Deep Learning for Human Emotion Recognition from video

Eleni Kamenou

Department of Electrical & Computer Engineering
University of Thessaly

Supervisors:
Prof. Gerasimos Potamianos
Dr. Jesus Martinez del Rincon

Volos, March 5, 2019

Αναγνώριση ανθρώπινου συναισθήματος από βίντεο με βαθιά μάθηση

Ελένη Καμένου

Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Επιβλέποντες:
Γεράσιμος Ποταμιάνος
Χεσούς Μαρτίνεζ Ντελ Ρινκόν

Βόλος, Μάρτιος 5, 2019

Deep Learning for Human Emotion Recognition from video

Eleni Kamenou

Department of Electrical & Computer Engineering
University of Thessaly

Supervisors:
Prof. Gerasimos Potamianos
Dr. Jesus Martinez del Rincon

Volos, March 5, 2019

Acknowledgements

I would like to express my great appreciation to Prof. Gerasimos Potamianos, my first supervisor, for his valuable and constructive suggestions during the planning and development of this research work. I would also like to thank Dr. Jesus Martinez del Rincon for his advice and assistance in keeping my progress on schedule. His willingness to give his time so generously has been very much appreciated. I would also like to extend my thanks to the staff members of Electronics Research Lab of my department and the staff members of School of Electronics, Electrical Engineering & Computer Science of Queen's University for their help in offering me the resources to run the program. Finally, I must express my very profound gratitude to my family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Abstract

Emotions play an important role in human-to-human communication and interaction, allowing people to express themselves beyond the verbal domain. The ability to understand human emotions is desirable for the computer as well, and thus several applications of automatic emotion recognition can be found in different domains. In this Thesis, an attempt has been made to classify human emotions from video recordings, based on the physical characteristics, into six main categories: anger, fear, happiness, disgust, sadness, and surprise. In order to create this emotion recognition system, ML methods such as transfer learning and Deep Neural Networks have been used. Initially, the required amount of data had to be obtained, and preprocessing techniques were applied to the videos in order to transform them into appropriate input for pretrained neural network architectures. The research then focused on applying fine-tuning on well known convolutional neural network architectures and making them capable of classifying emotions effectively. Finally, several different implementations of the developed framework were evaluated and the results were analyzed and examined thoroughly.

Περίληψη

Αδιαμφισβήτητα, τα συναισθήματα διαδραματίζουν κυρίαρχο ρόλο στην ανθρώπινη επικοινωνία, καθώς επιτρέπουν στα άτομα να εκφράζονται εξωγλωσσικά. Η ικανότητα αναγνώρισης των ανθρώπινων συναισθημάτων αποτελεί πρόκληση και στον τομέα της επιστήμης των υπολογιστών, γ' αυτό και σχετικές εφαρμογές χρησιμοποιούνται σε πολλούς τομείς. Στην παρούσα διπλωματική εργασία, γίνεται μία προσπάθεια για την κατηγοριοποίηση των συναισθημάτων ανθρώπων από βίντεο, βασισμένη στα φυσικά τους χαρακτηριστικά σε 6 βασικές κατηγορίες: θυμός, φόβος, χαρά, αηδία, λύπη και έκπληξη. Για τον σκοπό αυτό χρησιμοποιήθηκαν τεχνικές μηχανικής μάθησης όπως μεταφερόμενη μάθηση και βαθιά νευρωνικά δίκτυα. Αρχικά έπρεπε να συλλεχθεί ο απαιτούμενος όγκος δεδομένων και να εφαρμοστούν τεχνικές προεπεξεργασίας στα βίντεο ώστε να μετατραπούν σε κατάλληλη είσοδο για αρχιτεκτονικές προεκπαιδευμένων νευρωνικών δικτύων. Έπειτα, η έρευνα εστιάστηκε στην εφαρμογή ακριβούς προσαρμογής σε ευρέως γνωστές αρχιτεκτονικές συνελικτικών νευρωνικών δίκτυων, ωστέ να μπορούν να αναγνωρίζουν συναισθήματα με ακρίβεια. Τέλος, διαφορετικές υλοποιήσεις των εν λόγω συστημάτων εξετάστηκαν και τα αποτελέσματα των πειραμάτων απεικονίστηκαν και αναλύθηκαν διεξοδικά.

Contents

List of Figures

List of Tables

List of Acronyms

ML	Machine Learning
DNN	Deep Neural Networks
RNN	Recurrent Neural Networks
CNN	Convolutional Neural Networks
NN	Neural Networks
HCI	Human Computer Interaction
FER	Facial Emotion Recognition
SVM	Support Vector Machine
AAM	Active Appearance Model
LSTM	Long Short-Term Memory
BP	Backpropagation
ReLU	Rectified Linear Unit
Adam	Adaptive Moment Estimation
cdf	Cumulative Distributed Function
RMS	Root Mean Squared
FC	Fully Connected

Chapter 1

Introduction

One of the greatest abilities of human beings is their ability to feel. Emotions are of primary importance in social interaction, perception, and human intelligence [8]. Basically, emotions can be expressed through several social behaviors, including facial expressions, speech, text, gestures, etc. Since perception and experience of emotions are vital for communication in the social environment, understanding emotions becomes essential for humans in everyday life.

Over recent decades automated recognition of emotions by computers has become a challenging task for both academia and industry, since emotional intelligence is an important part of artificial intelligence. Emotion-aware systems greatly empower human computer interaction (HCI) and they have already been a revolutionizing computer vision research topic [9] [10] [11]. Such systems have an interesting and wide variety of applications, including monitoring driver state (e.g. fatigue state), measuring player's emotional experience in gaming, detecting depression in individuals or diagnosing developmental disorders of children through monitoring their facial expressions. Emotion-sensitive machines are also used to improve automatic tutoring by providing valuable information about whether the user is finding examples boring or irritating [12].

Inspired by the recent success of deep learning, emotion analysis tasks have also been enhanced by the adoption of deep learning algorithms, such as convolutional neural networks (CNN), recurrent neural networks (RNN), as well as hybrid methods that combine both [9] [10]. Furthermore, automated recognition of emotions has been addressed using different modalities: audio, visual, physiological measurements, and their combinations [13] [14].

Among nonverbal components, visual information is the main information channel in interpersonal communication. Especially facial expressions are responsible for a huge proportion of nonverbal communication. Therefore, in this work we focus on the visual modality of the data by extracting spatio-temporal information from video recordings of human faces and use it for emotion recognition based on fine-tuned pretrained CNN models.

Although as human beings we experience a plethora of different types of emotions in our lifetime, for automatic recognition it is necessary to declare a set of discrete emotional classes in order to simplify machine learning (ML) problems. In the early 1970s, Ekman indicated that humans perceive certain basic emotions with respect to facial expressions in the same way, regardless of culture [15]. According to his studies, humans share six basic emotions: happiness, sadness, fear, anger, disgust, and surprise. Ekman's theory has been widely accepted and inspired many researchers for future work. Therefore, we will focus on those six emotional classes for this thesis project.

1.1 Related Work

Within the last few years, interest in automatic facial emotion recognition (FER) has been increasing with the rapid development of artificial intelligence, and several different approaches of FER systems have been proposed during the past decades.

1.1.1 Conventional FER Approaches

In conventional FER approaches, the FER pipeline is composed of 3 major steps: (1) face or facial component detection, (2) feature extraction, and (3) emotion classification. First, a face image is detected from an input image, and facial components (e.g., eyes and nose) or landmarks are detected from the face region. Second, various spatial and temporal features are extracted from the facial components. Third, classification algorithms, such as a support vector machines (SVM), AdaBoost, and random forests, produce the recognition results using the extracted features [16] [17].

Shan et al. [18] utilized local binary patterns for feature extraction and classified various facial expressions from static images using an SVM. Other image approaches use an active appearance model (AAM) to extract facial feature points and combine useful local shape features to form a classifier [19]. The role of the AAM, which is built during the training procedure, is to match a statistical model of object shape and appearance to a new unseen image.

For video FER, many systems have been used to measure the geometrical displacement of facial landmarks between the current frame and previous frames as temporal features. The main difference between FER for static images and for video sequences is that the landmarks in the latter are tracked frame-by-frame and the system generates new dynamic features through displacement between the previous and current frames. Polikovsky et al. [20] presented facial micro-expressions recognition in video sequences by dividing face regions and then generating 3D-Gradients orientation histogram from the motion in each region.

1.1.2 Deep Learning Based FER Approaches

In contrast to traditional approaches using handcrafted features, deep learning has emerged as a general approach to ML, yielding state-of-the-art results in many computer vision studies with the availability of big data. Such emotion recognition approaches highly reduce the dependence on face-physics-based models and other pre-processing techniques by enabling learning directly from the input images. However, the amount of annotated training data has a great impact on deep neural network (DNN) models' performance, since deep learning algorithms require large datasets to achieve state-of-the-art results, and this is considered to be their main limitation.

Deep learning algorithms, including CNN and RNN, have been used for feature extraction and classification of emotions from video recordings. In 2015, Jung et al. [21] used two different types of CNN: the first extracted temporal appearance features from the image sequences, whereas the second extracted temporal geometry features from temporal facial landmark points. These two models were combined using a new integration method to boost the performance of facial expression recognition. In the 2016 Emotion Recognition in the Wild (EmotiW) Challenge, Bragal et al. [22] proposed a deep learning system for emotion recognition. Deep CNNs such as VGG and ResNet were used as feature extractors. Overall, the system consisted of the following pipelined modules: face detection, frames pre-processing, deep feature extraction, feature encoding through the video frames and, finally, an SVM classifier.

Many approaches have adopted a CNN directly for emotion recognition. However, due to the fact that CNN-based methods cannot reflect temporal variations in the facial components, several hybrid approaches combining a CNN for the spatial features of individual frames, and long short-term memory (LSTM) for the temporal features of consecutive frames, have been developed. LSTM is a special type of RNN capable of learning long-term dependencies in the data. Kahou et al. [23] proposed a hybrid RNN-CNN framework for propagating information over a sequence using a continuously valued hidden-layer representation. In this work, the authors presented a complete system for the 2015 Emotion Recognition in the Wild (EmotiW) Challenge, and proved that a hybrid CNN-RNN architecture for a facial expression analysis can outperform a CNN approach using temporal averaging for aggregation.

Another recently proposed architecture that made great progress in dealing with various video analysis tasks is that of deep 3-dimensional CNNs (eg. C3D [24]). 2-dimensional CNNs have a major limitation that they just handle spatial information while neglecting important temporal video information. On the contrary, C3D can simultaneously model appearance and motion information. Fan et al. [10] attempt to hybridize C3D with CNN-LSTM models. According to the proposed system, an RNN takes appearance features extracted by a CNN over individual video frames as input and encodes motion later, while a C3D models appearance and motion of video simultaneously. As a result, this hybrid network gives competitive results for emotion classification.

1.2 Contribution and Outline

Facial expressions play a vital role in social communications between humans. It is therefore unsurprising that automatic FER has become a subject of much recent research. For this Thesis, we present a method for classifying facial expressions from video recordings. Our approach leverages on the recent success of deep CNNs in several recognition problems.

Although effective, deep architectures require massive amounts of labeled data to be trained, and this is not yet available in the emotion recognition literature, especially for video data. Another issue related to data limitations is that existing databases often contain posed expressions, recorded into closed lab environments. As a result, a model trained using these data has very low performance when tested on real-world data. To overcome the above limitations of other FER approaches, in our experiments we make use of transfer learning techniques, using pre-trained CNN architectures. The pre-trained model is then fine-tuned with limited emotion labeled training data to obtain final classification models.

Given the fact that the network has already learned basic information about edges, curves, and shapes from a very large dataset with millions of images and can relate them to a new smaller dataset, fine-tuning can be beneficial for our approach. Moreover, for the construction of our emotion database we combined three different databases including some spontaneous facial expressions in one of them.

The remainder of this paper is organized as follows.

- Chapter 2. In the 2^{nd} chapter all information about emotion recognition databases is provided including many different categories of them. Also, the construction of our dataset as well as the preprocessing steps applied are described.

- Chapter 3. Here we analyze in detail the theory of Neural Networks (NNs). Then, we focus on describing CNNs as a special type of NN used successfully in many recognition tasks.
- Chapter 4. This chapter presents four different versions of the proposed approach as well as the experimental results.
- Chapter 5. Finally in the last chapter we draw some conclusions from the experiments.

Chapter 2
Databases

Despite the powerful feature learning ability of deep learning, problems remain when applied to emotion recognition tasks. Having sufficient labeled training data that include as many variations of the populations and environments as possible is crucial for the design of a deep emotion recognition system. Considering that DNNs require a large amount of training data in order to avoid over-fitting, collecting appropriate data is very important to begin with the construction of a deep learning emotion classifier. However, in the existing facial expression databases there is not sufficient amount of annotated video data to train well-known NNs with deep architectures that achieved the most promising results in other recognition tasks, such as object recognition.

A database gives a consistent platform in order to test and evaluate classification algorithms. The algorithms are usually examined against particular image or video databases. These databases consist of samples and their respective ground truths. There could be some variations in the content of images like different subjects, age, illumination, gender, ethnicities, and textures due to uncontrolled and varying circumstances.

Nevertheless, using a single FER database to construct a classifier is usually not enough to model real-world data variability. In order to prevent bias, one could combine multiple databases or measure cross-dataset performance of the developed framework.

2.1 Categorizations of Databases

Even though there are several collected databases in the literature that fit many specific criteria [25], it is important to note that there are many different aspects that affect the content of the database. The selection of the participants, the elicitation method, and the format of the data, all have a great impact on the performance of the final model [26]. The cultural and social background of participants as well as their mood during the recordings can also sway the results of the database to be specific to a particular group of people.

2.1.1 Elicitation Methods

An important choice to make in gathering data for emotion recognition databases is how to elicit different emotions from the participants. Audiovisual databases reported in the literature can be classified into three major categories: (a) posed, (b) induced, and (c) spontaneous (occurring during an interaction).

With regard to the posed databases, participants are usually professional actors who are asked to correctly express each emotion. Most facial emotion databases, especially the early ones, consist purely of posed facial expressions, as it is the easiest to gather. Some well-known posed databases are CK+ and MMI [1] [27]. However, this kind of databases is the least representative of real world authentic emotions, because forced emotions are often over-exaggerated. In addition, since the emotions are invoked in a lab environment with the supervision of authoritative figures, the subjects might subconsciously keep their expressions in check. For these reasons, human expression analysis models, created through the use of posed databases, often have very poor results when tested with real world data.

In terms of the induced databases, a subject's emotional responses are commonly evoked by films, stories or music. This method of elicitation displays more genuine emotions as the participants are usually subjected to audiovisual media in order to invoke real emotions [2]. Induced emotion databases have become more common due to the limitations of posed databases. The performance of the models in real life is greatly improved, since they are more natural and not hindered by over-emphasised and fake expressions. Nevertheless, as in posed databases, the process of filming the participants is still taking place in an organized lab environment.

Spontaneous emotion datasets are considered to be the closest to actual real-life scenarios [28]. The main drawback of this elicitation method is that the audiovisual data with spontaneous emotions is difficult to collect and label because the emotion expressions are relatively rare, short-lived, and often associated with a complex contextual structure. In addition, true emotions can only be observed, when the person is not aware of being recorded.

2.1.2 Database Types

Emotion recognition databases may come in many different forms, depending on how the data was collected. Existing facial expression databases could be classified into two broad categories:

1. static databases, including static face images along with their labels
2. video databases, consisting of video sequences allowing to consider the temporal information through the video frames.

Static databases are the oldest and most common type. They usually have a large number of participants and a considerably big sample size. While it is relatively easy to find a database suited for the task at hand, categories of emotions are quite limited, as static databases only focus on six primary emotions or smile/neutral detection. Most early facial expression databases only consist of frontal portrait images taken with simple RGB cameras. Newer databases try to design collection methods that incorporate data closer to real life scenarios by using different angles and occlusion (hats, glasses). Good examples are the MMI [27] and Multi-PIE [29] databases, which were some of the first well-known ones using multiple view angles.

In the case of video databases, scientists have tried to benefit from movements, vocal context, or any other type of body language features for emotion recognition. Besides, many video databases have tried to gather subtle emotional changes known as micro-expressions, since they do not show up in still images or are harder to catch. In addition, video is the most convenient format for capturing induced and spontaneous emotions. This is due to a lack of clear start and end points for non-posed emotions. However, video databases tend to be quite small in the number of participants and this is an obstacle when training DNN models.

2.2 Famous Databases for video FER

Some of the most well-known publicly available facial expression databases are described below:

- CK+: The Extended Cohn-Kanade (CK+) [1] database is among the most widely used databases for developing and evaluating algorithms for facial expression analysis. It was presented in 2010 as an extension to the first CK database. It includes 593 posed grayscale video sequences recorded

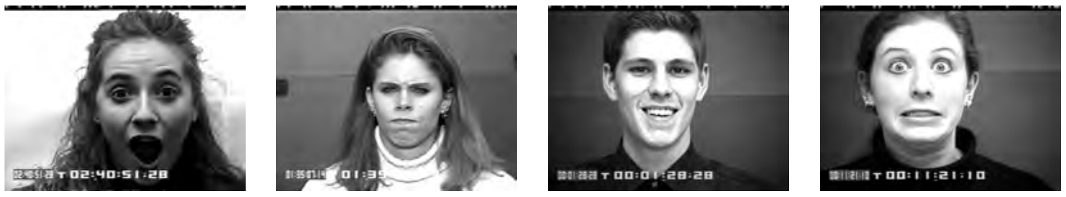

(a) Surprise (b) Anger (c) Happiness (d) Fear

Figure 2.1: Video frames from CK+ database [1].

from 123 subjects ranging from 18 to 30 years old. Each of the sequences contains images from onset (neutral frame) to peak expression (last frame) and varies in duration from 10 to 60 frames. However only 327 of the 593 sequences are finally annotated considering seven discrete emotional categories which are the six basic emotions and 'contempt'.

- MMI: The MMI database [27] contains more than 1500 samples of both static images and video sequences of faces in frontal and in profile view displaying various facial expressions of emotion. It has been developed as a web-based direct-manipulation application, allowing easy access and easy search of the available images. The database includes 19 subjects of both genders ranging in age from 19 to 62, having either a European, Asian, or South American ethnicity. The subjects were asked to display 79 sequences of expressions, including the six prototypical emotions. They were instructed by an expert on how to display the required facial expressions, and they were asked to include a short neutral state at the beginning and at the end of each expression. The static facial expression images are all true color (24-bit) images of 720×576 pixels. There are approximately 600 frontal and 140 dual-view images. Dual-view images combine the frontal and profile views of the face, recorded using a mirror. All video sequences have been recorded at a rate of 24 frames per second. There are approximately 30 profile-view and 750 dual-view facial-expression video sequences. The sequences are of variable length, lasting between 40 and 520 frames [30].

- MUG: The MUG Facial Expression Database consists of RGB videos of 86 subjects performing facial expressions. The database contains 35 women and 51 men, all of Caucasian origin, between 20 and 35 years of age. The subjects were sitting in a chair in front of one camera which records videos at a rate of 19 frames per second. Each image was saved in jpeg format and a size of 896×896 pixels. The MUG database consists of two parts. The first part includes posed facial expressions since the subjects were asked to perform the six basic emotions. Each sequence starts and ends at the neutral state and contains 50 to 160 frames. The goal was to imitate correctly the basic expressions, and, in order to accomplish this, prior to the recordings, a short tutorial about the basic emotions was given to the subjects. The sequences with correct imitation of the expressions were selected to be part of the database. As a result, the number of the available sequences of the first part is 1462. In the second part of the database, the subjects were recorded while they were watching a video that was created in order to induce emotions. The subjects were aware that they were being recorded. The goal of this part of the database was to elicit authentic expressions in the laboratory environment in contrast to the posed ones. The subjects display various emotions and emotional attitudes apart from the six basic emotions. However these image sequences are not labeled yet.

- ISED: The Indian Spontaneous Expression Database for Emotion Recognition (ISED) consists of spontaneous expressions of both male and female participants of Indian origin. The database includes 428 segmented video clips of 50 participants. All the video clips included in the database vary from 1 to 10 seconds in length. The emotions were elicited through passive elicitation by watching emotion inducing videos. Subjects were left alone in the experimental room and a hidden camera recorded their facial expressions. In order to elicit natural emotions, the participants were not informed about the purpose of the experiment beforehand and they were allowed to watch the videos alone in the experimental room. Additionally, in order to avoid suspicion, subjects were allowed to sit at their own ease without any other restrictions. Finally, all videos were annotated by four trained decoding machines to tag the six basic emotional expressions as well as the corresponding emotional intensity for each video on a six-point scale (from 0 to 5).

Table 2.1: Summary of the databases

Database	Elicitation	Emotion Classes	Format	Participants
CK+	posed	6 basic emotions & contempt	video	123
MMI	posed	6 basic emotions	static, video	19
MUG	posed, induced	6 basic emotions	video	86
ISED	spontaneous	Hap, Disg, Sad & Sur	video	50

2.3 Dataset Construction

For this Thesis the CK+, ISED and MUG databases were combined in order to create training, testing, and validation sets with sufficient amount of data. Samples of all three databases can be found in the training, testing, and validation set, but the same participant does not appear in any two of them to avoid bias.

All videos were properly modified so that they all have the following format: every video sequence starts with the frame of neutral facial expression and ends at the most intense frame, in terms of emotion. They were also converted into grayscale videos because of the fact that the CK+ database contains only grayscale data. The structure of the dataset is described in Table 2.2

Table 2.2: Number of videos in every class

No of Videos	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Total
Training Set	82	118	76	164	95	143	678
Testing Set	27	34	27	69	34	40	231
Validation Set	5	4	4	4	4	6	27

As might be seen at the above table, some classes' samples significantly outnumber the samples of others (eg. Happiness vs Fear). This phenomenon is known as class imbalance problem and it appears frequently in practice.

2.4 Data Pre-processing

Various explanations could be given for the role of data pre-processing and its importance, especially in the case of NN input data [31]. Although in a lot of cases, the pre-processing of the data is not needed from the mathematical point of view, it has a significant effect on the performance of NN models, since it improves network training. Moreover, the form of pre-processing applied to the data is a very important factor in determining the success of a practical application using NNs. Hence, choosing the right pre-processing steps for the input data is critical for increasing the network's ability to learn the association between inputs and outputs.

The developed FER framework for this Thesis is based on pre-trained CNN architectures. Particularly our database was used to fine-tune them and make them capable of classifying emotions. The usage of pre-trained architectures requires a specific format for the data in order to meet the standards of the database that was used for the pre-training. This is the reason for applying some of the pre-processing steps below like resizing, conversion to RGB, and normalization.

In order to prepare the database for our CNN model, the following pre-processing steps with this exact order were applied to the videos.

2.4.1 Frame Selection

As mentioned before, all videos of the database have a specific format. According to this format the neutral facial expressions appear in the first frame of every video, while the last frame shows the most intense, in terms of emotion, facial expression. Considering this, not all frames are suitable for the training procedure. Therefore, only the last half frames of every video are chosen to be included in our database.

2.4.2 Face Detection

When an image sequence is presented to a FER system, it is necessary to detect the facial regions as a preliminary preprocessing step. The purpose is to keep only the pixels of the face while disposing of any background pixels. Removing irrelevant information from the frames, helps us to reduce the dimensionality of the input data while keeping the important part of information for training.

There are several methods which can be used to achieve this task. One of the most popular, also used in this work, is the so-called Viola–Jones face detector [32]. The algorithm consists of 3 major steps, the image integral, the classifier learning with AdaBoost, and the attentional cascade structure.

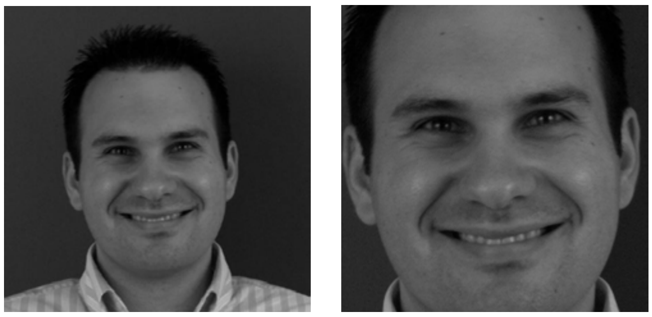


Figure 2.2: Video frame from the MUG database [2] before and after cropping by Viola-Jones face detection.

The first step of the Viola-Jones face detection algorithm is to turn the input image into an integral image. The integral image, also known as a summed area table, is a way for quickly and efficiently computing the sum of values in a rectangle subset of a pixel grid. The integral image at location x, y contains the sum of the pixels above and to the left of x, y as seen in (2.1) :

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y') \tag{2.1}$$

where $i(x,y)$ is the pixel value of the original image and $ii(x,y)$ is the corresponding image integral value. In this way, the sum of pixels of any image region can be calculated with only four values from integral images $A(x,y)$, $B(x,y)$, $C(x,y)$, $D(x,y)$:

$$\sum_{(x,y) \in ABCD} i(x,y) = ii(D) + ii(A) - ii(B) - ii(C) \tag{2.2}$$

Then, for feature extraction Haar-like features are used (Figure 2.3). A Haar-like feature value is calculated by the pixel sum of the darker rectangle minus the pixel sum of the lighter rectangle.

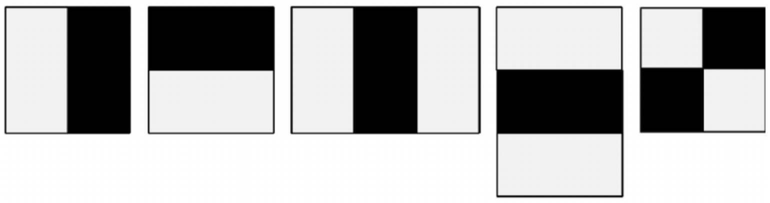


Figure 2.3: Five Haar-like features. Image from [3]

Given a feature set and a training set of positive and negative images, an ML approach could be used to learn a classification function. The Viola Jones uses a variant of AdaBoost to both select a small set of features and train the classifier. A single AdaBoost classifier consists of a weighted sum of many weak classifiers, where each weak classifier is a threshold on a single Haar-like rectangular feature.

Finally, the cascaded classifier is composed of stages each containing a strong classifier from AdaBoost. The job of each stage is to determine whether a given sub-window is definitely "not a face" or "maybe a face". When a sub-window is classified to be a non-face by a given stage it is immediately discarded. Conversely, a sub-window classified as "a maybe face" is passed on to the next stage in the cascade. It follows that the more stages a given sub-window passes, the higher the chance the sub-window contains a face.

2.4.3 Resizing and Linear Scaling

The architecture of the pre-trained model which was used in this Thesis requires a fixed input size of 224×224 features. Therefore, after face cropping, all frames were resized to 224×224 pixels. Moreover, grayscale image pixels are commonly integers in a range from 0 to 255 where 0 corresponds to black and 255 to white. However, in NN architectures data must be scaled into the range used by the input neurons of the NN. This is typically the range of -1 to 1 or 0 to 1. For that reason, a simple linear scaling was applied to the frames' pixels in order to convert them into a range of 0 to 1.

2.4.4 Histogram Equalization

The Histogram Equalization algorithm enhances the contrast of images by transforming the intensity values in an image. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

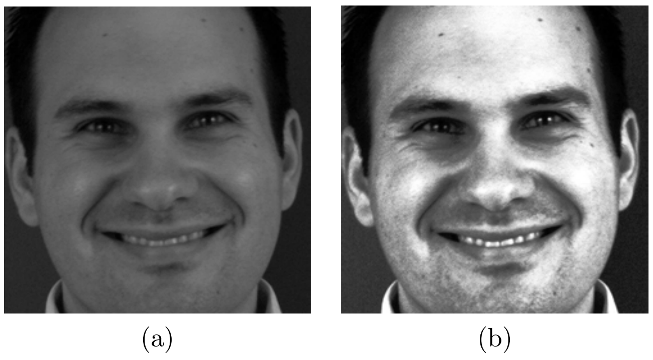


Figure 2.4: Video frame from the MUG database [2] before and after applying Histogram Equalization.

Here are the steps for implementing the Histogram Equalization algorithm.

1. Create the histogram for the image. Consider a discrete grayscale image x and let n_i be the number of occurrences of gray level i. The probability of an occurrence of a pixel of level i in the image is

$$p_x(i) = \frac{number\ of\ pixels\ with\ intensity\ i}{total\ number\ of\ pixels} = \frac{n_i}{n}, \quad 0 \leq i < L$$

with L being the total number of gray levels in the image (typically 256), n being the total number of pixels in the image, and $p_x(i)$ being in fact the image histogram for pixel value i, normalized to $[0, 1]$.

2. Calculate the cumulative distribution function (*cdf*) histogram. Let us define the *cdf* corresponding to p_x as:

   $$cdf_x(i) = \sum_{j=0}^{i} p_x(j)$$

   which is also the image accumulated normalized histogram.

3. Calculate the *cdf* of the new image y. We would like to create a transformation of the form $y = T(x)$ to produce the new image, with a flat histogram. Such an image would have a linearized *cdf* across the value range, i.e. $cdf_y(i) = iK$ for some constant K. The properties of the *cdf* allow us to perform such a transform which is defined as:

   $cdf_y(y) = cdf_y(T(k)) = cdf_x(k)$ where k is in the range $[0, L]$.

4. Assign new values for each gray value in the image. Notice that T maps the levels into the range $[0, 1]$, since we used a normalized histogram of x. In order to map the values back into their original range, the following simple transformation needs to be applied on the result to get the final image version y':

   $y' = y \cdot (\max\{x\} - \min\{x\}) + \min\{x\}$

2.4.5 Conversion to RGB and Normalization

The selected CNN architecture expects 3-channel RGB images. Therefore, after applying all previous pre-processing steps to grayscale video frames, it is necessary to convert them into 3-channel RGB format. This is obviously not possible, so the only simple solution found was to copy the value of every pixel into the other two channels in order to create 'fake' RGB images.

In addition, when using pre-trained CNN architectures, the data should be normalized in the same way as the data used for pre-training the network. Each image channel is normalized using the mean and standard deviation as described below.

$$input(channel) = \frac{input(channel) - mean(channel)}{std(channel)} \tag{2.3}$$

where $mean = [0.485, 0.456, 0.406]$ and $std = [0.229, 0.224, 0.225]$.

Chapter 3

Theory of Neural Networks

The field of NNs was originally inspired by the goal of modeling biological neural systems, but has since diverged and become a matter of engineering while achieving good results in ML tasks. The basic unit of the brain is a neuron. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately $10^{14} - 10^{15}$ synapses. Each neuron receives input signals from its dendrites and produces output signals along its single axon. The axon connects, via the synapses, the output of one neuron to dendrites of other neurons.

In the computational model of an artificial neuron, observed in Figure 3.1, the signals that travel along the axons (x_i) interact multiplicatively ($w_i * x_i$) with the dendrites of the other neuron based on the synaptic strength at that synapse (weight w_i). In the basic model, every neuron creates a weighted sum $\sum_i w_i x_i + b$ where b is the bias value of that neuron. NN technology started by the idea that the synaptic strengths, in other words the weights w and the biases b, are learnable and control the strength of influence of one neuron on another. If the final sum is above a certain threshold, the neuron can fire, sending a spike

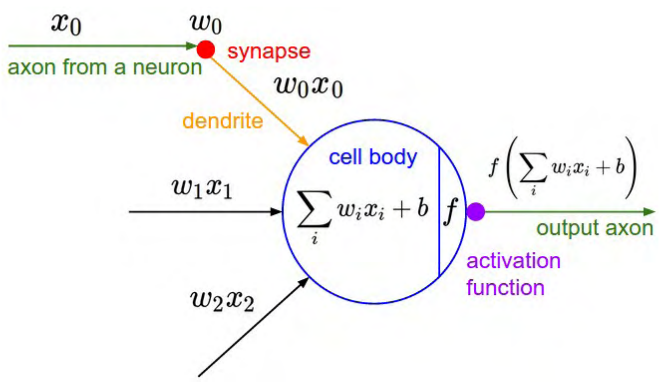


Figure 3.1: Schematic of a neuron model. Image from [4]

along its axon. In the computational model, we assume that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. Based on this, we need to model the firing rate of the neuron with an activation function, which represents the frequency of the spikes along the axon.

Each neuron, which from now on will be referred as node, performs a dot product with the input and its weights, adds the bias and applies the activation function.

3.1 How does a neural network work?

NNs are typically organized in layers. Layers are made up of a number of interconnected nodes which contain an activation function that is also referred to as 'non-linearity'. Patterns are presented to the network via the input layer, which communicates to one or more hidden layers where the actual processing is done by a system of weighted connections. Unlike all layers in an NN, the output layer nodes most commonly do not have an activation function. This is because the last output layer usually represents the class scores in classification problems, which are arbitrary real-valued numbers, or some kind of real-valued target in regression problems.

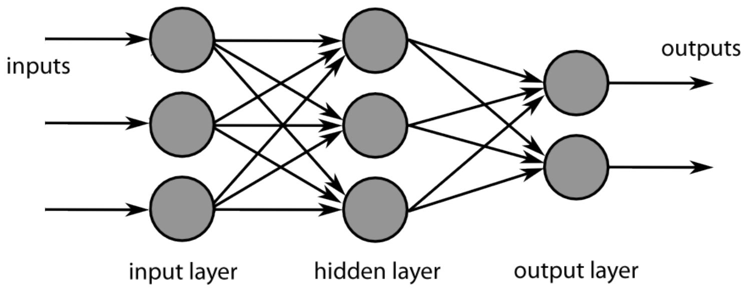


Figure 3.2: NN model with one hidden layer of three nodes, one output layer with two nodes, and three inputs.

Information flows through an NN in two ways. When the network is learning (training phase) or operating normally (testing phase), patterns of information are fed into the network via the input nodes, which trigger the layers of hidden nodes, and these in turn arrive at the output layer. This is the common design of a feedforward network. Not all nodes 'fire' all the time. Each node receives inputs from the nodes to its left, and the inputs are multiplied by the weights of the connections while travelling along the network. Every node adds up all the inputs it receives in this way and (in the simplest type of network) if the sum is more than a certain threshold value, the node 'fires' and triggers the units it is connected to, on its right.

For an NN to learn, there has to be an element of feedback involved, just as children learn by being told what they are doing right or wrong. As humans, after an action we compare the outcome we wanted with what actually happened, figure out the difference between the two, and use that to change our action next time. NNs learn things in exactly the same way, typically by a feedback process called backpropagation (BP). This involves comparing the output a network produces with the output it was meant to produce, and using the difference between them to modify the weights of the connections between the nodes in the network. To achieve this, we work backwards, from the output through the hidden nodes to the input nodes. In time, BP causes the network to learn, reducing the difference between actual and intended output.

During the training process, the entire training set is passed forwards and backwards through the network several times. A complete pass over the whole training set is called an epoch. Considering the large memory requirements of forward and backward propagation of the entire dataset at once, we divide the data into smaller batches to feed the network. Note that the training phase completes after many epochs.

Once the network has been trained with enough learning examples, it reaches a point where we can present it with an entirely new set of inputs it has never seen before and see how it responds. This process is called testing phase. Depending on how the network is trained, it will attempt to categorize new unseen samples in one of the classes, generalizing on the basis of its past experience.

3.2 Activation functions

Every activation function, also known as non-linearity, takes a single number and performs a certain fixed mathematical operation on it. Some of the most commonly used activation functions are the sigmoid, tanh, and ReLU activation functions, and they are described below:

- Sigmoid

  The sigmoid activation function has the mathematical form:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

The function takes a real-valued number and 'squashes' it into the range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1). In practice, the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used since it has two major drawbacks.

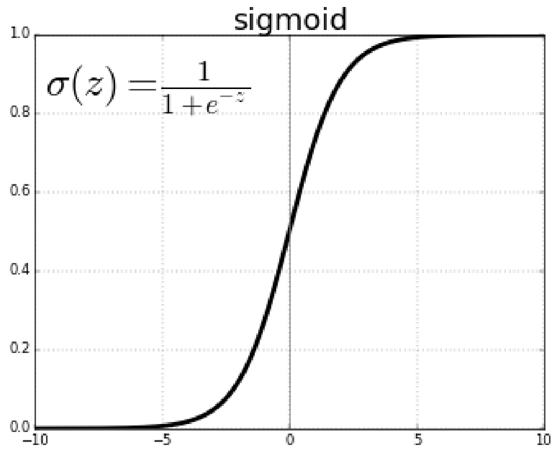


Figure 3.3: Sigmoid non-linearity squashes real numbers to the range [0,1]

The first undesirable property of the sigmoid neuron is that when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Considering that during BP, this local gradient will be multiplied to the gradient of this gate's output for the whole objective, if the local gradient is very small, it will effectively "kill" the gradient and almost no signal will flow through the neuron to its weights and recursively to its data. Additionally, one must pay extra caution when initializing the weights of sigmoid neurons to prevent saturation. For example, if the initial weights are too large then most neurons would become saturated and the network will barely learn.

Moreover, sigmoid outputs are not zero-centered. This is also an undesirable feature since neurons in later layers of processing in an NN would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive (e.g. $x > 0$ elementwise in $f = w^T x + b$), then the gradient on the weights w will during BP become either all positive, or all negative (depending on the gradient of the whole expression f). This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights. However, note that data are fed into the network grouped in batches. Once these gradients are added up across a batch of data the final update for the weights can have variable signs, somewhat mitigating this issue. Therefore, this is an inconvenience but it has less severe consequences compared to the saturated activation problem above.

- *Tanh*

  The *tanh* non-linearity, which is shown in Figure 3.4, squashes a real-valued number to the range [-1, 1]. Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. Therefore, in practice the *tanh* non-linearity is always preferred to the sigmoid non-linearity.

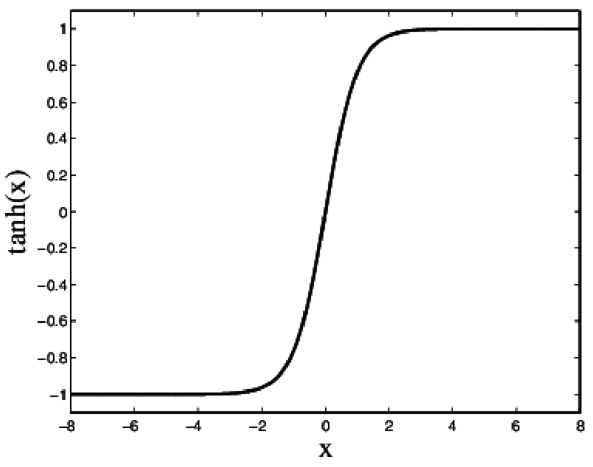


Figure 3.4: The tanh non-linearity squashes real numbers to range between [-1,1].

  Also note that the *tanh* neuron is simply a scaled sigmoid neuron, in particular: $tanh(x) = 2\sigma(2x) - 1$.

- ReLU

  The Rectified Linear Unit (ReLU) has become very popular in the last few years [33]. It computes the function $f(x) = max(0, x)$, so the activation is simply thresholded at zero. ReLU was found to greatly accelerate the convergence in the training phase compared to the sigmoid and tanh functions. It is argued that this is due to its linear, non-saturating form. In addition, compared to tanh or sigmoid neurons that involve expensive operations, e.g. exponentials, the ReLU can be implemented by simply thresholding a

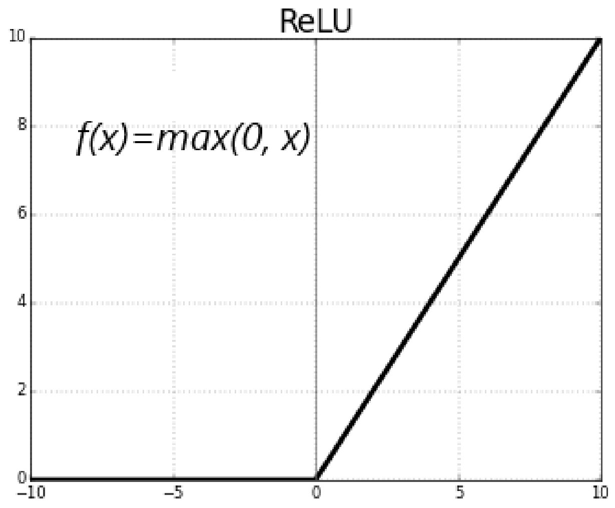


Figure 3.5: Rectified Linear Unit (ReLU) activation function is zero when $x < 0$ and then linear with slope 1 when $x > 0$.

matrix of activations at zero. The major drawback of ReLU is that units can be fragile during training and can 'die'. If this happens, then the gradient flowing through the specific neuron will forever be zero from that point on. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate again. If this happens, then the gradient flowing through the unit will forever be zero from that point on.

3.3 Backpropagation

This numerical method has been used by different research communities in different contexts. The algorithm of BP was first published in 1970 by Seppo Linnainmaa, although important concepts of it were known even earlier. Later, in 1986 David Rumelhart showed experimentally that this method can generate useful internal representations of incoming data in hidden layers of NNs [34]. During the 2000s the algorithm fell out of favour, but returned in the 2010s, benefitting from the development of cheap and powerful GPU-based computing systems. It has been one of the most studied and used algorithms for NN learning ever since.

The algorithm of BP can be decomposed and briefly described in the following four steps:

1. Feed-forward computation

2. Back propagation to the output layer
3. Back propagation to the hidden layer
4. Weight updates

The process stops when the value of the error function in the output layer has become sufficiently small. The algorithm of BP is described in detail below.

Given a feed-forward network with n input and m output units, it can consist of any number of hidden units and can exhibit any desired feed-forward connection pattern. We are also given a training set $(x_1, t_1), \cdots, (x_p, t_p)$ consisting of p ordered pairs of n and m-dimensional vectors, which are the input and output patterns. When the input pattern x_i from the training set is presented to this network, it produces an output o_i different in general from the target t_i. The goal is to make o_i and t_i identical for $i = 1, \cdots, p$, by using a learning algorithm. More precisely, we want to minimize the error function of the network, defined as:

$$E = \frac{1}{2} \sum_{i=1}^{p} \|o_i - t_i\|^2 \tag{3.2}$$

The BP algorithm is used along with an optimizer to find a local minimum of the error function or in other words the combination of weight values that minimize the error function of the network. The network is initialized with randomly chosen weights. Moreover the gradient of the error function is computed and used to correct the initial weights. Our task is to compute this gradient recursively.

The first step of the minimization process consists of extending the network, so that it computes the error function automatically. Every one of the j output units of the network is connected to a node which evaluates the function $E = \frac{1}{2}\|o_i - t_i\|^2$, where o_{ij} and t_{ij} denote the j^{th} component of the output vector o_i and of the target t_i. The outputs of the additional m nodes are collected at a node which adds them up and gives the sum E_i as its output. The same network extension has to be built for each pattern t_i. A computing unit collects all quadratic errors and outputs their sum $E_1 + \cdots + E_p$. The output of this extended network is the error function E. So the network is capable of calculating the total error for a given training set. The weights in the network are the only parameters that can be modified to make the quadratic error E as low as possible. Because E is calculated by the extended network exclusively through composition of the node functions, it is a continuous and differentiable function of the ℓ weights $w_1, w_2, \cdots, w_\ell$ in the network. We can thus minimize E by using an iterative process of gradient descent, for which we need to calculate the gradient:

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \cdots, \frac{\partial E}{\partial w_l}) \tag{3.3}$$

Each weight is updated using the increment

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i}, \; for \;\; i = 1, \cdots, \ell \tag{3.4}$$

where γ represents the learning rate, a proportionality parameter which defines the step length of each iteration in the negative gradient direction.

With this extension of the original network the whole learning problem now reduces to the question of calculating the gradient of a network function with respect to its weights. Once we calculate the gradient, we can adjust the network weights iteratively. In this way we expect to find a minimum of the error function, where $\nabla E = 0$.

Obviously we can use any optimization technique that modifies the internal weights of the network in order to minimize the total error function that we previously defined. Some of the most well-known optimization algorithms are described in the next section.

3.4 Optimization Algorithms

Optimization algorithms help us to minimize (or maximize) the error function $E(x)$ which is also called 'objective function'. The weights w and the bias b values of the NN are its internal learnable parameters which are used in computing the output values and are learned and updated in the direction of optimal solution. The internal parameters of a model are vital for efficiently and effectively training it and producing accurate results. This is why we use various optimization strategies and algorithms to update and calculate appropriate and optimum values of such model parameters. The selection of optimization algorithm greatly influences our model's learning process and outputs. Some of the most common optimization algorithms are described below.

- Gradient Descent

  Gradient Descent is the most important technique and the foundation of NN optimization theory. The formula of the parameter updates is

$$\theta = \theta - \gamma \nabla J(\theta) \tag{3.5}$$

where γ represents the learning rate and $\nabla J(\theta)$ is the gradient of loss function $J(\theta)$ with respect to the parameters θ.

It is the most popular optimization algorithm used for training an NN, update and tune the model's parameters in a direction so that the loss function gets minimized.

Considering the training procedure, during BP we first propagate forwards, calculate the dot product of input signals and their corresponding weights and then apply an activation function, which enables the model to learn arbitrary functional mappings. After this, we propagate backwards in the network. At this step an optimizer, in this case gradient descent, has to be used. Particularly, error terms are carried backwards and the weight values are updated using gradient descent.

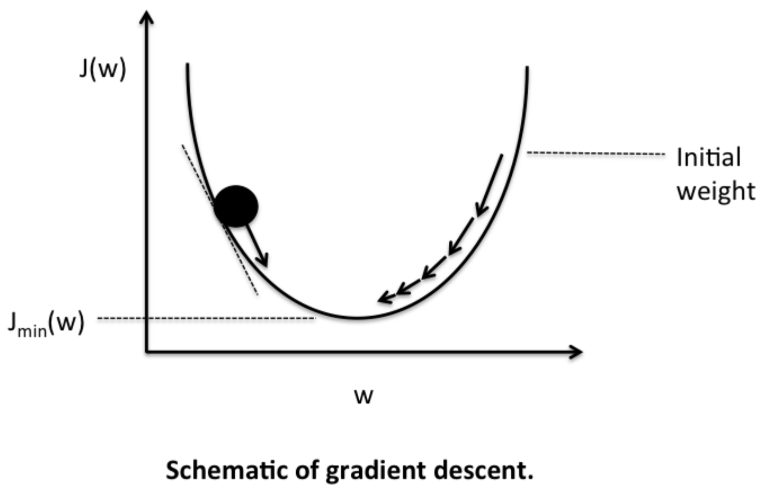


Figure 3.6: Schematic of gradient descent.

Figure 3.6 shows the process of weight updates in the opposite direction of the gradient. The U-shaped curve is the gradient slope. As one can notice, if the weight values are too small or too large then the errors increase significantly. Therefore the updates should be neither too small nor too large , in order to descent downwards opposite to the gradient until we find a local minima.

Apart from the simple version of the gradient descent algorithm, several variations of it appeared later on. Stochastic gradient descent (SGD), mini-batch gradient descent and Nesterov accelerated gradient are some of them.

- Adagrad

  The Adagrad optimizer allows the learning rate γ to adapt based on the parameters. In fact, it applies large updates for infrequent parameters and small updates for frequent parameters. For this reason, it is a well-suited optimizer for dealing with sparse data.

  According to Adagrad, a different learning rate is used for every parameter θ at every time step based on the past gradients which were computed for that parameter. In gradient descent there was an update for all parameters θ at once as every parameter θ_i used the same learning rate, whereas Adagrad uses a different learning rate for every parameter $\theta_{t,i}$ at every time step t. The formula of Adagrad for parameter updates is given below,

  $$\theta_{t+1,i} = \theta_{t,i} - \frac{\gamma}{\sqrt{G_{t,ii} + e}} g_{t,i} \tag{3.6}$$

  where γ represents the learning rate and $g_{t,i}$ is the gradient of the loss function with respect to the parameter θ_i at time step t. G_t is a diagonal matrix where each diagonal element at the (i, i) position is the sum of the squares of the gradients with respect to θ_i up to time step t and ϵ is a smoothing term that avoids division by zero.

  One of Adagrad's main benefits is that it eliminates the need to manually tune the learning rate. Most implementations use a default value of 0.01 and leave it at that. However its main weakness is its accumulation of the squared gradients in the denominator. Since every added term is positive, the accumulated sum keeps growing during training. This in turn causes the learning rate to shrink and eventually become significantly small, at which point the algorithm is no longer able to acquire additional knowledge. The following algorithms aim to resolve this flaw.

- Adadelta

  Adadelta [35] is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to a fixed size.

  Instead of inefficiently storing previous squared gradients, the sum of gradients is recursively defined as a decaying average of all past squared gradients. The running average $E(g^2)_t$ at time step t then depends only on the previous average and the current gradient.

  $$E(g^2)_t = aE(g^2)_{t-1} + (1 - a)g_t^2 \tag{3.7}$$

A typical value for a is around 0.9. For clarity, gradient descent update is written in terms of the parameter update vector.

$$\Delta\theta_t = -\gamma g_{t,i} \tag{3.8}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t \tag{3.9}$$

The parameter update vector of Adagrad that we derived previously thus takes the form:

$$\Delta\theta_t = -\frac{\gamma}{\sqrt{G_t + \epsilon}} \odot g_t \tag{3.10}$$

where $\odot$ is the symbol of the symmetric product [36].

We now simply replace the diagonal matrix G_t with the decaying average over past squared gradients $E(g^2)_t$

$$\Delta\theta_t = -\frac{\gamma}{\sqrt{E(g^2)_t + \epsilon}} g_t \tag{3.11}$$

- Adam

Adaptive Moment Estimation (Adam) [37] is another optimizing method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface. In 3.12 and 3.13 m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively and they are calculated as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.12}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3.13}$$

As m_t and v_t are initialized as vectors of zeros, the authors of Adam observed that the algorithm is biased towards zero, especially during the initial time steps, and especially when the decay rates are small (β_1 and β_2 are close to 1).

They counteracted these biases by computing bias-corrected first and second moment estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3.14}$$

$$\hat{\upsilon}_t = \frac{\upsilon_t}{1-\beta_2^t} \tag{3.15}$$

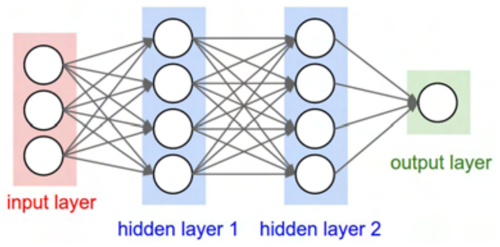
They then use these to update the parameters just like in Adadelta, which yields the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{\hat{\upsilon}_t} + \epsilon} \tag{3.16}$$

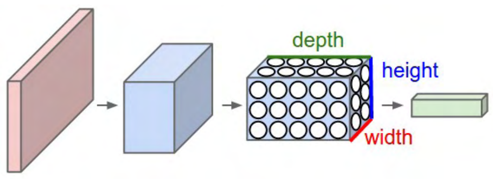
The authors propose default values of 0.9 for $\beta 1$, 0.999 for $\beta 2$ and 10^{-8} for ϵ. It has been empirically proved that Adam works well in practice and compares favorably to other adaptive learning algorithms.

3.5 Convolutional Neural Networks

CNNs are very similar to ordinary NNs as they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. CNNs usually contain some ordinary fully connected (FC) layers at the end of the model. They also still have a loss function on the last FC layer. The main difference is that rather than learning unstructured weights, CNN architectures apply filters using convolution where the internal values of the filters are the weights to be learned. In the case of 2D convolutional filters, there is the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture.



(a) NN



(b) CNN

Figure 3.7: A regular 3-layer NN on the left and on the right a CNN which arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Images from [4]

CNNs take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular

NN, the layers of a CNN have neurons arranged in 3 dimensions: width, height and depth (Figure 3.7). Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full NN, which is the total number of layers in a network. The neurons in a convolutional layer are only connected to a small region of the layer before it, instead of all of the neurons in an FC layer. Moreover, by the end of a CNN architecture the full image will be reduced into a single vector of class scores, arranged along the depth dimension.

A simple CNN is a sequence of layers, where every layer transforms one volume of activations to another through a differentiable function. Three types of layers are mainly used to build a CNN architecture: convolutional layers, pooling layers, and FC layers, and they are analyzed below.

3.5.1 FC Layer

Neurons in a FC layer have full connections to all activations in the previous layer, as seen in regular NNs. It is the most simple layer type, and it is usually seen at the last layers of CNN architectures, in order to perform the intended classification or regression task.

3.5.2 Convolutional Layer

The convolutional layer is the core building block of a CNN that does most of the computational heavy lifting. The convolutional layer parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a CNN might have size 5x5x3 (i.e. 5 pixels width and height, and 3 for RGB input images with 3 color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume, a 2-dimensional activation map that gives the responses of that filter at every spatial position is produced. Intuitively, the network will learn filters that activate when they see some type of visual feature, such as an edge of some orientation, or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of filters in each convolutional layer, and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

In order to control how many nodes there are in the output volume of a

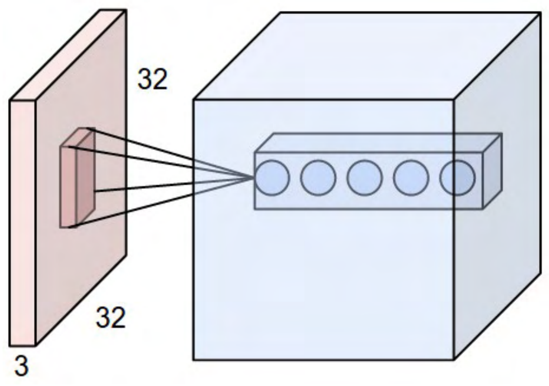


Figure 3.8: Schematic convolutional layer. Image from [4]

convolutional layer, three hyperparameters have to be defined: the depth, stride and zero-padding.

First, the depth of the output volume corresponds to the number of filters we would like to use, each learning to look for something different in the input. For example, if the first convolutional layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a depth column.

Second, we must specify the stride with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time during convolution. When the stride is 2 (or uncommonly 3 or more) then the filters jump 2 pixels at a time as we slide them around. Bigger strides produce smaller output volumes spatially.

Sometimes it is convenient to pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter. The purpose of applying zero padding is to better control the spatial size of the output volumes. In this way we can exactly preserve the spatial size of the input volume so that the input and output width and height are the same.

We can compute the spatial size of the output volume (O) as a function of the input volume size (W), the receptive field size of the convolutional layer neurons (F), the stride with which they are applied (S), and the amount of zero padding used (P) on the border.

$$O = \frac{W - F + 2P}{S} + 1 \tag{3.17}$$

For example for a 7×7 pixel input image and a 3×3 filter with stride 1 and pad 0 we would get a 5×5 output. With stride 2 we would get a 3×3 output.

3.5.3 Pooling Layer

When constructing a CNN architecture, it is common to periodically insert a pooling layer in-between successive convolutional layers. Its function is to progressively reduce the spatial size of the representation, to reduce the amount of parameters and computation in the network, and hence to also control overfitting. A pooling layer operates independently on every depth slice of the input and resizes it spatially, using (in the case of max-pooling) the MAX operation. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2. It downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2×2 region in some depth slice). The depth dimension remains unchanged.

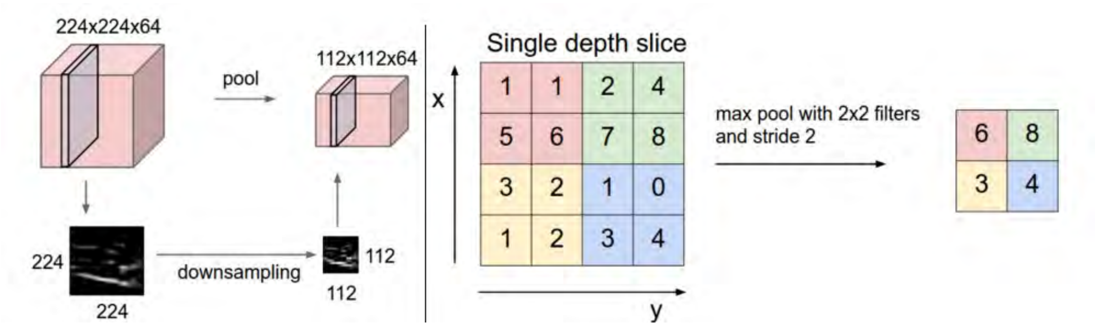


Figure 3.9: Left: The input volume of size $224 \times 224 \times 64$ is pooled with filter size 2 and stride 2 into output volume of size $112 \times 112 \times 64$. Note that the volume depth is preserved. Right: Max pooling, here shown with a stride of 2. Each max is taken over 4 numbers (little 2×2 square regions). Image from [4]

In addition to max pooling, the pooling units can also perform other functions, such as average pooling or even L2-norm pooling. However max-pooling is most commonly used.

3.5.4 Famous CNN architectures

Over recent decades, various CNN architectures have been developed and achieved great results for recognition tasks, and specifically for the ImageNet. The ImageNet project is a large visual database consisting of 1.4M images, designed for use in visual object recognition research. This project also runs an annual contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where different systems compete to correctly classify and detect objects and scenes. Some of the most popular CNN architectures of ILSVRC top competitors and their descriptions are summarized below.

- AlexNet. The first work that popularized CNNs in Computer Vision was AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton [5]. AlexNet was submitted to the ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). The network had a very similar architecture to LeNet [38], but was deeper, bigger, and featured convolutional layers stacked on top of each other. It consisted of $11 \times 11, 5 \times 5$, and 3×3 convolutional filters, max pooling, dropout, data augmentation and SGD with momentum. It also attached ReLU activations after every convolutional and FC layer.

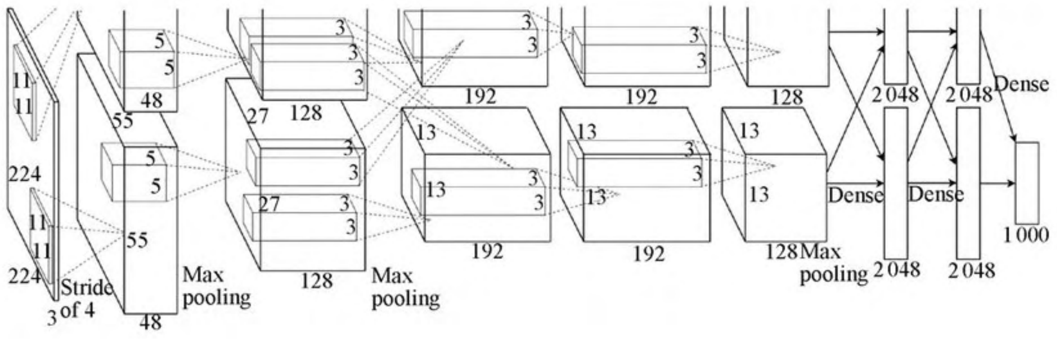


Figure 3.10: AlexNet architecture (distributed in 2 GPUs). Image from [5]

- ZFNet. The ILSVRC 2013 winner was a CNN architecture from Matthew Zeiler and Rob Fergus which is known as the ZFNet [39]. ZFNet was an improvement on AlexNet by tweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.

- GoogLeNet. The ILSVRC 2014 winner was an architecture proposed by Szegedy et al. from Google [6]. Its main contribution was the development of a model with 22 layers that dramatically reduced the number of parameters in the network to 4 million, compared to AlexNet with 60 million. Additionally, this paper uses average pooling instead of FC layers at the top of the CNN, eliminating a large amount of parameters that do not seem

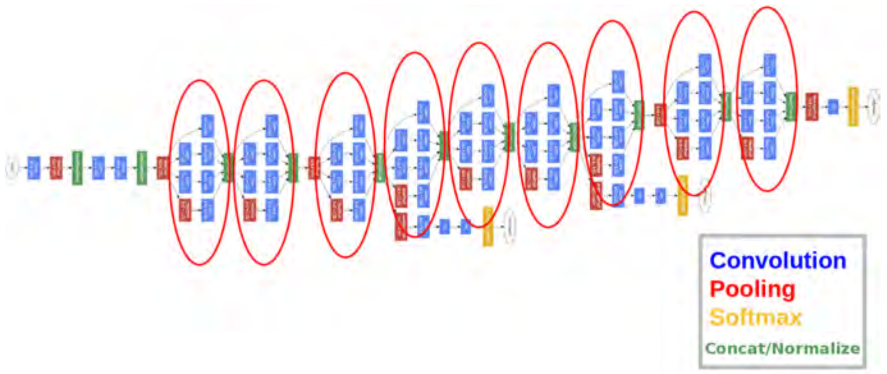


Figure 3.11: GoogLeNet architecture. Image from [6]

to matter much. There are also several followup versions to the GoogLeNet, most recently Inception-v4. [40].

- VGGNet. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet [41]. Its main contribution was in showing that the depth of the network is a

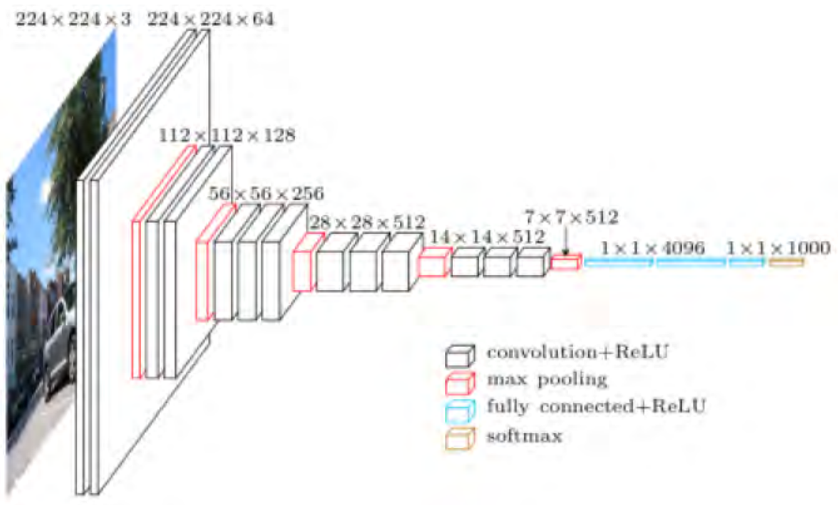


Figure 3.12: VGG16 architecture. Image from [7]

critical component for good performance. They developed several versions of VGGNet with different depth sizes. Their final best network contains 16 convolutional and FC layers and features an extremely homogeneous architecture that only performs 3×3 convolutions and 2×2 pooling from the beginning to the end (Figure 3.12). The model of VGG16 is also used for this Thesis experiments. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot of memory and 140 million of parameters.

- ResNet. Residual Network (ResNet) developed by Kaiming He et al. was the winner of ILSVRC 2015. They introduced a new architecture with 'skip connections' and featured heavy use of batch normalization. Such skip connections are also known as gated recurrent units and have a strong similarity to elements used in RNNs. The architecture is also missing FC layers at the end of the network. Thanks to this technique they were able to train an NN with 152 layers while still having lower complexity than VGGNet.

3.6 Overfitting

Generalization is a very important attribute in ML problems. The goal of a good ML model is to generalize well from the training data to any unseen data from the problem domain. This allows us to make predictions on data the model has never seen before.

Overfitting is the main cause for poor performance of ML algorithms. It refers to a model that models the training data too well. In other words, the model learns the detail and noise in the training data to the extent that it negatively impacts its performance on new data. This means that noise or random fluctuations in the training data are picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize.

A good way to measure the appearance of overfitting to our model is to use a small group of unseen data, named 'validation set', in order to test the model during the training procedure. The gap between the training and validation accuracy indicates the amount of overfitting (Figure 3.13). The blue validation error curve shows very small validation accuracy compared to the training accuracy, indicating strong overfitting. Note that, it is possible for the validation accuracy to even start to decrease after some point.

The main steps for reducing overfitting are:

- More data - Data augmentation: The first step is of course to collect more data. However, in most cases this is not possible. So the next step is data augmentation, something that is always recommended to use. Data augmentation includes methods like random rotation of the image, cropping,

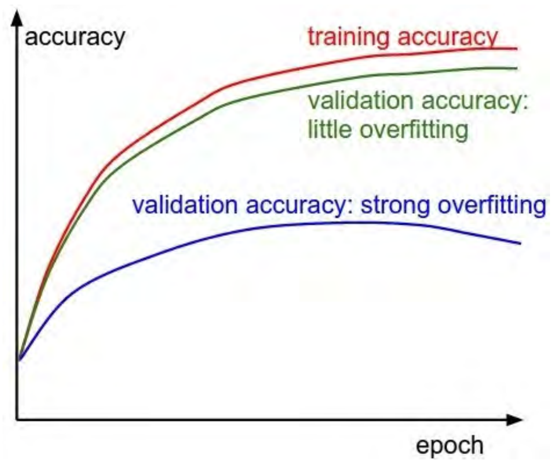


Figure 3.13: Overfitting occurence. Image from [4]

flipping, adding a color filter, etc. These techniques are only applied to the training data and not on the validation or testing set.

- Regularization (Dropout): Another important technique is regularization. Dropout is a common method for regularization. It ignores a randomly chosen sample of the activations (makes them zero) during each training epoch. In the VGG model this is only applied to the FC layers at the end of the model. However, it can also be applied to the convolutional layers. With dropout, only a part of the network is allowed to learn during every epoch. By randomly dropping out neurons, the procedure prevents any neuron from relying excessively on the output of any other neuron, forcing it instead to rely on the population behavior of its inputs. This method helps the model to generalize better to unseen data and prevents overfitting since it reduces interdependent learning amongst the neurons.

- Reduction of architecture complexity: The third and final option is to reduce the network's complexity, by decreasing the number of the network's parameters. This can be achieved by reducing either the number of layers or the layer size. Using a large number of parameters, while having a limited training data size, may force the model to map undesired properties of the data (noise). On the other extreme, if we have a significant small number of parameters, the model will possibly not be able to model complex problems. Actually, the more parameters we have, the more data we will need for training properly. So under limited data, decreasing the number of parameters can significantly improve the ability of the model to learn and generalize to unseen data.

3.7 Transfer Learning for CNNs

In practice, it is not common to train an entire CNN from scratch (with random weights initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a CNN on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use it either as an initialization or a fixed feature extractor for the task of interest. The two major transfer learning scenarios look as follows:

- CNN as fixed feature extractor

  One can use a pre-trained CNN as a feature extractor by removing the last FC layer (this layer's output size is equal to the number of classes), and then treating the rest of the network as a fixed feature extractor for the new dataset. In an AlexNet, this would compute a 4096-D feature vector for every image that contains the activations of the hidden layer immediately before the classifier. Once we extract the 4096-D feature vectors for all images, any linear classifier can be used (e.g. Linear SVM) for the new dataset.

- Fine-tuning a CNN

  The second strategy is to not only replace and retrain the classifier on top of the CNN on the new dataset, but to also fine-tune the weights of the pre-trained network by continuing the BP. It is possible to fine-tune all the layers, or to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a CNN contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the architecture become progressively more specific to the details of the classes contained in the original dataset.

The most important factors for deciding what type of transfer learning to use are the size of the new dataset, its similarity to the original dataset and the complexity of the problem.

Chapter 4
Proposed Method

In this chapter we present our framework for automatic FER from video recordings. The proposed framework is based on applying transfer learning (fine-tuning) to a pretrained CNN architecture and combining per frame predictions for classifying the predominant emotion in a video sequence. Transfer learning is the concept of transferring the weights of an already trained model (pre-trained) to another problem set on a different dataset. In other words, after loading weights of an architecture trained on a very large dataset, we can initialize the model with those trained weights and then fine-tune the network on a new smaller dataset, in our case: the emotion recognition dataset. The fine-tuned model provides information for the facial expression of each video frame. Then, during the testing procedure, per-frame information of all frames is combined in order to obtain the final prediction for the whole video. Several different implementations of the framework are presented and compared in the next sections.

The dataset used for creating the training, testing and validation sets is a combination of the CK+, ISED, and MUG databases. More information about the dataset and the preprocessing steps can be found in Chapter 2.

The open source deep learning platform of PyTorch was utilized for the development of this framework. On a system equipped with an NVIDIA GeForce GTX 950 GPU, training process lasted approximately 2 days, depending on the architecture. Note that as the pretained CNN, we use the VGG16 architecture. VGG16 was selected because of its simple architecture and the fact that its pre-trained version is publicly available.

4.1 VGG16 architecture

VGG16 architecture (Figure 3.12) is mainly composed of a sequence of convolutional, max-pooling, and FC layers, and it was first trained to classify objects among 1000 classes. A stack of convolutional layers is followed by three FC layers: the first two have 4096 nodes each, the third performs 1000-way classification and thus contains 1000 nodes (one for each class). The final layer is the soft-max layer which gives the probability of each class to be predicted.

The image is passed through a stack of convolutional layers, with filters of small receptive field: 3×3. The stride is fixed to 1 pixel and the spatial padding of each convolutional layer input is 1 pixel so that the spatial resolution is preserved after applying 3×3 convolutional filters. Moreover, spatial pooling is carried out by five max-pooling layers, which follow some of the convolutional layers. Max-pooling is performed over a 2×2 pixel window, with a stride of 2. All hidden layers are equipped with the ReLU non-linearity.

4.2 Fine-tuning VGG16

The model has been first trained on the large public image repository of ImageNet. This database was created for object recognition challenges. It includes images of 1000 classes, and it is split into three sets: training (1.3M images), validation (50K images), and testing set (100K images). Hence, rather than using randomly initialized weights, we use the pre-trained model as a base model for our training. This is a better starting point than the random weights since it allows us to reuse low-level visual information that the network has already learned in the initial task. The key concept is that after fine-tuning on our dataset, the model will be able to classify emotions from facial expressions, accurately.

After loading the pre-trained model, we first need to change the size of the last FC layer from 1000 to 6, since this is the number of emotion classes in our problem (Table 4.1). This last FC layer is then initialized with random weights. It should be noted that the fine-tuning step is not applied to the whole network. Generally in CNNs, the first few layers learn mostly about the edges and sharp corners, while the next layers combine these edges to learn a bit more complex figures. Considering that the first few layers are actually learning about the entire ImageNet dataset (millions of images with 1000 different classes containing cars, flowers, cats, dogs, etc.), these layers have the most information about the very basic shapes, figures, and artifacts. Therefore, in our experiments only the last 3 FC layers are updated during the fine-tuning epochs, since the first group of layers hold information about basic shapes that could be useful for our challenge of recognizing emotions. In addition, since not all layers are being trained during the fine-tuning process, the computational cost required is considerably less.

Table 4.1: VGG16 architecture

input ($224 \times 224 \times 3$)
conv3 - 64
conv3 - 64
maxpool
conv3 - 128
conv3 - 128
maxpool
conv3 - 256
conv3 - 256
conv3 - 256
maxpool
conv3 - 512
conv3 - 512
conv3 - 512
maxpool
conv3 - 512
conv3 - 512
conv3 - 512
maxpool
FC - 4096
FC - 4096
FC - 6
softmax

4.2.1 Hyperparameters

Setting the hyperparameters before starting training is a crucial task in ML problems. Hyperparameters are the variables which determine the network structure and the variables which determine how the network is trained. In our case the variables related to the network structure are predefined by the creators of VGG16 (e.g. activation function: ReLU). Table 4.2 shows the values of the main network hyperparameters. After using several combinations for setting the learning rate, the batch size and the kind of optimization method, the following combination was the most effective.

learning rate	0.001
batch size	30
optimizer	Adam

Table 4.2: Hyperparameters

The learning rate defines how quickly a network updates its parameters. A low learning rate slows down the learning process but converges smoothly, whereas a larger learning rate speeds up the learning but may not converge. For the Adam optimizer, the initial recommended learning rate is equal to 10^{-3}, and this is the value that we use for our training.

In addition, it would be inconvenient to pass the entire dataset into the CNN at once, therefore we divide it into batches. The batch size, which is set to 30 frames, is the number of samples given to the network, after which a parameter update happens. The pre-processing steps applied to the data are explained in detail in Section 2.4, and they are also summarized in the following block diagram.

Moreover, the choice of loss function also plays an essential role in producing optimum and faster results. The loss function is a measure of how good a prediction model operates in terms of being able to predict the expected outcome. For this Thesis experiments, the cross-entropy loss function was utilized to measure the performance of the classification model during the training epochs. The loss according to the cross-entropy algorithm can be described as:

$$loss(x, class) = -\log \frac{\exp(x[class])}{\sum_{j=0} J - 1 \exp(x[j])} = -x[class] + \log \sum_{j=0} J - 1 \exp(x[j]) \tag{4.1}$$

where $class$ is an index variable that indicates the ground truth for a specific input, x is a vector which contains the scores for each class and J is the number of classes.

In the next sections, we present the four main implementations of our approach. For the confusion matrices in the following pages the open source scikit-learn [42] software tool was utilized.

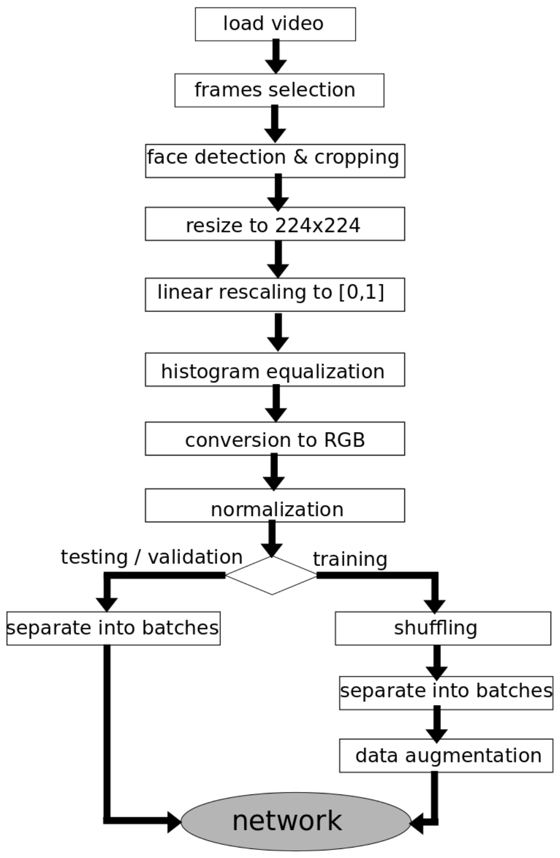


Figure 4.1: Pre-processing steps

4.3 Testing Methods

The participants of the testing videos are not the same with the ones used for training the model, in order to avoid bias. Every testing video, like all videos of our dataset, start with a frame of neutral facial expression and ends at the most intense, in terms of emotion, frame. So for testing our model we are mostly interested in the frames that show facial expressions rather than the ones with neutral faces.

The fine-tuned model gives per-frame class scores:

$$s = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_6 \end{bmatrix} \tag{4.2}$$

Per-video predictions are then calculated by using two different methods.

- Majority Vote

  First, a prediction for each frame is made by selecting the emotion with the maximum score at the output of the softmax layer $(s_1, s_2, \dots, s_6)$. Given the frame predictions $p_1, p_2, \dots, p_n$ where n is the number of frames for that video, we apply majority voting to $p_{\lceil n/2 \rceil}, \dots, p_n$ to find which emotion has more occurrences. We use only the second half of each video sequence since these frames show facial expressions.

- Mean Scores

  For the second method, the class scores of each frame are used to find the video prediction. In particular, we calculate the means of the emotion scores across the last half frames ($\lceil p/2 \rceil$ to p) of the video:

$$meanscores = \begin{bmatrix} mean(s_1) \\ mean(s_2) \\ \vdots \\ mean(s_6) \end{bmatrix} \tag{4.3}$$

  Then, we select the emotion class with the maximum mean value to find the final emotion prediction for each video.

4.4 1^{st} Implementation

In the 1^{st} implementation an attempt has been made to fine-tune the last three FC layers of the VGG16 model, using the hyperparameters mentioned above. At the end of every epoch the average loss of all batches is calculated to check its convergence (Figure 4.2). It can be observed that the loss decreases significantly during the first epochs and then it seems to converge.

At the end of every epoch the model was evaluated on the training and validation sets in order to observe the occurence of over-fitting during training (Figure 4.3). Note that the validation and training sets are evaluated by frame and not by video classification at this stage. The goal is to stop the training process earlier at a point when the model shows high accuracy on the unseen data of validation set but before it largely overfits on the training set, which will be represented by a drop in validation accuracy and an increase in training accuracy. Therefore, we stopped training at the end of the 170^{th} epoch to evaluate the model on the testing set.

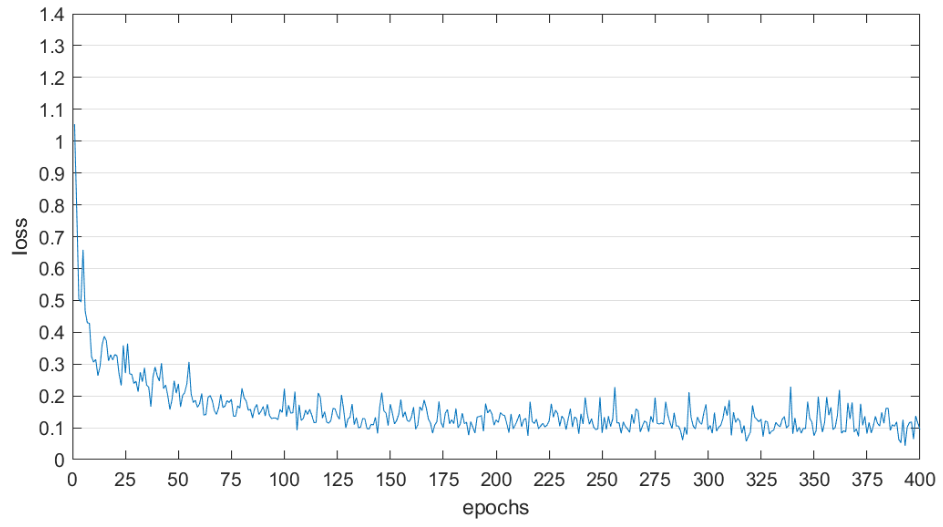


Figure 4.2: Loss value through epochs

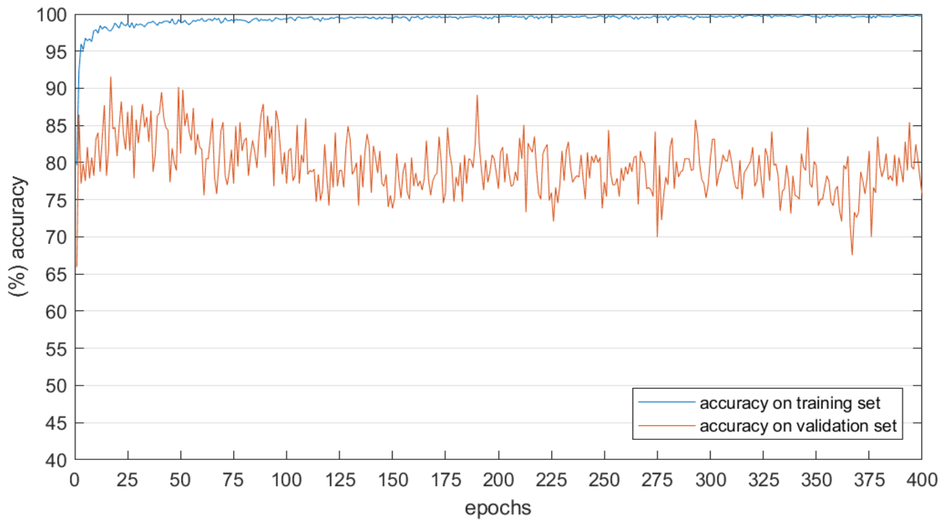
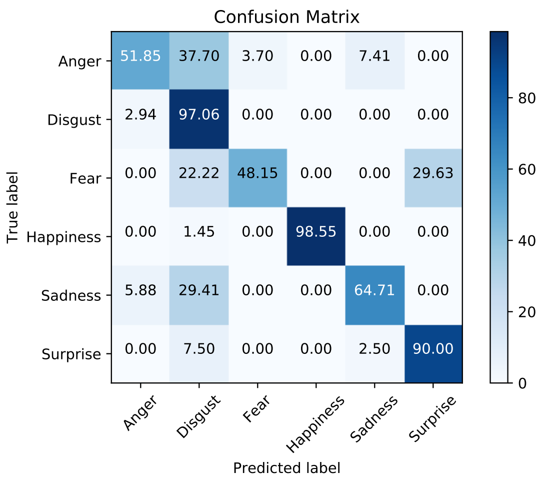
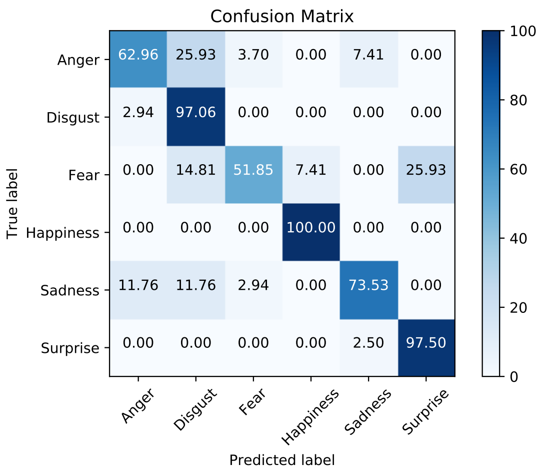


Figure 4.3: Model accuracy on the training and validation sets through the epochs.

The results of both evaluation methods are shown in Figure 4.4. As can be seen, the second method of mean scores provides more accurate results, compared to the majority voting method.



(a)



(b)

Figure 4.4: (a) Confusion matrix using the majority voting algorithm with average accuracy 75.05 %. (b) Confusion matrix using mean scores with average accuracy 80.48 %.

We can observe that some classes show significantly higher accuracy than others. "Happiness", "Surprise" and "Disgust" all perform better compared to "Fear". This observation can be attributed to the fact that some emotions are more distinctive than others and also some emotions share similar facial expressions [1].

4.5 2^{nd} Implementation

As previously mentioned, using a large number of parameters, while having a limited amount of training data, may force the model to map undesired properties of the data (noise). Therefore in the 2^{nd} implementation we apply a change in the architecture of VGG16. We reduce the network complexity, by decreasing the number of parameters in the second FC layer. Parameter reduction is expected to help the model generalize better on unseen data while avoiding over-fitting.

Fine-tuning is applied to the last three FC layers of the model, while keeping all hyperparameters the same as in 1^{st} implementation. Again we can can observe the loss convergence at Figure 4.5.

At the end of every epoch the model was evaluated on the training and validation sets in order to check the occurence of over-fitting during training (Figure 4.6). It can be seen that the vertical distance between the 2 graphs increases

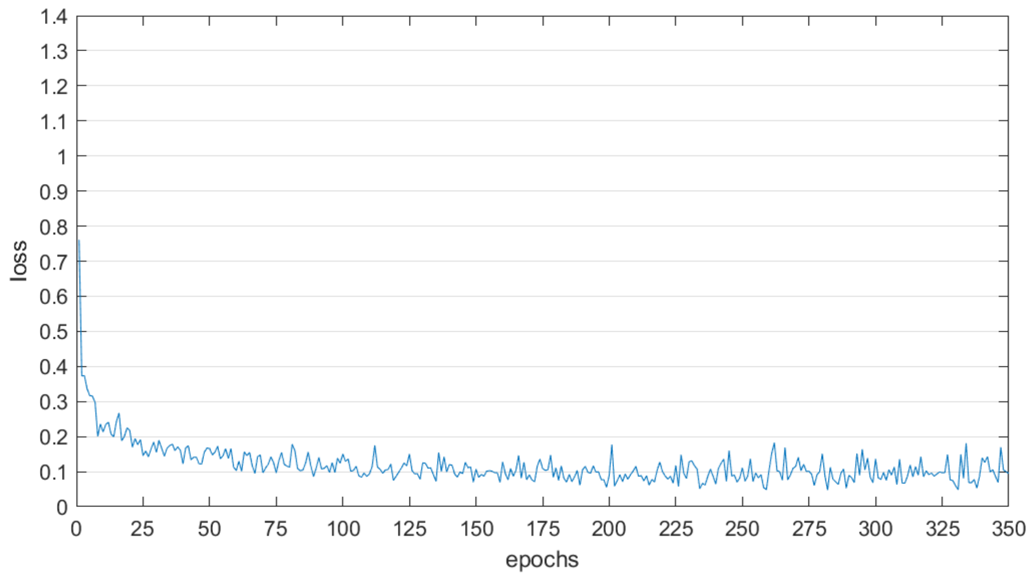


Figure 4.5: Loss value through epochs

significantly after the 200th epoch. Therefore the training process stops earlier at the end of 170th epoch to evaluate the model on the testing set.

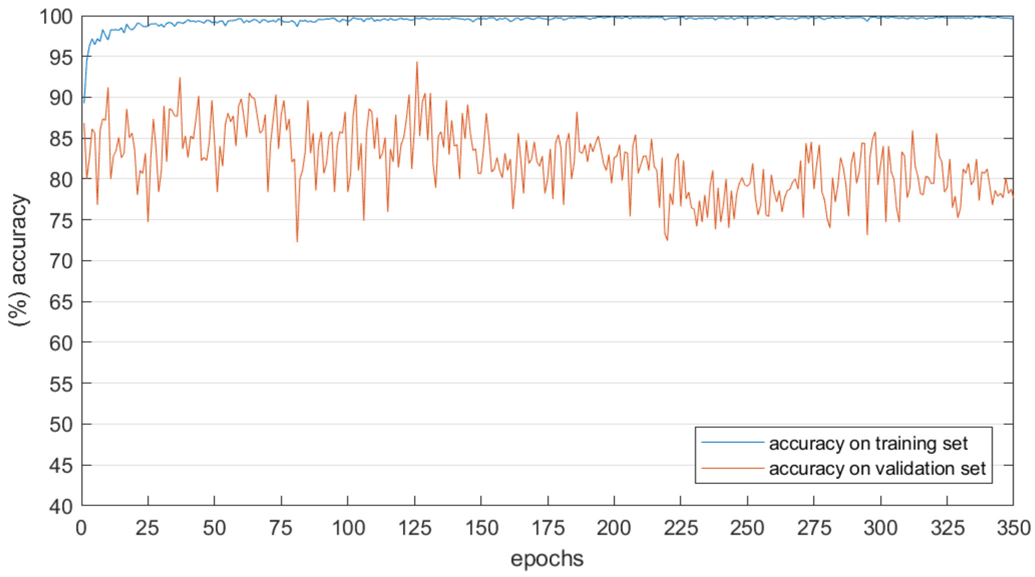
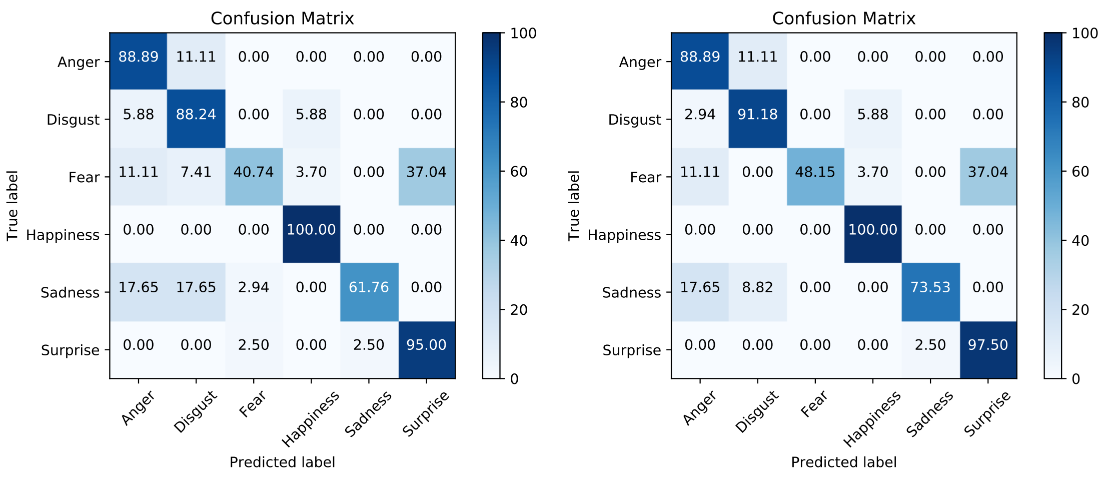


Figure 4.6: Model accuracy on the training and validation sets through the epochs.

After the evaluation process we can see how the model behaves on the data of the testing set (Figure 4.7) using both evaluation methods described in the previous section.



(a) (b)

Figure 4.7: (a) Confusion matrix using the majority voting algorithm with average accuracy 79.10%. (b) Confusion matrix using mean scores with average accuracy 83.20%

As can be seen, parameter reduction provides improved results compared to the initial network architecture of the previous implementation.

It can be also seen that some classes show higher accuracy than others. "Happiness" and "Surprise" both perform better compared to "Fear" and "Sadness". As mentioned before, this observation is intuitive as some emotions are more distinctive. Especially "Fear" presents a low hit-rate and this can be attributed to the fact that this emotion is quite subtle and it gets easily confused with "Surprise".

Additionally, another explanation could be given for the low accuracy of some classes. If we focus on the structure of the training set, we can see that classes do not share equal portions of training samples, and this leads to worse accuracy for weaker classes. In the next implementations we attempt to address this issue.

4.6 3^{rd} Implementation

Most real-world classification problems display some level of class imbalance. The problem of class imbalance occurs when the instances of one (or more) classes outnumber the instances of other classes in the training set. Imbalanced datasets degrade the performance of ML techniques as the overall accuracy and decision making is biased to the majority classes, which leads to misclassifying the weaker classes samples or furthermore to treating them as noise.

Several methods have been proposed to face the effects of class imbalance including oversampling instances of the minority class, undersampling instances of the majority class, and cost-sensitive learning. After applying all these methods, we focused on cost-sensitive learning since it was the most effective.

Cost-sensitive learning techniques take the misclassification cost into account by assigning higher cost of misclassification when the training samples belong to weaker classes [43]. In regular learning, all misclassifications are equally treated, but this causes issues in imbalanced classification problems, since there is no extra reward for identifying the minority class over the majority class. Cost-sensitive learning changes this, and uses a function $C(i)$ that specifies the cost of misclassifying an instance of class i. This allows us to penalize misclassifications of the minority class more heavily than we do with misclassifications of the majority class, in hopes that this increases the accuracy rate. This change is applied to the loss function of our network.

The structure of our training set is presented in Table 4.3. As can be seen, "Happiness" has more samples of all the other emotions, whereas "Fear" and "Anger" are the weakest classes as well as "Disgust".

	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Total
No of frames	1252	1549	1186	2908	1887	1989	10774

Table 4.3: Number of frames for every class in the training set

So, we use the weighted version of cross entropy loss function to calculate the loss value during the training phase for the same network architecture as the one of the 1^{st} implementation.

$$loss(x, class) = weight[class](-x[class] + \log \sum_{j=0} J - 1 \exp(x[j])) \tag{4.4}$$

The weights for each class are determined by the following formula:

$$weight[class] = \frac{n_{max}}{n_{class}} \tag{4.5}$$

where n_{class} is the number of frames of that specific class and n_{max} is the number of frames of the majority class, in our case "Happiness". In this way, the penalization increases as the class size decreases, and thus the weakest classes have larger weights.

At the end of every epoch the average loss of all batches is calculated to check its convergence (Figure 4.8).

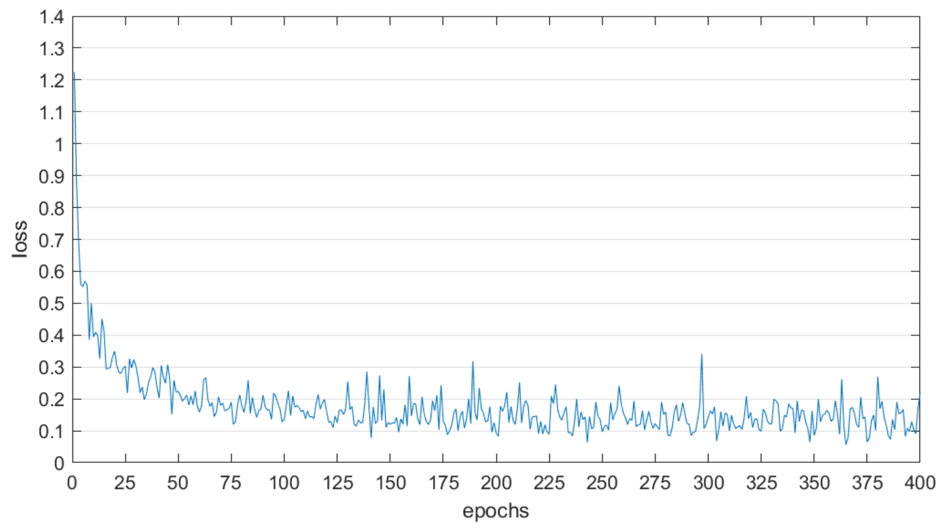


Figure 4.8: Loss value through epochs

Also the model was evaluated on the training and validation sets through the epochs and the training process stopped at the 270^{th} epoch (Figure 4.9)

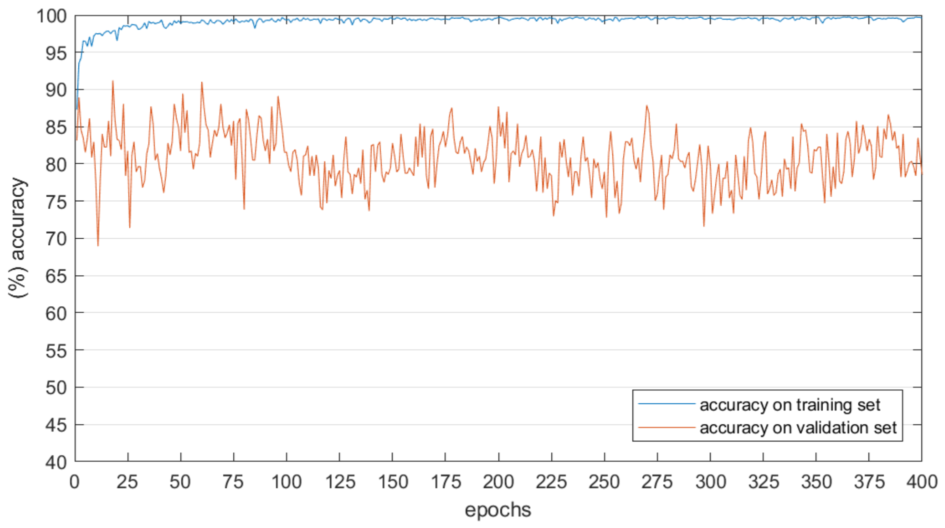
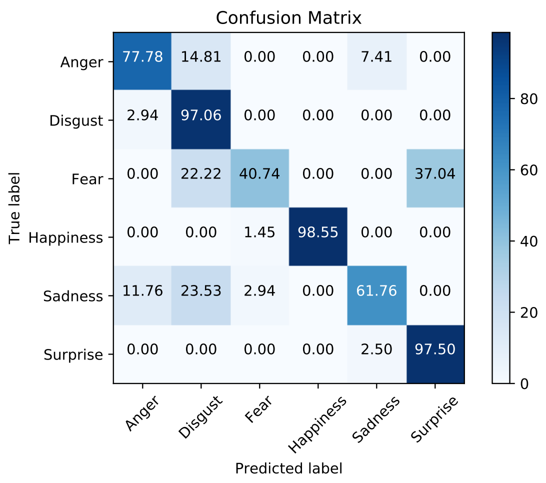
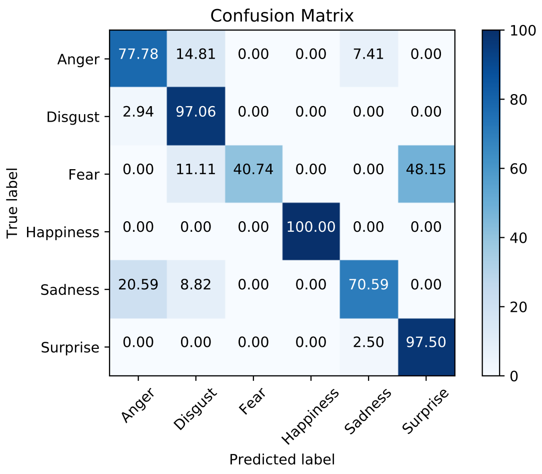


Figure 4.9: Model accuracy on the training and validation sets through the epochs.

Looking at the confusion matrices (a) and (b) of the 1st implementation (Figure 4.4), we can see that "Fear" shows the lowest accuracy rate: 48.15% and 51.85% respectively. Also "Anger" has a low accuracy rate of 51.85% in (a) and 62.96% in confusion matrix (b) at the same figure. Nevertheless, "Disgust", has a very high accuracy, 97.06%, despite the fact that it is one of the minority classes. Below we can see the results of cost-sensitive learning in the accuracy of the model on the testing set.



(a)



(b)

Figure 4.10: (a) Confusion matrix using the majority voting algorithm with average accuracy 78.9% (b) Confusion matrix using mean scores with average accuracy 80.61%.

Whilst the class of "Fear" does not improve and is still strongly misclassified as "Surprise", the "Anger" accuracy increases to 77.78% in both (a) and (b) confusion matrices of Figure 4.10. Moreover the "Disgust" accuracy remains at the same high level. It is also noticeable that the average accuracy of the system improves compared to the 1st implementation.

4.7 4th Implementation

In the 4th and last version of the proposed framework we apply cost sensitive learning tecniques to train the simplified network architecture of the 2nd Implementation. Below we can see the loss convergence after some epochs in figure 4.11.

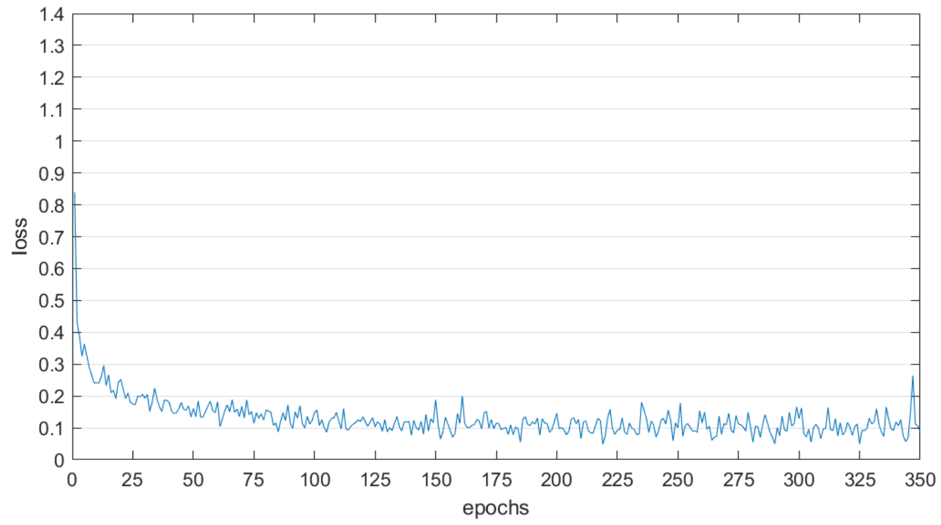


Figure 4.11: Loss value through epochs

In Figure 4.12 it can be observed that for the first 250 epochs the accuracy on the validation set stays above 80% in most epochs. This indicates that there is not a strong overfitting effect.

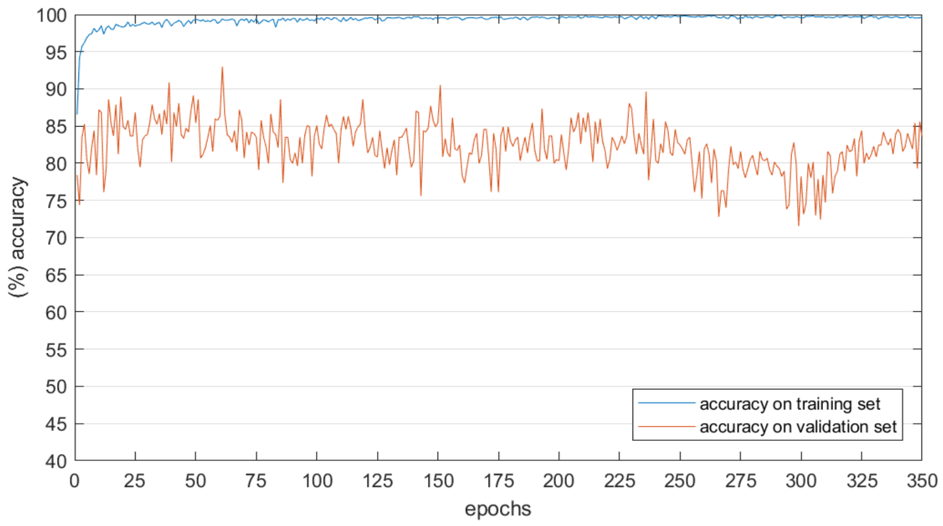
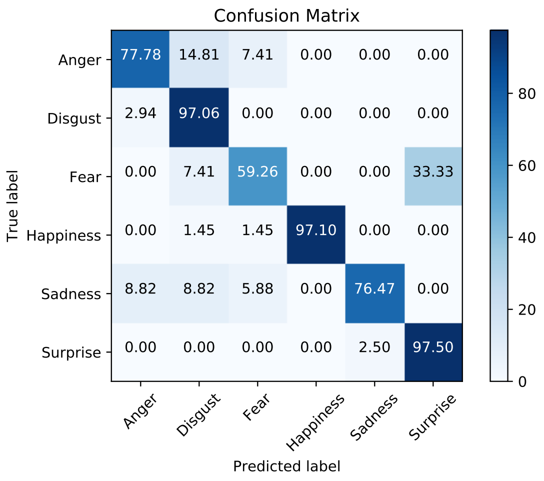
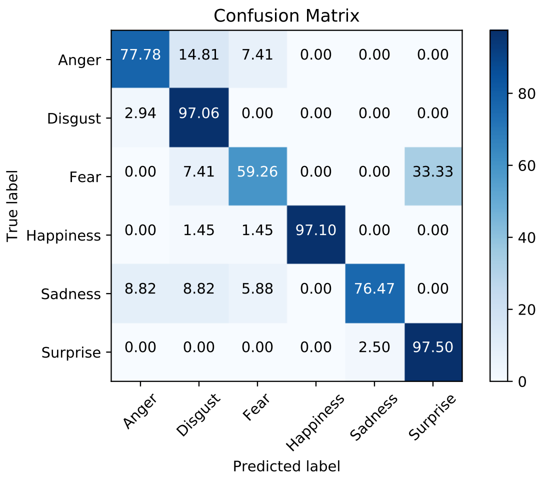


Figure 4.12: Model accuracy on the training and validation sets through the epochs.

Finally, the training process stopped at the 170th epoch, and the evaluation results are illustrated in the following confusion matrices.



(a)



(b)

Figure 4.13: (a) Confusion matrix using the majority voting algorithm with average accuracy 80.37%. (b) Confusion matrix using mean scores with average accuracy 84.2%

Recalling the results of the 2nd Implementation in Figure 4.7 of the previous section, we can see that "Fear" which is the weakest class, in terms of training samples number, has 40.74 % and 48.15 % accuracy on the testing set for the (a) and (b) evaluation methods respectively. Furthermore, in the same confusion matrices the "Anger" class has a high accuracy (88.89 % in both matrices) as well as "Disgust" (with 88.24% and 91.18 % for the (a) and (b) confusion matrices).

Comparing those results with the ones shown in Figure 4.13, it is noticeable that "Fear", which had the worst accuracy, improves significantly, and the accuracy of the new version is equal to 59.26% in both confusion matrices. The "Disgust" accuracy also increases to 97.06%. However, we see that "Anger" does not improve as its accuracy decreases. The network average accuracy is higher compared to the 2nd implementation, where we do not use cost-sensitive learning, but also compared to all previous implementations. In conclusion, parameter reduction along with the use of a weighted loss function produced optimum results in the testing procedure.

Chapter 5

Evaluation and Discussion

Facial expression analysis and recognition has become an active research topic in recent decades due to its potential contribution to a wide range of HCI applications. In this Thesis we present a deep learning approach to perform emotion recognition from video recordings. Inspired by the recent success of deep CNNs in image recognition tasks, we created a FER system based on applying transfer learning tecniques to the well-known VGG16 architecture.

Deep learning algorithms, including deep CNNs, require large amount of training data to achieve state-of-the-art results, and this is considered to be their main limitation. However there is not sufficient amount of labeled data for video emotion recognition in the existing literature. We overcame this issue by loading the pre-trained weights of the model which have been calculated after training the model on the very large dataset of ImageNet (1.2M images for object recognition). Despite the fact that our training dataset is significantly smaller ($\simeq$11K frames), it was proved to be enough for fine-tuning the initialized model and making it capable of classifying emotions.

We found that by reducing the number of network parameters, overfitting decreases, and the classifier's accuracy improves. However, there is still a noteworthy difference in the individual accuracies of the classes. "Happiness" and "Surprise" perform in general better than the other classes, whereas "Fear" has the lowest accuracy. Considering that this happens due to class imbalance in our training set, we applied cost-sensitive learning to train the model. We used a weighted cost function that penalizes more strictly misclassifications of samples that belong to weaker classes. The cost sensitive method was applied to both initial and "reduced" architectures. We found it difficult to draw conclusions for the cost-sensitive learning effectiveness, since it seems effective and it increases the model accuracy but not in all cases. For example, the "Fear" misclassification as "Surprise" stays at a high rate. This can be attributed to the fact that these two emotions share similar facial expressions, with "Surprise" being more distinctive (as it can be seen, "Surprise" is not misclassified as "Fear"). In conclusion, cost-sensitive learning helps the model to perform better under the limitations of psychological factors of the data.

Furthermore, we explore two testing methods, majority voting and score averaging, for calculating video predictions from frame predictions. Our experiments in Chapter 4 show that score averaging method outperforms majority voting in all confusion matrices, since it keeps more information from the class scores of all frames.

Finally, all implementations investigated in this work are static frameworks. This means that the training samples are video frames and the models produce per-frame predictions, based on 2D convolutional architectures. Nevertheless, they do not model the dynamic of the videos and avoid temporal information which may help to produce more accurate video predictions. A 3D CNN, in which each training sample is a video and the model gives per-video predictions was a potential approach for video emotion recognition. However this solution is computationally expensive with prohibitively high memory requirements.

Bibliography

[1] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, "The extended Cohn-Kanade dataset (CK+): A complete dataset for action unit and emotion-specified expression," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, June 2010, pp. 94–101.

[2] N. Aifanti, C. Papachristou, and A. Delopoulos, "The MUG facial expression database," in *11th International Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS 2010, Desenzano del Garda, Italy, April 12-14, 2010.* IEEE, 2010, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/5617662/

[3] N. Vállez, O. Deniz, and G. Bueno, "Sample selection for training cascade detectors," *PloS one*, vol. 10, p. e0133059, 07 2015.

[4] "Stanford university, course cs231n: Convolutional neural networks for visual recognition." [Online]. Available: http://cs231n.stanford.edu/

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1106–1114. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

[6] 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

[7] "A brief report of the heuritech deep learning meetup 5." [Online]. Available: https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/

[8] R. W. Picard, "Affective computing." MIT Press, 1997.

[9] B. Knyazev, R. Shvetsov, N. Efremova, and A. Kuharenko, “Convolutional neural networks pretrained on large face recognition datasets for emotion classification from video,” *CoRR*, vol. abs/1711.04598, 2017. [Online]. Available: http://arxiv.org/abs/1711.04598

[10] Y. Fan, X. Lu, D. Li, and Y. Liu, “Video-based emotion recognition using CNN-RNN and C3D hybrid networks,” in *Proceedings of the 18th ACM International Conference on Multimodal Interaction, ICMI 2016, Tokyo, Japan, November 12-16, 2016*, Y. I. Nakano, E. André, T. Nishida, L. Morency, C. Busso, and C. Pelachaud, Eds. ACM, 2016, pp. 445–450. [Online]. Available: https://doi.org/10.1145/2993148.2997632

[11] S. E. Kahou, C. J. Pal, X. Bouthillier, P. Froumenty, Ç. Gülçehre, R. Memisevic, P. Vincent, A. C. Courville, Y. Bengio, R. C. Ferrari, M. Mirza, S. Jean, P. L. Carrier, Y. Dauphin, N. Boulanger-Lewandowski, A. Aggarwal, J. Zumer, P. Lamblin, J. Raymond, G. Desjardins, R. Pascanu, D. Warde-Farley, A. Torabi, A. Sharma, E. Bengio, K. R. Konda, and Z. Wu, “Combining modality specific deep neural networks for emotion recognition in video,” in *2013 International Conference on Multimodal Interaction, ICMI ’13, Sydney, NSW, Australia, December 9-13, 2013*, J. Epps, F. Chen, S. Oviatt, K. Mase, A. Sears, K. Jokinen, and B. W. Schuller, Eds. ACM, 2013, pp. 543–550. [Online]. Available: https://doi.org/10.1145/2522848.2531745

[12] R. Cowie, E. Douglas-Cowie, N. Tsapatsoulis, G. Votsis, S. Kollias, W. Fellenz, and J. G. Taylor, “Emotion recognition in human-computer interaction,” *IEEE Signal Processing Magazine*, vol. 18, no. 1, pp. 32–80, Jan 2001.

[13] M. Soleymani, M. Pantic, and T. Pun, “Multimodal emotion recognition in response to videos (extended abstract),” in *2015 International Conference on Affective Computing and Intelligent Interaction, ACII 2015, Xi’an, China, September 21-24, 2015*, 2015, pp. 491–497. [Online]. Available: https://doi.org/10.1109/ACII.2015.7344615

[14] P. Tzirakis, G. Trigeorgis, M. A. Nicolaou, B. W. Schuller, and S. Zafeiriou, “End-to-end multimodal emotion recognition using deep neural networks,” *J. Sel. Topics Signal Processing*, vol. 11, no. 8, pp. 1301–1309, 2017. [Online]. Available: https://doi.org/10.1109/JSTSP.2017.2764438

[15] P. Ekman, W. V. Friesen, M. O’Sullivan, A. Y. C. Chan, I. Diacoyanni-Tarlatzis, K. G. Heider, R. Krause, W. Ayhan LeCompte, T. Pitcairn, and P. Ricci Bitti, “Universals and cultural differences in the judgments of facial

expressions of emotion," *Journal of personality and social psychology*, vol. 53, pp. 712–7, 11 1987.

[16] P. Michel and R. E. Kaliouby, "Real time facial expression recognition in video using support vector machines," in *Proceedings of the 5th International Conference on Multimodal Interfaces, ICMI 2003, Vancouver, British Columbia, Canada, November 5-7, 2003*, S. L. Oviatt, T. Darrell, M. T. Maybury, and W. Wahlster, Eds. ACM, 2003, pp. 258–264. [Online]. Available: https://doi.org/10.1145/958432.958479

[17] X. Pu, K. Fan, X. Chen, L. Ji, and Z. Zhou, "Facial expression recognition from image sequences using twofold random forest classifier," *Neurocomputing*, vol. 168, pp. 1173–1180, 2015. [Online]. Available: https://doi.org/10.1016/j.neucom.2015.05.005

[18] D. Huang, C. Shan, M. Ardabilian, Y. Wang, and L. Chen, "Local binary patterns and its application to facial image analysis: A survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 41, no. 6, pp. 765–781, Nov 2011.

[19] F. Tang and B. Deng, "Facial expression recognition using AAM and local facial features," in *Third International Conference on Natural Computation (ICNC 2007)*, vol. 2, Aug 2007, pp. 632–635.

[20] S. Polikovsky, Y. Kameda, and Y. Ohta, "Facial micro-expressions recognition using high speed camera and 3d-gradient descriptor," in *3rd International Conference on Imaging for Crime Detection and Prevention (ICDP 2009)*, Dec 2009, pp. 1–6.

[21] H. Jung, S. Lee, J. Yim, S. Park, and J. Kim, "Joint fine-tuning in deep neural networks for facial expression recognition," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 2983–2991.

[22] S. A. Bargal, E. Barsoum, C. Canton-Ferrer, and C. Zhang, "Emotion recognition in the wild from videos using images," in *Proceedings of the 18th ACM International Conference on Multimodal Interaction, ICMI 2016, Tokyo, Japan, November 12-16, 2016*. ACM, 2016, pp. 433–436. [Online]. Available: https://doi.org/10.1145/2993148.2997627

[23] S. E. Kahou, V. Michalski, K. R. Konda, R. Memisevic, and C. J. Pal, "Recurrent neural networks for emotion recognition in video," in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, Seattle, WA, USA, November 09 - 13, 2015*, Z. Zhang, P. Cohen, D. Bohus,

R. Horaud, and H. Meng, Eds. ACM, 2015, pp. 467–474. [Online]. Available: https://doi.org/10.1145/2818346.2830596

[24] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015.* IEEE Computer Society, 2015, pp. 4489–4497. [Online]. Available: https://doi.org/10.1109/ICCV.2015.510

[25] R. E. Haamer, E. Rusadze, I. Lüsi, T. Ahmed, S. Escalera, and G. Anbarjafari, "Review on emotion recognition databases," in *Human-Robot Interaction - Theory and Application*, 2018, exported from https://app.dimensions.ai on 2019/02/09. [Online]. Available: https://www.intechopen.com/chapter/pdf-download/58386

[26] A. Jaimes and N. Sebe, "Multimodal human-computer interaction: A survey," *Computer Vision and Image Understanding*, vol. 108, no. 1-2, pp. 116–134, 2007. [Online]. Available: https://doi.org/10.1016/j.cviu.2006.10.019

[27] M. Pantic, M. F. Valstar, R. Rademaker, and L. Maat, "Web-based database for facial expression analysis," in *Proceedings of the 2005 IEEE International Conference on Multimedia and Expo, ICME 2005, July 6-9, 2005, Amsterdam, The Netherlands.* IEEE Computer Society, 2005, pp. 317–321. [Online]. Available: https://doi.org/10.1109/ICME.2005.1521424

[28] S. L. Happy, P. Patnaik, A. Routray, and R. Guha, "The Indian spontaneous expression database for emotion recognition," *IEEE Transactions on Affective Computing*, vol. 8, no. 1, pp. 131–142, Jan 2017.

[29] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker, "Multi-pie," *Image Vision Comput.*, vol. 28, no. 5, pp. 807–813, May 2010. [Online]. Available: http://dx.doi.org/10.1016/j.imavis.2009.08.002

[30] M. Pantic, M. Valstar, R. Rademaker, and L. Maat, "Web-based database for facial expression analysis," *IEEE International Conference on Multimedia and Expo, ICME 2005*, vol. 2005, pp. 5 pp.–, 08 2005.

[31] F. Famili, W. Shen, R. Weber, and E. Simoudis, "Data preprocessing and intelligent data analysis," *Intell. Data Anal.*, vol. 1, no. 1-4, pp. 3–23, 1997. [Online]. Available: https://doi.org/10.1016/S1088-467X(98)00007-9

[32] P. A. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," in *2001 IEEE Computer Society Conference on*

Computer Vision and Pattern Recognition (CVPR 2001), 8-14 December 2001, Kauai, HI, USA. IEEE Computer Society, 2001, pp. 511–518. [Online]. Available: https://doi.org/10.1109/CVPR.2001.990517

[33] V. Nair and G. E. Hinton, “Rectified linear units improve restricted Boltzmann machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, J. Fürnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 807–814. [Online]. Available: http://www.icml2010.org/papers/432.pdf

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–, Oct. 1986. [Online]. Available: http://dx.doi.org/10.1038/323533a0

[35] M. D. Zeiler, “ADADELTA: an adaptive learning rate method,” *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[36] A. Kostrikin, P. Suetin, A. Kostrikin, I. Manin, and Y. Manin, *Linear Algebra and Geometry*, ser. Algebra, logic and applications. Taylor & Francis, 1989. [Online]. Available: https://books.google.gr/books?id=mPkZAQAAIAAJ

[37] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[38] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[39] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, ser. Lecture Notes in Computer Science, D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., vol. 8689. Springer, 2014, pp. 818–833. [Online]. Available: https://doi.org/10.1007/978-3-319-10590-1_53

[40] C. Szegedy, S. Ioffe, and V. Vanhoucke, “Inception-v4, inception-resnet and the impact of residual connections on learning,” *CoRR*, vol. abs/1602.07261, 2016. [Online]. Available: http://arxiv.org/abs/1602.07261

[41] 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

[42] 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.

[43] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 63–77, Jan 2006.