UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Study and implementation of machine learning algorithms for facial expression analysis

Diploma Thesis

Ioannis Athanasiadis

Supervisor: Panagiota Tsompanopoulou

February 2022

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Study and implementation of machine learning algorithms for facial expression analysis

Diploma Thesis

Ioannis Athanasiadis

Supervisor: Panagiota Tsompanopoulou

February 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη και εφαρμογή αλγορίθμων μηχανικής μάθησης για ανάλυση συναισθημάτων προσώπου

Διπλωματική Εργασία

Ιωάννης Αθανασιάδης

Επιβλέπουσα: Παναγιώτα Τσομπανοπούλου

Φεβρουάριος 2022

Approved by the Examination Committee:

Supervisor	**Panagiota Tsompanopoulou** Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Lefteris Tsoukalas** Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Dimitrios Bargiotas** Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

First of all, I would like to express my gratitude to my Prof. Panagiota Tsompanopoulou for providing guidance and feedback throughout this subject, but also for the opportunity of selecting this thesis. Additionaly, special thanks to Sofia Stylianou, assistant researcher of Department of Informatics and Telecommunications, whose support as part of her PhD allowed my studies to go the extra mile. Furthermore, I would like to thank Vangelis Neamonitis, for the thoughtful comments and recommendations on this dissertation. Finally, I would like to acknowledge my family and friends for the unwavering support throughout the writing of this dissertation.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Diploma Thesis

Study and implementation of machine learning algorithms for facial expression analysis

Ioannis Athanasiadis

Abstract

Both computer vision and machine learning have flourished as a result of ongoing improvements in computer hardware and the growing amount of information on image and video data. Therefore, a study research that overlaps pattern recognition and computer vision should always be beneficial. Our goal is to classify human emotions from video recordings in a variety of challenging situations. To do so, we are using ML methods, such as preprocessing techniques for video data, as well as a deep learning neural network architecture for emotion classification. More specifically, we utilize the obtained data by transforming them into appropriate input for pretrained neural network architectures. The next step is to adjust the parameters of some convolutional neural network architectures, increasing the effectiveness of emotion classification. Lastly, we evaluate the various implementations and analyse the results.

Διπλωματική Εργασία

Μελέτη και εφαρμογή αλγορίθμων μηχανικής μάθησης για ανάλυση συναισθημάτων προσώπου

Ιωάννης Αθανασιάδης

Περίληψη

Η υπολογιστική όραση και η μηχανική μάθηση έχουν ακμάσει πρόσφατα λόγω των συνεχών βελτιώσεων στο πεδίο του υλικού των υπολογιστών, καθώς και εξαιτίας του τεράστιου όγκου πληροφορίας στον τομέα της εικόνας και του ήχου. Συνεπώς, ένα ερευνητικό πεδίο που επικαλύπτει την υπολογιστική όραση και την αναγνώριση προτύπων είναι ιδιαίτερα επωφελές. Στόχος μας είναι η ταξινόμηση ανθρωπίνων συναισθημάτων από βιντεοσκοπήσεις, σε μια ποικιλία απαιτητικών περιπτώσεων. Για να το καταφέρουμε αυτό, χρησιμοποιούμε μεθόδους μηχανικής μάθησης, όπως τεχνικές προεπεξεργασίας για δεδομένα από βίντεο, καθώς και αρχιτεκτονικές νευρωνικών δικτύων βαθιάς μάθησης, με στόχο την ταξινόμηση συναισθημάτων.Πιο συγκεκριμένα, χρησιμοποιούμε τα ληφθέντα δεδομένα μετατρέποντάς τα σε κατάλληλη είσοδο για αρχιτεκτονικές προεκπαιδευμένων νευρωνικών δικτύων. Το επόμενο βήμα είναι η προσαρμογή των παραμέτρων κάποιων αρχιτεκτονικών συνελικτικών νευρωνικών δικτύων, αυξάνοντας έτσι την αποτελεσματικότητα της ταξινόμησης συναισθημάτων. Ώς τελικό βήμα, αξιολογούμε τις διάφορες υλοποιήσεις και αναλύουμε τα αποτελέσματα.

Table of contents

List of figures

List of tables

Abbreviations

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

Chapter 1

Introduction

1.1 Pattern Recognition

Pattern recognition is defined as the automated process of recognizing patterns, characteristics, and consistencies in data that offer information about a dataset or system[2]. A pattern is a repeated sequence of data across time that can be used to anticipate things such as trends, recurrent features in pictures, and even commonalities in human speech and handwriting. A current approach to artificial intelligence includes pattern recognition, and vice - versa. Pattern recognition can be divided into two types: a) machine learning based and b) rule / heuristics based.

1.2 AI

Machine learning [2] as well as all -related- applications is one rapidly expanding field of AI with a wide range of capabilities and uses, not just in research and scientific systems, but also in everyday practical scenarios utilized by a huge number of people. An extensive range of algorithms commonly referred to as machine learning is responsible for the great majority of AI discoveries and breakthroughs. These algorithms use mathematical and statistical models to find patterns and trends in massive volumes of data.

1.2.1 Learning Methodologies

Machine learning algorithms are classed according to how they handle data and make an effort to learn from them. As a result, three distinct learning methodologies are frequently explored:

- Supervised learning[2]: In the current situation, the input data is referred to as ”training data” and has a predetermined label. A training process is used to build and tune a model, which involves creating predictions about already known data, and adjusting the parameters once they are incorrect. This process is rehashed until our model achieves the target ”accuracy” on the training data. Thereafter, to evaluate it, we take into account another unused piece of data commonly referred to as ”testing data”. Classification (categorizating items) and regression (estimating variable relationships), are two common issues that supervised learning can tackle.

- Unsupervised Learning[2]: The input data in this category are not categorized and have no known outcome. The process of creating a model begins with the discovery of underlying structures in the input data. This way, general ”rules” could be extracted, whether via a mathematical process in order to remove redundancy, or via arranging data by relevance. Unsupervised learning methods (for example, the k-means classifier) can address problems including clustering and dimensionality reduction.

- Semi-Supervised Learning[2]: Perhaps there is a mix of labeled and unlabeled samples in the input data. Semi-supervised algorithms often train using a rather small amount of labeled data and a big amount of unlabeled data. Semi-supervised techniques are expansions to previous flexible approaches that are making assumptions about how to model the unlabeled data, and can be used to tackle issues like regression and classification[2]. Self-training and generative models are examples of such methods.

1.2.2 Emotions

In social interaction, perception, and human intelligence, emotions are truly important[12]. Fundamentally, an emotion can be expressed through quite a lot of social behaviors, such as facial expressions, gestures, text, speech, etc. Understanding emotions becomes important for humans in daily life because emotional awareness and expertise are necessary for effective social communication.

Since emotional intelligence is a vital part of artificial intelligence, computer-assisted emotion recognition has recently become a difficult task for academics and industry. Emotion-aware systems have already revolutionized computer vision research, and they significantly enhanced human-computer interaction (HCI) [13] [14][15]. Monitoring driver state (– for example- fatigue state), assessing a person’s emotional response during a game, identifying depression in adults, even diagnosing abnormalities in youngster development by monitoring their facial expressions, are only a few of the applications for such systems. Emotion-sensitive machines can also help boost automated tutoring by showing whether the examples are distracting or dull to the user [16].

Encouraged by the recent deep learning successes, the tasks of emotion analysis have advanced with the use of deep learning algorithms. Some examples would be convolutional, recurrent neural networks , and hybrid approaches that incorporate both methodologies [14][15]. Moreover, various modalities such as video, physiological measures, audio and their combinations have been used to resolve automated emotion recognition[17][18].

Among the nonverbal components of interpersonal communication, visual information is the most important. Gestures of the face, in particular, account for a significant portion of nonverbal communication. As a result, we concentrate on the data's visual modality in this paper, obtain information from faces through recordings (spatio-temporal), and focus on emotion detection using fine-tuned pretrained CNN models.

While we, as humans, experience a wide range of emotions during our lives, in order to simplify machine learning problems, it is important to declare a collection of distinct emotional groups for automatic recognition. In the early 70s, Ekman stated that people of all cultures interpret such essential emotions in facial expressions in the same way [19]. Humans share six basic emotions, according to his research: sadness, happiness, surprise, anger, fear, disgust. This hypothesis from Ekman has been generally acknowledged, and it has sparked the interest of many researchers. As a result, for this thesis project, we will concentrate on those six emotional categories.

1.3 Related Work

Through the technological advance of artificial intelligence, attention in automatic facial emotion recognition (FER) has grown in recent decades, and various alternative approaches to FER have been developed.

1.3.1 Traditional FER Methodologies

Face or facial part identification, feature extraction, and emotion classification are the three primary processes in the FER pipeline in classic FER approaches. After extracting a face image from a source image, facial components (e.g., eyes and nose) and landmarks are detected. Then, from these elements, a variety of temporal and spatial features are detected. Then, using the extracted features, classification algorithms support vector machines, and random forests produce recognition results[20] [21].

Pietikäinen [22] used local binary patterns to extract features and SVM to classify different facial expressions from static images. An active appearance model(AAM) is used by other image approaches to extract facial features points, and build a classifier by combining useful local shape features[23]. The AAM's role is to correlate a statistical model, to a unique undetected picture created throughout the training process.

Numerous systems were utilized [22] in video FER in order to calculate the geometrical displacement of facial landmarks (as temporal features) between the current frame and previous frames.

The main distinction between Facial Emotion Recognition for video sequences and the one for static images, would be that the former's landmarks are monitored frame by frame, and the system produces new, dynamic features through displacement (between current and previous frames) [24].By separating face regions and creating 3D-Gradients orientation histograms from each region's motion, Pfister et al. [24] introduced facial microexpressions recognition in video sequences.

1.3.2 FER Methodologies Based on Deep Learning

Deep learning evolved into being a more overall approach to Machine Learning(ML), providing innovative outcomes in a variety of computer vision research projects with the accessibility of big data, in comparison to conventional approaches using handcrafted features.By learning directly from the input images, such emotion recognition methods greatly reduce dependency on models which are face-physics-based [25]. However, as these algorithms need massive sets of data to produce the most advanced and latest results, the volume of data with labeled training does have a significant effect on the performance of deep NN models. This would be their main drawback.

CNN and RNN deep learning algorithms were utilized to extract features and categorize emotions. Käding et al. utilized two CNNs in 2016: one extracted temporal appearance features from image sequences, and the other extracted temporal geometry features from temporal facial landmark points[25]. To improve the performance of facial expression recognition, both models were integrated by using a novel integration approach. Bargal et al. [26] suggested an algorithm based on deep learning, in the 2015 Emotion Recognition in the Wild Challenge. Deep CNN's like ResNet and VGG were also used, with their main purpose being the role of feature extractors.

For emotion detection, a variety of methods are directly using a CNN. However, since methods based on CNN cannot account for ”temporal fluctuations among facial components”, several hybrid approaches have been developed, that combine a CNN for spatial features of individual frames and a long short-term memory (LSTM) for temporal features of consecutive frames. The LSTM is a form of RNN capable of learning long-term data dependencies. Kahou et al. [27] suggested a hybrid CNN-RNN architecture that uses a continuously evaluated hidden-layer representation to propagate information over a sequence. According to the authors [27] ”a hybrid CNN-RNN architecture for facial expression analysis would outperform a CNN approach using temporal averaging for aggregation” in their paper, which was submitted for the Wild Competition’ Emotion Recognition in 2015.

Deep 3-dimensional CNNs is another newly suggested architecture which has achieved significant progress in dealing with diverse tasks in video analysis. 2-dimensional CNNs have a significant drawback in that they can only accommodate one dimension. Although ignoring essential temporal video information, spatial information is omitted. C3D, on the other hand, is able to model both appearance as well as motion data at the same time. Khan and Reeshad [1] tried to combine C3D and CNN-LSTM models in their study. A RNN [1] ”takes appearance features extracted by a CNN over individual video frames as input and encodes motion later, while a C3D models both appearance and motion of video simultaneously”, according to the proposed system. Consequently, the hybrid network outcomes provided are competitive (regarding emotion classification).

1.4 Outline and Contribution

Facial expressions are extremely significant in human social contact. It’s no surprise, then, that automatic FER has been the focus of a lot of recent studies. In this study, we are presenting a cutting edge approach for classifying facial expressions. This approach is based on deep CNNs’ successful performance in a variety of recognition problems.

Deep architectures, despite their effectiveness, necessitate a substantial quantity of data for their training, which has yet to be recorded in the emotion recognition literature (especially in the case of video data). An additional problem relating to these limitations from data, is the fact that pose expressions, which were recorded in closed lab environments, are often found in existing databases. As a consequence, when evaluated on real-world data, a model trained using these data performs poorly. Regarding this experiment, we utilize transfer learning techniques with pre-trained CNN architectures to solve the aforementioned limitations of other FER approaches. To obtain final classification models[25], the pre-trained model is fine-tuned with limited emotion labeled training data.

Fine tuning would be useful in our case because the network has gained basic understanding of edges, shapes or curves from a vast dataset and therefore can correlate them in a smaller dataset. Furthermore, we merged three separate databases to create our emotion database, one of which included some spontaneous facial expressions.

1.5 Chapter Organization

The chapters are organized as:

- **Chapter 2.** The theory of Neural Networks is examined in depth here (NNs).Then we describe CNNs as a special type of NN that has been effective in a variety of recognition tasks.

- **Chapter 3.** This is where we covers every aspect of emotion detection databases, including a wide range of categories.

- **Chapter 4.** The experimental findings and the four distinct iterations of the proposed solution are presented in this chapter.

- **Chapter 5.** As a final step, we draw some conclusions from the experiments in the last chapter.

Chapter 2

Neural Networks

2.1 Introduction

The aim of NNs was originally to model biological neural networks, but the area has separated, evolving into an engineering topic, thus achieving decent accuracy in ML applications. The brain's most basic component is the neuron. The human nervous system is made up of over 86 billion neurons connected by approximately $10^{14} - 10^{15}$ synapses. A dendrite is where each neuron collects input, in order to generate output along its single axon.

This axon connects the output of one neuron to the dendrites of other neurons through synapses.

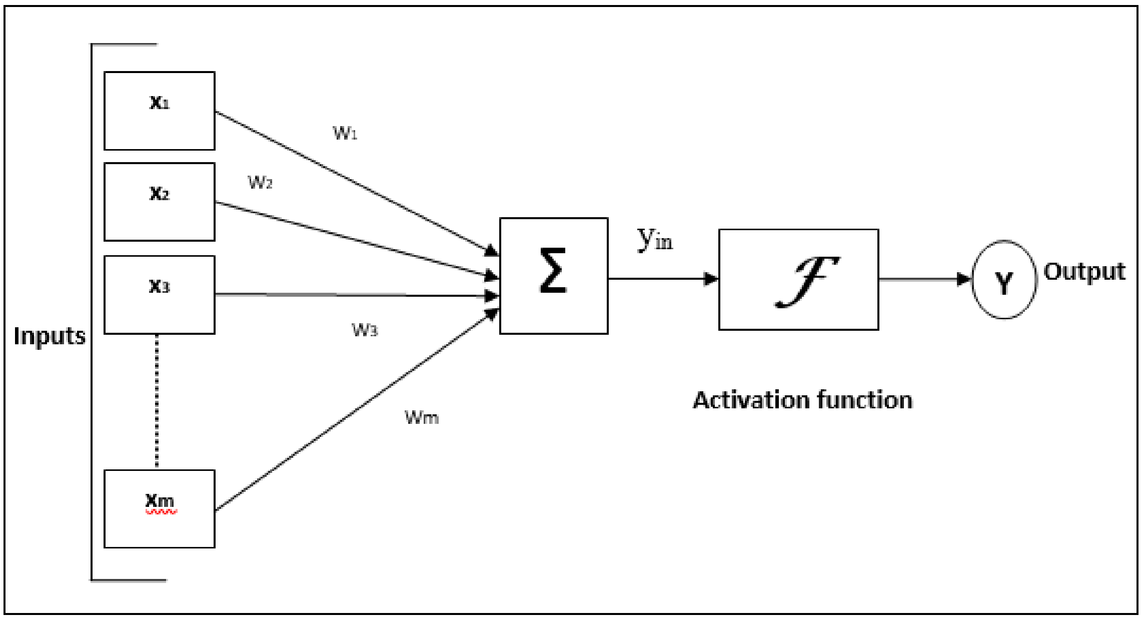


Figure 2.1: Artificial Neuron [1]

Moving on to the computational model describing an artificial neuron, each signal that moves along the (x_i) axons interacts multiplicatively $(w_i * x_i)$ with the dendrites of the other neuron, depending on the synaptic strength at that synapse (w_i). Every neuron generates a weighted sum S in the basic model, with b being the neuron's bias value. The concept behind NN technology is that synaptic strengths (or weights w and biases b) can be learned and used to alter the amount of impact that one neuron has on another. Then, the neuron will fire (sending a spike along its axon) if the final sum is greater than a certain limit. An assumption can be made, that the exact timings of the spikes are irrelevant in the computational model, and just the firing frequency sends information. As a resut, there is a need to use an activation function for the modeling of the neuron's firing rate, which describes the spike's frequency. Every single neuron, referred to as a node henceforth, creates a dot product with the input and its weights, then adds the bias and applies the activation function.

2.2 Neural Network Basics

Neural networks are commonly in layer format. Layers consist of a series of nodes which are interconnected, that have a non-linearity activation function. Through the input layer, patterns are introduced to the network. This layer interacts with at least one hidden layer, which conducts the processing through a system of weighted connections. The output layer nodes, unlike all other layers in a NN, rarely include some activation feature. The reasoning behind this is that the final layer in classification problems normally reflects the class scores, which are random numbers with real values, or a sort of real-valued target.

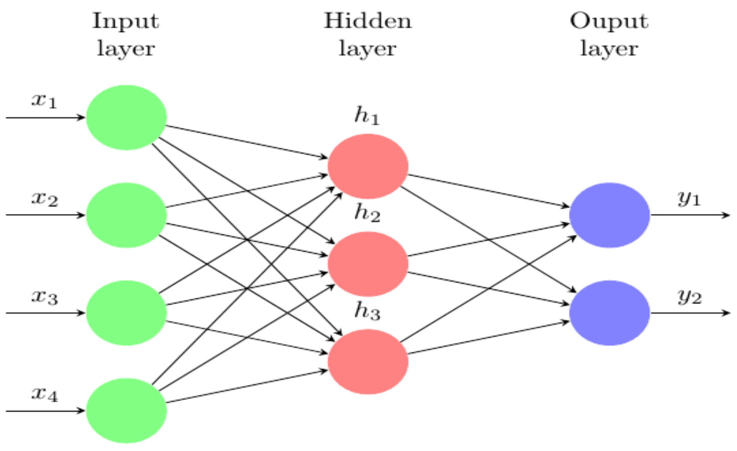


Figure 2.2: NN with 3 layers [1]

There are two ways in which data flows through a NN. The input nodes feed information patterns into the network, that activate the hidden nodes layers, which then arrive at the output layer, when [1] the network is learning (training phase) or operating normally (testing phase). A feedforward network is commonly designed in this way. Not every node fires all of the time. When traveling through the network, [1] each node receives inputs from the nodes to its left, and the inputs are multiplied by the weights of the connections. In this manner, each node combines all of the inputs it collects, and if the number exceeds a certain threshold value (in the simplest type of network), the node fires, triggering the units to its right.

Feedback is required for a NN to learn, much as youngsters do when they hear what is done correctly or incorrectly. We, as humans, compare the desired result to what actually truly happened, determine the difference, and use that information to alter our behavior in the future. NNs learn in the same way as humans do, usually through a feedback loop. This loop is backpropagation (BP). It entails the comparison of the network's output, to the one it was supposed to produce, then adjusting the weights of the connections between the nodes in the network based on the difference. We accomplish this by working in the opposite direction (from the output to the input nodes) passing through the hidden nodes. The network can learn over time, narrowing the gap of real and desired output.

The entire training set is transmitted across the network back and forth repeatedly during the training phase. An epoch would be complete pass over the whole training set. We split the information data in much smaller partitions, due to the high memory costs of back and forth propagation of the whole dataset. It's worth noting that the training process ends beyond a certain number of epochs.

When the network [1] has been trained with enough learning examples, we can provide new sets of inputs and observe its reaction. This is the testing phase. Considering the training of the network, it will try to classify new, unlabeled samples into one of the classes, generalizing from its previous experience.

2.3 Activation function

Each activation function starts with a single integer, then applies a certain mathematical operation to it. Tanh, ReLU, and sigmoid activation functions are three of the most widely used:

1. $(sigmoid)$ Mathematical form of the sigmoid activation function:

$$(x) = \frac{1}{1 + e^{x}} \tag{2.1}$$

The function squashes a real-valued number into 0-1 range. As this function can be a simple analogy of the firing rate of a neuron, the sigmoid feature has seen a lot of use: from firing at a fully saturated maximum frequency (1) to not firing at all (0). The sigmoid has gradually lost practical application and it is seldom used, due to two significant drawbacks.

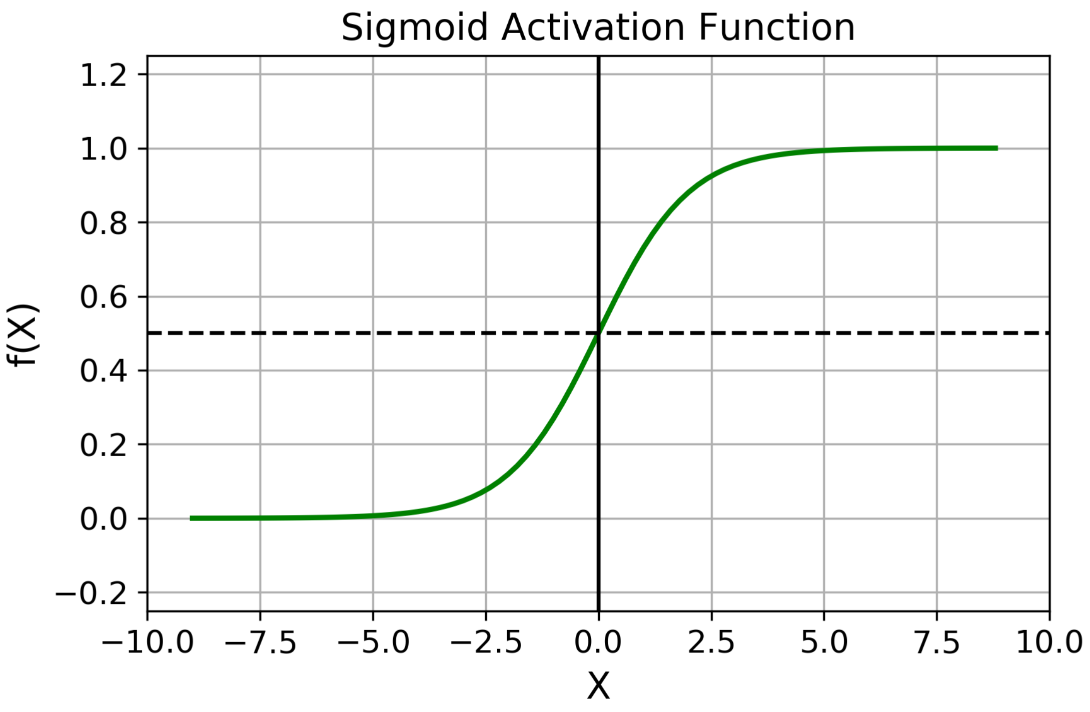


Figure 2.3: Sigmoid Activation Function [2]

The first drawback of the sigmoid neuron is that when its activation reaches a saturating point at either the 0 or 1 tail, the gradient in these areas is nearly zero. Since this local gradient will be multiplied by the gradient of this gate's output during BP, if the local gradient is very small, the gradient will be effectively killed, and almost no signal will flow through the neuron to its weights, and recursively to its data. Consequently, during the weights initialization of sigmoid neurons, extra caution must be exercised to avoid saturation.

Sigmoid outputs do not have zero-centering either. This is also a downside since neurons in the NN's later processing layers would receive data which are not zero-centered. This has consequences for gradient descent dynamics, as data entering a neuron will be positive, then the gradient on the weights w will become either all positive or all negative during BP. It's worth noting, however, that the data is supplied in smaller portions in the network. The last weight update may have a variety of signs when the gradients sum up over a data batch, which helps to mitigate this problem. As a result, although this is inconvenient, this will have fewer negative implications than the aforementioned saturated activation problem.

2. $(ReLU)$ Over the last years, the Rectified Linear Unit (ReLU) has gained a lot of momentum. The activation is actually thresholded at zero, since the function f(x) = max(0, x) is computed. In comparison to sigmoid and tanh functions, ReLU was found to significantly accelerate convergence in the training phase. This is thought to be due to its linear, non-saturating form. The main disadvantage of ReLU is that units and die

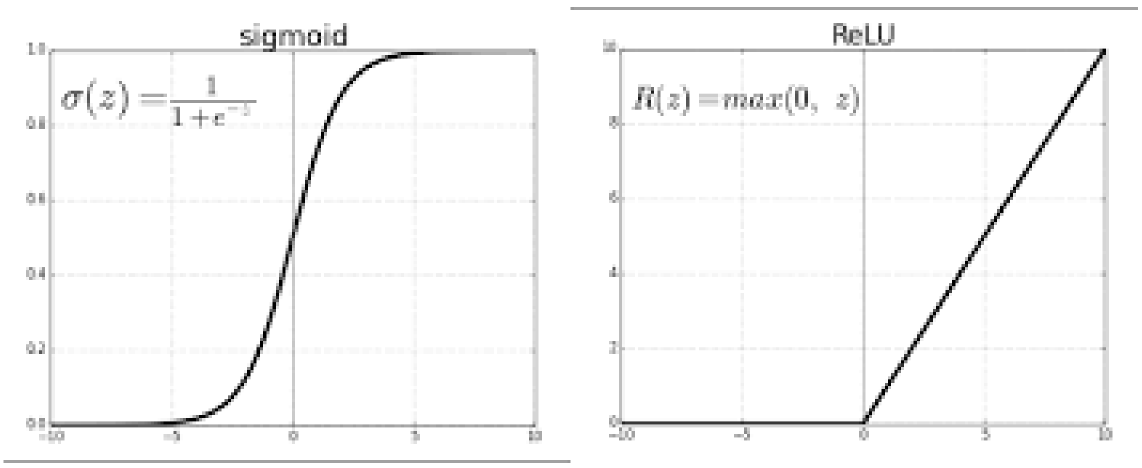


Figure 2.4: ReLU Activation Function (Compared to Sigmoid) [2]

during training as they are more fragile. In this case, the gradient flowing through the same neuron will be zero for the rest of the time. A large gradient flowing through a ReLU neuron, for example, could trigger the weights to update to the point where there is no further activation of the neuron. If this occurs, from then on the unit gradient flowing will become null.

3. $(tanh)$ The tanh non-linearity squashes a real valued number to the range [-1, 1]. Its activations saturate like the sigmoid neurons, but its performance is zero-centered. As a result, the tanh non-linearity is often favoured over the sigmoid non-linearity in practice. It's also worth noting that the tanh neuron is nothing more than a scaled sigmoid neuron [1], as shown: $(tanh(x) = 2(2x) - 1)$.

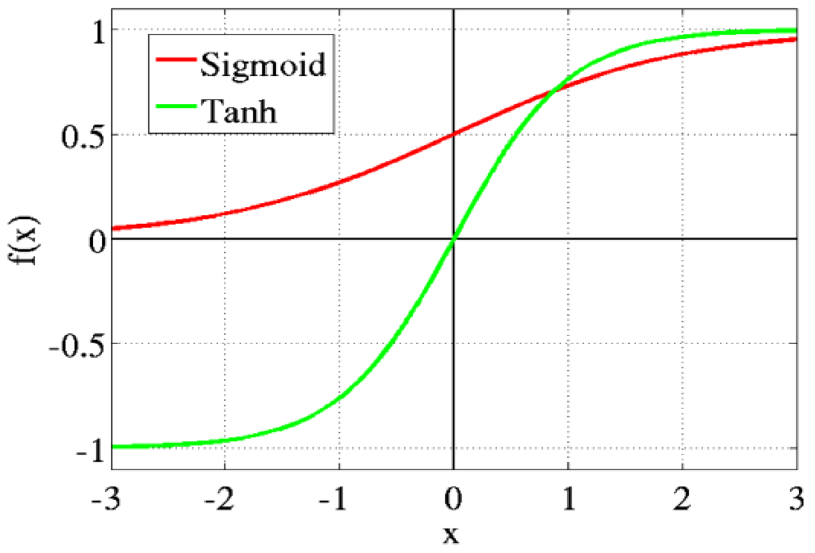


Figure 2.5: Tanh Activation Function (Compared to Sigmoid) [2]

2.4 Backpropagation

Different research communities have used this numerical method in various contexts. Seppo Linnainmaa first published the BP algorithm in 1970, though important concepts were already perceived quite sooner. David Rumelhart demonstrated in 1987 that by using a process as this, meaningful internal representations of data can be produced in NNs hidden layers. This algorithm began to fade during the 2000s, but it resurfaced in the 2010s, thanks to the advancement of low-cost, high-performance computing systems. Since then, it has indeed remained among the most researched NN learning algorithms.[28] The BP algorithm can be broken down into four stages, each of which is briefly listed below:

1. Feed-Forward Computation
2. Backpropagation to output layer
3. Backpropagation to hidden layer
4. Weight update

When the error function reaches a suitably low value in the output layer, the process ends. The BP algorithm is described in depth below.

Any number of hidden units can be added to a feed-forward network with n input and m output units, allowing to exhibit any desired feed-forward connection pattern. A training set

$(x_1, t, 1)...(x_p, t_p)$ containing p ordered pairs of n and m dimensional vectors, which are the input and output patterns, is also given. When this network is given the input pattern x_i from the training set, it generates an output o_i that is different from the target t_i in general. Using a learning algorithm, the aim is to make o_i and t_i identical for $i = 1, ..., p$. More specifically, we want to reduce the network's error function, which is defined as:

$$E = \frac{1}{2}\sum_{i=1}^{p} ||o_i - t_i||^2 \tag{2.2}$$

In order to identify the weight values combination which minimize the error function, we combine the BP algorithm with an optimizer. The network is set up with weights that are selected at random. In addition, the error function's gradient is calculated and utilized for the adjustment of initial weights. We have to recursively calculate this gradient.

A first phase in the minimization procedure is to expand the network in order to automatically calculate the error function. Each of the network's j output units is connected to a node that evaluates the function $E = \frac{1}{2}\sum_{i=1}^{p} ||o_i - t_i||^2$, with o_{ij} and t_{ij} denoting the j^{th} part of the output vector o_i and the target t_i, respectively. Each pattern t_i requires the development of the same network extension. All of the quadratic errors are gathered by a computing unit, which then outputs the sum $E_1 + ... + E_p$. The error function E is the output of this extended network. As a result, for a particular training set, the network will calculate the overall error. The only parameters left that may be changed in order to lower the quadratic error E are the weights of the network. Since E is computed exclusively through the composition of node functions by the extended network, it will be a continuous, differentiable function of the l weights $w_1, w_2, ..., w_l$ in the network. Consequently, we are able to minimize E. This can happen by utilizing an iterative gradient descent process, for which we must first calculate the gradient:

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_l}) \tag{2.3}$$

The increment is used to change each weight:

$$\triangle w_i = -\frac{\partial E}{\partial w_i}, for i = 1, ..., l \tag{2.4}$$

where l is the learning rate, a proportionality parameter that determines the step length each iteration in the negative gradient direction takes.

The entire learning problem is now reduced to measuring the gradient of a network function with respect to its weights, due to this extension of the original network.We can iteratively adjust the network weights after calculating the gradient. We expect to find a minimum of the error function in this manner, with $\nabla E = 0$. Undoubtedly, [3] any method for the modification of the network's weights can be used, in order to minimize the total error function that we defined earlier.

2.5 Optimization Algorithms

The error function E(x), also known as the 'objective function,' is minimized (or maximized) using optimization algorithms. The output values are calculated using the NN's internal learnable parameters, weights w and bias b, which are learned and modified in the direction of the optimum route. Internal model parameters are critical in swiftly and successfully training a model and producing high accuracy outcomes. And that is why, utilizing various optimization approaches, we compute acceptable values for these kind of model parameters. The optimization algorithm we chose has a significant impact on the learning process and outputs of our model. Some popular optimization algorithms are listed here:

1. **Gradient Descent**

   The most significant method and the cornerstone of NN optimization is gradient descent. Gradient descent is an optimization algorithm based on curved function which modifies its parameters iteratively to minimize a given function to its local minimum. Gradient Descent repeatedly reduces a function by moving it in the opposite direction direction from that of the steep ascent. It depends on its derivatives function loss to find minimums. Using the data of the whole training set to calculate the cost with the parameters requires a large amount of memory and thus slows down the procedure. The parameter updates have the following formula:

   $$\theta = \theta - \gamma \nabla J(\theta) \tag{2.5}$$

   where γ denotes the learning rate and $\nabla J(\theta)$ denotes the gradient of the loss function $J(\theta)$ with respect to θ [3][10].

   There are some disadvantages in the gradient descent algorithm. We must take a closer look at the amount of calculations we make for each iteration of the algorithm. Assuming we have 10,000 data points and 10 attributes. The sum of error squares consists of as many terms as data points, so we have 10000 terms in our case. We have to calculate the derivative of this function in relation to each of the characteristics, so we will actually do 10000 * 10 = 100,000 calculations per iteration. It is customary to take 1000 iterations, so in we actually have 100,000 * 1000 = 100000000 calculations to complete the algorithm. This is almost an insufficient cost therefore the gradient descent is slow in huge data. The weight updates process of the gradient is depicted in Fig. 2.6. The

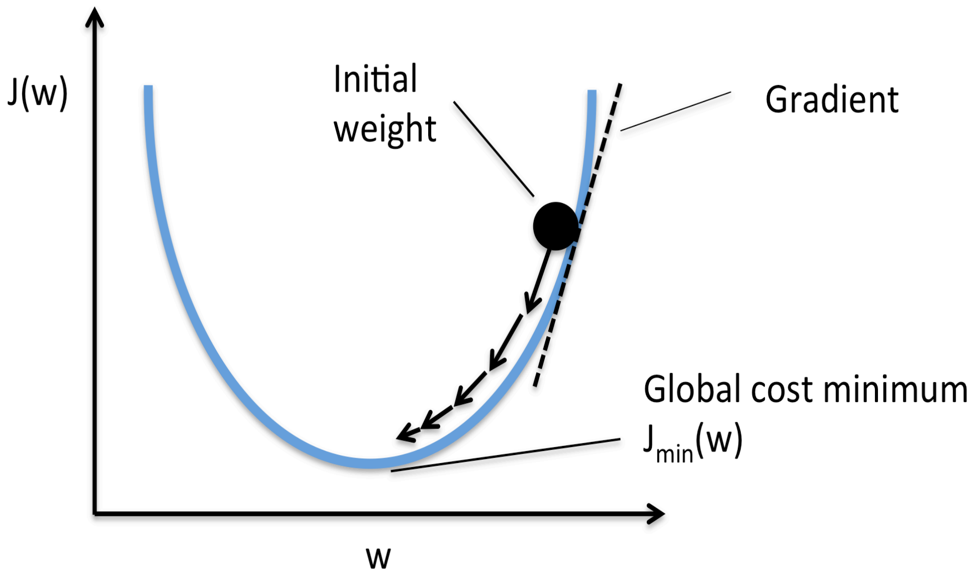


Figure 2.6: Gradient Descent [3]

gradient slope is represented by the U-shaped curve. As can be shown, depending on the weight values, the inaccuracies can grow substantially. As a result,[3] the updates should not be too big or too small, allowing us to decline in the opposite direction of the gradient until we hit a local minima. Several variations of gradient descent algorithm emerged later on, in addition to the simple version. Some of them include minibatch gradient descent, stochastic gradient descent (SGD) and Nesterov accelerated gradient.

2. **Adam** ADAM[29] stands for adaptive moment estimation, which translates to adaptive momentum approach. It is an algorithm for efficient stochastic optimization, which only needs first-order gradients. Particularly, focuses on optimizing high-dimensional stochastic goals parametric spaces (something we would encounter in the training of deep neural networks) where higher order methods would not be convenient, due to the larger their computational costs The algorithm itself calculates individual learning rates for different parameters of the model, from approaches it makes to the first and

second order moments of gradients, and are determined as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.6}$$

$$u_t = \beta_2 u_{t-1} + (1 - \beta_2) g_t^2 \tag{2.7}$$

, with m being the mean and u the uncentered variance. The authors of Adam noticed that the algorithm is biased towards zero since m_t and u_t are initialized as vectors of zeros, particularly during the initial time steps and when the decay rates are small (β_1 and (β_2 are close to one). By calculating bias-corrected first and second moment estimates, they were able to overcome these biases:

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{2.8}$$

$$\hat{u_t} = \frac{u_t}{1 - \beta_2^t} \tag{2.9}$$

Then, they are utilized in the same way as Adadelta, for the parameter update, resulting in the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{\hat{u_t}} + e} \tag{2.10}$$

In summary, some of the general advantages of ADAM are the following: There is little need for memory, the size of the update parameters is independent of gradient values, the practical steps of the algorithm are blocked by initial step given by the user, it does not need a fixed target, and it works when the slopes are sparse.

3. **Adagrad**

The Adagrad [30] [10] optimizer adjusts the learning rate according to the parameters. In reality, large updates are applied to infrequent parameters and small updates are applied to frequent parameters. As a result, it is an excellent optimizer for dealing with sparse data.

According to Adagrad[30], for each parameter θ at each time step, a different learning rate is being used based on the previous gradients computed for that parameter. Since every parameter θ_i used the same learning rate in gradient descent, all parameters were updated at the same time, while Adagrad used a different learning rate for each parameter $\theta_{t,i}$ on every time step t[30]. The Adagrad formula for parameter updates is:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\gamma}{\sqrt{G_{t,i+e}}} g_{t,i} \tag{2.11}$$

where $g_{t,i}$ is the gradient of the loss function with respect to the parameter θ_i at time step t, and γ represents the learning rate. G_t is a diagonal table in which every diagonal element at the (i, i) position, is equal to the sum of the squared gradients with respect to θ_i until time step t, and e is a smooth term in order to avoid division with zero.

Possibly the most significant advantage of Adagrad would be that it removes the need to manually adjust the learning rate. Most implementations use 0.01 as the default value and leave it there. The accumulation of squared gradients in the denominator, however, is its key flaw. Since each additional term adds to the total, the total grows during training. As a result, the learning rate decreases and gradually becomes insignificant, and our algorithm can not learn any new information. To fix this flaw, the algorithm that follows is designed.

4. **Adadelta**

An extension-improvement of Adagrad is Adadelta. We know that Adagrad has as a disadvantage due to the accumulation of its sums which reduce the learning rate over the course of repetitions. [31]. Adadelta, on the other hand, does not hold all the sums but only one part of them. The sum of the slopes depends on their mean previous inclinations. At time step t, the running average $E(g^2)_t$ is solely determined by the previous average and the current gradient.

$$E(g^2)_t = aE(g^2)_{t-1} + (1-a)g_t^2 \tag{2.12}$$

Around 0.9 is a standard value for a. The parameter update vector(PUV) is used to express the gradient descent update.

$$\Delta\theta_t = -\gamma g_{t,i} \tag{2.13}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t \tag{2.14}$$

As a result, the Adagrad PUV which we previously calculated:

$$\Delta\theta_t = \frac{\gamma}{\sqrt{G_t + e}} \otimes g_t \tag{2.15}$$

Where $\otimes$ the symbol of the symmetric product[32]. We can now easily substitute G_t with $E(g^2)_t$, which is the decaying average of past squared gradients.

$$\Delta\theta_t = \frac{\gamma}{\sqrt{E(g^2)_t + e}} g_t \tag{2.16}$$

2.6 Convolutional Neural Networks

Convolutional Neural Networks are common to NNs at the point that they aim at self-improvement through learning. The peculiarity of CNNs in relation to the traditional NN, is that the former are mainly used in pattern recognition in images. Only the last layer in the CNN is fully connected. In this way, its architecture parameters can be reduced to make learning a data type more targeted.[4]. At the end of a CNN, there are typically several ordinary fully connected (FC) layers. On the last FC layer, they still have a loss function. The key distinction is that instead of learning unstructured weights, CNN architectures use convolution to apply filters whose internal values are the weights to be learned. The explicit assumption that the inputs are images is made in the case of 2D convolutional filters, allowing us the encoding of those properties, into the architecture

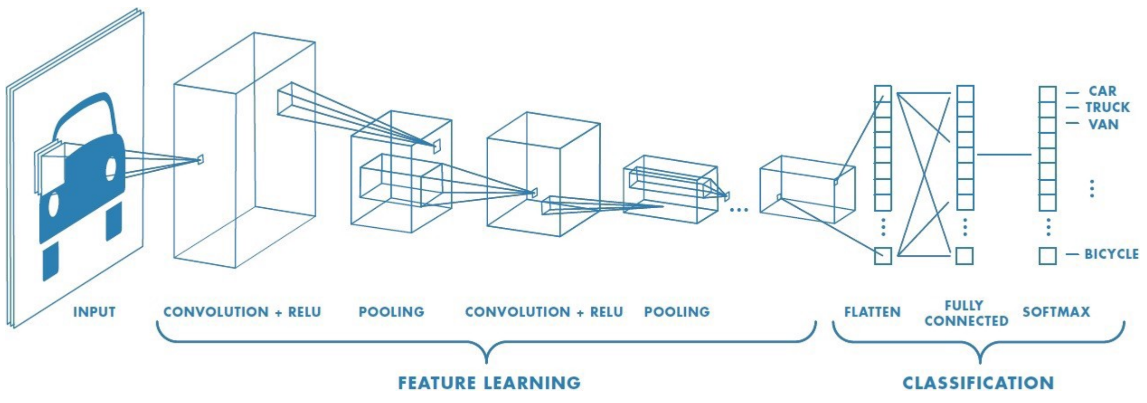


Figure 2.7: Convolutional Neural Network [4]

A key feature of a CNN is that it is made up of neurons that are organized in three dimensions. Height, width and depth. By depth, we mean the third dimension of the activation volume. Instead of the all to all neuron connection in a FC layer, the neurons in a convolutional layer are only linked to a small region of the layer before it. Furthermore, the full picture is condensed to a single vector of class scores ordered along the depth axis by the end of a CNN architecture.

CNNs are made up of three types of levels. These are convolutional layers, pooling layers and FC layers. When these levels stacked, CNN architecture is shaped. The network output can be a class or a probability of classes describing an image.

2.6.1 Layers: FC

As in normal NNs, the fully connected layer contains neurons that connect directly to the neurons of the two adjacent layers, without being connected to either layer inside them. It is the most basic type of layer, and is typically found at the bottom of CNN architectures, with the purpose of carrying out the required classification or regression task[4].

2.6.2 Layers: Convolutional

This is the central component of CNNs and is responsible for the majority of the computational work. The parameters of this layer are centered around the usage of learnable filters (or kernels). These filters are usually small in spatial layout, but extend to the entire width of the input. When the data enter the convolutional layer, the layer converges (slides) each filter in the spatial layout of the input, in order to produce a two-dimensional activation map.

To calculate the vectors of each layer, we first take the input values and apply vector multiplication by level. The result of the multiplication of this vector, is calculated for each of its neuron values with the step activation function. Out of this value, the network will find out which neurons are activated when a particular feature is detected from a specific input location. These values are also known as activations.

A simple example is by inserting an image with dimensions 64x64x3, that is, a three-channel RGB image with dimensions: 64 length and 64 width, with the receptive size set to 6x6, the result would be 108 weights for each neuron of the convolutional network. Unlike in NN we would have 12228 weights per neuron. Convolutional levels reduce the complexity of the model.

Three hyperparameters must be defined in order to control the number of nodes in a convolutional layer's output volume: stride, depth, and zero-padding.

Firstly, the number of filters-to-use corresponds to the depth of the output volume, with each of the filters learning to search for something unique from the input data. For example, if the raw image is fed into the first convolutional layer as input, different neurons along the depth dimension can fire in response to various oriented edges or color blobs. A depth column is from now on a set of neurons that all look at the same region of the input.

Secondly, the stride must be defined. When the stride is 1, we shift the filters one pixel at a time during convolution[4]. When we slide around with a stride of 2 (or rarely 3 or more), the filters jump 2 pixels at a time. In terms of production volume, larger strides yield smaller

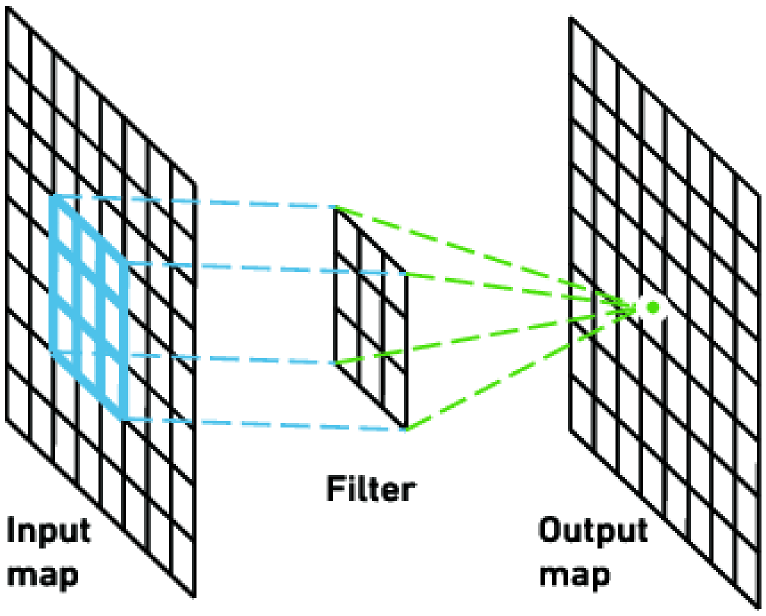


Figure 2.8: Convolutional Layer [4]

output volumes.

In certain cases, padding the input volume with zeros around the border is useful[4]. This zero-padding's size is a hyper-parameter. The aim of applying zero padding is to better manage the output volume's spatial size. We can precisely maintain the spatial size of the input volume in this way, ensuring that the input and output width and height are identical.

The spatial size of the output volume (O) can be calculated as a function of the input volume size (W), the convolutional layer neurons' receptive field size (F), the stride in which they are applied (S), and the amount of zero padding used on the border (P)[4].

$$O = \frac{W - F + 2P}{S} + 1 \tag{2.17}$$

2.6.3 Layers: Pooling

A pooling layer aims to gradually reduce the dimensions that representing the network and further reducing the number of parameters and the computational complexity of the model, thus regulating overfitting. The pooling layer operates in the input of each activation map, and scales its dimension using the ”MAX” function. In most CNNs, these maps come in the form of concentrated pooling layers with filters of 2×2 dimensions and with stride 2 along the spatial dimensions of the input. This reduces the size of the activation map by up to 25 percent of the original size, while maintaining the volume of the depth at its normal size. In this case below, each MAX operation will take a maximum of four numbers (little 2x2 region in some depth slice). The depth dimension stays the same.

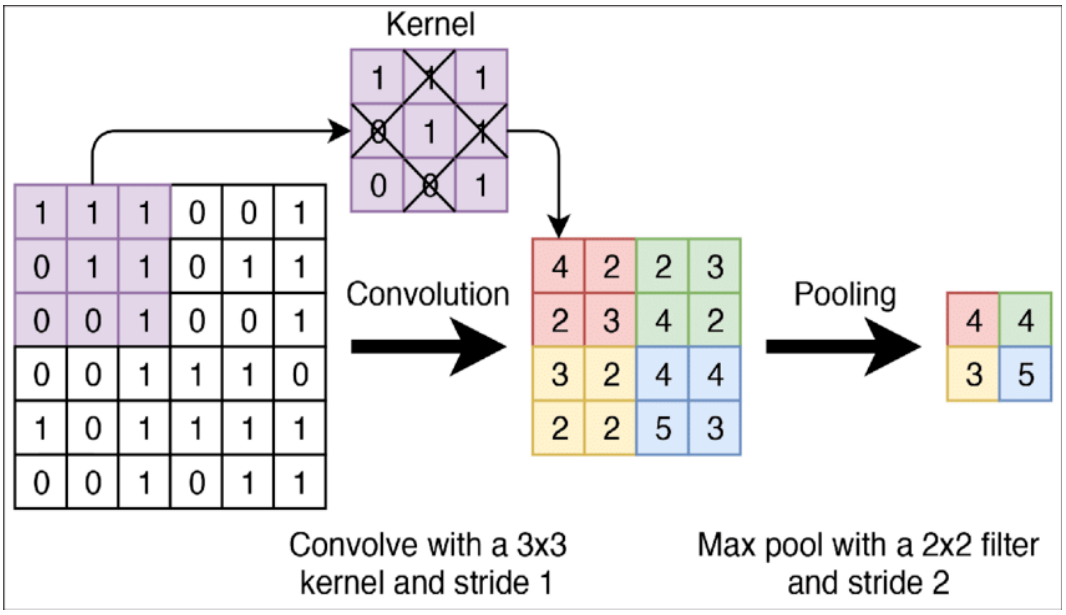

Kernel
Convolution
Pooling
Convolve with a 3x3 kernel and stride 1
Max pool with a 2x2 filter and stride 2

Figure 2.9: Pooling Layer [4]

The pooling units may perform other functions besides max pooling (eg. L2-norm). The most popular method is max-pooling.

2.6.4 CNN Architectures

Various CNN architectures have been established in recent decades and have achieved excellent outcomes on recognition tasks, especially for ImageNet. ImageNet project is declared as a 1.4 million-image visual repository intended for use in visual object recognition research. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition run by this project[5]. Some of the most prominent architectures used by ILSVRC's top finalists, are the following:

- AlexNet: AlexNet, created by Alex Krizhevsky, Geoff Hinton and Ilya Sutskever, was the first work to popularize CNNs in the field of Computer Vision [5]. AlexNet was introduced to the 2012 ILSVRC challenge, and far outperformed the second runner-up (top 5 error of 16 percent vs. 26 percent for the runner-up) [5]. The network's architecture shared similar components with LeNet's [33], but was much larger, deeper, with more width, and had stacks of convolutional layers on top of each other. Convolutional filters 11x11, 5x5, and 3x3 were used, as well as max pooling, dropout and SGD with momentum citeKrizhevsky. ReLU activations were introduced following each convolutional layer and each FC layer.

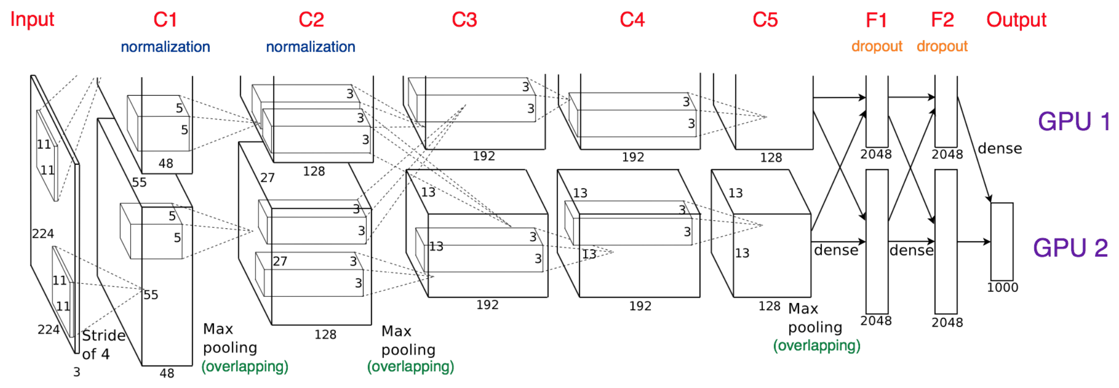


Figure 2.10: AlexNet [5]

- GoogleNet: The winner of the 2014 ILSVRC was a Google architecture proposed by Szegedy et al. [6]. When compared to AlexNet, it made a significant contribution by developing a 22-layer model that lowered the number of parameters from 60 million to 4 million. Furthermore, instead of using FC layers at the top of the CNN, this study utilized average pooling, which eliminated a large number of parameters that did not have great importance. The GoogLeNet has spawned numerous iterations, the most recent of which is Inception-v4 [34].

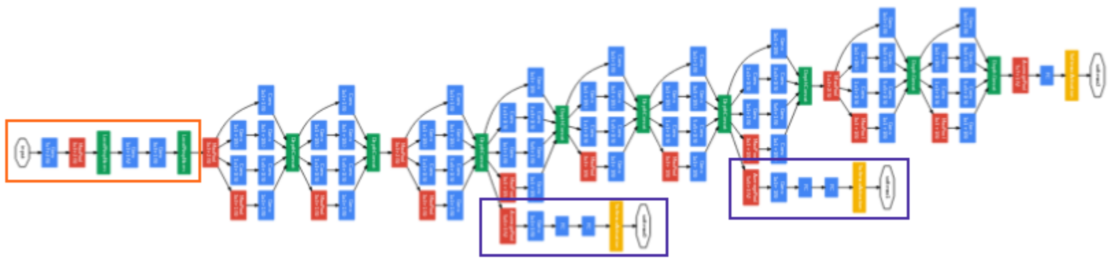


Figure 2.11: GoogleNet architecture [6]

- VGGNet: The VGGNet, a network created by Andrew Zisserman and Karen Simonyan [7], came in second place at 2014 ILSVRC. Its key contribution was to demonstrate that network depth is an important factor in achieving good performance. They created a range of VGGNet versions with varying depth sizes. The important thing in this

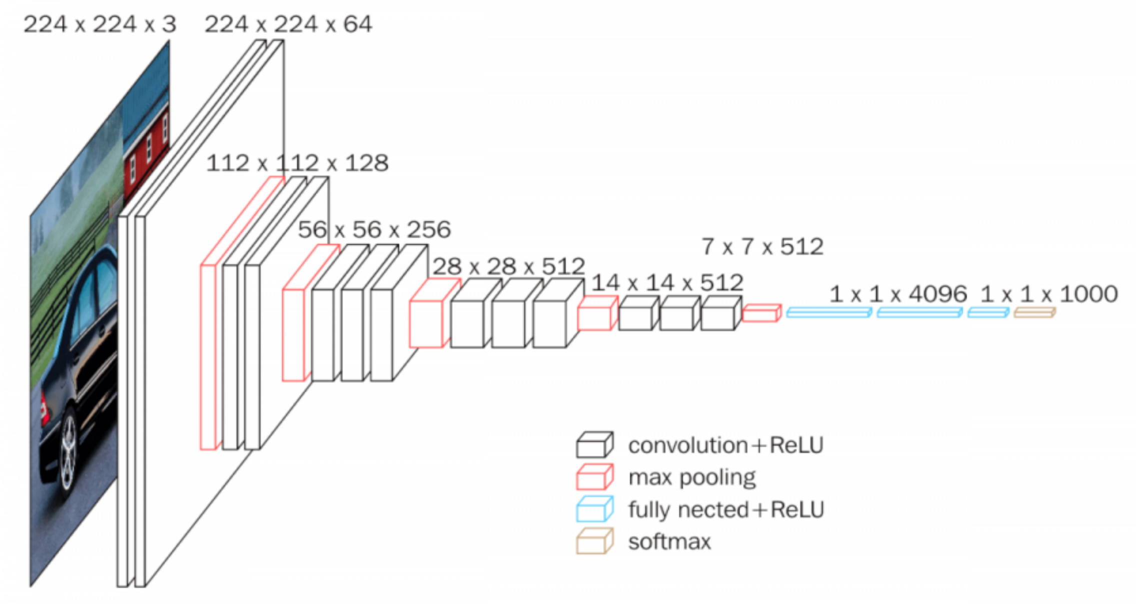


Figure 2.12: VGG16 architecture [7]

model is that in addition to its ability to detect objects in images, the model weights are available for free and can be used at models and applications for detecting objects in images. Their best network includes 16 convolutional FC layers, and displays an extremely homogeneous architecture. This network performs 3x3 convolutions and 2x2 pooling throughout the whole process. The VGG16 model is also used in the experiments of this thesis. The VGGNet has the disadvantage of being more expensive to assess and evaluate, while also requiring 140 million parameters and a large amount of memory.

- ResNet: The winner of 2015 ILSVRC was created by Kaiming He et al, namely Residual Network (ResNet). They implemented a new architecture that used 'skip connections' and batch normalization extensively. These skip connections are very resemblant to RNN elements. The architecture also lacks FC layers at the network's end. They were able to train a NN with 152 layers (while maintaining a lower complexity than VGGNet) using this technique.

- ZFNet: The winner of the 2013 ILSVRC was the ZFNet [35], a CNN architecture created by Rob Fergus and Matthew Zeiler. By tweaking the architecture hyperparameters, ZFNet was an improvement of AlexNet,by raising the volume of the middle convolutional layers, and reducing the stride and filter size on the first layer [35].

2.7 Overfitting

In machine learning problems, generalization is critical. A strong machine learning model should be able to generalize well from training data to any unknown data from the problem domain[8].

One of the key problems of ML algorithms is that there is no minimization of the desired error, i.e. the error which the network generates each time new entries are introduced into it. Therefore, the most important quality of a multilevel network is its ability to generalize in any new situations it comes against. In order to reach its maximum generalization capacity, a network is necessary to choose carefully on the characteristics of the data set: the size as well as the number of weights in the set. In fact, every network is trained to reduce the error in training data, which is not the same as reducing its gradient error. The issue that is created is the problem of overfitting/overlearning.

Essentially, the concept of overfitting is the phenomenon according to which, the network is highly trained in training data (input-output), with resulting in the network error being too small, but becoming too large for data in which the network has not been trained.

The difference between training and validation accuracy shows how much overfitting has occurred, as can be shown in Fig. 2.13. The blue validation error curve indicates that validation accuracy is very low relative to training accuracy, meaning that there is a lot of overfitting. It's worth noting that the validation accuracy will start to deteriorate after some point. The following are the key measures for minimizing overfitting:

- Data Augmentation: Of course, the first move is to gather more data. In certain instances, however, it is not feasible. As a next step, the use of data augmentation should be performed, which is always advocated. Data augmentation involves techniques such as random image rotation, flipping, cropping and applying some color filter, among others. These methods are solely used on training data; they are not used on either validation nor testing sets.

- Regularization or Dropout: Depending on each problem, the capacity and learning ability of a network can be in some cases either huge or minimal. In an effort to adapt the network capacity to each problem, the solution is given by adding and subtracting hidden layers and its neurons. These procedures are usually avoided and therefore the regularization process is preferred. In addition, in most of the neural network challenges

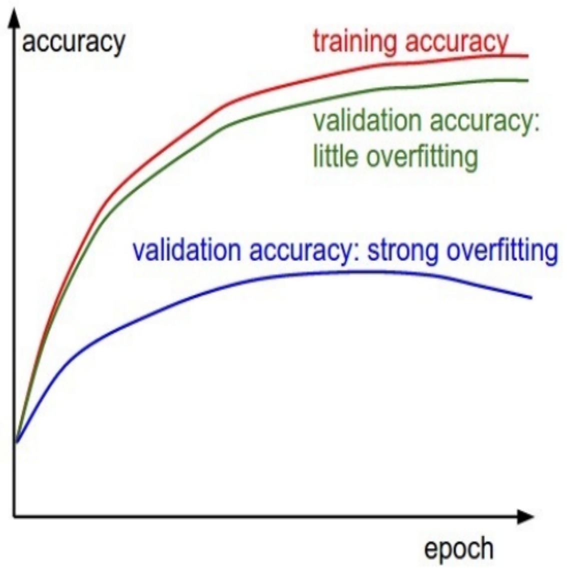


Figure 2.13: Overfitting [8]

there is an infinity of solutions, which is not desirable. For this reason, in order to impose small values and smoothness in weights, the method of regularization is used.[8]. Dropout is a popular regularization technique. During each training epoch, it 'ignores' (eliminates) a random sample of the activations. The procedure prevents any neuron from relying excessively on the output of any other neuron by dropping out neurons at random, forcing it to rely on the population behavior of its inputs instead.[8]. Since it reduces interdependent learning among neurons, this technique helps the model generalize better to unknown data and prevents overfitting.

- Reduce complexity of architecture: The third[8] alternative is to decrease the complexity of the network by reducing the number of parameters. The way to accomplish this is by reducing either the number of layers or the size of the layers. When a small size of training data is combined with a large number of parameters, the model may be forced to map undesired data qualities, such as noise. In contrast, if the model has a small number of parameters, it will be unable to recreate complicated problems. In reality, the more parameters you have, the more data you'll need to train properly. As a result, when data is limited, reducing the number of parameters improves the model's ability to learn and the generalization to unknown data.

2.8 CNN Transfer Learning

In reality, training an entire CNN from the start (with random weights initialization) would be uncommon due to the rarity of a sufficiently large dataset[36]. Instead, it's popular to pretrain a CNN on a broad dataset (such as ImageNet) and then use it as an initialization or a fixed feature extractor for the task at hand. The following are the two main scenarios for transfer learning:

- Fixed feature extractor CNN

  Feature extraction is the method that, using the representations learned by the pre-trained network, extracts characteristics from the new examples[36][10]. These characteristics are then passed through a pretrained classifier suitable for our case. Feature extraction consists of taking the convolutional base from a pre-trained network, pass the the data through the network, and train a classifier on top of the CNN, a layer above the output of the convolutional base.[10]

  [36].

- CNN fine tuning

  This second technique is to continue the backpropagation and not only substitute and retrain the classifier on top of the CNN on the new dataset, but also to fine-tune the weights of the pre-trained network[36][10]. It is possible to train the last levels of the CNN after the classifier has been trained. This is done because if the changes in the weights of fully connected levels are too drastic, they will destroy the representations that have been learned by the CNN layers.

The size of the new dataset, the resemblance to the original, as well as the complexity of the problem are the most important factors in determining the type of transfer learning to use.

Chapter 3

Databases and Preproccessing

3.1 Introduction

Due to the fact that DNNs need a significant training data amount to prevent overfitting, gathering sufficient data is critical before constructing a deep learning emotion classifier. However, there is not enough annotated video data in current facial expression databases to train well-known NNs with deep architectures that have shown promise in other recognition tasks, such as object recognition [37].

A database can definitely provide a dependable platform for testing and evaluating classification algorithms. Typically, the algorithms are tested against specific video or image databases. Such databases are made up of samples and the ground truths that go along with them. Due to uncontrollable and varying circumstances, there may be some variations in the content of images, such as different topics, age, lighting, gender, ethnicities, and textures [37].

Using a single FER database to construct a classifier, however, is rarely enough to model real-world data variability [37]. To ensure objectivity, we will have to use a combination of several databases or calculate the developed framework's cross-dataset efficiency.

3.2 Classification of Databases

Even though we have a variety of databases that meet many distinct requirements, it should be important to remember that the database's content is influenced by a variety of factors [38]. The participants' selection, elicitation process, and data format all have a significant impact on the final model's performance. Participants' cultural and social backgrounds, even their moods at the time of the experiment, may influence the database's output to target a section of the population [39].

3.2.1 Types of Databases

Depending on how the data was obtained, databases will take several different types. There are two types of facial expression databases currently available: a) static databases, which contain static face images and their respective labels b) video databases, which consist of video sequences that allow temporal information to be considered via video frames.

The most popular and oldest database form is static databases. These databases usually include a large amount of people and, additionally, a large sample size. Although it should be relatively simple to find one database that is appropriate for our task, emotional categories are severely restricted, because static databases can only detect six primary emotions or smiles/neutral.The majority of earlier facial expression databases include only frontal portrait photographs captured with basic RGB cameras. By using various angles and occlusion (e.g glasses, hats), some modern databases attempt to construct collection methods which integrate more realistic data.The Multi-PIE [40] and MMI [3] databases, which were among the first ones to use different view angles, are prime examples.

Regarding the case of video databases, scientists have attempted to recognize emotions using gestures, vocal context, head movement, and generally any other form of feature that describes body language[38]. Furthermore, several video databases attempted to collect subtle emotional changes (also known as micro-expressions), which are not easily detected in still images. Video is also by far the most practical medium for recording both induced and spontaneous emotions[38] This can be due to the absence of distinct beginning and ending points for non-posed emotions. That being said, video databases usually have a limited number of participants, which presents a challenge when training the DNN models.

3.2.2 Methods of Elicitation

How to elicit various emotions from participants is a significant decision to make when collecting data for emotion recognition databases. The three main categories of audiovisual databases listed in the literature are:

1. posed
2. induced

3. spontaneous (during an interaction)

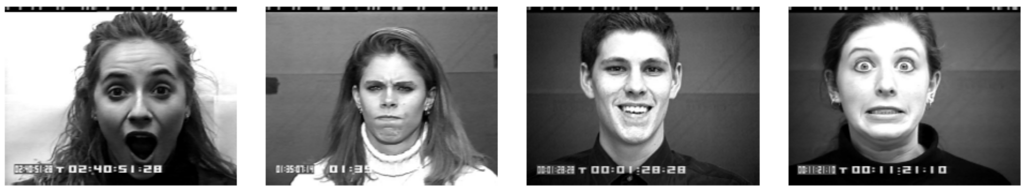
Participants in the posed datasets are normally real actors. They are instructed to accurately depict each emotion [38]. Since posed facial expressions are the easiest to collect, most facial emotion databases, particularly the early ones, are entirely made up of them. CK+ and MMI are two well-known posed databases [9] [3]. However, since forced emotions are often exaggerated, this type of database is actually the least reflective in comparison to realistic emotions. As a consequence, when evaluated with real-world data, human expression analysis models, generated using posed databases, often produce poor results.

In terms of induced databases, films, stories, music, or some combination of them, are often used to elicit emotional responses in subjects. Since the participants are normally exposed to audiovisual stimulations in order to evoke and depict real emotions, this form of elicitation reveals more genuine emotions [41].Due to the limitations of posed databases, induced emotion databases have become more popular. Since the models are more realistic and not hindered by over-emphasised and artificial gestures, their performance in real life is greatly improved. Nonetheless, the participants are filmed in a well-organized lab setting, similar to how they are filmed in posed databases.

Spontaneous emotion datasets are thought to be the most accurate representations of real-life situations.The key disadvantage of this form of elicitation is that audiovisual data consisting of impulsive emotions is quite hard to obtain and label. The reason for this disadvantage is that emotion expressions are uncommon and frequently connected with a complex contextual structure [42].

3.3 FER Databases

The following are a couple of the most famous, publicly accessible FER databases:



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

Figure 3.1: CK+ database [9]

1. CK+: This is an extension of the original version Cohn-Kanade Dataset (CK +)[9][10] first published in 2000. It is one of the most renowned databases for the development of facial expression analysis algorithms. This database is available for research purposes in 2 main editions, and a third edition is currently in preparation. The first edition includes 486 arrays of images that contain facial expressions in posed format from 97 subjects. The second edition is extended to 593 arrays of images, also containing facial expressions in posed format, among 123 subjects. Each expression includes images that start from a neutral shot and end at the peak of the expression of emotion, through a series of images. The third edition is scheduled to be released soon. Its current form includes images with subjects posing aligned with the camera. The updated version also includes footage with subjects having an inclination of 30 degrees in the video.

2. MUG: The MUG Facial Expression Database is made up of images of 85 people making various facial expressions. There are 34 Caucasian women and 50 Caucasian men in the database, all between the ages of 20 and 35. The participants took a seat in front of a single 19-frame-per-second camera. Each image had a resolution of 896 x 896 pixels and was saved in jpeg format. The MUG database is divided into two parts. As the participants were asked to portray the six basic emotions, the first phase contains posed facial expressions. Each sequence comprises 50 to 160 frames and begins and ends at the neutral state.The aim was for the subjects to correctly mimic the basic expressions, and in order to do so, they were given a brief lesson on the basic emotions prior to the recordings. The sequences that accurately imitated the expressions were chosen for inclusion in the database. As a consequence, there are 1462 sequences available in the first phase.The subjects were filmed while watching a video that was created to elicit emotions in the second phase. The participants were conscious that they were being recorded on video. In contrast to posed expressions, the purpose of this phase of the MUG database was to basically elicit genuine expressions, located in the laboratory setting. Apart from the six basic emotions, the subjects show a variety of emotions and emotional attitudes. These image sequences, however, have yet to be labeled.

3. ISED: The Indian Spontaneous Expression Database for Emotion Recognition (ISED) contains both male and female Indian participants' spontaneous expressions. There are 427 video clips, segmented, from 48 participants in the database. The video clips range in length from 2 to 11 seconds. Passive elicitation was used to evoke the emotions

by viewing emotion-inducing videos. A hidden camera captured the subjects' facial expressions as they were left alone in the experimental room. The participants weren't told the actual intent of this experiment ahead of time, but were able to watch the videos to evoke natural emotions, alone in the experimental room. Subjects were often permitted to watch at their leisure without further conditions in order to avoid suspicion. Finally, four trained decoding machines annotated all of the videos, tagging the six basic emotional expressions as well as the corresponding emotional intensity on a six-point scale for each video (from 0 to 5).

4. MMI: The MMI database was created with a goal of evaluating algoriths related to facial expression recognition, and consequently evaluating the contribution to the optical media community of a large volume of optical data[10]. The MMI database has been further developed as a direct web application available to specific members with the corresponding authorization. It can be directly manipulated in a way that allows for quick access and searching of the available images. This database contains 2900 high quality videos, which are then converted to quality images from 79 subjects. The MMI database approaches the issue of recognizing emotions not only through categorizing them into 7 fundamental categories, but includes additional information on the activation of specific facial muscle groups and other descriptive markers.

   The database contains 19 male and female subjects ranging in age from 19 to 62 years old and of European, South American or Asian ethnicity. The participants were asked to show 79 different expression sequences, including the six most common emotions.[10]. This database has static images as well as arrays of images with the faces in anfas and profiles. All video sequences were captured at a 24 frames per second frame rate. The sequences vary in length, lasting anywhere from 40 to 520 frames [3].

3.4 Dataset

The CK+ and MUG databases were combined for this thesis in order to establish a good amount of training, testing, and validation sets. Both databases have samples in the training, testing, and validation sets. Therefore, in order to prevent prejudice, the same participant does not appear in any two of them. Every single one of the videos was edited to have the following format: each video sequence begins with a neutral expression but then is ending with the most

Databases	Elicitation	EmotionClass	Participants	Format
CK+	posed	6 basic emotions(BE) + contempt	123	video
MUG	posed,induced	6 basic emotions	86	video
MMI	posed	6 basic emotions	19	static,video
ISED	spontaneous	4 basic emotions	50	video

Table 3.1: Databases In Short

intense -in terms of emotion- frame. Since the CK+ database only contains grayscale data, they were also converted into grayscale images. The dataset's structure is defined in Table 2.2.

By observing table 3.2 below, the samples of certain groups far outnumber those of others (eg. fear vs happiness). This is referred to as the class imbalance problem, and it occurs regularly in practice.

Number of videos	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Total
Training set	80	112	70	152	84	133	631
Testing set	25	31	25	58	28	35	202
Validation	4	3	3	3	3	5	21

Table 3.2: Construction of Dataset

3.5 Pre-Processing

The function and significance of data pre-processing, particularly regarding NN data, could be explained in a variety of ways. While pre-processing data is not needed in many cases from a mathematical standpoint, it actually has a major impact on the NN models performance because it greatly improves training of the network. Furthermore, the type of pre-processing used on data is critical in assessing the performance of a practical application based on NNs. As a result, selecting the appropriate pre-processing steps is a crucial factor for the network's ability to distinguish the relationship between inputs and outputs [10][43].

For this thesis, the developed FER framework is focused on CNN architectures that have been pre-trained. Our database was used in particular to fine-tune them and allow them to classify emotions [10]. In order to comply with the database's pre-training requirements, pre-trained architectures are necessary to follow a particular data format. This is why some of the pre-processing steps below, such as resizing, normalization, and RGB conversion, are essential.

3.5.1 Frame Selection

As previously mentioned, each video in the database follows a particular format. According to this specific format, the most extreme, in terms of emotion, facial expression appears in the last frame whereas neutral expressions are appearing within the opening frame of each video. As a result, not all frames are appropriate for the training process.

3.5.2 Face Detection

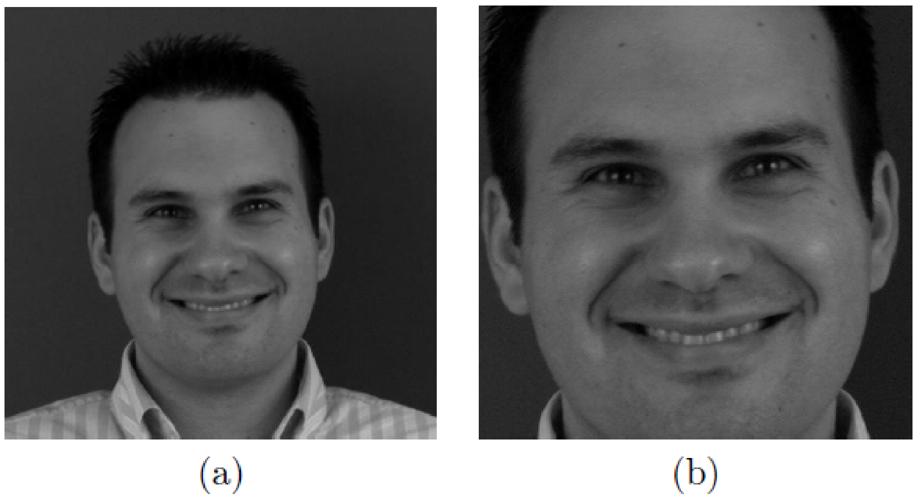


Figure 3.2: Before and After: Cropping with Viola-James [10]

As a preliminary preprocessing step before presenting an image sequence to a FER system, it is important to identify the regions of the face[10]. The aim is to retain just the pixels that make up the face while discarding any pixels in the background. We can minimize the dimensionality of the input data by removing irrelevant information from the frames while retaining the important information for the training phase.

There are many approaches that can be used to accomplish this goal. The Viola–Jones face detector is one of the most common which was also used in this thesis. The image integral, classifier learning with AdaBoost, and the attentional cascade structure are the three main steps in the algorithm[44].

The Viola-Jones face detection algorithm starts by converting the input image into an integral one[10]. The integral image (also known as a summed area table) is a method of computing the sum of values in a rectangle subset of a pixel grid, in a quick and efficient way[10].

By utilizing four values out of integral images, the pixel sum of any image area can be calculated. The next step is to use Haar-like featuresin order to extract features (Fig. 2.3). The pixel sum of the darker rectangle minus the pixel sum of the lighter rectangle yields a Haar-like feature value.

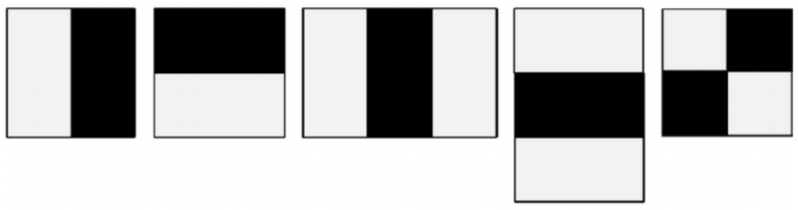


Figure 2.3: Five Haar-like features. Image from [3]

Figure 3.3: Haar-like features [10]

An ML approach can be used to learn a classification function, given a feature set and a training set of positive and negative images. To pick a limited collection of features and train the classifier, the Viola Jones employs a derivative of AdaBoost. Each weak classifier is a threshold on a single Haar-like rectangular feature, and an AdaBoost classifier is made up of a weighted sum of several weak classifiers[10].

Lastly, the cascaded classifier is made up of stages, and each one of them contains an AdaBoost strong classifier. Every stage's goal then is to decide whether a particular sub-window (SW) can be described as certainly not-a-face or maybe-a-face. A SW is automatically removed when it is labeled as a non-face by a given stage. A SW labeled as a-maybe-face, on the other hand, is moved on to the next stage in the cascade. As a result, the more stages a SW goes through, the more likely it is to include a face[10].

3.5.3 Histogram Equalizations

By converting the intensity values in an image, the Histogram Equalization algorithm improves the image contrast. The intensities on the histogram can be distributed in a better way, with this correction. This enables regions with low local contrast to achieve an increase in contrast. This is accomplished by histogram equalization, which is essentially a technique that adjusts the pixel volume of an image. The initial histogram is redistributed accross the whole scale of the target image's discrete values (intensity values), with the purpose of spreading out these discrete values and enhancing the contrast.

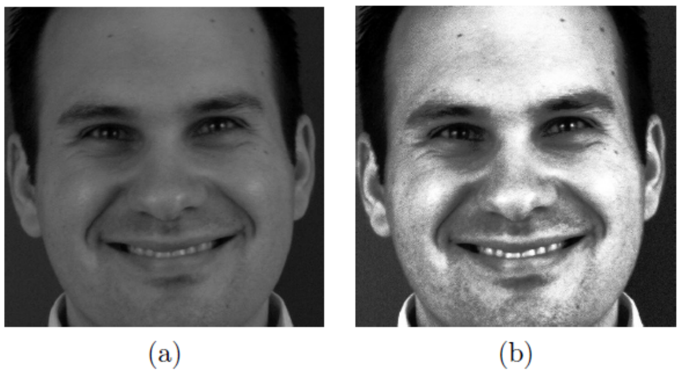


Figure 3.4: Histogram Equalization (Before And After) [10]

The Histogram Equalization algorithm is implemented in the following steps.

1. For the image, render a histogram. Let n_i be the number of occurrences of gray level i in a discrete grayscale image x. The probability of a pixel of level i appearing in the picture is

$$p_x(i) = \frac{numberofpixels(intensityi)}{totalpixels} = \frac{n_i}{n}, 0 <= i < S \tag{3.1}$$

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

2. Calculate the CDF histogram (cumulative distribution function). The definition of CDF corresponding to $p_x(i)$ is :

$$cdf_x(i) = \sum_{j=0}^{i} p_x(j) \tag{3.2}$$

,which is also the image's accumulated normalized histogram.

3. The cdf of the new image y has to be calculated. To generate the new image with a flat histogram, we'd like to construct a transformation of the form $y = T(x)$. For any constant K, such an image would have a linearized cdf across the value range, i.e. $cdf_y(i) = iK$. The properties of the cdf allow us to perform a transformation like this:

$$cdf_y(y) = cdf_y(T(k)) = cdf_x(k) \tag{3.3}$$

, where K is between [0, S]

4. For each gray value in the picture, assign a new value. As we utilized a histogram normalization of x, T maps the levels between [0,1]. To get the final image version y', apply the following simple transformation to the result:

$$y' = y(maxx - minx) + minx \tag{3.4}$$

This transformation is aiming to depict the values back into their previous span.

3.5.4 Linear Scaling and Resizing

The pre-trained model used here has a fixed input size of 224x224 features due to its architecture. As a result, all frames were resized to 224x224 pixels after face cropping. Furthermore, grayscale image pixels are usually integers ranging from 0 to 255, with 0 representing black and 255 representing white[10]. That being said, data must be scaled in NN architectures to suit the range of input neurons. This usually falls between -1 and 1 or 0 and 1. As a result, the frames' pixels were converted into a range of 0 to 1 using simple linear scaling.

3.5.5 RGB Conversion

3-channel RGB images are needed by the CNN architecture. As a result, after performing all of the previous pre-processing steps on grayscale video frames, they must be converted to 3-channel RGB format. Since that is clearly not feasible, the sole easy option was to transfer each pixel's value into through the rest of the channels to produce 'false' RGB images[10].

Furthermore, the data should be normalized similarly to the data used for the pre-training of the network. The mean and standard deviation are used to normalize each image channel, as stated below:

$$input(channel) = \frac{input(channel) - mean(channel)}{std(channel)} \tag{3.5}$$

,with std = [0.228, 0.223, 0.224], mean = [0.484, 0.455, 0.405]

Chapter 4

Methodologies

This is where we are going to present the framework for our automatic-FER. The basis of the suggested framework is the application of transfer learning -fine tuning- to a pretrained CNN and the combination of per-frame-predictions for classifying the prevailing emotion in a video sequence[10]. Each video frame's facial expression receives information determined by the fine-tuned model. The per-frame information from all frames is then combined during the testing process to produce a final prediction for the entire video. In the following pages, several different framework implementations are discussed and compared.

For the implementation of this framework, PyTorch, an open source deep learning platform, was used. The training phase took about 2 days on a machine with an NVIDIA GeForce GT 1080 GPU, depending on the architecture. It important to highlight the usage of the VGG16 architecture as the pretrained CNN. VGG16 was chosen due to its straightforward architecture and the fact that its pretrained version is accessible publicly.

A combination of CK+, ISED, and MUG databases created the dataset used for training, validation, and testing sets[10]. Chapter 3 contains more detail about the dataset and the preprocessing steps.

4.1 Methods of Testing

To eliminate bias, the participants in the testing films are not the same as those that were used to train the model. As with every video in our dataset, every testing video has a neutral expression frame at the beginning and the most intense one at the end. As a result, we will engage with the frames which display facial expressions and not ones with neutral faces frames, when we're evaluating our model. Per-frame class scores are calculated using the fine-tuned model:

$$s = \begin{bmatrix} s_1 \\ s_2 \\ . \\ . \\ s_6 \end{bmatrix} \tag{4.1}$$

Then, using two separate methodologies, per-video predictions are produced.

1. Majority Vote: Initially, for every frame, a prediction is formed by picking the emotion with the highest score at the softmax layer's output $s_1, s_2, ..., s_6$. We apply majority voting to $p_(n/2), ..., p_n$ to identify which emotion has more occurrences provided the frame predictions $p_1, p_2 ..., p_n$, where n represents the number of frames. Because in each video sequence, only the second half contains facial expressions, we just use these frames.

2. Mean Scores: The video prediction is found using the class scores of each frame in the second method. The mean of the emotion scores throughout the video's later half frames(p/2 to p) is calculated, specifically:

$$meanscores = \begin{bmatrix} mean(s_1) \\ mean(s_2) \\ . \\ . \\ mean(s_6) \end{bmatrix} \tag{4.2}$$

Then, for each video, we choose the emotion class having the highest mean value in order to determine the final prediction for the emotion.

4.2 VGG16

The VGG16 architecture is primarily made up of convolutional, max pooling, as well as FC layers, and it was being trained to classify objects into 1000 separate classes. Three FC layers follow a stack of convolutional layers: the first two each have 4096 nodes, while the third conducts 1000-way classification and hence has 1000 nodes. The soft-max layer is the final layer, and it calculates the probability of every class being predicted. Its image, then, is routed through a convolutional layer stack with small receptive field filters (3x3)[11]. The stride is set to 1 pixel and the spatial padding of each convolutional layer input is set to 1 px, preserving spatial resolution after 3x3 convolutional filters are applied. Maxpooling is done with a stride of 2 over a (2x2) pixel area. The ReLU non-linearity is present in all hidden layers.

4.2.1 Optimisation of VGG16

The model was first trained using ImageNet, a massive public image repository. This database was intended to help people solve object recognition problems. It consists of 1000-classes images divided into three groups: The first one is the training set with approx. 1.2 million images, then testing is consisted of approx. 100 thousand, and lastly validations set has 50 thousand. As a result, instead of utilizing randomly initialized weights, we base our training model on the pre-trained model. This is definitely a better arrangement than random weights as this enables us to reuse the network's low-level visual data, which were acquired in the initial input. The fundamental idea is once the model has been fine-tuned, it should then be in a position to accurately categorize emotions from facial expressions[10].

Once the pre-trained model is loaded, the next step is to reduce the last FC layer size from 1000 to 6, as our problem has six emotion classes (Fig. 4.1). Random weights are then applied to the final FC layer. It's worth noting that the optimisation process isn't extended to the entirety of the network. Overall, the starting layers of CNNs learn primarily about edges and sharp corners, whilst the subsequent ones merge these edges to build and learn somewhat more complicated forms. Since the starting layers are in the proccess of learning the whole dataset of ImageNet, these layers provide the most detail about the most fundamental objects[10].

As the starting layers store knowledge regarding specific shapes which might be relevant towards identifying emotions, just the last three FC layers are modified during the fine-tuning epochs in our simulations. Furthermore, as we are not training all the layers when the fine-tuning occurs, the computational cost is significantly reduced.

4.2.2 HyperParameters

In ML problems, setting the hyperparameters before training starts is critical. The variables that decide the network structure, as well as the ones which determine the training of the network, are known as hyperparameters. The developers of VGG16 have predefined the

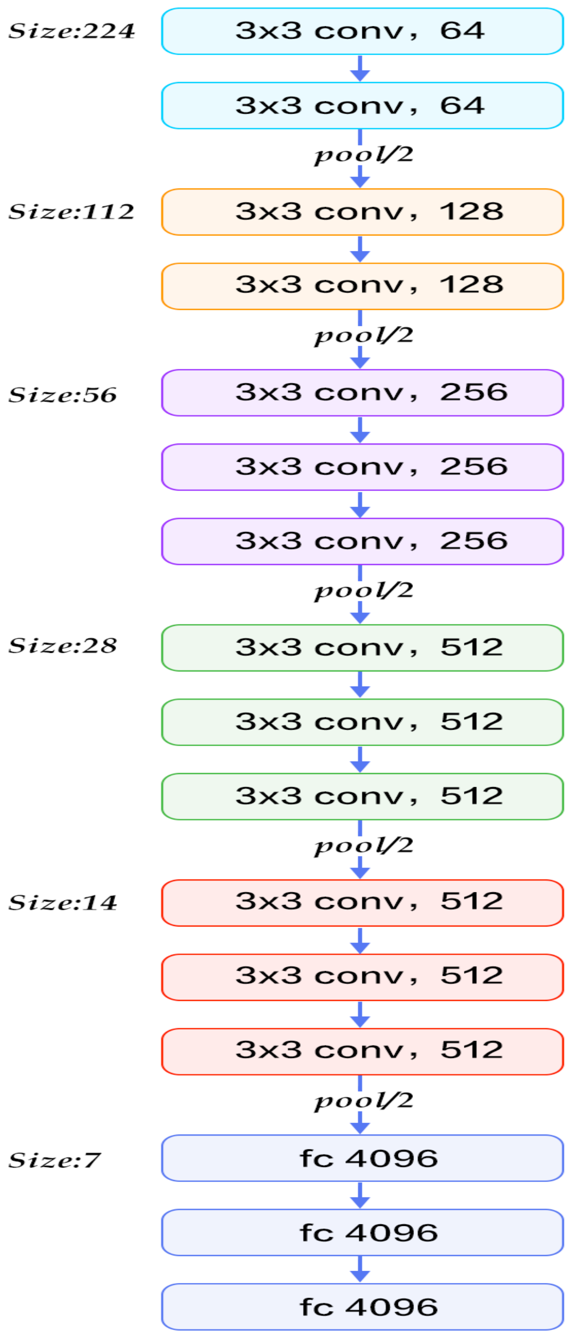


Figure 4.1: VGG16 example (last FC layer is changed to 6) [11]

learning rate	0.005
batch size	20
optimizer	Adam

Table 4.1: HyperParameters

variables relevant to the network structure in our situation (e.g. activation function: ReLU). The values of the main network hyperparameters are shown in Table 4.1. After experimenting with various combinations of learning rate, batch size, and optimization method, the following combination proved to be the most successful:

The rate with which the parameters of a network are adjusted and updated is called learning rate. A low learning rate delays the learning process but allows it to converge smoothly, whereas a higher learning rate accelerates learning but may not allow it to converge[10]. The initial recommended learning rate for the Adam optimizer is 10^-3, which is the number we utilize for our training.

Furthermore, passing the entire dataset into the CNN at once would be impractical, so we break the frames into batches of 30. This corresponds to how many samples the network receives before any parameter update. The entirety of data pre-processing steps are described in depth in Section 3.4, and can be seen in Fig. 4.2.

Additionally, selection the loss function is critical in achieving optimal and rapid outcomes. The loss function is a metric for determining how well a prediction model performs in terms of predicting the expected outcome. The cross-entropy loss function was used to measure the performance of the classification model (during the training epochs) for our experiment.

The main implementations of our methodology are presented in the following sections. Scikit Learn [45], an open source software tool, was used to create the confusion matrices on the following pages.

4.3 First Implementation

Using the hyperparameters given above, the first implementation attempted to optimize the three final fully-connected layers of our VGG16 model [10]. The average loss of all batches is calculated at the completion of each epoch, in order to check convergence. During

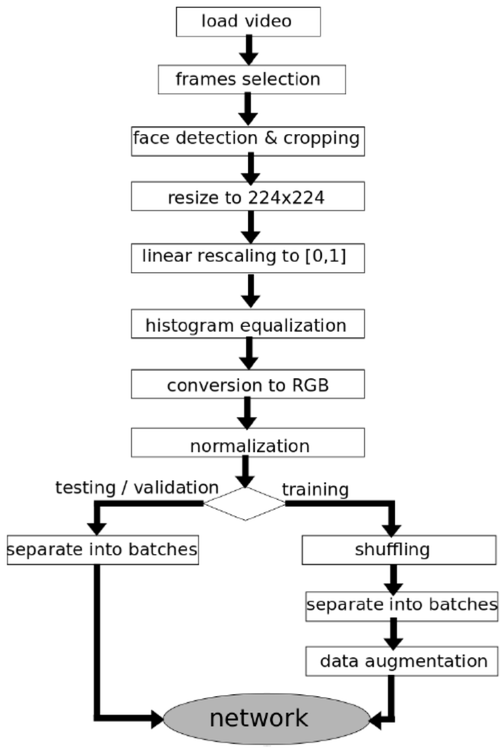


Figure 4.2: Preprocessing steps

the first epochs, the loss appears to reduce dramatically, and then it appears to converge.

The model was assessed on the training and validation sets at the conclusion of each epoch to see if over-fitting occurred during training, as can be seen in Fig. 4.3. At this level, the evaluation of the training and validation sets is happening by frame classification rather than video. The goal is to end the training process as fast as possible. We need to stop at a point when our model displays high precision on the validation set's unknown data, but before any overfitting happens during training, as seen in the decline of validation accuracy and a rise of training accuracy [10]. As a result, right after the 170th epoch, we terminated training in order to evaluate the model on the testing set.

Fig. 4.5 depicts the outcomes of both evaluation methods. As can be observed, when compared to the majority voting method, the second method by using mean scores achieves better accuracy in the results.

Some classes appear to have much higher accuracy than others. When compared to Fear, all other emotions perform better (Happiness, Surprise, and Disgust). This discovery can be explained by the presumption that a few emotions are more distinct, while the other emotions have facial expressions that are similar [10].

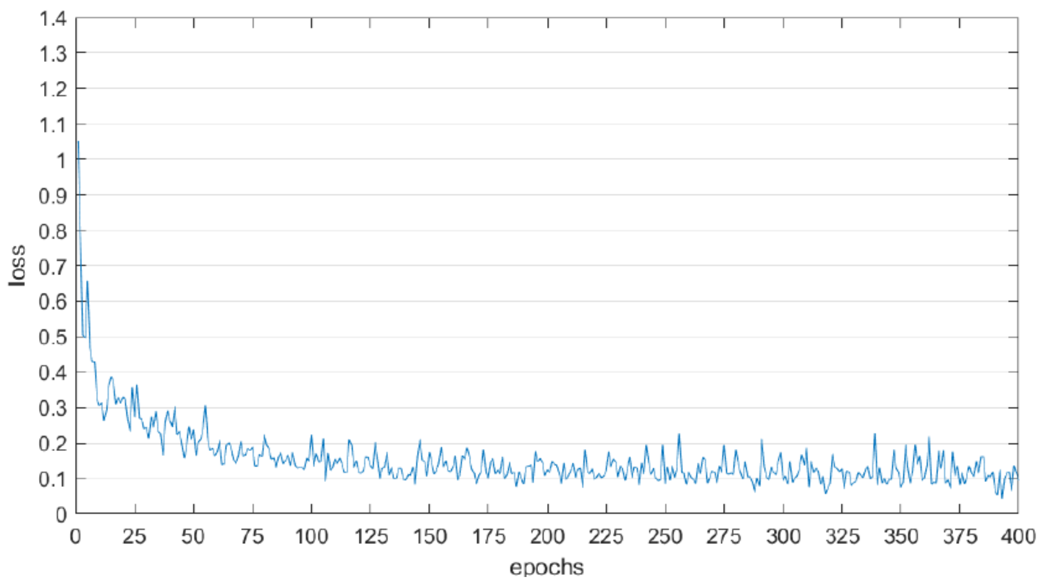


Figure 4.3: Loss throughout Epochs

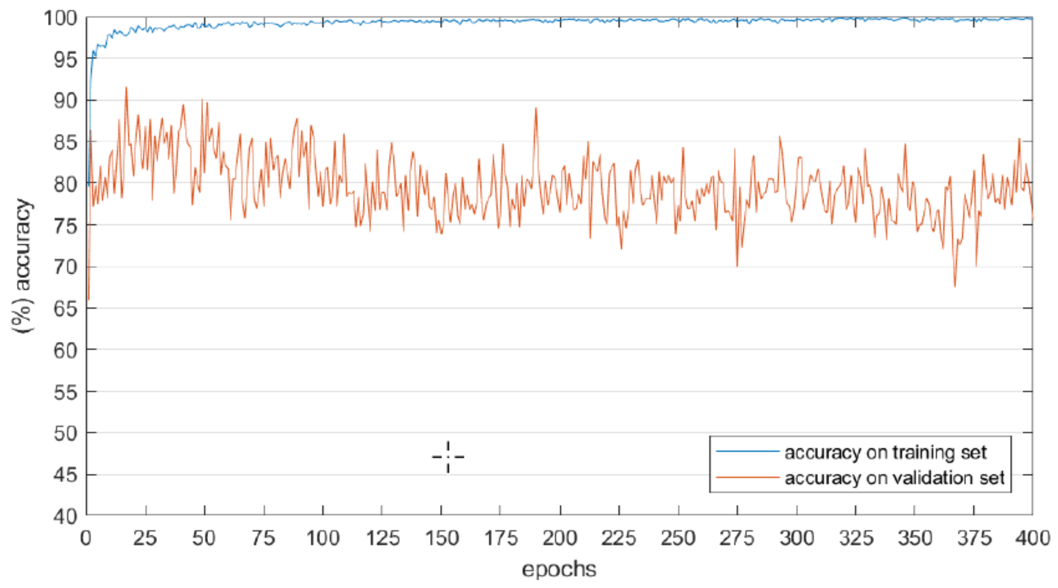


Figure 4.4: Model accuracy (training and validation sets)

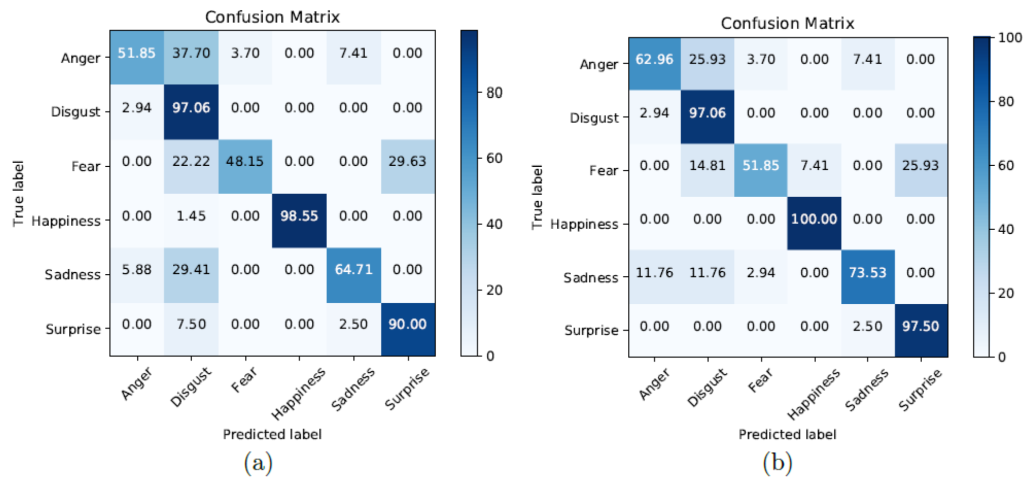


Figure 4.5: Confusion matrices: (a)Average accuracy of 75.02 using the majority voting algorithm (b)Average accuracy of 80.49 using mean scores

4.4 Second Implementation

Just as discussed earlier, having a small capacity training data and a large amount of parameters can encourage the model to map unwanted data properties (eg. noise). As a result, in the second implementation, we modify the VGG16 architecture. In the second FC layer, by reducing the number of parameters we can reduce network complexity. Parameter reduction should aid the model's generalization on unknown data while avoiding overfitting.

The model's last three FC layers are fine-tuned, but all hyperparameters remain the same as in the first implementation. The loss convergence can be seen in Fig. 4.6 once more. The model was assessed on the validation and training sets at the conclusion of each epoch to check for over-fitting during training, which can be seen in Fig. 4.7. The vertical gap between the two graphs, as can be observed, increases dramatically after the 200th epoch. As a result, the training phase ends early after the 170th epoch, allowing the model's evaluation upon the testing set.

Using both assessment methods previously mentioned, we can check how the model performs on the testing set data after the evaluation process.

We can easily observe that the reduction of parameters drastically improves the results, in contrast with the initial (first implementation) VGG16 architecture. It is also clear that some classes are more accurate compared to others. When compared to Fear and Sadness, Happiness and Surprise both score higher. As previously said, this finding is logical since

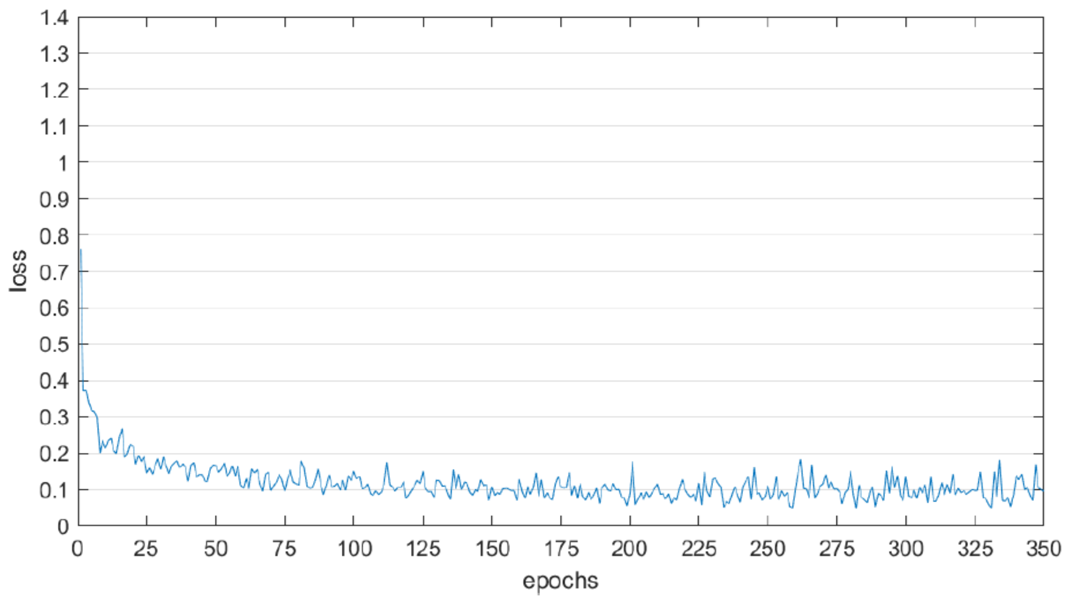


Figure 4.6: Loss throughout Epochs

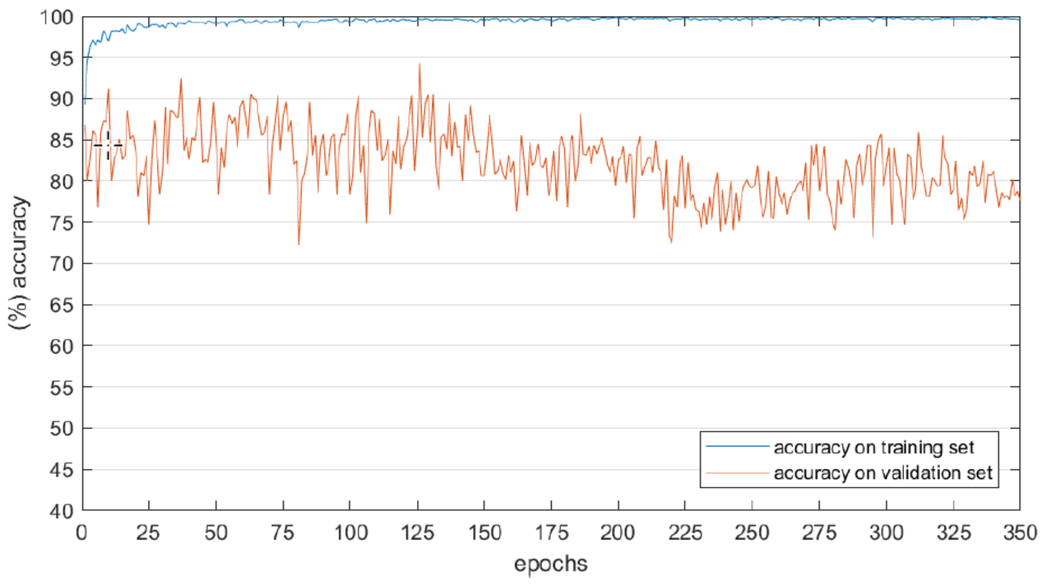


Figure 4.7: Model accuracy (training and validation sets)

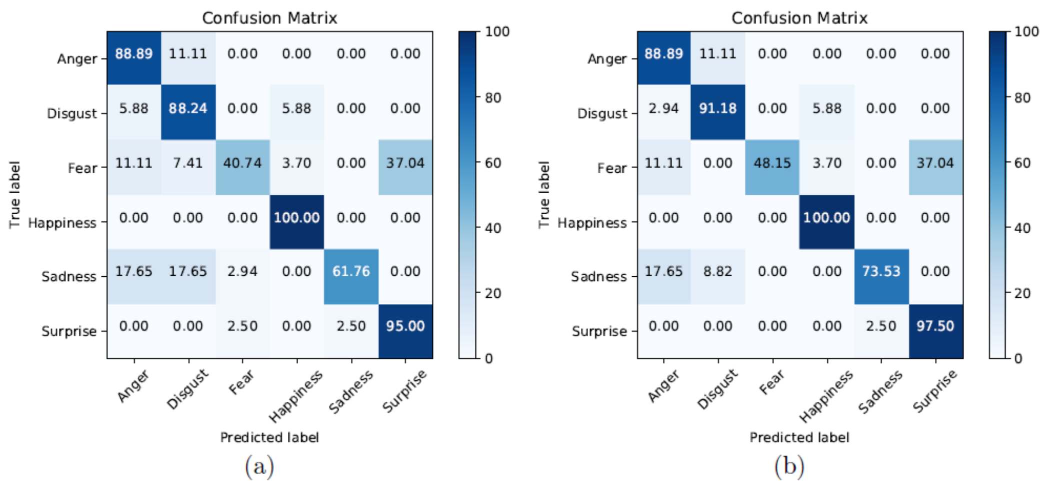


Figure 4.8: Confusion matrices: (a)Average accuracy of 79.12 using the majority voting algorithm (b)Average accuracy of 83.32 using mean scores

certain emotions are more distinct. Fear, in particular, has a poor hit rate, which can be based on the fact that it is a subtle emotion, and can be confused, often, with Surprise.

Further justification for the low accuracy of some classes could be provided. When we look at the training set's structure, we are able to observe that classes end up sharing different amounts of samples, which results in lower presicion for the so-called ”weaker” classes[10]. We will endeavor to resolve this issue in the future implementations.

4.5 Third Implementation

In reality, the majority of actual classification issues have some degree of imbalance. When the instances of a class in the training set outweigh the instances of the other classes, this is known as class imbalance. Datasets which are unbalanced hinder the performance of machine learning approaches since the accuracy level and decision-making are skewed toward the majority classes, resulting in misclassification of the weaker classes samples or, worse, interpreting them as noise.

To combat the implications of class imbalance, several schemes have been developed, involving oversampling instances of the minority class, undersampling instances of the majority class, as well as cost-sensitive learning [10]. We concentrated on cost-sensitive learning after trying all of these approaches, as it was the most effective.

In case some training samples correspond to weaker classes, cost-sensitive learning algorithms account for the cost of misclassification by assigning a larger cost of misclassification.

All misclassifications are handled similarly in normal learning, however this poses difficulty in the imbalanced classification situations, because it seems to add no additional benefit for correctly recognizing the minority class over the majority. This is changed by cost-sensitive learning, which employs a function C(x) which determines the cost for incorrectly classifying an instance of class x. Thus, we can punish minority-class misclassifications more severely than majority-class misclassifications, with a goal of improving the accuracy rate. This modification is made on our network's loss function [46].

As we can observe in table 4.2, there are plentiful Happiness samples in comparison to other emotions. The ”weakest” classes are Anger and Fear, followed by Disgust.

x	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Total
Number of frames	1246	1495	1165	2875	1904	1960	10645

Table 4.2: Frames for each class (training set)

Thus, during training phase, we can utilize the weighted version of the cross entropy loss function to determine the loss value for the same network design, as performed in the first implementation.

As a result, the lower the class's size becomes, the higher the cost rises, leading to increased weights for decreased classes. The average loss of every batch is computed by the conclusion of each epoch in order to measure convergence. The model was also assessed on the training and validation sets for a number of epochs, with the training process ending after the 260th epoch.

Looking at the first implementation's confusion matrices, we observe that the emotion accuracy rate of Fear is the lowest: 47.12 percent for (a) and 52.03 percent for (b). At the same instance, Anger has a poor accuracy rate of 52.15 percent for (a) and 63.28 percent for (b). Regardless of the issue that it is considered a minority class, Disgust has a high accuracy rate of 97.75 percent. The accuracy of the model (testing set) as a result of cost-sensitive learning is shown in Fig. 4.11.

While the accuracy of Fear does not rise and is still heavily misclassified as Surprise in both (a) and (b) confusion matrices of Fig. 4.11, Anger accuracy improves to 77.82 percent. Furthermore, the accuracy of Disgust continues its high level performance. In addition, as compared to the first implementation, the system's average accuracy is improved.

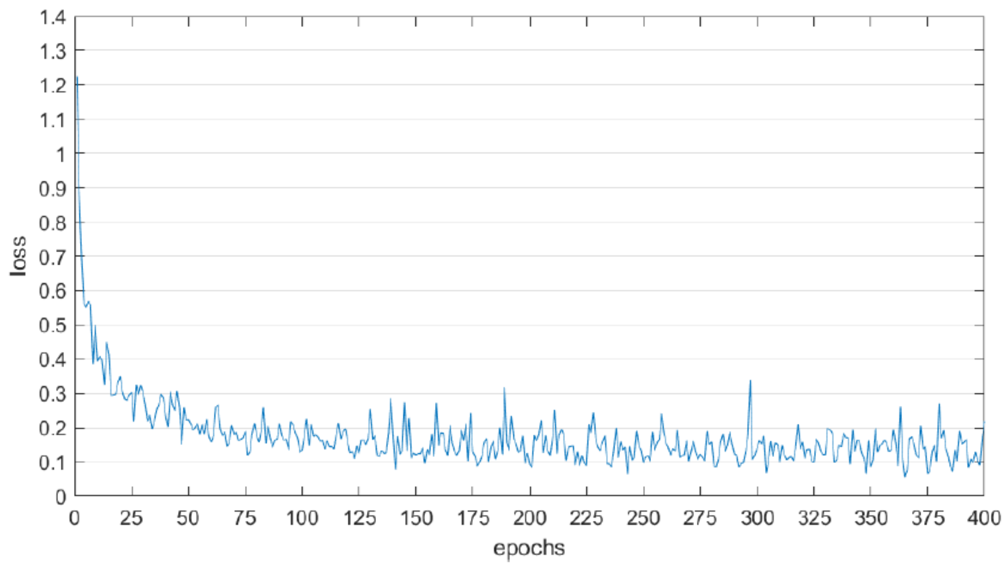


Figure 4.9: Loss throughout Epochs

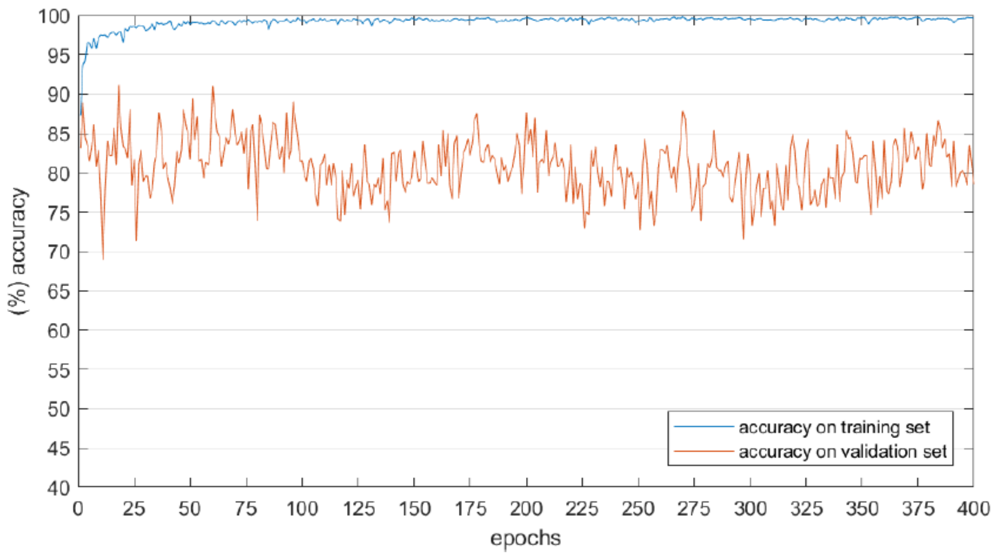
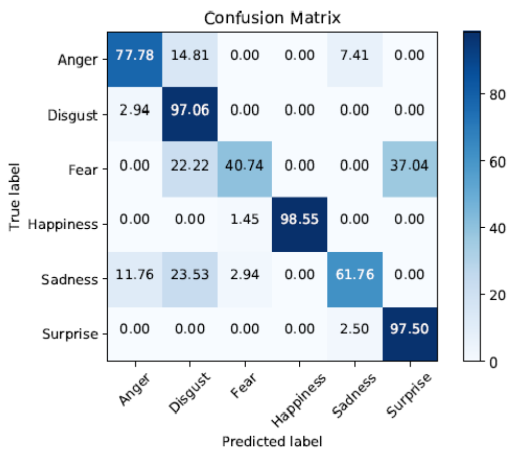
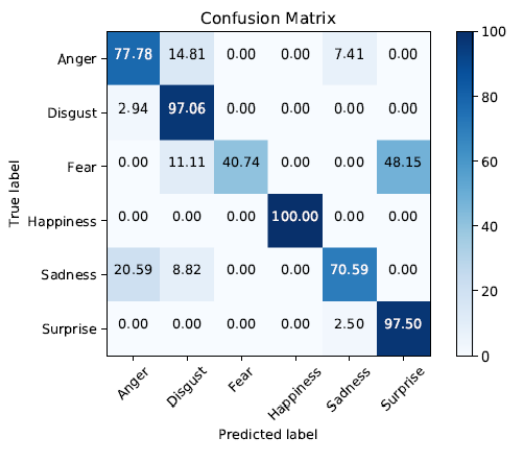


Figure 4.10: Model accuracy (training and validation sets)



	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Anger	77.78	14.81	0.00	0.00	7.41	0.00
Disgust	2.94	97.06	0.00	0.00	0.00	0.00
Fear	0.00	22.22	40.74	0.00	0.00	37.04
Happiness	0.00	0.00	1.45	98.55	0.00	0.00
Sadness	11.76	23.53	2.94	0.00	61.76	0.00
Surprise	0.00	0.00	0.00	0.00	2.50	97.50

(a)



	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Anger	77.78	14.81	0.00	0.00	7.41	0.00
Disgust	2.94	97.06	0.00	0.00	0.00	0.00
Fear	0.00	11.11	40.74	0.00	0.00	48.15
Happiness	0.00	0.00	0.00	100.00	0.00	0.00
Sadness	20.59	8.82	0.00	0.00	70.59	0.00
Surprise	0.00	0.00	0.00	0.00	2.50	97.50

(b)

Figure 4.11: Confusion matrices: (a)Average accuracy of 79.05 using the majority voting algorithm (b)Average accuracy of 81.12 using mean scores

4.6 Fourth Implementation

Following the same direction, we use cost-sensitive learning techniques for the training of the VGG16 architecture (second implementation), in this fourth and final iteration. In following Fig. 4.12, the loss convergence after a number of epochs can be observed. Fig. 4.13 shows that during the first 200 epochs, the validation set accuracy is consistently above 75 percent. This shows that we do not have much of an overfitting effect.

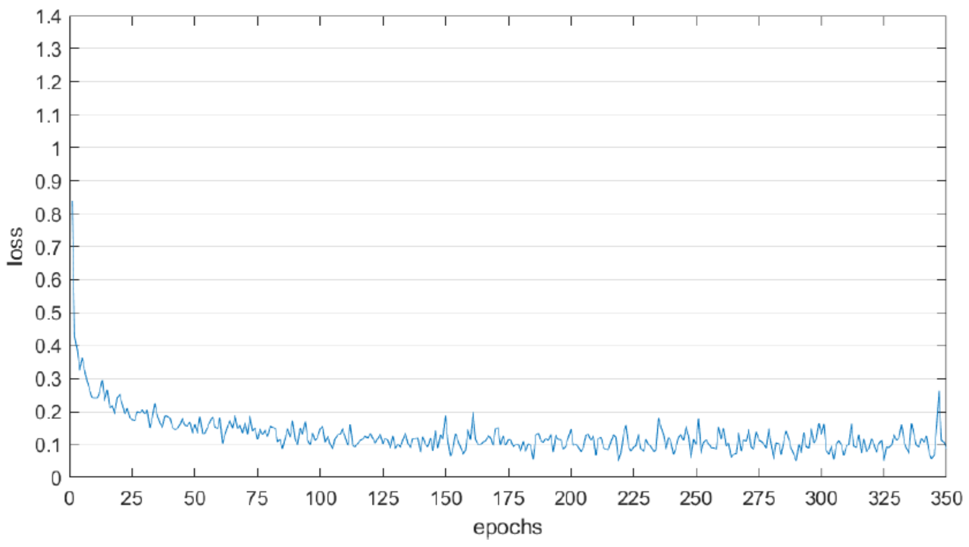


Figure 4.11: Loss value through epochs

Figure 4.12: Loss throughout Epochs

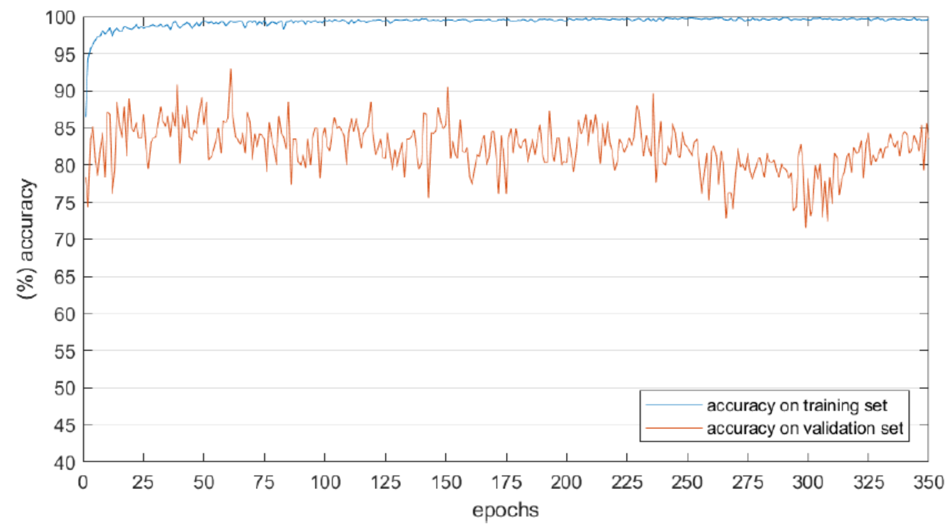


Figure 4.13: Model accuracy (training and validation sets)

Ultimately, at epoch 180, the training process came to an end, and the outcomes of the evaluation are shown in confusion matrices of Fig. 4.14.

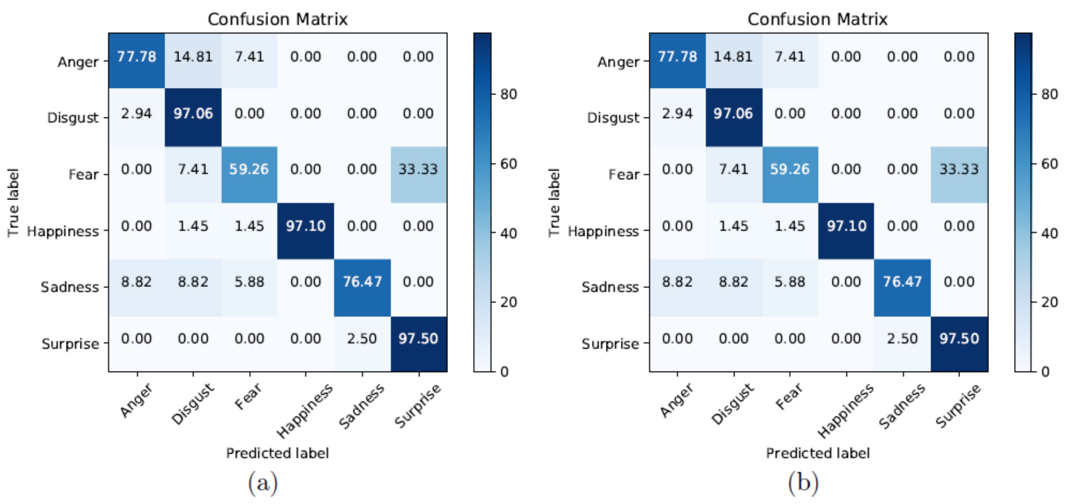


Figure 4.14: Confusion matrices: (a)Average accuracy of 81.12 using the majority voting algorithm (b)Average accuracy of 85.81 using mean scores

As a reference, we can observe from Fig. 4.8 of the second implementation, that Fear, the weakest class considering the training samples, had 40.15 and 48.74 percent accuracy on the testing set, for both evaluation methods. Moreover, the Anger class (88.80 percent in both of the matrices) along with Disgust (88.23 percent and 91.16 percent for the respective confusion matrices) had a high accuracy.

When comparing those findings to those in Fig. 4.14, it's clear that Fear, the one that would have the lowest accuracy, sees vast improvement, with the new version's accuracy equaling 59.15 percent in both confusion matrices. The accuracy of Disgust rises to 96.95 percent. However, we can observe that the accuracy of Anger declines, and the emotion as a whole does not improve. The network average accuracy is higher than it was in the second implementation, which did not apply cost-sensitive learning, as well as in all preceding implementations. Finally, in the testing procedure, a reduction of parameters in combination with the usage of a weighted loss function, yielded the best outcomes.

Chapter 5

Synopsis

In this thesis we presented a novel approach which aims to recognize human emotions in videos. Due to its significant impact to a large variety of human-computer-interaction applications, facial expression analysis and recognition is becoming a popular study area in recent times. Our approach was based on an FER system with VGG16 architecture, where we applied various transfer learning methods (fine-tuning).

Deep CNN's depend upon a significant quantity of training data in order to get optimal outcomes, which is widely regarded as their fundamental drawback. At the moment, however, there isn't enough data (with labels) for video emotion recognition. In order to resolve this obstacle, we loaded the model's pre-trained weights, which in turn were generated after training the model on ImageNet (1.2 million object recognition images). Although our own training dataset (around 11 thousand frames) is not so massive, it was considered sufficient for optimizing the model while also enabling it to identify emotions.

Overfitting is reduced when the number of network parameters is minimized, and the classifier's accuracy increases. Nevertheless, a significant disparity can still be noticed throughout the classes' individual accuracies. Overall, Surprise and Happiness outperform the other classes, while Fear has the lowest accuracy. As a result, we utilize cost sensitive learning for the training, since the issue occured because of the class imbalance in the training set. We also introduce a weighted cost function, which adversely affects classifications errors of samples from "weaker" classes. Both of the initial and reduced architectures were subjected to the cost-sensitive technique. There was a challenge in concluding the efficiency of this technique, as it may appear to be boosting model accuracy in some circumstances, though decreasing in some other cases. Particularly, the misclassification of the Fear emotion as

Surprise continues to be common. We can infer this from the similar facial expressions of these emotions, as Surprise is frequently standing out (we could also observe that Surprise is rarely classified as Fear). Finally, cost-sensitive learning aids the model's performance in the presence of data limits posed by psychological issues.

Furthermore, we investigate two testing approaches for deriving video estimates from frame estimates: majority voting and score averaging. In all confusion matrices, our tests indicate that the score averaging method performs better than majority voting.

In conclusion, we consider the outcome of this thesis as successful, since our expectations were met. Using knowledge from the domains of machine learning and signal processing, we were able to offer a novel methodology for human emotion recognition that yielded concrete results. Following our initial experimental results, we were also able to improve our approach. Our final modified model can be used in a variety of real-world scenarios. During our research, we identified a number of different results as well as issues with human emotion perception.

Bibliography

[1] Omar Sharif Reeshad Khan. A literature review on emotion recognition using various methods. *Global Journal of Computer Science and Technology*, 17:1–F, Apr. 2017.

[2] A. Papadakis. Human motion recognition: A geometric approach using deep learning regression. Διπλωματική εργασία, ΕΚΠΑ, 2019.

[3] T. Kanade, J.F. Cohn, and Yingli Tian. Comprehensive database for facial expression analysis. In *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, pages 46–53, 2000.

[4] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[5] I. Sutskever A. Krizhevsky and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, page 1105–1113, 2012.

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[8] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3):326–327, September 1995.

[9] Haibin Yan, Marcelo H. Ang, and Aun Neow Poo. Cross-dataset facial expression recognition. In *2011 IEEE International Conference on Robotics and Automation*, pages 5985–5990, 2011.

[10] E. Kamenou. Deep learning for human emotion recognition from video. Διπλωματική εργασία, Πανεπιστήμιο Θεσσαλίας, March. 2019.

[11] Hussam Qassim, Abhishek Verma, and David Feinzimer. Compressed residual-vgg16 cnn model for big data places image recognition. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 169–175, 2018.

[12] Jonathan Matthew Gratch Rafael A. Calvo, Sidney D'Mello. *The Oxford Handbook of Affective Computing*. Oxford University Press, 2015.

[13] Lucy Nwosu. *Deep Convolutional Neural Network for Facial Expression*. University of Houston-Clear Lake, 2017.

[14] Furkan Gürpınar Albert AliSalah Heysem Kaya. Video-based emotion recognition in the wild using deep transfer learning and score fusion. *IEEE Image Vision and Computing*, 65:66–75, Sept. 2017.

[15] Raghavendra Pappagari et al. Jaejin Cho. Deep neural networks for emotion recognition combining audio and transcripts. *Electrical Engineering and Systems Science: Audio and Speech Processing*, pages 2–4, Nov. 2019.

[16] Michael Glodek Günther Palm. Towards emotion recognition in human computer interaction. *Neural Nets and Surroundings*, pages 323–336, May 2013.

[17] Hiranmayi Ranganathan, Shayok Chakraborty, and Sethuraman Panchanathan. Multimodal emotion recognition using deep learning architectures. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, 2016.

[18] Panagiotis Tzirakis, Jiehao Zhang, and Bjorn W. Schuller. End-to-end speech emotion recognition using deep neural networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5089–5093, 2018.

[19] O'Sullivan et. Al Ekman P, Friesen W. V. Universals and cultural differences in the judgments of facial expressions of emotion. In *Journal of Personality and Social Psychology*, volume 53, page 712–717, 1987.

[20] Elsaeed E. AbdElrazek Hend Ab. ELLaban, A. A. Ewees. A real-time system for facial expression recognition using support vector machines and k-nearest neighbor classifier. *International Journal of Computer Applications*, 159:23–26, Feb. 2017.

[21] Seyed Mehdi Lajevardi and Margaret Lech. Facial expression recognition from image sequences using optimized feature selection. In *2008 23rd International Conference Image and Vision Computing New Zealand*, pages 1–6, 2008.

[22] Matti Pietikäinen. Image analysis with local binary patterns. In *Scandinavian Conference on Image Analysis*, pages 115–118, 2005.

[23] Yeongjae Cheon and Daijin Kim. A natural facial expression recognition using differential-aam and k-nns. In *2008 Tenth IEEE International Symposium on Multimedia*, pages 220–227, 2008.

[24] Tomas Pfister, Xiaobai Li, Guoying Zhao, and Matti Pietikäinen. Recognising spontaneous facial micro-expressions. In *2011 International Conference on Computer Vision*, pages 1449–1456, 2011.

[25] Alexander Freytag Christoph Käding, Erik Rodner and Joachim Denzler. Fine-tuning deep neural networks in continuous learning scenarios. In *Asian Conference on Computer Vision*, pages 588–605, 2016.

[26] C. Canton-Ferrer S. A. Bargal, E. Barsoum and C. Zhang. Emotion recognition in the wild from videos using images. In *ICMI '16: Proceedings of the 18th ACM International Conference on Multimodal Interaction*, page 433–436, 2016.

[27] V. Michalski K. R. Konda R. Memisevic S. E. Kahou and C. J. Pal. Recurrent neural networks for emotion recognition in video. In *ICMI '15: Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, page 467–474, 2015.

[28] E.D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, 1990.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[30] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[31] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[32] AO Remizov IR Shafarevich. *Linear algebra and geometry*. Russian Academy of Sciences, 2012.

[33] L. Bottou Y. LeCun, P. Haffner and Y. Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, volume 1681, pages 319–345, 1999.

[34] Vincent Vanhoucke Christian Szegedy, Sergey Ioffe and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2016.

[35] Wei Yu, Kuiyuan Yang, Yalong Bai, Hongxun Yao, and Yong Rui. Visualizing and comparing convolutional neural networks. *CoRR*, abs/1412.6631, 2014.

[36] Hoo-Chang Shin, Holger R. Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M. Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE Transactions on Medical Imaging*, 35(5):1285–1298, 2016.

[37] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. Database meets deep learning: Challenges and opportunities. *SIGMOD Rec.*, 45(2):17–22, September 2016.

[38] Chung-Hsien Wu, Jen-Chun Lin, and Wen-Li Wei. Survey on audiovisual emotion recognition: databases, features, and data fusion strategies. *APSIPA Transactions on Signal and Information Processing*, 3:e12, 2014.

[39] Jeffrey Cohn Pantic Maja, Nicu Sebe and Thomas Huang. Affective multimodal human-computer interaction. *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 669–676, 2005.

[40] Junho Yim, Heechul Jung, ByungIn Yoo, Changkyu Choi, Dusik Park, and Junmo Kim. Rotating your face using multi-task deep neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[41] Abir Fathallah, Lotfi Abdi, and Ali Douik. Facial expression recognition via deep learning. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pages 745–750, 2017.

[42] Nazil Perveen, Debaditya Roy, and Chalavadi Krishna Mohan. Spontaneous expression recognition using universal attribute model. *IEEE Transactions on Image Processing*, 27(11):5575–5584, 2018.

[43] A. Famili, Wei-Min Shen, Richard Weber, and Evangelos Simoudis. Data preprocessing and intelligent data analysis, Jan 1997.

[44] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *Proceedings. International Conference on Image Processing*, volume 1, pages I–I, 2002.

[45] G. Varoquaux, L. Buitinck, G. Louppe, O. Grisel, F. Pedregosa, and A. Mueller. Scikit-learn: Machine learning without learning the machinery. *GetMobile: Mobile Comp. and Comm.*, 19(1):29–33, June 2015.

[46] Chong Zhang, Kay Chen Tan, and Ruoxu Ren. Training cost-sensitive deep belief networks on imbalance data problems. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 4362–4367, 2016.

APPENDICES

Here is a part of my code responsible for: video to frame reduction, and face detection.

Code available at: https://github.com/gathanasiadis/EmotionRecognition

VIDEO TO FRAME:

```
import subprocess
import os
import threading
import pdb
import copy

VIDEO_EXTENSIONS = ['mp4', 'webm', 'avi']

def main(video_dir, frame_dir, n_thread):
   print('Starting: convert videos into frames\nvideo_dir:
     {:}\tframe_dir: {:}'.format(video_dir, frame_dir))
   threads = []
   for root, dirs, files in os.walk(video_dir):
      for file_name in files:
         if is_video_file(file_name):
            video_name = os.path.join(root, file_name)
            frame_output_path = os.path.splitext
            (video_name.replace(video_dir, frame_dir))[0]
            makefile(frame_output_path)
            threads.append(threadFun(video2frame,
            (video_name, frame_output_path )))

   run_threads(threads, n_thread)
   print('all threads is finished')

def is_video_file(filename):
   return any(filename.endswith(extension)
   for extension in VIDEO_EXTENSIONS)

def makefile(file_dir):
   if not os.path.exists(file_dir):
      os.makedirs(file_dir)
```

```
def run_threads(threads, n_thread):
   used_thread = []
   for num, new_thread in enumerate(threads):
      print('thread index: {:}'.format(num), end=' \t')
      new_thread.start()
      used_thread.append(new_thread)

      if num % n_thread == 0:
         for old_thread in used_thread:
            old_thread.join()
         used_thread = []
def video2frame(video_input, frame_output):
   linux_commod = 'ffmpeg -i {:} -f image2
     {:}/%07d.jpg'.format(video_input, frame_output)
   print('{:}'.format(video_input))
   subprocess.getstatusoutput(linux_commod)

class threadFun(threading.Thread):
   def __init__(self, func, args):
      super(threadFun, self).__init__()
      self.fun = func
      self.args = args
   def run(self):
      self.fun(*self.args)

if __name__ == '__main__':
   video_dir_train ='../video/train_afew/'
   frame_dir_train = '../frame/train_afew/'

   video_dir_val ='../video/val_afew/'
   frame_dir_val = '../frame/val_afew/'
```

```
    main(video_dir_train, frame_dir_train, n_thread =20)
    main(video_dir_val, frame_dir_val, n_thread = 20)
```

FER TRAINTEST:

```
import os
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.backends.cudnn as cudnn
from basic_code import load, util, networks
DEVICE = torch.device("cuda:0" if torch.cuda.is_available
    ↪ () else "cpu")
def main():
    parser = argparse.ArgumentParser
    (description='PyTorch Frame Attention Network Training'
       ↪ )
    parser.add_argument('--at_type', '--attention', default
       ↪ =1, type=int, metavar='N',
                   help= '0 is self-attention; 1 is self +
                      ↪ relation-attention')
    parser.add_argument('--epochs', default=180, type=int,
       ↪ metavar='N',
                   help='number of total epochs to run')
    parser.add_argument('--lr', '--learning-rate', default
       ↪ =4e-3, type=float,
                   metavar='LR', help='initial learning
                      ↪ rate')
    parser.add_argument('-e', '--evaluate', default=False,
       ↪ dest='evaluate', action='store_true',
                   help='evaluate model on validation set')
    args = parser.parse_args()
    best_acc = 0
    at_type = ['self-attention', 'self_relation-attention'
      ↪ ][args.at_type]
    logger = util.Logger('./log/','fan_afew')
    logger.print('The attention method is {:}, learning
      ↪ rate: {:}'.format(at_type, args.lr))

    ''' Load data '''
    root_train = './data/face/train_afew'
    list_train = './data/txt/afew_train.txt'
    batchsize_train= 48
    root_eval = './data/face/val_afew'
    list_eval = './data/txt/afew_eval.txt'
    batchsize_eval= 64
    train_loader, val_loader = load.afew_faces_fan(
      ↪ root_train, list_train, batchsize_train,
      ↪ root_eval, list_eval, batchsize_eval)
    ''' Load model '''
    _structure = networks.resnet18_at(at_type=at_type)
    _parameterDir = './pretrain_model/Resnet18_FER+_pytorch
      ↪ .pth.tar'
    model = load.model_parameters(_structure, _parameterDir
      ↪ )
    ''' Loss & Optimizer '''
    optimizer = torch.optim.SGD(filter(lambda p: p.
      ↪ requires_grad, model.parameters()), args.lr,
      ↪ momentum=0.9, weight_decay=1e-4)
    lr_scheduler = torch.optim.lr_scheduler.StepLR(
      ↪ optimizer, step_size=60, gamma=0.2)
    cudnn.benchmark = True
    ''' Train & Eval '''
    if args.evaluate == True:
        logger.print('args.evaluate: {:}', args.evaluate)
        val(val_loader, model, logger)
        return
    logger.print('frame attention network (fan) afew
      ↪ dataset, learning rate: {:}'.format(args.lr))

    for epoch in range(args.epochs):
        train(train_loader, model, optimizer, epoch)
        acc_epoch = val(val_loader, model, at_type)
        is_best = acc_epoch > best_acc
        if is_best:
            logger.print('better model!')
            best_acc = max(acc_epoch, best_acc)
            util.save_checkpoint({
                'epoch': epoch + 1,
                'state_dict': model.state_dict(),
                'accuracy': acc_epoch,
            }, at_type=at_type)

        lr_scheduler.step()
        logger.print("epoch: {:} learning rate:{:}".format(
          ↪ epoch+1, optimizer.param_groups[0]['lr']))

def train(train_loader, model, optimizer, epoch):
    losses = util.AverageMeter()
    topframe = util.AverageMeter()
    topVideo = util.AverageMeter()

    # switch to train mode
    output_store_fc = []
    target_store = []
    index_vector = []
```

```
model.train()
for i, (input_first, input_second, input_third,
   ↪ target_first, index) in enumerate(train_loader):
   target_var = target_first.to(DEVICE)
   input_var = torch.stack([input_first, input_second ,
      ↪  input_third], dim=4).to(DEVICE)
   # compute output
   ''' model & full_model'''
   pred_score = model(input_var)
   loss = F.cross_entropy(pred_score, target_var)
   loss = loss.sum()
   #
   output_store_fc.append(pred_score)
   target_store.append(target_var)
   index_vector.append(index)
   # measure accuracy and record loss
   acc_iter = util.accuracy(pred_score.data, target_var
      ↪ , topk=(1,))
   losses.update(loss.item(), input_var.size(0))
   topframe.update(acc_iter[0], input_var.size(0))
   # compute gradient and do SGD step
   optimizer.zero_grad()
   loss.backward()
   optimizer.step()

   if i % 200 == 0:
      logger.print('Epoch: [{:3d}][{:3d}/{:3d}]\t'
           'Loss {loss.val:.4f} ({loss.avg:.4f})\t'
           'Acc@1 {topframe.val:.3f} ({topframe.avg:.3f
              ↪ })\t'
         .format(
                epoch, i, len(train_loader), loss=losses,
                   ↪ topframe=topframe))

        index_vector = torch.cat(index_vector, dim=0) # [256]
           ↪ ... [256] ---> [21570]
        index_matrix = []
        for i in range(int(max(index_vector)) + 1):
            index_matrix.append(index_vector == i)

        index_matrix = torch.stack(index_matrix, dim=0).to(
           ↪ DEVICE).float() # [21570] ---> [380, 21570]
        output_store_fc = torch.cat(output_store_fc, dim=0) #
           ↪ [256,7] ... [256,7] ---> [21570, 7]
        target_store = torch.cat(target_store, dim=0).float() #
           ↪  [256] ... [256] ---> [21570]
        pred_matrix_fc = index_matrix.mm(output_store_fc) #
           ↪ [380,21570] * [21570, 7] = [380,7]
        target_vector = index_matrix.mm(target_store.unsqueeze
           ↪ (1)).squeeze(1).div(
            index_matrix.sum(1)).long() # [380,21570] *
               ↪ [21570,1] -> [380,1] / sum([21570,1]) -> [380]

        acc_video = util.accuracy(pred_matrix_fc.cpu(),
           ↪ target_vector.cpu(), topk=(1,))
        topVideo.update(acc_video[0], i + 1)
        logger.print(' *Acc@Video {topVideo.avg:.3f}    *
           ↪ Acc@Frame {topframe.avg:.3f} '.format(topVideo=
           ↪ topVideo, topframe=topframe))

    def val(val_loader, model, at_type):
        topVideo = util.AverageMeter()
        # switch to evaluate mode
```

```
model.eval()
output_store_fc = []
output_alpha = []
target_store = []
index_vector = []
with torch.no_grad():
    for i, (input_var, target, index) in enumerate(
        ↪ val_loader):
        # compute output
        target = target.to(DEVICE)
        input_var = input_var.to(DEVICE)
        ''' model & full_model'''
        f, alphas = model(input_var, phrase = 'eval')

        output_store_fc.append(f)
        output_alpha.append(alphas)
        target_store.append(target)
        index_vector.append(index)

    index_vector = torch.cat(index_vector, dim=0) #
        ↪ [256] ... [256] ---> [21570]
    index_matrix = []
    for i in range(int(max(index_vector)) + 1):
        index_matrix.append(index_vector == i)

    index_matrix = torch.stack(index_matrix, dim=0).to(
        ↪ DEVICE).float() # [21570] ---> [380, 21570]
    output_store_fc = torch.cat(output_store_fc, dim=0)
        ↪ # [256,7] ... [256,7] ---> [21570, 7]
    output_alpha = torch.cat(output_alpha, dim=0) #
        ↪ [256,1] ... [256,1] ---> [21570, 1]
    target_store = torch.cat(target_store, dim=0).float
```

```
            ↪ () # [256] ... [256] ---> [21570]
        ''' keywords: mean_fc ; weight_sourcefc; sum_alpha;
            ↪ weightmean_sourcefc '''
        weight_sourcefc = output_store_fc.mul(output_alpha)
            ↪ #[21570,512] * [21570,1] --->[21570,512]
        sum_alpha = index_matrix.mm(output_alpha) #
            ↪ [380,21570] * [21570,1] -> [380,1]
        weightmean_sourcefc = index_matrix.mm(
            ↪ weight_sourcefc).div(sum_alpha)
        target_vector = index_matrix.mm(target_store.
            ↪ unsqueeze(1)).squeeze(1).div(
            index_matrix.sum(1)).long() # [380,21570] *
                ↪ [21570,1] -> [380,1] / sum([21570,1]) ->
                ↪ [380]
        if at_type == 'self-attention':
            pred_score = model(vm=weightmean_sourcefc, phrase
                ↪ ='eval', AT_level='pred')
        if at_type == 'self_relation-attention':
            pred_score = model(vectors=output_store_fc, vm=
                ↪ weightmean_sourcefc, alphas_from1=
                ↪ output_alpha, index_matrix=index_matrix,
                ↪ phrase='eval', AT_level='second_level')
        acc_video = util.accuracy(pred_score.cpu(),
            ↪ target_vector.cpu(), topk=(1,))
        topVideo.update(acc_video[0], i + 1)
        logger.print(' *Acc@Video {topVideo.avg:.3f} '.
            ↪ format(topVideo=topVideo))
        return topVideo.avg
if __name__ == '__main__':
    main()
```