INVESTIGATION OF HIERARCHICAL DEEP NEURAL NETWORK STRUCTURE FOR FACIAL EXPRESSION RECOGNITION

by

DODI MOTEMBE

submitted in accordance with the requirements for
the degree of

MAGISTER TECHNOLOGIAE

In the subject

ELECTRICAL ENGINEERING

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF. ZENGHUI WANG

January 2020

DECLARATION

Name: DODI MOTEMBE

Student number: 50685740

Degree: MAGISTER TECHNOLOGIAE

The submission for examination title is shown below:

INVESTIGATION OF HIERARCHICAL DEEP NEURAL NETWORK STRUCTURE FOR FACIAL EXPRESSION RECOGNITION

I declare that *Investigation of hierarchical deep neural network* structure is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

I further declare that I submitted the dissertation to originality checking software and that it falls within the accepted requirements for originality.

I further declare that I have not previously submitted this work, or part of it, for examination at Unisa for another qualification or at any other higher education institution.

18 - 01- 2020

SIGNATURE DATE

ACKNOWLEGEMENTS

I would like to thank my wife Antoinette Motembe for her support and patience. Secondly, I would like to express my gratitude to my children, Shamah Motembe, Angela Motembe and Nissi Motembe as well as my younger sister Sandra Motembe for their love and support during the accomplishment of this work. Lastly, I am grateful to my supervisor Professor Wang for his support and patience during the phase of this work. I am humbled to have had him as my supervisor during the period of this dissertation.

ABSTRACT

Facial expression recognition (FER) is still a challenging concept, and machines struggle to comprehend effectively the dynamic shifts in facial expressions of human emotions. The existing systems, which have proven to be effective, consist of deeper network structures that need powerful and expensive hardware. The deeper the network is, the longer the training and the testing. Many systems use expensive GPUs to make the process faster. To remedy the above challenges while maintaining the main goal of improving the accuracy rate of the recognition, we create a generic hierarchical structure with variable settings. This generic structure has a hierarchy of three convolutional blocks, two dropout blocks and one fully connected block. From this generic structure we derived four different network structures to be investigated according to their performances. From each network structure case, we again derived six network structures in relation to the variable parameters. The variable parameters under analysis are the size of the filters of the convolutional maps and the max-pooling as well as the number of convolutional maps. In total, we have 24 network structures to investigate, and six network structures per case. After simulations, the results achieved after many repeated experiments showed in the group of case 1; case 1a emerged as the top performer of that group, and case 2a, case 3c and case 4c outperformed others in their respective groups. The comparison of the winners of the 4 groups indicates that case 2a is the optimal structure with optimal parameters; case 2a network structure outperformed other group winners. Considerations were done when choosing the best network structure, considerations were; minimum accuracy, average accuracy and maximum accuracy after 15 times of repeated training and analysis of results. All 24 proposed network structures were tested using two of the most used FER datasets, the CK+ and the JAFFE. After repeated simulations the results demonstrate that our inexpensive optimal network architecture achieved 98.11 % accuracy using the CK+ dataset. We also tested our optimal network architecture with the JAFFE dataset, the experimental results show 84.38 % by using just a standard CPU and easier procedures. We also compared the four group winners with other existing FER models performances recorded recently in two studies. These FER models used the same two datasets, the CK+ and the JAFFE. Three of our four group winners (case 1a, case 2a and case 4c) recorded only 1.22 % less than the accuracy of the top performer model when using the CK+ dataset, and two of our network structures, case 2a and case 3c came in third, beating other models when using the JAFFE dataset.

Key terms - Facial Expression Recognition (FER); Deep Learning; Convolutional Neural Network (CNN); Deep Convolutional Neural Network (DCNN); Artificial Intelligence; Hierarchical Deep Neural Network Structure; Face Detection; Facial Feature Extraction; Central Processing Unit (CPU); Graphics Processing Unit (GPU).

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AAM	Active Appearance Model
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASM	Active Shape Model
AU	Action Unit
AUC	Area under the ROC-curve
BDA	Bayes Discriminant Analysis
CK+	The Extended Cohn-Kanade Database
CLBP	Completed Local Binary Pattern
CLQP	Completed Local Quantized Pattern
CMFD	Component-based Multiple Feature Descriptor
CNN	Convolutional Neural Network
CSFD	Component-based Single Feature Descriptor
COPE	Infant Classification of Pain Expressions Database
CPU	Central Processing Unit
CRF	Conditional Random Field
DBN	Dynamic Bayesian Network
DCNN	Deep Convolutional Neural Network
DisCSFD	Discriminative Component-based Single Feature Descriptor
DisSFD	Discriminative Sparse Feature Descriptor
DL	Deep Learning
DLNN	Deep Learning Neural Network
DNPE	Discriminative Neighbor Preserving Embedding
DoM	Difference of Magnitude
DoO	Difference of Orientation

DoS	Difference of Sign
DRML	Deep Region and Multi-label Learning
FACS	Facial Action Coding System
FLs	Facial Landmarks
FER	Facial Expression Recognition
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
HCI	Human-Computer Interaction
HDNN	Hierarchical Deep Neural Network
HMMs	Hidden Markov Models
HOG	Histogram of Oriented Gradients
ICA	Independent Component Analysis
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
KLT	Kanade-Lucas-Tomasi
LBP	Local Binary Pattern
LDA	Linear Discriminant Analysis
LGBP	Local Gabor Binary Pattern
LMMBP	Local Monogenic Magnitude Binary Pattern
LMRBP	Local Monogenic Real Binary Pattern
LMIBP	Local Monogenic Imaginary Binary Pattern
LPP	Local Preserving Projection
LPQ	Local Phase Quantization
LQP	Local Quantized Pattern
LRCN	Long-term Recurrent Convolutional Network
LSTM	Long Short-Term Memory
LTP	Local Ternary Pattern
LUT	Look-Up Table

LXP	Local XOR operator
MCF	Multi-Classifier Fusion
MKL	Multiple Kernel Learning
MP	McCulloch-Pitts
MVDNPE	Multi-view Discriminative Neighbor Preserving Embedding.
NIR	Near-Infrared
NMF	Non-negative Matrix Factorization
NPE	Neighbor Preserving Embedding
OD	Occlusion Detection
PCA	Principle Component Analysis
PQDC	Phase-Quadrant Demodulation Coding
RBF	Radial Basis Function
RNN	Recurrent Neural Networks
SFD	Sparse Feature Descriptor
SIFT	Scale-Invariant Feature Transform
SRC	Sparse Representation Classifier
STLMBP	SpatioTemporal Local Monogenic Binary Pattern
STLMIBP	SpatioTemporal Local Monogenic Imaginary Binary Pattern
STLMMBP	SpatioTemporal Local Monogenic Magnitude Binary Pattern
STLMRBP	SpatioTemporal Local Monogenic Real Binary Pattern
SVM	Support Vector Machine
TOP	Three Orthogonal Planes
VIS	Visible light
VLPQ	Volume Local Phase Quantization
WL	Weight Learning

CHAPTER 1 INTRODUCTION

1.1 BACKGROUD

The genesis of Artificial Neural Network (ANN) which is a McCulloch-Pitts (MP) structure, and carries the two creators' names, started more than seven decades ago. The MP structure is an ensemble of neurons for the activation of brain functions. This structure was first presented by McCulloch and Pitts in 1943. ANN concepts were derived from the MP structure. After 41 years, another structure was designed, known as the Hopfield Neural Network, bearing the name of its founder John Hopfield. This structure is a hybrid of storage and memory arrangements to enable the activation of the memory, depending on the selection of the mobilizer category chosen. The categories are continuous and discrete. During that era, there were not many advanced researches, therefore the structure was not the subject of attention as compare to what it could have attracted [1][2][3].

The real progress for deep learning structure originated in 1989 when the grasping of the concepts of back-propagation algorithm became sound, even though this algorithm was introduced three decades earlier. The application of back-propagation algorithm in the network structure started the revolution of deep learning networks. This algorithm gives the networks an automatic learning capacity of features and it distinguishes deep learning networks from other intelligent systems [4][5][6]. This is the reason why Deep Learning Neural Network (DLNN) is popular in the research world. DLNN is able to provide solutions to many sectors (medical, education, military, economy, science and if not all spheres of our lives) and in addition, DLNN provides alternatives to make our lives easier.

Facial Expression Recognition (FER) has been and is being used in several spheres of our lives with huge benefits to society. In the security field, it has been extensively used and recently, many researchers are exploring new avenues for further improvement. From detecting diseases to being utilized for various needs in the medical sector and also for the design of robotics, FER is having a massive impact in our lives and has many valuable proficiencies to better the world[7][8][9][10][11].

There are seven main facial expressions to describe the human emotions [9][12][13][14][11]. These facial expressions are able to provides us with information about the state of emotions that humans are in at that specific moment of observation. Therefore, to have technologies which are capable of detecting each individual expression with accuracy is important. Facial expression recognition studies are gaining momentum recently and many researchers are trying to find solutions and share their expertise, so that quality and effective systems for facial expression recognition can be designed and improved.

1.2 PROBLEM STATEMENT

Facial expression recognition contributes to various societal needs. In South Africa, crime is on the rise and it has psychological impacts on the people. There is a need for a good technology to assist police officers to detect suspect's hidden intentions during interrogations. The same technology is also needed in the psychiatry field to enable psychiatrists to diagnose any existence of a mental illness in a patient. Attempts to recognize facial expressions in

psychology have been done using conventional models in the past, but their accuracy was poor and they did require many procedures. During the last decade, FER based structures have been developed with acceptable accuracy results but the current accuracy can be improved and there is another challenge caused by algorithms that have become deeper, the process is slow due to the training and the testing of data which takes more time. Many systems use expensive Graphics Processing Units (GPUs) to make the process faster. This research aims to find an optimal network structure with optimal parameters by investigating different deep network structures as well as various parameters for FER that can improve accuracy. Secondly, this work emphasizes on the hardware cost by using a standard Central Processing Unit (CPU) which is affordable, the optimal network structure with optimal parameters is expected to give good accuracy using an affordable hardware and to be trained and tested within expected computational time.

1.3 RESEARCH OBJECTIVES

The aim of this research is to investigate different network architectures with variable parameters in order to find optimal network architecture with optimal parameters which can improve accuracy. The existing architectures accuracy is not good enough and requires expensive hardware to make training faster. We will use a standard CPU which is affordable to find optimal network architecture in order to get good performance within an appropriate computational time.

1.4 DELIMITATIONS OF THE STUDY

- Lack or not enough diversity in the database because of fewer images per expression.
- Not enough similarities with images in the real world in the dataset.
- The computational time for training will be according to the settings of less than 120 minutes.
- The purpose is to try to find an optimal architecture which can give better accuracy using a standard laptop (a laptop with these minimum specifications: CPU Intel(R) Core(TM) 2 with a clock speed of 2.40GHz and a RAM of 4 GB) according to the set time. Currently, many existing FER models are using GPUs in order to increase the speed of training time.
- Currently, there is a challenge with web searches, the returned images in certain cases are not matching the keyword inserted.

1.5 METHODOLOGY

- Literature review:
  I did an extensive literature study on the different algorithms used for FER analysis. I used two research databases (IEEE Xplore and Elsevier) to read as many as possible publications on FER.
- Software:
  I used the python language for the implementation of the research with the deep learning frameworks (tensorflow & theano).
- Results:
  The results from training and testing of the data during simulations were collected for analysis.
- Analysis:
  Analyzed and compared the obtained results with other models used on FER using the same datasets to see if the accuracy of our proposed models has improved or has achieved better results than other tested FER based methods.
- Final conclusion:
  I concluded the final details according to the comparison and study of the results of my proposed models and other tested FER models. Also, gave directions for future work based on my research work.

1.6 FACIAL EXPRESSION RECOGNITION DATASETS

I implement my simulations by using two datasets described in [15]. These two datasets are used by many researchers to test various algorithms related to facial expression recognition. Therefore, they are suitable for evaluation against the latest technology. I perform my investigation by testing my proposed HDNN structures on these datasets.

1.7 LAYOUT OF THE DISSERTATION

Chapter 1 Introduction:

Chapter 1 presents the current problems for deep learning neural networks structures, objectives, delimitations of the study, methodology, facial expression recognition datasets and layout of the dissertation.

Chapter 2 Literature Review:

Chapter 2 studies facial expression recognition technique, from face detection to classification. Secondly, the chapter investigates the deep learning neural networks and in depth studies on convolutional neural network structures are accomplished. Finally, the existing Convolutional Neural Network (CNN) models are analysed.

Chapter 3 Hierarchical Deep Neural Network Structure:

Chapter 3 details the different proposed HDNN structures for facial expressions recognition. Additionally, the chapter presents the proposed HDNN structure case studies with variable parameters. Finally, the chapter explores the details of the proposed HDNN structures according to each case study based on the value settings.

Chapter 4 Experimental setup:

Chapter 4 explains the two FER datasets which will be used to test our proposed HDNN structures, the structure design, the training process and how the assessment of our investigation will be conducted.

Chapter 5 Simulation results and analysis:

Chapter 4 details the simulation outcomes and the analysis of all case studies of all the different proposed HDNN structures. The chapter compares the performances of the proposed HDNN structures. Finally, the chapter compares our optimal HDNN structure with the existing FER models.

Chapter 6 Conclusion and future work:

Chapter 6 the advantages of our optimal HDNN structure with optimal parameters are discussed and orientations are given for future studies.

CHAPTER 2 LITERATURE REVIEW

This chapter explains in detail the facial expression recognition processes. Secondly, the deep convolutional neural network is described. Key components of the CNN structure are detailed. Lastly, the existing CNN structures are detailed and studied.

2.1 FACIAL EXPRESSION RECOGNITION OVERALL PATH

There are a number of facial expressions for a single person, just as there are different images for the same person as shown in Figure 2.1.

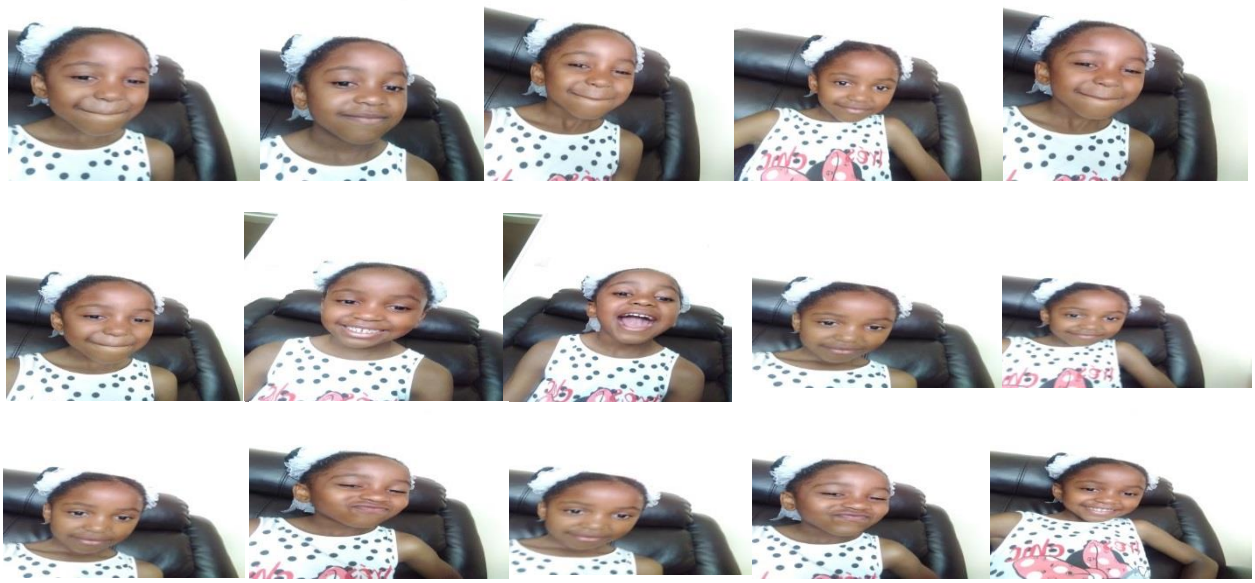

Figure 2.1: Facial expressions of human emotions.

The process of recognition of facial expressions follows a common pattern that is in the order of:

- Face detection
- Facial feature extraction
- Classification

Figure 2.2 displays the overall pattern which FER process follows. We will elaborate each part of the process individually in depth to grasp the concept.

Figure 2.2: Overall path of FER process.

2.1.1 FACE DETECTION

Face detection is the first part of the process and it is vital in order to achieve facial expression recognition. Many techniques have been explored to attain face detection in the past [16]. They are as follows;

- **Face tracking technique:** a specific algorithm that is mixed with a 3D technique to detect images from a video source.
- **Normalization technique:** the two points on the eyes are selected as points 1 and 2; the middle of the mouth is indicated as point 3. All these 3 points are the references of the method. From point 1 to point 2, we have d (fixed distance) which is the first condition of the technique. The second requirement is based on the face measurements: the width and the height of the face are estimated at $2d$ and $3d$ respectively.
- **Surface feature analysis technique:** it operates on the principle where light is used to stimulate the surface and information can be retrieved for analysis. Faces are displayed in triangular meshes.
- **Hybrid Haar–like-feature and skin colour detection technique:** it is based on eliminating the false detections in the process.
- **Registration technique:** placement of the eyes is selected and the face is rotated to match the eyes horizontally. The specific output image is finalised after a number of operations.
- **Cropping technique:** it was applied on the CK database, this method involves the cropping of images from the previous state and the resizing of the distance between the eyes placements.
- **Voila-Jones technique:** the two authors came up with *integral image* which is a fast technique for image operations and processing.

2.1.2 FACIAL FEATURE EXTRACTION

Facial feature extraction acquires the facial features that are distinctive and possess certain stability. Different systems for facial feature extraction are utilized [17];

- **Appearance features system:** it uses an image filter to work on face data to retrieve the modifications of facial exterior. Principal Component Analysis (PCA), Independent Component Analysis (ICA) or Local Binary Pattern has achieved good results.
- **Geometric features system:** consists of a creation of a feature vector that serves as facial geometry and is an ensemble of points of angles, shape and distances.
- **Hybrid appearance and geometric feature system:** when the two techniques are merged, good recognition performance has been achieved.

2.1.3 CLASSIFICATION

The final part of FER system is the stage of identifying the facial expression images and classify them accordingly as "happy" or "angry". The terms detailed below are methods of the classification that is implemented for FER systems [18].

Directed line segment Haudorff distance (dLHD): dLHD is the divergence identified between two lines sets and the difference in the output is measured.
Euclidean distance metric: the approximated distance is obtained after the matrix of normalized and similarity counts.
Minimum Distance Classifier (MDC): the distance measurement is used for classification and is the length from one vector to another.
KNN (k-Nearest Neighbours) algorithm: is a technique where prediction occurs during the phase of training and the allocation of classes is done through the liaison amidst algorithms.
Support Vector Machine (SVM): consists of a formation of a line (hyperlane) that dissociates images into classification.
Radial Basis Function (RBF): the technique incorporates a value allocation to an input by a function and the output is forever an absolute amount.
Hidden Markov Model (HMM): is based on a statistical model using observation of sequences of internal details to classify facial expressions. HMM uses one state per class.
Hidden Conditional Random Fields (HCRF): is an extension of Conditional Random Fields (CRFs) to tacle more complex data. HCRF uses few states per class.
Online Sequential Extreme Learning Machine (OSELM): consists of the first phase to initialize data training and the second phase is sequential learning.
ID3 Decision Tree: is based on set decisions to output efficient decisions for classification.
Classification and Regression Tree (CART): is a method based on the length between vectors.
Learning Vector Quantization (LVQ): is made of two sheets. The first is competitive and has neurons. The second sheet is the output sheets where the selected neuron is deposited.
Multi-Layer Perception (MLP): each knob has a neuron in the 3 layers and utilizes the activation function.

Multi-Layer Feed Forward Neural Network (NFFNN): it is exactly like the above technique except that the back propagation is added to classify images that consist of weights during the training phase for initialization and the prediction of the activation entities.

Bayesian Neural Network: uses graphs with probability calculations for classification.

Convolution Neural Network (CNN): uses neurons formed to classify with less pre-processing. It has input, subsampling, pooling and output phases.

Deep Neural Network (DNN): consist of many hidden layers, the neurons learn from the data to classify images.

2.2 DEEP CONVOLUTIONAL NEURAL NETWORK

CNNs are simple Artificial Neural Networks (ANNs) in a shape of multilayer perceptron (MLP). They have hidden layers which are referred to as convolutional layers. The convolutional layers define a CNN hence the description. CNNs are structured with the ability to grasp certain patterns of the data and to understand their meaning. That ability makes CNNs well suited for image classification. CNNs can be used for other classifications as well, for example, the language classification. The number of hidden layers is the only aspect differentiating CNNs from Deep Convolutional Neural Networks (DCNNs); they are both the same except DCNNs have many convolutional layers [19].

2.2.1 CNN STRUCTURE

CNNs change the input information from the input layer and proceed along the connected layers until the last stage where they give classification results to the output layer. CNN structure can be in various forms and there are different types of CNN structures. All CNN structures share the same characteristics which are given in Figure 2.3.

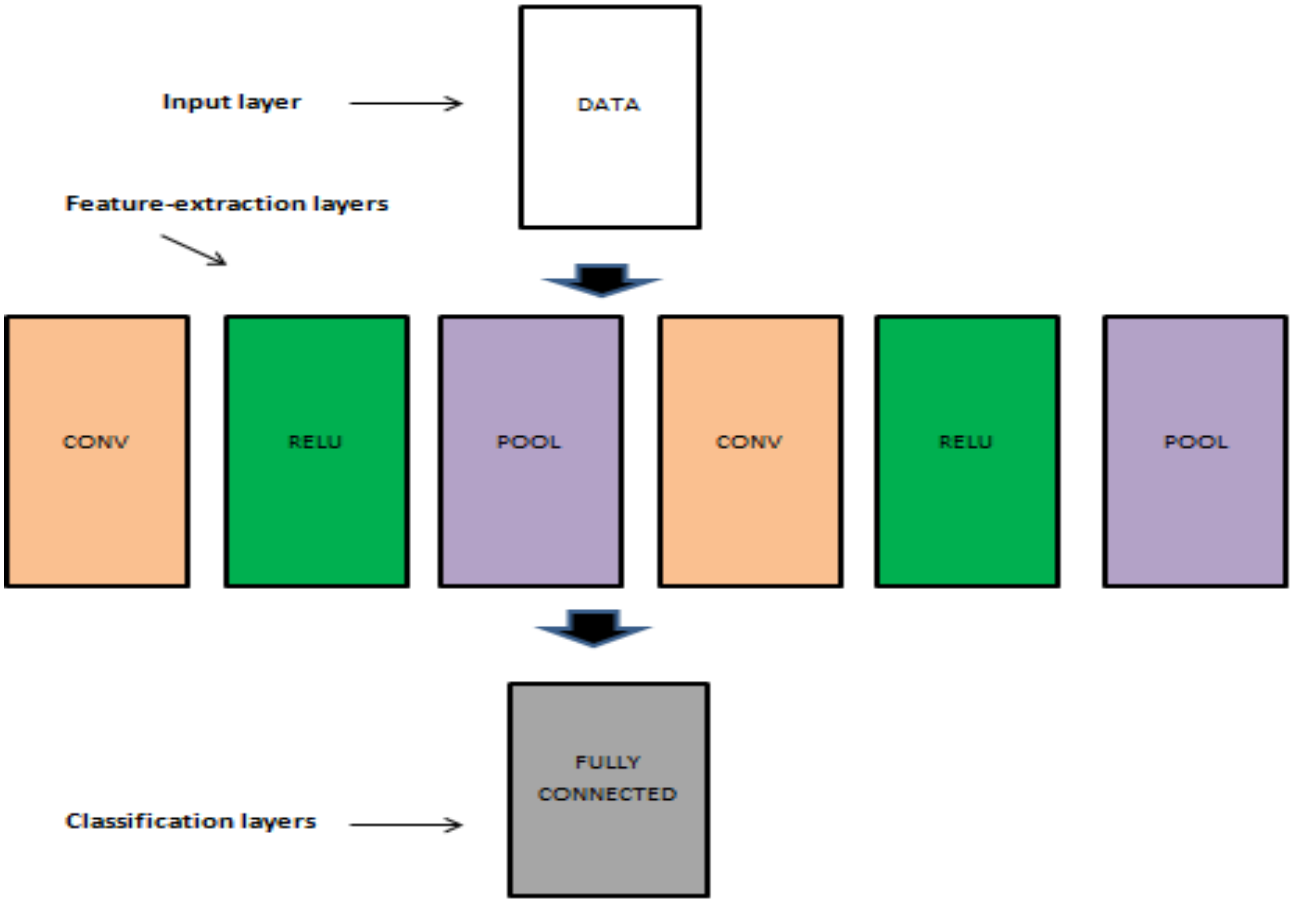

Figure 2.3: An advanced general CNN

We can see from Figure 2.3 above there is a general pattern of three components: the first component is the input layer which receives data. This input data has some settings. That is, it must;

- have a size in the shape of *width* x *height*
- be in three dimensions
- have depth to illustrate the calorific avenues, for example RGB has three avenues

The second component is the feature-extraction layers which have an arrangement characterised by a repetition model: convolution layer (CONV), rectified linear unit (*ReLu*), activation function level and pooling level (*Pool*). The last component is the classification layers where fully connected layers reside. It can be one or several layers. The fully connected layers transform the features into classes and have distinctive elements:

- they are attached to all the neurons housed in the preceding level
- they give an output of the *number of examples* X the *number of classes* [*b* x *N*]
- output is a two dimensional

2.2.1.1 Input layer

All unprocessed data images are deposited and housed in the input layers for network operation. There is clarification regarding the image width and height as well as how many channels will be used. Often, there are 3 routes that represent the RGB values for a particular pixel. Figure 2.4 displays the input layer in 3 dimensions.

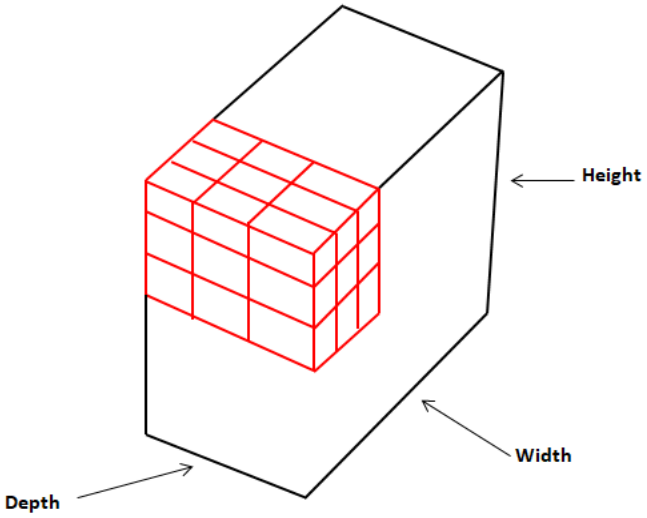

Figure 2.4: Input level 3D

2.2.1.2 Feature-extraction layers

2.2.1.2.1 Convolutional layers

The central building blocks of a CNN structure are the convolutional layers. These layers receive the input data and modify it by using a blot of linked neurons derived from the preceding level. The outcome is similar but smaller with geometric configurations.
A convolution is a simple analytical application which details a rule for merging two data items. It transforms an input by using convolution kernels and outputs a feature map.
Convolutional layers have some main constituents: Filters, Activation maps, Parameter sharing and hyper parameters.

Filters: are part of the structure that has a width and a height. A filter is tiny and is applied after the input volume. It is used in a sliding shape along the width and height of the input volume. The output of a filter results in an activation.
Activation maps: are an output number when a neuron allows data to take route. The mentioning of "activates" simply means the filter allows data to continue along the path from the input volume to the output volume.

Parameter sharing: it is important because it reduces the training time by using a small number of elements to learn during training. CNNs use parameter sharing from invariance to positioning.
Hyperparameters: are the main actors for producing the geometric shape and size that is an output volume. These hyper parameters are: size of filters, output depth, stride and zero-padding.

2.2.1.2.2 Pooling layers

They are often used in between convolutional layers that precede each other. The role of the pooling layers is to scale down the dimensions of data. This reduction process is done repeatedly along the network structure. Another function of the pooling layers is to manage over-fitting which comes from the complexity of the model that has a reflection of nearly closed data that will compromise the resulting predictions. The pooling layers do not depend on other elements for their working.

2.2.1.3 Classifications

Fully connected layers are the constituents of this component. These layers are used to numerically output the results for classification. The arrangement of the output is [1 x 1 x N], N is indicating the classes'number. They possess the linked neurons including those from the preceding level. The parameters of fully connected layers are *weights* and *biases* of the network neurons.

To effectively grasp the concept of a network structure in order for the classification to occur, the following elements need to be put in action [20]:

Score function: this element has the task of mapping the images from the input to scores for classification.
Loss function: this element has the role of determining how close the result of the prediction of the network structure is to the correct value.

Therefore, when building a network structure, it is important to use a connection between the two functions detailed above in order to create a situation whereby the optimization will diminish the loss function in relation to the specifications of the score function.

2.2.1.3.1 Linear classifiers

These classifiers consist of linear blends of predetermined nonlinear basis functions. Linear classifiers seem to be easy to comprehend but in actual fact there is a challenge created by the reality that the majority of data are nonlinearly detachable while these classifiers are linear [20].

2.2.1.3.2 Nonlinear classifiers

The challenge described in 2.2.1.3.1 can be resolved by making use of architectures that can understand nonlinear features. These architectures are neural networks which were conceived

from the idea of neurons in a human brain and they use basis functions like the ones mentioned in 2.2.1.3.1 whereby the ensemble of variables from the input to the ensemble of the variables from the output result in these nonlinear functions which is managed by a vector of modifiable constituents [20].

2.3 FER BASED STRURCTURES

2.3.1 IMAGENET LARGE SCALE VISUAL RECOGNITION CHALLENGE (ILSVRC) CNN STRUCTURE CONTESTANTS

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition where researchers come up with designed algorithms, with the aim of winning the competition and proving that their proposed algorithm was the best and most effective in image classification. The competition is based on a large spectrum of images considering that these models classify 1000 images into their respective categories [21]. Therefore, the models that contest in this challenge must be well built to handle the large database of images and be able to classify the images into their relevant groups. The following are some of the leading challengers on the ILSVRC competition [22].

1. LeNet-5

In 1998 *Lecun et al* designed a model structured with seven layers to classify numbers, and these numbers in the database were black and white. When a network operates on grey images, it does require powerful hardware for quality resolution detection and many convolutional blocks in the network for correct classification. The above enumerated points are some of the limitations of this network structure. This system was valuable to the banks in verifying the legitimacy of clients in respect to their writing on their checks. Therefore, refuting scammers who tried to copy clients writing.

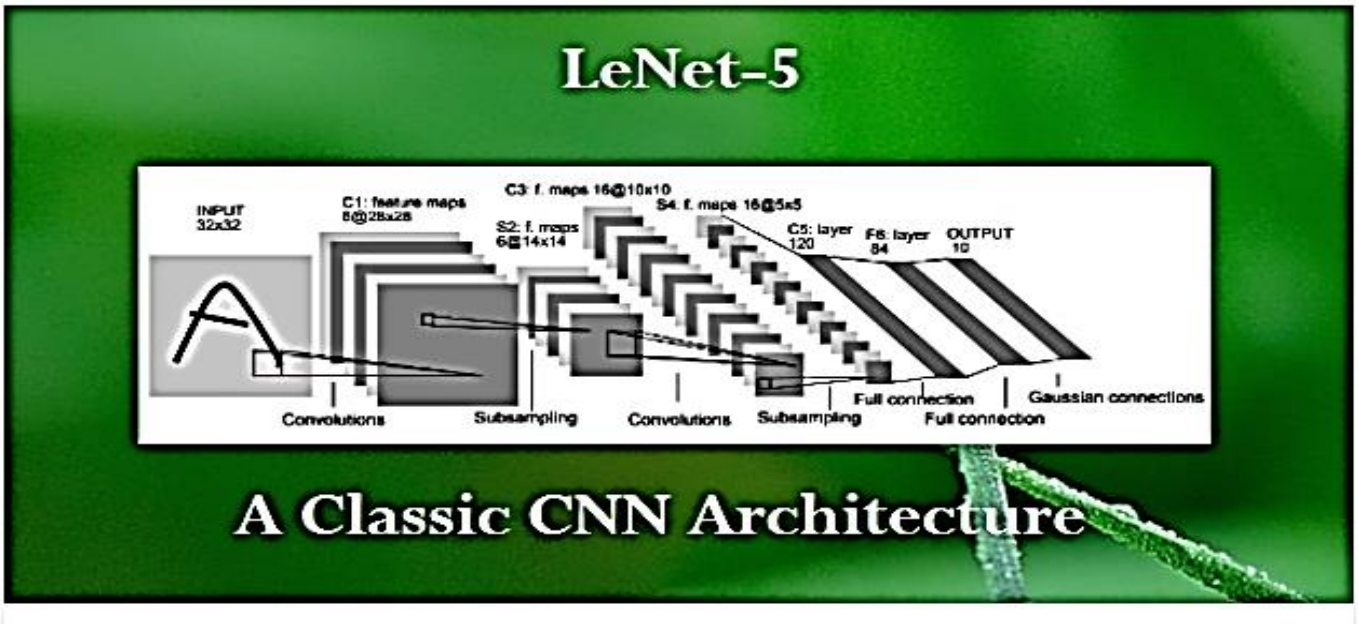

Figure 2.5: LeNet architecture [23].

2. AlexNet

Alex Krizhvsky et al in 2012 conceived this network structure and it took the first position that year in the ILSVRC competition by overwhelmingly beating other challengers. The network achieved 10.7 % reduction of the top-5 error results. The noticeable difference between this network and LeNet-5 is the application of additional filters per block in the AlexNet structure which makes it deeper and required solid hardware for its operation. The training becomes very slow as the network structure becomes deeper, hence the reason AlexeNet used two Graphics Processing Units (GPUs) to boost the process speed and it took close to one week of training for classification.

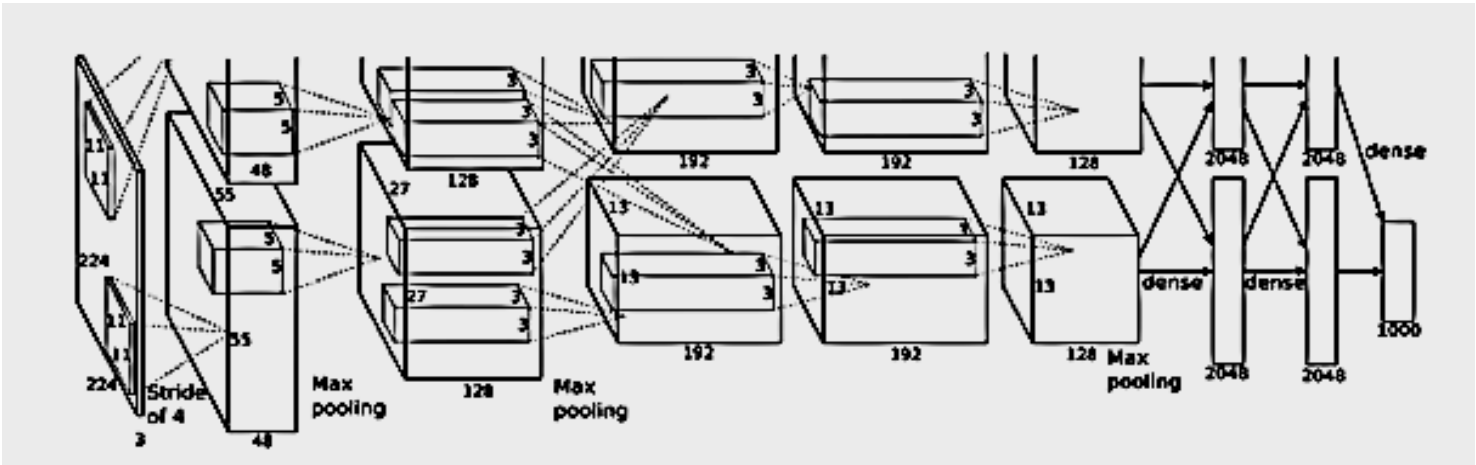

Figure 2.6: AlexNet architecture [24].

3. ZFNet

The winner of ILSVRC competition in 2013 was ZFNet, they managed to achieve top-5 error results, 0.5 % less than AlexNet. This network structure is like AlexNet, the only change made was the permutations of hyperparameters to increase its efficiency.

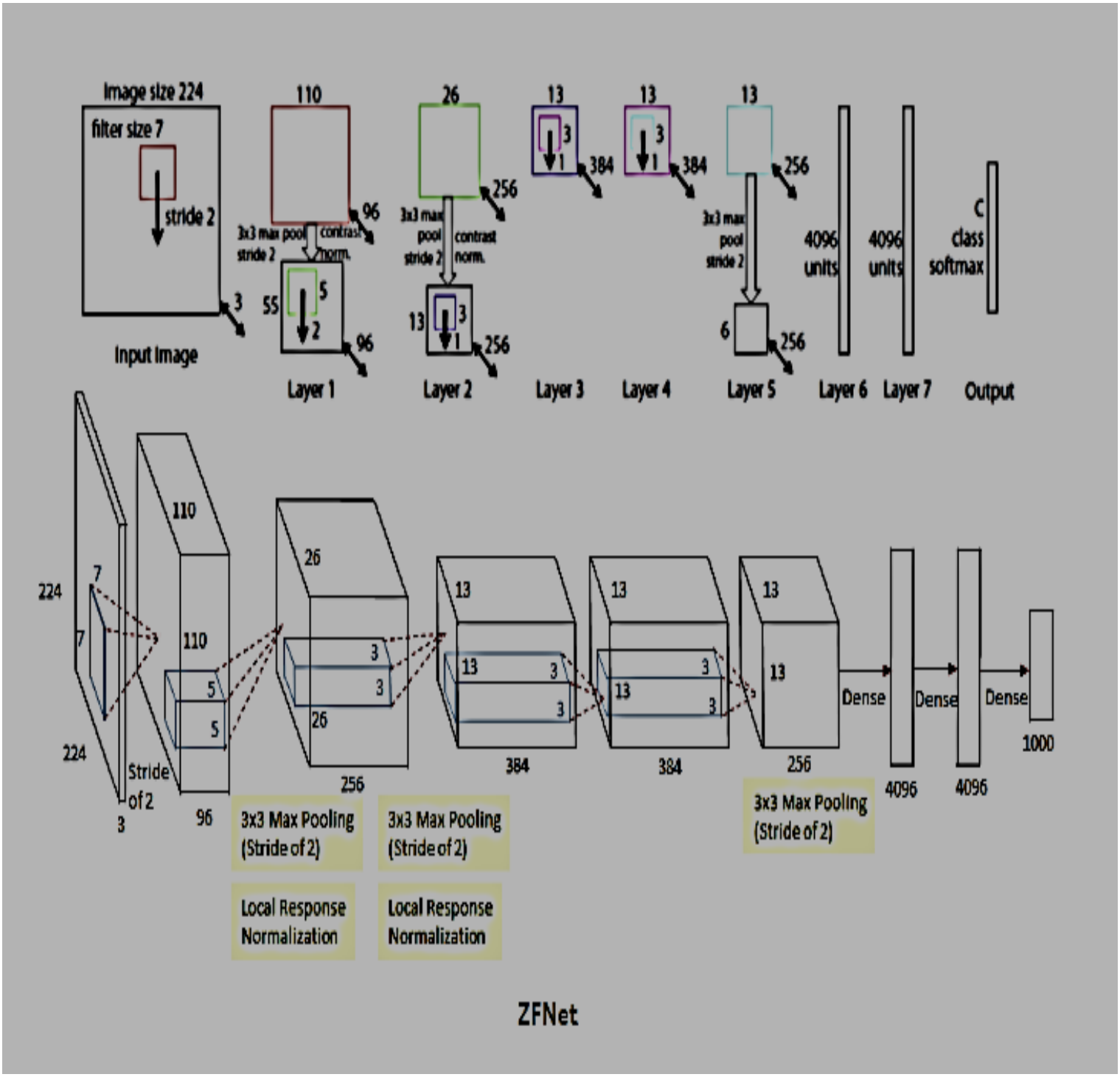

Figure 2.7: ZFNet architecture [25].

4. GoogLeNet

In 2014 Google's team of researchers designed GoogLeNet, which got its concepts from the network structure LeNet but incorporated many tiny convolutions hence the architecture consists of 22 convolutional blocks. This robust algorithm achieved outstanding top-5 error results which are nearly half the percentage of the top winner in the previous year. The results were comparable to human capacity of image recognition. Another key factor in the GoogLeNet design which enhanced its capacity is the use of parameters of the system network. This network structure used only around 6.67 % of parameters quantity which was used in AlexNet.

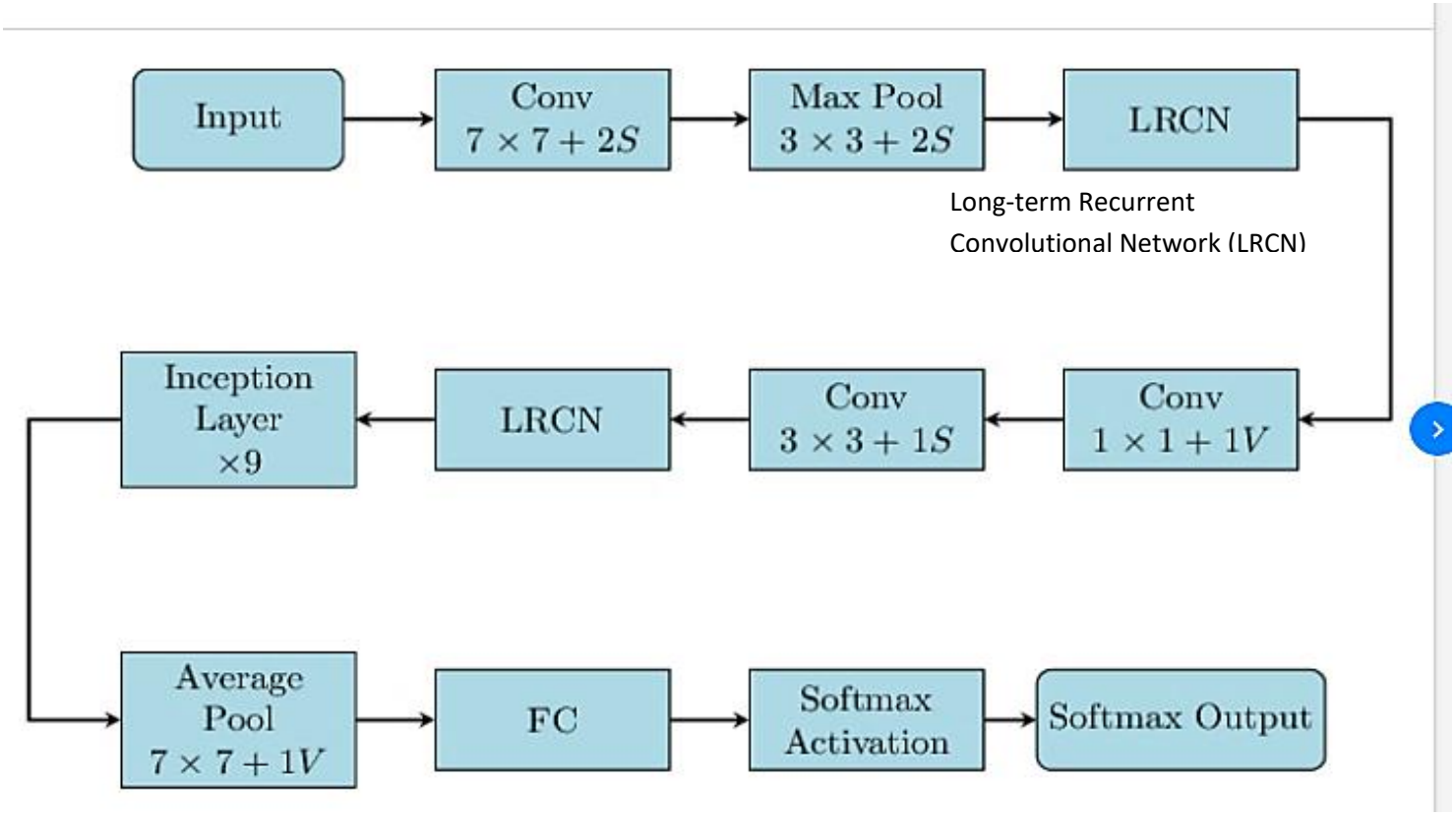

Figure 2.8: A simplified GoogLeNet architecture [26].

5. VGGNet

The network structure was designed by Simonyan and Zisserman in 2014 and finished behind GoogleNet in the ILSVRC 2014 competition. It is the most selected network model from the successful ILSVRC competition models to date by many researchers. The following are some of the reasons why the research world is in love with this architecture:

- It has a homogeneous structure of 3 by 3 convolutional layers that allows for easy replication.
- It is available on many online platforms.

The structure differs from AlexNet by only applying more filters in the network and houses 16 convolutional blocks. The limitation of this architecture is the high number of parameters. With over hundred million parameters, the structure requires more work for proper use. The system is very deep and it needed 14 to 21 days of training while using 4 GPUs for image classification during the competition.

Figure 2.9: VGGNet architecture [27].

6. ResNet

In 2015 *Kaiming He et al* proposed ResNet which finished top in the ILSVRC competition and it managed to reach overwhelming top-5 error results, not only at around half the percentage of what GoogLeNet performed but beating the human capacity so far in assessing the relevant dataset in the competition. The technique is based on relying on the batch normalization application and utilizing residual connections.

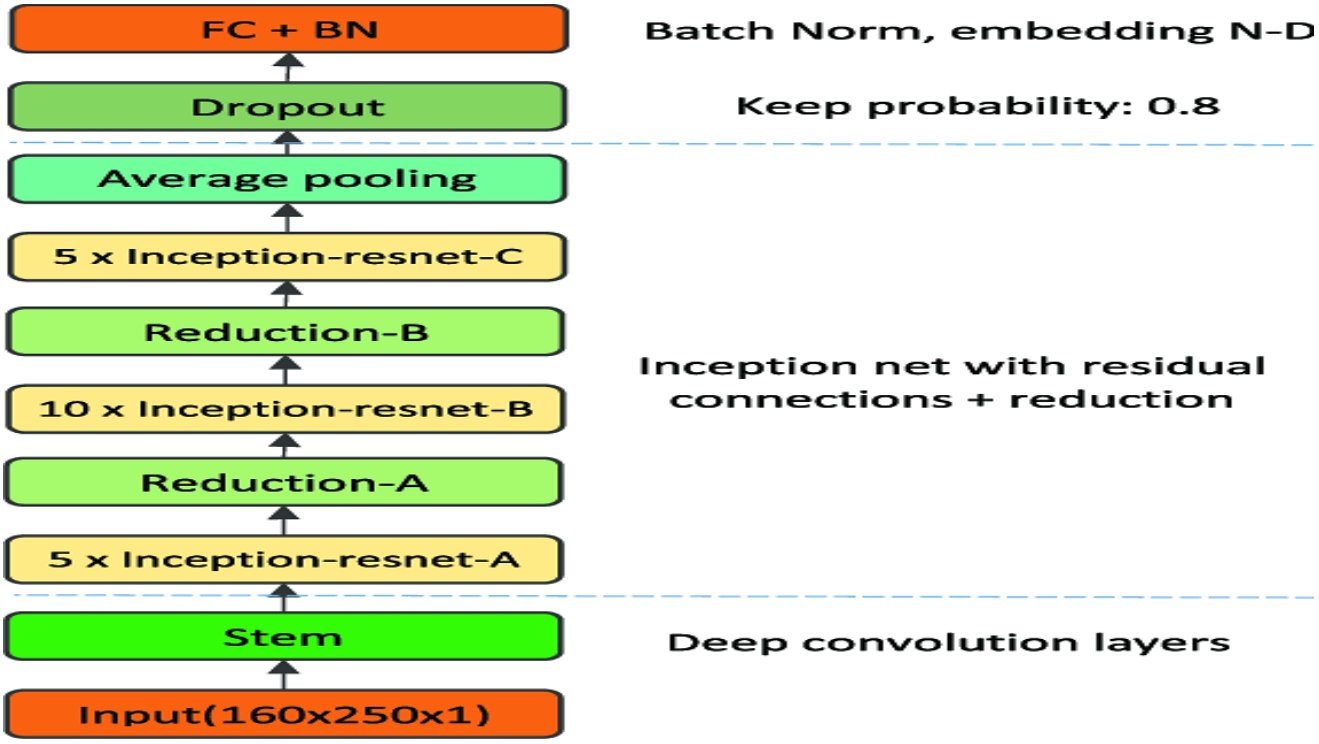

Figure 2.10: A simplified ResNet architecture [28]

2.3.2 EXISTING FER BASED STRURCTURES BESIDE THE ILSVRC CONTESTANTS

2.3.2.1 TRADITIONAL FER BASED METHODS

There is a massive record of FER studies which used traditional systems. All these traditional FER based systems share some similarities in their applications [29]:

- They capture the face area
- They retrieve geometric features
- They retrieve appearance features
- They extract from the facial object the combination of geometric and appearance features

In [29] a traditional FER based method is presented that was based on the fifty two dots located on the face. It was firstly applied to the geometric features according to the position of the dots as well as their angles. Secondly, they applied subtraction using the parallel position and angles from the first step of the video segment. All the face area is used to retrieve the appearance features. Another traditional FER based technique was used that emphasized only on certain sections of the face as the basis to apply extraction of the appearance features. An application of global features was demonstrated in [29] where several facial points and its distances in between were used as vectors for LBP, while the classification of different facial emotions was accomplished by PCA. This traditional FER based technique has its own limitations, such as; it fails the local variations reflections of the

various parts of the face which are linked to the vector, hence the accuracy results are poor. In [29] a FER based approach was proposed using videos, the approach consisted of calculating the distances of *x* and *y* facial landmarks between the recent frame and the precedent one as well as to retrieve the appearance features.

2.3.2.2 DEEP LEARNING FER BASED METHODS

In 2014, another FER deep based approach was proposed by Liu. It was used to recognise emotions in videos. The model operated in sequences of video pieces to be taken as a group of data references on Reimannian sets. To get the length of metrics, Reimannian kernels are used in relation to the references. For classification, a fusion technique is utilized [30].

A deep Genetic Algorithm (GA) was proposed by *Filipe et al*. in 2016. This technique was a key in deep learning neural networks for classification; its efficiency was based on the speed of the process which facilitated the neural networks operation to output good performance. The method was applied successfully in the Atari games [31].

A convolutional neural network (CNN) algorithm for FER was proposed in 2015 by Burkert. The model had four convolutional layers besides the input and the classification layers. The first convolutional layer is followed by two parallel feature-extraction convolutional layers which are the main segments of the design to produce good performance [32].

In 2016, Zhao proposed a deep FER model using Deep Region Multi-Label Learning (DRML). The method was built in a way that a CNN was incorporated directly to reach Action Unit (AU) detection. The model had seven layers in between the input and output layers preceded by two fully connected layers before classification. Convolutional layer 1 and convolutional layer 3 are separated by a region layer and a pooling layer [33].

The feature maps deriving from CNN recognition processes combined with Facial Action Coding System (FACS) and Action Units (AU) yielded good results in emotions recognition. The model applied the collaboration of the above, the features capacity in classifying the facial expressions was impressive [34].

A DCNN technique using multivariate ordinal variables was proposed by Walecki in 2017 and the method was efficient in solving AU intensity estimation [35].

CHAPTER3 HIERARCHICAL DEEP NEURAL NETWORK STRUCTURE

This chapter explains the different proposed HDNN structures for facial expressions recognition. The chapter explains the concepts of the generic structure which was the foundation that we used to derive our 24 proposed HDNN structures. Additionally, the chapter presents the HDNN structure case studies with variable parameters. All the proposed case studies of the HDNN structures to be investigated are covered in this chapter.

3.1 MOTIVATION FOR HDNN STRUCTURE

This research aims to find an optimal HDNN structure with optimal settings for facial expression recognition. This optimal HDNN structure has two goals. Firstly, to achieve high accuracy compared to current FER based models and secondly, to use inexpensive hardware such as a standard laptop to apply the optimal HDNN algorithm and achieve good performance results within appropriate training time.

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) structures detailed in chapter 2 have achieved outstanding results but have drawbacks. They require expensive hardware such as expensive computers with high-end processors which can operate complex structures. Additionally, the training of these deeper structures will take time hence the need to buy very expensive GPUs to alleviate the problem and improve the speed of the training process. Even though, GPUs are used, the training of these deeper network structures is long, sometimes days, or weeks to achieve image classification.

The existing FER based architectures have been progressing well in the research arena with some fair results, but those results can be improved. There is a need to find a system which can accurately recognize human facial expressions. The popularity of Artificial Intelligence (AI) these days can attest to that. From the use of facial recognition to unlock smart phones, to programmed robots, FER is still a problem for machines to comprehend effectively the dynamic shifts in facial expressions in emotions of human beings.

To remedy the above limitations, different HDNN structures with variable settings are proposed with the aim of investigating them to find the optimal structure with optimal settings.

3.2 HDNN STRUCTURES

We will apply the different proposed HDNN structures and compare them to find the optimal structure with optimal parameters. All the different HDNN structures will have fixed size input and output classification of human emotions, the other parameters will be variables. Included, the structures will be applying *Relu* as an activation function and S*oftmax* for classification.

We will test our HDNN structures using Keras with Theano as back-end, of which are Python libraries [36]. Each HDNN structure case will be trained using the dataset for 30 epochs as a standard

3.2.1 GENERIC STRUCTURE

We made a general structure to enable us to make the permutations of the variables. The size of filters of the convolutional maps are indicated by *i* x *j*, the size of filters of the pool are indicated by *k* x *l*, *m* is the value number of maps, *x* is the value number of dropout and *n* is the number of neurons. The generic structure is displayed in figure 2. Besides the variables, this generic structure inspired us to derive three different network structures which make the total of four network structures including the generic structure to be investigated in this research study. Table 3.1 illustrates the four case studies and Figure 3.1 shows the generic structure which was used to derive the other network structures.

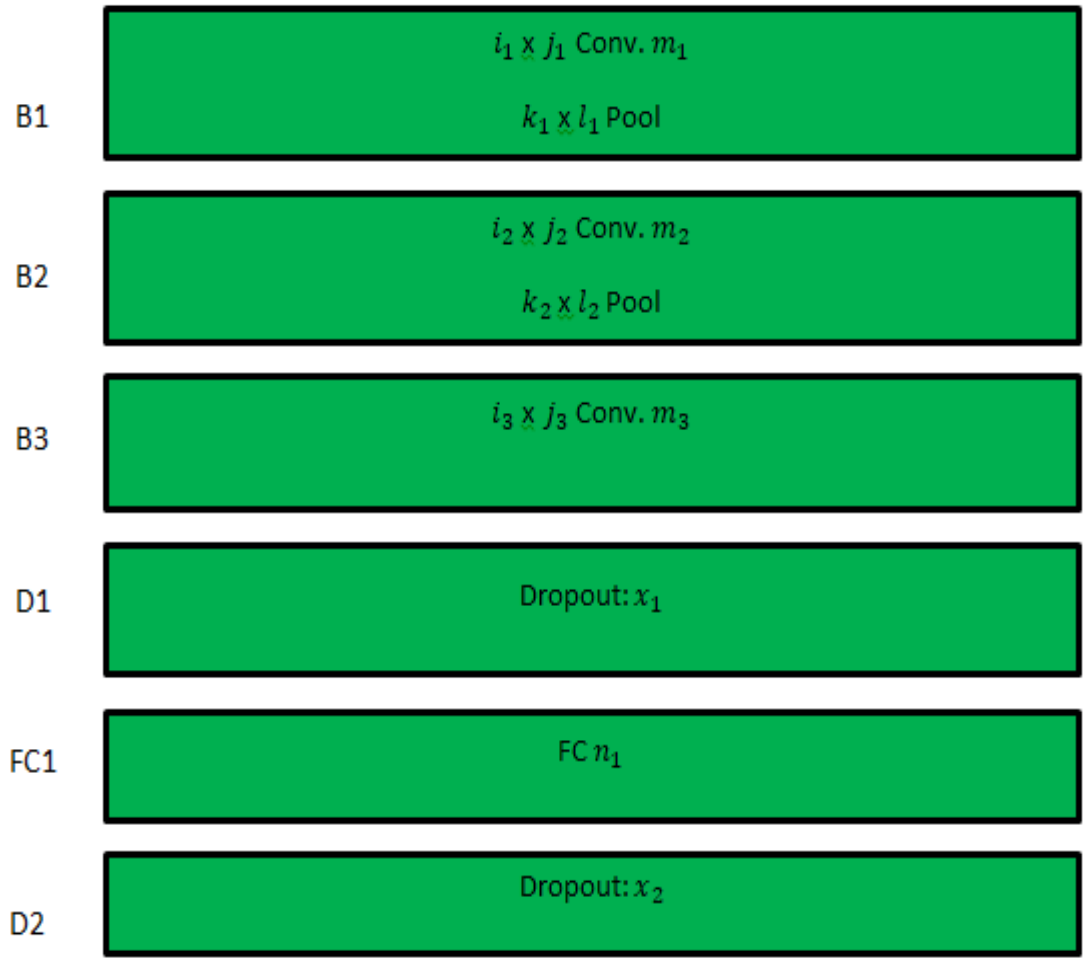

Figure 3.1: Generic Structure.

3.2.2 STRUCTURE CASE STUDIES

Table 3.1 shows the case studies that we will be investigated in order to find the optimal structure with optimal parameters. The description of each case study is explained below for suitable understanding of the concept behind the construction of each network structure. The four proposed case studies are the foundations where we will derive other case studies depending on the variable parameters defined below as part of the scope of this research study.

Table 3.1: Proposed structure case studies.

Case 1	Case 2	Case 3	Case 4
B1	B1	B1	B1
FC1	B2	B2	B2
D2	FC1	B3	B3
	D2	D1	D1
		FC1	FC1
			D2

The following criteria are used to choose the cases which will be studied:

Computational time: < 120 minutes

$1 \leq i_a , j_a \leq 8$

$1 \leq k_a , l_a \leq 3$

$0 < x_1 < 0.3$

$n_1 \leq 150$

$0 < x_2 < 0.6$

In case study 1, the structure is very simple and consists of one convolutional block noted as B1 and one fully connected block indicated as FC1. We also apply the drop out technique after the fully connected layer using D2 which was set at 0.5 in order to address the over-fitting challenge that might arise in the course of the training.

In case study 2, the concept was to add another convolutional block B2 to the structure in case study 1, without changing the previous network structure. This means the fully connected block and the application of dropout D2 remained unchanged except that the structure has two convolutional blocks.

In case study 3, the network structure becomes deeper with three convolutional blocks and one fully connected block. We had made some alterations, instead of using D2 which was set at 0.5 in the structure after the fully connected layer, we removed D2 completely from the structure and introduced D1 after the convolutional block B3 and had set it at 0.25 to take care of over-fitting in the training stage.

In case study 4, this network structure is similar to the one in case study 3, with only the inception of D2 in the structure which makes it different to the previous network structure. Therefore, the structure has two dropouts D1 and D2 set at 0.25 and 0.5 respectively. The rest remained unchanged, three convolutional blocks and one fully connected block.

STRUCTURE CASE 1

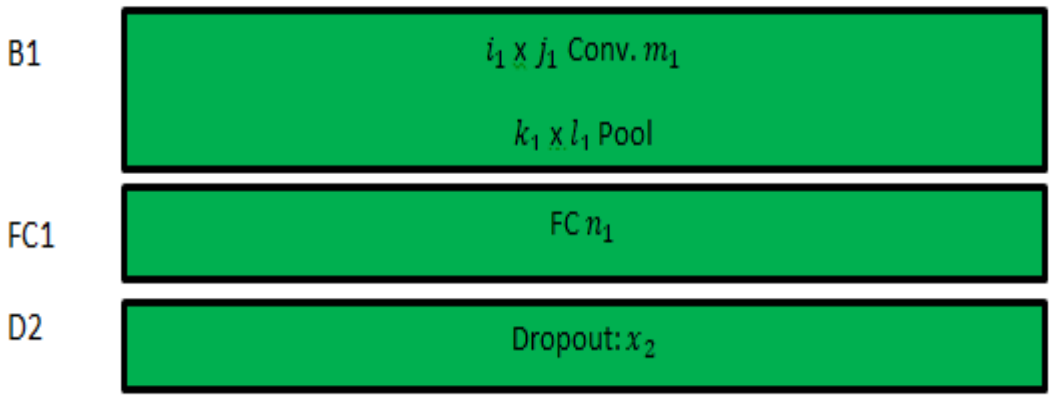

Figure 3.2: Case 1 Structure.

From this design, we derived six network structures which are shown in Table 3.2. The first three structures have common size filters for max-pooling set at 2 by 2; their difference is only based on their size of the filters for the convolutional maps. In case 1a, we applied 3 by 3 convolutional maps. We then modified our design in case 1b by using 4 by 4 convolutional maps. In case 1c, we set our convolutional maps at 5 by 5. Finally, the last three network structures have similar size of filters for max-pooling but it is modified and set at 2 by 1. Again, the same concept is repeated with case 1d using 3 by 3 convolutional maps, while case 1e and case 1f have their convolutional maps set at 4 by 4 and 5 by 5 respectively.

We will continue to use the same approach in the rest of the proposed case study structures. From structure 2 designs, we will have another six network structures by applying the same logic which will provides us with six network structures. In the structure 3 designs, with the same logic we will have another six network structures. Our last structure, structure 4

designs, will produce six network structures. In total, we will have 24 network structures with variable parameters to conduct our investigation.

Table 3.2: Cases for structure 1

CASE1a:	**CASE1b:**	**CASE1c:**
$i_1 = j_1 = 3$	$i_1 = j_1 = 4$	$i_1 = j_1 = 5$
$k_1 = l_1 = 2$	$k_1 = l_1 = 2$	$k_1 = l_1 = 2$
$m_1 = 6$	$m_1 = 6$	$m_1 = 6$
$n_1 = 84$	$n_1 = 84$	$n_1 = 84$
$x_2 = 0.5$	$x_2 = 0.5$	$x_2 = 0.5$
CASE 1d:	**CASE 1e**	**CASE 1f:**
$i_1 = j_1 = 3$	$i_1 = j_1 = 4$	$i_1 = j_1 = 5$
$k_1 = 2$	$k_1 = 2$	$k_1 = 2$
$l_1 = 1$	$l_1 = 1$	$l_1 = 1$
$m_1 = 6$	$m_1 = 6$	$m_1 = 6$
$n_1 = 84$	$n_1 = 84$	$n_1 = 84$
$x_2 = 0.5$	$x_2 = 0.5$	$x_2 = 0.5$

STRUCTURE CASE 2

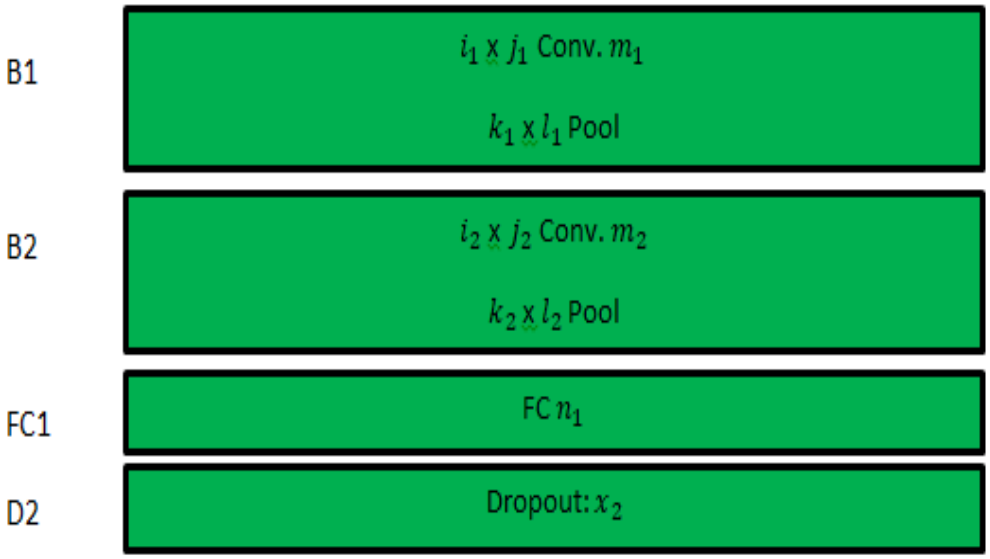

Figure 3.3: Case 2 Structure.

We applied the same logic as in the case study 1 and derived six network structures which are detailed in Table 3.3 by changing the values of the size of filters for max-pooling and convolutional maps.

Table 3.3: Cases for structure 2

CASE2a:	CASE:	CASE2c:
$i_1 = j_1 = 3$	$i_1 = j_1 = 4$	$i_1 = j_1 = 5$
$i_2 = j_2 = 3$	$i_2 = j_2 = 4$	$i_2 = j_2 = 5$
$k_1 = l_1 = 2$	$k_1 = l_1 = 2$	$k_1 = l_1 = 2$
$k_2 = l_2 = 2$	$k_2 = l_2 = 2$	$k_2 = l_2 = 2$
$m_1 = 6$	$m_1 = 6$	$m_1 = 6$
$m_2 = 16$	$m_2 = 16$	$m_2 = 16$
$n_1 = 84$	$n_1 = 84$	$n_1 = 84$
$x_2 = 0.5$	$x_2 = 0.5$	$x_2 = 0.5$
CASE2d:	**CASE2e:**	**CASE2f:**
$i_1 = j_1 = 3$	$i_1 = j_1 = 4$	$i_1 = j_1 = 5$
$i_2 = j_2 = 3$	$i_2 = j_2 = 4$	$i_2 = j_2 =$
$k_1 = k_2 = 2$	$k_1 = k_2 = 2$	$k_1 = k_2 = 2$
$l_1 = l_2 = 1$	$l_1 = l_2 = 1$	$l_1 = l_2 = 1$
$m_1 = 6$	$m_1 = 6$	$m_1 = 6$
$m_2 = 16$	$m_2 = 16$	$m_2 = 16$
$n_1 = 84$	$n_1 = 84$	$n_1 = 84$
$x_2 = 0.5$	$x_2 = 0.5$	$x_2 = 0.5$

3. STRUCTURE CASE 3

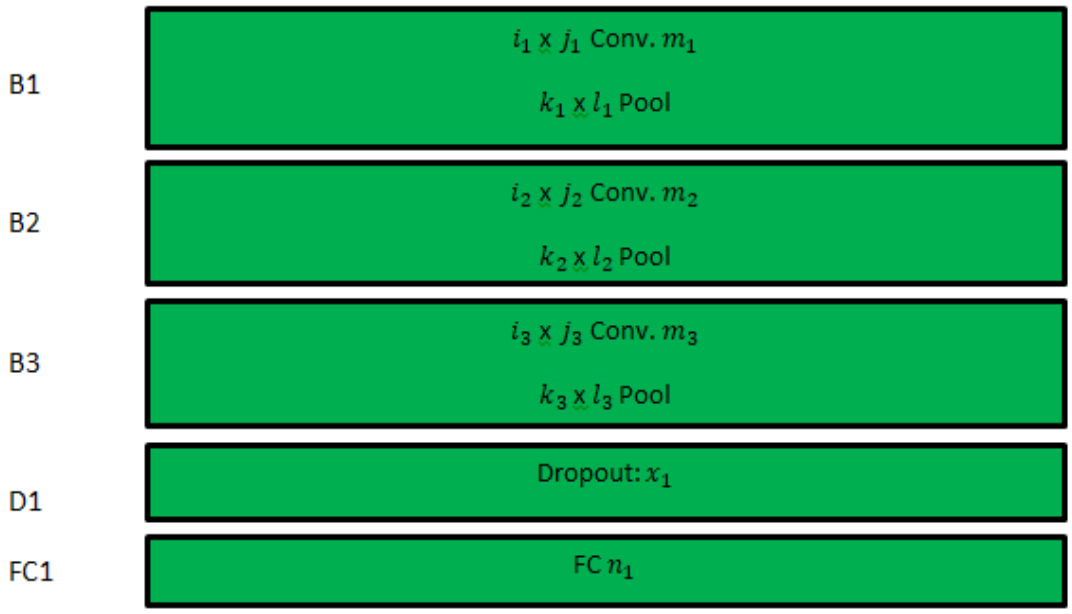

Figure 3.4: Case 3 Structure.

This structure also gives birth to six network structures by using the variables to be investigated. They are detailed in Table 3.4.

Table 3.4: Cases for structure 3

CASE3a:	**CASE3b:**	**CASE3c:**
$i_1 = j_1 = 3$	$i_1 = j_1 = 4$	$i_1 = j_1 = 5$
$i_2 = j_2 = 3$	$i_2 = j_2 = 4$	$i_2 = j_2 = 5$
$i_3 = j_3 = 3$	$i_3 = j_3 = 4$	$i_3 = j_3 = 5$
$k_1 = l_1 = 2$	$k_1 = l_1 = 2$	$k_1 = l_1 = 2$
$k_2 = l_2 = 2$	$k_2 = l_2 = 2$	$k_2 = l_2 = 2$
$k_3 = l_3 = 2$	$k_3 = l_3 = 2$	$k_3 = l_3 = 2$
$m_1 = 6$	$m_1 = 6$	$m_1 = 6$
$m_2 = 16$	$m_2 = 16$	$m_2 = 16$
$m_3 = 120$	$m_3 = 120$	$m_3 = 120$
$n_1 = 84$	$n_1 = 84$	$n_1 = 84$
$x_1 = 0.25$	$x_1 = 0.25$	$x_1 = 0.25$
CASE3d	**CASE3e:**	**CASE3f:**
$i_1 = j_1 = 3$	$i_1 = j_1 = 4$	$i_1 = j_1 = 5$
$i_2 = j_2 = 3$	$i_2 = j_2 = 4$	$i_2 = j_2 = 5$
$i_3 = j_3 = 3$	$i_3 = j_3 = 4$	$i_3 = j_3 = 5$
$k_1 = k_2 = 2$	$k_1 = k_2 = 2$	$k_1 = k_2 = 2$
$l_1 = l_2 = 1$	$l_1 = l_2 = 1$	$l_1 = l_2 = 1$
$k_3 = 2$	$k_3 = 2$	$k_3 = 2$
$l_3 = 1$	$l_3 = 1$	$l_3 = 1$
$m_1 = 6$	$m_1 = 6$	$m_1 = 6$
$m_2 = 16$	$m_2 = 16$	$m_2 = 16$
$m_3 = 120$	$m_3 = 120$	$m_3 = 120$
$n_1 = 84$	$n_1 = 84$	$n_1 = 84$
$x_1 = 0.25$	$x_1 = 0.25$	$x_1 = 0.25$

4. STRUCTURE CASE 4

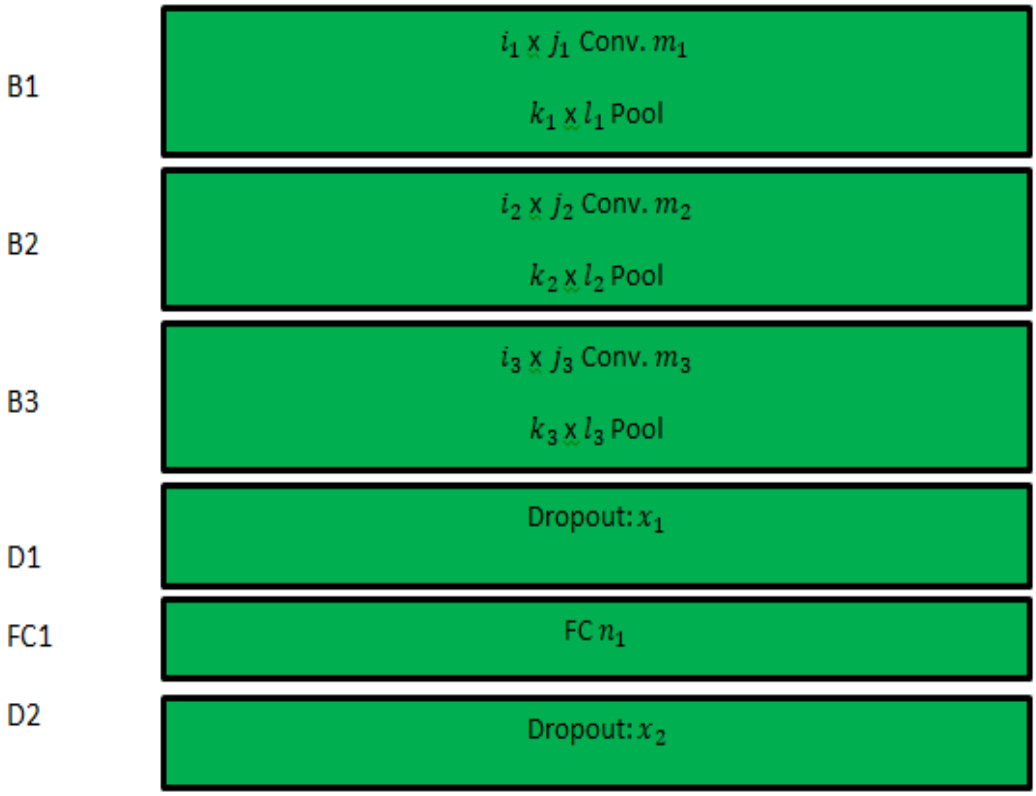

Figure 3.5: Case 4 Structure.

Six network structures are derived from the above structure concept by modifying the parameter variables, which are shown in Table 3.5. As explained in the case study 1, the same concept is applied regarding the size of filters for max-pooling and convolutional maps. The difference between the cases is based on the values set for size of filters, but the dropouts values remained unchanged and are set at 0.25 and 0.5 for D1 and D2.

Table 3.5: Cases for structure 4

CASE4a:	CASE4b:	CASE4c:
$i_1 = j_1 = 3$	$i_1 = j_1 = 4$	$i_1 = j_1 = 5$
$i_2 = j_2 = 3$	$i_2 = j_2 = 4$	$i_2 = j_2 = 5$
$i_3 = j_3 = 3$	$i_3 = j_3 = 4$	$i_3 = j_3 = 5$
$k_1 = l_1 = 2$	$k_1 = l_1 = 2$	$k_1 = l_1 = 2$
$k_2 = l_2 = 2$	$k_2 = l_2 = 2$	$k_2 = l_2 = 2$
$k_3 = l_3 = 2$	$k_3 = l_3 = 2$	$k_3 = l_3 = 2$
$m_1 = 6$	$m_1 = 6$	$m_1 = 6$
$m_2 = 16$	$m_2 = 16$	$m_2 = 16$
$m_3 = 120$	$m_3 = 120$	$m_3 = 120$
$n_1 = 84$	$n_1 = 84$	$n_1 = 84$
$x_1 = 0.25$	$x_1 = 0.25$	$x_1 = 0.25$
$x_2 = 0.5$	$x_2 = 0.5$	$x_2 = 0.5$
CASE4d:	**CASE4e:**	**CASE4f:**
$i_1 = j_1 = 3$	$i_1 = j_1 = 4$	$i_1 = j_1 = 5$
$i_2 = j_2 = 3$	$i_2 = j_2 = 4$	$i_2 = j_2 = 5$
$i_3 = j_3 = 3$	$i_3 = j_3 = 4$	$i_3 = j_3 = 5$
$k_1 = k_2 = 2$	$k_1 = k_2 = 2$	$k_1 = k_2 = 2$
$l_1 = l_2 = 1$	$l_1 = l_2 =$	$l_1 = l_2 = 1$
$k_3 = 2$	$k_3 = 2$	$k_3 = 2$
$l_3 = 1$	$l_3 = 1$	$l_3 = 1$
$m_1 = 6$	$m_1 = 6$	$m_1 = 6$
$m_2 = 16$	$m_2 = 16$	$m_2 = 16$
$m_3 = 120$	$m_3 = 120$	$m_3 = 120$
$n_1 = 84$	$n_1 = 84$	$n_1 = 84$
$x_1 = 0.25$	$x_1 = 0.25$	$x_1 = 0.25$
$x_2 = 0.5$	$x_2 = 0.5$	$x_2 = 0.5$

3.2.3 SUMMARY

This chapter focused on the various proposed HDNN structures with variable parameters. There are 4 HDNN structures and each of these structures has six case studies that are investigated. In all the case studies, the size input is fixed at 128 x 128 and the output results classify seven human emotions. The number of maps for each convolution m_1 , m_2 and m_3 is also fixed in all case studies and the number of neurons is not varied but set at 84 neural. The rest of the parameters are variables because we had to change the size of the filters of max-poling and the convolutional maps in order to have various case studies structures in order to conduct our investigation. The first three structures have a common size of filters for max-pooling set at 2 by 2; their difference is only based on their size of filters for the convolutional maps. In case a, we applied 3 by 3 convolutional maps. We then modified our design in case b by using 4 by 4 convolutional maps. In case c, we set our convolutional maps at 5 by 5. Finally, the last three network structures have similar sizes of filters for max-

pooling but it is modified and set at 2 by 1. Again, the same concept is repeated with case d by using 3 by 3 convolutional maps while case e and case f have their convolutional maps set at 4 by 4 and 5 by 5 respectively. In total, we have 24 HDNN structures to be analyzed in order to test each case studies performance with the goal of finding the optimal HDNN structure with optimal parameters.

CHAPTER 4 EXPERIMENTAL SET UP

This chapter details the two FER datasets used in research. The chapter also elaborates the structure design. It is then followed by detailing the training process which is subdivided in eight points.

4.1 INTRODUCTION

In preparation for this research study, some decisions were made in order to accomplish this project. The experiments needed to be performed so that we could conduct our investigation of network structures. The first decision was about which datasets to use so that we could test our 24 proposed network structures. We chose the databases which are publicly available with simple procedures to access them. Secondly, the decision about which programming language to use for our code was also based on the accessibility; we had chosen Python which is available on the internet without requiring any licence for the purchase. Initially considered Matlab but it requires a purchasing licence. Python was used as our final choice. In addition, Python has many libraries which are accessible on the internet.

4.2 EXTENDED COHN-KANADE AND JAPANESE FEMALE FACIAL EXPRESSION DATASETS

We have a number of existing FER datasets in the research arena with good image data but many of these superb datasets demand registration before being granted access to use their services. Some require simple procedures, used FER datasets and their images are well categorised.

The first dataset which we used is the CK+. It is composed of a branded emotion number for human expression faces. There were over 100 participants and close to 600 pictures, with around half of those related to the seven human emotions which are: Anger, Disgust, Fear, Happy, Neutral, Sad and Surprise [15].

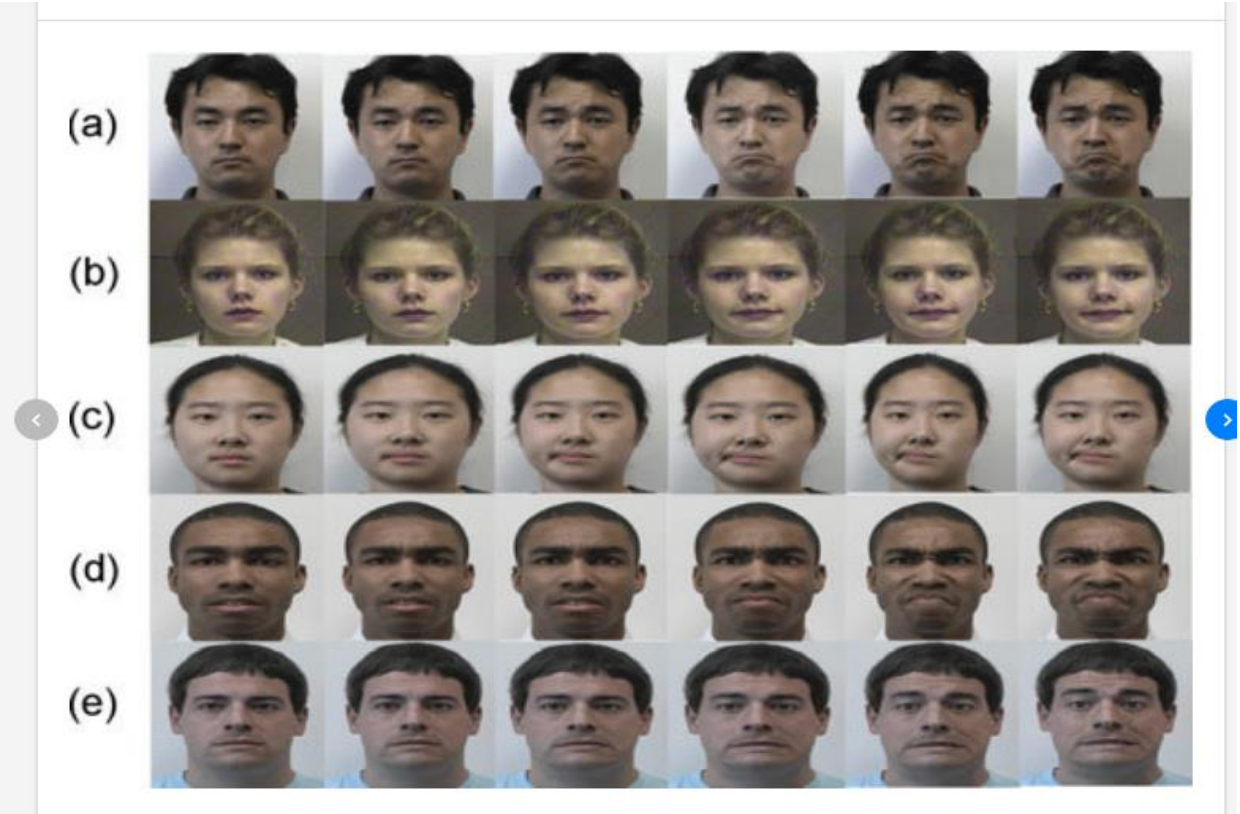

Figure 4.1: CK+ Dataset images' examples [37].

JAFFE dataset detailed in [15] is also used to validate the generalization of architecture.

Figure 4.2: JAFFE Dataset images' examples [38].

4.3 STRUCTURE DESIGN

The components we utilized to train our proposed network structures. Firstly, we simply used a standard laptop as reducing financial costs of hardware was one of the objectives of this research, we wanted a cheap system but effective in delivering good performance. Our device had a CPU Intel(R) Core(TM) 2 with a clock speed of 2.40GHz and a RAM of 4 GB. We did not want to use expensive GPUs but wanted to achieve comparable results with models using GPUs. Secondly, the choice of the programming language to use was decided based on the fact that python language is free to access without requiring any licencing to purchase it. The following are the python libraries that were used [36][39]:

- Numpy is a python platform which allows many scientific operations to be executed through its several utilities [36].
- Theano: makes it possible for the incorporation of Numpy which facilitates the execution of the codings using Numpy functions. Theano offers more features than Numpy; the latter cannot offer other features which Theano can only deliver. It is a Python library that facilitates to "define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently" [39]. Theano is a python package that gives us tools to operate and handle multi-dimensional ranges at the same time it still maintains the performance. Being part of python enables Theano to successfully deliver a platform to process faster mathematical operations. A key feature of theano is the stability that it offers and capitalises optimization that makes it useful for challenging operations. Theano is utilized in high level technical studies. A debugging feature is another key element that makes Theano as one of the best choice for researchers [36].
- OpenCV: can work with many programming languages and is an open source software with thousands of optimized and AI models. These models make it possible to implement image classification and other functions [40].
- Keras: was designed to work with Theano or TensorFlow and is an application programming interface (API) for neural networks. It has the capacity for faster operations and gives the possibility to make changes when needed. This API is user friendly and allows the user to operate functions to enhance the network. Elements can be put together to build a neural network structure. The structure can be investigated while bypassing small details. It is compatible with many operating systems [36].

We used Anaconda which is a Python distribution and it is not difficult to operate and is accessible on the internet with simple installation procedures. It has hundreds of packages for computations and is a python distribution [41].

4.4 TRAINING PROCESS

For appropriate training to take place, hyper parameter settings are needed. These prerequisites allow a model to perform better. The programmer needs to perform settings allocation to these parameters in order to achieve desirable results. The following are the

parameters to be set: activation functions, learning rate, batch size, epochs, gradient descent, max-pooling, convolutional filters and number of neurons.

4.4.1 ACTIVATION FUNCTIONS

This hyper parameter is in charge of neurons spreading and is a key element when a network structure has many levels. When the activation function is initialized, the following occurs [42]:
Exploding gradient: this issue occurs during back propagation of the network. This network instability arises when there is a large error gradient.
Vanishing gradient: this issue occurs when the gradient tends to zero and the training of the model becomes very difficult.

The two most used activation functions are the *ReLu* and the *Softmax*, the first helps to remedy the above challenges and has a goal of presenting non-linearities to the structure while the latter is applied at the end of the structure for classification, this function can be illustrated as an input vector of totals which gives an output vector of a probability.
The following are other types of activation functions [42]:

- **Sigmoid:** consists of placing the input value in an array of zero to one. The sigmoid activation function converted to zero for big numbers which are negative and converted to one for big numbers which are positive.
- **Tanh:** consists of placing the input value in an array of negative one to positive one. One point to mention is that the gradient is stronger for tanh than sigmoid.
- **Maxout:** this activation function is mostly used in the following cases; firstly, it is when the dropout technique is applied in a network structure to improve the training process. Secondly, it is for complex network structures which are very deep and should give the *ReLu* activation function problems. Thirdly, it is used where all the parameters in a network structure would make use of the dropout so that the improvement of the training process can take place. Lastly, it is used where the *ReLu* activation function would expose its limitations, the Maxout activation function would take over and still provide all the advantages of *ReLu* activation function.

4.4.2 LEARNING RATE

When the training process is under way, the model weights are regularly updated. It can affect the estimated error, in order to have control over this process, we have a valuable hyper parameter which is the learning rate, it manages how the model can transform in reaction to the estimated error whenever there is occurrence of any weights update. Big learning rate can destabilise the training process while the lesser learning rate can effect the training resulting in a failure. Therefore, choosing the correct learning rate value is fundamental in order to have a successful training with better performance [43]. In this research, we will use callbacks. Keras supports learning rate schedules via callbacks. We will use the keras feature to adjust the learning rate accordingly by specifying the metric to monitor during the training via the “monitor” argument and the validation loss that will be monitored.

4.4.3 BATCH SIZE

Model parameters are updated internally during the training process. The hyper parameter which specifies the number of samples to operate with prior to the above update occurrence is called batch size. Batch size gives predictions as outcome at the end of a cycle. Therefore, selecting a proper batch size value can improve your training performance with better results [44]. In this research, we will set our batch size to 7; this number was proven to be effective in performance with our selected FER database. For other database, for example a larger database can have a batch size set to a big number. Batch size setting depends on the number of images of the database.

4.4.4 EPOCHS

The number of times that certain architectures need to train in order to learn from the dataset is always set at certain value during the training process and the hyper parameter in charge of that task is called the number of epochs. This value can range from ten to thousands depending on the user's choice, and bigger numbers give good performance. In this study, we will use 30 as the number of epochs because the current existing models also applied the same. Therefore, we will limit ourselves to that number to be fair and to accurately compare the results of our network structures to other FER existing deep learning models [44].

4.4.5 GRADIENT DESCENT

Gradient descent is assigned the duty of finding the values that can reduce the cost function. These parameter's values are noted as coefficients of a certain function indicated as *f*. This optimization algorithm is very important for a successful training. In this research, we will be using *Adam* as our optimization algorithm [45].

4.4.6 MAX-POOLING

Down-sampling an input data is the purpose of this hyper parameter. By making the dimensions of the data smaller, it enables correct analysis of the features. Therefore, the resulted data representation form does assist in alleviating the over-fitting. It is a cost effective hyper parameter that lessens the number of parameters during the learning process hence the computational costs also decrease as a result of applying this hyper parameter. The size of filters for max-pooling has a huge impact in the performance and they are described in the form of k x l [46].

4.4.7 CONVOLUTIONAL FILTERS

4.4.7.1 SIZE

They are in the form of i x j, choosing the correct size of filters for the convolutional layers is essential to get better results. These hyper parameters carry out convolutions over an input size data which results in an activation map. The settings depend on the programmer and the need to achieve good results [20].

4.4.7.2 NUMBER

This is the amount of activation maps representing the number of convolutional filters. Each convolutional filter is applied in order to create a feature map according to the input. We described this number in our research as *m* [47].

4.4.8 NUMBER OF NEURONS IN THE FULLY CONNECTED BLOCK

It is indicated in our research as *n*, it is up to the programmer to select any number from one up to ten thousand. Any number will work because the task of this hyper parameter is to replace the following layer's weights mode [47].

4.5 ASSESSMENT

This stage stresses on investigating thoroughly the simulation results and comparing them to the current existing FER deep learning models. We selected the existing FER deep learning models which used the same two FER datasets detailed above, and managed to achieve top accuracy most recently.
The following apparatus are utilized in this study to give us the ability to reach the intention of this work:

- **Tables**
- **Figures**
- **Plots**

From analysing the above apparatuses, we were able to give conclusions about our investigation in this research.

CHAPTER 5 SIMULATION RESULTS AND ANALYSIS

This chapter details the research results of our study. The chapter also gives the summary of our observations regarding the investigation.

5.1 INTRODUCTION

The simulation results regarding the investigation of the case studies conceived in chapter 3 and the derived 24 different proposed HDNN structures are recorded and analysed. The comparison of performances of all of the 24 different structures with variable parameters is also covered per case study, and also the top performers are compared with other existing models.

We use the databases described in [15]. These databases are the most used in the research arena of facial expressions recognition.

The implementation is done using a standard CPU with windows OS, Intel(R) Core(TM) 2 with a clock speed of 2.40GHz, RAM of 4 GB. We test our proposed network structures using Keras with Theano as backend which are Python libraries. Each network architecture case was trained using the dataset for 30 epochs as a standard.

5.2 INVESTIGATION OF THE RESULTS

The tests on the CK+ Dataset: Table 5.1 to Table 5.4 shows the comparison results of our proposed network architectures. We compare the simulation results for all 24 HDNN network structures separated in four groups with each group having six case studies.

Table 5.1 details the accuracy results for the six case studies of architecture 1. For the sake of proper accuracy evaluation, we recorded the minimum, average and maximum accuracy results because the accuracy results were not stable. When conducting the simulations, the accuracy kept on changing which means two consecutive simulations could give two different accuracy results for the same network structure.

Therefore, we recorded our simulations in a statistical form. Observing Table 7, we noticed that case 1a performed better than all other cases with 87.50 % accuracy and case 1c came second with the accuracy of 87.46 % based on the average accuracy results. When considering the maximum accuracy results, case 1a recorded 98.11 % accuracy while case 1c managed 96.23 % accuracy. We noted that the two case studies gave good performance even though case 1a finished as winner of the group. The following are observations of the overall performance of the group:
The difference can be noticed that the first three case studies, case 1a, case 1b and case 1c achieved better performance than the last three case studies, case 1d, case 1e and case 1f. The first three case studies used the max-pooling of 2 by 2 and the last three case studies used the max-pooling of 2 by 1. Max-pooling of 2 by 2 improved the performance of the first three

network structures; they all achieved the average accuracy of above 85 % considering the stability of the accuracy results rather than the maximum accuracy results which can be misleading in other scenarios. Not only that the max-pooling of 2 by 1 cases achieved less than 85 % accuracy, they also took longer time of training than the other cases. For each network structure case study, the 2 by 1 max-pooling will take around three times the amount of training time than that of the 2 x 2 max-pooling. For future research, max-pooling needs to be taken in pairs for good performance, for example 1 by 1, 2 by 2, 3 by 3 instead of 1 by 2, 2 by 1, 3 by 1, but we need to clarify that the training time was the main point in our research. One of the objectives of this study was to acquire good results in an appropriate time using a standard laptop. Therefore, training time of many hours is out of scope for this research. So future research where the amount of training time is not a problem, max-pooling of different numbers like 1 by 2, 2 by 1, 2 by 3, 3 by 2, 3 by 1, 1 by 3 needs to be researched and there is a possibility they might yield good accuracy results.

Another observation, when considering the first three cases or the last three cases with max-pooling not a factor analysis but with emphasis on the size of filters for the convolutional maps, we noted that the 3 by 3 and 5 by 5 were performing better than 4 by 4. The first two has less difference in terms of the performance between them but the 5 by 5 took a longer time to train than the 3 by 3. For future research, any of these two would be a good option for any network structure in order to achieve better performance. Other future research could also consider 6 by 6, 7 by 7, 8 by 8 or 9 by 9 if the amount of training time is not a major issue. We noted case 1c with 5 by 5 convolutional maps had the best minimum accuracy result at 83.54 % which shows that there is a potential for improvement of the accuracy for a bigger number like 6 by 6 or 7 by 7 if more time of training is allocated for further analysis in the future. The 4 x4 case were behind when max-pooling of 2 by 2 was used but did outperform other cases in the category of 2 by 1 max-pooling. Case 1e did better than case 1d and case 1f which opens door for future research directions especially when training time has no restrictions.

Table 5.1: Comparison between different cases of Architecture 1 with CK+ dataset (%)

Architecture	**Minimum accuracy**	**Average accuracy**	**Maximum accuracy**
a	81.12	**87.50**	**98.11**
b	81.12	85.31	90.57
c	**83.54**	87.46	96.23
d	72.64	77.36	86.79
e	72.64	83.27	97.17
f	72.64	82.70	96.23

Table 5.2 shows the simulations results of architecture 2, which is an increase of one convolutional block in the structure of architecture 1. This change in the design brought improvement in accuracy as we noticed case 2a which ultimately became our found optimal HDNN network structure with optimal parameters. Case 2a managed to achieve stability in accuracy; we tested this 15 times and achieved the same accuracy results of 98.11 %. The

results also confirm our observations in architecture 1 that max-pooling of 2 x 2 was the better option and that 3 x 3 or 5 x 5 size of filters for convolutional maps was also a good choice because case 2a which has 3 x 3 convolutional maps came top of the group with the average accuracy of 98.11 % and case 2c came second with the average accuracy of 86.70 %. The addition of the convolutional block made a huge impact because the 3 x 3 which is case 2a achieved 11.41 % higher average accuracy than the 5 x 5 case 2c. Definitely, case 2a was our best performing HDNN network structure and gave stability of accuracy results. Again like in architecture 1, the same observation repeated itself in architecture 2 whereby case 2e which is a 4 x 4 size filters did perform better than the other cases in the max-pooling of 2 x 1 category recording 81.13 % average accuracy. Hence it confirms our observation that if the amount of training is not restricted case 2e has a potential of improving the accuracy when 2 x 1 max-pooling is applied in the network structure.

Table 5.2: Comparison between different cases of Architecture 2 with CK+ dataset (%)

Architecture	**Minimum accuracy**	**Average accuracy**	**Maximum accuracy**
a	**98.11**	**98.11**	**98.11**
b	72.64	82.70	91.50
c	81.13	86.96	94.34
d	72.64	80.19	95.28
e	72.64	81.13	98.11
f	72.64	80.81	97.17

Table 5.3 shows the accuracy records for architecture 3 which is different from the previous two architectures in the design construction. This architecture consists of three convolutional blocks instead of one, and two in the previous architectures and also we removed the dropout D2 which was all along set at 0.5 in the previous network structures after the fully connected layer, we removed D2 completely from the structure and introduced D1 after the convolutional block B3 and set it at 0.25 to take care of over-fitting in the training stage. We noted that case 3c emerged the winner with average accuracy of 89.94 % and followed by case 3a with average accuracy of 82.64 % which also confirms repeatedly our observations that 3 x 3 or 5 x 5 size of filters for convolutional maps is the best option in a network structure for accuracy improvement and especially when the same numbers of max-pooling like 2 x 2 is applied. Our second observation did not materialize in this group as case 3e which is a 4 x 4 size of filters came second in the 2 x 1 max-pooling category. We can therefore assume that with many convolutional blocks in the hierarchy of architecture, the network structure becomes tricky in the 2 x 1 max-pooling categories. Case 3f emerged the winner in the category of 2 x 1 max-pooling with 80.81 % and also came top of all other cases in the group when considering only the maximum accuracy with accuracy of 97.17 %. This was also the first time when the winner of the group did not also top the group in the maximum accuracy category and was only selected as winner of the group based on the average accuracy results because stability was our key factor when evaluating the performances of our proposed network structures.

Table 5.3: Comparison between different cases of Architecture 3 with CK+ dataset (%)

Architecture	Minimum accuracy	Average accuracy	Maximum accuracy
a	72.64	82.64	90.40
b	72.64	80.61	91.51
c	72.64	**89.94**	96.23
d	72.64	75.26	80.75
e	72.64	78.16	89.20
f	72.64	80.81	**97.17**

Table 5.4 shows the simulations accuracy results for architecture 4. This architecture is similar to architecture 3 except that the dropout D2 is applied in the architecture after the fully connected layer. Therefore, this architecture posseses two dropouts D1 and D2 set at 0.25 and 0.5 respectively. Our first observation still confirms that 3 x 3 or 5 x 5 is the best choice as case 4a emerged victorious with a higher average accuracy of 91.18 % and case 4c came second with average accuracy of 86.70 %. We also noted that with many convolutional blocks in a network structure the training time was longer but still in the scope of our research and again as in the architecture 3 our second observation did not match the two previous architectures results as case 4e came second and case 4d emerged the winner in the max-pooling of 2 x 1 category. Our assumptions remained the same as in architecture 3, the deeper the network structure becomes the tricky the accuracy results in the 2 x 1 or less numbers of max-pooling categories. Also the training time becomes very slow in these categories.

Table 5.4: Comparison between different cases of Architecture 4 with CK+ dataset (%)

Architecture	Minimum accuracy	Average accuracy	Maximum accuracy
a	72.64	86.70	96.23
b	83.02	86.16	94.34
c	**84.91**	**91.18**	**98.11**
d	72.64	83.02	98.11
e	72.64	80.00	94.40
f	72.64	79.24	92.45

To attain our research goal, we repeated the training and testing of the four winners of the four groups to find the network structure with better accuracy and to ensure the results were reliable; therefore we tested six times for each top of the group architecture case. After comparison, CASE 2a emerged as the optimal network architecture with optimal parameters. Table 5.5 shows the comparison accuracy results of the four winners of the four groups. We noted that case 2a performance was good with 98.11 % average accuracy and had consistency

on CK+ dataset and case 4c came second with 91.18 % average accuracy. We also confirm our observations of 3 x 3 or 5 x 5 size filters for convolutional maps as the best option because all the four winners belong in these categories. Two winners belong to 3 x 3 categories including our found optimal HDNN structure and the other two winners belong to 5 x 5 categories.

Table 5.5: Comparison between optimal structures of each architecture case with CK+ dataset (%)

Architecture	**Minimum accuracy**	**Average accuracy**	**Maximum accuracy**
CASE 1a	81.12	87.50	98.11
CASE 2a	**98.11**	**98.11**	**98.11**
CASE 3c	72.64	89.94	96.23
CASE 4c	84.91	91.18	98.11

Experiments on the JAFFE Database: We also found optimal HDNN structure for 15 times on the JAFFE database; Table 5.6 displays the results which show minimum accuracy of 68.75 %, average accuracy of 76.56 % and maximum accuracy of 84.38 % for case 2a which is our optimal HDNN structure while case 3c came second with average accuracy of 72.00 % and maximum accuracy of 81.25 %. Case 1a performed poorly in the JAFFE dataset which was an exception and we assumed that because it is a one convolutional block and the JAFFE dataset contains fewer images than the CK+ dataset might contribute to the poor performance as we noted the training time was quicker than the other cases.

Table 5.6: Comparison between optimal structures of each architecture case with JAFFE dataset (%)

Architecture	**Minimum accuracy**	**Average accuracy**	**Maximum accuracy**
CASE 1a	15.62	35.94	59.38
CASE 2a	**68.75**	**76.56**	**84.38**
CASE 3c	62.50	72.00	81.25
CASE 4c	62.50	68.75	71.88

Figure 5.1 shows the prediction results after the simulations have completed the training of the algorithm. It can be observed that the sad emotion and the disgust emotion confused each other in two instances while the rest of emotions predictions are correct.

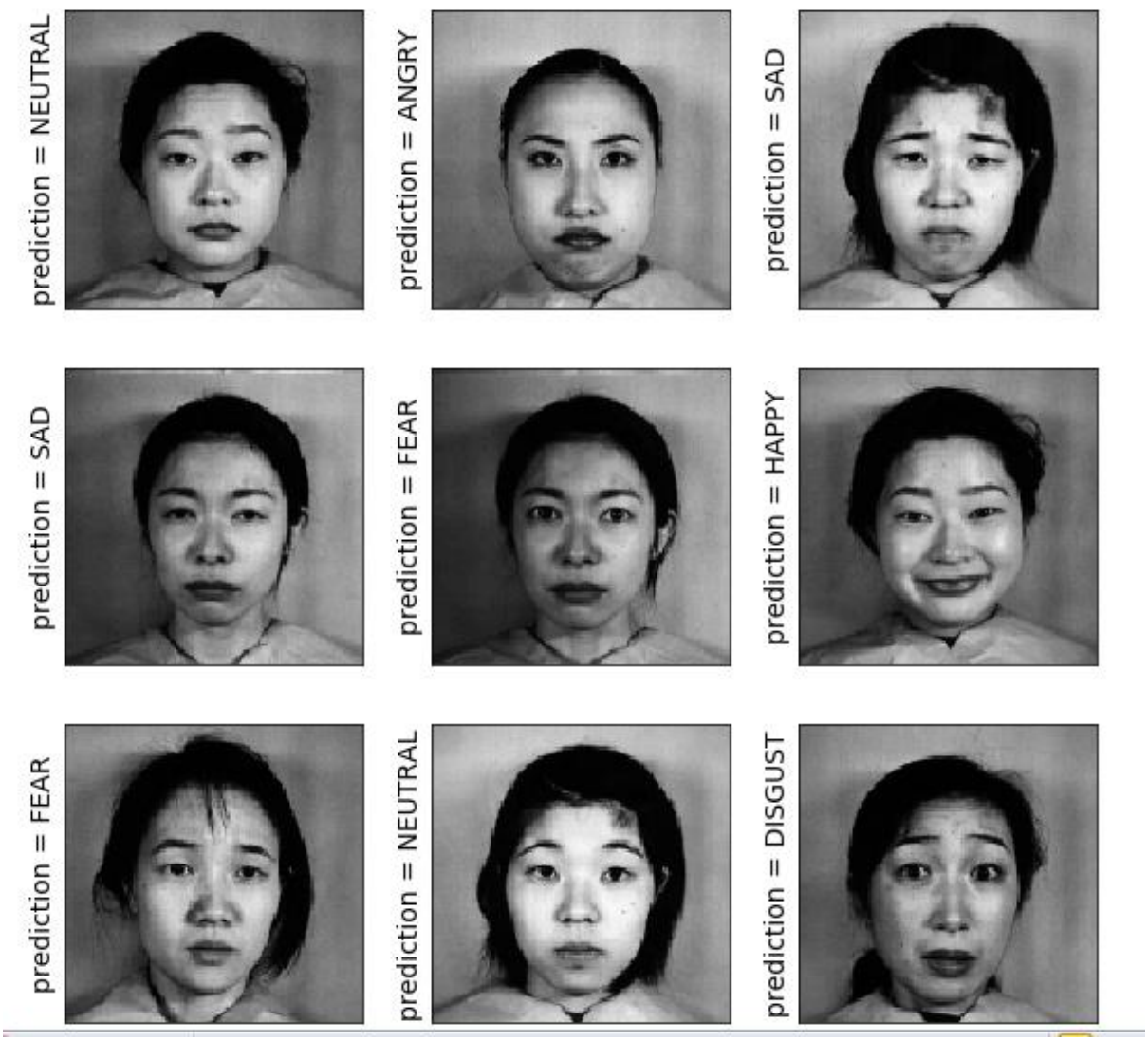

Figure 5.1: Predictions results after classification in pictures.

Figure 5.2 shows the simulation results for our found optimal HDNN structure (case 2a) using the CK+ dataset in relation to the training loss in comparison to the validation loss in a graphical form. It can be observed that the two graphs of validation loss and training loss are converging and there is less difference between the two graphs which shows that our application of the dropout technique carried out its role properly as there is a perfect fitting outcome after training round of 30 epochs. It can also be observed that the validation loss is minimal than the training loss, it is because of the 50% dropout we applied in case 2a which gave the resulting outcome because the system was stronger at the validation time.

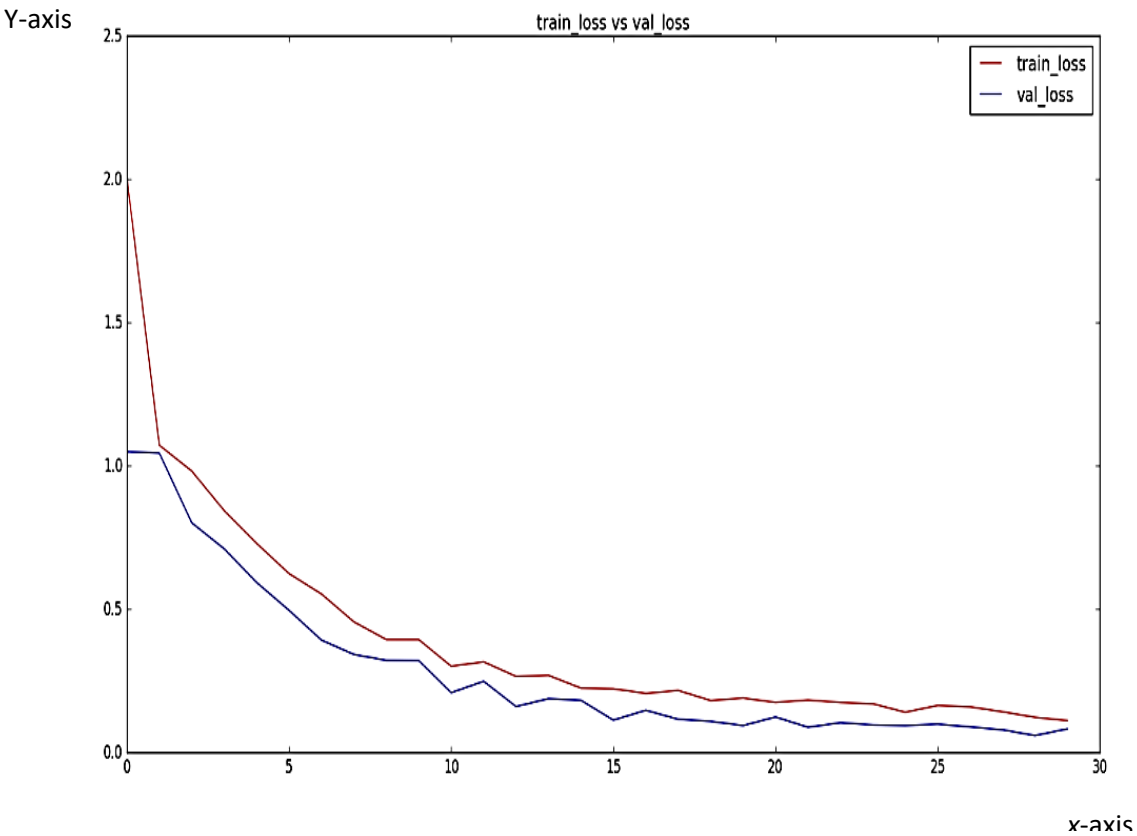

Figure 5.2: Training loss vs Validation loss on CK+.

Figure 5.3 below displays a graphical form of the accuracy results of case 2a which is our found optimal HDNN structure. It can be noticed that the network structure managed to achieve 98.11 % validation accuracy after 30 epochs of training using the CK+ dataset. With careful observation, the validation accuracy is a little bit greater than the training accuracy. It is due to the fact that we used a dropout of 50% in case 2a network structure. The system performed stronger at validation time which gives the outcome of higher validation accuracy.

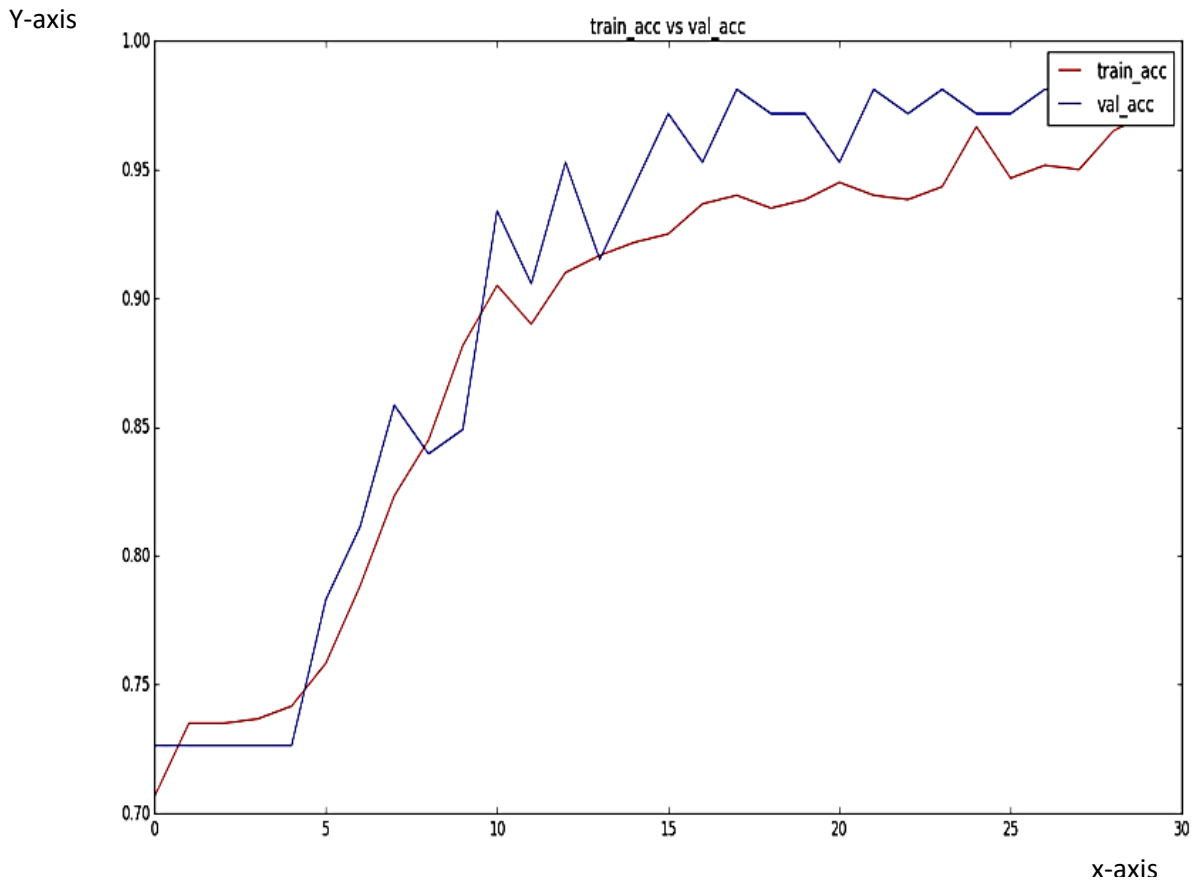

Figure 5.3: Training accuracy vs Validation accuracy on CK+.

When we observe the confusion matrix using the CK+ dataset in Figure 5.4 the surprise emotion and the sad emotion predictions obtained better results with greater accuracy. The neutral emotion and the happy emotion predictions were also acceptable but the angry emotion and the disgust emotion predictions were poor. The network structure did achieve improved validation accuracy with CK+.

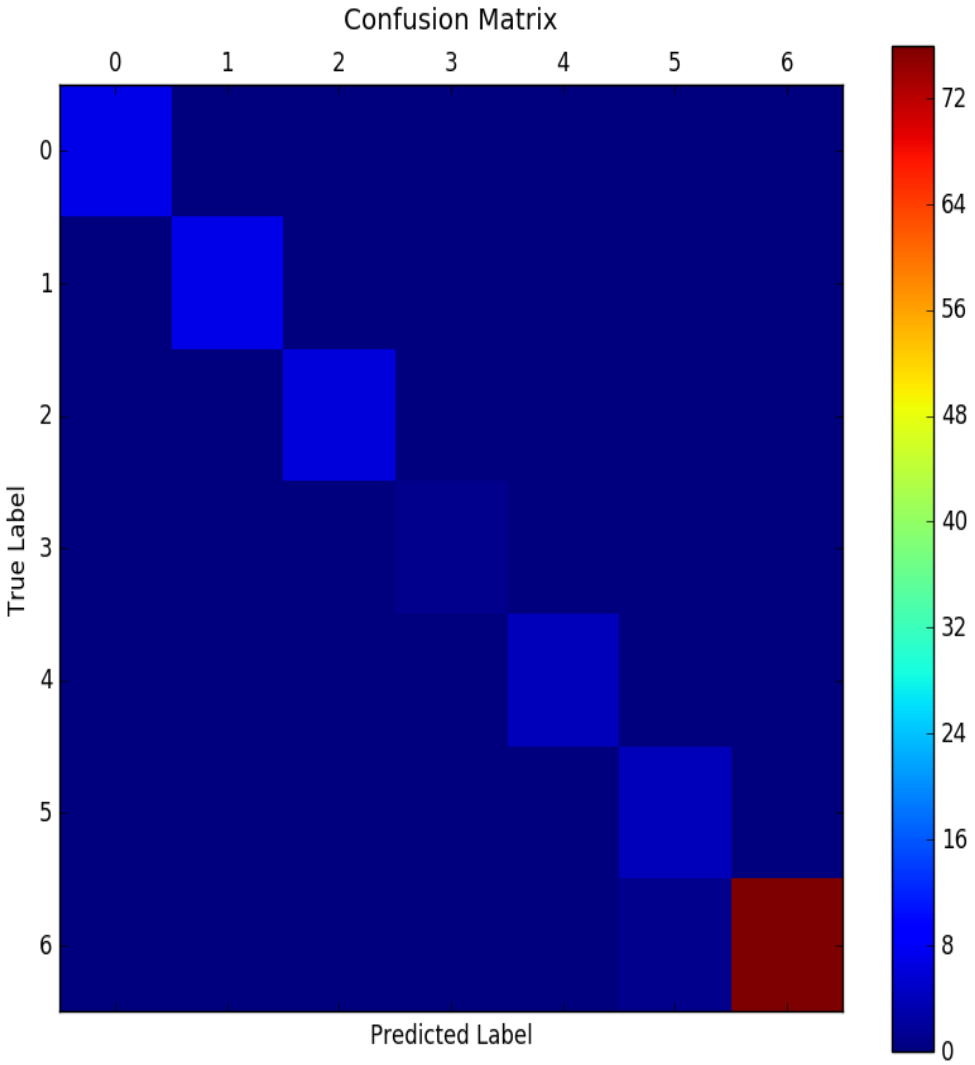

Figure 5.4: Optimal HDNN Structure training output confusion matrix on CK+.

Angry – 0, Disgust – 1, Fear – 2, Happy – 3, Neutral – 4, Sad – 5 and Surprise - 6

We can notice in Figure 5.5 when testing our optimal HDNN structure which is case 2a with the JAFFE dataset that the validation loss is a little larger than the training loss even though we applied the dropout, and the outcome results are the opposite to the ones we had when using the CK+ dataset. These are the effects of fewer dataset because the JAFFE dataset has fewer images than the CK+ dataset because we observed that the training time of JAFFE dataset was quicker which could not allow the dropout to gain momentum at a certain stage

of the epoch to become stronger on the validation time. This outcome can improve with larger dataset as we observed when we were using CK+.

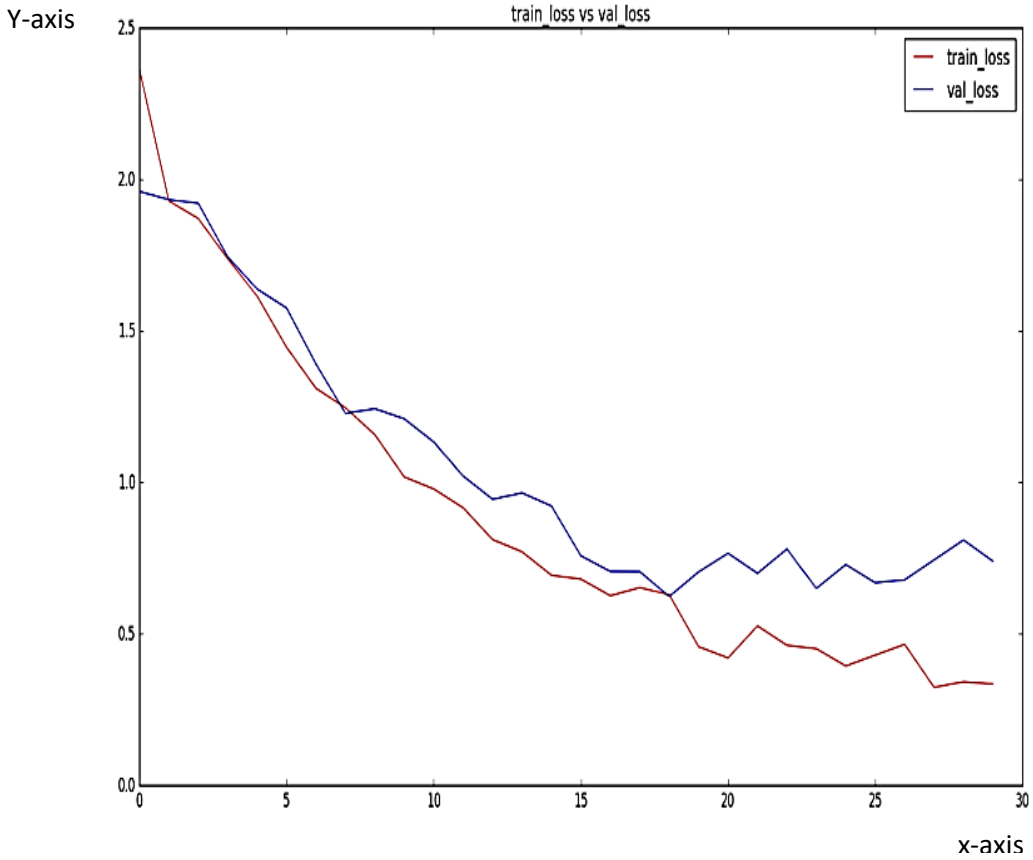

Figure 5.5: Training loss vs Validation loss on JAFFE.

Figure 5.6 below displays the simulation accuracy results of the training accuracy compared to the validation accuracy using the JAFFE dataset. We can acquire the same observations like we did on the validation loss in comparison to the training loss using the JAFFE dataset above. Despite the use of the dropout the validation accuracy is a little bit lower as compared to the training accuracy, in addition the two graphs converged well. It is because the JAFFE dataset is fewer than the CK+ dataset and the quick process of training could allow the effect of the dropout to pick its dominance in the training stage to finish stronger on the validation time.

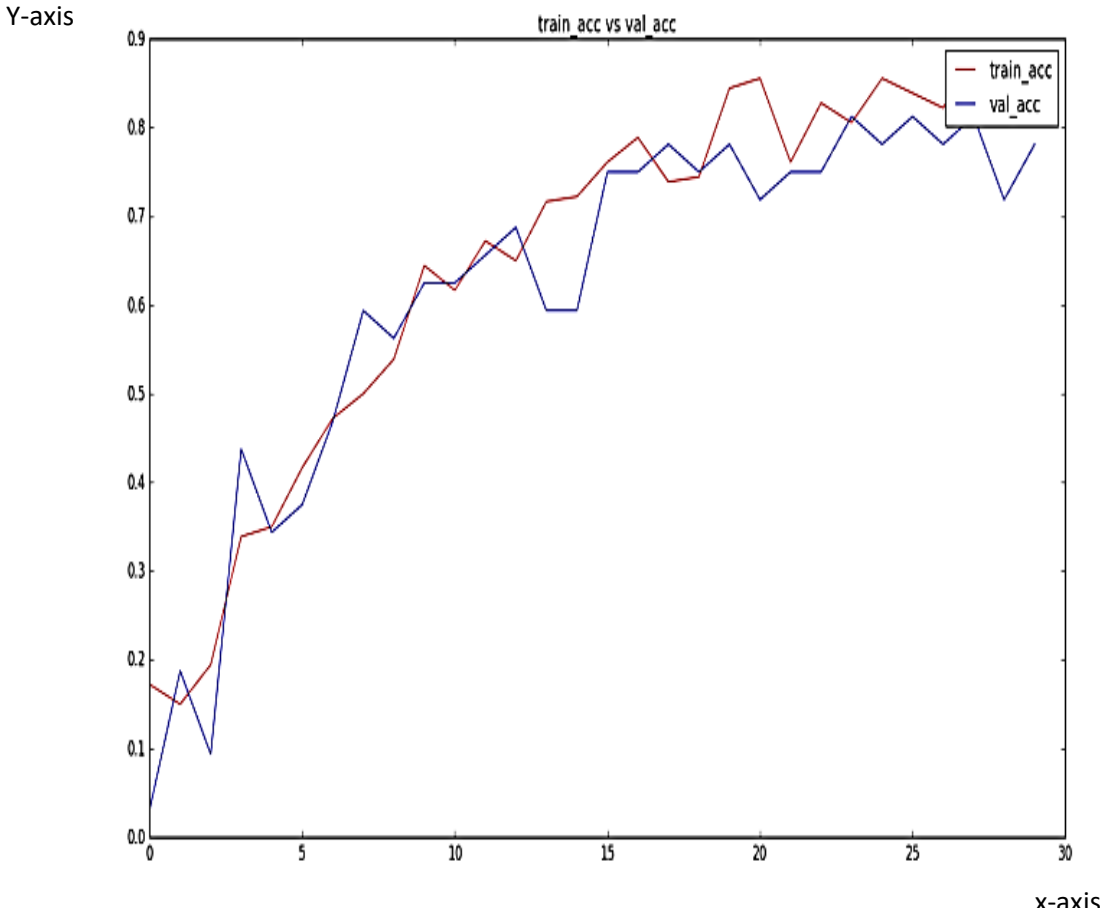

Figure 5.6: Training accuracy vs Validation accuracy on JAFFE.

Confusion matrix illustration of the network structure using the JAFFE dataset is presented in Figure 5.7. The angry emotion and the disgust emotion had confused each other and performed poorly. The fear emotion also got poor accuracy while the happy emotion, the neutral emotion, the sad emotion and the surprise emotion achieved better accuracy. The validation accuracy of the network structure managed to get acceptable results using JAFFE.

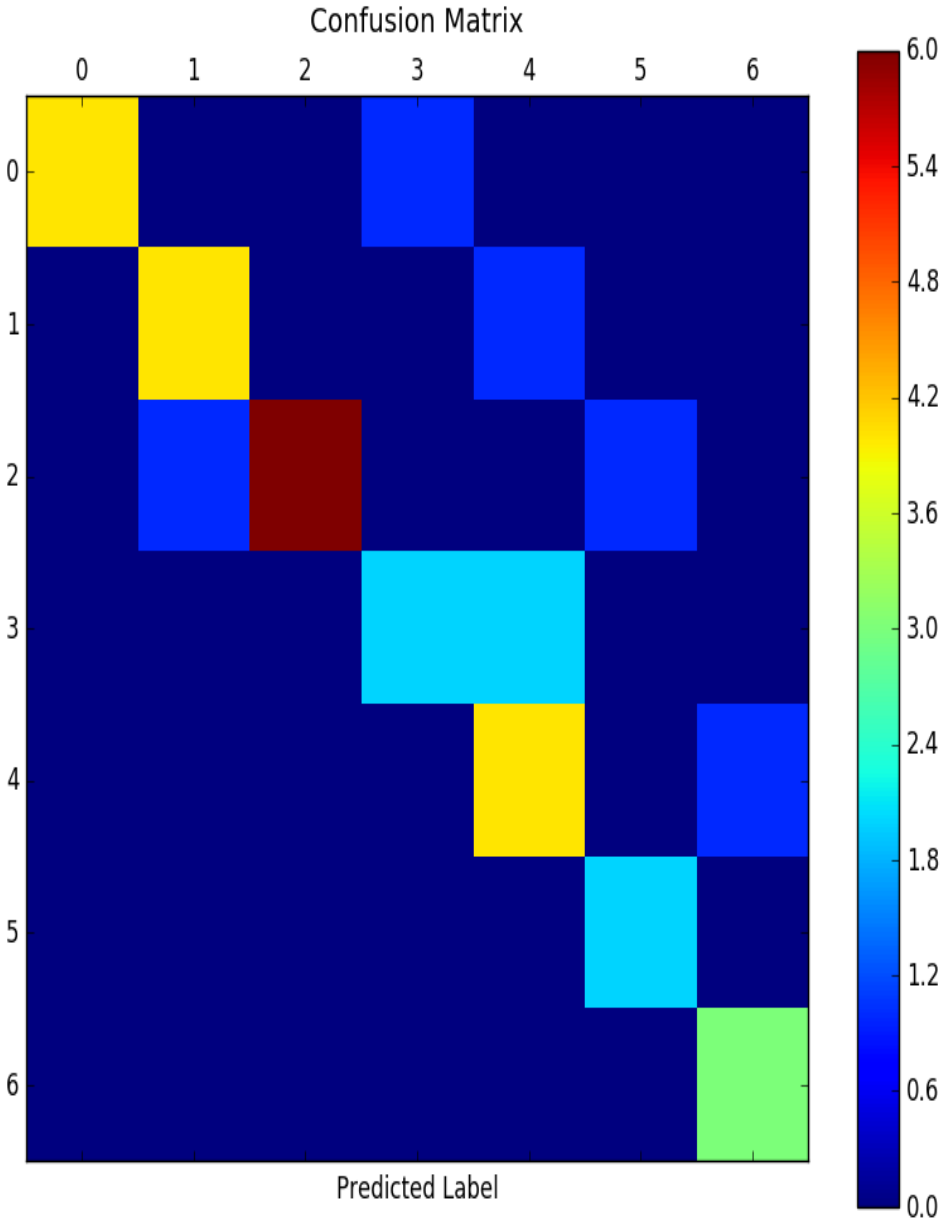

Figure 5.7: Optimal HDNN Structure training output confusion matrix on JAFFE.

Angry – 0, Disgust – 1, Fear – 2, Happy – 3, Neutral – 4, Sad – 5 and Surprise – 6

In order to test the efficiency of our new found HDNN optimal structure and the other three top winners of the groups; we compare them with the latest accuracy records in two recent studies.

We compare the performance of our optimal structures of our cases with the existing models on FER using both datasets of CK+ and Jaffe mentioned in [15]. We will describe the above models in Table 13 and Table 14 as the following:

- Winner of that study: WS
- Appearance feature-based network: M1
- RBM: M2
- Salient Facial Patches: M3
- DCMA-CNNs: M4
- Multi-Level Haar Wavelet: M5
- Salient feature: M6
- CNN: M7

Table 5.7 details the comparison of results of our four best performers of the four groups with existing FER models recently recorded in the above mentioned research using CK+ dataset. The results show that three of our four top models including our found HDNN structure emerged winners with 98.11 % maximum accuracy beating the second existing FER model which achieved 96.46 % accuracy and our last top model of the four winners came third with 96.23 % maximum accuracy which is 0.23 % less the second.

Table 5.7: Comparison between optimal structures of each architecture case with existing architectures for FER with CK+ dataset (%)

Architecture	**Minimum accuracy**	**Average accuracy**	**Maximum accuracy**
M4	-	-	93.46
M3	-	-	94.09
M2	-	-	95.66
M1	-	-	95.15
WS	-	-	96.46
CASE 1a	81.12	87.50	**98.11**
CASE 2a	98.11	98.11	**98.11**
CASE 3c	72.64	89.94	96.23
CASE 4c	84.91	91.18	**98.11**

Table 5.8 shows the recorded accuracy results of the same research mentioned above using the JAFFE dataset in comparison with our four top winners. The existing model which came second when using CK+ dataset managed to beat our top four winners with 91.27 %

accuracy. Our found optimal HDNN structure achieved 84.38 % maximum accuracy while case 3c managed to achieve 81.25 % maximum accuracy.

Table 5.8 Comparison between optimal structures of each architecture case with existing architectures for FER with JAFFE dataset (%)

Architecture	**Minimum accuracy**	**Average accuracy**	**Maximum accuracy**
M7	-	-	84.48
M6	-	-	90.00
M5	-	-	90.56
M1	-	-	89.33
WS	-	-	**91.27**
CASE 1a	15.62	35.94	59.38
CASE 2a	68.75	76.56	84.38
CASE 3c	62.50	72.00	81.25
CASE 4c	62.50	68.75	71.88

Table 5.7 and Table 5.8 displayed the comparison between our proposed network structures with the proposed model and its comparison models in [15]. The results showed that our proposed structures case 1a, case 2b and case 4c outperformed the models using CK+ and when using Jaffe dataset our proposed structure case 2a obtained comparable results with less than 6.89 % to the performed model.

We also compare the performance of our optimal structures of our cases with other existing models on FER using both datasets of CK+ and Jaffe mentioned in [48]. Table 5.9 shows the results of our comparison when using CK+ dataset. We noted that three of our top winners achieved 98.11 % which is 1.22 % less than the top performer of this comparison, and an existing model (Model 7) which achieved 99.33 %.

Table 5.9: Comparison between optimal structures of each architecture case with existing architectures for FER with CK+ dataset (%)

Architecture	**Minimum accuracy**	**Average accuracy**	**Maximum accuracy**
Model 1 in [49]	-	-	95.79
Model 2 in [50]	-	-	99.16
Model 3 in [51]	-	-	83.00
Model 4 in [52]	-	-	90.00 (Around)
Model 5 in [53]	-	-	80.303
Model 6 in [54]	-	-	98.50
Model 7 in [55]	-	-	**99.33**
CASE 1a	81.12	87.50	98.11
CASE 2a	98.11	98.11	98.11
CASE 3c	72.64	89.94	96.23
CASE 4c	84.91	91.18	98.11

Table 5.10 shows the accuracy results of the same research study mentioned above instead by using the JAFFE dataset. Two of our four top winners case 2a and case 3a came third and fourth when compared to the current existing best FER models.

Table 5.10 Comparison between optimal structures of each architecture case with existing architectures for FER with JAFFE dataset (%)

Architecture	**Minimum accuracy**	**Average accuracy**	**Maximum accuracy**
Model 1 in [49]	-	-	53.57
Model 8 in [56]	-	-	**96.10**
Model 5 in [53]	-	-	76.7442
Model 2 in [50]	-	-	87.74
CASE 1a	15.62	35.94	59.38
CASE 2a	68.75	76.56	84.38
CASE 3c	62.50	72.00	81.25
CASE 4c	62.50	68.75	71.88

5.3 SUMMARY

We have demonstrated through this chapter our intensive investigation of the performances of our 24 proposed HDNN structures which were grouped in four categories based on their architecture designs. Each of the four categories had six different HDNN network structures according to their variable parameters. After evaluation of these 24 HDNN structures using the CK+ dataset we managed to achieve four winners of the four categories, case 1a, case 2a, case 3c and case 4c. We compared the four HDNN structures to find our optimal HDNN structure with optimal parameters and case 2a emerged as the best architecture with 98.11 % average accuracy and consistency of the same results after several repeated simulations because our key factor in evaluating architectures was the stability of accuracy results. Architecture could give three different accuracy results after testing. Therefore, we selected the optimal architectures with optimal parameters by assessing only the average accuracy results. Case 4c came second with 91.18 % average accuracy. We also tested our top four architectures using JAFFE. Case 2a still maintains its position as our found optimal HDNN structure with optimal parameters with 76.56 % average accuracy and 84.38 % maximum accuracy. Finally, we compared our four optimal architectures with existing FER models recently published in two different research studies. The tables showed that three of our four optimal architectures came top when using the CK+ dataset and were beaten when using the dataset in one research study and when compared with the other research study. The three optimal architectures obtained comparable results and trailed by 1.22 % of the top existing model. The simulations result and analysis showed that our optimal architectures are efficient when using cost effective hardware. We limited the amount of training time to be less than 120 minutes and will explore adding more time, but not days in order to investigate architectures using different numbers of max-pooling in depth in the future.

CHAPTER 6 CONCLUSIONS AND FUTURE WORK

This chapter gives the final conclusions made after our investigation and answer to the objective of this research study. Future recommendations are also given in this chapter.

6.1 INTRODUCTION

This research study was conducted with the main objective of finding an optimal HDNN structure with optimal parameters which could improve the accuracy using cost effective hardware by investigating the hierarchical deep learning network structure for facial expression recognition. We had to apply our 24 proposed HDNN structures and investigated the results after the simulations were performed using a standard laptop to find the optimal HDNN structure with optimal parameters. Four HDNN structures case 1a, case 2a, case 3c and case 4c emerged as the optimal network structures from the 24 proposed network structures which were separated into four groups. Ultimately, case 2a became the optimal HDNN structure with optimal parameters. This is reinforced by the conclusions in 6.2.

6.2 CONCLUSIONS

We have presented in this research our investigation on FER hierarchical deep neural network structures in search of finding the optimal HDNN structure with optimal parameters to answer to our research problem. We started by creating a generic hierarchical structure with variable settings. This generic structure has a hierarchy of three convolutional blocks, two dropout blocks and one fully connected block. From this generic structure we derived four different network structures to be investigated according to their performances. From each network structure case, we again derived six network structures in relation to the variable parameters. The variable parameters under analysis are the size of filters of the convolutional maps and the max-pooling as well the number of convolutional maps. In total, we had 24 network structures to investigate, six network structures per each case.

After simulations, the results assembled after many repeated experiments showed in the group of case 1; case 1a emerged as the top performer of that group and case 2a, case 3c and case 4c outperformed others in their respective groups. We compared the winners of the 4 groups to find the optimal network structure with optimal parameters. Case 2a answered the research question we were investigating in this study; case 2a network structure outperformed other group winners. Considerations were done when choosing the best network structure, considerations were minimum accuracy, average accuracy and maximum accuracy after 15 times of repeated training and analysis of results.

All our 24 proposed network structures were tested using two most used FER datasets CK+ and JAFFE; we discovered that even the four group winners achieved higher results with CK+ dataset than JAFFE dataset. It might be because that the CK+ dataset have more images than the JAFFE dataset which indicated that our optimal structure would need to be tested on larger datasets for further investigation.

After presenting 24 different network architectures with different parameters for automatic facial expression recognition, we can conclude that our inexpensive optimal network architecture achieved 98.11 % accuracy in the CK+ dataset. We also tested our optimal network architecture with the JAFFE, the results show 84.38 % by using a standard CPU and easier procedures.

We also compared the four group winners with other existing FER models performances recorded in two recent studies [15] [48]. These FER models used the same two datasets, the CK+ and the JAFFE. Three of our four group winners (case 1a, case 2a and case 4c) recorded only less 1.22 % than the top performer model when using the CK+ dataset and two of our network structures case 2a and case 3c came in third, beating other models when using JAFFE dataset. The hardware used for the winner model of the existing models mentioned in Tables 5.7 – 5.10 is better than the hardware we used in this research i.e. i7-8700 CPU is powerful than CPU Intel(R) Core(TM) 2, clock speed (3.20 GHz > 2.40 GHz), RAM (8GB > 4GB) and a GTX 1070 GPU. If our optimal model managed to achieve good performance with an inferior hardware set and without a GPU which is very expensive, the optimal model could achieve improved results if applied with the hardware used by the winner of the existing models. The permutations we chose included a model case 4c that has similar permutations as the winner of the existing models (the convolutional maps: 5x5 and the max-pooling: 2x2) but our optimal model case 2a that has 3x3 convolutional maps and 2x2 max-pooling achieved comparable results with the winner model of the existing models and has achieved better results than model case 4c. Case 2a answered our research question and is the optimal solution.

6.3 FUTURE WORK

After finding the optimal HDNN structure which showed us that it is effective and from our observations during the experiments of this research study, we propose three possibilities for future research:

- Our study focused only on the databases which are publicly available. In future, the optimal HDNN structure can be tested on big FER databases in order to evaluate the generalization ability of the model.
- All our 24 proposed HDNN structures convolutional maps were fixed. Also the neurons'number in the fully connected block did not change but was set at 84 in our research study. Future research can explore with different numbers, hundreds or thousands of neurons can be investigated in future.
- The max-pooling size filters of different numbers for example 1 by 2, 3 by 1, 1 by 3, etc. can be explored and especially if the amount of training time is not restricted in the research but relaxed to allow a window for more hours as acceptable.

LIST OF PUBLICATIONS

1. D. Motembe and Z. Wang, “Analysis of Deep Learning Neural Network Architectures for Facial Expression Recognition,” submitted to *2020 Int. Conf. Neural Comput. Adv. Appl. July 3-5, 2020, Shenzhen, China*, 2020.

REFERENCES

[1] H. Wan, "Deep Learning : Neural Network , Optimizing Method and Libraries Review," *2019 Int. Conf. Robot. Intell. Syst.*, pp. 497–500, 2019.

[2] K. Noda, Y. Yamaguchi, K. Nakadai, H. G. Okuno, and T. Ogata, "Audio-visual speech recognition using deep learning," *Springer Sci. Media New York*, pp. 722–737, 2015.

[3] K. H. Cha, L. Hadjiiski, R. K. Samala, H. Chan, E. M. Caoili, and R. H. Cohan, "Urinary bladder segmentation in CT urography using deep-learning convolutional neural network and level sets," *2016 Am. Assoc. Phys. Med.*, vol. 43, no. 4, pp. 1882–1896, 2016.

[4] D. E. Rumelhart, G. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *1986 Nat.*, vol. 323, pp. 533–536, 1986.

[5] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *2015 Nat.*, vol. 521, pp. 436–444, 2015.

[6] A. Shrestha, "Review of Deep Learning Algorithms and Architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019.

[7] V. Bruce, Z. Henderson, C. Newman, and A. M. Burton, "Matching Identities of Familiar and Unfamiliar Faces Caught on CCTV Images," *J. Exp. Psychol.*, vol. 7, no. 3, pp. 207–218, 2001.

[8] Z. Sun, A. A. Paulino, J. Feng, Z. Chai, T. Tan, and A. K. Jain, "A Study of Multibiometric Traits of Identical Twins," *Biometric Technol. Hum. Identif. VII 7667, 76670T*, pp. 1–12, 2010.

[9] S. Biswas, K. W. Bowyer, and P. J. Flynn, "A Study of Face Recognition of Identical Twins by Humans," *2011 IEEE Int. Work. Inf. Forensics Secur.*, pp. 1–6, 2011.

[10] Z. I. A. Uddin, S. Member, and W. Khaksar, "Facial Expression Recognition Using Salient Features and Convolutional Neural Network," *IEEE Access*, vol. 5, pp. 26146–26161, 2017.

[11] K. Chengeta and S. Viriri, "A Review of Local , Holistic and Deep Learning Approaches in Facial Expressions Recognition," *2019 Conf. Inf. Commun. Technol. Soc.*, pp. 1–7, 2019.

[12] O. M. Way and M. J. Jones, "Robust Real-Time Face Detection," *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 1–18, 2004.

[13] N. Dalal *et al.*, "Histograms of Oriented Gradients for Human Detection To cite this version : HAL Id : inria-00548512 Histograms of Oriented Gradients for Human Detection," *HAL Arch.*, pp. 1–9, 2010.

[14] A. Ade-ibijola and K. Aruleba, "Automatic Attendance Capturing Using Histogram of Oriented Gradients on Facial Images," *2018 IST-Africa Week Conf.*, p. Page 1 of 8-Page 8 of 8, 2018.

[15] J. Kim, B. Kim, S. Member, P. P. Roy, and D. Jeong, “Efficient Facial Expression Recognition Algorithm Based on Hierarchical Deep Neural Network Structure,” *IEEE Access*, vol. 7, pp. 41273–41285, 2019.

[16] I. I. Symposium and A. C. Intelligence, “Face Expression Recognition : a Brief Overview of the Last Decade,” *8th IEEE Int. Symp. Appl. Comput. Intell. Informatics*, no. Section III, pp. 157–161, 2013.

[17] Y. Lv, “Facial expression recognition via deep learning,” *2014 Int. Conf. Smart Comput.*, pp. 303–308, 2014.

[18] I. M. Revina and W. R. S. Emmanuel, “A Survey on Human Face Expression Recognition Techniques,” *J. King Saud Univ. - Comput. Inf. Sci.*, pp. 1–10, 2018.

[19] A. Gibson and J. Patterson, “4. Major Architectures of Deep Networks - Deep Learning [Book],” *Oreilly*, 2019. [Online]. Available: https://www.oreilly.com/library/view/deep-learning/9781491924570/ch04.html. [Accessed: 07-Dec-2019].

[20] “CS231n Convolutional Neural Networks for Visual Recognition.” [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: 28-Dec-2019].

[21] Deeplizard, “Fine-tune VGG16 Image Classifier with Keras | Part 1: Build - YouTube,” *YouTube*, 2017. [Online]. Available: https://www.youtube.com/watch?v=oDHpqu52soI. [Accessed: 07-Dec-2019].

[22] S. Das, “CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more…,” *Medium*, 2017. [Online]. Available: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5. [Accessed: 07-Dec-2019].

[23] “LeNet-5 - A Classic CNN Architecture - engMRK.” [Online]. Available: https://engmrk.com/lenet-5-a-classic-cnn-architecture/. [Accessed: 28-Dec-2019].

[24] “AlexNet: The Architecture that Challenged CNNs - Towards Data Science.” [Online]. Available: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951. [Accessed: 28-Dec-2019].

[25] “Review: ZFNet — Winner of ILSVRC 2013 (Image Classification).” [Online]. Available: https://medium.com/coinmonks/paper-review-of-zfnet-the-winner-of-ilsvlc-2013-image-classification-d1a5a0c45103. [Accessed: 28-Dec-2019].

[26] “A simplified block diagram of the GoogLeNet Architecture. | Download Scientific Diagram.” [Online]. Available: https://www.researchgate.net/figure/A-simplified-block-diagram-of-the-GoogLeNet-Architecture_fig7_292722442. [Accessed: 10-Jan-2020].

[27] “CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more….” [Online]. Available: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5. [Accessed: 28-Dec-2019].

[28] “A simplified architecture of Inception-Resnet-v1 network. The Stem is a... | Download Scientific Diagram.” [Online]. Available: https://www.researchgate.net/figure/A-simplified-architecture-of-Inception-Resnet-v1-

network-The-Stem-is-a-particular_fig2_324850578. [Accessed: 16-Jan-2020].

[29] B. C. Ko, "A Brief Review of Facial Emotion Recognition Based," *Sensors*, pp. 1–20, 2018.

[30] M. Liu, R. Wang, S. Li, S. Shan, Z. Huang, and X. Chen, "Combining Multiple Kernel Methods on Riemannian Manifold for Emotion Recognition in the Wild," *Proc. 16th Int. Conf. multimodal Interact.*, pp. 494–501, 2014.

[31] F. Petroski, S. Vashisht, M. Edoardo, C. Joel, L. Kenneth, and O. S. Jeff, "Deep Neuroevolution : Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning," *arXiv*, pp. 1–16, 2017.

[32] P. Burkert, F. Trier, M. Z. Afzal, A. Dengel, and M. Liwicki, "DeXpression : Deep Convolutional Neural Network for Expression Recognition," *arXiv*, no. 1–8, pp. 1–8, 2016.

[33] K. Zhao and W. C. Honggang, "Deep Region and Multi-label Learning for Facial Action Unit Detection," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 3391–3399, 2016.

[34] R. Breuer and R. Kimmel, "arXiv : 1705 . 01842v2 [cs . CV] 10 May 2017 A Deep Learning Perspective on the Origin of Facial Expressions," *arXiv Prepr. arXiv 1705.01842*, pp. 1–16, 2017.

[35] R. Walecki, O. O. Rudovic, V. Pavlovic, and M. Pantic, "Deep Structured Learning for Facial Action Unit Intensity Estimation," *2017 Comput. Vis. Found.*, pp. 3405–3414, 2017.

[36] V. Negi, S. Mann, and V. Chauhan, "Devanagari Character Recognition Using Artificial Neural Network," *2017 Int. J. Eng. Technol.*, vol. 9, no. August, pp. 2161–2167, 2017.

[37] "Examples from the CK+ dataset [10] illustrating the strong temporal... | Download Scientific Diagram." [Online]. Available: https://www.researchgate.net/figure/Examples-from-the-CK-dataset-10-illustrating-the-strong-temporal-links-present-within_fig1_221411582. [Accessed: 29-Dec-2019].

[38] "Example of images from JAFFE facial expression dataset. | Download Scientific Diagram." [Online]. Available: https://www.researchgate.net/figure/Example-of-images-from-JAFFE-facial-expression-dataset_fig1_336287978. [Accessed: 29-Dec-2019].

[39] "Welcome — Theano 1.0.0 documentation." [Online]. Available: http://deeplearning.net/software/theano/. [Accessed: 29-Dec-2019].

[40] "OpenCV: Face Detection using Haar Cascades." [Online]. Available: https://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html# gsc.tab=0 [. [Accessed: 29-Dec-2019].

[41] "404 — Anaconda documentation." [Online]. Available: https://docs.continuum.io/anaconda/index. [Accessed: 29-Dec-2019].

[42] "CS231n Convolutional Neural Networks for Visual Recognition." [Online].

Available: http://cs231n.github.io/. [Accessed: 28-Dec-2019].

[43] “Understand the Impact of Learning Rate on Neural Network Performance.” [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/. [Accessed: 28-Dec-2019].

[44] “Difference Between a Batch and an Epoch in a Neural Network.” [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/. [Accessed: 28-Dec-2019].

[45] “Gradient Descent For Machine Learning.” [Online]. Available: https://machinelearningmastery.com/gradient-descent-for-machine-learning/. [Accessed: 28-Dec-2019].

[46] “Max-pooling / Pooling - Computer Science Wiki.” [Online]. Available: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling. [Accessed: 28-Dec-2019].

[47] “CS 230 - Convolutional Neural Networks Cheatsheet.” [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks. [Accessed: 28-Dec-2019].

[48] A. S. Vyas and A. F. Expressions, “Survey on Face Expression Recognition using CNN,” *2019 5th Int. Conf. Adv. Comput. Commun. Syst.*, pp. 102–106, 2019.

[49] A. Teixeira, E. De Aguiar, and A. F. De Souza, “Facial Expression Recognition with Convolutional Neural Networks : Coping with Few Data and the Training Sample Order,” *Pattern Recognit.*, pp. 1–54, 2016.

[50] X. Chen, X. Yang, M. Wang, and J. Zou, “Convolution neural network for automatic facial expression recognition,” *2017 Int. Conf. Appl. Syst. Innov.*, pp. 814–817, 2017.

[51] M. Li, “A Deep-Learning Approach to Facial Expression Recognition,” *2015 14th IAPR Int. Conf. Mach. Vis. Appl.*, pp. 279–282, 2015.

[52] R. K. G. A, R. K. Kumar, and G. Sanyal, “Facial Emotion Analysis using Deep Convolution Neural Network,” *Int. Conf. Signal Process. Commun.*, no. July, pp. 369–374, 2017.

[53] K. Shan, J. Guo, W. You, D. Lu, and R. Bie, “Automatic Facial Expression Recognition Based on a Deep Convolutional-Neural-Network Structure,” *2017 IEEE SERA 2017, June 7-9, 2017, London, UK*, pp. 123–128, 2017.

[54] E. S. Networks *et al.*, “Facial Expression Recognition Based on Deep,” *IEEE Trans. IMAGE Process.*, vol. 26, no. 9, pp. 4193–4203, 2017.

[55] A. Fathallah and A. Douik, “Facial Expression Recognition via Deep Learning,” *2017 IEEE/ACS 14th Int. Conf. Comput. Syst. Appl.*, pp. 745–750, 2017.

[56] A. Uçar, “Deep Convolutional Neural Networks for Facial Expression Recognition,” *2017 IEEE*, pp. 1–5, 2017.

APPENDICES

Appendix A Python codes using FER dataset CK+

```
# File: Deep Learning
# Author: Dodi Motembe:University of South Africa (UNISA)
# Student No: 50685740
# Department of Electrical and Mining Engineering
# College of Science, Engineering and Technology
# University Of South Africa


# we start by importing the necessary Python libraries needed for our
# model to work

from sklearn.metrics import confusion_matrix
from keras import callbacks
from PIL import Image
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import SGD, RMSprop, adam
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.models import Sequential
from keras import backend as K
from keras.utils import plot_model
from keras.utils import np_utils
import keras
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from pylab import rcParams
rcParams['figure.figsize'] = 20, 10


# Datapath needs to be defined properly so that images from the dataset
# can be found and be used

data_path = 'C:\\Users\\DODI\\Workspace\\Python\\ck\\dataset'
data_dir_list = os.listdir(data_path)
img_rows = 256
img_cols = 256
num_channel = 1
num_epoch = 10
img_data_list = []

for dataset in data_dir_list:
    img_list = os.listdir(data_path + '\\' + dataset)
    print('Loaded the images of dataset-' + '{}\n'.format(dataset))
    for img in img_list:
        input_img = cv2.imread(data_path + '\\' + dataset + '\\' + img)
        #input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
        input_img_resize = cv2.resize(input_img, (128, 128))
        img_data_list.append(input_img_resize)


img_data = np.array(img_data_list)
img_data = img_data.astype('float32')
img_data = img_data / 255
img_data.shape

# The number of classes needs to be defined properly for the recognition
# of the different classes to be done


num_classes = 7
```

```
num_of_samples = img_data.shape[0]
labels = np.ones((num_of_samples,), dtype='int64')
labels[0:29] = 0  # 30
labels[30:59] = 1  # 29
labels[60:92] = 2  # 32
labels[93:124] = 3  # 31
labels[125:155] = 4  # 30
labels[156:187] = 5  # 31
labels[188:] = 6  # 30

names = ['ANGRY', 'DISGUST', 'FEAR', 'HAPPY', 'NEUTRAL', 'SAD', 'SURPRISE']


def getLabel(id):
    return [
        'ANGRY',
        'DISGUST',
        'FEAR',
        'HAPPY',
        'NEUTRAL',
        'SAD',
        'SURPRISE'][id]


# Conversion of class labels to on-hot encoding

Y = np_utils.to_categorical(labels, num_classes)
x, y = shuffle(img_data, Y, random_state=2)
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.15, random_state=2)


# Our optimal HDNN Structure with optimal parameters


input_shape = img_data[0].shape


model = Sequential()

model.add(Convolution2D(6, 3, 3, input_shape=input_shape, border_mode='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(16, 3, 3, border_mode='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(84))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compilation of the optimal HDNN Structure with optimal parameters

model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=["accuracy"])

# We can visualize the optimal HDNN Structure with optimal parameters and
# acquire details of its construction

model.summary()


model.get_config()

model.layers[0].get_config()
```

```
model.layers[0].input_shape

model.layers[0].output_shape

model.layers[0].get_weights()

np.shape(model.layers[0].get_weights()[0])

model.layers[0].trainable


# The Training process of our optimal HDNN Structure with optimal parameters

filename = 'model_train_new.csv'
filepath = "Best-weights-my_model-{epoch:03d}-{loss:.4f}-{acc:.4f}.hdf5"
csv_log = callbacks.CSVLogger(filename, separator=',', append=False)
checkpoint = callbacks.ModelCheckpoint(
    filepath,
    monitor='val_loss',
    verbose=1,
    save_best_only=True,
    mode='min')
callbacks_list = [csv_log, checkpoint]
callbacks_list = [csv_log]


hist = model.fit(
    X_train,
    y_train,
    batch_size=7,
    nb_epoch=30,
    verbose=1,
    validation_data=(
        X_test,
        y_test),
    callbacks=callbacks_list)

model.save_weights('model_weights.h5')
model.save('model_keras.h5')

# We can visualize our optimal HDNN Structure with optimal parameters loss
# and accuracy through graphs


train_loss = hist.history['loss']
val_loss = hist.history['val_loss']
train_acc = hist.history['accuracy']
val_acc = hist.history['val_accuracy']

epochs = range(len(train_acc))
plt.plot(epochs, train_loss, 'r', label='train_loss')


plt.plot(epochs, val_loss, 'b', label='val_loss')


plt.title('train_loss vs val_loss')


plt.legend()

plt.figure()


plt.plot(epochs, train_acc, 'r', label='train_acc')


plt.plot(epochs, val_acc, 'b', label='val_acc')
```

```
plt.title('train_acc vs val_acc')


plt.legend()


plt.figure()


# The Evaluation of our optimal HDNN Structure with optimal parameters


score = model.evaluate(X_test, y_test, verbose=0)

print('Test Loss:', score[0])


print('Test accuracy:', score[1])


test_image = X_test[0:1]
print(test_image.shape)


print(model.predict(test_image))


print(model.predict_classes(test_image))


print(y_test[0:1])


res = model.predict_classes(X_test[:9])
plt.figure(figsize=(10, 10))

for i in range(0, 9):
    plt.subplot(330 + 1 + i)
    plt.imshow(X_test[i], cmap=plt.get_cmap('gray'))
    plt.gca().get_xaxis().set_ticks([])
    plt.gca().get_yaxis().set_ticks([])
    plt.ylabel('prediction = %s' % getLabel(res[i]), fontsize=14)
    plt.show()

# We can visualize our optimal HDNN Structure with optimal parameters
# confusion matrix

results = model.predict_classes(X_test)
cm = confusion_matrix(np.where(y_test == 1)[1], results)
plt.matshow(cm)


plt.title('Confusion Matrix')


plt.colorbar()

plt.ylabel('True Label')


plt.xlabel('Predicted Label')


plt.show()

plt.gca().get_xaxis().set_ticks([])

plt.gca().get_yaxis().set_ticks([])

plt.xlabel('prediction = %s' % getLabel(results[0]), fontsize=25)
```

```
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1900 64 bit (AMD64)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/DODI/Naisha 2.py
==========
Using Theano backend.
Loaded the images of dataset-ANGRY

Loaded the images of dataset-DISGUST

Loaded the images of dataset-FEAR

Loaded the images of dataset-HAPPY

Loaded the images of dataset-NEUTRAL

Loaded the images of dataset-SAD

Loaded the images of dataset-SURPRISE


Warning (from warnings module):
  File "C:/Users/DODI/Naisha 2.py", line 106
    model.add(Convolution2D(6, 3, 3, input_shape=input_shape, border_mode='same'))
UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(6, (3, 3), padding="same",
input_shape=(128, 128,...)`

Warning (from warnings module):
  File "C:/Users/DODI/Naisha 2.py", line 110
    model.add(Convolution2D(16, 3, 3, border_mode='same'))
UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(16, (3, 3), padding="same")`
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 6)       168
_________________________________________________________________
activation_1 (Activation)    (None, 128, 128, 6)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 16)        880
_________________________________________________________________
activation_2 (Activation)    (None, 64, 64, 16)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 16)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_1 (Dense)              (None, 84)                1376340
_________________________________________________________________
activation_3 (Activation)    (None, 84)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 84)                0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 595
_________________________________________________________________
activation_4 (Activation)    (None, 7)                 0
=================================================================
Total params: 1,377,983
Trainable params: 1,377,983
Non-trainable params: 0
_________________________________________________________________

Warning (from warnings module):
  File "C:/Users/DODI/Naisha 2.py", line 173
    callbacks=callbacks_list)
UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
Train on 600 samples, validate on 106 samples
Epoch 30/30
```

```
  7/600 [..............................] - ETA: 18s - loss: 0.3574 - accuracy: 0.8571
 14/600 [..............................] - ETA: 19s - loss: 0.1788 - accuracy: 0.9286
 21/600 [>.............................] - ETA: 20s - loss: 0.1419 - accuracy: 0.9524
 28/600 [>.............................] - ETA: 21s - loss: 0.1099 - accuracy: 0.9643
 35/600 [>.............................] - ETA: 21s - loss: 0.1263 - accuracy: 0.9429
 42/600 [=>............................] - ETA: 22s - loss: 0.1334 - accuracy: 0.9524
 49/600 [=>............................] - ETA: 22s - loss: 0.1163 - accuracy: 0.9592
 56/600 [=>............................] - ETA: 22s - loss: 0.1745 - accuracy: 0.9464
 63/600 [==>...........................] - ETA: 22s - loss: 0.1573 - accuracy: 0.9524
 70/600 [==>...........................] - ETA: 22s - loss: 0.1943 - accuracy: 0.9286
 77/600 [==>...........................] - ETA: 22s - loss: 0.1779 - accuracy: 0.9351
 84/600 [===>..........................] - ETA: 21s - loss: 0.1716 - accuracy: 0.9405
 91/600 [===>..........................] - ETA: 21s - loss: 0.1585 - accuracy: 0.9451
 98/600 [===>..........................] - ETA: 21s - loss: 0.1514 - accuracy: 0.9490
105/600 [====>.........................] - ETA: 21s - loss: 0.1561 - accuracy: 0.9524
112/600 [====>.........................] - ETA: 21s - loss: 0.1847 - accuracy: 0.9464
119/600 [====>.........................] - ETA: 21s - loss: 0.1750 - accuracy: 0.9496
126/600 [=====>........................] - ETA: 21s - loss: 0.1653 - accuracy: 0.9524
133/600 [=====>........................] - ETA: 21s - loss: 0.1708 - accuracy: 0.9474
140/600 [======>.......................] - ETA: 21s - loss: 0.1721 - accuracy: 0.9429
147/600 [======>.......................] - ETA: 21s - loss: 0.1640 - accuracy: 0.9456
154/600 [======>.......................] - ETA: 21s - loss: 0.1644 - accuracy: 0.9416
161/600 [=======>......................] - ETA: 21s - loss: 0.1797 - accuracy: 0.9379
168/600 [=======>......................] - ETA: 21s - loss: 0.1726 - accuracy: 0.9405
175/600 [=======>......................] - ETA: 21s - loss: 0.1668 - accuracy: 0.9429
Warning (from warnings module):
  File "C:\Users\DODI\Anaconda3\lib\site-packages\keras\callbacks\callbacks.py", line 95
    % (hook_name, delta_t_median), RuntimeWarning)
RuntimeWarning: Method (on_train_batch_end) is slow compared to the batch update (0.205047). Check
your callbacks.

182/600 [========>.....................] - ETA: 21s - loss: 0.1641 - accuracy: 0.9451
189/600 [========>.....................] - ETA: 20s - loss: 0.1586 - accuracy: 0.9471
196/600 [========>.....................] - ETA: 20s - loss: 0.1602 - accuracy: 0.9439
203/600 [=========>....................] - ETA: 19s - loss: 0.1744 - accuracy: 0.9360
210/600 [=========>....................] - ETA: 19s - loss: 0.1688 - accuracy: 0.9381
217/600 [=========>....................] - ETA: 18s - loss: 0.1657 - accuracy: 0.9401
224/600 [==========>...................] - ETA: 18s - loss: 0.1607 - accuracy: 0.9420
231/600 [==========>...................] - ETA: 18s - loss: 0.1581 - accuracy: 0.9437
238/600 [==========>...................] - ETA: 17s - loss: 0.1539 - accuracy: 0.9454
245/600 [===========>..................] - ETA: 17s - loss: 0.1496 - accuracy: 0.9469
252/600 [===========>..................] - ETA: 17s - loss: 0.1627 - accuracy: 0.9444
259/600 [===========>..................] - ETA: 16s - loss: 0.1730 - accuracy: 0.9421
266/600 [============>.................] - ETA: 16s - loss: 0.1701 - accuracy: 0.9436
273/600 [============>.................] - ETA: 16s - loss: 0.1684 - accuracy: 0.9451
280/600 [=============>................] - ETA: 15s - loss: 0.1682 - accuracy: 0.9464
287/600 [=============>................] - ETA: 15s - loss: 0.1660 - accuracy: 0.9477
294/600 [=============>................] - ETA: 15s - loss: 0.1624 - accuracy: 0.9490
301/600 [==============>...............] - ETA: 15s - loss: 0.1616 - accuracy: 0.9502
308/600 [==============>...............] - ETA: 14s - loss: 0.1581 - accuracy: 0.9513
315/600 [==============>...............] - ETA: 14s - loss: 0.1549 - accuracy: 0.9524
322/600 [===============>..............] - ETA: 14s - loss: 0.1520 - accuracy: 0.9534
329/600 [===============>..............] - ETA: 14s - loss: 0.1519 - accuracy: 0.9544
336/600 [===============>..............] - ETA: 13s - loss: 0.1494 - accuracy: 0.9554
343/600 [================>.............] - ETA: 13s - loss: 0.1541 - accuracy: 0.9534
Warning (from warnings module):
  File "C:\Users\DODI\Anaconda3\lib\site-packages\keras\callbacks\callbacks.py", line 95
    % (hook_name, delta_t_median), RuntimeWarning)
RuntimeWarning: Method (on_train_batch_end) is slow compared to the batch update (0.227555). Check
your callbacks.

350/600 [================>.............] - ETA: 13s - loss: 0.1516 - accuracy: 0.9543
Warning (from warnings module):
  File "C:\Users\DODI\Anaconda3\lib\site-packages\keras\callbacks\callbacks.py", line 95
    % (hook_name, delta_t_median), RuntimeWarning)
RuntimeWarning: Method (on_train_batch_end) is slow compared to the batch update (0.224054). Check
your callbacks.

357/600 [================>.............] - ETA: 12s - loss: 0.1502 - accuracy: 0.9552
364/600 [=================>............] - ETA: 12s - loss: 0.1496 - accuracy: 0.9560
```

```
371/600 [=================>............] - ETA: 11s - loss: 0.1554 - accuracy: 0.9542
378/600 [=================>............] - ETA: 11s - loss: 0.1529 - accuracy: 0.9550
385/600 [==================>...........] - ETA: 11s - loss: 0.1530 - accuracy: 0.9558
392/600 [==================>...........] - ETA: 10s - loss: 0.1574 - accuracy: 0.9541
399/600 [==================>...........] - ETA: 10s - loss: 0.1554 - accuracy: 0.9549
406/600 [===================>..........] - ETA: 10s - loss: 0.1537 - accuracy: 0.9557
413/600 [===================>..........] - ETA: 9s - loss: 0.1511 - accuracy: 0.9564
420/600 [====================>.........] - ETA: 9s - loss: 0.1545 - accuracy: 0.9548
427/600 [====================>.........] - ETA: 8s - loss: 0.1526 - accuracy: 0.9555
434/600 [====================>.........] - ETA: 8s - loss: 0.1549 - accuracy: 0.9539
441/600 [=====================>........] - ETA: 8s - loss: 0.1526 - accuracy: 0.9546
448/600 [=====================>........] - ETA: 7s - loss: 0.1513 - accuracy: 0.9554
455/600 [=====================>........] - ETA: 7s - loss: 0.1493 - accuracy: 0.9560
462/600 [======================>.......] - ETA: 7s - loss: 0.1537 - accuracy: 0.9545
469/600 [======================>.......] - ETA: 6s - loss: 0.1521 - accuracy: 0.9552
476/600 [======================>.......] - ETA: 6s - loss: 0.1498 - accuracy: 0.9559
483/600 [=======================>......] - ETA: 6s - loss: 0.1484 - accuracy: 0.9565
490/600 [=======================>......] - ETA: 5s - loss: 0.1464 - accuracy: 0.9571
497/600 [=======================>......] - ETA: 5s - loss: 0.1484 - accuracy: 0.9557
504/600 [========================>.....] - ETA: 5s - loss: 0.1467 - accuracy: 0.9563
511/600 [========================>.....] - ETA: 4s - loss: 0.1453 - accuracy: 0.9569
518/600 [========================>.....] - ETA: 4s - loss: 0.1460 - accuracy: 0.9556
525/600 [=========================>....] - ETA: 4s - loss: 0.1477 - accuracy: 0.9562
Warning (from warnings module):
  File "C:\Users\DODI\Anaconda3\lib\site-packages\keras\callbacks\callbacks.py", line 95
    % (hook_name, delta_t_median), RuntimeWarning)
RuntimeWarning: Method (on_train_batch_end) is slow compared to the batch update (0.230552). Check
your callbacks.

532/600 [=========================>....] - ETA: 3s - loss: 0.1459 - accuracy: 0.9568
539/600 [=========================>....] - ETA: 3s - loss: 0.1445 - accuracy: 0.9573
546/600 [==========================>...] - ETA: 2s - loss: 0.1451 - accuracy: 0.9560
553/600 [==========================>...] - ETA: 2s - loss: 0.1442 - accuracy: 0.9566
560/600 [==========================>..] - ETA: 2s - loss: 0.1446 - accuracy: 0.9554
567/600 [==========================>..] - ETA: 1s - loss: 0.1480 - accuracy: 0.9541
574/600 [==========================>..] - ETA: 1s - loss: 0.1463 - accuracy: 0.9547
581/600 [===========================>.] - ETA: 1s - loss: 0.1465 - accuracy: 0.9552
588/600 [===========================>.] - ETA: 0s - loss: 0.1463 - accuracy: 0.9541
595/600 [===========================>.] - ETA: 0s - loss: 0.1501 - accuracy: 0.9513
600/600 [==============================] - 33s 56ms/step - loss: 0.1515 - accuracy:
0.9500 - val_loss: 0.0653 - val_accuracy: 0.9811
Test Loss: 0.06534121902483814
Test accuracy: 0.9811320900917053
(1, 128, 128, 3)
[[  3.04679908e-08   1.07606972e-10   1.08493807e-13   9.51358065e-14
    4.66445051e-08   3.75608485e-12   9.99999940e-01]]
[6]
[[ 0.  0.  0.  0.  0.  0.  1.]]
```

Appendix B: Python codes using FER dataset JAFFE

```
# File: Deep Learning
# Author: Dodi Motembe:University of South Africa (UNISA)
# Student No: 50685740
# Department of Electrical and Mining Engineering
# College of Science, Engineering and Technology
# University Of South Africa


# we start by importing the necessary Python libraries needed for our
# model to work

from sklearn.metrics import confusion_matrix
from keras import callbacks
from PIL import Image
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import SGD, RMSprop, adam
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.layers.core import Dense, Dropout, Activation, Flatten
```

```
from keras.models import Sequential
from keras import backend as K
from keras.utils import plot_model
from keras.utils import np_utils
import keras
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from pylab import rcParams
rcParams['figure.figsize'] = 20, 10


# Datapath needs to be defined properly so that images from the dataset
# can be found and be used

data_path = 'C:\\Users\\DODI\\Workspace\\Python\\jaffe\\dataset'
data_dir_list = os.listdir(data_path)
img_rows = 256
img_cols = 256
num_channel = 1
num_epoch = 10
img_data_list = []

for dataset in data_dir_list:
    img_list = os.listdir(data_path + '\\' + dataset)
    print('Loaded the images of dataset-' + '{}\n'.format(dataset))
    for img in img_list:
        input_img = cv2.imread(data_path + '\\' + dataset + '\\' + img)
        #input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
        input_img_resize = cv2.resize(input_img, (128, 128))
        img_data_list.append(input_img_resize)


img_data = np.array(img_data_list)
img_data = img_data.astype('float32')
img_data = img_data / 255
img_data.shape

# The number of classes needs to be defined properly for the recognition
# of the different classes to be done


num_classes = 7
num_of_samples = img_data.shape[0]
labels = np.ones((num_of_samples,), dtype='int64')
labels[0:29] = 0  # 30
labels[30:59] = 1  # 29
labels[60:92] = 2  # 32
labels[93:124] = 3  # 31
labels[125:155] = 4  # 30
labels[156:187] = 5  # 31
labels[188:] = 6  # 30

names = ['ANGRY', 'DISGUST', 'FEAR', 'HAPPY', 'NEUTRAL', 'SAD', 'SURPRISE']


def getLabel(id):
    return [
        'ANGRY',
        'DISGUST',
        'FEAR',
        'HAPPY',
        'NEUTRAL',
        'SAD',
        'SURPRISE'][id]
```

```
# Conversion of class labels to on-hot encoding

Y = np_utils.to_categorical(labels, num_classes)
x, y = shuffle(img_data, Y, random_state=2)
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.15, random_state=2)


# Our optimal HDNN Structure with optimal parameters


input_shape = img_data[0].shape


model = Sequential()

model.add(Convolution2D(6, 3, 3, input_shape=input_shape, border_mode='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(16, 3, 3, border_mode='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(84))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compilation of the optimal HDNN Structure with optimal parameters

model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=["accuracy"])

# We can visualize the optimal HDNN Structure with optimal parameters and
# acquire details of its construction

model.summary()


model.get_config()

model.layers[0].get_config()

model.layers[0].input_shape

model.layers[0].output_shape

model.layers[0].get_weights()

np.shape(model.layers[0].get_weights()[0])

model.layers[0].trainable


# The Training process of our optimal HDNN Structure with optimal parameters

filename = 'model_train_new.csv'
filepath = "Best-weights-my_model-{epoch:03d}-{loss:.4f}-{acc:.4f}.hdf5"
csv_log = callbacks.CSVLogger(filename, separator=',', append=False)
checkpoint = callbacks.ModelCheckpoint(
    filepath,
    monitor='val_loss',
    verbose=1,
    save_best_only=True,
    mode='min')
callbacks_list = [csv_log, checkpoint]
```

```
callbacks_list = [csv_log]


hist = model.fit(
    X_train,
    y_train,
    batch_size=7,
    nb_epoch=30,
    verbose=1,
    validation_data=(
        X_test,
        y_test),
    callbacks=callbacks_list)

model.save_weights('model_weights.h5')
model.save('model_keras.h5')

# We can visualize our optimal HDNN Structure with optimal parameters loss
# and accuracy through graphs


train_loss = hist.history['loss']
val_loss = hist.history['val_loss']
train_acc = hist.history['accuracy']
val_acc = hist.history['val_accuracy']

epochs = range(len(train_acc))
plt.plot(epochs, train_loss, 'r', label='train_loss')


plt.plot(epochs, val_loss, 'b', label='val_loss')


plt.title('train_loss vs val_loss')


plt.legend()

plt.figure()


plt.plot(epochs, train_acc, 'r', label='train_acc')


plt.plot(epochs, val_acc, 'b', label='val_acc')


plt.title('train_acc vs val_acc')


plt.legend()


plt.figure()


# The Evaluation of our optimal HDNN Structure with optimal parameters


score = model.evaluate(X_test, y_test, verbose=0)

print('Test Loss:', score[0])


print('Test accuracy:', score[1])


test_image = X_test[0:1]
print(test_image.shape)
```

```
print(model.predict(test_image))


print(model.predict_classes(test_image))


print(y_test[0:1])


res = model.predict_classes(X_test[:9])
plt.figure(figsize=(10, 10))

for i in range(0, 9):
    plt.subplot(330 + 1 + i)
    plt.imshow(X_test[i], cmap=plt.get_cmap('gray'))
    plt.gca().get_xaxis().set_ticks([])
    plt.gca().get_yaxis().set_ticks([])
    plt.ylabel('prediction = %s' % getLabel(res[i]), fontsize=14)
    plt.show()

# We can visualize our optimal HDNN Structure with optimal parameters
# confusion matrix

results = model.predict_classes(X_test)
cm = confusion_matrix(np.where(y_test == 1)[1], results)
plt.matshow(cm)


plt.title('Confusion Matrix')


plt.colorbar()

plt.ylabel('True Label')


plt.xlabel('Predicted Label')


plt.show()

plt.gca().get_xaxis().set_ticks([])

plt.gca().get_yaxis().set_ticks([])

plt.xlabel('prediction = %s' % getLabel(results[0]), fontsize=25)

Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1900 64 bit (AMD64)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/DODI/dmotembe1.py
==========
Using Theano backend.
Loaded the images of dataset-ANGRY

Loaded the images of dataset-DISGUST

Loaded the images of dataset-FEAR

Loaded the images of dataset-HAPPY

Loaded the images of dataset-NEUTRAL

Loaded the images of dataset-SAD

Loaded the images of dataset-SURPRISE


Warning (from warnings module):
  File "C:/Users/DODI/dmotembe1.py", line 106
```

```
    model.add(Convolution2D(6, 3, 3, input_shape=input_shape, border_mode='same'))
UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(6, (3, 3), input_shape=(128, 128,..., padding="same")`

Warning (from warnings module):
  File "C:/Users/DODI/dmotembe1.py", line 110
    model.add(Convolution2D(16, 3, 3, border_mode='same'))
UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(16, (3, 3), padding="same")`
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 6)       168
_________________________________________________________________
activation_1 (Activation)    (None, 128, 128, 6)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 16)        880
_________________________________________________________________
activation_2 (Activation)    (None, 64, 64, 16)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 16)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_1 (Dense)              (None, 84)                1376340
_________________________________________________________________
activation_3 (Activation)    (None, 84)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 84)                0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 595
_________________________________________________________________
activation_4 (Activation)    (None, 7)                 0
=================================================================
Total params: 1,377,983
Trainable params: 1,377,983
Non-trainable params: 0
_________________________________________________________________

Warning (from warnings module):
  File "C:/Users/DODI/dmotembe1.py", line 173
    callbacks=callbacks_list)
UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
Train on 180 samples, validate on 32 samples

Epoch 30/30

  7/180 [>.............................] - ETA: 10s - loss: 0.9542 - accuracy: 0.7143
 14/180 [=>............................] - ETA: 10s - loss: 0.7767 - accuracy: 0.7143
 21/180 [==>...........................] - ETA: 10s - loss: 0.7055 - accuracy: 0.7143
 28/180 [===>..........................] - ETA: 10s - loss: 0.6125 - accuracy: 0.7143
 35/180 [====>.........................] - ETA: 9s - loss: 0.6157 - accuracy: 0.7143
 42/180 [======>.......................] - ETA: 9s - loss: 0.6178 - accuracy: 0.7143
 49/180 [=======>......................] - ETA: 8s - loss: 0.5749 - accuracy: 0.7551
 56/180 [========>.....................] - ETA: 8s - loss: 0.5672 - accuracy: 0.7679
 63/180 [=========>....................] - ETA: 7s - loss: 0.5362 - accuracy: 0.7778
 70/180 [==========>...................] - ETA: 7s - loss: 0.5226 - accuracy: 0.7714
 77/180 [===========>..................] - ETA: 6s - loss: 0.5792 - accuracy: 0.7403
 84/180 [============>.................] - ETA: 6s - loss: 0.5395 - accuracy: 0.7619
 91/180 [==============>...............] - ETA: 6s - loss: 0.5127 - accuracy: 0.7802
 98/180 [===============>..............] - ETA: 5s - loss: 0.4956 - accuracy: 0.7857
105/180 [================>.............] - ETA: 5s - loss: 0.4908 - accuracy: 0.8000
112/180 [=================>............] - ETA: 4s - loss: 0.4845 - accuracy: 0.7946
119/180 [==================>...........] - ETA: 4s - loss: 0.5080 - accuracy: 0.7815
126/180 [====================>.........] - ETA: 3s - loss: 0.4942 - accuracy: 0.7857
133/180 [=====================>........] - ETA: 3s - loss: 0.5041 - accuracy: 0.7820
140/180 [======================>.......] - ETA: 2s - loss: 0.4962 - accuracy: 0.7857
147/180 [=======================>......] - ETA: 2s - loss: 0.4771 - accuracy: 0.7959
154/180 [========================>.....] - ETA: 1s - loss: 0.4785 - accuracy: 0.7922
```

```
161/180 [============================>....] - ETA: 1s - loss: 0.4768 - accuracy: 0.7888
168/180 [=============================>..] - ETA: 0s - loss: 0.4776 - accuracy: 0.7917
175/180 [==============================>.] - ETA: 0s - loss: 0.4761 - accuracy: 0.7886
180/180 [==============================] - 13s 74ms/step - loss: 0.4647 - accuracy:
0.7944 - val_loss: 0.7941 - val_accuracy: 0.6875
Test Loss: 0.7940693497657776
Test accuracy: 0.6875
(1, 128, 128, 3)
[[ 1.25928875e-02  6.56534161e-04  2.93277553e-03  1.16468444e-02
   9.33720112e-01  3.42316218e-02  4.21920326e-03]]
[4]
[[ 0.  0.  0.  0.  1.  0.  0.]]
```