relative differences between the tested models.

4.2 Hardware

For running performance tests, NVIDIA Jetson TX2 Developer Kit was used. The kit includes NVIDIA Jetson TX2 Module, which is an embedded computer for AI related development. The developer kit includes also an embedded camera, but for this application an external web camera with 1080p resolution was used.

The Jetson TX2 module includes both *central processing unit* (CPU) and GPU which can be used to accelerate computation especially with neural networks. It has ARM Cortex-A57

and NVIDIA Denver2 CPUs, an NVIDIA GPU with 256 CUDA cores and 8 GB of memory, which is shared between the GPU and CPUs. Jetson TX2 Developer kit was chosen for implementation in this thesis since the computational power is sufficient for using neural networks in face recognition pipeline yet the price of the Jetson TX2 module is relatively low.

4.3 Detecting faces from image

OpenCV library provides pre-trained models for Viola-Jones object detector (Section 3.1.1), where one of the models is for detecting faces in front (both eyes, nose and mouth are visible), and sample for a lightweight ResNet CNN that utilize Residual blocks (presented in Section 2.4.4) in its architecture for feature extraction and SSD for object detection, that has been trained with *Caffe* framework [50]. These models are trained to detect faces only and they can be used as is, without any modifications for implementation.

For training object detection models, two separate frameworks, *Tensorflow Object Detection API* [51] and *Darknet* [52], were used. For both networks, pre-trained models are available and they were utilized for transfer learning. The available pre-trained models are trained with COCO dataset, where 80 different object classes are available. However, object class for the face does not include in COCO. While objects in COCO dataset vary, the pre-trained models can be considered as a good base for the most of the different custom object detectors. For both of the frameworks, training the model follows similar pattern. First, a pre-trained model and dataset with annotations were loaded, then training was configured and data was converted to the format that framework accepts, and finally the model was trained with the training script that are provided by the framework. For training, WIDER FACE dataset was used.

For *Tensorflow Object Detection API*, *Tensorflow Detection Model Zoo* includes pre-trained models, that can be used for training SSD detector. MobileNetV1 and V2 were tested because their inference time were expected to be fast enough. The MobileNetV1 and V2 were configured so that the input image size to the detector was set to 300×300 and 320×320 respectively in order to reduce inference time, but still achieve good performance. After all, MobileNetV2 provided slower inference time in tests with very similar accuracy to MobileNetV1. Therefore, MobileNetV2 was left out.

Darknet project provides pre-trained YOLO object detection models. YOLOv3 Tiny model, which is a smaller version of YOLOv3 model, was chosen because of its fast inference time similarly to MobileNetV1. Similarly to trained MobileNet+SSD CNNs, the input size was configured to 320×320 pixels due to inference time. Also bigger input sizes were tested (416×416 and 608×608 pixels), but their accuracy improvement was not significant considering the increased inference time which is more important factor for this application.

While models that has either provided as open-source or trained, use different frameworks,

also the models are saved and stored in different formats and different models require special preprocessing (format changes, resizing, mean subtraction of the pixel values and scaling of the intensities) for the images. In addition, converting the models between the frameworks is not straightforward. OpenCV library provides tools for accessing each of the described detectors. However, the OpenCV library is not capable of accessing the GPU on the hardware in Section 4.2. The CNN based models benefit greatly from the GPU's parallel computation capabilities and effective matrix multiplication. Therefore, *Tensorflow* [53] and *pydarknet* libraries according to the format of the model were used. After all, a common interface was developed, where each model was implemented as a class, which inherits the base class that describes the required functionality for the detector.

For the following steps of the face recognition system pipeline from Figure 3.1, face detection step is responsible for extracting detected faces from the image and cropping them. For cropping, the bounding boxes that are detected by the detector framework are used in cropping. Some examples of the detected bounding boxes with different frameworks are shown in Figure 4.1. For the models which outputs are added to the Figure 4.1, (a) Viola-Jones and (b) ResNet+SSD are models provided open-source by the OpenCV library. Models (c) MobileNetV1+SSD and (d) YOLOv3 Tiny were trained for this thesis.

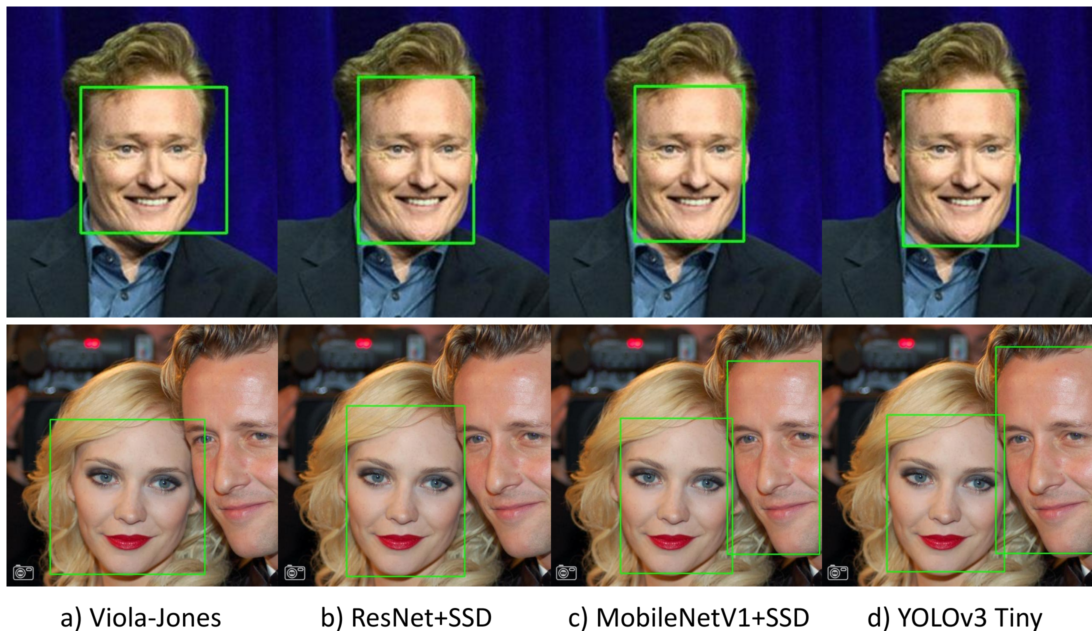

Figure 4.1. *Bounding boxes detected with different frameworks. (images taken from VGGFace2 dataset [2])*

From the Figure 4.1 it can be seen that the bounding boxes differ from each other based on the used framework. In general, CNN based approaches (b-d) output relatively similar results, but their bounding box locations have slight differences. YOLOv3 Tiny detector was capable of finding more challenging faces than the others. Viola-Jones detector did not detect face from profile view, but that kind of images have not most likely been used

in training the model because it is developed for frontal view detection. In addition, the framework that was used in performing the prediction affected to the output. In Figure 4.2, the frameworks that were used in training the CNNs achieved better detection results than OpenCV which provides an interface to run detections with deep learning models that have been trained with different frameworks.

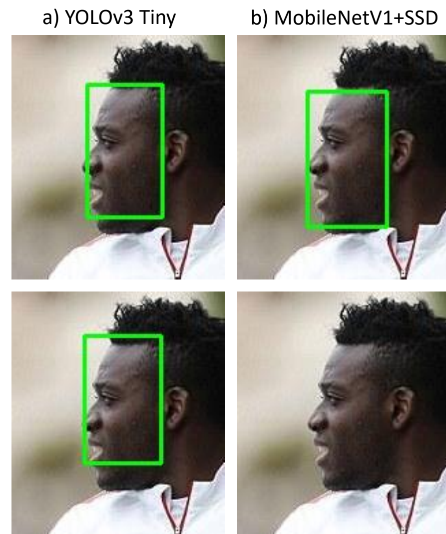

Figure 4.2. *Differences in detection results related to framework. On upper row, Darknet framework with YOLOv3 Tiny CNN and Tensorflow with MobileNetV1+SSD are used. On bottom row, OpenCV library is used with both detectors. (image taken from VGGFace2 dataset [2]).*

For following steps in the pipeline, a unified cropping between the frameworks is required. Now that the cropped facial images are processed with CNNs in the following steps of the pipeline, the cropped facial image was cropped so that the outputting facial image is a square and the face covers most of the facial image without being stretched. While the image size affects to the processing speed, the size of the facial image was fixed to 96×96 pixels. The implemented face cropping is presented in Figure 4.3

Figure 4.3. *Examples of implemented unified face cropping (images taken from VGGFace2 dataset [2]).*

Now that the focus in implementation for this thesis is in following steps of the pipeline presented in Figure 3.1, less effort was used in optimizing the face detector. The main goal

was to implement a detector that is reliable enough from our subjective perspective and is fast enough to fulfil the goal of the thesis. For objective evaluation, metrics presented in Section 3.1.4 were used. COCO [34] provides a tool for evaluating the metrics based on numerical values that define the location of the bounding boxes in the image. In order to use the tool, each image in the evaluation dataset (FDDB and WIDER FACE test partition were used in this case) was run through the detector to get bounding box coordinates, and then the detector was evaluated by comparing detected coordinates to the groundtruth annotations provided with the datasets. In addition, the inference time of the detection process for each image was measured for speed evaluation. Based on the output of the detectors and implemented evaluation, which results are shown in Section 5.1, MobileNetV1+SSD detector was chosen to be used for pre-processing that is required for following steps.

4.4 Aligning facial image

Based on the cropping of the facial images defined in face detection step (Section 4.3), the alignment of the facial images was implemented in three steps. First, models for landmark localization were trained, then, the landmarks were utilized in alignment and finally the facial image was aligned by applying an image transform based on difference of the detected landmarks and fixed locations.

The most of the datasets that include annotations for facial landmark detection, does not allow commercial usage, which was required in this thesis. In addition, pre-trained models that are trained with these datasets, cannot be used in commercial applications. Therefore, a VGGFace2 dataset with annotated landmarks and preprocessing explained in Section 4.1 was used for training the models from scratch. Since there are over 3 million images in training set, only one tenth of the images in the training dataset were randomly picked for training. For testing the model, test partition of the VGGFace2 dataset was used in its entirety. Two different approaches presented in Section 3.2.1 were implemented.

Dlib [54] provides implementation of the *Ensemble of regression trees* method [38]. There is also a script available for training and evaluating own custom landmark detectors. In order to utilize the implementation, training and testing dataset meta information (image paths in file system and annotations) were parsed to specific format. Based on Dlib script implementation, the model was configured based on the hyperparameter settings from method paper [38].

For CNN based landmark detector, the architecture of the MTCNN ONet shown in Figure 3.10 was implemented in Keras [18]. For this ONet detector, two outputs related to bounding box localization and object class probabilities were removed so that the CNN has only one output, which is a vector with length ten (x and y coordinates of five facial landmarks). For training SGD was used as an optimizer with euclidean loss (Equation 2.6). MTCNN ONet is designed for $48 \times 48 \times 3$ input image. Therefore, the training and testing datasets were scaled to this size.

Similarly to face detection, also the aligners require different libraries for detecting the landmarks with different models. Therefore, a common interface for landmark detector was created. MTCNN ONet detector, was converted to *Tensorflow* model, and in implementation Tensorflow was used for accelerating the inference with GPU. For *Ensemble of regression trees* model, Dlib was used in implementation. Four examples of detected facial landmarks utilizing the implementation are shown in Figure 4.4.

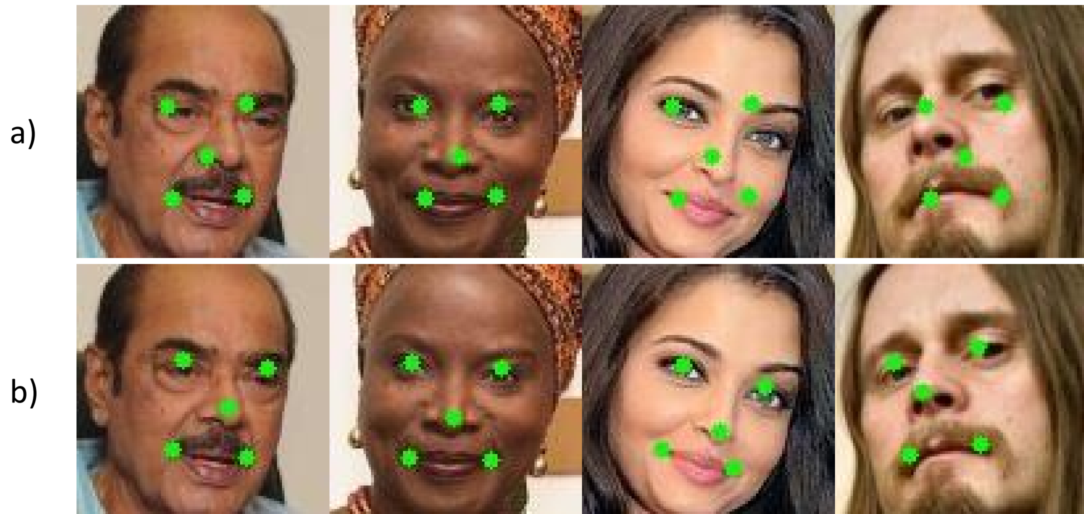

Figure 4.4. *Examples of facial landmark detector output. a)Dlib detector, b) MTCNN ONet detector (images taken from VGGFace2 dataset [2])*

From the Figure 4.4, it can be seen that the MTCNN detector performs better with faces that are aligned straight towards the camera. In addition to pose, also the cropping of the facial image affects significantly to the performance of the output. The affection is visualized in Figure 4.5. For objective evaluation of the landmark detectors, the testing partition of the VGGFace2 was used. The landmarks were detected with both of the detectors for each facial image in the dataset and evaluated with metrics presented in Section 3.2.2. In addition, inference times were computed simultaneously.

The cropping of the facial images also affected to the landmark detector accuracy. In general, the implemented detectors expect relatively similar patters of the facial landmark. If the landmarks differ significantly from the expected, the accuracy weakened. The affection of the the cropping can be seen from Figure 4.5. Because of this limitation, it was essential to use similar cropping in both training dataset and in environment, where the detector is used.

As shown in Section 3.2.3, two or three landmark are required for alignment depending on the alignment. Similarity (Equation 3.4) and affine (Equation 3.5) transforms were chosen for implementations. For both of the transforms were implemented with OpenCV library, that provides functions for performing the transformations.

In order to perform alignments, groundtruth locations of the three landmarks to perform

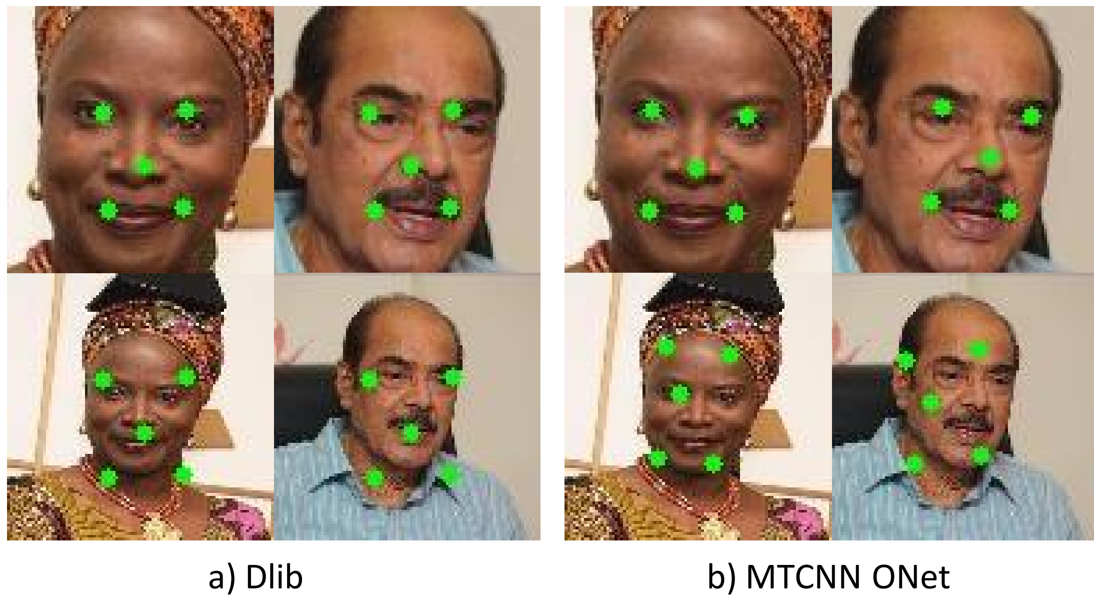

Figure 4.5. *Affect of the cropping in landmark detection with different detectors. (images taken from VGGFace2 dataset [2])*

the transformation were fixed. First, both eyes were located so that the mean point of the eyes was horizontally centered to the image and the distance and vertical location was set so that the whole face is visible in the image.

For similarity transformation, the locations of the eyes were used in alignment. For affine transformation, also the third landmark was needed. While the relative location of the nose tip to the other landmarks is not constant with different poses, as seen from Figure 4.4, the third landmark was chosen to be a mean of the two landmarks of the mouth. The examples of the implemented alignments based on detected landmarks utilizing MTCNN ONet based detector are presented in Figure 4.6.

The main difference between the transformations in Figure 4.6 is that the affine transformation stretches the face in order to fit it into the model. The effect of the alignment for the feature extraction from the facial images, and eventually to the classification result, cannot be evaluated by examining the differences of the alignment output. The affection of the alignment was estimated by training different feature extraction models for each type of preprocessing of the image.

In addition, fixing the desired locations of the landmarks in the template that the transform follows, was defined. The locations of the landmarks were chosen by experimenting the relative locations in the facial image from small set of images, so that the face would fill the most of the outputting cropping, the majority of the face is visible, and its landmarks relative locations would remain the same compared to the unaligned facial images.

Figure 4.6. *Differences between the alignments. a) unaligned, b) similarity aligned and c) affine aligned. (images taken from VGGFace2 dataset [2])*

4.5 Extracting face embedding

In order to limit the number of different combinations of CNNs and alignments in implementation, one lightweight architecture for facial feature extractor CNN was chosen. MobileFaceNet [35] architecture fulfilled the requirements and had proven to achieve high performance. It was implemented with Tensorflow using 96×96 input image size based on the output size of the face detector defined in Section 4.3. For the following steps, the length (dimensionality) of the outputting embedding was defined. Recent papers such as MobileFaceNet [35] use 512 dimensional output. However, the dimensionality was fixed to 128 in order to limit the size as much as possible to improve inference time of the system. In addition, FaceNet paper [8] has proven that the 128 dimensional embedding is capable of achieving very similar accuracy to current state-of-the-art methods such as ArcFace loss based models [40].

While developing the feature extractor, three different alignments (no alignment, similarity transformation and affine transformation) with two different losses, triplet (Section 3.3.1) and ArcFace (Section 3.3.2) loss were trained and evaluated. Different alignments for pre-processed images were created utilizing methods defined in Section 4.4. For training, the train partition of VGGFace2 was used in its entirety. In order validate training results, LFW dataset was used.

The hyperparameter configuration for learning rate and margin of the loss in training

the models with triplet loss was similar to FaceNet [8]. Also different configuration for the settings were tested, but the FaceNet settings yielded best results in validation. Similar pattern was followed with ArcFace loss. The training was configured based on hyperparameters presented in ArcFace [40] and MobileFaceNet [35] papers. Batch size was however reduced to 128, because of the limited GPU memory size, and number of epochs was increased instead. Again, different values were tested, but the resulting models were weaker respect to the accuracy with LFW dataset validation.

To decide either the embeddings represent same identity or not, a threshold for the distance metric was defined. Optimal thresholds for classification were searched for cosine similarity metric defined in Section 3.4.1, to maximize the accuracy. Different distance metrics such as euclidean distance did not affect the results. In order to find the optimal threshold, LFW 10-fold cross-validation [43] was used for evaluation so that each of the ten folds of positive and negative pairs were tested with different thresholds individually. The averages of optimal thresholds for cosine similarity metric for the folds and the corresponding accuracy are presented in Table 4.1 for both of the tested loss functions.

Table 4.1. *Optimal thresholds and corresponding accuracies with different alignments utilizing triplet and ArcFace losses in training.*

	Triplet loss		ArcFace loss	
alignment	accuracy	threshold	accuracy	threshold
none	0.895	0.32	0.978	0.61
similarity	0.875	0.30	0.961	0.26
affine	0.892	0.36	**0.980**	0.64

On hardware described in Section 4.2, the feature extraction from the facial utilizing implemented CNN image took 0.119 s on CPU and 0.023 s on GPU. In general, training with ArcFace loss results to higher performance compared to training with triplet loss. In addition, applying different alignments as presented in Section 4.4, does not improve accuracy of the classification utilizing LFW evaluation metrics in estimation. ROC curves based on average TPR and FPR of the folds with LFW evaluation with both losses and alignments are visualized in Figure 4.7.

4.6 Classifying embedding

Based on the experiments in Section 4.5, networks trained without alignment and with affine alignment utilizing ArcFace loss were chosen to further evaluation, because they outperformed other combinations of alignment and loss function.

In order to develop different classification methods effectively, a base class for all the classifiers was developed. The classifier has three different tasks in this application. First, groundtruth embeddings with labels can be either added or removed from the classifier. Secondly, if the classifier requires fitting while changes to the groundtruth embeddings occur, the classifier must be fitted. Finally, the samples are classified by comparing them

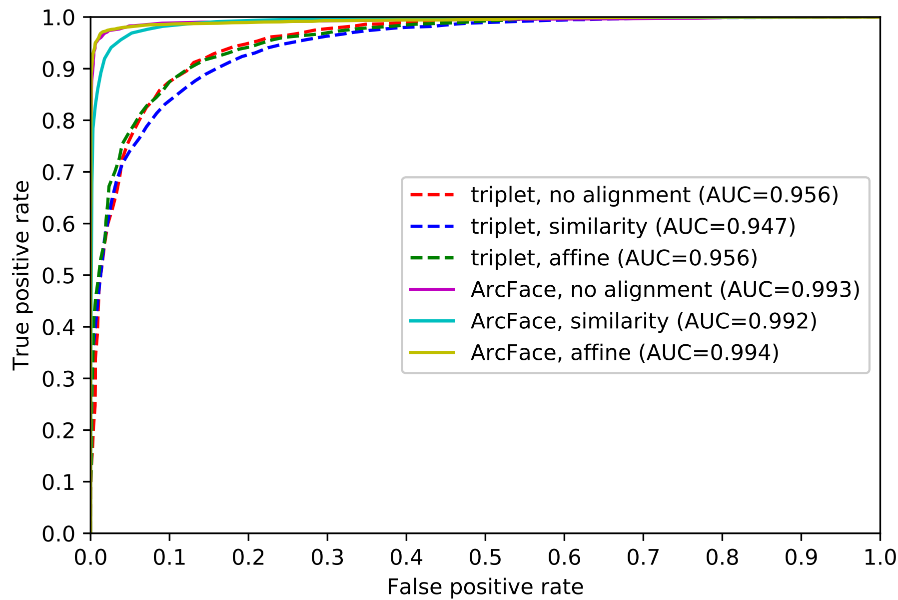

Figure 4.7. *ROC curves for LFW evaluation with different alignments for triplet and ArcFace losses.*

to the known groundtruth embeddings. All of these tasks were implemented in the base classifier. For this base classifier, where different classifiers can be derived, the groundtruth embeddings are stored in a list and the closest item in a list is found for classifying an unknown sample.

Since the base classifier uses list to store the embeddings, adding or removing embeddings do not require fitting. However, the search operation is directly proportional to the length of the list. In order to reduce search space, two different clustering based classifiers were inherited from base classifier. The basic idea behind these classifiers is to reduce the search space to the cluster that is most similar to the sample, and then find the most similar groundtruth embedding from the reduced number of embeddings. Two different clustering techniques algorithms, k-means clustering (Section 3.4.2) and hierarchical correlation clustering algorithm CHUNX (Section 3.4.3), were tested for creating data structure for the groundtruth embeddings. For k-means based approaches, *Scikit-learn* library [55] was used and CHUNX implementation was modified from the implementation of the author [45].

To reduce the number of fits while adding or removing groundtruth samples from classifier, two approaches were tested for k-means clustering. First, the clusters were refitted dynamically, only if any of the existing cluster become empty (no groundtruth samples) or full (more groundtruth samples in a cluster than set hyperparameter), with the embeddings that were in set of groundtruth embeddings. Otherwise changes in the set of groundtruth embeddings are added to the existing cluster structure. k-means clustering has one adjustable hyperparameter, which defines the number of clusters. The parameter was

converted to maximum number of items per cluster, which was computed based on the number of the embeddings to be clustered so that in average each cluster is 70% full.

Also the Faiss library [56] was considered in the implementation. However, it was noted based on the documentation that the library did not suit to the problem, where the items in the data structure might change rabidly. For dynamically adjustable set of groundtruth embeddings, following the changes in the set and tracking the indices (classes) for each embedding seemed challenging, because of the requirements for the implementation, where more information than the embedding itself is required to be linked to the indices of the classifier.

The search speed with different maximum cluster sizes as a function of search space size (number of groundtruth embeddings) with dynamically fitted k-means clustering is shown in Figure 4.8. There are some noise in the measurements related to the other processes that run simultaneously on the hardware. For example with hyperparameter setting s=20, some values are significantly lower than expected based on the trend. The search speed was computed by first adding defined number of embeddings from different identities in LFW dataset to the structure and then searching the nearest neighbour for randomly chosen 100 identities from the 1680 identities that has more than 1 facial image in the dataset.

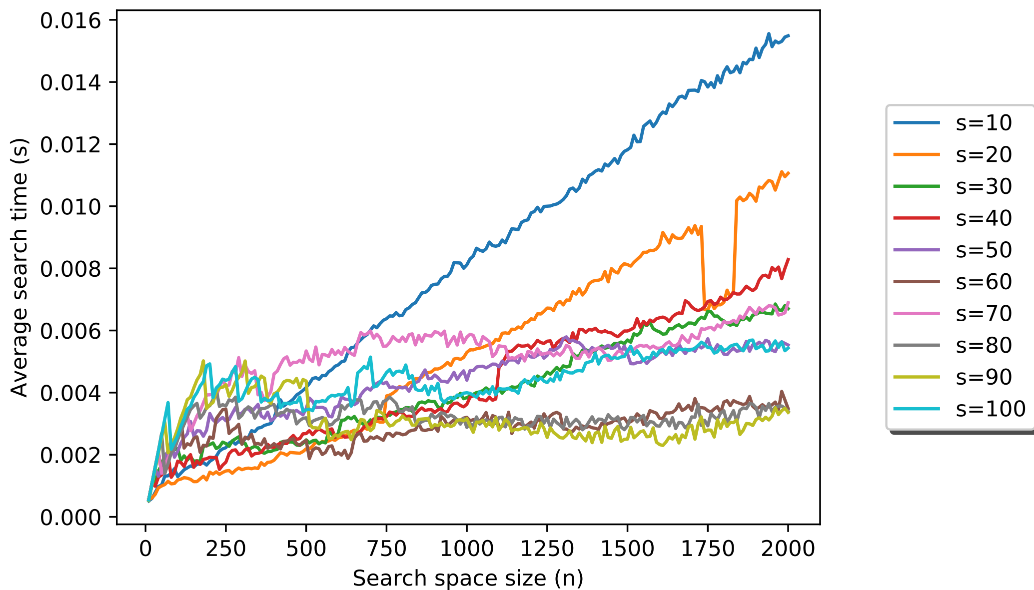

Figure 4.8. *Search speed as a function of search space size with different maximum cluster sizes (s) with dynamically adjustable* k*-means clustering classifier.*

The other approach was to create a general static cluster structure by fitting it with high number of embeddings and utilizing that as a data structure for groundtruth embeddings. For k-means, the static cluster structure does not optimize the search speed based on the number of embeddings, since the search speed is also directly proportional to the number of clusters. For fitting the static clustering model, one facial image from 9035 identities in

VGGFace2 dataset were used. The image from each identity was chosen randomly. The search speed respect to the number of cluster as a function search space size is plotted in Figure 4.9 with similar settings compared to Figure 4.8.

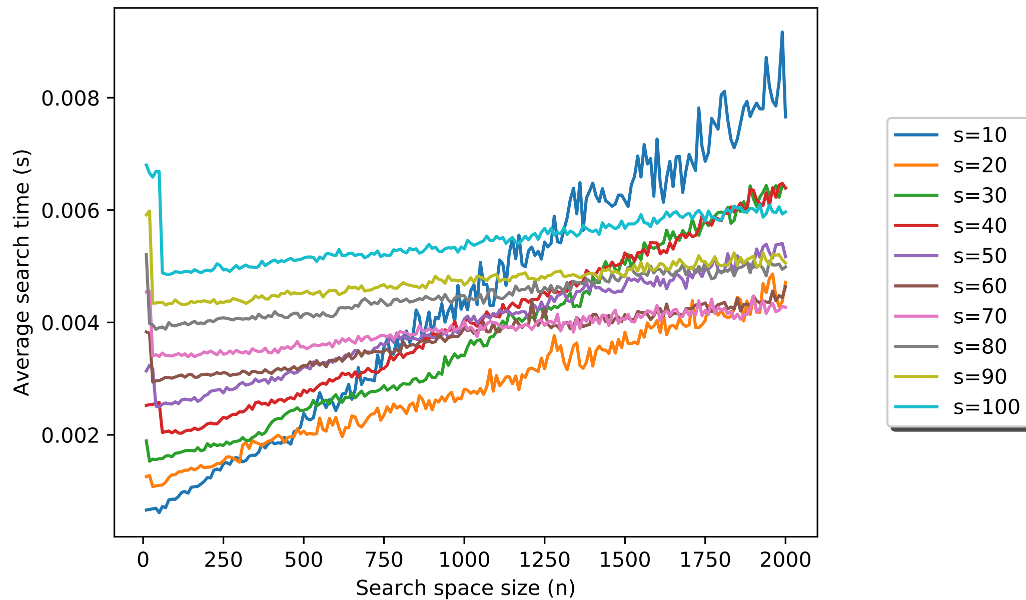

Figure 4.9. *Search speed as a function of search space size with different number of clusters (s) using static* k*-means clustering classifier.*

In order to compute eigenvectors in CHUNX algorithm, there must be at least as many embeddings in the set of groundtruth embeddings as the dimensionality of the embedding is. Therefore, the eigenvectors were computed with one facial image from 9035 identities in VGGFace2 dataset, and used in classification. CHUNX has one hyperparameter related to the maximum size of the cluster. Based on the maximum cluster size, the groundtruth embeddings were clustered dynamically so that the hierarchical tree structure was updated based on changes in the set of groundtruth embeddings. The search speed with different maximum cluster sizes as a function of search space size is plotted in Figure 4.10.

Changing the set of groundtruth embeddings was left from the speed comparisons, because the changes are very similar operations compared to searching the nearest neighbour. In addition, refitting the model can be performed parallel to the search operation and in general the classifier does not have to be located in the same edge hardware than the rest of the system.

In addition to search speed comparison, also the search accuracy from the set of groundtruth embeddings was estimated in order to estimate the classification accuracy in practice for the developed system. For this purpose, accuracy for finding the matching identity within one, five and ten nearest neighbours was computed utilizing LFW dataset. First, four classifiers described in this section, were fitted so that there were an image from each of the 5749 in

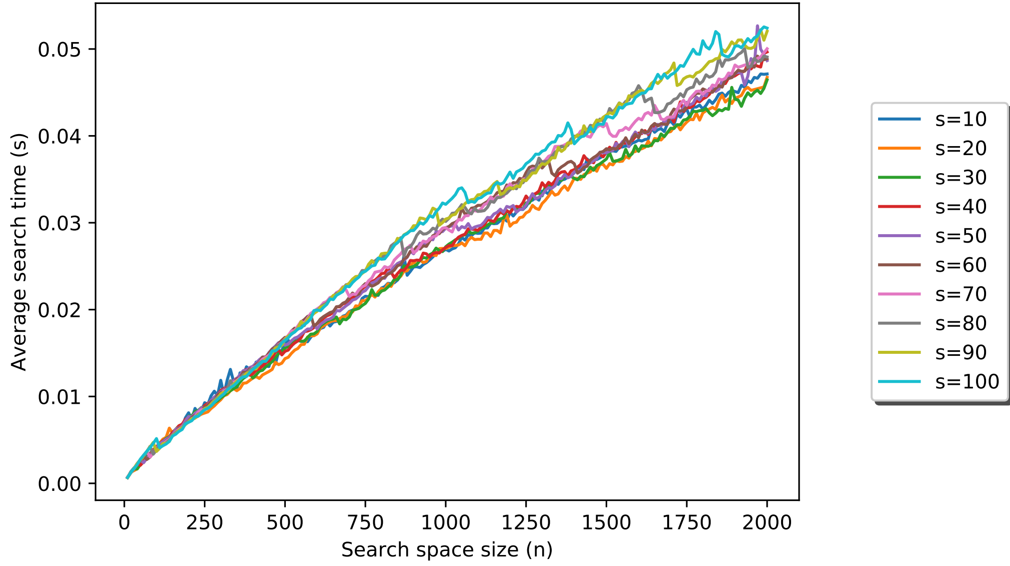

Figure 4.10. *Search speed as a function of search space size with different maximum cluster sizes (s) utilizing CHUNX based clustering classifier.*

the classifier. List classifier corresponds to base classifier. Then, one image from each of the 1680 identities that has more than one image in the dataset was collected and nearest neighbours were searched. The accuracy for each of the classifier types and two alignments (none and affine) that achieved the highest accuracy in LFW benchmark are shown in Table 4.2. The accuracies of the base classifier can be considered to be groundtruth, since it finds the nearest neighbours in whole search space while other clustering based methods first iterate to subspace and find the nearest neighbour within that space, which leads to only an approximation of the nearest neighbour.

Table 4.2. *Accuracy of the nearest neighbour search with different methods for finding correct match within one, five and ten nearest neighbours using two different alignments. Base classifier accuracies from nearest neighbour search and the rest are approximations of it utilizing cluster structure. For dynamic* k*-means and CHUNX, maximum cluster size was set to 60 and for static* k*-means, number of clusters was 100.*

	No alignment			Affine alignment		
Classifier	top-1	top-5	top-10	top-1	top-5	top-10
Base	0.762	0.875	0.905	0.826	0.0.911	0.929
Dynamic k-means	0.439	0.474	0.482	0.477	0.498	0.500
Static k-means	0.408	0.445	0.451	0.411	0.441	0.445
CHUNX	0.757	0.861	0.887	0.820	0.901	0.920

In general, k-means based classifiers had weak performance. Their performance improved if the number of the cluster is very small or close to the size of the whole search space that correspond to base classifier. For CHUNX, the maximum size of the cluster did not affect the accuracy. In addition, applying affine alignment increased the accuracy compared to

unaligned although their performance in LFW benchmark were very similar.

In order to reduce the number of false positive matches, the threshold was set low as described in Section 4.5. By reducing the threshold, the most of the false classifications were left out, but it also reduced the number of correct predictions, since classifier was not capable in finding the embedding that is similar enough.

Both VGGFace2 and LFW datasets include identities for example with different ethnicity, age and gender. While fitting the clustering models, different distribution of the identities most likely affects the search speed since the appearance of individuals may be more similar to each other than in fitted clustering models. This can be noticed from the Figures A.1, A.2 and A.3 in Appendix A, where one facial image from each cluster was added to the collections, for fitted static clustering models utilizing VGGFace2 dataset.

4.7 System implementation

For implementing the demo of the system, the pipeline in Figure 3.1 was followed. A solution to the pipeline steps was collected from Sections 4.3–4.6. Two processes were started while starting the application. These processes share the classifier object, which includes the face classifier.

The first of the processes connects to the camera that was used for recognition. For the frames of the camera stream, the pipeline of the system was followed to output the recognition results. In addition, new groundtruth images can be added to the classifier of the system before each new frame is processed and they are also processed through the pipeline. These two operations were not divided to different processes, because of the limited amount of available memory (this way, the session utilizing the GPU are only initialized once). The other process was responsible for fitting the classifier model if needed. Also, other problems related to utilizing different frameworks that access the GPU of the hardware, were encountered. Therefore, only Tensorflow framework was used demo, because it offers tools for controlling the GPU resources manually.

In production environment, the classifier could be implemented for example in the cloud in order to simplify the application. This way, also the controlling the set of groundtruth embeddings, which are the identities that are added in the system, could be separated from the face recognition pipeline. These changes would improve the performance compared to implemented demo application.

For demo application, two different pipelines were tested. For both of the pipelines in demo application, face detection uses developed MobileNetV1+SSD detector and the hierarchical classifier perform classification, since its performance was the best and it also survived well from high number of groundtruth embeddings.

The difference between the pipelines was that the first one did not include an alignment step, and the other performed affine alignment based on landmarks detected with trained

MTCNN ONet detector. Also the version of trained extractor CNN with ArcFace loss, depends based on the chosen alignment. Two different alignments were tested in demo in order to find out in practice, if the alignment could provide improvement in the performance by improving accuracy more than it slows down the application. After all, the demo application ran with 11.8 FPS without alignment step, and 11.5 FPS with affine alignment with four identities added to the set of groundtruth embeddings of the classifier. No difference was noted between the pipelines by running the demo. The threshold for positive match was set to 0.4 in order to minimize the number of false positives in classification although it caused few positive matches to be classified as false.

5. RESULTS AND DISCUSSION

The results are collected based on the implementation in Chapter 4. First, the outcome of the evaluation for the main components of the pipeline in the system are evaluated individually. For result collection, feature extraction and classification are presented as a single component since they are closely related to each other. Based on the outcome, the decisions related to the used methods in system implementation are motivated. After all, the performance of the entire system described in Section 4.7 is evaluated.

5.1 Face detection

Based on the implementation described in Section 4.3, the different methods are objectively evaluated and compared. First, by running the detectors with OpenCV library interface, COCO metrics explained in Section 3.1.4 are shown in Table 5.1. Two different mAP values are shown in the table. First mAP value with 11 different IoU thresholds is shown to measure, how accurately bounding boxes correspond to groundtruth annotation where weaker IoU's are penalized. In addition, the mAP with minimum 0.5 IoU is added to find out, how many of the bounding boxes are found with more loose estimation. For OpenCV detectors, the information related to training datasets is not available, which might affect the results.

Table 5.1. *Detector's performance.*

Dataset	Detector	mAP (%)	mAP@IoU=0.5 (%)
FDDB	Viola-Jones *	15.7	55.4
	ResNet+SSD *	**40.8**	**84.9**
	YOLOv3 Tiny	32.7	84.3
	MobileNetV1+SSD	33.3	83.6
WIDER FACE	Viola-Jones *	3.7	10.8
	ResNet+SSD *	7.4	14.0
	YOLOv3 Tiny	**9.7**	**22.4**
	MobileNetV1+SSD	5.8	11.4

* OpenCV implementation

In addition to accuracy estimation, also inference times with different methods are measured by computing inference times while predicting bounding boxes for FDDB dataset with hardware described in Section 4.2. The results are shown in Table 5.2. For each of the CNN based detector a input size is either 300×300 or 320×320 and for Viola-Jones detector FDDB dataset default image sizes are used. For measurements with CPU, OpenCV library interface is used in detection. In case of MobileNetV1+SSD detector, running the detection

on CPU with Tensorflow framework, the inference time was 157 ms which is 31% higher compared to OpenCV. However, the detection result is more accurate with Tensorflow which can be seen from Figure 4.2. For GPU inference times, the frameworks described in Section 4.3 were used.

Table 5.2. *Face detector inference times in milliseconds (ms).*

Detector	CPU	GPU
Viola-Jones *	**48**	-
ResNet+SSD *	107	-
YOLOv3 Tiny	203	**38**
MobileNetV1+SSD	120	54

* OpenCV implementation

From the tables, it can be seen that the CNN based detectors have very similar performance with FDDB dataset and their ability to detect faces does not differ from each other significantly. Small differences can also be explained with differences in bounding box localization which is respective to the model, although the detection accuracy is practically the same. However, YOLOv3 Tiny detector is capable of performing better with WIDER FACE dataset, which can be considered as more challenging dataset. Therefore, it is the best option for detection in difficult cases because it is capable of learning more detailed face description from the data and being more tolerant to factors related to image capturing environment. For Viola-Jones detector, the performance is weaker compared to the other detectors. As seen from Figure 4.1 however, the square shape of the output with Viola-Jones algorithm also weakens the score since the face cropping is more inaccurate.

By testing the detectors with video stream, all three CNN based detectors offers performance which is good enough for this application from subjective view. From this view, the Viola-Jones detector performance is too weak, because the detection works only with pose where face is aligned straight towards the camera. Therefore, the choice between the CNN detectors can be made based on inference time and other implementation related reasons.

As shown in table 5.2, the performance is proportional to the accuracy of the detector. However, even a relatively low power GPU, as on the Jetson TX2 module, is capable of boosting the performance significantly. Also the framework and its optimizations matter significantly to the performance. Since 50 ms inference time corresponds to 20 FPS and the system has also other components than the detector, the real-time requirement of the application limits the options for choosing the detector so that the inference time should be under this value. After all, MobileNetV1+SSD detector implemented with Tensorflow and utilizing GPU, provides the best compromise for this application although the performance of YOLOv3 Tiny is better, because of the reasons related to implementation covered in Section 4.7.

5.2 Facial image alignment

For evaluating the alignment, two different approaches are used. Since the alignment is based on first detecting the keypoints (landmarks), the robustness of the landmark localization is an essential component in the alignment. For estimating the landmark localization, the accuracy of two different models implemented in Section 4.4 are collected to Table 5.3 based on metrics defined in Section 3.2.2 by utilizing VGGFace2 dataset test partition. The facial image width which is 96×96 based on the output of the face detector, is used in normalizing the distances. For some of the facial images, the groundtruth annotations was noticed to be unreliable because the inter-ocular distance was impossible to define due to pose or occlusion.

Table 5.3. *Landmark detector performance for each of the five detected landmarks. The scores are percentages of the width of the image.*

Detector	left eye	right eye	nose tip	mouth left	mouth right	average
Dlib	8.7	8.8	13.2	8.3	8.4	9.5
MTCNN ONet	3.3	3.3	3.9	3.5	3.5	3.5

The CNN based MTCNN ONet landmark detector outperforms the Dlib based detector by wide margin. Because of the input image size, the percentages percentages in Table 5.3 correspond approximately to average error in pixels. Therefore, MTCNN ONet detector has average error of bit under four pixels for each landmark. Based on experiments and visualizations shown in Section 4.4, this detector is accurate enough to be used for alignment. As it can be seen from Table 5.4, a lightweight carefully selected CNN architecture is also capable of beating more traditional methods in terms of inference time, especially if a GPU is available for accelerating computing. With the inference time of 8 ms, the landmark localization and the alignment does not affect the inference time of the complete system significantly because the face detection is significantly slower operation. Therefore, the decision whether to include the alignment in the pipeline or skip it completely, can be made only based on classification accuracy that the feature extractor is capable of achieving.

Table 5.4. *Landmark detector inference times in milliseconds (ms).*

Detector	CPU	GPU
Dlib	24	-
MTCNN ONet	17	8

The implementation of the similarity alignment did not work out well which can be seen from the Table 4.1. Most likely fixed locations of the eyes were set badly, which lead to loss of information in the facial images. However, the affine transformation with same fixed groundtruth locations than similarity transformation, achieved slightly improved accuracy of the classification with LFW dataset. The main difference between the alignments is that similarity transform can only scale the input image while affine transformation can perform

also stretching. This way, more uniform facial image was obtained which resulted to higher accuracy although the fixed locations of the landmarks might not have been optimal.

For estimating the output from the alignments, average images with different alignments were created utilizing test partition of VGGFace2 dataset. They are shown in Figure 5.1. From the figure it can be seen that booth the similarity and affine alignments are capable of locating eyes accurately compared to unaligned case. In addition, affine alignment aligns mouth more accurately compared to similarity alignment. However, the face fills smaller area with average images computed with alignments, which leads to loss of information and weakens the accuracy.

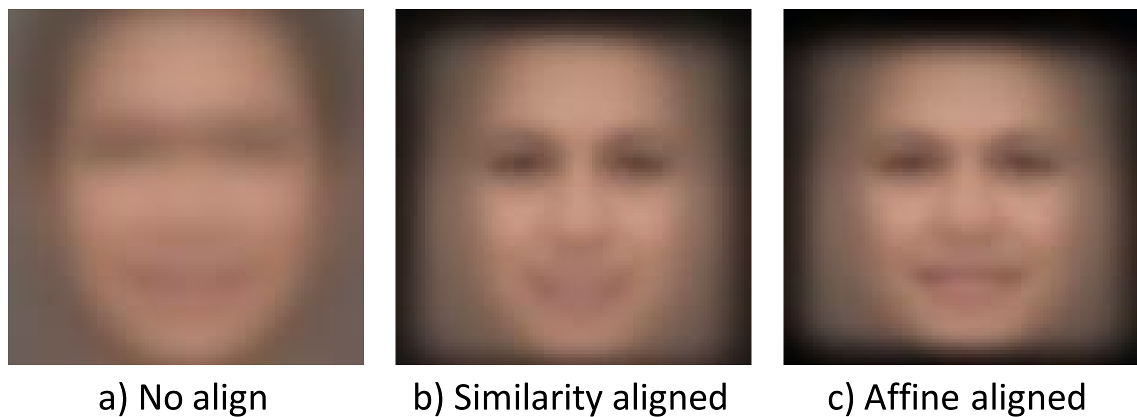

Figure 5.1. *Average images with different alignments utilizing test partition of VGGFace2 dataset.*

5.3 Face recognition

Firstly, the performance of the feature extractor and the classifier that are responsible for performing the identity recognition based on the facial image, are evaluated in theory with the LFW benchmark. In Table 5.5, the accuracy of the best performing extractor, where affine transformation was applied and the extractor CNN was trained utilizing ArcFace as a loss function, is compared to other approaches. Based on classification results, a network trained with ArcFace loss and applying affine alignment before extraction, achieves the highest performance based on experiments and is therefore used in this section.

Table 5.5. *Accuracy of the feature extractor of this thesis compared to other methods.*

Extractor	Accuracy (%)
FaceNet [8]	99.63
MobileFaceNet (96 $\times$ 96 input) [35]	99.08
this thesis	**97.97**
Human, cropped [57]	97.53
DeepFace [7]	97.35

The accuracy achieved in this thesis beats for example human and DeepFace [7] model. However, other methods presented in this thesis (FaceNet[8] and MobileFaceNet [35])

achieve higher accuracy. On the other hand, FaceNet has significantly larger CNN structure and MobileFaceNet's outputting embedding dimensionality is four times higher, which would make nearest neighbour search more difficult problem.

In addition, the distribution of the distances between the embeddings with the LFW evaluation protocol was tested. While it was noted in Section 4.1 that the LFW dataset has overlapping identities with VGGFace2 dataset, also similar pairs were extracted randomly from VGGFace2 dataset test partition. The distributions of the pairs with both datasets are plotted in Figure 5.2. From the plots it can be seen that the optimal threshold for rejecting false positives is approximately 0.45. In general, the VGGFace2 dataset includes bigger variety related to the environment, where the images have been captured, compared to LFW dataset, which causes an increase higher distances between facial images in pairs where the images represent same identity.

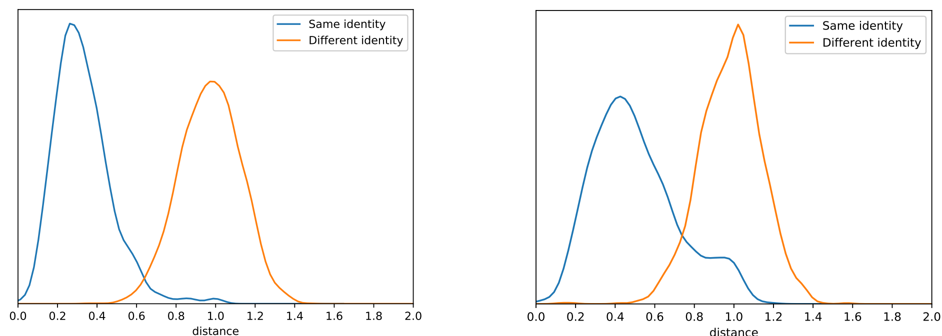

Figure 5.2. *Distribution of the pairs with same identity and different identities with LFW (left) and VGGFace2 (right) datasets with magnitude on y-axis.*

The distributions shown in Figure 5.2 visualize, how well the feature extraction is capable of separating different identities from each other. The most interesting region is at the point where the line of the positive pairs intersect with the line of the negative pairs. In general, the smaller the value at y-axis is at the point of intersection, the better the extractor is for separating different identities.

The task of the classifier differs from pairwise comparisons that LFW evaluation protocol is based on. In order to test the performance in practise, the nearest neighbour was performed with the set of known embeddings where each of the embeddings correspond different identity, utilizing LFW dataset. The achieved accuracy for finding the nearest neighbour are presented in Table 4.2, where it was noted that base classifier and CHUNX based classifier both achieved very similar nearest neighbour search accuracy. Therefore, CHUNX is capable of clustering the embeddings very accurately without problems related to high dimensionality of the data. On the other hand, k-means based classifiers had significantly lower accuracy that is related to curse of dimensionality problem that exists with high dimensional data. CHUNX provide effective solution to this problem and it can be used in this application without weakening the accuracy unlike other tested clustering approaches.

Also, the search speed relative to the search space size was measured. The results are shown in Figures 5.3 and 5.4. The hyperparameters for the methods were chosen to consider both accuracy and the speed. For dynamic k-means, the maximum number of ground embeddings per cluster was set to 40, static k-means had 20 clusters and the maximum number of ground embeddings per cluster for CHUNX was 50. Compared to base classifier, different clustering based classifiers are capable of speeding up the search with almost any search space size. From Figure 5.3 can be seen that the chosen clustering approaches does not have significant computational complexity that is not related to the size of the search space. The dynamically adjustable clustering methods is similar to base classifier as long as the defined maximum cluster size is full. After that the speed increases slower compared to base classifier since items are located in multiple clusters.

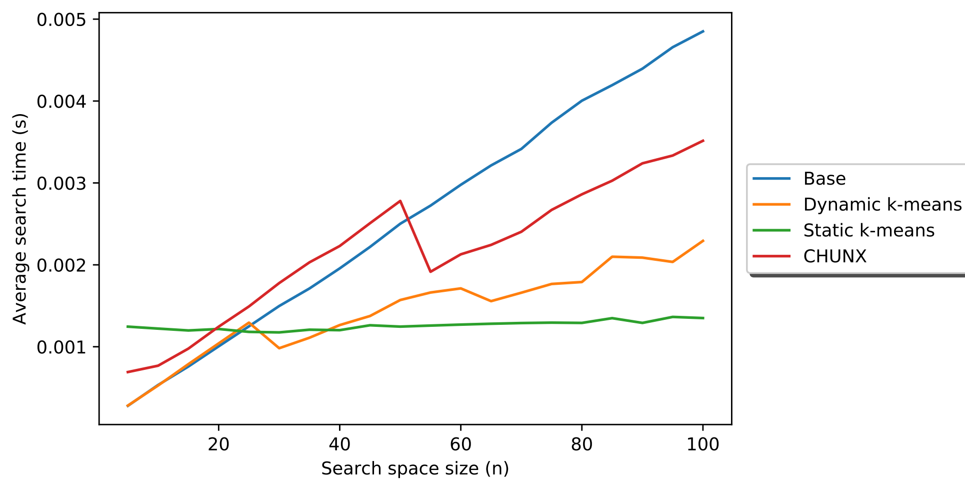

Figure 5.3. *Search speed as a function of search space size with different classifiers with small number of groundtruth embeddings.*

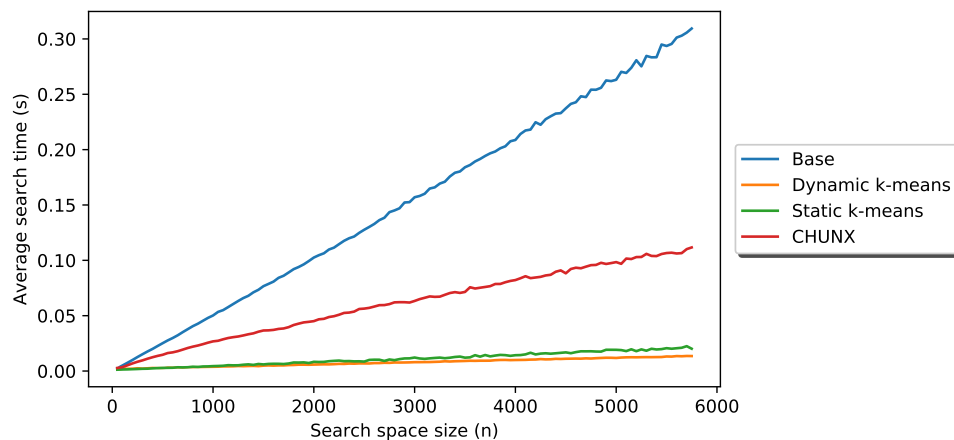

Figure 5.4. *Search speed as a function of search space size with different classifiers with high number of groundtruth embeddings.*

6. CONCLUSION

The solution for face recognition system was presented in this thesis. The system was designed to run in real-time with low inference time on hardware, which computational capability was limited. In order to solve the problem, the pipeline that the system followed, was defined. In the pipeline, the problem was divided to steps that were solved and evaluated individually. After all, a solution for each of the steps were collected and a demo system was developed to demonstrate the system in a real life environment.

The pipeline consisted of face detection, where faces were detected from the images and cropped for the following steps of the pipeline. After that, different preprocessing methods were tested for the facial images before the following steps. Next, a feature embedding was extracted from the facial images and finally a nearest neighbour search was performed to compare extracted features to the set of known groundtruth embeddings that represents classes for the output of the system.

For face detection different methods were compared. The tested CNN based approaches, two lightweight network architectures (ResNet and MobileNetV1) with SSD detector and YOLOv3 Tiny, achieved similar performance. The fourth tested detector, Viola-Jones, did not perform as well as the other approaches. Each of the three CNN could have been implemented to demo application. The decision for using MobileNetV1+SSD detector in the demo application was made based on the implementation, because it was the easiest to integrate to the demo.

In order to classify facial images extractor by the face detector utilizing nearest neighbour search, the features from the facial images were extracted first. In addition, different image transformation based methods were tested if they could help feature extractor to separate different identities better. For image transformation, five facial landmarks were detected first. It was noted that MTCNN ONet architecture was superior compared to the detector implemented in Dlib. Based on detections, three different preprocessing approaches were tested. Two different transformations, similarity and affine, were tested and compared to the situation, where no transformation was applied before feature extraction. In LFW benchmark, the unaligned and affine transformed inputs for the extractor achieved similar accuracy, while similarity transform weakened the accuracy little.

Two different loss functions were tested while developing the feature extractor. The architecture of the CNN for feature extraction was taken from MobileFaceNet. ArcFace loss outperformed triplet loss in feature extraction with LFW benchmarks at least with the chosen CNN architecture. With Arcface loss trained models, applying the affine transformation before extraction increased the accuracy of the nearest neighbour search,

which models the accuracy of the system in practice compared to extraction with unaligned facial images.

For nearest neighbour search, different clustering approaches, partitioning (k-means) and hierarchical clustering (CHUNX) were tested and their approximations for search accuracy compared to the exact nearest neighbour search from the list of groundtruth embeddings. It was noted that k-means based approaches were very fast with bigger search space sizes, but the accuracy of their nearest neighbour approximations were significantly worse than the exact search. With CHUNX, the slope relative to the increase of search space size and the time elapsed in search was significantly higher compared to k-means based approaches, but still around three times smaller compared to exact search from list. In addition, the accuracy for the approximate search with CHUNX was almost as good compared to exact search. After all, the CHUNX was noted to be the best solution for nearest neighbour search.

The goals that were set for the thesis, were partially achieved. The system was capable of running in real-time on Jetson TX2 hardware, but the FPS was lower than desired. The face detection was capable of detecting well visible faces with high confidence, but for example larger occlusions caused problems for the detection. The set recognition accuracy was almost reached in LFW benchmark, but tests in practice achieved weaker accuracy than desired. However, these false recognition can be removed by setting tight thresholds for the recognition that minimized very well the number of false positives which is very important aspect for face recognition system. The search speed was directly proportional to the search space size, but the slope was decreased significantly without weakening the accuracy of the search. After all, the developed demo application suits very well at least for demo purposes, but for the production ready system, some optimization should be done.

In future, the face detector could be improved compared to the implemented detectors, that are originally designed for general object detection. Currently the object detection takes approximately 60% of the inference time of the developed system and it could be lower. In addition, the transformation applied before the feature extraction could be tuned. Especially the similarity transformation lost information from the data rather than transforming it to the format that would help feature extractor to achieve higher accuracy in classification by separating different identities more accurately.

All in all, the development of the face recognition system presented in this thesis succeeded well. With limited computational resources, the overall design of the system is a key element. Luckily, even a small size GPU is capable of increasing the computational power of an embedded hardware enough for utilizing multiple CNNs simultaneously to solve the problems such as face recognition. The CNN architectures with low computational cost are currently the optimal solution for the problem. However, while the size of the CNN decreases in order to achieve lower computational cost, also the task that the model is solving, must be defined more accurately. With smaller model sizes, the representational

capability of the model decreases and models ability to ignore different factors related to environment such as a pose of the face in the image, weakens.

REFERENCES

[1] (2016) Vgg16 architecture, image. Accessed: 17 February, 2019. [Online]. Available: https://heuritech.files.wordpress.com/2016/02/vgg16.png?w=768

[2] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "Vggface2: A dataset for recognising faces across pose and age," in *International Conference on Automatic Face and Gesture Recognition*, 2018.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, Dec 2001, pp. 511–518.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[5] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, Dec 1943. [Online]. Available: https://doi.org/10.1007/BF02478259

[6] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, July 1959.

[7] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 1701–1708.

[8] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *CoRR*, vol. abs/1503.03832, 2015. [Online]. Available: http://arxiv.org/abs/1503.03832

[9] M. Wang and W. Deng, "Deep face recognition: A survey," *CoRR*, vol. abs/1804.06655, 2018. [Online]. Available: http://arxiv.org/abs/1804.06655

[10] M. Sharif, F. Naz, M. Yasmin, M. A. Shahid, and A. Rehman, "Face recognition: A survey," *Journal of Engineering Science and Technology Review*, vol. 10, no. 2, pp. 166–177, 2017.

[11] R. Ramachandra and C. Busch, "Presentation attack detection methods for face recognition systems: A comprehensive survey," *ACM Comput. Surv.*, vol. 50, no. 1, pp. 8:1–8:37, Mar. 2017. [Online]. Available: http://doi.acm.org.libproxy.tuni.fi/10.1145/3038924

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] K. P. Murphy, *Machine learning: a probabilistic perspective*. Cambridge, MA: MIT Press, 2012.

[14] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction. 2nd ed., corrected at 12th printing*. New York: Springer, 2017.

[15] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[16] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML’10. USA: Omnipress, 2010, pp. 807–814. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104425

[17] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation functions: Comparison of trends in practice and research for deep learning,” *CoRR*, vol. abs/1811.03378, 2018. [Online]. Available: http://arxiv.org/abs/1811.03378

[18] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[20] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[23] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[24] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: common objects in context,” *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[26] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[27] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[28] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation,” *CoRR*, vol. abs/1801.04381, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381

[29] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct 2010.

[30] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[31] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: towards real-time object detection with region proposal networks,” *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[32] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, Sep. 2010.

[33] P. Jaccard, “Etude de la distribution florale dans une portion des alpes et du jura,” *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 01 1901.

[34] COCO - Common Objects in Context: detection evaluation. Accessed: 17 February, 2019. [Online]. Available: http://cocodataset.org/#detection-eval

[35] S. Chen, Y. Liu, X. Gao, and Z. Han, “Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices,” *CoRR*, vol. abs/1804.07573, 2018. [Online]. Available: http://arxiv.org/abs/1804.07573

[36] B. Amos, B. Ludwiczuk, and M. Satyanarayanan, “Openface: A general-purpose face recognition library with mobile applications,” CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

[37] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*, 2nd ed. Cambridge, UK;New York;: Cambridge University Press, 2003.

[38] V. Kazemi and J. Sullivan, “One millisecond face alignment with an ensemble of regression trees,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 1867–1874.

[39] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, “Joint face detection and alignment using multi-task cascaded convolutional networks,” *CoRR*, vol. abs/1604.02878, 2016. [Online]. Available: http://arxiv.org/abs/1604.02878

[40] J. Deng, J. Guo, and S. Zafeiriou, “Arcface: Additive angular margin loss for deep face recognition,” *CoRR*, vol. abs/1801.07698, 2018. [Online]. Available: http://arxiv.org/abs/1801.07698

[41] T. Fawcett, “An introduction to roc analysis,” *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.

[42] G. B. H. E. Learned-Miller, “Labeled faces in the wild: Updates and new reporting procedures,” University of Massachusetts, Amherst, Tech. Rep. UM-CS-2014-003, May 2014.

[43] Labeled faces in the wild. Accessed: 19 February, 2019. [Online]. Available: http://vis-www.cs.umass.edu/lfw/#download

[44] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*, 3rd ed. Amsterdam;Boston;: Elsevier/Morgan Kaufmann, 2012.

[45] I. Kampman and T. Elomaa, “Hierarchical clustering of high-dimensional data without global dimensionality reduction,” in *Foundations of Intelligent Systems*, M. Ceci, N. Japkowicz, J. Liu, G. A. Papadopoulos, and Z. W. Raś, Eds. Cham: Springer International Publishing, 2018, pp. 236–246.

[46] H. Abdi and L. J. Williams, “Principal component analysis,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/wics.101

[47] S. Yang, P. Luo, C. C. Loy, and X. Tang, “Wider face: A face detection benchmark,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[48] V. Jain and E. Learned-Miller, “Fddb: A benchmark for face detection in unconstrained settings,” University of Massachusetts, Amherst, Tech. Rep. UM-CS-2010-009, 2010.

[49] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, “Labeled faces in the wild: A database for studying face recognition in unconstrained environments,” University of Massachusetts, Amherst, Tech. Rep. 07-49, October 2007.

[50] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.

[51] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, “Speed/accuracy trade-offs for modern convolutional object detectors,” *CoRR*, vol. abs/1611.10012, 2016. [Online]. Available: http://arxiv.org/abs/1611.10012

[52] J. Redmon, “Darknet: Open source neural networks in c,” http://pjreddie.com/darknet/, 2013–2016.

[53] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[54] D. E. King, “Dlib-ml: A machine learning toolkit,” *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[56] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with gpus,” *arXiv preprint arXiv:1702.08734*, 2017.

[57] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar, “Attribute and simile classifiers for face verification,” in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 365–372.

APPENDIX A: VISUALIZATIONS OF *K*-MEANS AND CHUNX CLUSTERING RESULTS

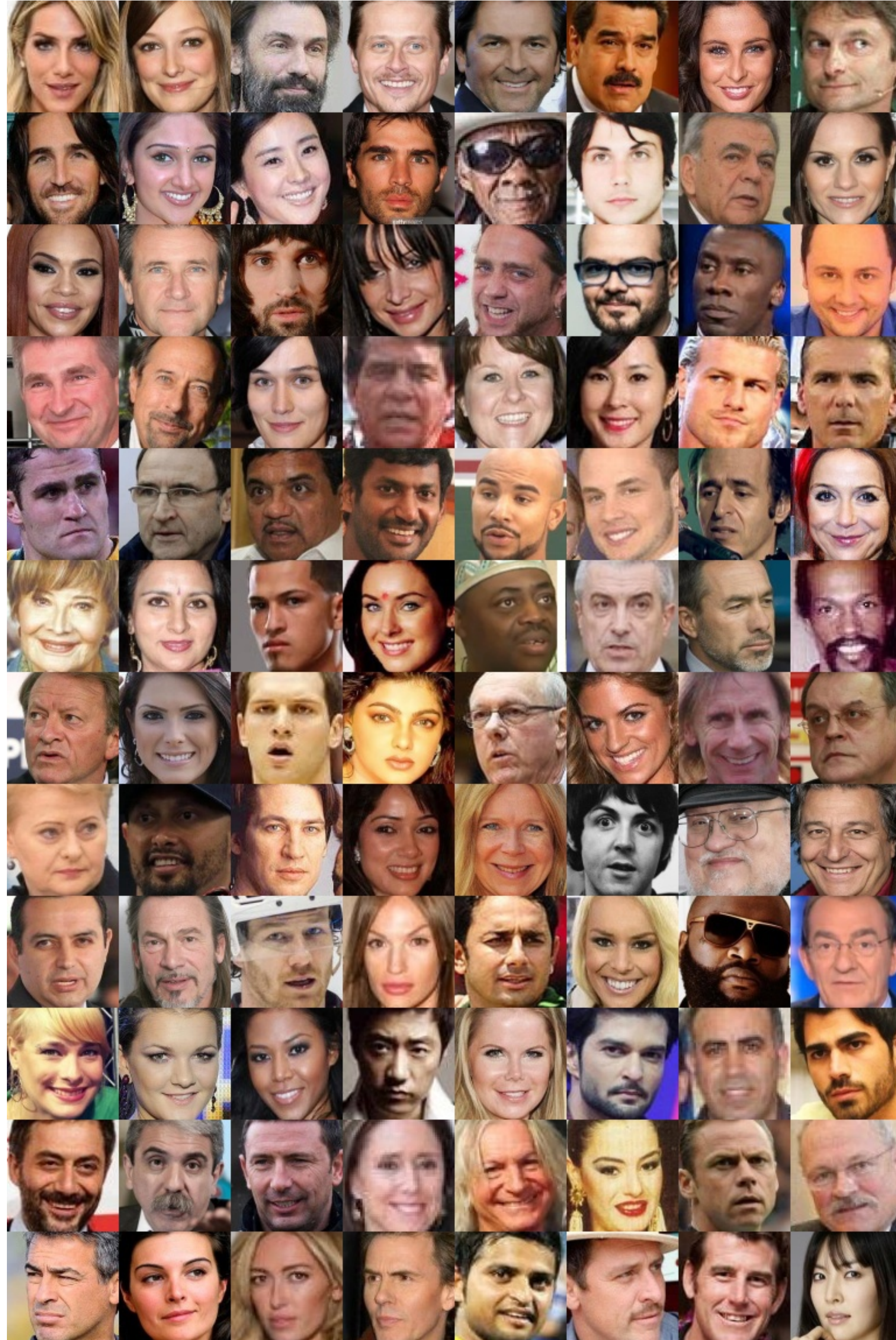

Figure A.1. *Visualization of* k*-means clustering with 96 clusters utilizing 9035 identities in VGGFace2 dataset, where one image from each cluster was picked (images taken from VGGFace2 dataset [2]).*

Figure A.2. *Visualization of CHUNX clustering, where maximum number of images per cluster was set to 50 and only clusters with more than 16 facial images were chosen, utilizing 9035 identities in VGGFace2 dataset, where one image from each cluster was picked (images taken from VGGFace2 dataset [2]).*

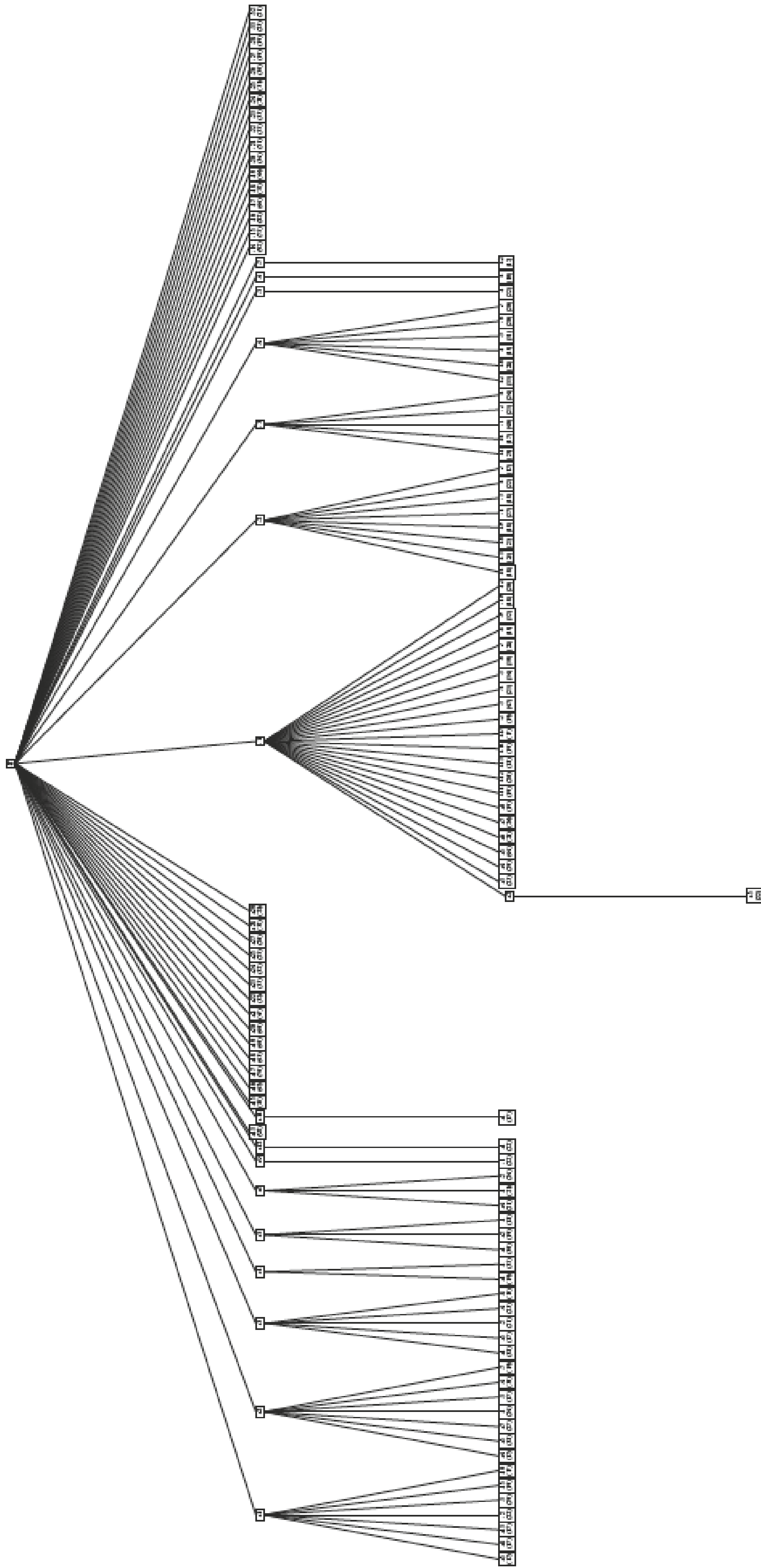

Figure A.3. *Visualization of CHUNX clustering tree, where maximum number of images per cluster was set to 100 and only clusters with more than 16 facial images were chosen, utilizing one image from 9035 identities in VGGFace2 dataset. The nodes in the leaf include the facial images. For the nodes that have only one child in the visualization, the other children are due to the defined minimum cluster size.*