A FACIAL EXPRESSION RECOGNITION APPLICATION DEVELOPMENT USING DEEP CONVOLUTIONAL NEURAL NETWORK FOR CHILDREN WITH AUTISM SPECTRUM DISORDER TO HELP IDENTIFY HUMAN EMOTIONS

by

Md Inzamam Ul Haque, B.S.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Engineering
August 2019

Committee Members:

Damian Valles, Chair

Maria Resendiz

George Koutitas

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

To my parents, who supported me throughout my life and for whom I have come this far.

ACKNOWLEDGEMENTS

I am thankful to each of the members of my thesis committee who has provided me extensive personal and professional guidance and taught me a great deal about both scientific research and life in general. I would especially like to thank Dr. Damian Valles, the chairman of my committee as well as my thesis supervisor. As my mentor, with his time, effort, guidance and vision, he has taught me more than I could ever give him credit for. He has worked actively to provide me with sufficient academic time, supported my career and research goals and shown me, by his example, what a good engineer should be. I would also like to thank Dr. Maria Resendiz for her continued support with the thesis resources and guidance. Actually, this thesis work would not have been possible without her as the thesis idea originated from her.

The most important part during the span of this thesis was the support and love from my family. I would like to thank my parents; whose love and care gave me strength and who are always supportive of what I pursue.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

ABSTRACT

In this thesis, a novel idea is presented, which is to teach young children with Autism Spectrum Disorder (ASD) to recognize human facial expressions through the help of computer vision and image processing. Universally, there are seven facial expressions categories: angry, disgust, happy, sad, fear, surprised and neutral. To recognize all these facial expressions and to predict the current mood of a person is a difficult task for a child. For a child with ASD, this problem presents itself in a more complex manner due to the nature of the disorder. The main goal of the thesis was to develop a deep Convolutional Neural Network (DCNN) for facial expression recognition, which can help young children with ASD to recognize facial expressions, using mobile devices. Previously, different neural network models and classifiers have been presented to achieve state of the art accuracy in this sector. Separately, different studies have been performed in studying the ability and performance of children with ASD for recognizing facial expressions. In this thesis, additional features have been added to the DCNN model such that it can correctly classify facial expressions in different lighting conditions and from different viewpoints as the model is trained to do so. Upon developing the DCNN model, an iOS app has been developed implementing this deep learning model as a byproduct and as a medium to use this model in clinical trials for children with autism as a medium of enhancing their communication abilities. The implementation of this proposed idea started with finding datasets containing images of faces with different expressions from different angles.

Further datasets were produced from the original dataset with images of different contrast and brightness with the help of image processing. The performance of the DCNN model was evaluated using these datasets. Once an optimal accuracy is achieved with good generalizability, an app suitable for iOS platform was developed for running both the DCNN model and image processing algorithms. The function of the app is to open the camera of the device, detect a face, classify the facial expression, and show the expression with an emoticon on the screen. As a product of this work, the app can be used by speech-language pathologies, teacher, care-takers, and parents as a technological tool when working with children with ASD. The design of the model and application is targeted to children with ASD to recognize and identify facial expressions in real-time to practice social skills during everyday social interaction.

I. INTRODUCTION

Strong and meaningful human interaction is necessary to convey feelings and communicate with another person. Human interaction is the process of exchanging information. Information can be conveyed as words, tone of voice, facial expressions and body language. Along with verbal communication, conveying communicative feelings can also be carried out by a person via nonverbal communication such as body language, facial expression, attitude, movement, etc. [1]. Words account for seven percent (7%) of the information communicated. Vocal tone accounts for fifty-five percent (55%) and body language accounts for thirty-eight percent (38%) [2]. The use and understanding of non-verbal communication grow naturally and many of us are unaware that we are using things like body language or facial expression or reading these cues to enhance words. Facial expressions play a significant role in interpersonal communication. In 1971, Ekman et al. [3] identified six facial expressions that are universal across all cultures - anger, disgust, fear, happiness, sadness and surprise.

As infants, nonverbal communication is learned from social-emotional communication, making the face rather than voice the dominant communication channel [4]. Children with stronger face-reading skills may achieve more popularity at school as discussed in [5], and they tend to perform better academically as shown in [6]. In addition, experiments hint that people who are better at identifying *fearful* expressions are kinder and more generous [7].

On the flip side, children who have more trouble identifying emotion in faces are more likely to have peer problems and learning difficulties as discussed in [8]. Preschoolers with poor face-reading skills for their age are more likely to have externalizing

behavioral problems, like hyperactivity [9], or, if they tend to be shy, such children are also more likely to suffer from anxiety [10].

For children with Autism Spectrum Disorder (ASD), face processing is a challenging task. It has been argued that the ability of children with ASD to understand facial expression is impaired and this difficulty may account for other problems that they demonstrate during social interactions [11]. In a study discussed in [12], accuracy and latency of emotion recognition were evaluated in children with autism spectrum disorder (ASD) and typically developing children while viewing videos of faces slowly transitioning from a neutral expression to one of six basic emotions (e.g., anger, disgust, fear, happiness, sadness, and surprise). Children with ASD were slower in emotion recognition and selectively made more errors in detecting anger. Several other studies including [13] and [14] showed impairment for children with ASD in classifying and understanding facial expressions compared to normal children of the same age.

Children with ASD improved their social skills by using a smartphone app – paired with Google Glass to help them understand the emotions conveyed in people's facial expressions, according to a pilot study by researchers at the Stanford University School of Medicine. The therapy, described in findings published online Aug. 2, 2018, in the Nature Partner Journals Digital Medicine [15], uses a Stanford-designed app that provides real-time cues about other people's facial expressions to a child wearing Google Glass. The biggest downside of this app is the price of the Google Glass which is currently around $1500. A cloud-based software company, Affectiva [16], gives powerful emotion metrics back when images, videos or audio files are fed to their service. Users upload digital content through the cloud-based user-interfaces and visualize the results

online or download the CSV or JSON files. However, the service also comes with a cost of $1.00 per video-minute and $0.50 per audio-minute. Other apps exist for facial expression recognition, but they are not designed to focus on the utilization of children with ASD. Moreover, the other developed app-solutions do not consider different lighting conditions and viewpoints of the camera.

In this thesis, a novel idea has been implemented of teaching young children with ASD to recognize human facial expressions in a friendly and practical environment. The main goal of this thesis was to develop a Deep Convolutional Neural Network (DCNN) which can recognize seven universal facial expressions. To make this deep learning model robust to complexities of viewpoints and lighting conditions, the model was trained with not only front-view facial images but also with images of faces from different orientation angles such as side view, top view, and bottom view. Also, the model is able to predict facial expressions in different lighting environments at different darker or lighter shades of contrast. As most children with ASD like to play with mobile devices such as smartphones and tablets, an iOS app was developed as a byproduct for this DCNN model to teach them to recognize facial expressions by using the device's camera. The app can run on any Apple device running iOS version 11 and beyond. Usage of the app is made simple and convenient for a child with ASD. The child with ASD points the camera toward a person, the app then automatically detects the face and classifies the facial expression of the person in real-time. The facial expression is then shown on the gadget's screen in the form of an emoticon. The goal of the app was to use these facial expressions as an emoticon to show children with ASD how the person, to

whom they are pointing the camera, is feeling and displaying emotional characteristics. Figure 1 shows a simple workflow diagram of the application.

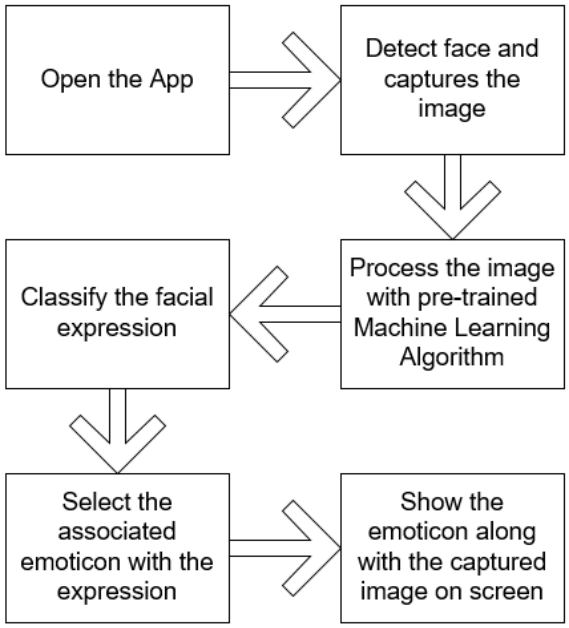

Figure 1. Workflow diagram of the iOS application

The facial expression recognition algorithm is designed by utilizing computer vision and deep neural network techniques. In recent years, deep learning has made noticeable progress in the task of classifying facial expressions. The famous Kaggle's "Challenges in Representation Learning: Facial Expression Recognition Challenge" [17] leaderboard showed that 71.16% test accuracy was achieved in classifying seven different classes of facial expression with the FER2013 dataset [18]. Convolutional Neural Networks (CNNs) are an alternative type of neural network that can be used to classify objects in an image. Therefore, by utilizing CNNs, the quality of image recognition and object classification had been progressing at a significant pace. Not only the result of powerful hardware, larger databases and bigger models, but also as a consequence of new

ideas, algorithms and improved network architectures has encouraged advancement in this field [19].

After performing the facial expression recognition phase, the algorithm is then installed in an app that runs in any compatible device running on iOS 11 or higher. For operation, the app opens the camera of the device and, if a face is detected by the camera, the facial expression is shown on the display in the form of an emoticon. The overall effort of the app is to be used by speech-language pathologists as a technological tool when working with children with ASD. In conjunction with tele-practice, the app can be used to teach children with ASD to recognize and identify facial expressions and receive therapy in real time to practice social skills during everyday social interactions.

II. BACKGROUND AND LITERATURE REVIEW

Background

In this section, how the thesis idea originated and presented itself is discussed. Why an engineering solution is much needed for facial expression recognition for children with ASD is also discussed. Finally, a preview of the completed research solution is presented.

When one communicates with a child, lots of communication take place without any words. This type of communication is called nonverbal communication. Nonverbal communication includes facial expressions, body language, body contact, eye contact, personal space and tone of voice. Positive nonverbal communication can improve the relationship with the child and boost emotional connections with the other person. Most children love being hugged and kissed, for example. This warm and caring body language sends the nonverbal message that one wants to be close to their child. Negative nonverbal communication – for example, a grumpy tone of voice or a frown – presents itself as a message of disagreement, indifference, or other negative characteristic. Children can feel rejected or let down if this happens consistently.

Among the nonverbal communications, facial expression is one of the most difficult to master because of the subtle meanings and micro-expressions. With our 80 facial muscles, we can create more than 7,000 facial expressions. Universally, there are a total of six facial expressions that are found across all cultures – Angry, Disgust, Happy, Sad, Surprise and Fear. All the work that has been done on facial expression recognition use these six universal expressions. Some work also includes another category – neutral.

Along with the neutral expression, these seven categories of facial expressions are the baseline for any facial expression recognition related work.

For children to correctly classify facial expressions is a difficult task. It is even more difficult for children with ASD. Children with ASD have major difficulties in recognizing and responding to emotional and mental states in others' facial expressions [20]. Babies who are later diagnosed with ASD can recognize feelings in a similar way to typically developing babies. However, these children are slower to develop emotional responses than typically developing children. By the time children reach 5-7 years old, they can recognize happy and sad, but they have a harder time with subtle expressions of fear and anger. By adolescence, teenagers with ASD still experience difficulty recognizing fear, anger, surprise, and disgust when compared to typically developing teenagers. As adults, they continue to have trouble recognizing some emotions [21].

In this thesis, a deep learning approach is presented to close the gap in how to help children with ASD accomplish this difficult task of classifying facial expressions correctly. The approach was to develop a DCNN (Deep Convolutional Neural Network) with the help of computer vision and image processing to perform the facial expression computation. After completing this main task, the iOS app is developed and tested with emotion metrics of different persons to have a better understanding of the performance of the model. The app, supported by any iOS device, can open the camera of that device, correctly classify the facial expression when the camera detects a face and show it on the screen with an emoticon. This app is able to perform under any lighting environment conditions and capture the face from many angles.

Literature Review

This thesis deals with performance of children with ASD in recognizing human facial expressions and the technical development of a model with DCNN to classify different facial expressions. The thesis idea has been achieved by studying facial expression recognition studies for children with ASD and also studies of deep learning models for classifying facial expressions. The novel idea of this thesis was inspired by merging these two sections from which different aspects and motivations of the thesis can be better understood.

Facial Expression Recognition Studies for Children with Autism

Many studies have been done about the impairments of children with ASD typically face while processing human facial expressions and different methods have been proposed to teach them how to learn different facial expressions. The authors in [11] assessed the influence of motion on facial expression recognition in young children with ASD. They were compared on their ability to match videotaped “still,” “dynamic,” and “strobe” emotional and non-emotional facial expressions with photographs. Compared to previous studies showing a lower performance in children with ASD than in control children when presented with static faces, the author’s data suggest that slow dynamic presentations facilitate facial expression recognition by children with ASD. Another paper [22] presented a detailed study of the implementation of serial and parallel implementation of Principal Component Analysis (PCA) to identify the most feasible method for realization of a portable emotion detector for children with ASD. They achieved 82.3% detection accuracy implementing the architecture on Field Programmable Gate Array (FPGA). Different experiments, surveys, and comparisons

have been made about the facial expression recognition difficulties and impairments of children with ASD, including [23], [24]. Analysis of the results suggests that impaired face recognition does not result from impaired attention or discrimination and indicates that most people with ASD have severe expression recognition deficits.

CNN Models for Facial Expression Recognition

In the past few years, deep learning in DCNN has made considerable progress in the task of recognizing human facial expressions. In the work found in [25], the authors reviewed the state of the art in image-based facial expression recognition using CNNs and highlighted algorithmic differences and their performance impact. They demonstrated that overcoming one of these bottlenecks – the comparatively basic architectures of the CNNs utilized in this field – leads to a substantial performance increase in identifying expressions. By forming an ensemble of modern deep CNNs, the authors reached a FER2013 test accuracy of 75.2%, outperforming previous works without requiring auxiliary training data or face registration.

As part of another project such in [26], the authors applied various deep learning CNN methods to identify the key seven human emotions: angry, disgust, fear, happy, sad, surprise and neutral. Using the FER2013 dataset, they leveraged ensemble and transfer learning techniques to achieve optimal results. The authors achieved an accuracy of 67.2% using ensemble learning and 78.3% with transfer learning. Solid results are given by the winner of the Kaggle Facial Expression Recognition Challenge, who had an accuracy of 71.2%. Those who ranked in the top 10 of the same competition only achieved accuracies averaging around 60%. Using visual saliency and deep learning on Compound Facial Expressions of Emotion Dataset (CFEE) and Radbound Faces

Database (RaFD) datasets, the authors from [27] achieved test accuracies of respectively 74.79% and 95.71%. In the work described in [28], the authors proposed methods of refining individual models and combining different CNNs which achieved significant end results, with high test accuracy of 87.4% from their Linear SVM approach. Using a combination of algorithms for face detection, feature extraction, and classification, in [29], authors achieved average expression recognition accuracy of 97.71% and 95.72% respectively for the Japanese Female Facial Expression (JAFFE) and the Extended Cohn-Kanada (CK+) datasets.

All these papers discuss different methods and techniques for facial expression recognition. For this thesis, a DCNN is developed and experimented with using different techniques to obtain a better generalized performance of recognizing facial expressions. Then with this model an iOS app is developed which can open the camera of the device in which it is installed and recognize facial expressions correctly. This thesis combines the two main approaches of facial expression recognition with CNN and using it for the study of facial expression recognition with children with ASD.

Thesis Contributions to the Advancements of Facial Expression Recognition

All the research papers discussed in the previous sections deal with front facial images for facial expression recognition. In this thesis, the developed deep learning CNN model can classify facial expression from seven different viewpoints (front, 45-degree left, 45-degree right, 90-degree left, 90-degree right, 45-degree up, 45-degree down). It can also classify facial expressions in different lighting contrasts including dark and bright conditions. This has been achieved through preprocessing techniques before feeding the images to the model. The iOS app is also a new addition to the research

which can show the performance of the model in real time. The app has been tested in different conditions to effectively evaluate performance of the CNN model.

III. DEEP LEARNING

Deep learning is a subfield of Machine Learning and Machine Learning is a subfield of Artificial Intelligence (AI). This relationship can be best described by a graphical representation shown in Figure 2.

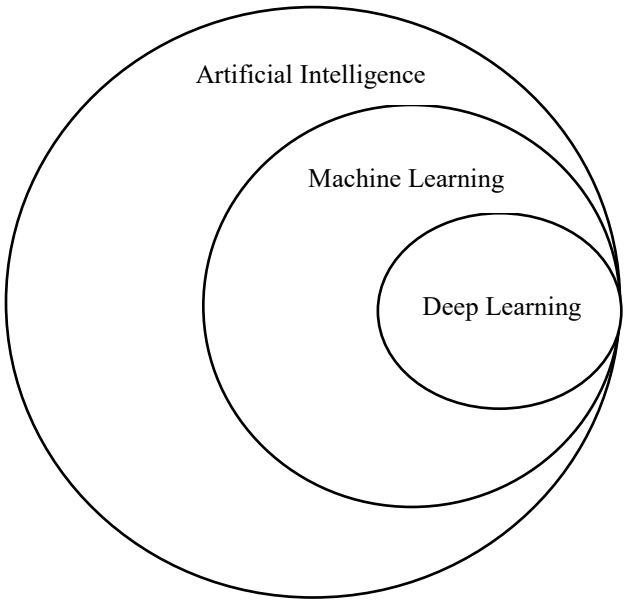

Figure 2. A Venn diagram describing relationship between Deep Learning, Machine Learning and Artificial Intelligence

For a computer, translating and recognizing the contents of an image has proven to be a very difficult task. The same task can be performed by humans very easily and with minimal effort. The primary goal of AI is to provide a set of algorithms that can be used to solve problems that humans can perform instinctively and automatically but are otherwise very challenging for computers [30].

Machine Learning can be defined as a large sub-field of AI dealing with the field of study that gives computers the ability to learn without being explicitly programmed [31]. This means a single program, once developed, will be able to learn by itself how to do some intelligent activities outside the notion of programming. Therefore, it can be said

that Machine Learning is an approach to achieve AI. This is exactly the form humans learn as well. The human brain automatically learns features of an object and when seen later, that object can be recognized by a human through memory and experience.

The human brain is undoubtably one of the best machines we know for learning and solving problems. The neuron is said to be the main computational element of the human brain. Neurons communicate with each other through the basic working unit of the brain, a specialized cell designed to transmit information to other nerve cells, muscle, or gland cells. The complex connected network of neurons forms the basis of all the decisions made based on the various information gathered as shown in Figure 3. Artificial Neural Networks (ANNs) are a simplified representation of the human brain neural system and work in the same way as a human brain does. A simple ANN has an input layer, a hidden layer, and an output layer as shown in Figure 4.

Within the domain of neural networks, there is an area called Deep Learning (DL) in which neural networks have more than three layers, i.e. more than one hidden layer. These neural networks used in DL are called Deep Neural Networks (DNNs) [32]. In the remainder of this chapter, details will be reviewed about when a neural network can be called “deep”, concept of “hierarchical learning” and when to use deep learning networks for classification problems.

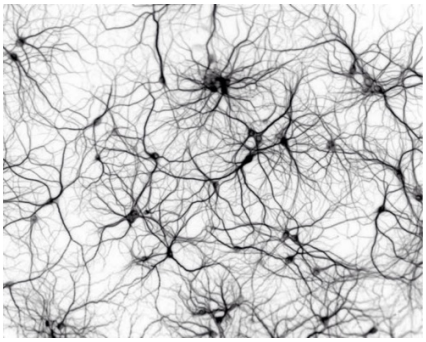

Figure 3. Complex connected network of neurons of a human brain

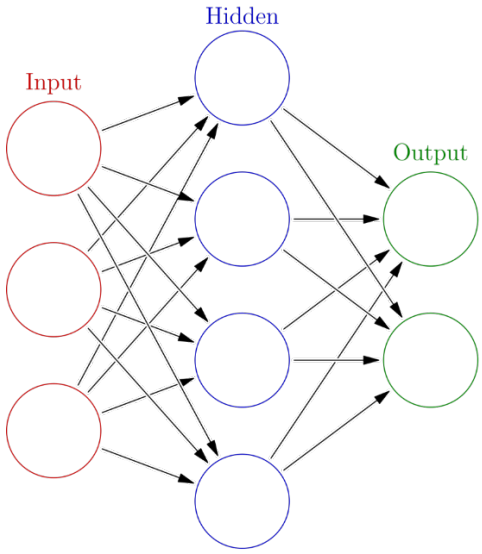

Figure 4. An Artificial Neural Network. Here, each circular node is an artificial neuron an arrow is a connection between nodes

Hierarchical Learning

Machine Learning algorithms (generally) can be divided into four categories – supervised, unsupervised, semi-supervised learning, and reinforcement learning. The Machine Learning algorithm developed in this thesis falls into the category of supervised learning. Hence, supervised learning will be discussed here in a nutshell.

Supervised learning is the Machine Learning task of learning a function that maps an input to an output based on example input-output pairs [33]. It infers a function from labeled training data consisting of a set of training example [34]. In supervised learning, a Machine Learning algorithm is given labeled training data consisting of both a set of inputs and target outputs. The algorithm then tries to learn patterns that can be used to automatically map input data points to their correct target output. For example, given a dataset of labeled pictures of human facial expressions, the algorithm can learn to predict and classify unseen pictures of human facial expression to its corresponding classes (labels). This is exactly what has been done in this thesis. Another example can be having

a bunch of molecules and information about which are drugs and training a model to answer whether a new molecule is also a drug. Figure 5 shows another example of supervised learning where a deep neural network is trained to classify beagles in images.

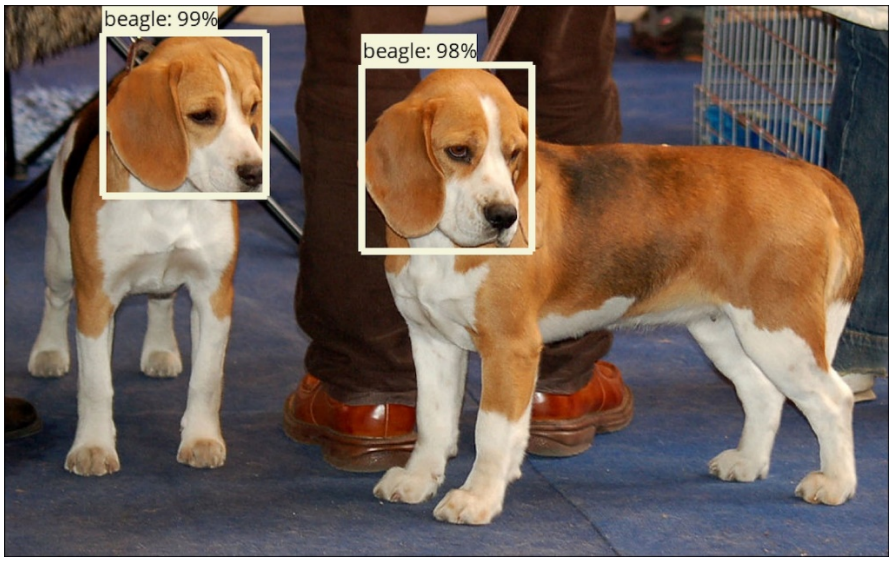

Figure 5. Result of a supervised learning algorithm using DNN classifying beagles in images

Previously, hand-engineered features were used to quantify the contents of an image. Raw pixel intensities were rarely used as inputs to Machine Learning models, which is now common with DL [30]. First, the feature extraction was performed which is known as the process of taking an input image. The features extracted were quantified according to some algorithm. This type of algorithm was called a feature extractor or image descriptor. Finally, the feature extraction algorithm returned a vector, list of numbers, that aimed to quantify the contents of an image. These featured vectors were then used as the input of a Machine Learning algorithm.

DL does not use any hand-defined algorithm to extract features from an image, instead these features are learned automatically by the DL model itself through the training process. As previously discussed, a DL model has more than one hidden layer. A simple feedforward neural network with two hidden layers is shown in Figure 6. The

learning process of a DL model can be described in terms of hierarchy. A series of hidden layers are used to extract features from an image. From the input layer to the output layer, these hidden layers build upon each other in a hierarchical way. The lower level of the hidden layers usually learns to detect edges, shapes, regions, and combine those to form contours and corners. In the next layer, these contours and corners in turn form abstract object parts of the image. This learning process is completed by filters which are learnt automatically by the model. Finally, the output layer, a classification algorithm, is used to categorize the image and obtain the output class label.

Each layer in the DL network utilizes the output of the previous layers to construct more conceptual features of an image and these features and layers are learned automatically. For this reason, the learning process is called hierarchical learning. A comparison between classic image classification algorithms using Machine Learning and current DL algorithm is shown in Figure 7.

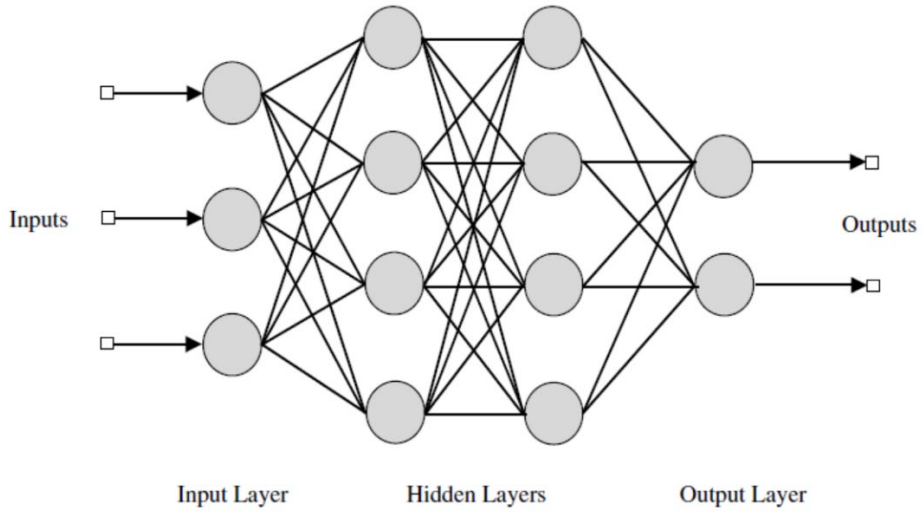

Figure 6. A feedforward neural network with two hidden layers of a DL model

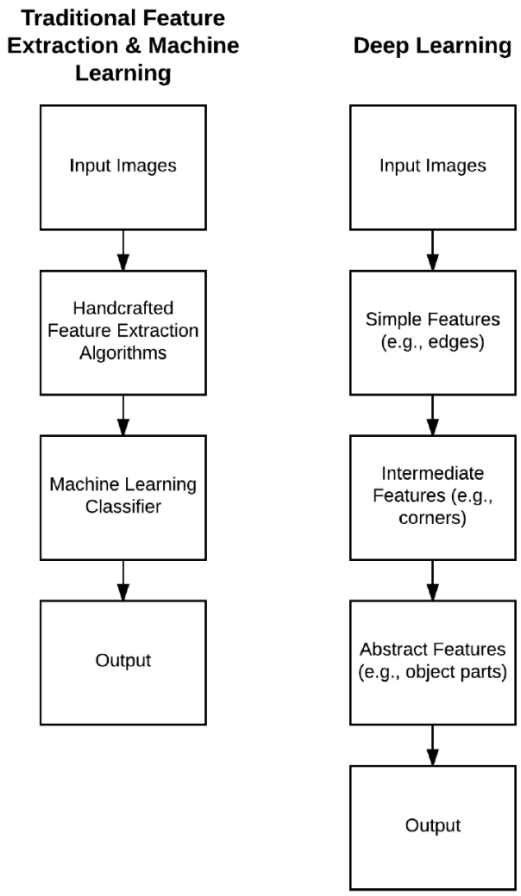

Figure 7. Comparisons between traditional Machine Learning Algorithm and DL algorithm [30]

Using Deep Learning

The "deep" term depends on the type and depth of the network. The CNN, Recurrent Neural Network, Long-Short Term Memory (LSTM) network are considered to be DNNs for their specialized type. Normally, the depth of a neural network is greater than two, and it is considered a DNN. Here, the depth refers to the number of hidden layers in the network. If depth is greater than ten, the network can be called very deep.

As the depth of a DNN increases, classification accuracy also increases opposed to the case for traditional Machine Learning algorithms. This behavior can be better show by a plot inspired by Andrew Ng's 2015 talk, *what data scientists should know about DL* [35] shown in Figure 8. From the plot, it can be seen that when amount of training data

increases, accuracy also increases for DNN whereas it comes to a standstill for traditional Machine Learning algorithms. This is the reason to associate large datasets with DL. For image classification, it is said that if the dataset contains more than 1000 images, then DL will surely achieve better results compared to other Machine Learning algorithms such as logistic regression, SVMs, decision trees, etc.

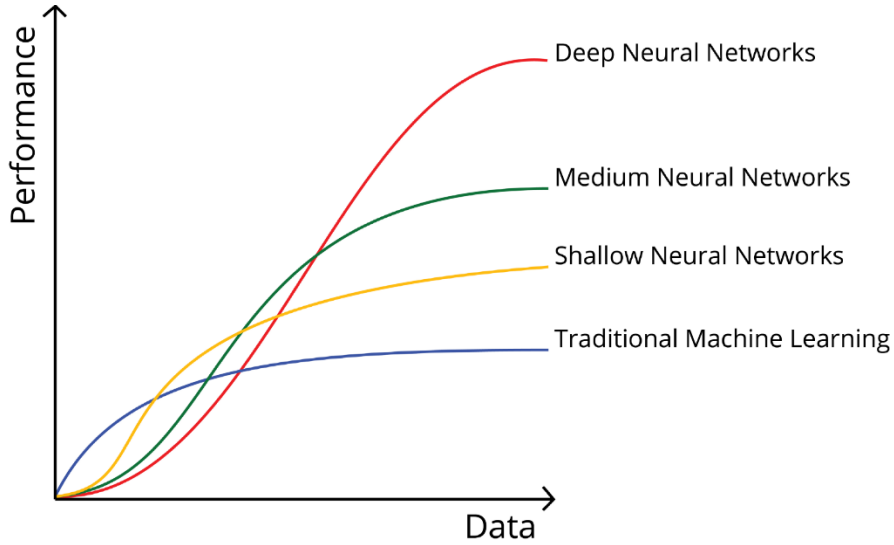

Figure 8. Relationship between performance and training data for different neural networks

In this chapter, fundamentals of DL are discussed. It belongs to the family of Artificial Neural Networks (ANNs). Hierarchical learning of DL networks and usage of DL are also discussed compared to traditional Machine Learning algorithms.

IV. CONVOLUTIONAL NEURAL NETWORK

Computer vision is a field of Artificial Intelligence which enables machines to see and perceive objects as humans do. The advancement and popularity of computer vision with Deep Learning is driven upon the Convolutional Neural Network (CNN) algorithm. A CNN is a Deep Learning algorithm which can take in input images, train itself by learning filters to distinguish individual features between images and be able to differentiate one image from the other [36]. Figure 9 shows a CNN that can distinguish and recognize different type of automobiles.

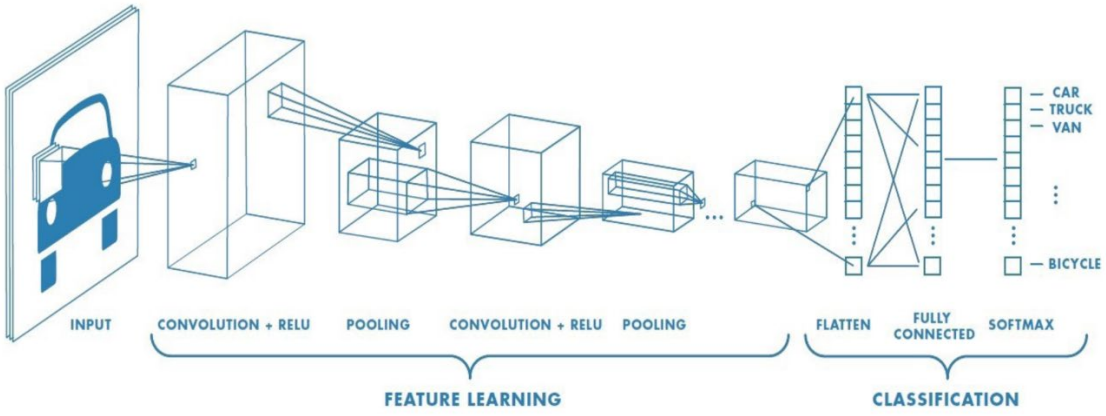

Figure 9. A CNN to classify different type of vehicles [36]

Traditional feed-forward Machine Learning algorithms use Fully- Connected layers with each neuron of one layer connected to every output neuron of the next layer. CNNs use a Convolution layer to begin the network with and use Fully-Connected networking at the end of the architecture. Each layer of a CNN applies a different set of filters, typically hundreds or thousands, and combines the results by feeding the output into the next layer in the network [30]. During the training of a model, the CNN automatically learns the values for these filters. “In the context of image classification, a CNN may learn to:

- Detect edges from raw pixel data in the first layer.
- Use these edges to detect shapes (i.e., “blobs”) in the second layer.
- Use these shapes to detect higher-level features such as facial structures, parts of a car, etc. in the highest layers of the network.” [30]

Why CNN for Image Recognition

Traditional Machine Learning algorithms were used for image classification before CNN became successful for computer vision applications. However, there was no direct way to input images directly to the algorithm. First, a separate feature extraction algorithm was used to extract features from an image. Then these features were fed into a classification algorithm like Support Vector Machine (SVM), k-Nearest Neighbor, Logistic Regression, and others. Some algorithms also used the pixel level values of images as a feature vector too. In the case of SVM, the training of the model with 2,304 features would require each feature to be each pixel-value for a 48×48 image.

A preferred component of a CNN over other algorithms for image recognition is that no feature development is required for the network. This provides the CNN algorithm with feature extraction capacity in its design. The CNN learns those extracted features through the training process of the model. Thus, CNNs can be thought of an automatic feature extractor from images.

This concept of CNN was first presented by Yann LeCun [37] in 1998, in where the author reviewed various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. In this paper, the author used a single Convolution layer. It was later popularized by AlexNet [38] in 2012, which

used multiple Convolution layers to achieve high performance on the ImageNet dataset, which is the largest dataset used for object detection.

CNN Implementation for Facial Expression Recognition

For the development of the facial recognition model, the purpose is to classify seven different facial expressions from input images. For image classification, the CNN is the best option for computer vision applications, and it has been used as the Deep Learning algorithm for the development of the model. Without the utilization of CNN, filters for various image processing operations such as smoothing, sharpening, and edge detection will require further development increasing the complexity of the process.

CNNs are able to learn filters that can detect edges and blob-like structures in lower-level layers of the network by applying convolutional filters, nonlinear activation functions, pooling, and backpropagation. Also, CNNs are capable of using the edges and structures as "building blocks" that eventually detect high-level objects, similar to facial expressions, in the deeper layers of the network. The process of using the lower-level layers to learn high-level features is achieved by stacking specific set of layers in a purposeful manner. In the next section, these different types of layers will be discussed which are also known as the building blocks of a CNN.

Building Blocks of a CNN

A feedforward neural network does not scale well as the image size increases and also leaves much accuracy to be desired as every layer in this network is a Fully-Connected network. On the other hand, a CNN define a network architecture in a more sensible way. Unlike a standard neural network, the layers of a CNN are arranged in a 3D volume in width, height, and depth where depth refers to the third dimension of the

volume, such as the number of channels in an image or the number of filters in a layer [30]. The volume concept can be explained with a feedforward neural network is taking a color image of size 24×24 pixels as its input. The total inputs to the network will be 24×24×3 = 1,728, in where the value of 3 denotes the RGB channels for a color image. This amount is not that much for a neural network, but if images of 500×500 pixels are used, then the total inputs to the network will be 500×500×3 = 4,50,000. These values do not include the other connecting lines in the network's hidden layers and output layer. Therefore, these parameters can quickly add up and a poor performance is evident.

On the other hand, for a CNN the input volume will have dimensions 24×24×3 (width, height, and depth, respectively). Neurons in subsequent layers will only be connected to a small region of the layer before it (rather than the fully-connected structure of a standard neural network) – this is called local connectivity which enables to save a huge number of parameters in the network. Finally, the output layer will be a 1×1×N volume which represents the image distilled into a single vector of class scores. In the case of this thesis which classifies seven different emotions, N = 7, yielding a 1×1×7 volume.

There are different types of layers in a CNN. The most common layers of a CNN are as follows.

- Convolutional Layer
- Activation Layer
- Pooling Layer
- Fully-Connected Layer
- Batch Normalization Layer

- Dropout Layer

A CNN architecture is nothing but a stack of these layers in a specific manner. Often, A CNN architecture can be described as a collection of input layer, Convolution layer, Activation layer, and Fully-Connected layer respectively. Here a simple CNN is defined that accepts an input, applies a convolution layer, then an activation layer, then a Fully-Connected layer. Convolutional, Fully-Connected and Batch Normalization layers contain parameters that are learned during the training process. Among these layers, Convolutional, Fully-Connected, Activation and Pooling layers are the most important ones to define the actual network architecture.

Convolutional Layers

The first layer of a CNN is a Convolutional layer as the core building block of a CNN. It consists of a set of learnable filters or kernels. These kernels are usually of small size such as 3×3 or other squared-dimension sizes. These filters slide across the input image. A filter/kernel is also an array of numbers which are called weights or parameters. A very important note is that the depth of this filter has to be the same as the depth of the input. In Figure 10, it can be seen in the top left corner that as the filter is sliding, or **convolving**, around the input image, it is multiplying the values in the filter with the original pixel values of the image (aka computing **element wise multiplications**). These multiplications are all summed up and a single number is the final output of the convolution operation. The number is a representative of when the filter is at the top left of the image. The process is repeated for every location on the input volume. The next step would be moving the filter to the right by 1 or 2 unit, known as a stride, then right again by 1 or 2 units, and continues.

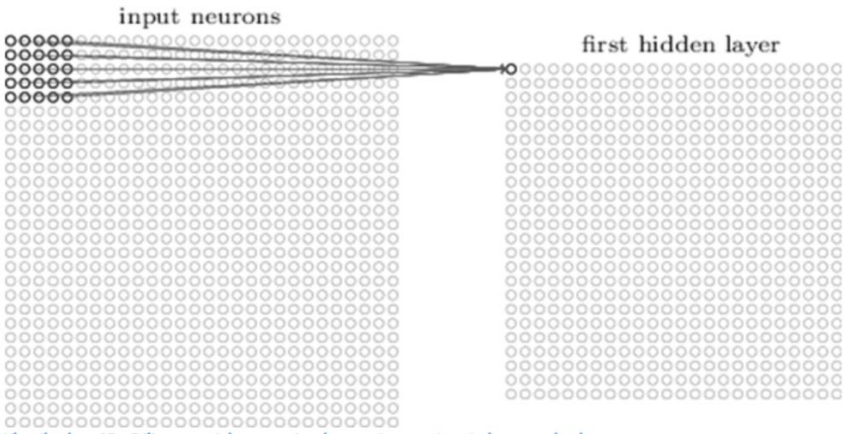

Figure 10. Convolution operation of a CNN [39]

After applying all *N*-filters to the input volume, there are *N*, 2-dimensional activation maps. These *N* activation maps are then stacked together along the depth dimension of our array to form the final output volume as shown in Figure 11.

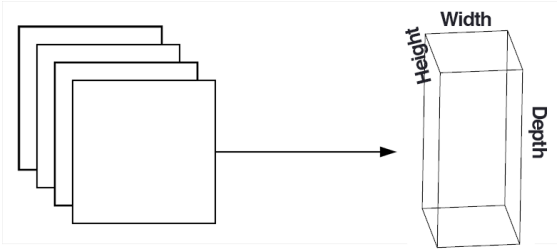

Figure 11. After obtaining the *N* activation maps, they are stacked together to form the input to the next layer [30]

"Every entry in the output volume is thus an output of a neuron that 'looks' at only a small region of the input. In this manner, the network 'learns' filters that activate when they see a specific type of feature at a given spatial location in the input volume. In lower layers of the network, filters may activate when they see edge-like or corner-like regions. Then, in the deeper layers of the network, filters may activate in the presence of high-level features, such as parts of the face, the paw of a dog, the hood of a car, etc." [30]

Activation Layers

After each Convolutional layer, an Activation layer is placed in the CNN. The Activation layers are nonlinear functions such as Sigmoid, Rectified Linear Unit (ReLU) [40], Exponential Linear Unit (ELU) [41], and others. For the model development, ReLU and ELU were used for the design of the facial expression recognition model. Plots of different activation functions with their corresponding mathematical function are shown in Figure 12.

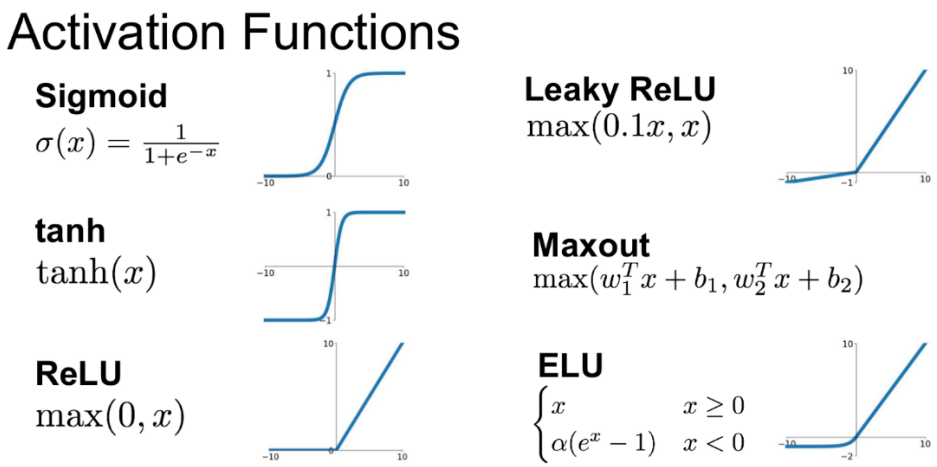

Figure 12. Plots of different activation functions

"An activation layer accepts an input volume of size $W_{input} \times H_{input} \times D_{input}$ and then applies the given activation function as shown in Figure 13. The activation function is applied in an element-wise manner, and the output of an activation layer is always the same as the input dimension, $W_{input} = W_{output}$, $H_{input} = H_{output}$, $D_{input} = D_{output}$." [30]

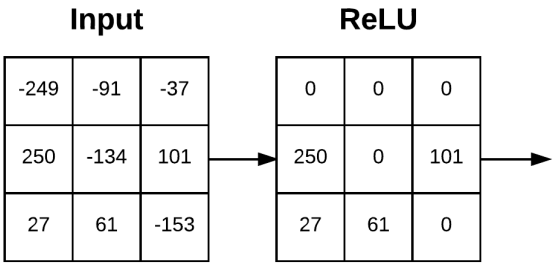

Figure 13. An example of an input volume going through a ReLU activation, max (0, x) [30]

Pooling Layers

Pooling layers are primarily used to reduce the spatial size of an input volume and are placed in-between consecutive Convolutional layers. Pooling allows to reduce the number of parameters and computation in the network and helps to control overfitting. Pooling layers operate on each of the depth slices of an input independently using either the max or average function. Max Pooling is typically done in the middle of the CNN architecture to reduce spatial size, whereas average Pooling is normally used as the final layer of the network, like GoogLeNet, SqueezeNet, ResNet where it is wished to avoid using FC layers entirely. A pooling operation is shown in Figure 14. As seen from the figure, decrease in input size depends on the size of pooling layer and stride.

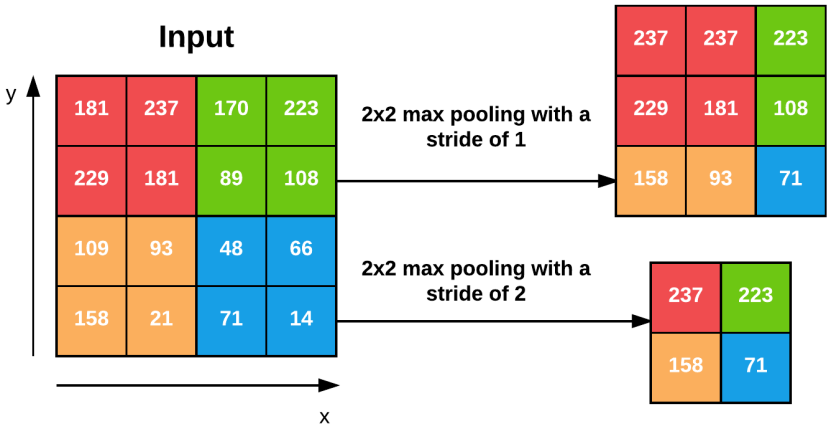

Figure 14. An example of a Pooling operation with different strides [42]

Fully-Connected Layers

As the name suggests, in a Fully-Connected layer, neurons are fully connected to all activations in the previous layer. Fully-Connected layers are always placed at the end of the network. This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector; where N is the number of classes that the program must realize. The development of the model will use classes denoted by numerical order representing Angry = 0, Disgust = 1, Happy = 2, Sad = 3, Fear = 4, Surprise = 5, Neutral = 6. The output vector can be represented as a vector of probability values, such as [0 .1 .1 .75 0 0 .05], representing a 10% probability that the image detects a Disgust expression, a 10% probability that the image detects Happy expression, a 75% probability that the image contains Sad expression, and a 5% probability that the image displays a Neutral expression.

Batch Normalization

Batch Normalization layers [43] are used to normalize the activations of a given input volume before passing it into the next layer in the network. If x is considered as the mini-batch of activations, then the normalized $\hat{\boldsymbol{x}}$ can be computed using the following equation,

$$\hat{\boldsymbol{x}} = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}} \tag{1}$$

Here, μ_β and σ_β^2 are respectively mean and variance over each mini-batch of training images, β. The error ε is set equal to a small positive value in the range of 1e-7 to avoid dividing by zero. Applying (1) implies that the activations leaving a Batch Normalization layer will have approximately zero mean and unit variance (i.e., zero-

centered). At testing time, the mini-batch μ_β and σ_β^2 are replaced with running averages of μ_β and σ_β^2 computed during the training process. This ensures that images through the network can be passed and obtain an accurate prediction without being biased by the μ_β and σ_β^2 from the final mini-batch passed through the network during the training phase [30].

Batch Normalization has been shown to be extremely useful for the following effects in the network.

- It reduces the number of epochs a NN takes to train itself.
- It stabilizes training by allowing for a wide variety of learning rate and regularization techniques.
- It provides stable loss curve as it helps to minimize loss.

Dropout

Dropout is actually a form of regularization that aims to help prevent overfitting by increasing the testing accuracy, and possibly at the expense of training accuracy. For each mini-batch in the training set, Dropout layers, with probability *p*, randomly disconnect inputs from the preceding layer to the next layer in the network architecture [30]. Figure 15 shows dropout operation with a value of 0.5 where 50% connections from previous layer are dropped.

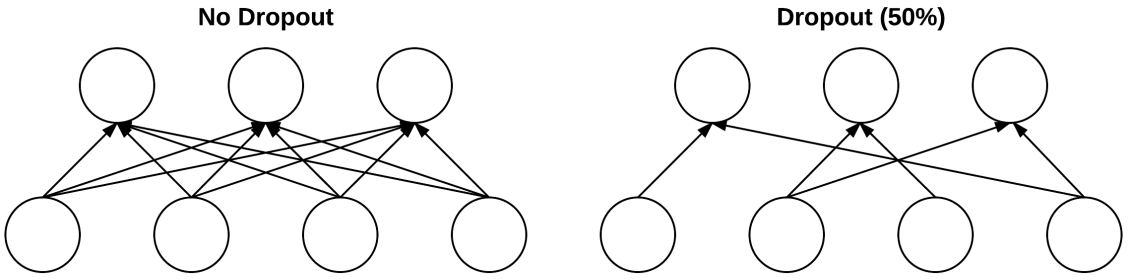

Figure 15. Effect of no Dropout (left) and Dropout with a value of 0.5 (right)

The Dropout method helps the model to generalize by reducing overfitting because dropping some connections to the previous layer ensures there are multiple nodes instead of one when presented with a given pattern i.e. filter.

In this chapter, the role of a CNN in image recognition is discussed, and why it is the best path for facial expression recognition. Finally, the building blocks of a CNN are discussed in detail.

V. METHODOLOGY FOR DETECTING FACIAL EXPRESSIONS

In this chapter, the four main components of a CNN are discussed pertaining of its datasets, loss function, model Architecture, and optimization approach. These components will be discussed as in how these components have a significant role in the outcome of facial expression recognition. Training and testing the model along with some preprocessing, hyperparameter tuning and regularization will also be discussed in detail. The work flow of the methodology starts with collecting and labeling datasets followed by specifying the CNN model architecture. Next step is to specify the loss function then apply preprocessing techniques. To better understand the methodology of each layer of CNN, mathematical explanation of how each layer of the model works is also added. Finally, training of the model with specified optimization method and tuning of the hyperparameters are discussed.

Datasets

The problem of finding a facial expression recognition model is a supervised learning problem in which the CNN model is trained with examples or data so that it can teach itself to perform the classification when unobserved data is given. For this thesis, facial expressions were needed to be classified by a CNN model and for that it was essential to have datasets which contain images of seven different facial expressions. Two datasets were used for training the facial expression recognition CNN model. The dataset is the first component in training a neural network – the data itself along with the problem we are trying to solve define our end goals.

The first dataset that was used in this thesis is the dataset used in the Kaggle's "Challenges in Representation Learning: Facial Expression Recognition Challenge" [17].

It is known as the **FER2013** dataset and can be found in [18]. The dataset consists of 48x48 pixel grayscale images of faces with seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 unique examples of the seven categories. The public test set used for the leaderboard consists of 3,589 examples. The final test set, which was used to determine the winner of the competition, consists of another 3,589 examples. A preview of images of the FER2013 dataset is shown in Figure 16.

Figure 16. Image contents of FER2013 dataset showing different emotions

The Karolinska Directed Emotional Faces (**KDEF)** dataset was used to address the problem of recognizing facial expressions from different viewpoints, different angles. The KDEF dataset provides images to address the different angles for training and testing of the model. The KDEF dataset contains 4,900 pictures of human facial expressions with indicated emotions. The dataset contains 70 individuals, each displaying seven different emotional expressions, and photographed twice from five different angles. An example of the KDEF dataset can be better seen in Figure 17.

Figure 17. Image contents of KDEF dataset showing image of surprise expression from five different angles

It was also considered to develop the DCNN model in such a way that will enable children with ASD to use the app while lying on bed and for this reason upward and downward viewpoints were also necessary. Unfortunately, no dataset was found that have these types of images containing the seven different expressions. For this reason, images showing different expressions from upward and downward angles were collected from different sources, and they were stacked together to form a dataset of nearly 500 images. The dataset was used on top of the FER2013 and KDEF datasets while training the model. However, only the happy and sad expressions were found in these images. A sample of images from this dataset is shown in Figure 18. In proof of usability, having top and bottom viewpoints is not that much necessary as the child can easily move the gadget and tilt it in order to recognize a facial expression. Also, as the FER2013 dataset contains nearly 30,000 images and they vary slightly in angles, the model was trained to deal with some variations in upward and downward direction. Therefore, the model is capable of handling upward and downward viewpoints variations through tilting the device towards the face and is able to recognize the expressions.

Figure 18. Images captured from upward direction showing happy expression

The other aspect to overcome in the model design is to recognize the face and associated expressions in darker and brighter lighting conditions. For this component, the datasets were modified by changing their brightness using OpenCV in Python and train the model including with those datasets. An example of the modified datasets with

different lighting contrasts are shown in Figure 19. All of these datasets were included to make the dataset more robust and flexible when dealing with different lighting conditions.

Figure 19. Original picture from the FER2013 dataset (middle), darker versions of the picture are on the left and brighter versions of the picture are on the right. Each image represents a dataset containing images of same brightness

DCNN Model Architecture

The deep convolutional neural network (DCNN) used for this model design is inspired by the family of VGGNet networks [44], specially the VGG16 architecture design. The design idea is also heavily inspired by [30] in terms of layers formation and their output size. As the model was first trained with the FER2013 dataset, the output size of different layers of the model are consistent with the FER2013 dataset and those were modified later when other datasets came into play. The initial DCNN model architecture is shown below in Figure 20. The initial model takes an input image of 48 × 48 pixel and processes it through several Convolution, Max-Pooling, and Fully- Connected layer providing the final output of the either seven classes: Anger, Disgust, Fear, Happiness, Sadness, Surprise, and Neutral. The initial DCNN model is also tabulated in Table 1.

All the convolution layers have a filter size of 3×3 and all the Pooling layers have a pool size of 2×2. These are part of hyperparameters and were tuned into other values during training process to find the optimized model. Neither in the diagram in Figure 20 nor in the Table 1, the Activation, Batch Normalization or the Dropout layers are included. Every Convolution layer is followed by an Activation layer, and the Batch

Normalization is also included after every activation layer to gain higher accuracy. The Pooling and Dropout layer values are also hyperparameters and were tuned throughout the whole training process. The Dropout layer is normally used after Fully-Connected layers, but after performing run trials, it has been seen that using the Dropout after each Pooling layer helps reducing overfitting of the model. This idea was also taken from [30]. Therefore, the convolution-pooling group consists of Convolution, Activation, Batch Normalization, Pooling, and Dropout layers respectively. The Fully-Connected layers consists of similar architecture such as Fully-Connected, Activation, Batch Normalization, Pooling, and Dropout layers respectively. A Softmax classifier is used to categorize the image into the seven distinct facial expressions categories. For all the datasets, the same architecture was used by only updating the output value, hyperparameters values, and finally coming up with the final model. The initial model is shown to specify the starting point of optimization and understanding all the consecutive training process of the evolved model. The final model architecture with output values used for the app will be discussed during the training and optimization process section.

Table 1. DCNN Architecture

Type of Layer	***Output Size***	***Filter / Pool Size***
Input Layer	48 × 48 × 1	3 × 3
Convolution	48 × 48 × 32	3 × 3
Convolution	48 × 48 × 32	3 × 3
Max-Pooling	24 × 24 × 32	2 × 2
Convolution	24 × 24 × 64	3 × 3
Convolution	24 × 24 × 64	3 × 3
Max-Pooling	12 × 12 × 64	2 × 2
Convolution	12 × 12 × 128	3 × 3
Convolution	12 × 12 × 128	3 × 3
Convolution	12 × 12 × 128	3 × 3
Max-Pooling	6 × 6 × 128	2 × 2
Fully Connected	64	
Fully Connected	64	
Fully Connected	7	
Softmax	7	

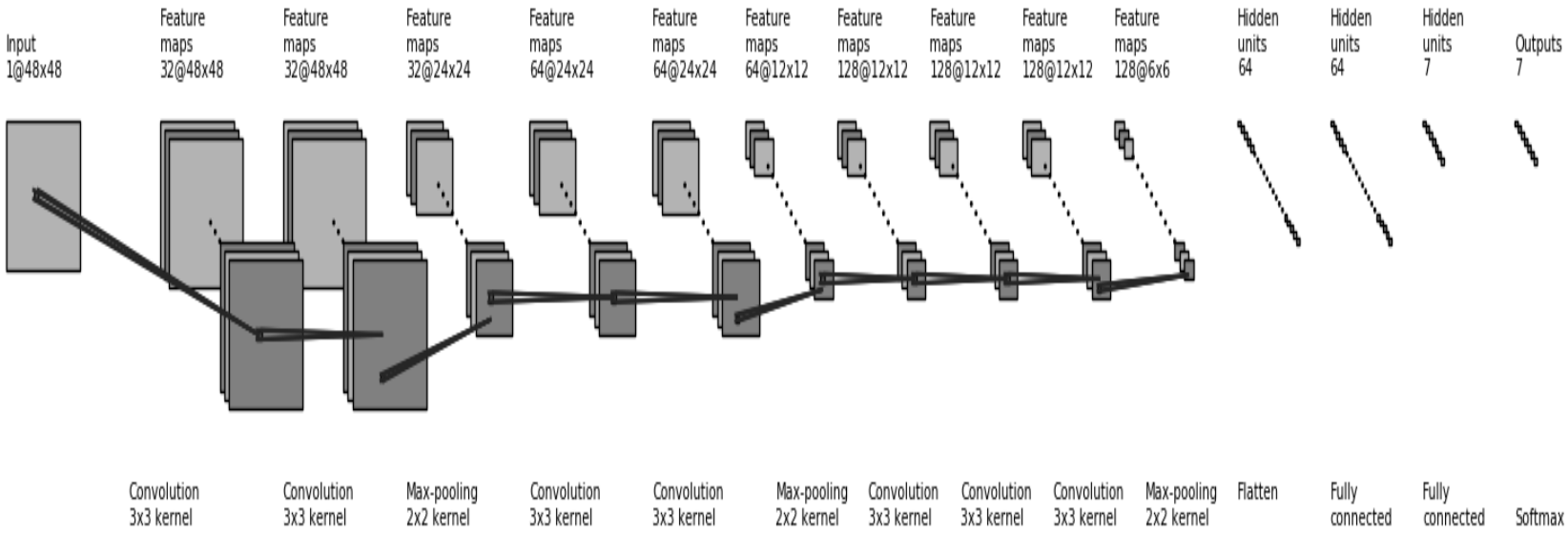

Figure 20. Architecture of the implemented DCNN (Figure is generated by adapting the code from https://github.com/gwding/draw_convnet)

The Loss Function

A loss function is needed to be defined for a neural network to find the optimal solution through gradient descent. For nearly all image classification problem, and specially for a CNN, Softmax classifier is the most popular loss function which is a categorical cross-entropy loss function. The Softmax classifier is a generalization of the binary form of Logistic Regression. The mapping function f is defined by the set of input data x_i and weight matrix W such as,

$$f(x_i, W) = Wx_i \tag{2}$$

This is also known as the scoring function and denoted by S. The function can be used to derive normalized probability for each class label, and the loss function for a single data point, L_i can be found as,

$$L_i = -\log\left(\frac{e^{S_{y_i}}}{\sum_j e^{S_j}}\right) \tag{3}$$

Therefore, the loss function for the whole dataset, L can be found by taking the average:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i \tag{4}$$

For facial expression recognition DCNN model, other loss functions were also considered and tested during the training process of the model. But it was found that Softmax classifier performed better which is usually the case for CNNs. It is also useful in the sense that it provides percentage probability as an output for each class label which makes it easier to extract the model output results effectively.

Preprocessing

As a preprocessing tool, histogram equalization has been used in this thesis. It is done to all the images in a dataset before feeding it into the CNN model. It helps to correctly classify darker images as well as brighter images. After applying histogram equalization, each image from different lighting conditions looks similar. As a result, it is easy for the CNN model to recognize and classify different expressions and the performance of CNN increases significantly.

The histogram equalization can be better explained if the term 'Histogram' is well known. Histogram can be defined as a graphical representation of the intensity distribution of an image. In other words, it represents the number of pixels for each intensity value considered. Figure 21 shows the histogram value mapping (right) of an image (left) with poor feature contrast and with high concentrated color values. From Figure 21, the x-axis represents the tonal scale (black at the left and white at the right), and the y-axis represents the number of pixels in an image. Here, the histogram shows the number of pixels for each brightness level (from black to white), and when there are more pixels, the peak at the certain brightness level is higher.

Figure 21. Histogram of an image [45]

The histogram equalization method is a computer image processing technique used to improve contrast in images. It accomplishes the transformation by effectively spreading out the most frequent intensity values of the image throughout the color spectrum. This method usually increases the global contrast of images when its usable data is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast [45]. Result of the histogram equalization is shown in Figure 22.

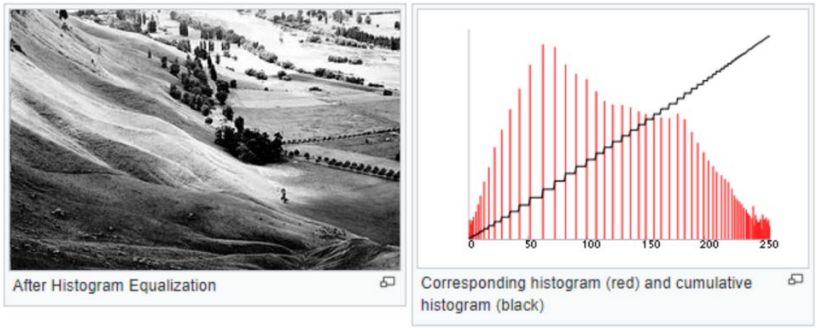

Figure 22. Result of histogram equalization

The effect of the histogram equalization transforms images with low and high contrasts to a sharper image with enhanced identifiable outlines. The preprocessing phase to the model improves the recognition of faces and expression in very dark and very bright lighting contrasts. A sample image from the KDEF dataset showing the conversion of the dark and bright images to the histogram equalized image before feeding the transformed image to the CNN is shown in Figure 23.

Figure 23. Effect histogram equalization in this thesis. Dark image(left), histogram equalized image(middle) and bright image(right)

The CNN Model Mathematical Representation

Histogram equalization

Let, $x\ [i, j]$ be the input image of size 48×48 pixels. Number of possible intensity values can be from 0 to 255, total of 256, 0 means full black and 255 means full white. If this is denoted by l, then $l = 256$. Now, normalized histogram of x can be defined as,

$$P_n = \frac{Number\ of\ pixels\ with\ intensity\ n}{Total\ number\ of\ pixels} \tag{5}$$

Here, $n = 0, 1, 2, \ldots\ldots\ldots\ldots, l - 1$. If this histogram equalized image is denoted as $A[i, j]$, then it can be expressed as,

$$A[i, j] = \left\lfloor (l - 1) \sum_{n=0}^{x[i,j]} P_n \right\rfloor \tag{6}$$

Here, $\lfloor . \rfloor$ denotes the floor operation. The floor operation returns the largest integer that is equal or smaller to the input. For example, $floor(2.56) = \lfloor 2.56 \rfloor = 2$. The image $A[i, j]$ is fed into the Convolution layer outputting an image $B[i, j]$.

$$A[i, j] \xrightarrow{Convolution} B[i, j] \tag{7}$$

Model Layers

(1) First Set of Convolutional Layers

Next, the math will be broken down for each set of convolutions, activations, Batch Normalization and Pooling layer. From the model architecture discussed previously and also from Table 1, first set of such pair can be written as CONV followed by ELU, and then BN and after that again CONV followed by ELU, BN and POOL

respectively. The CONV denotes a Convolution layer, ELU is the Activation function, BN refers to Batch Normalization and POOL refers to a Pooling layer. The model was trained with both the ReLU and ELU Activation functions but better result, discussed in the next chapter, were obtained by utilizing ELU Activation function. Therefore, the mathematical expression associated with the ELU function are described here.

For the first CONV layer, (7) can be rewritten as,

$$B[i,j] = \sum_{k_1=0}^{2} \sum_{k_2=0}^{2} A[i+k_1, j+k_2]\, W_1[k_1, k_2] \tag{8}$$

Here, W_1 is the kernel, or filter, of size 3×3. Due to the filter size, $k_1 = 0{:}\,2, k_2 = 0{:}\,2$. It is to denote that all the Convolution layer used in this model have the same zero padding at $p = 1$ and stride $s = 1$. There are three types of zero padding – full, same, and valid. For the CNN model, the same padding configuration is used due to its preservation of the height and width of the input images, or tensors, which makes the design of the network architecture more efficient. For this model design, the spatial size was preserved in the Convolution layer using the same padding and spatial size and decreased via the Pooling layers. The padding is shown in Figure 24.

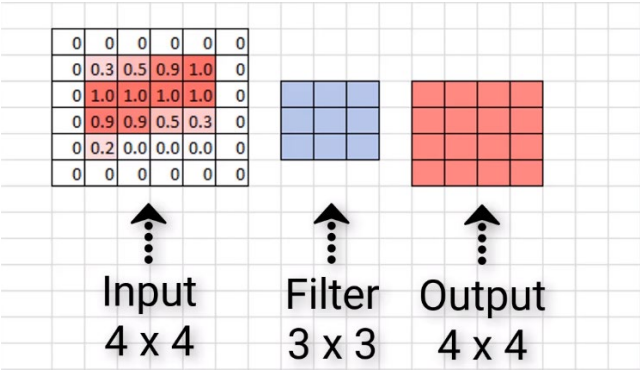

Figure 24. An example of same padding

In Figure 24 with same padding at $p = 1$, it refers to an extra layer of zeros all around the input matrix, and the output of the same size is preserved. In the Pooling operation, it is observed how this spatial size is decreased using valid padding at $p = 0$ and stride at $s = 2$. The layers output size can be computed as,

$$o = \left\lfloor \frac{n + 2p - m}{s} \right\rfloor + 1 \tag{9}$$

Here, the n term refers to the input pixel size of the image, p is the padding used, m is the filter size, and s is the stride value. Therefore, the input image is a size of 48, $n = 48$, and the output size = 48 as required for the design. The size operation is shown in (10).

$$o = \left\lfloor \frac{48 + 2 \times 1 - 3}{1} \right\rfloor + 1 = 48 \tag{10}$$

For the ELU layer, it takes the convoluted image $B[i,j]$ as its input and outputs a $C[i,j]$ image such that

$$C[i,j] = f_{ELU}(B[i,j]) = \begin{cases} B[i,j]\,;\ if\ B[i,j] \geq 0 \\ e^{B[i,j]} - 1;\ Otherwise \end{cases} \tag{11}$$

The image $C[i,j]$ is batch normalized and the BN layer comes into effect. The output from the Batch Normalization is a $D[i,j]$ image. For the transformation, the mean and variance are calculated first as shown below in (12) and (13). If the mini-batch mean and variance are denoted by $\mu_B,\ \sigma_B^2$ respectively, then,

$$\mu_B = \frac{1}{48 \times 48} \sum_{i=1}^{48} \sum_{j=1}^{48} C[i,j] \tag{12}$$

The input image of size 48 × 48 pixels is fed as an input, that's why it has been used in this equation. One point to be noted here that, the math formulas shown here in this chapter will remain same throughout the thesis except the only change will occur is in the size of the image such that 48 × 48 may become 100 × 100 after training and optimization if the model is tuned. The formulas for computation of different layers will remain same, just the values will change. So, for the final model used in the app that will be discussed later in the results chapter will contain no more mathematical explanation.

The variance σ_B^2 is obtained as,

$$\sigma_B^2 = \frac{1}{48 \times 48} \sum_{i=1}^{48} \sum_{j=1}^{48} (C[i,j] - \mu_B)^2 \tag{13}$$

The normalized image of $C\ [i,j]$ can be written as the output $D\ [i,j]$ as,

$$D[i,j] = \hat{C}[i,j] = \frac{C[i,j] - \mu_B}{\sqrt{\sigma_B^2}} \tag{14}$$

During the training of the model, $D[i,j]$ can be written in terms of the learnable parameters that are learned and updated by the CNN itself. This equation is shown below

$$D[i,j] = \gamma \hat{C}[i,j] + \beta \tag{15}$$

Here, γ and β are learnable parameters which are functions of μ_B and σ_B^2.

The three layers of CONV, ELU and BN again repeats for the model design. Using of two Convolution layers consecutively is necessary with a large number of pictures, supports the learning convergence, and performance of the CNN model. The mathematical representation is same for these three layers as shown in (8), (11), (12),

(13), and (14). Therefore, after performing computation in these three layers, an output image of $G[i,j]$ will be available as the input to the POOL layer in (16).

$$D[i,j] \xrightarrow{CONV} E[i,j] \xrightarrow{ELU} F[i,j] \xrightarrow{BN} G[i,j] \tag{16}$$

After the Batch Normalization layer, a Pooling layer is applied and, the spatial size of $G[i,j]$ is decreased to a 24×24 by applying a 2×2 filter, a padding at $p = 0$, and stride $s = 2$. The effect of applying such parameters can be seen through the example in Figure 25.

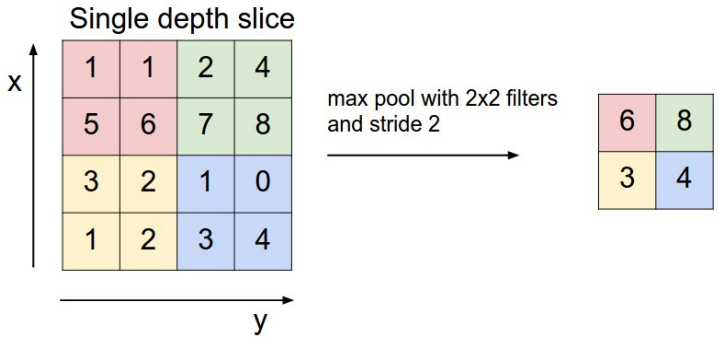

Figure 25. Effect of using a filter of size 2×2 with zero padding with a stride, s=2 on an input image of size 4×4.The resultant image is of size 2×2.

The mathematical equation representing the POOL operation is given below

$$H[i,j] = max(G[i+m, j+n]) \tag{17}$$

Here, $i = j = 0{:}\,23$ as the output size is 24, and $m = n = 0{:}\,1$, and using a 2×2 filter. Verification of the output size is shown by using (9) as,

$$o = \left\lfloor \frac{48 + 2 \times 0 - 2}{2} \right\rfloor + 1 = 24 \tag{18}$$

Throughout these set of layers shown above, number of filters used in each convolution layer and Pooling layer were 32. In the next set of layers, this number will be

doubled to 64 to better learn the low-level features of the images showing different expressions.

(2) Second Set of Convolutional Layers

These set of layers work exactly the same way as shown for the previous set of layers with the exception of input size and number of filters used. Throughout these set of layers, 64 filters have been used for each layer. However, the filter size, padding and stride value remained same for the corresponding layers. For the first CONV layer, the input image will be the output from the previous POOL layer denoted as $H[i,j]$ and the output of this layer is $I[i,j]$ as shown in (19). This output image is the input of the ELU layer and output of ELU layer is $J[i,j]$ as shown in (20). Batch Normalization is then applied to $J[i,j]$ and output of this layer is $K[i,j]$ as shown in (21).

$$I[i,j] = \sum_{l_1=0}^{2}\sum_{l_2=0}^{2} H[i+l_1, j+l_2]\, W_3[l_1,l_2]; i = j = 0{:}\,23 \tag{19}$$

$$J[i,j] = f_{ELU}(I[i,j]) = \begin{cases} I[i,j]\,;\ if\ I[i,j]\ \geq 0 \\ e^{I[i,j]} - 1;\ Otherwise \end{cases} \tag{20}$$

$$K[i,j] = \frac{J[i,j] - \frac{1}{24\times 24}\sum_{i=1}^{24}\sum_{j=1}^{24} J[i,j]}{\sqrt{\frac{1}{24\times 24}\sum_{i=1}^{24}\sum_{j=1}^{24}(C[i,j] - \frac{1}{24\times 24}\sum_{i=1}^{24}\sum_{j=1}^{24} J[i,j])^2}} \tag{21}$$

Keeping $i = j = 0{:}\,23$ and same number of filters these three steps repeats again such that

$$K[i,j] \xrightarrow{CONV} L[i,j] \xrightarrow{ELU} M[i,j] \xrightarrow{BN} N[i,j] \tag{22}$$

Output size of $N[i,j]$ is $24 \times 24 \times 64$, where 64 is the number of kernels, which is also known as the depth of the image. This output will be the input of the next max-pooling layer, in where the spatial size is reduced to 12×12, and number of filters will be doubled to 128 after this Pooling layer. For this phase, the height and width of the image will be reduced, and depth will be increased in order to learn the high-level features of the image in the next set of Convolution layers. The Pooling operation at this stage is expressed by,

$$O[i,j] = max(N[i+m, j+n]) \tag{23}$$

Here, $i = j = 0:11$ as the output size is 12 and $m = n = 0:1$ as the filter size is 2×2. The output size can again be verified by equation (9).

$$o = \left\lfloor \frac{24 + 2 \times 0 - 2}{2} \right\rfloor + 1 = 12 \tag{24}$$

(3) Third Set of Convolutional Layers

The set of layers work exactly the same way as shown for the previous set of layers with the exception of input size and number of filters used. Also, here, there is an extra set of CONV, ELU, and BN layers compared to previous layers. Throughout these set of layers, 128 filters have been used for each layer. But, the filter size, padding and stride value remained same for the corresponding layers. For the first CONV layer, the input image will be the output from the previous POOL layer indicated by $O[i,j]$ and the output of this layer $P[i,j]$ is shown in (25). Output of ELU is $Q[i,j]$ as shown in (26) and output of BN is $R[i,j]$ as shown in (27).

$$P[i,j] = \sum_{m_1=0}^{2} \sum_{m_2=0}^{2} O[i+m_1, j+m_2]\, W_5[m_1, m_2]; i = j = 0:11 \tag{25}$$

$$Q[i,j] = f_{ELU}(P[i,j]) = \begin{cases} P[i,j]\,;\ if\ P[i,j] \geq 0 \\ e^{P[i,j]} - 1;\ Otherwise \end{cases} \quad (26)$$

$$R[i,j] = \frac{Q[i,j] - \frac{1}{12 \times 12} \sum_{i=1}^{12} \sum_{j=1}^{12} Q[i,j]}{\sqrt{\frac{1}{12 \times 12} \sum_{i=1}^{12} \sum_{j=1}^{12} (Q[i,j] - \frac{1}{12 \times 12} \sum_{i=1}^{12} \sum_{j=1}^{12} Q[i,j])^2}} \quad (27)$$

Keeping $i = j = 0{:}\,11$ and same number of filters these three steps repeat twice such that

$$R[i,j] \xrightarrow{CONV} S[i,j] \xrightarrow{ELU} T[i,j] \xrightarrow{BN} U[i,j] \xrightarrow{CONV} V[i,j] \xrightarrow{ELU} W[i,j] \xrightarrow{BN} X[i,j] \quad (28)$$

The output of $X[i,j]$ is fed to the Pooling layer where spatial size is decreased to a 6×6 using a 2×2 filter with a padding at $p = 0$ and stride $s = 2$. The POOL operation is given by:

$$Y[i,j] = max(X[i+m, j+n]) \quad (29)$$

Here, $i = j = 0{:}\,5$ as the output size is 12 and $m = n = 0{:}\,1$. The output size is verified by (9).

$$o = \left\lfloor \frac{12 + 2 \times 0 - 2}{2} \right\rfloor + 1 = 6 \quad (30)$$

(4) Fully-Connected Layer (Dense)

Fully-Connected layers work in the similar way as a Convolution layer works. After performing Dense operation, ELU and BN are applied. Also, the same workflow repeats again as seen from the model architecture in Table 1. However, before applying the dense layer, output from the POOL layer $Y[i,j]$ is flattened to convert it from 2D to

1D. This is needed to predict the probability of each class in the Softmax layer. Because Fully-Connected layer takes input as a one-dimensional vector; it can't process 2D images as a Convolutional layer operates. Therefore, after performing a Flatten operation, the total data points remaining is $6 \times 6 \times 128 = 4608$. This Flatten operation is shown in (31).

$$Z[p] \times 128 = Y[i, j] \times 128 \tag{31}$$

Here, $p = i \times j = 6 \times 6 = 36$. The Flatten operation can be visualized as shown in Figure 26.

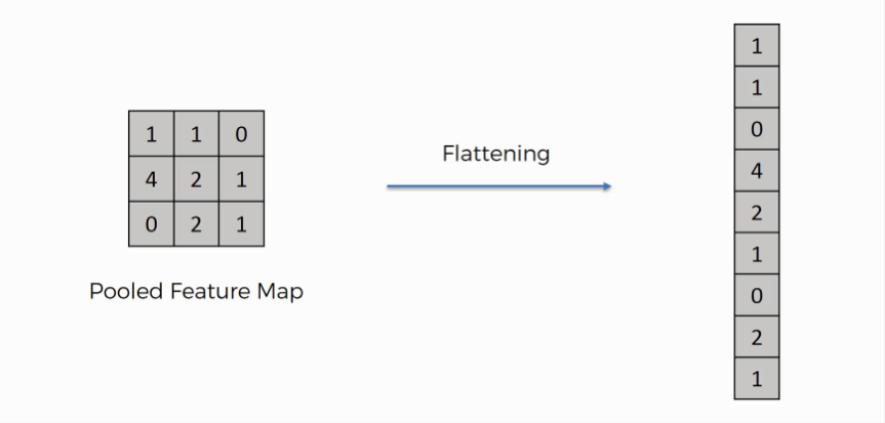

Figure 26. An example of Flatten operation

Now, the first set of Dense, ELU, and BN layers can be described mathematically as in (32), (33), and (34) respectively.

$$a[q] = Z[p] \cdot \boldsymbol{W} \tag{32}$$

Here, $q = 0{:}\,63$ and $\boldsymbol{W}$ is a learnable kernel/filter during training which is knows as weight matrix. The size of this matrix is learnt by CNN itself as the output size of Dense layer is given by the user. Now, this output is the input to the ELU layer.

$$b[q] = f_{ELU}(a[q]) = \begin{cases} a[q]\,;\ if\ a[q] \geq 0 \\ e^{a[q]} - 1;\ Otherwise \end{cases} \tag{33}$$

$$c[q] = \frac{b[q] - \frac{1}{64}\sum_{q=1}^{63} b[q]}{\sqrt{\frac{1}{64}\sum_{q=1}^{63}(b[q] - \frac{1}{64}\sum_{q=1}^{63} b[q])^2}} \tag{34}$$

After the Batch Normalization, the second set of Dense, ELU, and BN is applied containing the same output of 64 vectors.

$$c[q] \xrightarrow{Dense} d[q] \xrightarrow{ELU} e[q] \xrightarrow{BN} f[q] \tag{35}$$

The final dense layer is used to provide the scoring function for each class, in this case seven, and the scoring function is determined as,

$$g[r] = f_{ELU}(f[q] \cdot \boldsymbol{K}) \tag{36}$$

Here, $r = 0{:}\,6$ (total seven classes) and $\boldsymbol{K}$ is learnable filter which is learnt and updated by the CNN itself during the training process. Finally, the probability of each class and training loss is calculated by the Softmax layer. The probability of each class P_c can be found by the scoring function $g[r]$,

$$P_c = \frac{e^{g[r]}}{\sum_{i=1}^{7} e^{g[i]}} \tag{37}$$

Loss of each class, L_c can be computed as,

$$L_c = -\log(P_c) \tag{38}$$

Total gross-entropy loss, L_{Total} is the average of total seven classes,

$$L_{Total} = \frac{1}{7}\sum_{i=1}^{7} L_i \tag{39}$$

Training the CNN model

All the to train the CNN model such as datasets, preprocessing, working principal of CNN layers, and their mathematical expressions have been discussed. The parameters needed to tune a CNN model during training, known as hyperparameters, and the optimizer and regularization techniques that were used for the model design are discussed.

Optimization Method

Before the discussion of hyperparameter tuning, it is essential to cover the final component of any neural network architecture – the optimization method. The literature review reveals there are several optimization methods available for deep learning training such as the Stochastic Gradient Descent (SGD) [46], RMSprop [47], Adagrad [48], Adadelta [49], and Adam [50] techniques. The SGD continues to be one of the most utilized in model design as the main optimization method. For the model approach, The SGD and Adam optimizers were tested against the datasets and with the model. During experimentation, the SGD outperforms the Adam optimizer in performance and accuracy. In common initial steps, the CNN is designed with the SGD optimization method. The Adam method is used for the CNN model design for image recognition problems and found to provide optimized parameters for the model. The optimizer choice depends on the datasets and their size, image quality and also on the model architecture.

The standard gradient descent algorithm updates weight matrix on the entire training set, and it finds the approximation of the gradient. As a result, this algorithm

increased the training time and computationally wasteful for large datasets. Instead, the SGD is a simple modification to the standard gradient descent algorithm that computes the gradient and updates the weight matrix on small batches of the training data, rather than the entire training set [30]. For training a deep neural network, the SGD is considered to be the most important algorithm. The batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. The SGD method removes redundancy by performing one update at a time. Therefore, it is faster to converge with no negative effects to loss and classification accuracy.

For the model to learn faster and reduce the training time, an extension to the SGD was used for the model, which is known as Nesterov Acceleration [51]. Momentum methods are normally used to accelerate learning in order to fast convergence. But what if momentum is uniform and unintelligent such that it overshoots a local minimum. In order to avoid such scenario, Nesterov acceleration is used which is intelligent enough to know when to slow down its momentum. A visual representation can better explain this method which is shown below in Figure 27.

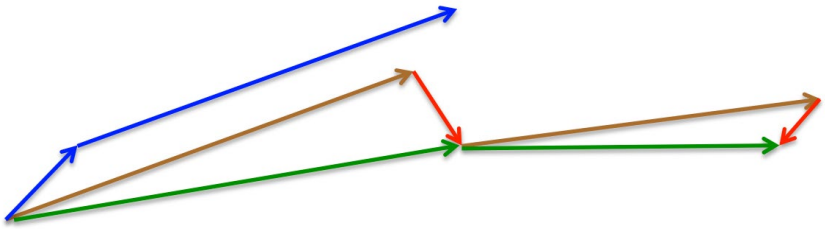

Figure 27. Graphical representation of the Nesterov Acceleration [52]

In Figure 27, the blue vectors represent the standard momentum. First, the Nesterov acceleration makes a big jump in the direction of the previous accumulated gradient. Then, it measures the end point value of the gradient where it ends up and make

a correction. This jump is shown by the brown vector, correction is shown by the red vector, and the green vector represents the accumulated gradient [52].

Regularization

Regularization helps to reduce overfitting and provides the model the ability to better generalize the model. During experimentation and optimization of the model, the L1 and L2 regularization is added to the model to see if the model reduces overfitting and be able to better generalize. However, no improvement of generalizability for the final model were observed; therefore, no regularization was used except the inclusion of the Dropout layer. Additionally, some data augmentation techniques were used that essentially provides the model more flexibility to unobserved data, reduce overfitting, and enhance the ability to generalize. These techniques will be discussed in the next chapter while discussing result of training.

Hyperparameter Tuning

Tuning the hyperparameters in the training process is essential to accelerate the execution time of the training and efficiency of the model. In this section, different hyperparameters are discussed starting with the most impactful parameters and effects that each present to the design of the model.

Learning rate

The learning rate is an impactful hyperparameter in a neural network training. It controls by how much to update the weight in the optimization algorithm. The design approach is to use a fixed learning rate, a gradually decreasing learning rate, a momentum-based methods or adaptive learning rates, and it varies on the choice of an optimizer such as SGD, Adam, Adagrad, AdaDelta or RMSProp.

For the model design and to alleviate the overfitting problem, a learning rate scheduler is used with the SGD optimizer. The large consideration of the learning rate schedulers is to increase the model's accuracy, and its standard weight update formula is described as,

$$W = -\alpha \times gradient \tag{40}$$

Larger values of the learning rate α such as 0.01, 0.1,0.5 represent the model is taking big steps towards minimum loss. Small values of α such as 0.001, 0.0005, 0.0001 represent the model is taking small steps towards convergence. The learning rate schedules decay the initial learning rate by some decay parameter. It is necessary because if the learning rate is constant and high then it can overshoot some local minima and again if the learning rate is low initially, then the training time will be much higher. So, it is a good practice to initially start with a reasonable learning rate and then decay it throughout the training process. This decreased rate enables the network to descend into areas of the loss landscape that are "more optimal" and would have otherwise been missed entirely by larger learning rate.

For all the experiments for the model, an initial learning rate of 0.01 was selected after trying out other rates such as 0.1, 0.001, and 0.0001. The best result was achieved with the learning rate of 0.01. The default Keras learning rate schedulers was used with a decay parameter of respectively 0.01/10, 0.01/20, 0.01/30, 0.01/40, and 0.01/50 with an optimal result achieved with a decay value at 0.01/20. The SGD optimizer was initialized with a momentum value of 0.9, and the Nesterov accelerated gradient was used. Keras applies the following formula for adjusting learning rate after every batch update.

$$lr = initial\_lr \times (1 / (1 + decay \times iterations)) \tag{41}$$

Here, the term lr refers to the learning rate, and $initial\_lr$ rate is 0.01. Iterations can be calculated as total number of training images in the dataset divided by the batch size.

Number of epochs

The number of epochs is the number of times the entire training set pass through the neural network. Number of epochs should be increased until a small gap between the test error and the training error is achieved. Initially, training of the model started with 70 epochs and gradually increased up to 200 epochs as the dataset increased.

Batch size

The mini-batch is usually preferable in the learning process of a CNN because it reduces the training time and makes convergence faster. A range of batch sizes from 16 to 128, by log Base-2 numbers, were utilized for experimentation steps. The model was tested with batch sizes of 32, 64, and 128. The best result was achieved with a batch size of 64. It should be noted that a CNN is sensitive to batch size as in the performance and accuracy depends on the batch size.

Activation function

The activation function introduces non-linearity to the model by mathematical conversion which is necessary for any machine learning algorithm as it should have the ability to process any function universally. As previously discussed, both ReLU and ELU functions have been tested on the model, and the optimal result was achieved with the ELU function.

Number of hidden layers and units

It is usually a good practice to add more hidden layers until the test error no longer improves. The trade-off of adding more hidden layers is that it is computationally expensive to train the network. The final DCNN model has three CONV-POOL layers with two Fully-Connected layers at the end of the network. Having a small number of hidden units may lead to underfitting but having more units are usually not harmful with appropriate regularization.

Weight initialization

The process of initializing weight matrices and bias vectors are known as weight initialization. There are different types of weight initialization such as constant initialization with zeros, or ones, uniform and normal, LeCun uniform and normal [53], Glorot/Xavier uniform [54] and normal, He et al./Kaiming/MSRA uniform and normal [55]. Keras default uniform initializer is used for the final model. But to test for better performance of the model, MSRA initialization was also used. The MSRA initialization is typically used when very deep neural networks are trained that use a ReLU-like activation function. To initialize the weights in a layer using He et al./MSRA initialization with a uniform distribution the limit is set to be,

$$limit = \sqrt{6/F_{in}} \tag{42}$$

where F_{in} is the number of input units in the layer. MSRA initialization couldn't provide better results than Keras default initializer.

Dropout for regularization

Dropout is a preferable regularization technique to avoid overfitting in deep neural networks. The method applies a dropout of units in neural network according to the

desired probability. Dropout values were also tuned after each Convolution and Fully-Connected layer in order to improve the performance of the model. It has been found that, the best accuracy was achieved with a dropout value of 0.25 after the Convolutional layers and a 0.5 dropout value after the Fully-Connected layers.

As a summary, the components needed for to train a DCNN has been discussed such as datasets, model architecture, loss function, preprocessing, and optimization methods along with mathematical explanation of each layer of the model architecture. The training results and experimentation methods are denoted in the following chapter. The algorithm development for the facial expression recognition and the methodology discussed in this chapter are discussed with obtained results. Further, the import of the model and development of the iOS app will be discussed in the later chapter.

VI. RESULTS AND DISCUSSION

In this section, the training and test results with different datasets are discussed starting with the FER2013 dataset and its corresponding brighter and darker datasets. The KDEF dataset is discussed for its side angle images, as well as, its modified versions for different lighting conditions. Different experiments were performed by changing the hyperparameters which are shown via plots and tables for both of the datasets to understand the accuracy effects of the model. Also, the computational resources used in this thesis for training the CNN are discussed here.

Computational Resources

For training the DCNN model used in this thesis, the LEAP (Learning, Exploration, Analysis, and Processing) next-generation High-Performance Computing (HPC) Cluster [56] of Texas State University was used. The cluster's head-node and 120 compute-nodes are configured with the Dell PowerEdge C6320 each with 28 CPU cores via two (14-core) 2.4 GHz E5-2680v4 Intel Xeon (Broadwell) processors. With 128GB of memory and 400GB of SSD storage per node, the compute-nodes provide an aggregate of 15TB of memory and 48TB of local storage. Additionally, the LEAP cluster features two large memory (1.5TB of RAM) nodes with 64 CPU cores via four (16-core) 2.5 GHz E7-8867v3 Intel Xeon (Haswell) processors. Figure 28 shows the features and components of the LEAP cluster.

All scripts used in this research were written in Python 3.5.7 programming language. Conventional and popular deep learning libraries such as Keras (TensorFlow backend), Numpy, Scikit-learn, pandas, Matplotlib are used. For image processing, OpenCV for python was utilized.

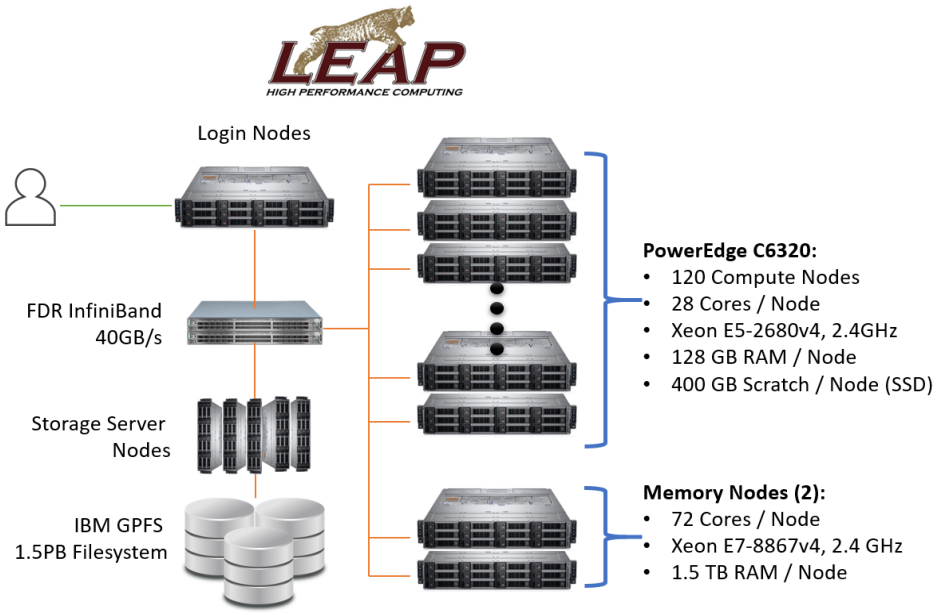

Figure 28. Features and components of LEAP cluster

Training and Testing Results

The initial experimentations will be discussed done with the FER2013 dataset and its modified version for different lighting conditions to get the best initial model for facial expression recognition. The model was then optimized for better results utilizing sideview images from the KDEF dataset and their corresponding brighter and darker versions. Finally, the best model will be discussed that has been imported into the iOS application.

Combination of experimentation were performed to get a reliable test accuracy for different version of the FER2013 dataset. Different parameters of the model stated in previous chapter are changed in each experiment and the corresponding result is then compared to get the best result. Initially, the standard SGD optimizer with the ReLU activation function was utilized. This model had a batch size of 128 and Keras default kernel initializer. The model was trained with different learning rates of 0.01, 0.005, 0.001, 0.0005, and 0.0001. However, it was seen from the loss/accuracy curves that for

all the above learning rates was causing the model to overfit. It is worth mentioning here that all these models had no Dropout layer before the Fully-Connected layers. Only after the Fully-Connected layers a Dropout layer with a value of 0.5 was utilized for the initial experiments. The best validation accuracy achieved was 60.23% with a learning rate of 0.01.

The next experiment was of the same configuration from the initial experiment but including a Dropout layer with a value of 0.25 after every Pooling layer. This exception made a significant increase in the validation accuracy up to 65.24% was achieved with a learning rate of 0.01. However, it was seen from the loss/accuracy plot that the model was continuing to overfit. All these models were run for 70 epochs. The loss/accuracy plot for this model with a learning rate of 0.01 is given in Figure 29. The Matplotlib library was used to generate these loss/accuracy plots.

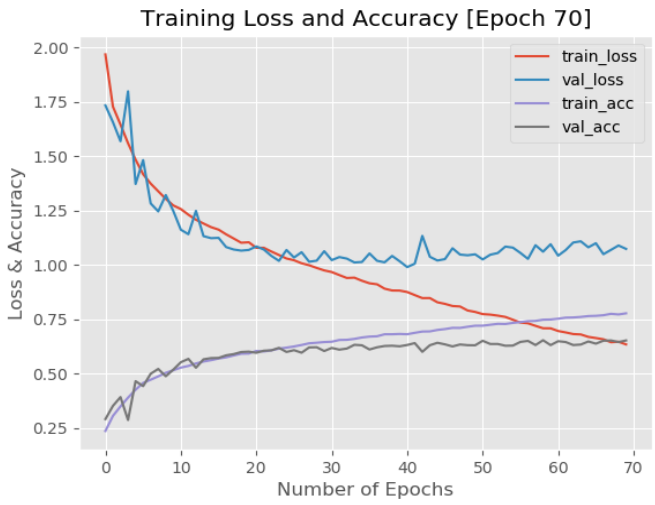

Figure 29. Loss/ Accuracy plot for the model with a learning rate of 0.01 for 70 epochs

The learning rate schedulers, which was discussed in the previous chapter, were established to solve this overfitting problem in the next batch of experiments. For this batch of the experiment, an initial learning rate of 0.01 was selected as the best result so

far was achieved with this learning rate. The default Keras learning rate schedulers was used with a decay parameter of respectively 0.01/10, 0.01/20, 0.01/30, 0.01/40, and 0.01/50. The SGD optimizer was initialized with a momentum value of 0.9 and Nesterov accelerated gradient was used. Nesterov accelerated gradient was described in the previous chapter.

From this batch of the experiment, the overfitting problem is removed in expense of validation accuracy. The best result from this experiment was with a decay parameter of 0.01/20 and the validation accuracy was 63.77%. The loss/accuracy plot for this model is given in Figure 30. From the loss/accuracy plot it is seen that both loss and accuracy of the model do not actually stagnate after epoch 70. For the next experiment the approach was to increase the number of epochs from 70 to 100.

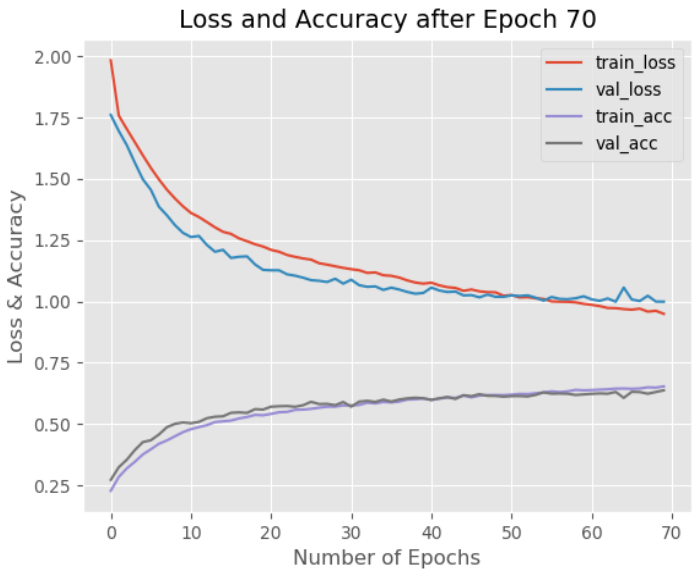

Figure 30. Loss/ Accuracy plot for the model of the third experiment with a starting learning rate of 0.01 and a decay parameter of 0.01/20.

For the next experiment, the number of epochs were increased to 100, and the activation function was also changed from the ReLU to the ELU function. This yielded a

validation accuracy of 62.50%. However, changing the batch size from 128 to 64, and the initializer to MSRA initialization from Keras default yielded a validation accuracy of 63.26% and a test accuracy of 62.20%. The plot/accuracy plot for this model is given in Figure 31.

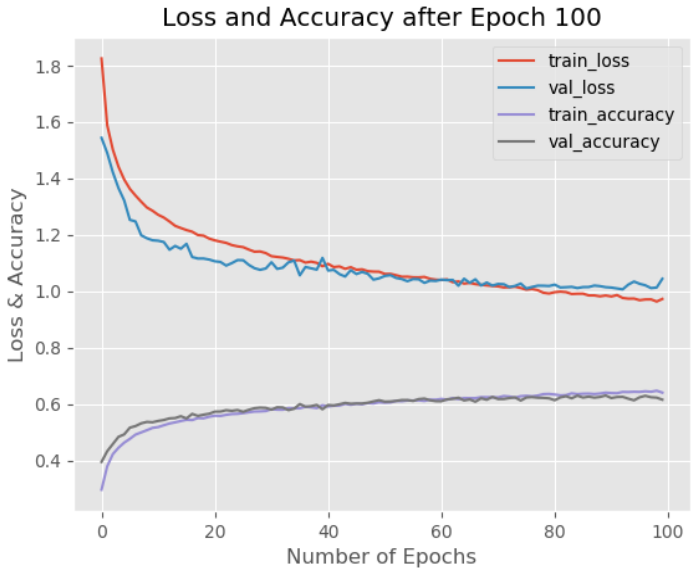

Figure 31. Loss/ Accuracy plot for the model with a starting learning rate of 0.01and a decay parameter of 0.01/20.

For the next experiment, the Adam optimizer was initially considered over the SGD. Though this optimizer provided the best validation accuracy achieved so far which is 66.81%, the model was overfitting due to these changes. Therefore, sticking with SGD, the change to the Keras default initializer was made and a validation accuracy of 64.88% was achieved without overfitting. The loss/accuracy plot for this model is shown in Figure 32. It should be mentioned that some data augmentation techniques were applied to this model such as rotation, changes in scale and horizontal flip. These data augmentation techniques help to obtain more training data by changing geometric features of the original images. As a result, data augmentation can increase

generalizability of the model and test accuracy. For this reason, the validation accuracy is slightly improved with the training accuracy of 63.74%. Also, the use of Dropout layers after every Pooling layers while training the dataset, as well as, the batch size can be a reason for the improvement of accuracy. The test accuracy achieved from this model was 63.11%.

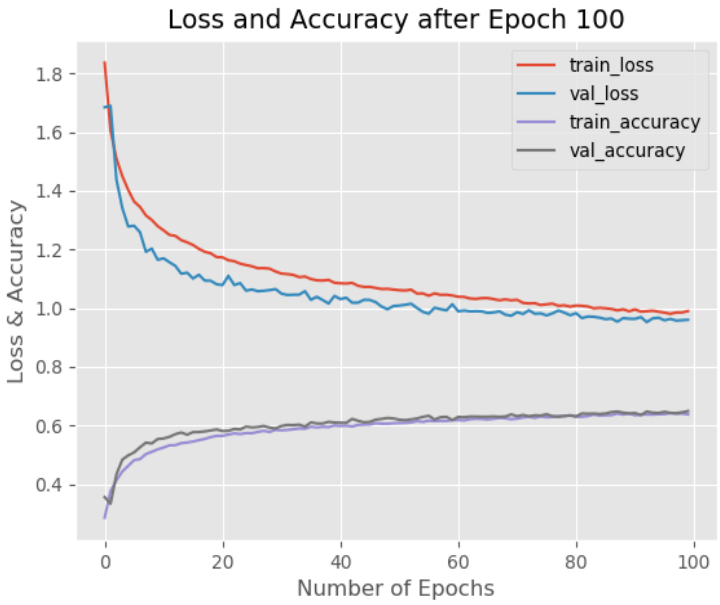

Figure 32. Loss/Accuracy plot for the model with Adam optimizer

Next, the same model obtained from the last experiment was used to train four different modified FER2013 dataset containing images of different lighting (darkest, darker, brighter, brightest) as discussed in the previous chapter. The modified datasets were produced by utilizing Python and OpenCV. For the datasets with brighter and brightest images, test accuracies of 59.65% and 58.37% were achieved respectively, which are close to the test accuracy from the original FER2013 dataset of 63.11%. For the datasets with darker and darkest images, test accuracies achieved were 53.10% and 46.46% respectively and performed worse compared to the brighter contrast images.

Therefore, it is concluded that the model was having some difficulties recognizing dark contrast facial images. Figure 33, Figure 34, Figure 35, and Figure 36 show the loss/accuracy plots for these four different datasets - darkest, darker, brighter, brightest respectively using the final model. A comparison for test accuracies with all the different datasets is shown in Figure 37. The chart indicates that this model outperforms in accuracy with the bright contrast images than the dark contrast images.

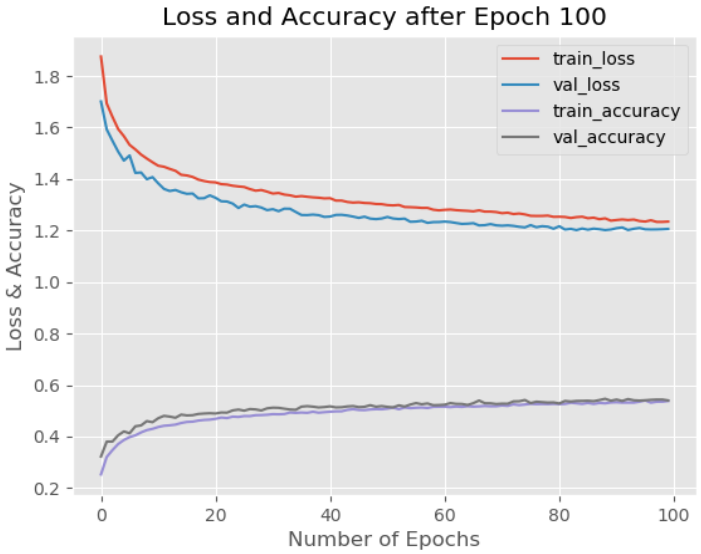

Figure 33. Loss/Accuracy plot for modified FER2013 dataset with darkest images

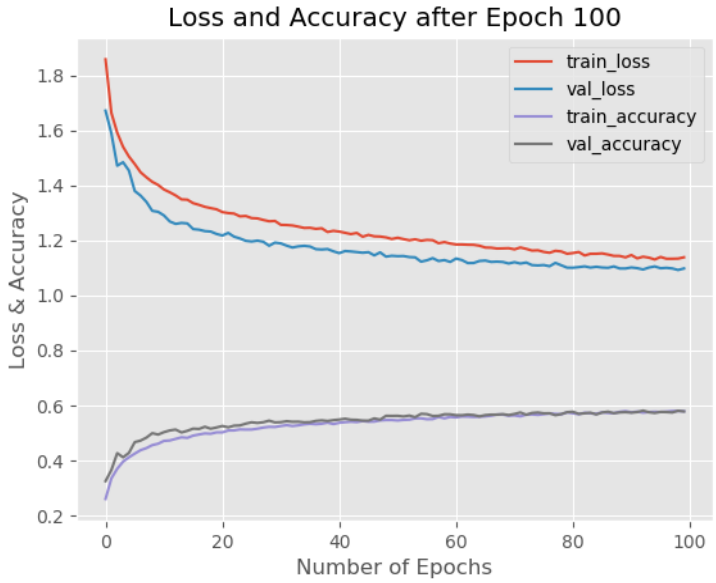

Figure 34. Loss/Accuracy plot for modified FER2013 dataset with darker images

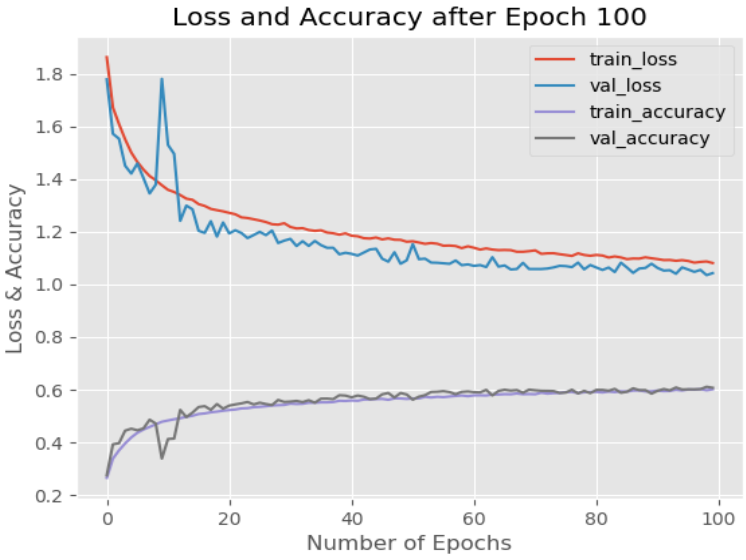

Figure 35. Loss/Accuracy plot for modified FER2013 dataset with brighter images

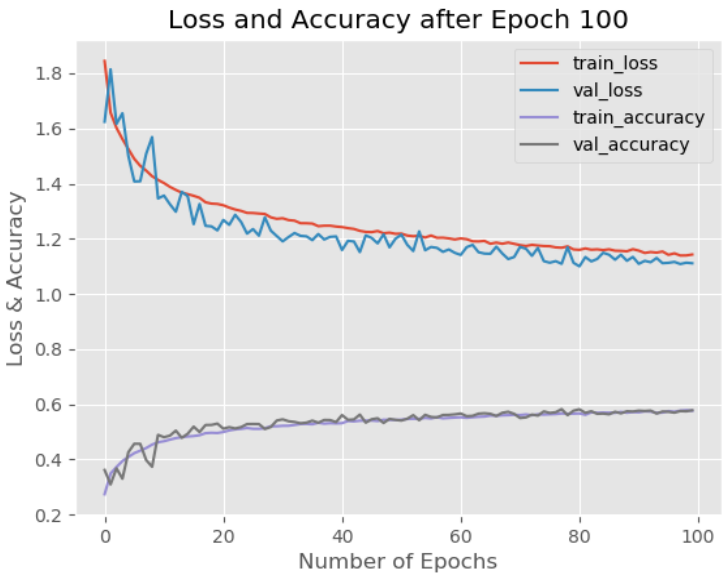

Figure 36. Loss/Accuracy plot for modified FER2013 dataset with brightest images

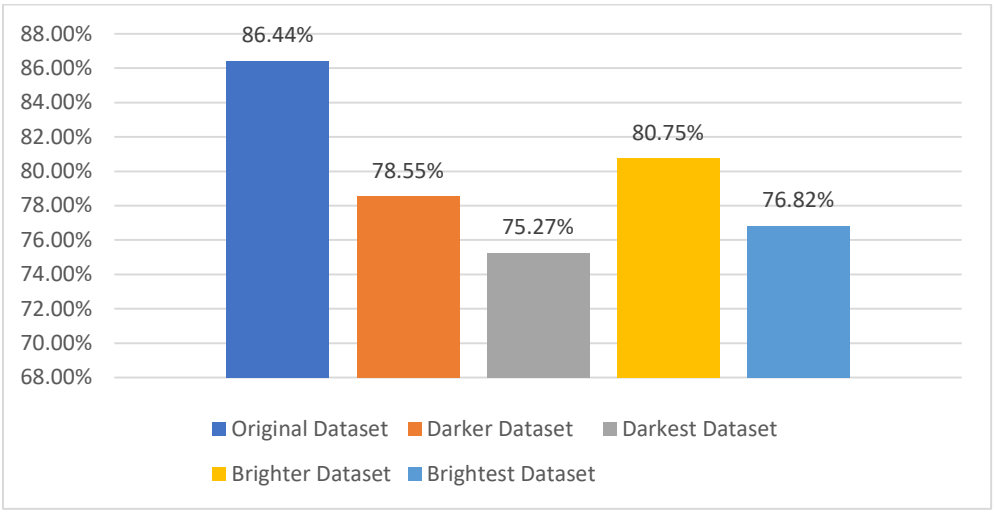

Figure 37. Comparison of test accuracies with different FER2013 datasets

The inclusion of the sideview images form the KDEF dataset to the model are essential to recognize facial expressions from different viewpoints and increase the accuracy and performance of the model for darker images. The same model hyperparameters values are used from the last experiment with the FER2013 dataset.. These set of experiments are described below.

The learning rate was set at 0.001, and the default Keras learning rate scheduler was used with a decay parameter of 0.001/20. The SGD optimizer was initialized with a momentum value of 0.9, and the Nesterov accelerated gradient was used. A training accuracy of 53.30% was achieved after training the DCNN model with the KDEF dataset, and the validation accuracy was 62.50%. This experiment was run for a total of 100 epochs with a batch size of 64. The loss and accuracy plot shown in Figure 38 indicates that the model is still learning which means the training loss is still decreasing and training accuracy is still increasing. The experimentation included data augmentation techniques using the Keras library to the dataset at the preprocessing stage of the DCNN.

These data augmentation techniques include rotation, changes in scale and horizontal flip which are done automatically by the Keras library. These techniques help to obtain more training data by changing the geometric features of the original images. As a result, the data augmentation can increase the generalizability of the model and improve the test accuracy. For this reason, the validation accuracy is greater than the training accuracy. Also, the train-test split is done per emotion category such that there are only 500 images for training, 100 images for validation and 100 images for testing. Due to the spilt, the model obtains a low training accuracy and a high validation accuracy. The learning rate is also a factor to this discrepancy between accuracies.

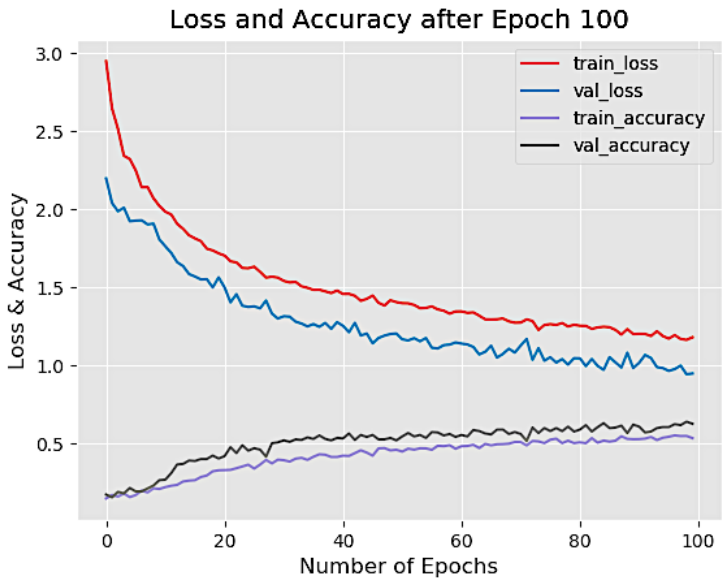

Figure 38. Loss/ Accuracy plot for the model with a starting learning rate of 0.001and a decay parameter of 0.001/20

A starting learning rate of 0.01 is used with a learning rate schedule of 0.01/20 to alleviate this problem of discrepancy between accuracies. Everything else was the same with the exception the fact that this experiment was run for a total of 200 epochs with a batch size of 64. This experiment boosted the training accuracy to 89.76%, and the validation accuracy reached 80.82%. The test accuracy was found to be 78.32%. From the performance plot in Figure 39, it is seen that both validation loss and accuracy converge to an average steady state condition which concludes small to neglectable change to the test accuracy if this experiment is continued to run for more epochs. The peaks seen in the validation loss and validation accuracy curve are due to a higher learning rate at the start and nature of the dataset, which includes images from five different angles.

The results achieved show that the model is capable of handling images from different angles with high accuracy. Also, the reason that the model is performing better

with the KDEF dataset compared to the FER2013 dataset is that the KDEF dataset contains RGB images with a resolution of 562×762 pixels. In contrast, the FER2013 dataset contains grayscale images of only 48×48 pixels. Here, the model takes the input image of higher resolution, preprocess it, and converts it to a 48×48-pixel image while keeping the aspect ratio the same. Due to the higher resolution of the original image in the data, the model can recognize features with higher accuracy and performance with the KDEF dataset than the FER2013 dataset. As a result, the development of the overall application required for the model to continue training that would resemble the high-resolution images of the KDEF dataset in order to increase accuracy and performance.

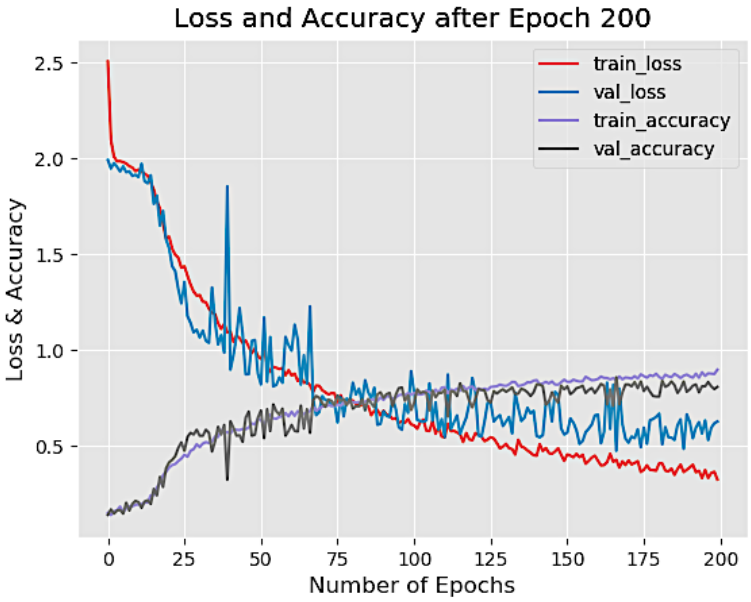

Figure 39. Loss/ Accuracy plot for the model of the first experiment with a starting learning rate of 0.01and a decay parameter of 0.01/20

As 78.32% test accuracy was achieved with this model. At this level of accuracy can be classified as a satisfactory result of its generalization and higher accuracy levels might overfit the model. From this point, the model was then tested with different values of hyperparameters to see if there was any considerable change in test accuracy. First, the

input image resolution size was changed and increased to different pixel values such as 90×90, 100×100, 128×128, 150×150, 400×400, and 512×512. Among all of these input resolutions, the model performed with the highest accuracies using the 100×100 pixel size. For the mini-batch size, the values of 32, 64, and 128 were tested among, which value of 64 gave the best result as expected from the experiments done already performed. The activation function, optimizer, learning rate, and number of epochs are the hyperparameters that were tuned and selected in previous experiments. The ELU activation function was used with the SGD optimizer. An initial learning rate of 0.01 was used with the default Keras learning rate scheduler with a decay parameter of 0.01/20. The Dropout values were also tuned after every Pooling layer with values of 0.25 and 0.5. The Pooling layers are divided into two categories, such that after convolution Pooling and after Fully-Connected Pooling, four combinations of Dropout values for these two types of Pooling layers were considered – 0.25:0.25, 0.5:0.5, 0.25:0.5, and 0.5:0.25. Here, the 0.25:0.25 means the Pooling layer after the convolution layers have a 0.25 (or 25%) value of Dropout connections, and the Pooling layers after Fully-Connected layers have 0.25 value of Dropout. It was found through experiments that the best accuracy was achieved by configuring the Dropout layers with the 0.25:0.5 combination. The final DCNN model was trained for 200 epochs, and the best result obtained was with a test accuracy of 86.44%. The loss/accuracy plot for this result of the final DCNN model, which can recognize facial expressions from different angles is shown in Figure 40.

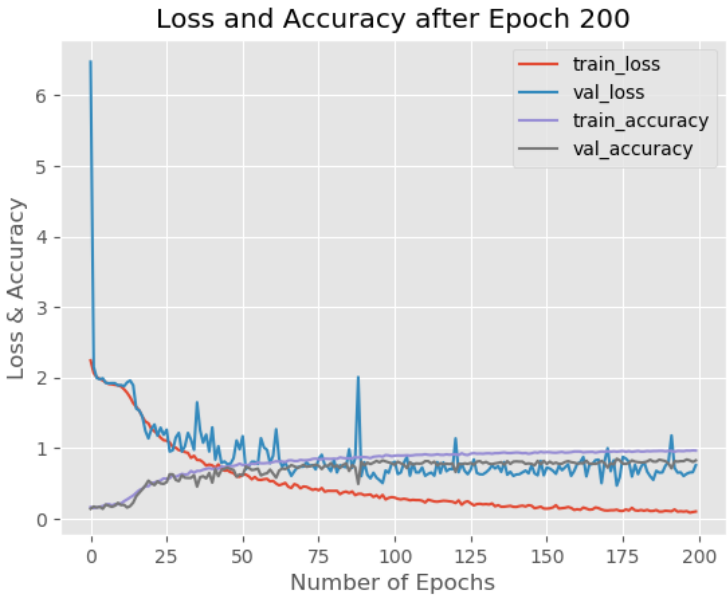

Figure 40. Loss/Accuracy curve for the final DCNN model

The final DCNN model for different lighting condition was trained and tested with modified versions of the KDEF dataset for darker and brighter images, as shown in the previous chapter. The hyperparameters were again tuned to observe any changes for these modified datasets if for brighter images with a batch size 32 of is performing better or if the Adam optimizer provides higher accuracies. The same conditions and observations apply for the darker versions of the dataset. However, after tuning and experimentation, it was found as expected that the model performs best when trained with the modified datasets with the same hyperparameter values as of the final model discussed above. The final DCNN model gave a test accuracy of 75.27%, 78.55%, 76.82%, and 80.75% respectively for darkest, darker, brightest and brighter versions of KDEF dataset and respective loss/accuracy plots are given in Figure 41, Figure 42, Figure 43 and Figure 44. The comparison of these results is shown in Figure 45.

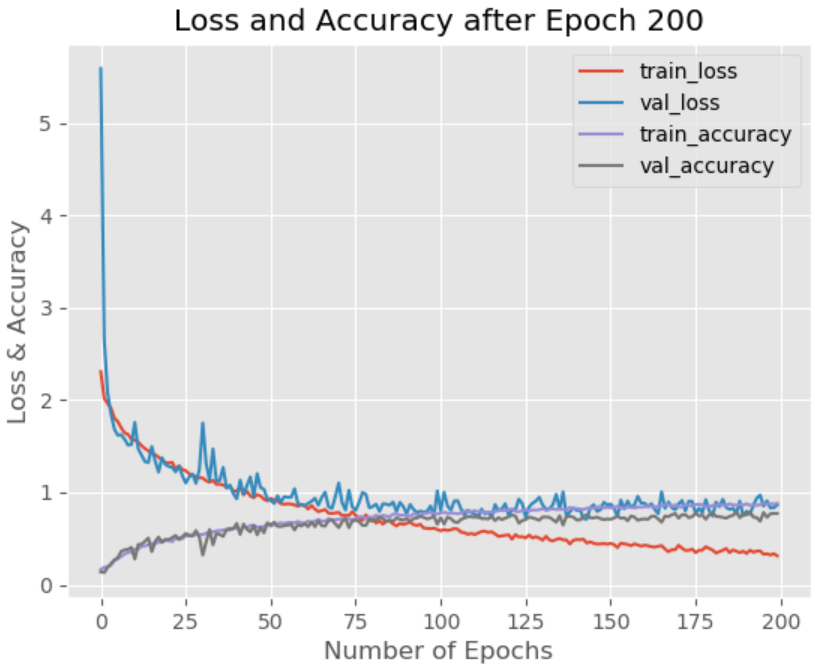

Figure 41. Loss/Accuracy plot for darkest version of KDEF dataset

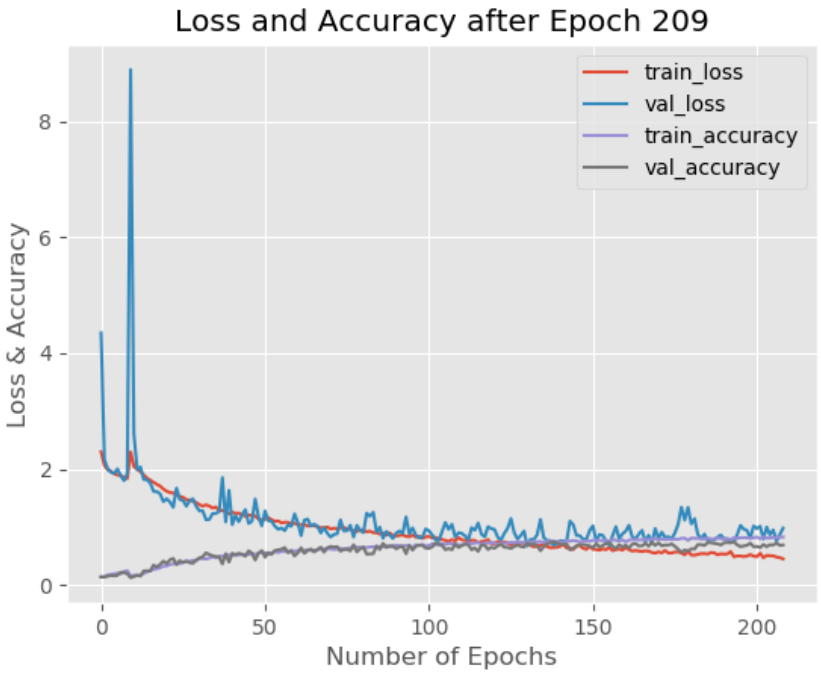

Figure 42. Loss/Accuracy plot for darker version of KDEF dataset

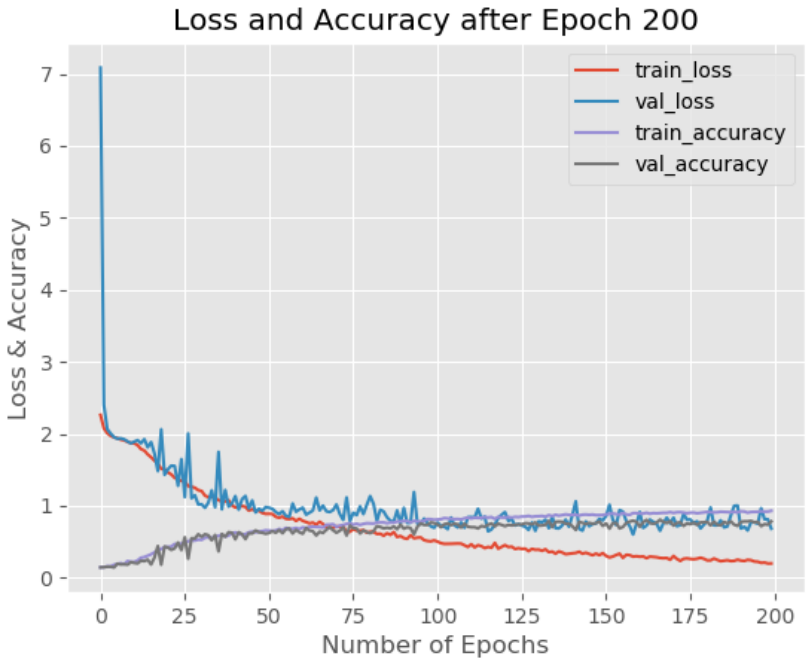

Figure 43. Loss/Accuracy plot for brighter version of KDEF dataset

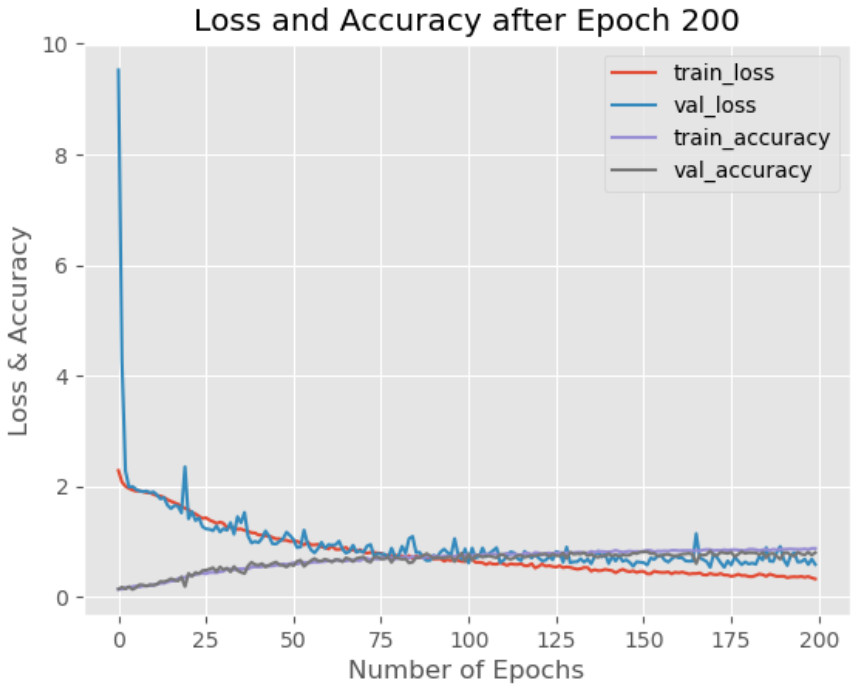

Figure 44. Loss/Accuracy plot for brightest version of KDEF dataset

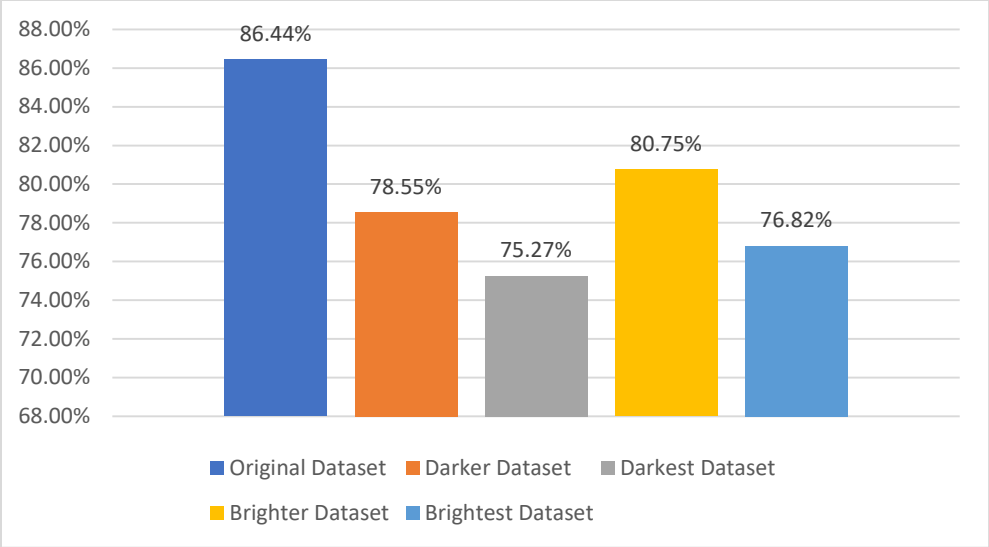

Figure 45. Comparison of test accuracies for different versions of KDEF dataset

Comparing this final result with the previous one shown in Figure 37, it can be stated that the model is able to recognize facial expressions in every lighting condition that includes dark condition as the accuracy improved from 46.46% to 75.27%. Also, the model can recognize images from five different angles due to the KDEF dataset on top of FER2013 dataset and obtaining a test accuracy of 86.44%. The top and bottom-view angles can also be tracked by the algorithm for happy and sad faces. The model was trained with the third dataset built with top and bottom-view images as discussed in the previous chapter. In terms of execution time, the latency of the overall algorithm from taking an input image to label the correct class was measured at .0028 seconds, which includes the preprocessing time of 0.0005 seconds. At this current performance of the model, the utilization and test of the app for clinical trials of children with ASD can be used for evaluation and effectiveness by users. The next chapter discusses the development of the iOS application.

VII. iOS APPLICATION

In this chapter, the iOS app development will be discussed using the final DCNN trained model that has been discussed in the previous chapter. This chapter will be divided into several sections including the app idea and documentation used to develop the app, platform and the programming language used, the machine learning framework, and the computer vision framework for Apple iOS app development and finally the app results.

App Idea and Documentation

The app was designed to be a very straightforward app that executes in real-time and approximating the DCNN model latency of 0.0028 seconds. The app was designed considering it should work in real-time, classifying the facial expression, and showing the corresponding emoticon at minimal execution time. The app idea can again be referred to in Figure 1. The design idea had three steps.

- Once the app is opened, it will directly open the camera and look for human faces.
- The second step is the face detection algorithm. If the algorithm detects a human face, it will go to the next step and feed this face image to the pre-trained DCNN model. If no face is detected, it will show on screen – "No face detected."
- Once a face image is fed into the DCNN model, it will classify the expression of the face and associate it with the correct emoticon for that class. Finally, this emoticon will be shown on the screen.

Apple developer documentation [57] has been extensively used to develop the app. This document contains all the information necessary to develop an iOS app for MacOS, iOS, watchOS, tvOS and beyond, including API references, articles and sample code.

This documentation is divided into several categories including app frameworks, graphics and games, app services, media, web, developer tools, system, etc. The key frameworks used in this thesis for the app development were vision and CoreML which will be discussed later. Xcode was utilized as integrated development environment (IDE) for the development of the app. The app will execute and be compatible for iOS version 11 and beyond.

Xcode is an IDE for macOS containing a suite of software development tools developed by Apple for developing software for macOS, iOS, iPadOS, watchOS, and tvOS. The latest version, Xcode11, was used in this thesis. Using Xcode, the app was developed for universal platforms on iPads and iPhones. It also provides iPhone and iPad simulators, which makes it easy to debug and optimize the app before installing it into a mobile device. A preview of the Xcode is shown below in Figure 46.

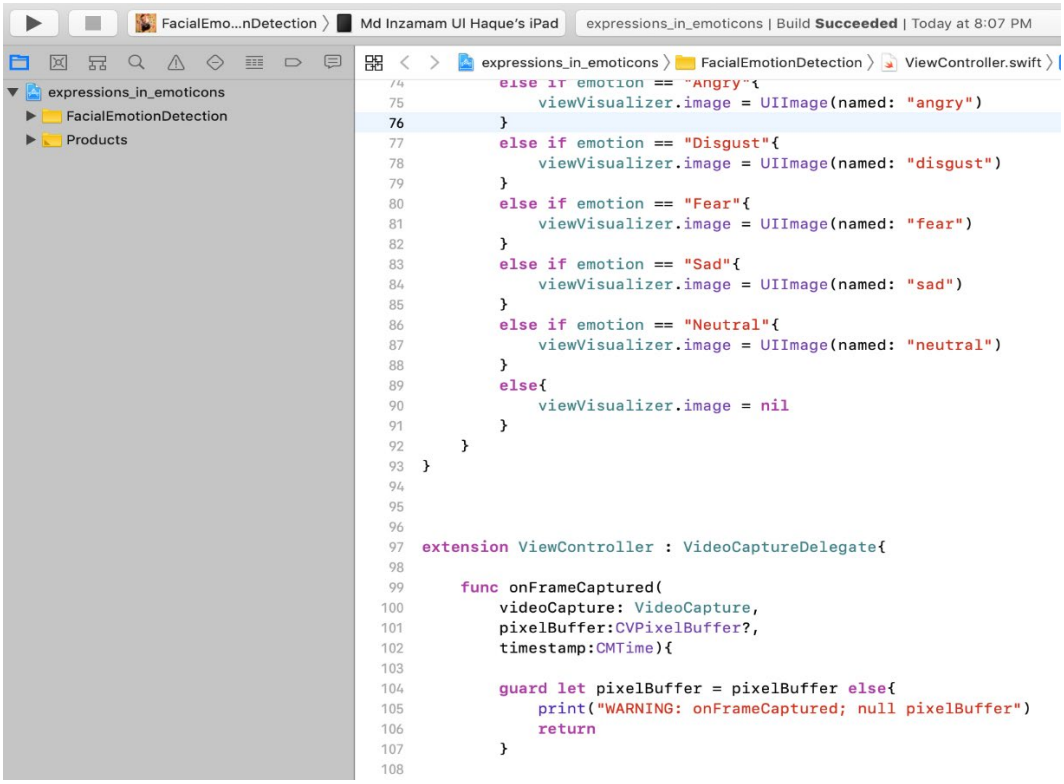

Figure 46. A preview of Xcode IDE

The Swift programming language was used to develop the app. Swift is a useful language to write software, whether it is for phones, desktops, servers, or anything else that runs code. It is a safe, fast, and interactive programming language that combines the best in modern language thinking with wisdom from the broader Apple engineering culture and the diverse contributions from its open-source community. The compiler is optimized for performance, and the language is optimized for development, without compromising on either. Swift code is compiled and optimized to get the most out of modern hardware. The syntax and standard library have been designed based on the guiding principle of logical order to write code and provide execution performance. For this app to work, image processing and computer vision were needed to deal with and the framework that does this for iOS development is known as Vision for iOS. Vision framework is discussed in the next section.

Vision for iOS

The second step for the app is to detect human faces, and this face detection is done by Apple algorithms, which are a part of the Vision framework [58]. Though the algorithm from Apple was used for face detection, some changes were made to the algorithm in order to correctly specify and detect the face contours for this specific case of facial expression detection.

The vision framework applies computer vision algorithms to perform a variety of tasks on input images and video including face and face landmark detection, text detection, barcode recognition, image registration, general feature tracking, and other image-based applications etc. An object detection algorithm from this framework, which deals with real-time face detection, was used in this thesis for the app development.

Although the DCNN model can recognize facial expressions in different lighting condition, this face detection algorithm cannot detect faces in the dark. As a result, the full capacity of the DCNN model cannot be displayed in the app when dealing with dark conditions. Future work of this thesis can reach to develop a detection algorithm that can distinguish the human faces in different lighting conditions.

CoreML for iOS

The final step for the app development was to integrate the pre-trained DCNN model to classify and label the seven different facial expressions with the appropriate emoticon for each class. In order to import the pre-trained model to the iOS app, the CoreML framework [59] has been used. CoreML provides converters that can change Keras models to a CoreML model. The inference CoreML model can be used to make predictions in the app with the integration of the CoreML framework shown in Figure 47. Core ML is the foundation for domain-specific frameworks and functionality and supports Vision for image analysis, Natural Language for natural language processing, and Gameplay Kit for evaluating learned decision trees.

Figure 47. Steps for implementing a Machine Learning model into an iOS app

CoreML allows making a network API call by sending the data and then waiting for a response. The communication handshaking can be critical for applications such as video processing of successive frames from the on-device camera. Core ML is optimized

for on-device performance, which minimizes its memory footprint and power consumption. Running strictly on the device ensures the privacy of user data and guarantees that the app remains functional and responsive when a network connection is unavailable.

Results of the app

The working app and its preview is shown below in Figure 48. These screenshots are taken from an iPad running the latest iOS version 12.3. In figure 48, from the left, three expressions have been detected correctly and the correct emoticons for the expressions have been showed by the app. These three expressions are respectively neutral, happy and surprise. The app has been tested for other expressions and it works fine. The app can now classify the seven different human facial expressions and show them with the correct emoticon on the device's screen. The last two pictures on the right are to show the ability of the app to correctly classify expressions from side views.

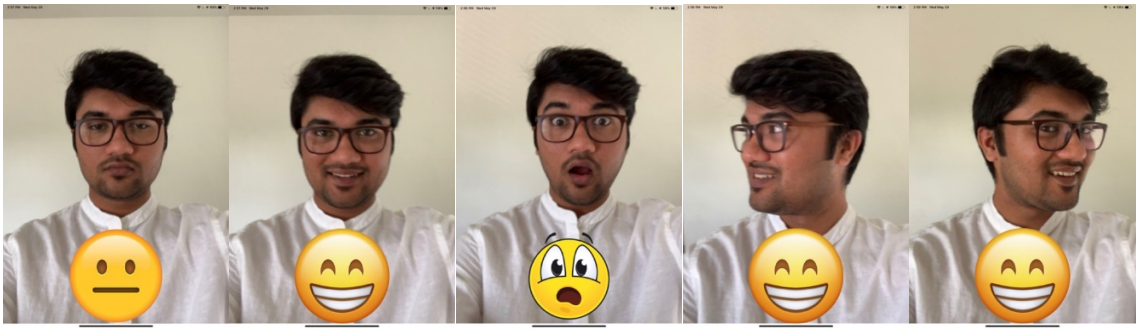

Figure 48. Screenshots of the working iOS app

The app was tested with different subjects and provides high accuracy of detected facial expressions. The latency to correctly detect expressions and show the emoticon is 0.0026 seconds. So, now an app has been developed to utilize and test the DCNN model for clinical trials for children with ASD and effectiveness of the app will be determined by the users and evaluators. Also, it will also be known from the test trials if there any

improvement is needed for the app to have better impact to the main goal of the app – teach children with ASD to correctly recognize seven universal human facial expressions.

VIII. CONCLUSION

In this thesis, the goal was to design a deep convolutional neural network for facial expression recognition that can help autistic children to recognize emotions. The approach of detecting facial expression was through a design and development of a Deep Convolutional Neural Network (DCNN) capable of predicting human facial expressions. The DCNN model was also tested through a developed iOS app for mobile device settings to emulate clinical utilization. The DCNN consisted of three CONV layers stacked on top of each other with the number of filters doubling in each block. It was important that the CNN be:

- Deep enough to obtain high accuracy, and
- Be small enough to run in real-time on the CPU.

The DCNN model was first trained on the FER2013 dataset, part of the Kaggle Emotion and Facial Expression Recognition challenge. The FER dataset was utilized as the trial image dataset and was modified to versions containing images of different lighting contrast conditions. From part of the initial results obtained, the best test accuracy of 63.11% was achieved without overfitting the model marking the result satisfactory. Training the same model, it was seen that the datasets with bright-contrast images yields to a better test accuracy than datasets with dark-contrast images. Then experimentation with datasets containing images from different angles such as top view, bottom view and side views were considered. It would enable this effort to have a model which can recognize human facial expression from any angle in any lighting condition in any environment.

To accomplish this goal, the same DCNN model was trained and tested using KDEF dataset. KDEF dataset contains images of facial expressions from five different angles and it was the best dataset to train the model for recognizing facial expressions from different viewpoints. It was found that further accuracy could likely be obtained by performing image preprocessing being more aggressive with data augmentation, hyperparameter tuning and adding in regularization. Histogram equalization was used as a preprocessing stage before feeding the images into the DCNN model which boosted the accuracy of the model. SGD optimizer with Nesterov acceleration was used in the training process. Moreover, different hyperparameters of the model such as learning rate, batch size, number of epochs, dropout values etc. were tuned and optimized to better improve the accuracy and performance of the model. Finally, the test accuracy of 86.44% was achieved which out-performed the performance metrics utilizing the frontal images from the FER2013 dataset and the leaderboard of the Kaggle Emotion and Facial Expression Recognition challenge. The results showed the necessary training analysis of the DCNN model utilizing sideview angled images and to foresee the improvements necessary for the overall facial expression application. Then using this same model and through optimizing the network, the model accuracy achieved was above 75% for all types of the different lighting conditions which was a huge improvement from earlier results with the FER dataset.

Finally, the pretrained DCNN model was implemented into an iOS app so that the app can be used as a byproduct of the model to use it in clinical trials for the children with ASD. CoreML and Vision frameworks have been used for the iOS app development with Xcode as the IDE and Swift as the programming language. The operation of the app

is to open the camera, detect human faces, correctly classify the expressions, associate the expression with the correct emoticon and show the emoticon on the device screen. The app can recognize and classify facial expressions in real time as the pretrained DCNN model was imported and implemented in the app. The app was tested against different subjects and it has been found that it is able to correctly classify facial expressions and display it with the correct associated emoticon for each class. Now, the goal for the app is to be used by speech-language pathologists as a technological tool when working with children with ASD. In conjunction with tele practice, the app can be used to teach children with ASD to recognize and identify facial expressions while receiving therapy in real-time to practice social skills during everyday social interaction.

IX. FUTURE WORK

A face detection algorithm can be developed to compliment the DCNN model so that it can detect faces from different viewpoints and in different lighting conditions before feeding the detected images to DCNN model. The DCNN model is able to deal with different viewpoints and lighting conditions; but for the app to be able to work in different conditions, a face detection algorithm with similar extensions is a must. As the final model for this thesis was trained with upward and downward pictures containing only happy and sad expressions, it lacks other emotions as well as accuracy in this domain. A dataset can be developed by collecting facial expression images from different sources or by taking pictures from an upward and downward direction for seven universal expressions.

Further work will be required for the app as the effectiveness of the model is tested in the clinical setting with children with Autism. The DCNN model can be improved with the inclusion of a microphone because children with ASD can't identify sarcasm. In that case, speech processing can be done along with image processing in the DCNN algorithm. Also, a simple game can be integrated to the app of picking the right emoticon for the right expression when the app detects any face. This can also be a test to see whether the children with Autism are learning from the game or not.

In the clinical trials, if the app becomes successful, then children with ASD will not have to frequently see a speech-language pathologist in person with this application. The mobility of the app will allow users for a quick and easy access to speech-language therapy using tele practice. In fact, it allows children with ASD to practice receiving and participating in speech-language therapy in real-time. Once the app is in production, it

will be free and opened-sourced so others may work to improve the model. The application will be available to everyone through the iOS app store, and the code through a repository. This will allow the audience availability of the application and get crowd-sourced feedback of its effectiveness.

REFERENCES

[1] T. A. Rashid, "Convolutional Neural Networks based Method for Improving Facial Expression Recognition," in *Intelligent Systems Technologies and Applications 2016*, vol. 530, J. M. Corchado Rodriguez, S. Mitra, S. M. Thampi, and E.-S. El-Alfy, Eds. Cham: Springer International Publishing, 2016, pp. 73–84.

[2] "module4notebookhumaninteractionandcommunication.pdf."

[3] P. Ekman and W. V. Friesen, "Constants across cultures in the face and emotion.," *J. Pers. Soc. Psychol.*, vol. 17, no. 2, pp. 124–129, 1971.

[4] "Nonverbal communication," *Wikipedia*. 09-Aug-2018.

[5] J. M. Leppänen and J. K. Hietanen, "Emotion recognition and social adjustment in school-aged girls and boys," *Scand. J. Psychol.*, vol. 42, no. 5, pp. 429–435, Dec. 2001.

[6] K. Kang, L. Anthoney, and P. Mitchell, "Seven- to 11-Year-Olds' Developing Ability to Recognize Natural Facial Expressions of Basic Emotions," *Perception*, vol. 46, no. 9, pp. 1077–1089, Sep. 2017.

[7] A. A. Marsh, M. N. Kozak, and N. Ambady, "Accurate identification of fear facial expressions predicts prosocial behavior," *Emot. Wash. DC*, vol. 7, no. 2, pp. 239–251, May 2007.

[8] S. Goodfellow and S. Nowicki, "Social adjustment, academic adjustment, and the ability to identify emotion in facial expressions of 7-year-old children," *J. Genet. Psychol.*, vol. 170, no. 3, pp. 234–243, Sep. 2009.

[9] G. Chronaki, M. Garner, J. A. Hadwin, M. J. J. Thompson, C. Y. Chin, and E. J. S. Sonuga-Barke, "Emotion-recognition abilities and behavior problem dimensions in preschoolers: evidence for a specific role for childhood hyperactivity," *Child Neuropsychol. J. Norm. Abnorm. Dev. Child. Adolesc.*, vol. 21, no. 1, pp. 25–40, 2015.

[10] S. Sette, E. Baumgartner, F. Laghi, and R. J. Coplan, "The role of emotion knowledge in the links between shyness and children's socio-emotional functioning at preschool," *Br. J. Dev. Psychol.*, vol. 34, no. 4, pp. 471–488, 2016.

[11] B. Gepner, C. Deruelle, and S. Grynfeltt, "Motion and Emotion: A Novel Approach to the Study of Face Processing by Young Autistic Children," p. 11.

[12] E. Bal, E. Harden, D. Lamb, A. V. Van Hecke, J. W. Denver, and S. W. Porges, "Emotion recognition in children with autism spectrum disorders: relations to eye gaze and autonomic state," *J. Autism Dev. Disord.*, vol. 40, no. 3, pp. 358–370, Mar. 2010.

[13] S. J. Weeks and R. P. Hobson, "The Salience of Facial Expression for Autistic Children," *J. Child Psychol. Psychiatry*, vol. 28, no. 1, pp. 137–152.

[14] R. P. Hobson, "The Autistic Child's Appraisal of Expressions of Emotion: A Further Study," *J. Child Psychol. Psychiatry*, vol. 27, no. 5, pp. 671–680.

[15] "npj Digital Medicine," *npj Digital Medicine*. [Online]. Available: https://www.nature.com/npjdigitalmed/. [Accessed: 29-May-2019].

[16] A. McManus, "20 Awesome Emotion-Enabled Campaign Ideas." [Online]. Available: https://blog.affectiva.com/20-awesome-emotion-enabled-campaign-ideas. [Accessed: 29-May-2019].

[17] "Challenges in Representation Learning: Facial Expression Recognition Challenge." [Online]. Available: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge. [Accessed: 13-Aug-2018].

[18] N. Pinto, *fer2013: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data*. 2018.

[19] X.-P. Huynh, T.-D. Tran, and Y.-G. Kim, "Convolutional Neural Network Models for Facial Expression Recognition Using BU-3DFE Database," in *Information Science and Applications (ICISA) 2016*, vol. 376, K. J. Kim and N. Joukov, Eds. Singapore: Springer Singapore, 2016, pp. 441–450.

[20] S. Baron-Cohen, O. Golan, and E. Ashwin, "Can emotion recognition be taught to children with autism spectrum conditions?," *Philos. Trans. R. Soc. B Biol. Sci.*, vol. 364, no. 1535, pp. 3567–3574, Dec. 2009.

[21] "Emotional development in kids with autism | Raising Children Network." [Online]. Available: http://raisingchildren.net.au/articles/autism_spectrum_disorder_emotional_development.html. [Accessed: 23-Sep-2018].

[22] K. G. Smitha and A. P. Vinod, "Facial emotion recognition system for autistic children: a feasible study based on FPGA implementation," *Med. Biol. Eng. Comput.*, vol. 53, no. 11, pp. 1221–1229, Nov. 2015.

[23] E. Loth, L. Garrido, J. Ahmad, E. Watson, A. Duff, and B. Duchaine, "Facial expression recognition as a candidate marker for autism spectrum disorder: how frequent and severe are deficits?," *Mol. Autism*, vol. 9, no. 1, p. 7, Jan. 2018.

[24] J. Boucher and V. Lewis, "Unfamiliar Face Recognition in Relatively Able Autistic Children," *J. Child Psychol. Psychiatry*, vol. 33, no. 5, pp. 843–859.

[25] C. Pramerdorfer and M. Kampel, "Facial Expression Recognition using Convolutional Neural Networks: State of the Art," *ArXiv161202903 Cs*, Dec. 2016.

[26] A. Savoiu and J. Wong, "Recognizing Facial Expressions Using Deep Learning," p. 7.

[27] V. Mavani, S. Raman, and K. P. Miyapuram, "Facial Expression Recognition Using Visual Saliency and Deep Learning," in *2017 IEEE International Conference on Computer Vision Workshop (ICCVW)*, 2017, pp. 2783–2788.

[28] C. Huang, "Combining convolutional neural networks for emotion recognition," in *2017 IEEE MIT Undergraduate Research Technology Conference (URTC)*, 2017, pp. 1–4.

[29] L. Nwosu, H. Wang, J. Lu, I. Unwala, X. Yang, and T. Zhang, "Deep Convolutional Neural Network for Facial Expression Recognition Using Facial Parts," in *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, Orlando, FL, 2017, pp. 1318–1321.

[30] A. Rosebrock, *Deep Learning for Computer Vision with Python*, 1.3.0. PyImageSearch.com, 2018.

[31] A. L. Samuel, "Some studies in machine learning using the game of Checkers," *Ibm J. Res. Dev.*, pp. 71–105, 1959.

[32] U. India, "Difference between Machine Learning, Deep Learning and Artificial Intelligence," *Medium*, 29-Mar-2018.

[33] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: a modern approach*, 3rd ed. Upper Saddle River: Prentice Hall, 2010.

[34] M. Mohri, A. Rostamizadeh, and A. S. Talwalkar, "Foundations of Machine Learning," in *Adaptive computation and machine learning*, 2012.

[35] import.io, *Andrew Ng, Chief Scientist at Baidu*.

[36] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," *Towards Data Science*, 15-Dec-2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: 02-Jun-2019].

[37] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[39] M. A. Nielsen, "Neural Networks and Deep Learning," 2015.

[40] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, Jun. 2000.

[41] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *ArXiv151107289 Cs*, Nov. 2015.

[42] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: 06-Jun-2019].

[43] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *ArXiv150203167 Cs*, Feb. 2015.

[44] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *ArXiv14091556 Cs*, Sep. 2014.

[45] S. Sudhakar, “Histogram Equalization,” *Towards Data Science*, 10-Jul-2017. [Online]. Available: https://towardsdatascience.com/histogram-equalization-5d1013626e64. [Accessed: 09-Jun-2019].

[46] S. U. S. E. Laboratories, B. Widrow, U. S. O. of N. Research, U. S. A. S. Corps, U. S. A. Force, and U. S. Navy, *Adaptive “adaline” neuron using chemical “memistors.”*. 1960.

[47] S. Ruder, “An overview of gradient descent optimization algorithms,” *ArXiv160904747 Cs*, Sep. 2016.

[48] J. Duchi, E. Hazan, and Y. Singer, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,” *J Mach Learn Res*, vol. 12, pp. 2121–2159, Jul. 2011.

[49] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” *ArXiv12125701 Cs*, Dec. 2012.

[50] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *ArXiv14126980 Cs*, Dec. 2014.

[51] Y. E. NESTEROV, “A method for solving the convex programming problem with convergence rate O(1/k^2),” *Dokl Akad Nauk SSSR*, vol. 269, pp. 543–547, 1983.

[52] “lecture_slides_lec6.pdf.” .

[53] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient BackProp,” in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, London, UK, UK, 1998, pp. 9–50.

[54] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[55] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” *ArXiv150201852 Cs*, Feb. 2015.

[56] leap, “LEAP - High Performance Computing Cluster: Division of Information Technology : Texas State University,” 08-May-2018. [Online]. Available: http://doit.txstate.edu/rc/leap.html. [Accessed: 31-Jul-2018].

[57] “Apple Developer Documentation.” [Online]. Available: https://developer.apple.com/documentation/. [Accessed: 13-Jun-2019].

[58] “Vision | Apple Developer Documentation.” [Online]. Available: https://developer.apple.com/documentation/vision. [Accessed: 13-Jun-2019].

[59] “Core ML | Apple Developer Documentation.” [Online]. Available: https://developer.apple.com/documentation/coreml. [Accessed: 14-Jun-2019].