UPPSALA
UNIVERSITET

IT 16 040
Examensarbete 30 hp
Juni 2016

Facial emotion detection using deep learning

Daniel Llatas Spiers

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Facial emotion detection using deep learning

Daniel Llatas Spiers

The use of machines to perform different tasks is constantly increasing in society. Providing machines with perception can lead them to perform a great variety of tasks; even very complex ones such as elderly care. Machine perception requires that machines understand about their environment and interlocutor's intention. Recognizing facial emotions might help in this regard. During the development of this work, deep learning techniques have been used over images displaying the following facial emotions: happiness, sadness, anger, surprise, disgust, and fear.

In this research, a pure convolutional neural network approach outperformed other statistical methods' results achieved by other authors that include feature engineering. Utilizing convolutional networks involves feature learning; which sounds very promising for this task where defining features is not trivial. Moreover, the network was evaluated using two different corpora: one was employed during network's training and it was also helpful for parameter tuning and for network's architecture definition. This corpus consisted of facial acted emotions. The network providing best classification accuracy results was tested against the second dataset. Even though the network was trained using only one corpus; the network reported auspicious results when tested on a different dataset, which displayed facial non-acted emotions. While the results achieved were not state-of-the-art; the evidence gathered points out deep learning might be suitable to classify facial emotion expressions. Thus, deep learning has the potential to improve human-machine interaction because its ability to learn features will allow machines to develop perception. And by having perception, machines will potentially provide smoother responses, drastically improving the user experience.

Handledare: Maike Paetzel
Ämnesgranskare: Ginevra Castellano
Examinator: Edith Ngai
IT 16 040
Tryckt av: Reprocentralen ITC

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank:

My supervisor, Maike Paetzel because of her dedication and commitment with this project. It was great to work next to someone that is always pushing you forward to make a better job. It was a bless having her as my supervisor and I will miss our Wednesday meetings.

My reviewer, Ginevra Castellano because of her guidance during all the research process since the very beginning when she welcomed me into the Social Robotics Lab. Her precious input allow to keep the direction of the research on the right path.

To all the people in the Division of Visual Information and Interaction, it was great to share these last months with you. Specially, the interesting conversations during lunch.

Dedication

Life has placed some wonderful people in my way. People that have always believed in me and to whom I love: Silvia, Sean, and Cristhian. Thank you for always inspiring me.

Chapter 1

Introduction

The use of machines in society has increased widely in the last decades. Nowadays, machines are used in many different industries. As their exposure with humans increase, the interaction also has to become smoother and more natural. In order to achieve this, machines have to be provided with a capability that let them understand the surrounding environment. Specially, the intentions of a human being. When machines are referred, this term comprises to computers and robots. A distinction between both is that robots involve interaction abilities into a more advanced extent since their design involves some degree of autonomy.

When machines are able to appreciate their surroundings, some sort of machine perception has been developed [95]. Humans use their senses to gain insights about their environment. Therefore, machine perception aims to mimic human senses in order to interact with their environment [65][68]. Nowadays, machines have several ways to capture their environment state trough cameras and sensors. Hence, using this information with suitable algorithms allow to generate machine perception. In the last years, the use of Deep Learning algorithms has been proven to be very successful in this regard [1][31][35]. For instance, Jeremy Howard showed on his Brussels 2014 TEDx's talk [43] how computers trained using deep learning techniques were able to achieve some amazing tasks. These tasks include the ability to learn Chinese language, to recognize objects in images and to help on medical diagnosis.

Affective computing claims that emotion detection is necessary for machines to better serve their purpose [80]. For example, the use of robots in areas such as elderly care or as porters in hospitals demand a deep understanding of the environment. Facial emotions deliver information about the subject's inner state [74]. If a machine is able to obtain a sequence of facial images, then the use of deep learning techniques

would help machines to be aware of their interlocutor's mood. In this context, deep learning has the potential to become a key factor to build better interaction between humans and machines, while providing machines with some kind of self-awareness about its human peers, and how to improve its communication with natural intelligence [48][51].

1.1 Motivation and goals

This project is part of the research performed by Social Robotics Lab. The Social Robotics Lab at the Division of Visual Information and Interaction is interested in the design and the development of robots that are able to learn to interact socially with humans. The idea is that society can benefit from the use of robots in areas such as education, e-learning, health care, and assistive technology.

Technically, the project's goal consists on training a deep neural network with labeled images of static facial emotions. Later, this network could be used as part of a software to detect emotions in real time. Using this piece of software will allow robots to capture their interlocutor's inner state (at some extent). This capability can be used by machines to improve their interaction with humans by providing more adequate responses. Thus, this project fits the purpose and research of the Social Robotics Lab well.

Finally, this is a multidisciplinary project involving affective computing, machine learning and computer vision. Learning how these different fields are related, and to understand how they can provide solutions to complex problems is another project's goal.

1.2 Methods and structure

This project has been divided into two phases. The first phase consisted on the use of a facial emotion labeled data set to train a deep learning network. The chosen data set is the Extended Cohn-Kanade Database [49]. More details about the corpus can be found in **Chapter 3**. Additionally, evaluations were performed on several network topologies to test their prediction accuracy. The use of convolutional neural networks on the topologies was preferred given its great achievements on computer vision tasks [5]. An overview of deep learning concepts, with an emphasis on convolutional networks is presented in **Chapter 2**. In order to perform the implementation

of the network and the training process, Google's library TensorFlow [64] was used. **Chapter 4** introduces TensorFlow functions for computer vision, and the reasons it was selected compared to other frameworks.

The second phase focused on testing the model against a new data set, AM-FED [20]. Similarly to corpus on the previous phase, a detailed explanation is presented in **Chapter 3**. The idea is to make a comparison on both data sets, and evaluate the generalization property of the network. Also, a focus on some parameters and its effect on the model's accuracy prediction was performed. These parameters were chosen because their influence over the network's behavior:

- Network loss
- Learning rate
- Dropout
- Optimizers

More information about parameter's value selection is displayed in **Chapter 5**. Experiments results are supplied in **Chapter 6**; while a discussion about them regarding the literature is exhibited in **Chapter 7**. Finally, future work and conclusions are addressed in **Chapter 8**.

Chapter 2

Theoretical Background

In this section, a description of relevant concepts for this project is presented. This section aims to provide a background on the topics to be discussed during the rest of the report. This background is accomplished by means of a chronological revision of fields such as affective computing and machine learning. In order to describe the concepts, a top down approach is going to be utilize. Moreover, related research to the approach used in this project is introduced, as well.

2.1 Affective Computing

As described by Rosalind Picard [75], "... affective computing is the kind of computing that relates to, arises from, or influences emotions or other affective phenomena". Affective computing aims to include emotions on the design of technologies since they are an essential part of tasks that define the human experience: communication, learning, and decision-making.

One of the main foundations behind affective computing is that without emotions, humans would not properly function as rational decision-making beings. Some researches show that there is no such a thing as "pure reason" [19]. Emotions are involved in decision-making since a fully scientifically approach would turn into an extreme time consuming process, not suitable for daily tasks. Researches around this particular topic have shown that the brain does not test each probable option, but it is biased by emotion to quickly make a decision [47].

An emotion is defined as a class of qualities that is intrinsically connected to the motor system. When a particular emotional state is triggered, the motor system will

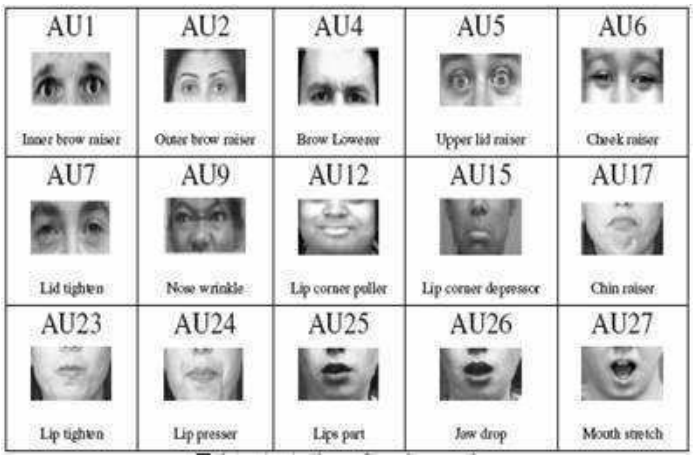

Figure 2.1: Facial action units [78]

provide the corresponding set of instructions to reproduce the particular modulations connected to that class [13]. So far, emotions' importance has been addressed without taking human interaction into consideration. Empathy is a human capacity that makes us aware and provides us with understanding about what other beings might be experiencing from their current's position [76]. Moreover, empathy allows us to build close relationships and strong communities. Therefore it is fundamental towards a pro-social behavior, which includes social interaction and perception [27]. Thus, it is very important for affective computing to develop ways to properly measure these particular modulations since they can lead to a better understanding of a subject's emotional state. The two main ways to do so is by detecting facial and vocal emotions. However, in this project, only facial emotions were used.

2.1.1 Facial Emotion Recognition

The work by psychologist Paul Ekman has become fundamental to the development of this area. Nowadays, most of the face emotion recognition studies are based on Ekman's Facial Action Coding System [28]. This system provides a mapping between facial muscles and an emotion space. The main purpose behind this system is to classify human facial movements based on their facial appearance. This classification was first developed by Carl-Herman Hjortsj, who is a Swedish anatomist. Figure 2.1 displays a set of facial emotion units and their corresponding facial gesture.

However, this mapping might face some challenges. For instance, gestures involved on facial emotions can be faked by actors. The absence of a real motivation behind the

emotion does not prevent humans to fake it. For instance, an experiment describes when a patient, who is half paralyzed is asked to smile. When it is asked, only a side of the mouth raises. However, when the patient is exposed to a joke, both sides of the mouth raise. [18]. Hence, different paths to transmit an emotion depend on the origin and nature of a particular emotion.

With respect to computers, many possibilities arise to provide them with capabilities to express and recognize emotions. Nowadays, it is possible to mimic Ekman's facial units. This will provide computer with graphical faces that provide a more natural interaction [30]. When it comes to recognition, computers have been able to recognize some facial categories: happiness, surprise, anger, and disgust [101]. More information about facial emotion recognition can be found in section 2.6 on page 19.

2.2 Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence. A simple ML explanation is the one coined by Arthur Samuel in 1959: "... field of study that gives computers the ability to learn without being explicitly programmed". This statement provides a powerful insight in the particular approach of this field. It completely differs from other fields where any new feature has to be added by hand. For instance, in software development, when a new requirement appears, a programmer has to create software to handle this new case. In ML, this is not exactly the case. The ML algorithms create models, based on input data. These models generate an output that is usually a set of predictions or decisions. Then, when a new requirement appears, the model might be able to handle it or to provide an answer without the need of adding new code.

ML is usually divided into 3 broad categories. Each category focuses on how the learning process is executed by a learning system. These categories are: supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning is when a model receives a set of labeled inputs, which means that they also contain the corresponding belonging class. The model tries to adapt itself in a way that can map every input with the corresponding output class. On the other hand, unsupervised learning receives a set of inputs without them being labeled. In that sense, the model tries to learn from the data by exploring patterns on them. Finally, reinforcement learning is when an agent is rewarded or punished accordingly the decisions it took in order to achieve a goal.

On this project, our problem falls into the supervised learning category since the images to be processed are labeled. In our case, the label is the emotion that the image represents.

2.3 Artificial Neural Networks

Supervised learning has a set of tools focused on solving problems within its domain. One of those tools is called Artificial Neural Networks (ANN). An ANN is a set of functions that perform label prediction. If the ANN is analyzed as a black box; the input would consist of labeled examples, and the output would be a vector containing a set of predictions. Usually, these predictions are expressed as a probability distribution for all labels [7]. Other definitions of ANN emphasize on other aspects such as its processing properties: "A massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use." [37]. However, an ANN might not necessarily be massively. Small implementations are made just for the sake of trying new ideas. Engelbrecht provided a definition with a different intention, more focused on the topology: "It is a layered network of artificial neurons. An artificial neural network may consist of an input layer, hidden layers, and an output layer. An artificial neuron is a model of a biological neuron." [29].

An ANN can be explained trough the following three steps:

1. Input some data into the network.
2. Transformation over the input data is accomplished by means of a weighted sum.
3. An intermediate state is calculated by applying a non-linear function to the previous transformation.

From the previous steps, it can be said that all of them constitute a layer. A layer represents the block with the highest-level on a network. The transformation is usually referred as a unit or neuron, although the latter is more related with neurobiology. Finally, the intermediate state acts as the input into another layer or into the output layer. In figure 2.2 at page 8, a typical neural network topology is presented.

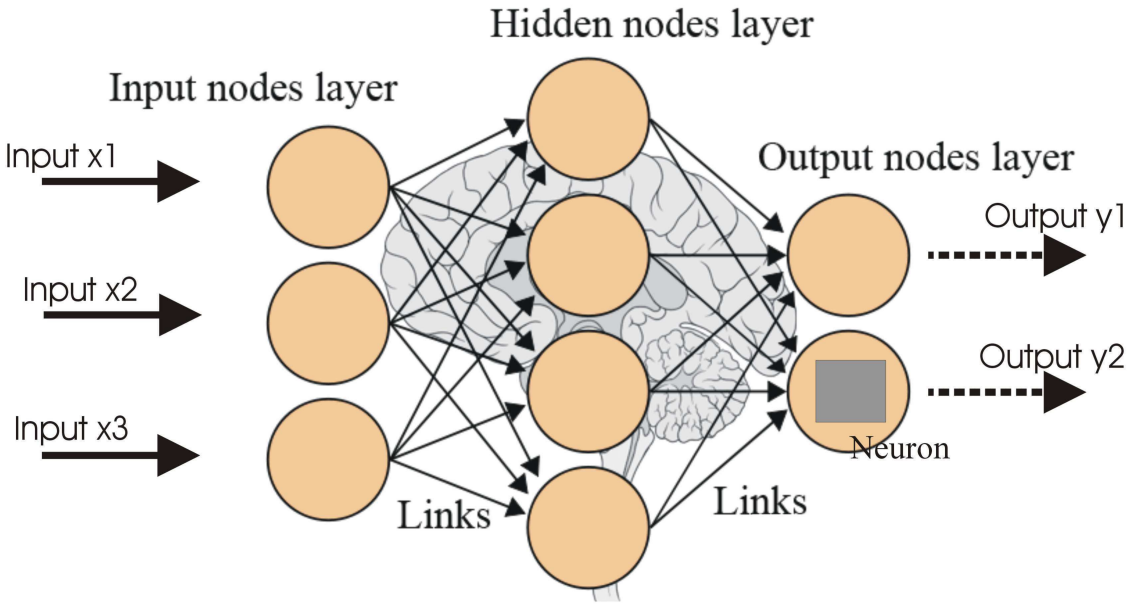

Figure 2.2: Artificial neural network topology [52]

Returning to Engelbrecht's definition [29], an interesting question arises: how is that a model inspired by the brain mechanics ended up as a computational model? In order to provide an answer, some historical background is necessary.

2.3.1 Rise and fall of ANN

ANN dates back from the 1940's. While some researchers started to study the brain structure, none of them were able to formulate it as means of a computational device. It was until 1943, when Warren McCulloch and Walter Pitts were able to formulate an ANN as a model suitable to perform computations [67]. Some years later (1949), Donald Hebb provided a theory to describe how neurons adapt on the brain while the learning process happens [38]. After that, it took almost a decade for an ANN implementation: the perceptron. The perceptron was introduced by Frank Rosenblatt [81]. It is the simplest ANN architecture. Moreover, it was the first time that by means of supervised learning, an ANN was able to learn.

In the figure 2.3, the topology of a perceptron is introduced. Luckily, most of ANN concepts can be explained in this simple architecture. As it can be seen, there

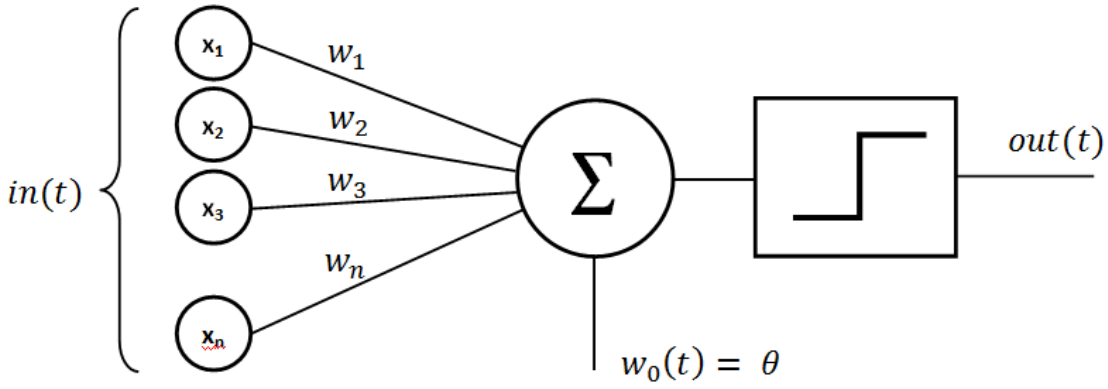

Figure 2.3: Perceptron topology [16]

is a set of inputs, X_1 to X_n. This layer is coined as the input layer. Each of these inputs has a corresponding weight, W_n.

On the neuron (unit), a weighted sum is performed. Also, a bias is added to the neuron so it can implement a linear function. The independence of the bias will move the curve on the X-axis.

$$y = f(t) = \sum_{i=1}^{n} X_i * W_i + \Theta$$

After that, the result of $f(t)$ is the input of an activation function. The activation function defines the output of the node. As the perceptron is a binary classifier, the binary step function is suitable for this topology. It will output only a couple of classes, 0 or 1.

$$output = \begin{cases} 0, & y < 0 \\ 1, & y \geq 0 \end{cases}$$

Finally, the prediction is measured against the real value. This error signal is going to be used to update weights on the first layer to improve the prediction results. This is performed trough backpropagation learning [39].

During 1960's, ANN were a hot research topic. However, a publication by Minsky and Papert on 1969 finished this golden era [70]. In their publication, it was stated that a perceptron has several limitations. Specially, that it would not be suitable to perform general abstractions. Moreover, more complex architectures derived from a perceptron would not be able to overcome these limitations, as well. While time has

proven that this was not true, back on those days it was a huge drawback for ANN. In general, the beginning of the 70's was not good at all for AI. This period is called the "AI Winter" since many projects in different areas were canceled or any of them bring any benefits at that time.

2.3.2 ANN revival

During the 1970s and 1980s, the shift on AI went exclusively to symbolic processing. However, a set of factors helped ANN to be on the spot again on the mid 1980s. These factors belong to different areas.

One factor was that symbolic processing showed slow progress and it was narrowed to small simulations. Another factor was computer accessibility and hardware improvement compared to the one on previous decades. Nowadays, this factor is still relevant since experiments demand a lot of computational power. Most of current simulations would not be feasible in those times. Finally, connectionist researchers started to show some interesting results. For instance, in 1988, Terrence J. Sejnowski and Charles R. Rosenberg published a paper with the results of NETtalk [84]. NETtalk is an ANN that was trained to learn to pronounce English words. Thus, connectionism started to gain some momentum.

A year later, Yann LeCun showed some impressive results on handwritten zip code recognition by using a multilayer ANN. This paper results to be quite interesting since it was a pioneer on the collaboration between image recognition and machine learning. Also, this paper introduced concepts related to convolutional neural networks such as feature maps and weight sharing.[56]

During the 1990s, support vector machines (SVM) were highly used by many researchers. Its popularity was due to its simplicity compared to ANN, and also because they were achieving great results. While ANNs were back on the map again, they were not the main star on machine learning. This situation would not change much until the beginning of the 2010's. Nowadays, there is a lot of data in the world. Also, computational devices has become more powerful and cheaper during the last 20 years. The Big Data era is pulling Machine Learning into a very exciting period. This period is providing the tools, and the data to perform extremely big computations. Nowadays, we dont talk anymore about ANN, but a very popular term: Deep Learning.

2.4 Deep Learning

The latest reincarnation of ANN is known as Deep Learning (DL). According to Yann LeCun, this term designates "... any learning method that can train a system with more than 2 or 3 non-linear hidden layers."[32]. DL has achieved success on fields such as computer vision, natural language processing, and automatic speech recognition. One of the main strengths of using DL techniques is that there is no need for feature engineering. The algorithms are able to learn features by themselves over basic representations. For instance, on image recognition, an ANN can be feed with pixel representations of images. Then, the algorithm will determine if certain pixel combination represents any particular feature, that is repeated through the image. As the data is processed through the layers, the features will go from very abstract forms to meaningful representation of objects.

DL started to become popular after some better than state-of-the-art results were achieved on several fields. For instance, the first paper containing information about a major industrial application was one related to automatic speech recognition [31]. In this paper from 2012, ANN outperformed Gaussian mixture models in several benchmarking tests. This paper is a collaboration between four research groups: University of Toronto, Microsoft Research, Google Research and IBM Research. Two years later, another breakout publication was on the field of natural language processing [45]. This research presented that Long-Short Term Memory (a particular ANN architecture called recurrent neural network specialized on sequences) provided better results than statistical machine translation, which was the default tool for translation at that time. This network was able to translate words and phrases from English to French.

Finally, a deep learning technique that is relevant for this project is presented: Convolutional Neural Networks (CNN). A paper published in 2012 by a group of researchers from Toronto University [1] showed results never achieved before on the ImageNet classification competition. This research has become a foundational work on DL. On the 2012 edition, its solution using deep CNN achieved an error rate of 15.3% on top-5 classification while the second best achieved 26.2%. More details about CNN are introduced in section 2.5 on page 13.

In this research, two concepts, which the ML community has widely adopted, are stressed: the use of rectified linear unit as the activation function [34] and the use of GPU for training [100].

2.4.1 Rectified linear unit

The activation function of a unit (neuron) is an essential part of an ANN architecture. The use of different functions has been used by researchers since ANN early days. In section 2.3.1 on page 8, the step function was introduced as the activation function. However, the binary nature of the step function does not allow to have a good error approximation.

In order to overcome this situation, sigmoid functions were utilized. They provided a very good performance for small networks. Though, using sigmoid function proved not to be scalable on large networks [1]. The computational cost of the exponential operation might be really expensive since it could lead to very long numbers [34]. Another factor against the use of sigmoid function is the gradient vanishing problem. This means that the gradient value on the curve tails does become too small that it prevents learning [63][71][105].

Under this scenario, the rectified linear unit function (ReLU) provided benefits compared to previous common activation functions: its computational cost was cheaper, it provided a good error approximation and it did not suffer from the gradient vanishing problem. ReLU is displayed on Figure 2.4 at page 13, and it is defined as:

$$f(x) = max(0, x)$$

Krizhevsky et al. research [1] showed that using ReLU reduced the number of epochs required to converge when using Stochastic Gradient Descent by a factor of 6.

However, a major drawback when using ReLU is its fragility when the input distribution is below zero. This happens when the neuron reaches a point when it will not be activated by any datapoint again during training. For more information, refer to [4].

2.4.2 Use of GPU

The use of GPU for training has become fundamental for training deep networks because of practical reasons. The main reason is the reduction of the training time compared to CPU training [14]. While different speedups are reported depending on the network topology, it is common to have around 10 times speed when using GPU [102].

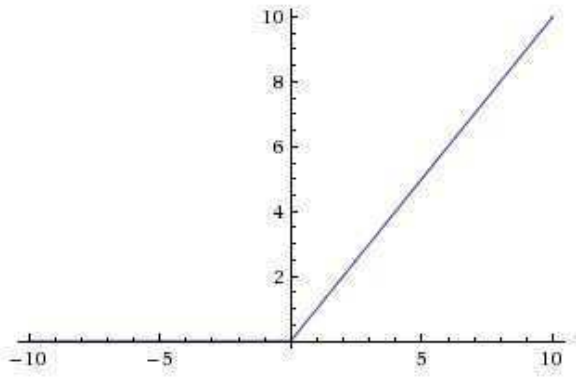

Figure 2.4: Rectified Linear Unit (ReLU) [3]

The difference between CPU and GPU is how they process tasks. CPU are suitable to perform sequential serial processing on few cores. On the other hand, GPU encompasses a massive parallel architecture. This architecture involves thousands of small cores designed to handle multiple tasks simultaneously [73].

Thus, DL operations are suitable to train on GPU since they involve vector and matrix operations that can be handled in parallel. Despite that during this project only a limited amount of experiments were conducted using GPU, it is important to stress its practical importance reducing training time.

2.5 Convolutional Neural Networks

The study of the visual cortex is closely related to the development of the convolutional neural networks. Back in 1968, Hubel and Wiesel presented a study focused on the receptive fields of the monkeys visual cortex [44]. This study was relevant because of the striate cortex (primary visual cortex) architecture description and the way that cells are arranged on it.

Moreover, it also presented two different type of cells: simple, and complex. The simple ones are focused on edge-like shapes; while the complex ones cover a broader spectrum of objects and they are locally invariant. Therefore, the different sets of cell arrangements in the cortex are able to map the entire visual field by exploiting the correlation of objects and shapes in local visual areas.

One of the first implementations inspired by Hubel and Wiesel ideas was one called Neocognitron. Neocognitron [33] is a neural network model developed by Kunihiko Fukushima in 1980. The first layer of the model is composed by units that

represent simple cells; while the second layer units' represent complex cells. The implementation of the local invariance property of the visual cortex is the greatest Neocognitron's achievement. Furthermore, the output mapping is one to one. Each complex cell maps to one and only one specific pattern.

However, a main drawback around Neocognitron was at its learning process. At that time, there was not a method to tune weight values with respect to an error measure for the whole network such as backpropagation. While the finish mathematician Seppo Linnainmaa derived the backpropagation modern form in 1970 [58] [59], its use in ANN was not applied until 1985. During this time, few applications were developed using backpropagation [98]. In 1985, the work by Rumelhart, Hinton, and Williams [82] introduce the use of backpropagation into ANN.

Conceptually, backpropagation measures the gradient of the error with respect to the weights on the units. The gradient will change every time the values for the weights are changed. Then, the gradient will be used on gradient descent in order to find weights that will minimize the error of the network. When using backpropagation with some optimizer such as Gradient Descent (GD), the network is able to autotune its parameters. GD is a first-order optimization algorithm. It looks for a local minimum of a function taking steps proportional to the negative of the gradient[6]. Figure 2.5 at page 15 shows the general steps to perform backpropagation. Algorithm details and further explanation can be found in the literature [39][79][99].

As it has been previously stated, Yann LeCun is a pioneer on the research of CNN. The first true backpropagation practical application was LeCun's classifier on handwritten digits (MNIST) [56]. This system was one of the most successful uses of CNN at that time since it read a large amount of handwritten checks. LeCun's research has led to CNN topologies that have been used as an inspiration for future researchers, one of the most popular is LeNet-5 [57].

LeNet-5 was implemented as part of an experiment on document recognition. This paper underlines the idea that in order to solve pattern recognition problems, it might be a better idea to use automatic learning solutions instead of hand designed ones. Since addressing all the different cases that input data could have on a natural way is quite a complex task, machine learning suits better for this purpose. Additionally, a simple pattern recognition system is described. This system consists of two main modules: a feature extractor, which transforms the input data into low dimensional vectors; and a classifier, which most of the time is general purpose and trainable.

Furthermore, key components of CNN are described: local receptive field, weight

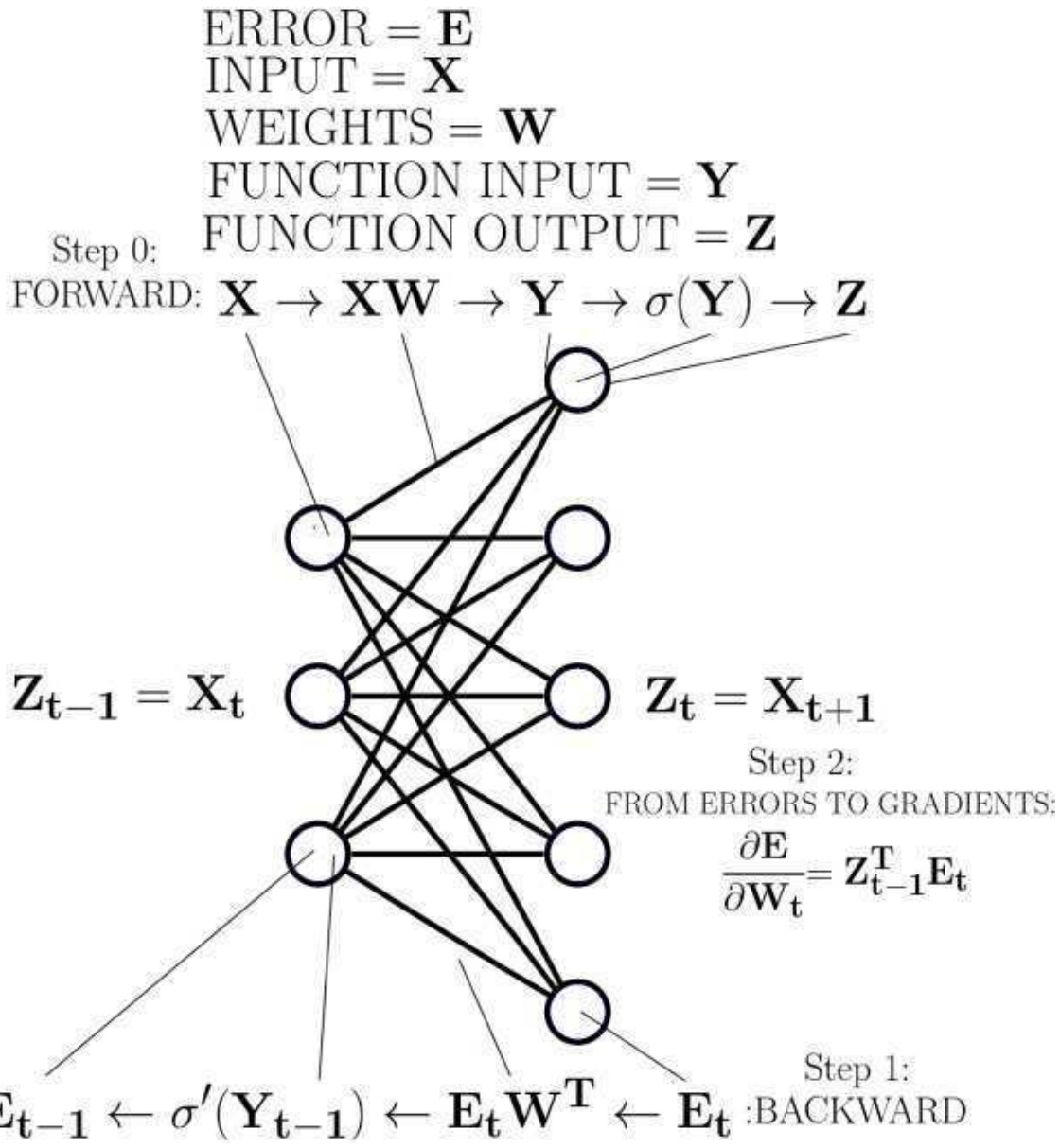

Figure 2.5: Backpropagation algorithm [93]

sharing, convolution operation, spatial sub-sampling, dropout, and stochastic gradient descent.

2.5.1 Convolution operation

In mathematics, a convolution operation is defined as a way to mix two functions. An analogy commonly used is that this operation works as a filter. A kernel filters

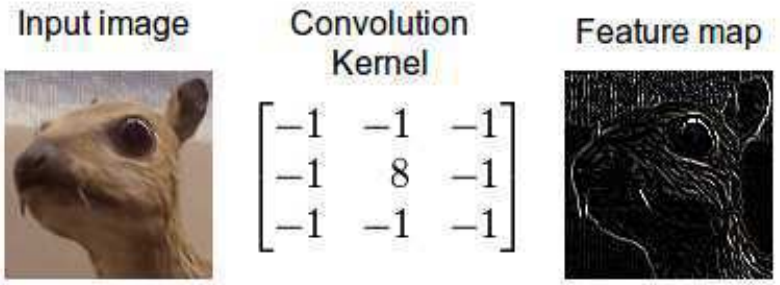

Figure 2.6: Convolution operation [3]

everything that is not important for the feature map, only focusing on some specific information.

In order to execute this operation, two elements are needed:

- The input data
- The convolution filter (kernel)

The result of this operation is a feature map. Figure 2.6 on page 16 provides a graphical explanation about the mechanics on the convolutional operation. The number of feature maps (output channels) provides the neural network with a capacity to learn features. Each channel is independent since they aim to learn each a new feature from the image that is being convoluted.

Finally, the type of padding defines the algorithm to be used when performing the convolution. There is a special case on the input's edges. One type of padding will discard input's border, since there is no more input next to it that can be scanned. On the other hand, the other padding will complete the input with a value of 0. It is a matter of reducing parameters while convoluting.

For an extended explanation on the mechanics behind the convolution operation, refer to [94].

2.5.2 Weight sharing

When a feature is detected, the intuition is that it will be meaningful regardless of its position. Weight sharing exploits the translationally-invariant structure of high-dimensional input. For example, the position of a cat in an image is not relevant to recognize the cat. Moreover, in a sentence, the position of a verb or noun should not change its meaning.

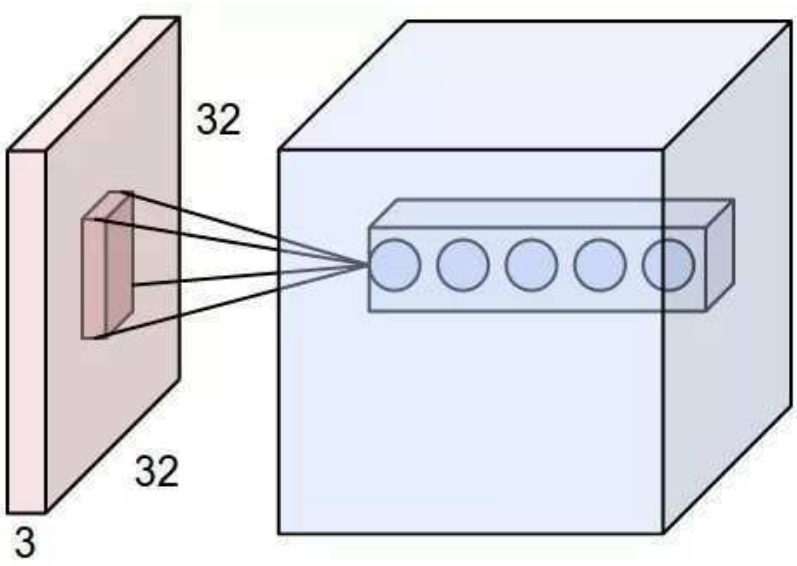

Figure 2.7: Local receptive field of size 5x5x3 for a typical CIFAR-10 image, 32x32x3 [3]

As it was previously stated, after the convolution operation a plane composed of the results of applying the same filter trough the entire input is generated. This plane is named feature map. Each feature map is the result of a convolutional operation with a kernel. Kernels are initialized with different weights in order to perceive different features. Thus, the feature found is kept trough the whole feature map, and its position is irrelevant for the network.

A convolutional layer usually has a set of feature maps that extract different features at each input location, as defined by the filter size. The process of scanning the input, and then storing the units state on the feature map is termed as convolution operation.

2.5.3 Local receptive field

Also known as filter size or kernel size, a local receptive field is the area to which a neuron will be connected on the high-dimensional input. The local receptive field is a hyperparameter of the network, it means that its shape is defined in advance.

This concept has a strong influence from neurons on the visual cortex that are locally-sensitive. The idea is not to connect all the neurons to the whole input space, but to focus on locally-connected areas. These local connections only happen on the width and height dimensions. The input depth dimension is not locally-connected,

but fully-connected trough all its channels. For instance, a high-dimensional input is an image. An image is represented by 3 dimensions: width, height, and depth. When a local receptive field is applied over an image, its kernel only acts locally on the width and height dimensions; not on the depth, where it takes all the dimensions into account. The depth dimension is similar to the number of channels. For example, an RGB image has 3 channels: red, green, and blue. The final image is a composition of all these 3 images in each color.

By having input allocated on this way, neurons are able to extract elemental features like edges, end-points or corners. Applying this idea to subsequent layers, the network will be able to extract higher-order features. Moreover, the reduction of connections also reduces the number of parameters, which helps to mitigate overfitting.

Figure 2.7 on page 17 displays how a single neuron is connected to a feature map of size 5x5x3. The convolution operation will iterate trough the entire input (image) using this filter. This means that the width and height of the input will decrease after the operation; but this is not true for the input depth dimension.

After the convolution operation, the depth dimension will be the number of filters applied to the input. A set of filters is initialized to capture different features that can be found in the image. Each filter is initialized with different weights. However, the weights keep the same for a filter while it convolutes trough the whole input; this is called weight sharing.

It is important to remind that these operations had a strong focus on feature learning, but not on classification. The use of fully connected layers (also known as multi-layer-perceptrons) along with convolutional networks provides both capabilities. The main advantage of these layers is that they can be optimized using stochastic gradient descent on back propagation style, along with the weights for convolutional layers.

2.5.4 Spatial sub-sampling

Spatial sub-sampling is an operation also known as pooling. The operation consists of reducing the values of a given area to a single one. So, it reduces the influence of the feature position on the feature map by diminishing its spatial resolution. This is done by choosing the most responsive pixel after a convolution operation.

There are two types of polling: average and maximum. The average one computes the mean on the defined area; while, the maximum only selects the highest value on

the area. The area size can lead to reduction on the prediction performance, if the value is too large. It proceeds in a similar fashion compared to a convolution operation, as a filter and a stride is defined. Given the filter region, this operation returns the pixel with the higher value [41].

Thus, the dimension of the feature map is reduced. This reduction prevents that the system learns feature by position. Then, it helps to generalize feature for new examples. This is important since features on new examples might be on different positions.

2.5.5 Dropout

Dropout minimizes the impact of units that have a strong activation. This method shutdowns units during training, so other units can learn features by itself [88].

Providing with more independence to all units reduces the strong unit bias leading to strong regularization and better generalization.

2.5.6 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) has only one difference with respect to Gradient Descent (GD). The difference is the number of examples considered to calculate the gradients of the parameters. The original version performs this operation using all the examples on the training set. The stochastic one only uses few examples defined by the batch size [11].

It is important to notice that when using SGD, the learning rate and its decrease scheduling is more difficult to set compared to GD since there is much more variance in the gradient update [89].

2.6 Related work

Affectiva is the worlds leading commercial research group on emotion recognition. Its current patent portfolio is the largest, compared to startups in this field. Their research has adopted deep learning methodologies since its private corpus consists of 3.2 million facial videos. Also, their data gathering has been done in 75 countries, which prevents the research to fall on cultural or regional behaviors [96]. In order to measure its detector accuracy, the area under a Receiver Operating Characteristic

(ROC) curve is used. The value of ROC score ranges between 0 and 1. The classifier is more accurate when the value is closer to 1. Some emotions such as joy, disgust, contempt, and surprise have a score greater than 0.8. While expressions such as anger, sadness, and fear achieve a lower accuracy since they are more nuanced and subtle. Moreover, Affectiva has been able to successfully identify facial action units on spontaneous facial expressions without using deep learning techniques [85].

On the following paragraphs, approaches involving the use of feature engineering are introduced. While the approaches on feature extraction and classification are different; all of them involved Cohn-Kanade dataset, as part of its work. It is worth to mention that Cohn-Kanade dataset was used in this research, so the results give a valuable comparison.

Kotsia et al. [53] focused on the effect of occlusion when classifiying 6 facial emotion expressions. In order to achieve this, several feature engineering techniques and classification models were combined. Gabor features, which is a linear filter used for edge detection, and Discriminant Non-negative Matrix Factorization (DNMF), which focuses on the non-negativity of the data to be handled, are the feature extractors techniques. To classifiy these features multiclass support vector machine (SVM) and multi-layer perceptron (MLP) were used. The results over Cohn-Kanade are the following: Using a MLP with Gabor 91.6% and with DNMF: 86.7%. While using SVM achieved 91.4%. Another corpus used was JAFFE: Gabor combined with MLP achieved 88.1% and when using it with DNMF, it resulted on 85.2% classification accuracy.

Wang and Yin [97] examined how the distortion of detected face region and the different intensities of facial expressions affect robustness on their model. Topographic context (TC) expression descriptors were selected in order to perform feature extraction. This techniques performs a topographic analysis. In this analysis, the image is trated as a 3D surface. Each pixel is labeled taking into account its terrain features. The use of several classifiers was reported: quadratic discriminant classifier (QDC), linear discriminant classifier (LDA), support vector classifier (SVC) and naive bayes (NB). Results using a Cohn-Kanade subset (53 subjects, 4 images per subject for each expression, 864 images): with QDC: 81.96%, with LDA: 82.68%, with NB: 76.12%, with SVC: 77.68%. Results in MMI facial expression dataset (5 subjects, 6 images per subject for each expression. 180 images) were also reported: with QDC: 92.78%, with LDA: 93.33%, and with NB: 85.56%.

Kotsia and Pitas work [54] focused on recognizing either the six basic facial expres-

sions or a set of chosen AUs. A technique using geometric displacement of candide nodes was used during feature selection. For expression recognition, a six-class SVM was used. Each class corresponds to one expression. Some impressive results were achieved using this technique: 99.7% for facial expression recognition. Another work that provided a high classification accuracy was Ramachandran et al. [77] focus on developing a novel facial expression recognition system. During feature extraction, features resulting from principal component analysis (PCA) are fine-tuned by applying particle swarm optimization (PSO). Later, these features are used on a feed forward neural network (FFNN). The best classification result achieved was 97%.

For more information on techniques using feature engineering, Vinay Bettadapura [10] provided an extensive list of researches between 2001 and 2008 on facial expression recognition and analysis.

As it can be inferred from the literature, facial affect detection is a complex task. Its complexity has lead to several approaches that has something in common: the need for feature extraction, and then applying a classifier on top. However, in this project, the idea is to use convolutional networks to avoid the feature extraction stage; since the network would be able to detect features by itself.

Chapter 3

Datasets evaluation

As it was stated previously, the following research belongs to the supervised learning category. The need for a data set containing images of facial emotions and their corresponding label is crucial. For this purpose, a couple of data sets were chosen to perform the experiment:

1. Extended Cohn-Kanade [49][61]
2. Affectiva-MIT Facial Expression Dataset [20]

Another pair of data sets seemed promising at the beginning, as well. This pair is composed by EURECOM Kinect face dataset [69] and The Florence Superface dataset [9]. However, they were discarded because their lack of labels or any information that could lead to an automatic generation of them.

3.1 Extended Cohn-Kanade

This data set is an extension of the previous original Cohn-Kanade one. The Extended Cohn-Kanade (CK+) includes 593 sequences from 123 subjects. CK+ was recorded using a couple of Panasonic AG-7500 cameras in a lab. The images are frontal views or 30 degree views. Its size can be either 640 x 490 pixels or 640 x 480 pixels with 8-bit gray-scale or 24-bit color values PNG files. Figure 3.1 on page 23 shows a particular image sequence example from the dataset.

The participants were requested to perform a series of 23 facial displays. Each sequence starts with a neutral face, and the last one is the proper emotion displayed.

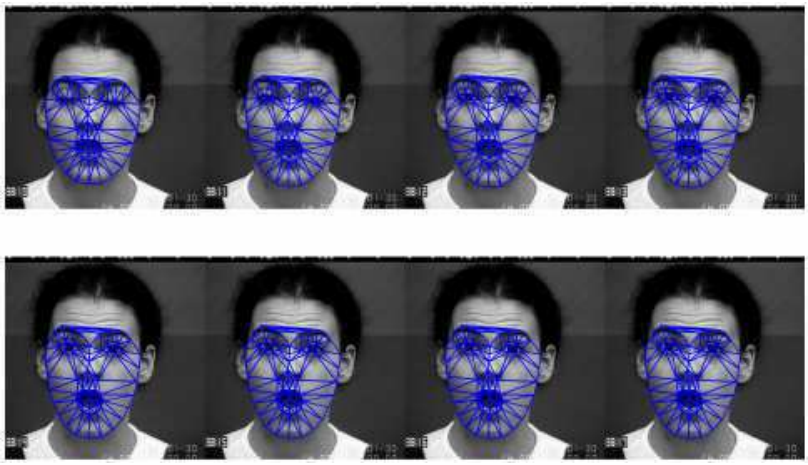

Figure 3.1: Image sequence for subject S130 from CK+ [61]. Subject displays the surprise emotion.

While not all the subjects have a corresponding sequence for each label, the labeled emotions on the sequences are:

1. Anger
2. Contempt
3. Disgust
4. Fear
5. Happiness
6. Sadness
7. Surprise

3.1.1 Download and Folder Structure

The procedure to download CK+ starts when filling a form on `consortium.ri.cmu.edu/ckagree/`. Some conditions apply for the use of the dataset. Later, an email with a download address and credentials was delivered to the specified email. Finally, the download options are displayed. One of them corresponding to the image data, and the second one to the metadata.

The image data is supplied in a compressed file named Cohn-kanade.tgz. Its size is 1.7 GB. The folder structure contains 585 directories and 8795 image files. The folder structure is very detailed, since it gives each sequence of images a unique directory.

The filename encodes the subject (first set of 3 digits), the sequence (second set of 3 digits), and the image file (set of 8 digits), as follows:

```
S001_001_01234567
```

On the other hand, the metadata provides 4 files with different kind of information, such as:

1. Cohn-Kanade Database FACS codes
2. ReadMeCohnKanadeDatabase_website
3. Consent-for-publication.doc
4. Translating AU Scores Into Emotion Terms

The last file provides a methodology for translating action unit scores into emotions. This methodology was the one used to generate the labels automatically for the AM - FED data set.

3.2 Affectiva-MIT Facial Expression Dataset

Affectiva-MIT Facial Expression Dataset (AM-FED) is the result of a collaboration between Affectiva and MIT in March 2011. The experiment consisted of recording spontaneous facial emotions from viewers that allow access to their webcams. The viewers were exposed to three Super Bowl ads. All this videos were recorded in the wild. The spontaneous component is important since it provides real emotions; the facial action units are not faked. Thus, to test a network using it becomes more challenging.

AM-FED encompasses 242 facial videos (168 359 frames). All these frames have been labeled by four different criteria and a minimum of three coders labeled each frame of the data. The presence of the following information allows to generate emotion labels similar to CK+ automatically:

Figure 3.2: Action units found on AMFED images [20]

1. Frame- by-frame manual labels for the presence of: a) 10 symmetrical FACS action units; b) 4 asymmetric (unilateral) FACS action units; c) 2 head movements, smile, general expressiveness, feature tracker fails; d) Gender.

2. The location of 22 automatically detected landmark points.

3. Self-report responses of familiarity with, liking of, and desire to watch again for the stimuli videos.

4. Baseline performance of detection algorithms on this dataset. We provide baseline results for smile and AU2 (outer eyebrow raise) on this dataset using custom AU detection algorithms.

3.2.1 Download and Folder Structure

AM-FED download procedure is quite similar to the one at CK+. First, an end user license agreement has to be filled. This agreement can be found at `affectiva.com/facial-expression-dataset/`. After that, an electronic copy of it has to be sent to `amfed@affectiva.com`. Later, a response arrived with the download link.

The whole dataset, containing videos and metadata, is contained on a compressed file. The name of the file is AM-FED-AMFED.zip. The folder structure contains 5 directories.

1. AU labels

2. Baseline performance of classification for smile and AU2.
3. Landmark Points
4. Videos - AVI
5. Videos - FLV

The information on AU labels was used to generate the emotion labels in a similar way that the ones provided at CK+. Due to lack of action unit information, it was not possible to generate labels for all of the emotions presented at CK+. Most of the images were gathered only for happiness or joy emotion. The AU present on this corpus are:

- AU2, Outer Brow Raiser
- AU4, Brow Lowerer
- AU5, Upper Lid Raiser
- AU9, Nose Wrinkler
- AU12 (unilateral and bilateral), Lip Corner Puller
- AU14 (unilateral and bilateral), Dimpler
- AU15, Lip Corner Depressor
- AU17, Chin Raiser
- AU18, Lip Puckerer
- AU26, Jaw Drop

Videos - AVI and Videos - FLV provide the same amount of videos, 243 files. The only difference between them is the video format. The filename is named after a unique value that does not seem to follow any particular rule, in comparison with CK+:

`09e3a5a1-824e-4623-813b-b61af2a59f7c`

	CK+	**AM-FED**
Action Units	Optionally for download	Yes
Emotion Label	Yes	No
Format	Images	Video
Size (num. of people)	123	242
Environment	Lab	On the wild

Table 3.1: Datasets comparison

3.3 Differences between CK+ and AM-FED

There are two main differences between both datasets. The fist one is related on the environment in which the subjects were recorded. While the subjects on CK+ were recorded in a lab and following instructions, AM - FED captured subjects in the wild. The second difference is about the information provided, especially for labels. CK+ delivers a set of images with a corresponding emotion label. AM - FED is a set of videos with a chronological order on some action units presented on the video.

First, the subject record environment presents always a challenge for training process. The spontaneous component is related to the ability of the network to generalize. This means that the neural network can have a better prediction on an example not similar to one that is already presented on the training set. When someone is asked to perform an emotion in a lab, then it is not a natural reaction, but somehow acted. Thus, it is interesting to explore the different performances that can be achieved using both datasets.

Second, CK+ delivers an easy to use emotion label for training purposes. This situation is not similar for AM - FED given that the information is displayed in terms of action units over a timeline. On this behalf, AM - FED privileges accuracy over usability. The development of a piece of software to generate labels in an automatic way became mandatory to continue with the training. The method is better explained in section 5.2.1 on page 40. For a better visualization of the differences refer to Table 3.1 on page 27.

Finally, on the first stage of the project CK+ was used for training and testing the ANN. Later, on the second stage, AM-FED was used to evaluate the performance of the network.

Chapter 4

Implementation Framework

Nowadays, many frameworks have been developed for deep learning. Some of the most popular ones include libraries such as: Caffe, Theano, and TensorFlow. Also, implementing a framework from scratch using a programming language was never considered. It would have been out of scope since it requires a big amount of effort, and the duration of such a project usually takes years. The use of Python as the front-end API on all these frameworks shows that it is the preferred language for machine learning. Usually, Python is combined with a programming language that provides support for low level operations such as: C or C++, to act on the back end.

4.1 Frameworks

4.1.1 Caffe

Caffe [46] started as a PhD project by Yangqing Jia while working at Berkeley Vision and Learning Center (BVLC). Nowadays, Caffe is a project maintained by BVLC, and has acquired many contributors from its growing community. It was written in Python and C++; and it has support for the main platforms: Linux, OSX and Windows. Caffe was mainly designed to perform computer vision computations.

4.1.2 Theano

Theano [8] has its origin on the Montreal Institute for Learning Algorithms at the University of Montral. Nowadays, it has became an open source project with a big community. Theano is a Python-library that has the ability to produce CPU or GPU

instructions for some graph computations. The performance of these instructions is closer to the one provided by C, and it is much faster than pure Python.

Theano has a focus on mathematical expressions, especially those that include the use of tensors. Moreover, Theano compiler takes advantage of many optimization techniques to generate C code that is suitable for specific operations.

4.1.3 TensorFlow

TensorFlow (TF) [64] is an open source software library for machine learning written in Python and C++ . Its release some months ago (Nov 15) had a strong press coverage. The main reason behind it is that TF was developed by Google Brain Team. Google has already been using TF to improve some tasks on several products. These tasks include speech recognition in Google Now, search features in Google Photos, and the smart reply feature in Inbox by Gmail.

Some design decision in TF have lead to this framework to be early adopted by a big community. One of them is the ease of going from prototype to production. There is no need to compile or to modify the code to use it on a product. Then, the framework is not only thought as a research tool, but as a production one. Another main design aspect is that there is no need to use different API when working on CPU or GPU. Moreover, the computations can be deployed over desktops, servers and mobile devices.

A key component of the library is the data flow graph. The sense of expressing mathematical computations with nodes and edges is a TF trademark. Nodes are usually the mathematical operations, while edges define the input / output association between nodes. The information travels around the graph as a tensor, a multidimensional array. Finally, the nodes are allocated on devices where they are executed asynchronously or in parallel when all the resources are ready.

4.2 Why TensorFlow?

While Caffe and Theano seemed suitable frameworks to perform this project, in the end, the chosen one was TF r0.7. TF was chosen because of a pair of main reasons: The first one is that TF has support by Google. The fact that millions of people have used products running TF on the background means that the framework has been properly tested. Moreover, Google has a vast amount of resources to continue

working on the framework, and to provide documentation and learning resources. Another reason is that TF has benefit from the development experience around other frameworks, specially Theano. So, the scope of the framework is not only limited to research and development; but also to deployment.

Google's support has positioned TF as one of the main libraries for machine learning in a relative short time since its release in November 2015. Google is committing to a long term development of the framework by using it on its own products. For instance, Google DeepMind, which are the AlphaGo creators (AlphaGo is a computer that was able to beat a professional human Go player for the first time), decided to move all their projects from a framework named Torch7 [15] to TF. Also, a distributed version of TF was released in April 2016. All of these are signals that Google is pushing TF to become its main tool for machine learning research. When it comes to documentation, TF webpage offers a detailed explanation of the entire Python API, and for all the major releases to date. Also, a massive open online course on Deep Learning taught by Google was released just after a couple of months after TF release. The instructor is Vincent Vanhoucke, who is a Principal Research Scientist at Google. His work is related to the Google Brain team, and TF is the tool used to complete the assignments on the course.

Another reason to choose TF is that the framework encompasses a high maturity level despite the short time since it was released. The fact that many of TF developers were previously involved in other projects such as Theano and Torch7 is really relevant. TF has benefit from the experience on developing such frameworks. TF was able to correct many issues found on early stages of other frameworks since its initial design. As a consequence, TF has achieved state of the art performance without compromising code readability. Moreover, flexibility to define different operations; especially neural network topologies leads to rapid prototyping. The flexibility is not just related to network composition or operations definition but computation deployment platforms. TF API is the same even when the computations are executed on CPU or GPU in a desktop, server or mobile device.

Chapter 5

Methodology

This chapter is divided into two main sections; each corresponding to the two work phases mentioned on **Chapter 1**. On each section, it is described how image and video are pre-processed, the reasons behind choosing a particular topology and values for several parameters, and how the network's accuracy is evaluated. This chapter describes how techniques and concepts described so far on the report interact during the experimental part.

5.1 First phase

On this phase, the work around the CK+ dataset is presented. It includes pre-processing CK+ images, defining a network topology, feeding the network with the modified CK+ images, training the network, tuning parameters, and evaluating the network's accuracy.

5.1.1 CK+ image pre-processing

The image pre-processing is presented in the following paragraphs. The input is the CK+ image dataset, and the output is a binary file. The binary file was used to feed the network during training. This was the beginning of the project's experimental part.

Initially, some complications arose because of the dataset's folder structure and missing labels for some image sequences. By using *walk* Python function, all image files found on the CK+ tree directory where moved into the same destination folder. This was possible given that the filename was enough to identify each image to its

corresponding sequence and subject. A similar approach was used to move the label files. Moreover, all image sequences without label were moved into a separate folder.

Each labeled image sequence starts with a neutral face and finishes with the image of the facial expression. As it can be inferred, the first images are not so meaningful for the training process since no facial emotions are displayed on them. In order to minimize the impact of these images, the first 3 images of each sequences were discarded and only the last 10 images were taken into account for training. These number were defined as an heuristic since not all the sequences have the same number of images, nor the same emotion display distribution for each image. The size of the dataset changed according to the different selections that were applied over it:

- A total of 4895 labeled images; after excluding the first 3 images of each sequence.
- 3064 labeled images; just taking into account the last 10 images per sequence.
- 1538 labeled images; after considering only the last 5 images.

All the operations performed on the following paragraphs were only applied to the set of 3064 labeled images, as they were the only ones used during training.

The size of CK+ images involves a great computational power. Given the hardware limitations, a solution was to perform a crop. Also, another reason to support this idea is that it was a lot of free space both on left and right in the image. This space was the background, meaningless for our project and it might also lead to overfitting. Based on these observations, it was decided to perform a crop only on the facial area. *OpenCV* library was used to achieve this task. It uses a technique called face cascade [36] to achieve this purpose. Some parameter-tuning needs to be done, since the algorithm was detecting two faces on some images. After face detection is performed, the image is cropped only on that area.

The next step is to rescale the images. *OpenCV* library has built-in support to perform this operation. The cropping was done on two values: 32 pixels (similar to CIFAR-10 images) and 64 pixels. The idea is to compare between them during training. Before converting this image set into a binary file, they were converted into grayscale images. *Python Imaging Library (PIL)* was the selected tool to accomplish this task. While color related features might be interesting to explore; they also increase the number of parameters (weights and bias) on the network by orders of magnitude. More parameters involve a larger training time and a highest overfitting

chance. Those were the main reasons to use grayscale images. This transformation uniforms the input data in size (width and height) and number of channels (depth). This is important because CK+ contains images belonging to the original corpus, and the extended one. Images on the original corpus were recorded on different conditions.

The last step is to create a binary file from this image set. The need to create a binary file containing the label and the image was generated by unsuccessful tries of loading images and labels separately into TF.

In order to generate the bin file, a label dictionary is generated. This dictionary was used to match each sequence with its corresponding label. The dictionary is represented by a list of lists. Moreover, a record size is defined for each example. This size is defined as the byte sum of the label and the product of the image's height, width and channel. For the 32-pixel images, the value is 1025 bytes per example, while it is 4097 on 64-pixel images.

Finally, the bin file is generated by transforming each image and label into a *Numpy* array. All these *Numpy* arrays are stored into a final vector, which has a fixed number of positions similar to the number of images being processed. In the end, the resulting vector is materialized into a file. The name of the file is *ck.bin*.

For implementation details, please refer to appendix A.

5.1.2 Data input into TF

On this section, the first interaction with TF is encountered. The binary file containing the images and their corresponding label is feed into a record reader. This operation created a queue consisted of all the training examples.

```
1 filenames = [os.path.join(PATH, FILENAME)]
2 # Create a queue that produces the filenames to read.
3 filename_queue = tf.train.string_input_producer(filenames)
4 # Read examples from files in the filename queue.
5 read_input = read(filename_queue)
```

After that some cast operation are performed on the key and value.

- Key: The value of the label after being cast as an int32.
- Value: The tensor containing the tensor representing the image.

Later, the tensor is reshaped from depth, height and width to height, width, and depth. Finally, it is time to apply transformations into the image to perform data augmentation.

5.1.3 Data augmentation

One of the main drawbacks behind supervised learning is the need of labeled data. The manual work involved on data labeling demands many people following a strict set of rules [55]. The bigger the dataset, the more complex to label it. Deep Learning requires big amounts of data for training. Since this is a very expensive task, data augmentation has been proven an efficient way to expand the dataset [62][100]. A small dataset can lead the network to either overfitting or underfitting [42]. Also, they help to better cover the example space, not just focusing on the region delimited by the original dataset. Thus, networks training using data augmentation will generalize better when exposed to new examples [12].

Data augmentation consists of applying transformation on the corpus. In this case, transformations were applied over CK+ images. Modifying image properties helps to exploit the invariant features that the network can learn [1][17].

TF provides a set of functions suitable for image transformation: Flipping the image from left to right, and adjusting the brightness and contrast. All the parameters were defined following TF tutorial configuration on convolutional neural networks [92].

Finally, the whitening operation is performed over the image. The whitening operation computes the mean pixel value and then, it subtracts this value from the image. As a consequence, the pixel's mean is centered around the value of zero.

The following code snippet shows how these operations are performed in TF:

```
1 # Randomly flip the image horizontally.
2 distorted_image = tf.image.random_flip_left_right(distorted_image)
3 distorted_image = tf.image.random_brightness(distorted_image,
4                                              max_delta=63)
5 distorted_image = tf.image.random_contrast(distorted_image,
6                                              lower=0.2, upper=1.8)
7 # Subtract off the mean and divide by the variance of the pixels.
8 float_image = tf.image.per_image_whitening(distorted_image)
```

5.1.4 Network description

The topology to be used is inspired by the one developed on the Visual Geometry Group (VGG) at University of Oxford [87]. VGG network was awarded as the runner-up on ImageNet 2014's competition, and it stressed network depth's importance to improve classification accuracy. Moreover, VGG proved to generalize well to other tasks and data sets, in comparison with other kind of architectures [87]. This network was chosen since it has a strong emphasis on using pure convolutional network architectures compared to other state-of-the-art topologies such as GoogleNet [91].

After finishing the image processing, the next step is to create image batches to feed the network. The batch size is another parameter to take into account. Since the network optimization is performed by stochastic gradient descend (SGD) with momentum, choosing a correct batch size is critical. Some aspects to take into account to set the batch size are the data set size, and hardware availability. It would be optimal to take into account the whole data set in each step to optimize towards the gradient; however such an approach would be computationally expensive and time consuming [102]. It is common to find on the literature that the batch size is a power of two. For instance, on Krizhevsky et al. [1], the batch size is 128. In this project, the batch size was set to 64. This value was defined after trying several values. It provided a good trade-off between the training time and loss reduction.

Topology

The network topology is presented in table 5.1 and in figure 5.1 on page 36. The network is composed of 4 different types of operations: 1 convolutional layer, 1 max pooling operation, 2 fully connected layers, and a softmax layer.

Layer	**Description**
Conv1	ReLU. 64 output channels.
Pool1	Max pooling.
FC1	Fully connected layer with ReLU and 384 units
FC2	Fully connected layer with ReLU and 192 units
Softmax	Cross entropy

Table 5.1: Network topology model for CK+ dataset on phase 1

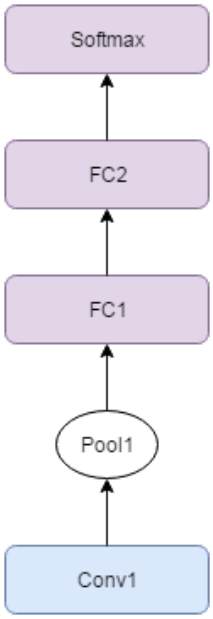

Figure 5.1: Network topology diagram

Layers specification

The convolutional layer contains the following hyperparameter set; there is no rule of thumb to determine values for these parameters [1][56]. Most of the time they are tuned by trial and error:

1. Number of output channels
2. Size of the kernel's receptive field
3. Shape of the kernel's stride
4. Type of padding operation

In this configuration, the number of output channels is set to 64. This value is similar to the one specified on Tensor Flow's Convolutional Network tutorial based on CIFAR-10 dataset [92]. An assumption is made here. CIFAR-10 contains 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. If a network with 64 output channels is able to determine with great accuracy each of these categories; then, it should be enough to extract features for different facial emotions. Moreover, adding more feature maps increases the training time, specially if the training is not executed over GPU. Also, adding more output channels did not increase the prediction accuracy.

In order to perform the convolution operation, three hyperparameters need to be set: kernel's receptive field size, kernel's stride shape, and the padding operation. Regarding kernel's receptive field size, it was defined as 3 x 3 pixels. This is the smallest size that is able to capture features considering the notion of left/right, up/down and center [87]. Moreover, the kernel's stride shape, which defines how to move the filter around the image, was kept fixed in one pixel. This is done in order to preserve the spatial resolution after each convolution operation [87]. Finally, the padding operation is defined as *SAME*, which means it completes the edges with pixels of value zero [104].

The sub-sampling operation chosen is max pooling. The pixel window for this operation is 3 x 3 with a stride of 2. This configuration is similar to the one provided by Tensor Flow's Convolutional Network tutorial based on CIFAR-10 dataset [92].

After the first convolutional layer, this topology presents two fully connected layers. The first one contains 384 units and the second one 192 units. These numbers were defined after trial and error optimizing the classification accuracy. All the activation functions on these nodes are ReLU. In order to avoid influence of features with a big activation, dropout was applied to these both layers. The dropout value was tuned trough trial and error to 0.99.

Finally, the last layer of the model is the softmax linear that will give a probability for that particular example on each label. When a vector provides this information is called logit. The result of this layer has a particular shape, which is defined by the number of classes being trained, 6 in this case, since the contempt emotion was removed because of few examples in the dataset. More details about the implementation can be found in appendix C.

5.1.5 Training

In this section, the training configuration is displayed: the cost function and the learning rate.

Cross entropy

The training objective is to adjust weights and biases values on the network's layers. This is done by minimizing a cost function. The cost function is the cross-entropy between the original labels and the model's prediction. This function is chosen because the last layer returns a distribution over all the 6 labels. Cross-entropy is a specifi-

cally error measurement for probability distributions. The fact that each image has one and only one label (mutually exclusive labels) was exploited to use cross entropy. For more information about cross-entropy refer to [66][72]

An algorithm to optimize the cross-entropy function has to be defined. The one used is momentum [89][90] with a value of 0.9, similar to Krizhevsky et al. [1]. The next step is to update weights and biases values accordingly. While the cross entropy loss is minimized, it moves the weights towards the gradients.

Learning rate

The learning rate is a hyperparameter to take into account. The learning rate indicates the search's speed trough the weight space. If the learning rate is too small, then the search will not be too exhaustive. On the other hand, a larger learning rate will allow for a better search, but it can lead to the weights growing too large [40][86][103].

In this experiment, a decaying learning rate was implemented. It means that the initial learning rate will start diminishing after some iterations. In order to perform this computation, three hyperparameters are set:

1. Initial learning rate
2. Decay factor
3. Epochs per decay

The initial learning rate was set to 0.1. This value is a heuristic that usually performs good on training convolutional networks. The decay factor indicates the proportion in which the initial learning rate will be diminished. Finally, epochs per decay parameter is the trigger to start the decreasing. An epoch is when all the images defined has been processed from the queue. This value is set to 50 epochs. The learning rate and the decay factor values were calculated by trial and error. The epochs per decay was calculated based on the size of the dataset and the batch size.

5.1.6 Evaluation

The evaluation was made by assigning one point to each correct prediction on a batch. After running an arbitrary number of batches (10), the sum of all the correct predictions is divided by the total number of examples. Two different types of evaluation

are performed: on the top-1 class and on the top-3 class. Top-1 class prediction means that the label is only compared against the highest probability returned by the last layer. On the same way, Top-3 class prediction returns true if the label is predicted on the 3 higher probabilities. In the following code snippet, k defines how many classes are taken into account for the classification.

```
def in_top_k(logits, label, k)
```

Then, the prediction returning a vector with the size of the batch is returned. Each element is either true or false, depending on the prediction. All true elements are counted. And, finally the total precision of the model is calculated.

```
while step < num_iter and not coord.should_stop():
        predictions = sess.run([top_k_op])
        true_count += np.sum(predictions)
        step += 1

precision = true_count / total_sample_count
```

Each batch is randomly generated from the image queue. This means that accuracy might change because of the image selection. In order to minimize this effect, accuracy results were presented after running the evaluation for 10 times. Moreover, the average is also reported.

During evaluation, data augmentation is not applied over images. The only image operation is the whitening one. That's the only difference when feeding the network compared to the training stage.

5.2 Second phase

The second phase evaluates AM-FED dataset on the network trained in phase 1. The only different section on the methodology is on the data pre-processing since AM-FED dataset is composed of videos.

5.2.1 AM-FED video processing

The methodology proposed by Lucey at al. [61] is followed to generate the labeled images from AM - FED videos.

In order to extract frames that correspond to the emotions expressed as action units, the action unit label files have to be inspected. Using Python walk function, action unit label files are opened. The CSV library is imported since it will provide an easy way to access this type of files. Usually, the complexity around CSV files is that delimiter encoding is not standard.

When the file's first row is reached, a header is created for that particular file. The header is an array that keeps the column position for time, smile and all action units found. Since not all the files have the same information regarding AU, building a header for each file was necessary.

$$header = \begin{bmatrix} Time & SmileLevel & AU7 & AU15 \\ 0:15 & 50 & 75 & 80 \end{bmatrix}$$

When reaching the second row and afterwards, the header is used to get the content of that particular row. The content is passed to a function that checks for a set of emotions defined by CK+. If an emotion is found, then the image is generated.

In order to generate the image, the CV2 library was installed. Using the CSV file name, the corresponding video file is found. Then, the column time is used to read the frame on that particular millisecond.

Finally, if that frame exists, a jpg file is generated into a destination path. The CK+ label is added as the file name last position.

When concluding this process, the result is a set of labeled images similar to CK+. These images will be used to generate a BIN file that will be inputted into Tensor Flow.

For implementation details, please refer to appendix B.

5.2.2 Evaluation

The network trained over six emotions was evaluated by a subset of AM-FED images. This subset of images only comprehend the happiness emotion. Surprise and fear emotions were not possible to be extracted. This is because AM-FED metadata does not include the corresponding action units for these labels. Moreover, the action units combination for angry and sadness did not provide any image. The emotions that

were able to be extracted were only three: happiness, contempt and disgust. From these emotions, the first one reached 520 images; the second one, 9 images; and the last one, only one.

Regarding the evaluation method, it is similar to the one used on **Phase 1**.

Chapter 6

Results

In this chapter, experimental results are displayed. A set of baselines are presented to better understand the performance of the experiments compared to other publications.

6.1 Baselines

In this section, a set of baselines are introduced. The first of them exhibits the accuracy of a random guess proposal. The following baselines were extracted from the Emotion Recognition In The Wild Challenge and Workshop (EmotiW) [23]. Moreover, a research presented in **Related Work** section is introduced as a state-of-the-art baseline.

All the numbers accounted on this section refers to the corresponding model prediction accuracy.

6.1.1 Random guess

A random guess computed over the 7 labels provided by CK+ gives a prediction performance of 14.27%. This result is equivalent to calculate an uniform probability for each label. As it was mentioned before, the contempt emotion was removed from the dataset. In that case, the baseline for 6 labels only is 16.67%.

The most common label in CK+ is surprise with 83 sequences, if we would assign always this label, the prediction performance would be of 25.38%. The same approach for the 3 most common labels (surprise, joy and disgust) would have resulted in 64.5% of prediction accuracy.

6.1.2 EmotiW

EmotiW entails an emotion classification contest based on audio and/or video, which mimics real world conditions. The main EmotiW's objective is to outline a common evaluation for emotion recognition. The challenge has been organized since 2013. The database used is Acted Facial Expressions in the Wild (AFEW), which stresses the relevance of its spontaneous nature compared to other corpus recorded on a controlled environment. It contains video and audio material [21][24]. Compared to CK+, AFEW replaces contempt emotion label by neutral one.

The 2014 EmotiW version results are defined as baselines. However, the video only is considered since no audio was used during the development of this project for recognition purposes. AFEW 4.0 was the database used.

The baseline result is 33.15%. It was calculated by means of a non-linear radial basis function (RBF) kernel based support vector machine (SVM) over the validation set [22].

The challenge's winner was LIU, Mengyi, et al. [60]. In this work, many approaches are combined: Histogram of Oriented Gradients (HOG), Dense Scale- Invariant Feature Transform (Dense SIFT), and Deep CNN feature (CNN). In order to provide a better comparison, only the results achieved using CNN on the validation set are reported: 39.35% when trained against ImageNet corpus.

In the 2015 challenge, a new sub-challenge was added into the contest. This sub-challenge aims to classify facial emotions on static images. Thus, a new data set was elaborated: static facial expression in the wild (SFEW). The corpuses used were AFEW 5.0 and SFEW 2.0 [25]. Given that static images are also used on this project, the baseline results reported are the ones on that particular sub-challenge. The baseline value on the validation set is 35.93%. It was also computed using a non-linear SVM. In this edition, the first place went to KIM, Bo-Kyeong, et al. [50]. Its approach was a pure convolutional neural network one, achieving 44.7% performance.

6.1.3 State-of-the-art

One of the latest works on facial emotion detection corresponds to Ramachandran et al. [77]. Its research achieved 97% of prediction accuracy over CK+. It is good to emphasize that this work employed a mixture of ANN with feature engineering.

6.2 Hardware

In order to run the experiments, two machines were used: a server set up on Amazon Web Services (AWS) and a Lenovo T440p laptop. The main difference between both is that the AWS provided GPU processing.

The AWS server specification is as follows, as described by [2]:

- Intel Xeon E5-2670 (Sandy Bridge) processors with 8 vCPU
- Nvidia grid K520, with 1,536 CUDA cores and 4GB of video memory
- 15GB RAM and 60GB SSD

While the laptop technical conditions are the following:

- 4th Gen Intel Core i7-4700MQ
- 16GB RAM
- 256GB SSD

The operative system on both was Ubuntu Server 14.04 LTS; other software used include: Python 2.7, and TensorFlow 0.7.

6.3 First phase results

In Table 6.1 on page 45, a parameter summary is introduced. The experiments on this phase are conducted following this configuration, except when explicitly stated otherwise. These parameters are described in detail in section 5.1.

6.3.1 Network loss and learning rate

The loss chart presented in Figure 6.1 on page 45 shows how the cost function is minimized over training steps.

During the first 450 steps, the loss is reduced smoothly. After that point, the curve starts to converge. On the step 800, the loss value is 0.79; while on step 900, loss value is 0.77. It is just a reduction of 0.02 on a hundred steps. Given this behavior, it can be said that the model has converged. It means that a longer training won't reduce the network's loss substantially. For instance, using a smaller learning rate

Parameter	Value
Batch size	64
Output channels	64
Kernel size	3x3
Kernel stride	1
Padding	same
Pooling	max pooling
Pool size	3x3
Pool stride	2
FC1 neurons	384
FC2 neurons	192
Dropout	0.99
Activation function	ReLU
Number of classes	6
Cost function	cross entropy
Optimizer	momentum
Momentum	0.9
Learning rate	0.1

Table 6.1: Configuration summary for first phase experiment

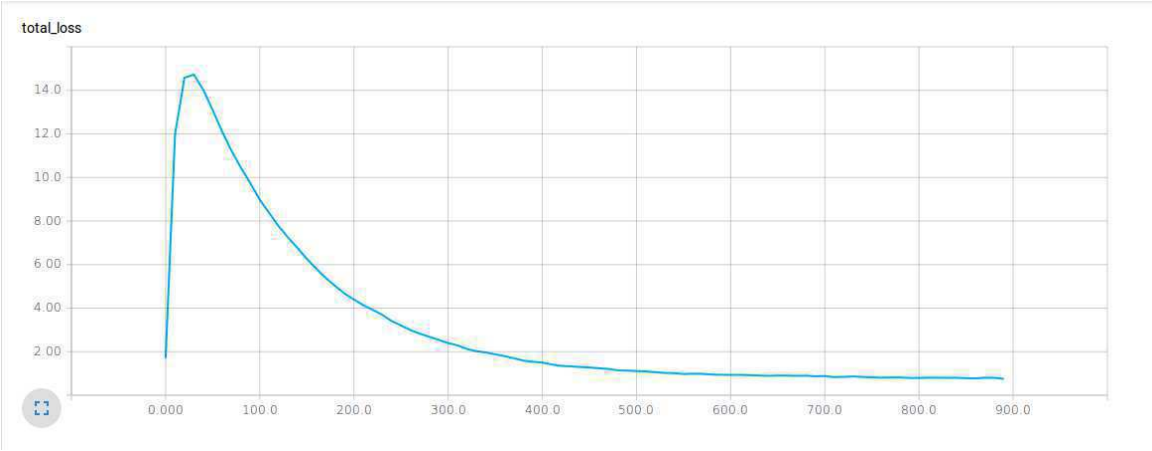

Figure 6.1: Total loss over 900 training steps with learning rate at 0.1

(0.01) provides a different curve as it can be seen in Figure 6.2 on page 46. The model has not converged yet, so a longer training time is necessary. While a longer training time might improve network's accuracy; exposing a network to a prolonged training will ended up in overfitting. The average training time was around 10 minutes and 37 seconds.

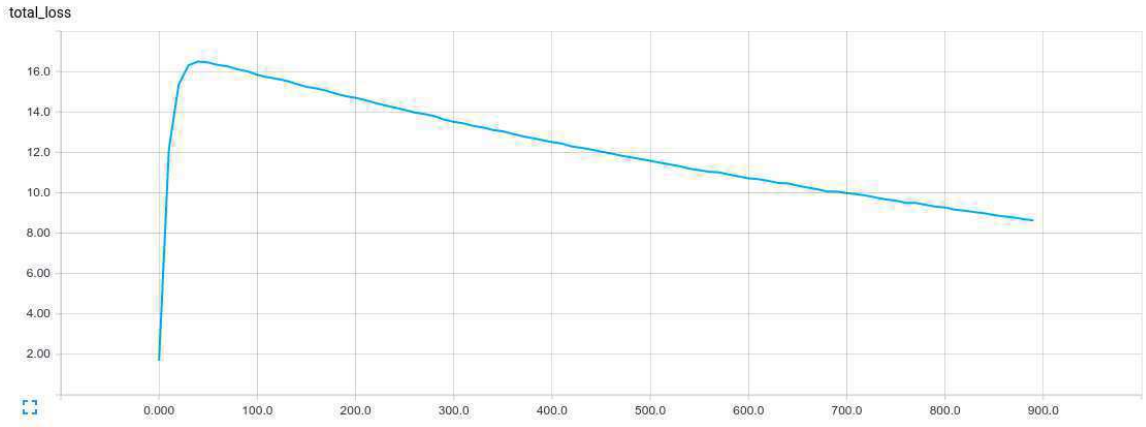

Figure 6.2: Total loss over 900 training steps with learning rate at 0.01

6.3.2 Classification accuracy

The accuracy of the network on the training set is reported in Table 6.2 on page 46. The results reported correspond to classification on the Top-1 and Top-3 prediction.

On Top-1 predictions, the highest accuracy was 72.5%, and the average one is 70.54%. On the contrary, for Top-3 predictions, the highest accuracy was 80%; with an average of 77.9%. Table 6.3 on page 47 shows that the results using configuration on table 6.1 performed better than the random guess baselines.

	Run									
Top	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	70.0	71.1	72.5	70.8	68.1	71.9	71.1	71.3	68.4	70.2
3	75.8	78.4	77.8	77.7	80.0	77.8	78.7	76.4	79.1	77.3

Table 6.2: Network classification accuracy on 6 emotion labels on CK+ dataset for learning rate set to 0.1

6.3.3 The dropout effect

Table 6.4 on page 47 shows the network accuracy when the dropout is set to 0.5. As can be seen, the accuracy on Top-1 prediction is reduced compared to results showed in table 6.2. The accuracy average is 57.24%. It is around 13% less accuracy compared to average when dropout was set to 0.99, 70.54%.

While on Top-3 prediction, there is no such effect. Results are quite similar to the ones displayed at table 6.2. The average is 77.5%, while the original average reported was 77.9%.

Model	Accuracy
6-label random guess	16.67
Most frequent label	25.38
EmotiW 2014's winner	39.35
EmotiW 2015's winner	44.70
3 most frequent labels	64.50
Top-1 prediction	70.54
Top-3 prediction	77.90
Ramachandran	97.00

Table 6.3: Result comparison against proposed baselines

	Run									
Top	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	56.2	57.0	58.8	58.1	58.9	55.6	57.3	57.7	55.9	56.9
3	75.9	76.9	78.9	77.2	78.6	77.5	78.7	78.0	77.7	76.1

Table 6.4: Network accuracy when dropout is set to 0.5

	Adam Optimizer / Run									
Top	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	64.4	62.2	67.0	64.2	66.6	64.8	66.4	65.9	65.2	63.4
3	75.3	75.8	75.5	74.7	75.6	76.1	76.1	78.1	76.2	76.1

Table 6.5: Classification accuracy when using Adam optimizer

6.3.4 Different optimizers

Besides momentum, a couple of new optimizers were tried: Adam Optimizer and FTRL Optimizer.

When using Adam Optimizer, the model already converges around the 100th step, as can be seen in figure 6.3 on page 48. However, results presented on table 6.5 show that the average prediction accuracy is 65.01% on Top-1 prediction and 75.95% on Top-3 prediction. Both values are slightly lower than the ones achieved using momentum.

Figure 6.4 reveals that using FTRL optimizer caused an early convergence around the 30th step. The oscillation from that step until the end is not smooth. Regarding classification accuracy, an average value of 19.06% on Top-1 prediction was achieved. This value is on the range of the random guess baselines. However, Top-3 prediction achieved 79.02% on Top-3 prediction, which is slightly better than the Top-3 prediction reported using momentum.

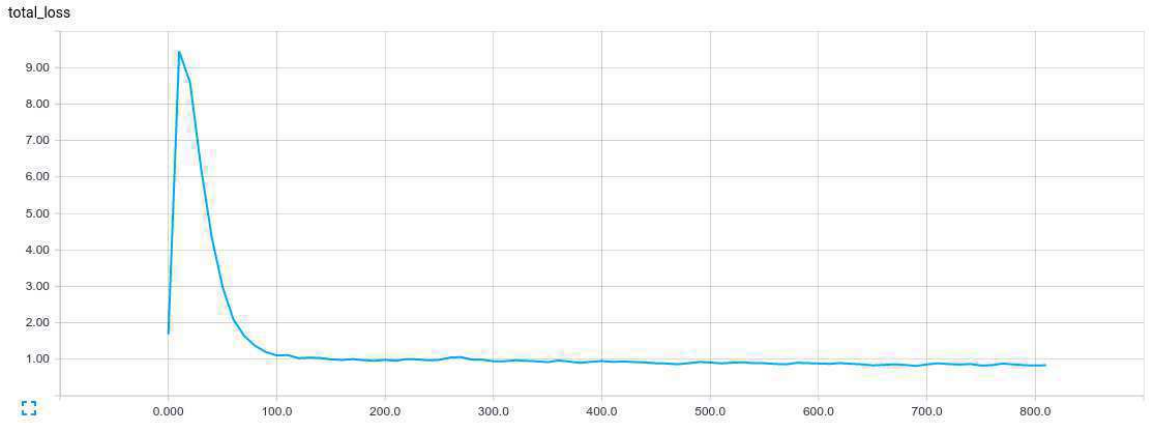

Figure 6.3: Total loss over 900 training steps using Adam optimizer

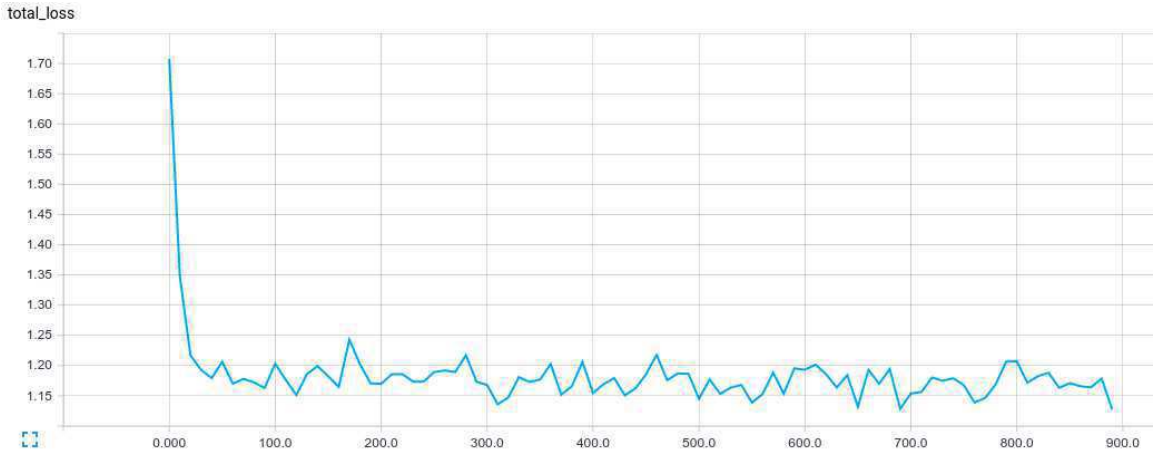

Figure 6.4: Total loss over 900 training steps using FTRL optimizer

	FTRL Optimizer / Run									
Top	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	18.6	18.9	18.6	20.0	20.3	18.0	18.9	19.2	19.8	18.3
3	78.9	79.2	79.5	79.7	78.9	79.2	79.1	78.7	78.3	78.7

Table 6.6: Classification accuracy when using FTRL optimizer

6.4 Second phase results

The second phase involved evaluating the network's accuracy using AM-FED dataset. The network was trained using the parameters displayed in table 6.1 on page 45.

	AMFED / Run									
Top	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	53.6	54.5	53.7	52.7	53.1	55.5	55.5	52.8	53.7	52.7
3	89.7	89.1	89.5	88.7	89.4	88.6	89.8	89.5	89.2	89.7

Table 6.7: Network accuracy using AM-FED dataset.

Model	**Accuracy**
6-label random guess	16.67
Most frequent label	25.38
EmotiW 2014's winner	39.35
EmotiW 2015's winner	44.70
Top-1 prediction (AMFED)	53.78
3 most frequent labels	64.50
Top-1 prediction (CK+)	70.54
Top-3 prediction (CK+)	77.90
Top-3 prediction (AMFED)	89.32
Ramachandran	97.00

Table 6.8: Result comparison including Phase 2 results against proposed baselines

6.4.1 Classification accuracy

The accuracy of the network on the AM-FED set is reported in Table 6.7 on page 49. The results reported correspond to classification on the Top-1 and Top-3 prediction.

On Top-1 predictions, the highest accuracy was 55.5%, and the average one is 53.78%. On the contrary, for Top-3 predictions, the highest accuracy was 89.8%; with an average of 89.32%. Table 6.8 on page 49 shows that the results using configuration on table 6.1 performed significantly better than the random guess baselines. Moreover, Top-3 prediction was better than the one reported on CK+. Wilcoxon test was executed over both result sets. The p-value was less than 0.001, which means that Top-3 prediction is still significantly better, even though the difference is not as large as expected.

Chapter 7

Discussion

In this chapter, results are discussed to get an insight of how different parameters affect the network's prediction accuracy. This illustrates the complexity on the different components that articulate a machine learning experiment.

7.1 First phase

Some interpretations on the results presented in section 6.3 are introduced on the following pages.

7.1.1 Classification accuracy

When comparing the results provided in section 6.3 on page 44 against our baselines, it can be determined that it achieved much better than a random guess for six emotion labels and just assigning the most frequent label.

The comparison with EmotiW might be tricky since different corpora are used. The 2014 winner provided some ideas about improving its classification accuracy. One of those was to try on few difficult categories; while the other one was focused on exploring more effective fusion strategy. As it is explained later, the use of dropout boosted the accuracy performance on our research. The use of dropout is not reported on their experiments. The differences between Top-1 and Top-3 predictions are not as large as it would have been expected. The average difference is 7.36%.

The use of feature engineering has a relevant influence over the classification accuracy. In the **Related Work** section, the research by Kotsia and Pitas [53] achieved 99.7% on facial expression recognition. Moreover, Ramachandran et al. [77] got

classification results around 97%. These numbers largely outperformed the method introduced in this paper. This evidence stresses the fact that DL needs a large corpus when a supervised learning experiment is set up. However, some strategies were tried to boost the network accuracy on this dataset. Besides dropout, a smaller network and image rescaling from 640 x 480 pixels to 32 x 32 pixels also helped on improving the prediction accuracy.

7.1.2 Network loss and learning rate

Figures 6.1 and 6.2 show learning rate importance when minimizing the cost function. Moreover, it provides a good way to determine the training duration.

An accurate training duration is important to avoid network overfitting. Overfitting reduces the network ability to generalize. In this context, generalizing means to predict with the same accuracy on examples not used during training.

From a practical view, reducing training duration has some benefits. One of them is that new ideas can be easily tried on a network. Another benefit is resource saving. This impact can be further notice on a large research centre where a longer training time involves less productivity on several people. GPU use on DL is true evidence of the community's concern regarding training time reduction.

The learning rate can also lead to loss explosion. This happens when the learning rate is set to a value too high for the network. During the beginning of this project, loss explosion was a common issue. Several learning rate values were used, but they just prolonged it for happening. The solution for this problem was to return the output of the softmax layer as a probability distribution.

Finally, the network loss influences the classification accuracy. However, it would be hard to mathematically defined an equation that relates both values since the initial loss value is related to the network topology.

7.1.3 The dropout effect

The dropout value variation had a great influence over top-1 prediction accuracy. As reported in section 6.3.3 on page 46, reducing the dropout value to 0.5 lessened the classification's accuracy by 13%. However, it did not have the same effect over the top-3 prediction.

Dropout is directly connected on how neurons on the fully connected layers classify features. It seems that each neuron was forced to learn features for each class. Big

unit activations were discarded given the high droput probability (0.99). When this value was reduced to 0.5, the effect of features that were more common to all the categories was stronger. Thus, the network was not able to classify correctly since it was biased by those features.

7.1.4 Different optimizers

The experience of using momentum, Adam optimizer and FTRL provided some insights on the learning process, as well. Momentum has a balanced training time, classification accuracy tradeoff Moreover, it only adds a new hyperparameter: the momentum value. The best results during this research for Top-1 prediction were achieved using momentum set to 0.9.

Another optimizer used on this research was Adam's one. Adam optimizer converged around 400 steps earlier compared to momentum as reported on figures 6.1 and 6.3. However, there was some loss on the classification accuracy. An average difference of 5.53% on Top-1 prediction and 1.95% on Top-3 prediction. Based on these results, it seems that momentum allow units to learn more specific features that let them achieve a better accuracy. Though, a scenario where Adam optimizer's use might be more beneficial is when there are not enough resources for a prolonged training time and some accuracy can be lost. One benefit of this optimizer is that it does not add any additional hyperparameter.

The use of FTRL brought two different results when applied on Top-1 and Top-3 prediction. Its accuracy on Top-1 performed better only when compared to random guess. The average difference between both was 2.39%. However, its Top-3 accuracy reached 79.02%. This difference value is marginally better than the one achieved using momentum (77.90%), 1.12%. A possible explanation is that FTRL got stuck on a local minima. By staying most of the time in this region, as can be seen in figure 6.4, some features might have become really influential. Then, a class containing this feature might always be the winner on Top-1 prediction.

7.2 Second phase

Results reported in section 6.4 on page 48 are discussed on the following section.

7.2.1 Classification accuracy

Evaluating the network trained on **Phase 1** using AM-FED provided with information about how the network generalization. The main difference between both datasets is that AM-FED was captured on the wild. Therefore, faces are not always on the center of the image, and with the participants looking in front of the camera. Also, no manual revision of the images was performed. Some images might not be considered happiness for a human.

As it was previously stated, only one label was used for evaluation: the happiness emotion. The network's Top-1 accuracy for that particular emotion was 53.78%. The accuracy reduction was expected because of the differences explained on the previous paragraph. However, the Top-3 accuracy achieved 89.32%. It means that the network was able to learn features for smiling people; even on a different dataset. Wilcoxon test was executed over both result sets. The p-value as less than 0.001, which means that Top-3 prediction is still significantly better, even though the difference is not as large as expected.

Chapter 8

Conclusions

8.1 Conclusions

In this project, a research to classify facial emotions over static facial images using deep learning techniques was developed. This is a complex problem that has already been approached several times with different techniques [26][53][54][83][97]. While good results have been achieved using feature engineering, this project focused on feature learning, which is one of DL promises.

While the results achieved were not state-of-the-art, they were slightly better than other techniques including feature engineering. It means that eventually DL techniques will be able to solve this problem given an enough amount of labeled examples. While feature engineering is not necessary, image pre-processing boosts classification accuracy. Hence, it reduces noise on the input data.

Nowadays, facial emotion detection software includes the use of feature engineering. A solution totally based on feature learning does not seem close yet because of a major limitation: the lack of an extensive dataset of emotions. For instance, ImageNet contest uses a dataset containing 14 197 122 images. By having a larger dataset, networks with a larger capability to learn features could be implemented. Thus, emotion classification could be achieved by means of deep learning techniques.

8.2 Future Work

Further improvement on the network's accuracy and generalization can be achieved through the following practices. The first one is to use the whole dataset during the

optimization. Using batch optimization is more suitable for larger datasets. Another technique is to evaluate emotions one by one. This can lead to detect which emotions are more difficult to classify. Finally, using a larger dataset for training seems beneficial.

Nowadays, optimizers work on a stochastic basis because of large datasets (millions of examples). However, this was true for our project. Given a limited dataset, trying on the whole dataset could have lead to a better feature learning. Also, the use of some optimizers reported on this research would have had a different behaviour. This behaviour can be displayed on the loss curve having a smoother shape or by avoiding an early convergence.

Second, due to time constraints, it was not possible to evaluate each emotion. On this way, it would have been possible to detect which emotions are easier to classify, as well as, which ones are more difficult. Moreover, pre-training on each emotion could lead to a better feature learning. After that, the network could have received this learning (transfer learning). This could have improved on reducing the training time; as well as, minimizing to a higher degree the cost function.

Also, using a larger dataset can lead to higher scale training. Training into a larger input space and for more time improves the network accuracy. A larger training scale allows the network to learn more relevant features. If this is not achieved, feature engineering is still required for this task. However, such a dataset might not exist nowadays. Using several datasets might be a solution, but a careful procedure to normalize them is required.

Finally, using full dataset for training, pre-training on each emotion, and using a larger dataset seem to have the possibility to improve the network's performance. Thus, they should be addressed in future research on this topic.

Appendix A

Create BIN file code

Source code to generate the BIN file for the extended Cohn - Kanade dataset.

```
import os
import numpy as np
from PIL import Image

def crop_image(path, witdh, height):
        offset_h = 10
        for root, dirs, files in os.walk(path, True):
                for name in files:
                        image = Image.open(os.path.join(root, name))
                        w, h = image.size
                        offset_w = (w - witdh) / 2
                        image.crop((0 + offset_w, 0 + offset_h, witdh + offset_w,
                                    height + offset_h)).save(os.path.join(root, name))


def convert_to_greyscale(path):
        for root, dirs, files in os.walk(path, True):
                for name in files:
                        image = Image.open(os.path.join(root, name))
                        #image.convert('LA').save(os.path.join(root, name))
                        image.convert('L').save(os.path.join(root, name))

def rename_filename(filename):
        #end_position = 8
        end_position = 17
        return filename[:end_position]

def search_label_dictionary(label_dict, filename):
        name = rename_filename(filename)
         for index, value in label_dict:
                 if index == name:
                         return value

def generate_label_dictionary(path):
```

```
35          label = []
36          for root, dirs, files in os.walk(path, True):
37                  for name in files:
38                          f = open(os.path.join(root, name), 'r')
39                          label.append([rename_filename(name), int(float(f.read()))])
40          return label
41
42  def set_record_size(label, witdh, height, channel):
43          return label + (witdh * height * channel)
44
45  def generate_bin(path, total_images, record_size, label_dict):
46          result = np.empty([total_images, record_size], dtype=np.uint8)
47          i = 0
48          for root, dirs, files in os.walk(path, True):
49                  for name in files:
50                          label = search_label_dictionary(label_dict, name)
51                          if label is not None:
52                                  image = Image.open(os.path.join(root, name))
53                                  image_modifed = np.array(image)
54                                  grey = image_modifed[:].flatten()
55                                  result[i] = np.array(list([label]) + list(grey), np.uint8)
56                                  i = i + 1
57          result.tofile("/home/neo/projects/deepLearning/data/kh.bin")
58
59  def main(argv=None):  # pylint: disable=unused-argument
60          label_path = "/home/neo/projects/deepLearning/data/label/"
61          image_path = "/home/neo/projects/deepLearning/data/image/"
62          #total_images = 4895
63          total_images = 327
64          witdh = 640
65          height = 480
66          channel = 1
67          label = 1
68          #crop_image(image_path, witdh, height)
69          #convert_to_greyscale(image_path)
70          label_dict = generate_label_dictionary(label_path)
71          record_size = set_record_size(label, witdh, height, channel)
72          generate_bin(image_path, total_images, record_size, label_dict)
73
74  if __name__ == '__main__':
75          main()
```

Appendix B

Generate labeled images from AM - FED videos

Source code to generate labeled images from AM - FED videos.

```
import csv
import cv2
import os

"""
1 (e) Angry    - AU 4+5+15+17,          (4/5)
2 (f) Contempt - AU 14,                  (1/1)
3 (a) Disgust  - AU 1+4+15+17,          (3/4)
4 (d) Fear     - AU 1+4+7+20,           (1/4)
5 (b) Happy    - AU 6+12+25,           (Smile)
6 (g) Sadness  - AU 1+2+4+15+17  (4/5)
7 (c) Surprise - AU 1+2+5+25+27  (2/5)
SURPRISE AND FEAR GG
"""

def change_to_video_name(csv_name, suffix):
        return csv_name[:-10]+"."+suffix

def generate_frame(video_path, video_name, second, label, dest_path):
        vidcap = cv2.VideoCapture(os.path.join(video_path, video_name))
        vidcap.set(0, int(second*1000))
        success, image = vidcap.read()
        if success:
                cv2.imwrite(os.path.join(dest_path, video_name+"_"+str(second)+"_"+str(label)+".jpg"), image)

def check_angry(content):
        baseline = 50
        #disgust = ["AU4", "AU15", "AU17"]
        sadness = ["AU2", "AU4", "AU15", "AU17"]
        #angry = ["AU4", "AU5", "AU15", "AU17"]
        label = 1
```

```
32          emotion_time = content[0][1]
33          emotion = []
34          for c in content:
35                  for h in sadness:
36                          if c[0] == h:
37                                  emotion.append(c[1])
38          print emotion
39          factor = sum(emotion)/len(sadness)
40          if factor >= baseline:
41                  return emotion_time, label
42
43  def check_contempt(content):
44          baseline = 100
45          contempt = ["AU14"]
46          label = 2
47          emotion_time = content[0][1]
48          for c in content:
49                  for h in contempt:
50                          if c[0] == h and c[1] >= baseline:
51                                  return emotion_time, label
52
53  def check_happiness(content):
54          baseline = 100
55          happiness = ["Smile"]
56          label = 5
57          emotion_time = content[0][1]
58          for c in content:
59                  for h in happiness:
60                          if c[0] == h and c[1] >= baseline:
61                                  return emotion_time, label
62
63  def get_content(header, row):
64          content = row[0].split(',')
65          result = []
66          for h in header:
67                  result.append([h[0], float(content[h[1]])])
68          return result
69
70  def get_header_au(row):
71          rules = ["Time", "Smile", "AU"]
72          header = row[0].split(',')
73          result = []
74          i = 0
75          for h in header:
76                  if h in rules or h[0:2] in rules:
77                          result.append([h, i])
78                  i = i + 1
79          return result
80
81  def process_video(csv_path, video_path, dest_path, suffix):
82          for root, dirs, files in os.walk(csv_path, True):
83                  for name in files:
84                          with open(os.path.join(root, name), 'rU') as csvfile:
```

```
85                                  reader = csv.reader(csvfile, delimiter=' ', quotechar='|')
86                                  for row in reader:
87                                          if reader.line_num == 1:
88                                                  header = get_header_au(row)
89                                          else:
90                                                  if len(header) > 1:
91                                                          content = get_content(header, row)
92                                                          emotion = check_angry(content)
93                                                          if emotion is not None:
94                                                                  generate_frame(video_path,
95                                                                          change_to_video_name(name, suffix),
96                                                                          emotion[0], emotion[1], dest_path)
97
98  def main(argv=None):  # pylint: disable=unused-argument
99          csv_path = "/home/neo/projects/deepLearning/data/au_labels"
100         video_path = "/home/neo/projects/deepLearning/data/videos"
101         dest_path = "/home/neo/projects/deepLearning/data/amfed/sadness"
102         suffix = "flv"
103         process_video(csv_path, video_path, dest_path, suffix)
104
105
106 if __name__ == '__main__':
107         main()
```

Appendix C

Network model TF implementation

Source code of the network model using TF.

```
def inference(images, eval=False):
  # conv1
  with tf.variable_scope('conv1') as scope:
    kernel = _variable_with_weight_decay('weights', shape=[3, 3, 1, 64],
                                         stddev=1e-4, wd=0.0)
    conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
    biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.0))
    bias = tf.nn.bias_add(conv, biases)
    conv1 = tf.nn.relu(bias, name=scope.name)
    _activation_summary(conv1)

  pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1],
                         strides=[1, 2, 2, 1], padding='SAME', name='pool1')

  # local1
  with tf.variable_scope('local1') as scope:
    # Move everything into depth so we can perform a single matrix multiply.
    dim = 1
    for d in pool1.get_shape()[1:].as_list():
      dim *= d
    reshape = tf.reshape(pool1, [FLAGS.batch_size, dim])
    weights = _variable_with_weight_decay('weights', shape=[dim, 384],
                                          stddev=0.04, wd=0.004)
    biases = _variable_on_cpu('biases', [384], tf.constant_initializer(0.1))
    local1 = tf.nn.relu(tf.matmul(reshape, weights) + biases, name=scope.name)
    _activation_summary(local1)
    if eval is False:
      local_drop1 = tf.nn.dropout(local1, DROPOUT)
      _activation_summary(local_drop1)

  # local2
  with tf.variable_scope('local2') as scope:
    weights = _variable_with_weight_decay('weights', shape=[384, 192],
                                          stddev=0.04, wd=0.004)
```

```
35      biases = _variable_on_cpu('biases', [192], tf.constant_initializer(0.1))
36      if eval is False:
37        local_drop2 = tf.nn.dropout(tf.nn.relu(tf.matmul(local_drop1, weights) + biases, name=scope.name), DROPOUT)
38        _activation_summary(local_drop2)
39      else:
40        local2 = tf.nn.relu(tf.matmul(local1, weights) + biases, name=scope.name)
41        _activation_summary(local2)
42
43    # softmax, i.e. softmax(WX + b)
44    with tf.variable_scope('softmax_linear') as scope:
45      weights = _variable_with_weight_decay('weights', [192, NUM_CLASSES],
46                                            stddev=1/192.0, wd=0.0)
47      biases = _variable_on_cpu('biases', [NUM_CLASSES],
48                                tf.constant_initializer(0.0))
49      if eval is False:
50        softmax_linear = tf.nn.softmax(tf.matmul(local_drop2, weights) + biases, name=scope.name)
51      else:
52        softmax_linear = tf.nn.softmax(tf.matmul(local2, weights) + biases, name=scope.name)
53      _activation_summary(softmax_linear)
54
55    return softmax_linear
```

Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems.*, pages 1097–1105, 2012.

[2] Amazon Web Services, Inc. Ec2 instance types. `http://aws.amazon.com/ec2/instance-types/`, 2016. [Online; Accessed 27 May 2016].

[3] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. `http://cs231n.github.io/convolutional-networks/`, 2015. [Online; Accessed 24 May 2016].

[4] Andrej Karpathy. Stanford university: Cs231n convolutional neural networks for visual recognition. `http://cs231n.github.io/neural-networks-1`, 2016. Online; Accessed 07 June 2016.

[5] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar and Li Fei-Fei. Large-scale video classication with convolutional neural networks. In *Proceedings of International Computer Vision and Pattern Recognition (CVPR 2014)*, 2014.

[6] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *arXiv preprint arXiv:1606.04474*, 2016.

[7] José Manuel Benítez, Juan Luis Castro, and Ignacio Requena. Are artificial neural networks black boxes? *Neural Networks, IEEE Transactions on*, 8(5):1156–1164, 1997.

[8] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and

Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[9] Stefano Berretti, Alberto Del Bimbo, and Pietro Pala. Superfaces: A super-resolution model for 3d faces. In *ECCV Workshops (1)*, pages 73–82, 2012.

[10] Vinay Bettadapura. Face expression recognition and analysis: the state of the art. *arXiv preprint arXiv:1203.6722*, 2012.

[11] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[12] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[13] Manfred Clynes and Yehudi Menuhin. *Sentics: The touch of emotions*. Anchor Press Garden City, NY, 1977.

[14] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *Proceedings of the 30th international conference on machine learning*, pages 1337–1345, 2013.

[15] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[16] Conner DiPaolo. Perceptron. `https://github.com/cdipaolo/goml/tree/master/perceptron`, 2015. [Online; Accessed 23 May 2016].

[17] Xiaodong Cui, Vaibhava Goel, and Brian Kingsbury. Data augmentation for deep neural network acoustic modeling. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 23(9):1469–1477, September 2015.

[18] R. E. Cytowic, 1994. Personal Communication.

[19] Antonio R Damasio. *Descartes' error*. Random House, 2006.

[20] Daniel McDuff, Rana El Kaliouby, Thibaud Senechal, May Amr, Jeffrey Cohn, Rosalind Picard. Affectiva mit facial expression dataset (am-fed): Naturalistic and spontaneous facial expressions collected in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops.*, 2013.

[21] Abhinav Dhall et al. Collecting large, richly annotated facial-expression databases from movies. 2012.

[22] Abhinav Dhall, Roland Goecke, Jyoti Joshi, Karan Sikka, and Tom Gedeon. Emotion recognition in the wild challenge 2014: Baseline, data and protocol. In *Proceedings of the 16th International Conference on Multimodal Interaction*, pages 461–466. ACM, 2014.

[23] Abhinav Dhall, Roland Goecke, Jyoti Joshi, Michael Wagner, and Tom Gedeon. Emotion recognition in the wild challenge 2013. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 509–516. ACM, 2013.

[24] Abhinav Dhall, Roland Goecke, Simon Lucey, and Tom Gedeon. Static facial expression analysis in tough conditions: Data, evaluation protocol and benchmark. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 2106–2112. IEEE, 2011.

[25] Abhinav Dhall, OV Ramana Murthy, Roland Goecke, Jyoti Joshi, and Tom Gedeon. Video and image based emotion recognition challenges in the wild: Emotiw 2015. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pages 423–426. ACM, 2015.

[26] Fadi Dornaika and Franck Davoine. Simultaneous facial action tracking and expression recognition in the presence of head motion. *International Journal of Computer Vision*, 76(3):257–281, 2008.

[27] John F Dovidio, Jane Allyn Piliavin, David A Schroeder, and Louis Penner. *The social psychology of prosocial behavior.* Lawrence Erlbaum Associates Publishers, 2006.

[28] Paul Ekman and Wallace V Friesen. Facial action coding system. 1977.

[29] Andries P Engelbrecht. *Computational intelligence an introduction.* John Wiley & Sons, Chichester, England Hoboken, NJ, 2nd edition, 2007.

[30] Irfan Aziz Essa. *Analysis, interpretation and synthesis of facial expressions.* PhD thesis, Citeseer, 1994.

[31] Geoffrey Hinton et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[32] KDnuggets Exclusive. Interview with Yann LeCun, Deep Learning Expert, Director of Facebook AI Lab. `http://www.kdnuggets.com/2014/02/exclusive-yann-lecun-deep-learning-facebook-ai-lab.html`. Accessed: 2016-02-24.

[33] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[34] Glorot, X., Bordes, A., & Bengio, Y. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[35] Google Research. Research blog: A picture is worth a thousand (coherent) words: building a natural description of images. `https://research.googleblog.com/2014/11/a-picture-is-worth-thousand-coherent.html`, 2014. [Online; Accessed 06 June 2016].

[36] Anna Gorbenko and Vladimir Popov. On face detection from compressed video streams. *Applied Mathematical Sciences*, 6(96):4763–4766, 2012.

[37] Simon O. Haykin. *Neural Networks and Learning Machines.* Prentice Hall., 3rd edition, 2008.

[38] D.O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Psychology Press., new ed edition, 2002.

[39] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 593–605. IEEE, 1989.

[40] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[41] Minh Hoai. Regularized max pooling for image categorization. In *BMVC*, volume 2, page 6, 2014.

[42] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.

[43] Howard, J. Jeremy howard: The wonderful and terrifying implications of computers that can learn. `https://www.ted.com/talks/jeremy_howard_the_wonderful_and_terrifying_implications_of_computers_that_can_learn`, December 2014. Accessed: 2016-02-10.

[44] David H. Hubel and Torsten N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, 195, 1968.

[45] Ilya Sutskever, Oriol Vinyals, Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems.*, pages 3104–3112, 2014.

[46] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[47] Philip N Johnson-Laird and Eldar Shafir. The interaction between reasoning and decision making: an introduction. *Cognition*, 49(1):1–9, 1993.

[48] K. Han, D. Yu, and I. Tashev. Speech emotion recognition using deep neural network and extreme learning machine. In *Proceedings Interspeech*, 2014.

[49] Kanade, T., Cohn, J. F., & Tian, Y. Comprehensive database for facial expression analysis. In *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG'00)*, pages 46–53, Grenoble, France., 2000.

[50] Bo-Kyeong Kim, Hwaran Lee, Jihyeon Roh, and Soo-Young Lee. Hierarchical committee of deep cnns with exponentially-weighted decision fusion for static

facial expression recognition. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pages 427–434. ACM, 2015.

[51] Y. Kim, H. Lee, and E. M. Provost. Deep learning for robust feature generation in audiovisual emotion recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3687–3691, May 2013.

[52] Kon Mamadou Tadiou. The future of human evolution - artificial neural networks. `http://futurehumanevolution.com/artificial-intelligence-future-human-evolution/artificial-neural-networks`, 2010. [Online; Accessed 24 May 2016].

[53] Irene Kotsia, Ioan Buciu, and Ioannis Pitas. An analysis of facial expression recognition under partial facial image occlusion. *Image and Vision Computing*, 26(7):1052–1067, 2008.

[54] Irene Kotsia and Ioannis Pitas. Facial expression recognition in image sequences using geometric deformation features and support vector machines. *Image Processing, IEEE Transactions on*, 16(1):172–187, 2007.

[55] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[56] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989.

[57] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE, 86(11)*, pages 2278–2324, 1998.

[58] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, University of Helsinki, Finland, 1970.

[59] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976.

[60] Mengyi Liu, Ruiping Wang, Shaoxin Li, Shiguang Shan, Zhiwu Huang, and Xilin Chen. Combining multiple kernel methods on riemannian manifold for

emotion recognition in the wild. In *Proceedings of the 16th International Conference on Multimodal Interaction*, pages 494–501. ACM, 2014.

[61] Lucey, P., Cohn, J. F., Kanade, T., Saragih, J., Ambadar, Z., & Matthews, I. The extended cohn-kanade dataset (ck+): A complete expression dataset for action unit and emotion-specified expression. In *Proceedings of the Third International Workshop on CVPR for Human Communicative Behavior Analysis (CVPR4HB 2010)*, pages 94–101, San Francisco, USA., 2010.

[62] Jiang-Jing Lv, Cheng Cheng, Guo-Dong Tian, Xiang-Dong Zhou, and Xi Zhou. Landmark perturbation-based data augmentation for unconstrained face recognition. *Signal Processing: Image Communication*, 2016.

[63] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, page 1, 2013.

[64] Martn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Vigas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org, 2015.

[65] G. Mather. *Essentials of Sensation and Perception*. Foundations of Psychology. Taylor & Francis, 2014.

[66] James D. McCaffrey. Why You Should Use Cross-Entropy Error Instead Of Classification Error Or Mean Squared Error For Neural Network Classifier Training. `https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-me` Accessed: 2016-05-15.

[67] Walter. McCulloch, Warren S.; Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics.*, 5:115 – 133, 1943.

[68] Michael Cooney. Ibm: In the next 5 years computers will learn, mimic the human senses — network world. http://www.networkworld.com/article/2162228/data-center/ibm--in-the-next-5-years-computers-will-learn--mimic-the-human-senses.html, 2016. [Online; Accessed 06 June 2016].

[69] Rui Min, Neslihan Kose, and Jean-Luc Dugelay. Kinectfacedb: A kinect database for face recognition. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, 44(11):1534–1548, Nov 2014.

[70] Marvin Minsky. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press., expanded edition, 1987.

[71] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[72] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[73] NVIDIA. What is gpu computing? — high-performance computing. http://www.nvidia.com/object/what-is-gpu-computing.html, 2015. Online; Accessed 07 June 2016.

[74] Pentland, A. S. Social signal processing [exploratory dsp]. *Signal Processing Magazine, IEEE*, 24(4):108–111, 2007.

[75] Rosalind W Picard. Affective computing. 1995.

[76] Bellet PS and Maloney MJ. The importance of empathy as an interviewing skill in medicine. *JAMA*, 266(13):1831–1832, 1991.

[77] Vedantham Ramachandran and E Srinivasa Reddy. Facial expression recognition with enhanced feature extraction using pso & ebpnn. *International Journal of Applied Engineering Research*, 11(10):6911–6915, 2016.

[78] Rensselaer Polytechnic Institute. Automatic facial action units recognition. https://www.ecse.rpi.edu/~cvrl/tongy/aurecognition.html, 2015. Online; Accessed 20 June 2016.

[79] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

[80] Rosalind W. Picard. *Affective Computing*. MIT Press., reprint edition, 2000.

[81] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[82] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[83] Nicu Sebe, Michael S Lew, Yafei Sun, Ira Cohen, Theo Gevers, and Thomas S Huang. Authentic facial expression analysis. *Image and Vision Computing*, 25(12):1856–1863, 2007.

[84] Sejnowski, T. J., & Rosenberg, C. R. Nettalk: A parallel network that learns to read aloud. *MIT Press*, pages 661–672, 1988.

[85] Senechal, T., McDuff, D., & Kaliouby, R. Facial action unit detection using active learning and an efficient non-linear kernel approximation. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 10–18, 2015.

[86] Alan Senior, Georg Heigold, Marc'Aurelio Ranzato, and Ke Yang. An empirical study of learning rates in deep neural networks for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6724–6728. IEEE, 2013.

[87] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[88] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[89] Stanford University. Unsupervised feature learning and deep learning tutorial. `http://ufldl.stanford.edu/tutorial/supervised/`

OptimizationStochasticGradientDescent/, 2014. [Online; Accessed 24 May 2016].

[90] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1139–1147. JMLR Workshop and Conference Proceedings, May 2013.

[91] 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*, pages 1–9, 2015.

[92] TensorFlow. Convolutional Neural Networks. https://www.tensorflow.org/versions/r0.8/tutorials/deep_cnn/index.html. Accessed: 2016-05-15.

[93] Tim Dettmers. Deep learning in a nutshell: History and training. https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-history-training, 2015. [Online; Accessed 24 May 2016].

[94] Tim Dettmers. Understanding convolution in deep learning. http://timdettmers.com/2015/03/26/convolution-deep-learning/, 2015. [Online; Accessed 24 May 2016].

[95] Matthew Turk. Perceptive media: machine perception and human computer interaction. *CHINESE JOURNAL OF COMPUTERS-CHINESE EDITION-*, 23(12):1235–1244, 2000.

[96] Vandal, T., McDuff, D., & El Kaliouby, R. Event detection: Ultra large-scale clustering of facial expressions. In *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on Vol. 1*, pages 1–8, 2015.

[97] Jun Wang and Lijun Yin. Static topographic modeling for facial expression recognition and analysis. *Computer Vision and Image Understanding*, 108(1):19–34, 2007.

[98] Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

[99] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[100] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. Deep image: Scaling up image recognition. *arXiv preprint arXiv:1501.02876*, 22:388, 2015.

[101] Yaser Yacoob and Larry Davis. Computing spatio-temporal representations of human faces. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference On*, pages 70–75. IEEE, 1994.

[102] Omry Yadan, Keith Adams, Yaniv Taigman, and MarcAurelio Ranzato. Multi-gpu training of convnets. *arXiv preprint arXiv:1312.5853*, 9, 2013.

[103] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[104] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer vision–ECCV 2014*, pages 818–833. Springer, 2014.

[105] Matthew D Zeiler, Marc'Aurelio Ranzato, Rajat Monga, Min Mao, Kun Yang, Quoc Viet Le, Patrick Nguyen, Alan Senior, Vincent Vanhoucke, Jeffrey Dean, et al. On rectified linear units for speech processing. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3517–3521. IEEE, 2013.