Understanding Facial Expression of Children with Autism using Facial Recognition and Deep Learning

By

Tonmoy Bagchi
15301043
Evea Zerin Khan
16101129
Ishrak Sami
16301107
Anuska Zaman Titly
16301175

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
October 2020

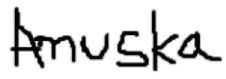
Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Evea Zerin Khan
16101129



Anuska Zaman Titly
16301175

Tonmoy Bagchi
15301043

Ishrak Sami
16301107

Approval

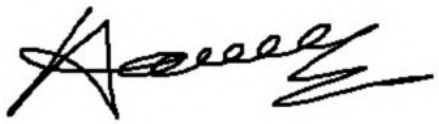
The thesis titled "Understanding Emotional Health of Children with Autism Using Facial Recognition and Deep Learning" submitted by

1. Evea Zerin Khan (16101129)
2. Tonmoy Bagchi (15301043)
3. Ishrak Sami (16301107)
4. Anuska Zaman Titly (16301175)

Of Summer, 2020 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on October 5, 2020.

Examining Committee:

Supervisor:
(Member)



Amitabha Chakrabarty, PhD
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Mahbubul Alam Majumdar, PhD
Professor and Dean, School of Data and Sciences
Department of Computer Science and Engineering
Brac University

Abstract

A prominent research subject in recent times has been the use of Facial Expression Recognition (FER) through Machine Learning. Being a common topic, the challenging part is emotion detection from these facial expressions which can itself become a new sector in this field, but it is yet to be implemented in the dataset concerning Bangladesh. In our research, we have taken pictures of children and teenagers aged between 5 to 20 with Autism Spectrum Disorder (ASD) and inaugurated their emotions from their images. It is challenging for autistic individuals to gather socially and empathically, as we understand, so this subject matter differs from the identification of a common human facial expression. Our goal is to detect a way such that their emotions can be perceived authentically and thus, make it easy for them including other individuals around them to interact socially without any barriers. In our paper, we implemented 3 models and 7 Machine Learning Algorithms along with 1 Feature Extraction technique to figure out the best accuracies for each algorithm. The proposed system has showcased 89% accuracy for Convolutional Neural Network (CNN), 90.97% for VGG16, 89.17% for Inception v3 models. Our system has also shown an accuracy of 72% for K-Nearest Neighbors (KNN) and 55% for KNN along with Principal Component Analysis (PCA), 61% for Support Vector Machine (SVM) and 67% for SVM with PCA, 56% for Logistic Regression and 56.36% for Logistic Regression along with PCA, 52% for Linear Discriminant Analysis (LDA) and 54% for LDA with PCA, 46% for Decision Tree and 56.36% for Decision Tree along with PCA and 46% for Naïve Bayes and 38% for Naïve Bayes along with PCA.

Keywords: Facial Expression Recognition (FER), Autism Spectrum Disorder (ASD), Machine Learning, Convolutional Neural Network (CNN), Inception v3, Support Vector Machine (SVM).

Acknowledgement

We are grateful to our thesis supervisor, Dr.Amitabha Chakrabarty, for his consistent guidance and to make us envision that this research is possible. It is through him that we realized the significance of this topic, and to help people communicate more effectively with children of special needs.

We are also grateful to our co-advisor Dr.Golam Rabiul Alam Sir for his kind support and advice in our work, and for always helping us improve ourselves. He helped us whenever we needed any sort of advice or guidance.

For our parents, without their support throughout, it would not be have been possible to reach this far. With their kind support and prayer, we are now on the verge of completing the final phase of our thesis.

And most importantly, we are humbly grateful to the administrators and teachers of the school Society for the Welfare of Autistic Children (SWAC), situated in Dhaka, Bangladesh, who have trusted us and permitted us to interact with their children, without whom we could not have carried out our research.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the field of research, facial expression recognition and image processing on humans is a very common topic, but very little research has been done on the facial expression recognition of autistic children. Autistic children have varieties of emotions just like any other human being. For example, sometimes they are sad, happy, angry, and so on. But the difference is, it is sometimes very difficult to explain how they are feeling at a particular time by just seeing their face [7], and hence we decided to visit a school of such children of special needs, and we decided to talk to their instructors and understand why they behave in a certain way and how they try to understand their students to get them what is best for them. This itself was so inspiring and thoughtful of the teachers of the school, who help the students selflessly and make them capable of small tasks that they have to perform. According to an instructor, the main problems they face include behavioral, social, and communication. To overcome this barrier and to help other people understand them, we decided to take images and videos of the children, who were just being themselves doing various tasks. We decided to take the images and label them, to further analyze them using different machine learning and deep learning algorithms and classifiers, and we also decided to use feature extraction techniques like Principal Component Analysis (PCA), to further help us determine their expressions and hence emotions. From the perspective of Bangladesh, very few people worked on this similar project. On that note, the autistic children would be highly benefited if our project comes up with near to perfect prediction. This would help in normalizing the autistic children in our society.

1.2 Objectives

Facial Recognition refers to the process we use to identify certain emotions by evaluating the human face via machine learning techniques. Emotion recognition is now so much easier; thanks to the advancement of machine learning and artificial intelligence all together. Emotion recognition always depends on various factors, such as, environment, appearance, culture, etc. As our research is to work on the behavioral expressions of autistic children, it covers a vast area of complexity from the very first step of this search being started. We are mainly doing this so that the approach of our society to- wards autism gets some positive light. We are going to extract a

dataset of facial expressions of autistic children so that we can train our machine to predict their expressions (whether they are happy, sad, angry, or in a neutral state). As our topic is sensitive and confidential and there are also some ethical obligations, there is no secondary data available from any sources. That is why, we will use our own dataset that we have collected from a school of the children of such special needs, which will be a great resource for the people who will be doing work on this topic furthermore. However, we needed a huge number of images of autistic children in order to research properly. Moreover, we try to find out if the different models of CNN and other machine learning algorithms can perform as good as they perform for emotion recognition for non-ASD people. Moreover, we show which algorithms perform better for recognizing emotions and which emotions are easily recognized for autistic children. However, there are many therapies, which help to improve their expressions and behavior, but we will work on their initial expression. As they cannot express their needs properly, we are trying to find a way so that people around them can understand their needs and feelings accurately and show more empathy towards them.

1.3 Methodology

Our first step was to collect several images of different autistic children from a school in Dhaka, Bangladesh. This required fieldwork. We went to collect the images and videos of the children, and with the help of their instructor we labeled them, and then tried to extract certain facial expression parameters using feature extraction and algorithms of machine learning. We used CNN along with various other algorithms namely SVM Classifier, Decision Tree Classifier, K-Nearest Neighbors Classifier, Support Vector Machine, Naive Bayes Classifier, Logistic Regression, Random Forest Classifier, and Linear Discriminant Analysis. We then compare the results and accuracy rate of previous works with our work and try to achieve a better accuracy rate.

1.4 Thesis Overview

The remaining part of this thesis report has been organized as follows:

- Chapter 2 analyzes a thorough study of the related works of our thesis.
- Chapter 3 describes the dataset we collected from the autistic children's facial expressions, which was used in our thesis.
- Chapter 4 discusses the various algorithms and classifiers applied in our research, and the accuracies obtained.
- Chapter 5 compares the algorithms and analyzes the results obtained and comments on the performance.
- Chapter 6 concludes our thesis and proposes our future work.

Chapter 2

Literature Review

2.1 Related Works

In recent years, there has been tremendous progress in emotion recognition from facial expression using machine learning and deep learning. We have studied many of the related works on various research papers on the topic emotion recognition, machine learning, image processing, and feature extraction and so on. In this part of our report, we have discussed some of the previous research works that we have studied. Reading through the paper written by Yuehan Zhu, we got to know that the author used his own dataset that he obtained from a factory [38]. The author used different methodologies to look into the various outcomes. Binarization Algorithm, General Edge Detection, Contour Detection, Neural Networks and Machine Learning were used by him. Out of all the methods, the LSTM method performed better with just 25.18% error rate with Basic character and only 0.3% error rate with Fine-tuned characters. Limitations were - general methods from literature do not perform well. The canny detection fails to detect all labels and optimum thresholds perform relatively poor.

Reading the paper authored by Nitisha Raut, we get to know that she got her datasets from extending the Cohn-Kanade database (CK+) and the Radbound Faces database (RaFD) [26]. CK+ has about 593 images for the 123 subjects, among which only 327 files have labeled or identified emotions. Support Vector Machine (SVM) was used to predict the emotions of subjects. CNN was having near about 85.25% accuracy, k-nn and CRF was giving near 82.2% accuracy, whereas SVM was acquiring 89.78% accuracy. Limitation was – If there were multiple faces in the picture, the machine was not able to detect or identify all the emotions of the faces accurately.

On another paper written by JIAQI HU, the author has used a dataset of four volunteers chosen by the writer herself [22]. They used CNN Model Architecture. In the results - Facial expression recognizer reaches 100% accuracy on disgust expression. Happy and sad expression had over 80% accuracy with or without error images. Surprise had 78.57% accuracy without error, but only 61.1% accuracy with the error image. Neutral is the most difficult expression for the facial expression recognizer, only 45.8% with the error and 47.8% without error image. The facial expression recognizer had 73.8% accuracy on fixed user, but the facial expression recognizer had a hard time handling an expression while it is in transition. The

accuracy for neutral is only 45.8%. This is the weak link of the facial expression recognizer.

As we know that there has been active research on this field, most of the works were based on using Convolutional Neural Networks (CNNs). These works vary significantly in terms of the different CNN architectures and other factors. This paper reviews six methods for CNN based Facial Expression Recognition (FER), highlights the differences in the methodologies, and discusses the resulting performances [18]. In the original papers, most of these methods were evaluated on several databases. The most common dataset used was FER2013, which is a large and publicly available dataset of facial expression recognition, consisting of 35,887 face crops. The pictures in the dataset vary significantly in terms of the person's age, facial orientation, and other factors, reflecting realistic conditions. Basic expression labels are given for all the samples. All images are on a grayscale and have resolution of 48 by 48 pixels. The human accuracy after using this dataset is about 65.5%. They obtained a FER2013 test accuracy of 75.2%, which outperformed the previous works without needing any auxiliary training data or facial registration.

The authors of this paper have used several Machine Learning algorithms along with CNN using the datasets of the different expressions of children with autism spectrum disorder and compared the accuracies to find out which of the algorithms can successfully identify their expressions most precisely [28]. They have also used two feature extraction methods namely LBP and PCA, in order to improve the accuracy of the algorithms and to extract better features from the images. After they have used SVM with PCA feature extraction,they got an accuracy of 68.2%. They collected the data on their own, and retrieved approximately 2000 images as their final dataset. They used supervised machine learning techniques so they had to train and test data. They tried to identify four categories of emotion- happy, angry, sad and neutral. 80% of their images were used for training and 20% for testing. After comparing all algorithms with PCA and LBP, alone SVM Classifier gave the lowest accuracy of 25.4% whereas the highest accuracy they got was after implementing SVC with PCA, that is 68.2

Studying the paper by the authors Shervin Minaee and Amirali Abdolrashidi, we got to know that they provided the experimental analysis on some of the most popular facial expression recognition datasets such as FER2013, CK+, Japanese Female Facial Expression or JAFFE, and Facial Expression Research Group Database or FERG. [35]. The different classification accuracies on the FER 2013 dataset came down for each of them. Starting with Bag of Words with 67.4% accuracy, then VGG+SVM: 66.31% accuracy, then GoogleNet with 65.2% and Mollahosseini et al with 66.4% accuracy. Thus, the proposed algorithm with 70.02% accuracy. Again, for the classification Accuracy on FERG dataset the method accuracy rate came down as for DeepExpr with 89.02%, Ensemble Multi-feature with 97% Adversarial NN with 98.2%, thus the proposed algorithm of 99.3% accuracy. Furthermore, the classification accuracy on JAFFE dataset came down as for Fisherface 89.2%, Salient Facial Patch 91.8%, CNN with SVM 95.31%, and hence the proposed algorithm 92.8%. Lastly, the classification accuracy on CK+ was for MSR 91.4%, 3DCNN-DAP 92.4%, Inception 93.2%, IB-CNN 95.1%, IACNN 95.37%, DTAGN

97.2%, ST-RNN 97.2%, PPDN 97.3% and the proposed algorithm 98.0%.

The author D Y Liliana in her paper has discussed about deep convolutional neural network in case of detecting human emotion from facial expression [33]. She used the occurrence of facial Action Units (AUs) as a subpart of Facial Action Coding System (FACS). The method she used is called "dropout" in CNN fully-connected layers which reduced overfitting. The author used CK+ dataset which comprises 10.708 pictures from 123 unique subjects. This includes eight categories of emotions: anger, neutral, contempt, disgust, happiness, sadness, fear, and surprise. Images were reshaped into 100 by 100 pixels before using the CNN model. Moreover, the writer has used a CNN algorithm which consists of two convolutional layers and two subsampling layers. First convolutional layer included 6 masks, the next subsampling layer had 2 layers, the second convolutional layer had 12 masks and the last subsampling layer had 2 layers. For class classification the last layer is a fully connected layer. Average accuracy rate of her research was 92.81%. The least accuracy rate was for the anger class, 87.73% and the highest one was for the surprise class, 98.09%. The limitation of the author's research was the misclassification of results for each class. Besides that, the author states that CNN architecture could be more developed to get better performance.
On another paper, the authors have tried to improve the accuracies of facial expression recognition system [25]. The authors have used datasets from online such as JAFFE and the CK+ datasets as we ave seen previously. They stated that, the major steps for the work were image data preprocessing, feature extraction of the images, construction of a codebook and the data classification. They used techniques of SIFT and SURF descriptors for the extraction of features that are scale, illumination, and rotation invariant from the segmented facial patches. For building the dictionary, the authors implemented an ordered spatial bag of feature (SBoF) technique. They have optimised parameters such as facial region of interest (ROI) and the feature vector length that gave better results for getting accuracies. For CK+ dataset, the SBoF and SSIFT feature performance was not very well when they used it with RBF and polynomial kernels. Kernel accuracy was 69.5. Moreover, RBF and polynomial kernel accuracy was 73.7%. Also, CK+ dataset accuracy was 95.8% and JAFFE dataset accuracy was 98.3%. In their opinion, it could be extended to develop a recommendation system.

Another research paper was about recognition of facial expressions of autistic children using the method of ASM tracker where there were 116 facial images taken using webcam input. The tracked landmark points were used to extract facial expression features using a SVM classifier which was used to recognize seven different expressions [15]. The accuracy of the system was 93%. On the other hand, from their opinion, fear and surprise expressions are misclassified. Sadness and disgust expressions were misclassified.

From another research paper, the authors have proposed a model for children with ASD, that uses computer vision and image processing to help these children identify human facial expressions. The facial expressions are analysed by a deep CNN model, and have used the most popular Kaggle's FER2013 dataset to implement this model [24]. They have used six expressions of normal humans namely fear, anger,

happiness, sadness disgust, and surprise. At first they used an optimizer known as standard SGD along with a ReLU activation function, with an initial batch size of 128. They have trained it with different learning rates of 0.01, 0.005, 0.001, 0.0005 and 0.0001 respectively. The accuracy they obtained was 60.23%, where the dropout layer had a value of 0.5. Again, the value of the dropout layer was tested with 0.25 after every pooling layer was traversed. This had made an increase in validation accuracy of 65.24%, and was obtained for a learning rate of 0.01. The Adam optimizer was used and it resulted in an accuracy of 63.11%. The test accuracy obtained was of 63.11%, which was achieved without overfitting the model, resulting in satisfactory outcomes. When the same model was trained, they found that the datasets which had very bright images yielded a better test accuracy than the datasets with less bright images. This observation can have a positive impact for anyone doing image processing, since it would be helpful from an initial stage if it is followed.

Chapter 3

Dataset Analysis

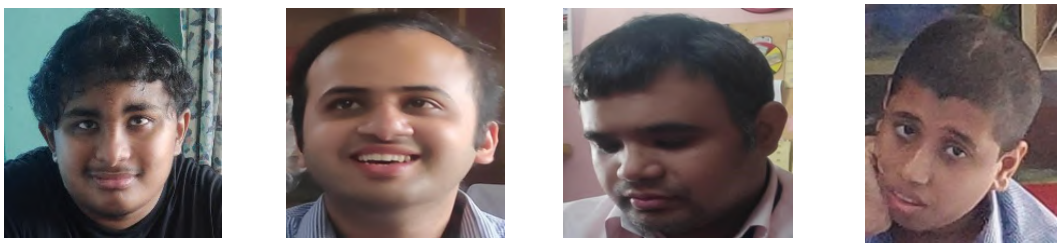
In this chapter, we have provided details of the data collection process, data labelling and refining stage, dataset validation and the test data.

3.1 Data Collection

In the field of research, facial expression recognition and image processing on humans is a very common topic, but very few research has been done on autistic children. Autistic children have different varieties of emotions just like any other human being. For example, sometimes they are sad, happy, angry, and so on. But the difference is, it is sometimes very difficult to explain how they are feeling at a particular time, and hence we decided to visit a school of such children of special needs known as Social Welfare for Autistic Children (SWAC), and we decided to talk to their instructors and understand why they behave in a certain way and how they try to understand their students to get them what is best for them. On the very first day, we faced difficulties to have permission to take pictures, and only on the second day did we get permission from them to do our thesis work. This itself was so inspiring and thoughtful of the teachers of the school, who help the students selflessly and make them capable of small tasks that they have to perform in their daily lives. According to an instructor, the main problems they face include behavioral, social, and communication. To overcome this barrier and to help other people understand them, we decided to take images and videos of the children who were doing their regular tasks and we ensured they were not distracted, and as we took pictures, the instructors guided us to understand their emotion at that instant, and we noted it down sequentially. They were performing several tasks such as making envelopes, drawing pictures, attending music class, and attending their gym sessions. Some students were restless and some students liked to stay idle. After an entire day of data collection, we decided to divide the work and start data processing and labelling. Figure 3.1 shows the four categories of images we collected of the children with ASD condition.

3.2 Data Labelling and Refining

We noted the time and emotion of any child at any time, and matched it with the picture or video frame we had. We cropped the pictures to the appropriate



(a) Angry. (b) Happy. (c) Neutral. (d) Sad.

Figure 3.1: Image data of children with ASD.

dimensions as required, but cropping somewhat degraded picture quality in certain cases. All four of our group members divided the tasks among ourselves, and we collected thousands of facial picture frames of the children. We later coded the color to be converted to the gray scale using Python, and also resized the images to 150 by 150 pixels. We discarded some pictures that were not suitable for our dataset.

We categorized the data into four labels: happy, angry, sad and neutral. We kept separate folders for the same expressions and used it accordingly for every algorithm coding. We shuffled the data before feeding it into each Machine Learning Model.

3.2.1 Training Dataset

The dataset that we labeled was used to train the model. We used around 1800 data images for training the model which was 75% of the whole dataset. We fed the data to our model and our model learned from it.

3.2.2 Validation Dataset

We followed an unbiased model fit on our training dataset. We used 70% of the 75% training dataset in the creation of our model. On the other hand, 30% of the training dataset was used to fit the model. Thus, our model was perfectly fitted.

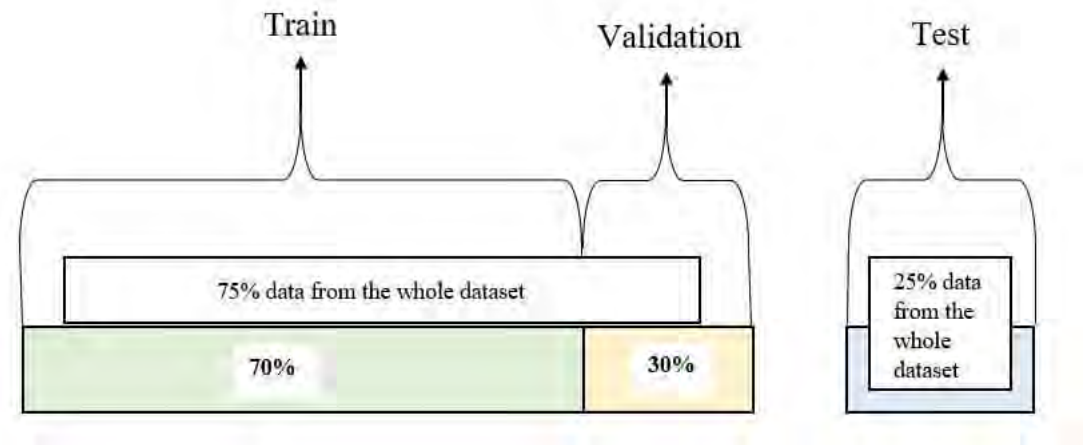


Figure 3.2: Data split visualization.

By splitting the training dataset into 70% and 30%, we tried to create an unbiased model. We used a completely different dataset for testing after creating the model.

None of the training data was in the testing dataset. In addition, no testing data was used in the creation of the model [28].

3.3 Test Dataset

Testing: The test dataset provided us the standard in the evaluation of the model. After completion of the training of our model, we provided the test dataset. Different types of algorithms gave us different type of accuracy. A dataset of 600 images was used for testing. The testing dataset was 25% of the whole dataset. The dataset that we used for testing was not in the training dataset[28].

Chapter 4

Model Specification

In this section, we will look at the different machine learning algorithms as well as deep learning algorithms that we have used to determine the results.

4.1 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is defined as a Deep Learning algorithm. This algorithm can take an image as input. Assigning importance (learnable weights and biases) to various aspects in the image, it differentiates one from the other. CNN requires much lower pre-processing as compared to other classification algorithms. This neural network consists of some convolutional layers, pooling layers and fully connected layers. A convolutional layer includes number of filters. This layer does convolutional operation. The convolution of f and g, written as f * g or f ° g, is defined as the integral of the product of the two functions after one is reversed and shifted.

$$(f * g)(t) = \int_0^t f(\tau)g(t-\tau)d\tau \tag{4.1}$$

$$c = f * g \tag{4.2}$$

Convolutional neural network (ConvNet or CNN) is one of the most efficient model in case of image recognition, image classification in neural networks. Detection of objects, faces recognition etc. are some of the areas where CNNs are broadly implemented. At first, this image classifier model takes an image as input. After processing, the model classifies the image under specific categories (Angry, Happy, Normal and Sad). An input image is converted to an array of pixels and is seen by computers. This method has a dependency on the resolution of an image. The algorithm will see h x w x d (h = Height, w = Width, d = Dimension) based on the resolution of an image. To illustrate, CNN will see an image of 6 x 6 x 3 array of matrix of RGB where 3 means RGB values and an image of 4 x 4 x 1 array of matrix of grayscale image[36].

Each input image will pass through a convolution layers series with filters, pooling layers and fully connected layers. After that, Softmax function is applied to classify an object with values between 0 and 1 to train and test Convolutional Neural Network models[36].

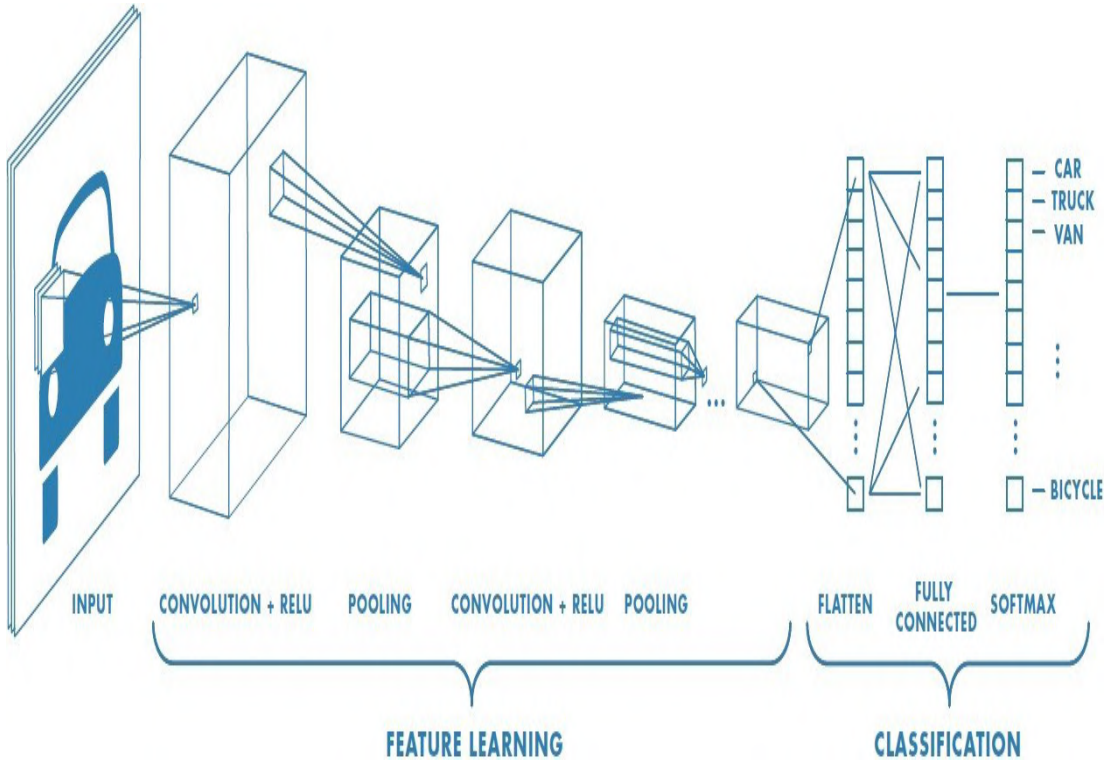


Figure 4.1: CNN Architecture [36].

4.1.1 Layers of CNN

There are three layers of CNN:

1. This is the first layer for feature extraction from an input which is an image. Convolution learns image features from small squares of input and maintains the relationship between pixels. This process is an arithmetical operation that takes two inputs. One is an image matrix and other is a filter. Edge detection, blurring and sharpening can be performed by applying filter operations by Convolution with different filters.

    - **Strides**: Stride is a parameter of Conv2D which is an integer value. We used the default value of stride (1, 1). This means that our Conv2D filter was applied to the present location of the input volume. The filters moved 1-pixel step to the right and then the filter was applied to the input volume again. This procedure continued till we reached the rightmost border of the volume [42].
    - **Padding**: Padding is used when the filter does not fit properly with the input image. This parameter pads the picture with zeros for fitting in zero-padding. In valid padding, the part of the image is removed where the filter did not fit [36].
    - **Non Linearity (ReLU)**: The full form of ReLU is Rectified Linear Unit. It is applied for the non-linear operation. The output is f(x) = max(0,x). ReLU introduces non-linearity in our ConvNet. ConvNet needs to learn non-negative linear values while working with real word data [36].

2. **Pooling Layers**: The responsibility of the pooling layer is to reduce the spatial size of the Convolved Feature. While processing data, it is used to decrease the computational power through dimensionality reduction. Moreover, it extracts dominant features which are rotational and positional invariant. In this way, the layer effectively trains the model. The maximum value from the portion of the image covered by the Kernel is returned in max pooling. Another type of pooling layer is average pooling is used to get the average of all the values from the portion of the image covered by the Kernel. Max Pooling also works as a Noise Suppressant. Max pooling is not only used for dimensionality reduction, it also conducts de-noising. Max Pooling performs a lot better than Average Pooling. That is why we used MaxPool2D() in our CNN model. [27] Sum pooling is another type of pooling where the sum of all elements remains in the feature map[36].

3. **Dense (Fully connected) Layers**: In FC layer, our matrix are flattened into vector and then we fed it into a fully connected layer like a neural network.

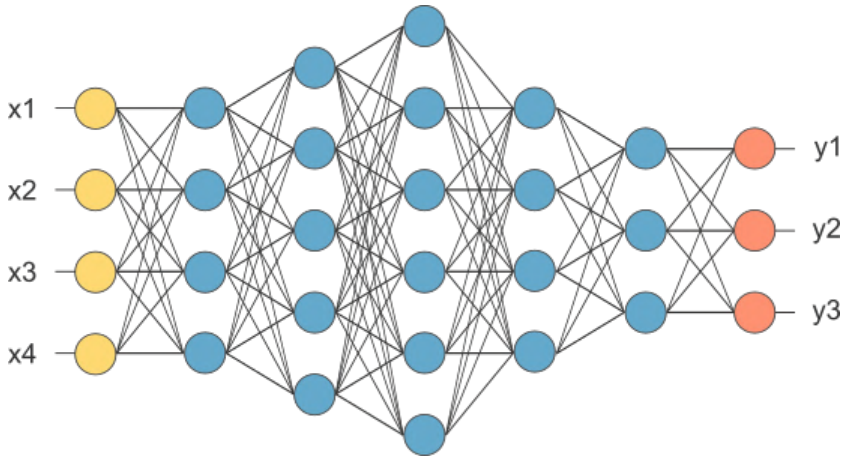


Figure 4.2: Flattened as FC layer[36].

The matrix of feature map conversion as vector (x1, x2, x3 ...) is shown in the above diagram. These features were merged together to build the CNN model in fully connected layers. Finally, an activation function named softmax is used for classification of the outputs as angry, happy, sad and normal [36].

4.1.2 CNN Procedure

We will be explaining the implementation of our CNN model here for predicting the emotions of autistic children. First of all, overall summary of the layers are given below:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 200)	5600
conv2d_1 (Conv2D)	(None, 146, 146, 180)	324180
max_pooling2d (MaxPooling2D)	(None, 29, 29, 180)	0
conv2d_2 (Conv2D)	(None, 27, 27, 180)	291780
conv2d_3 (Conv2D)	(None, 25, 25, 140)	226940
conv2d_4 (Conv2D)	(None, 23, 23, 100)	126100
conv2d_5 (Conv2D)	(None, 21, 21, 50)	45050
max_pooling2d_1 (MaxPooling2	(None, 4, 4, 50)	0
flatten (Flatten)	(None, 800)	0
dense (Dense)	(None, 180)	144180
dense_1 (Dense)	(None, 100)	18100
dense_2 (Dense)	(None, 50)	5050
dropout (Dropout)	(None, 50)	0
dense_3 (Dense)	(None, 4)	204

Total params: 1,187,184
Trainable params: 1,187,184
Non-trainable params: 0

Figure 4.3: Summary of CNN model.

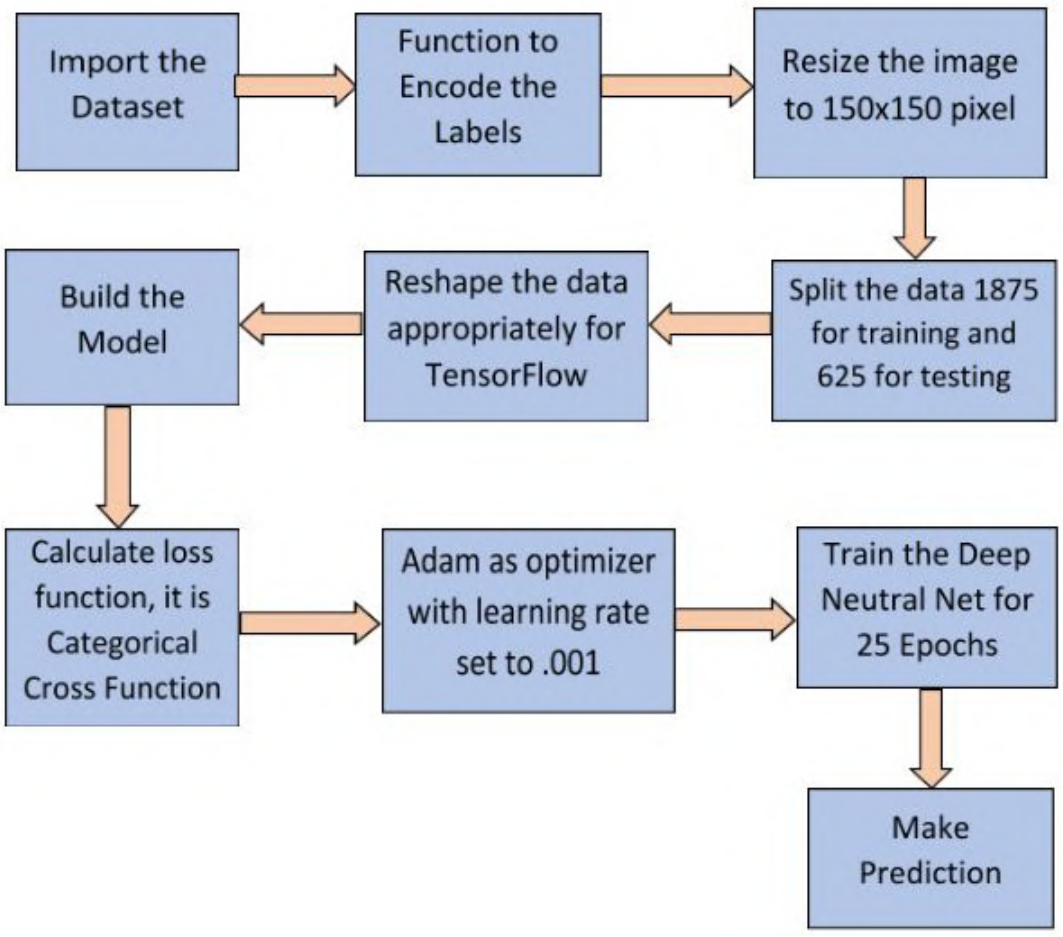


Figure 4.4: Flowchart for CNN implementation.

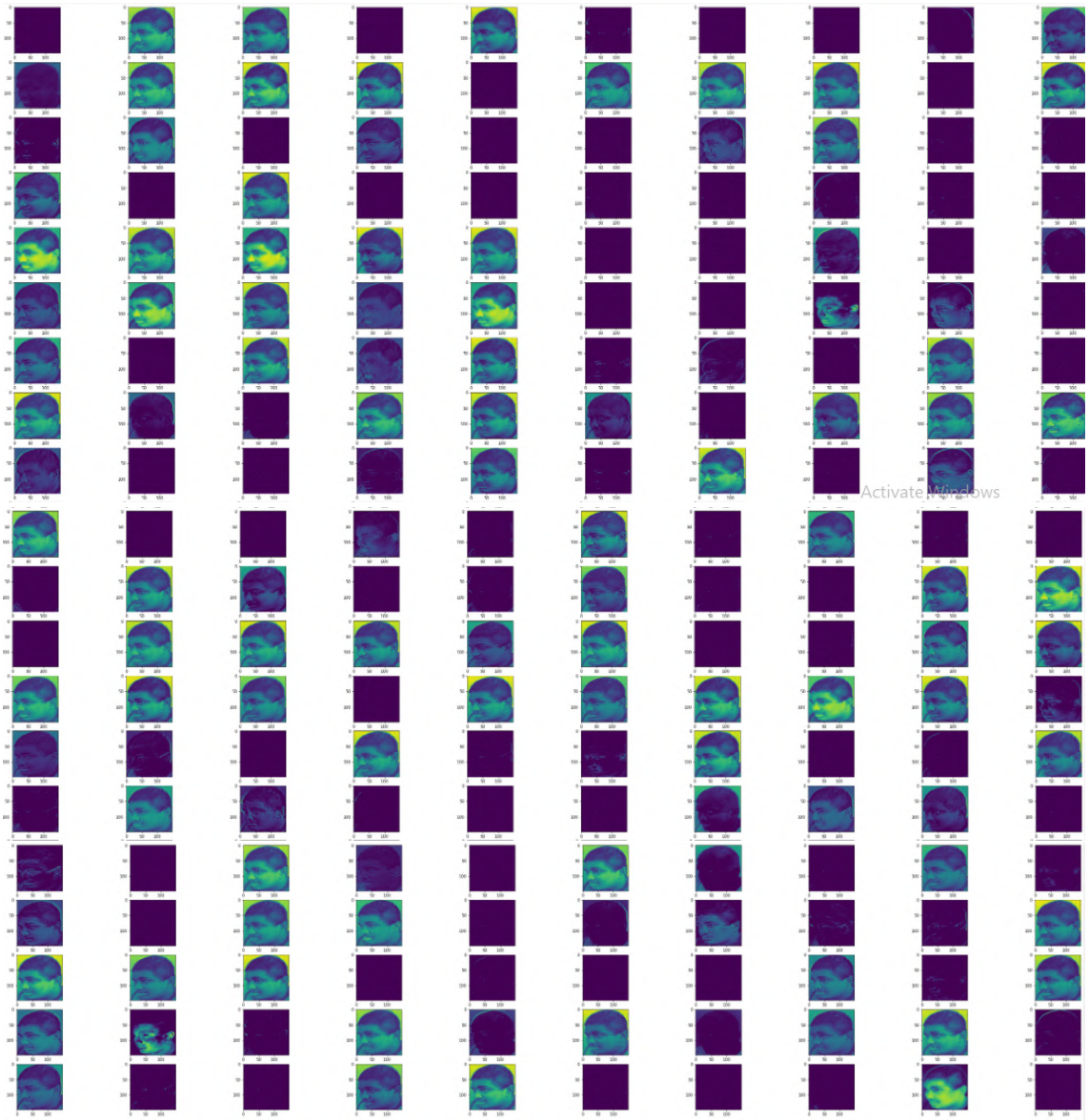


Figure 4.5: Feature map of input layer 1.

Input Layer: While working with CNN, image data remains in the input layer. Image data is a three dimensional matrix representation. We used the following method for two dimensional input image data:

- Batch size: We have used about 2500 images during performing the algorithms.
- Image height: We reshaped the height of the input image as 150.
- Image Weight: The width of our input image was 150.
- Channels: We used polychromatic images. For this reason, we kept the number of color channels in our model as 3.
- Data format: The shape for our input layer was [batch size, 150, 150, 3] [28].

Here, we can see the image data after input layer in figure 4.5.

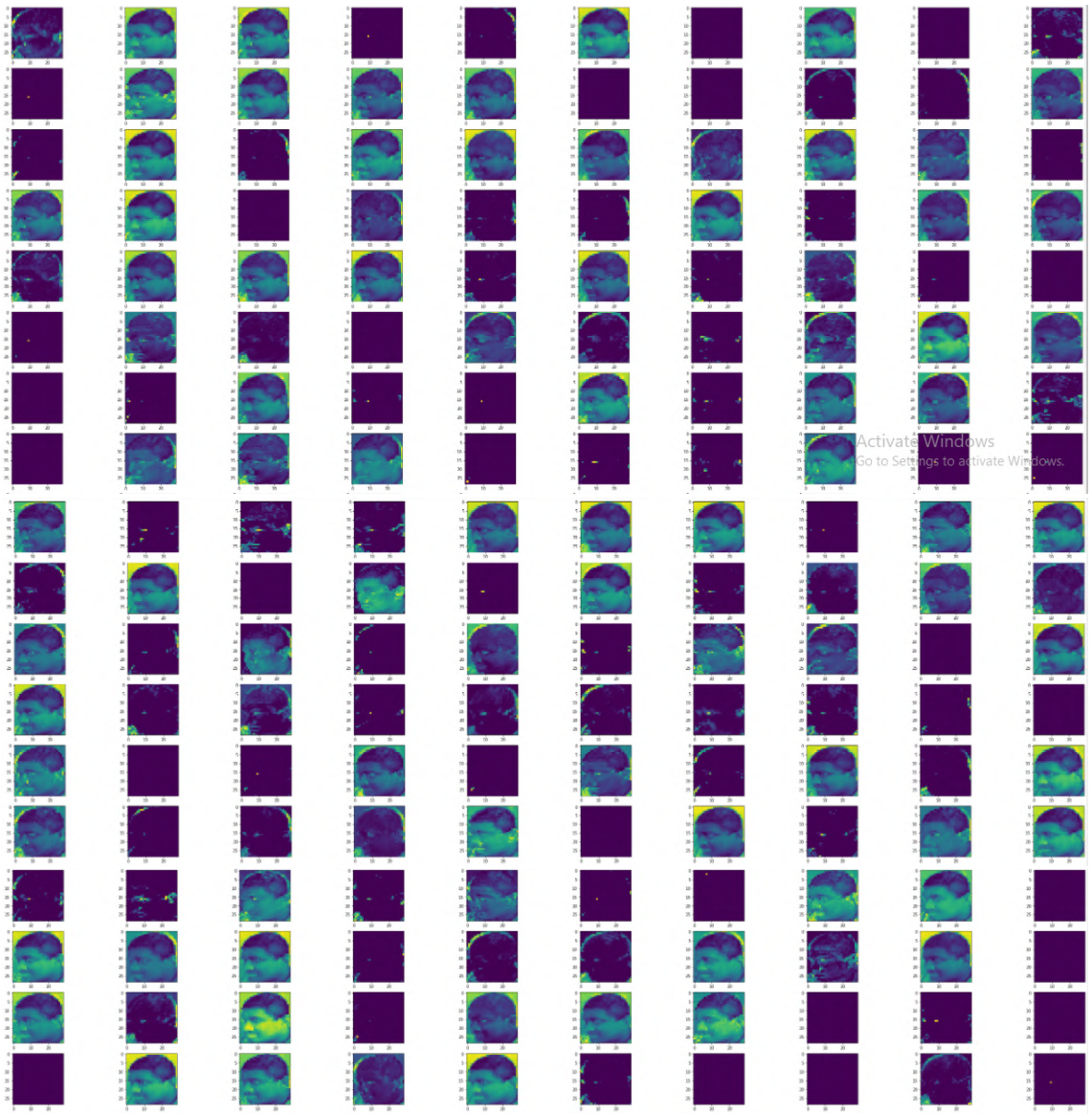


Figure 4.6: Feature map of convolutional layer 1.

Convolutional Layer 1: The straightforward application of a kernel to an input image that results in an activation is called convolution [39].180 3x3 filters are applied to the input layer along with a ReLU activation function. We used conv2d() method to create the layers. To illustrate, (200,kernel_size=(3,3),activation='relu')[39]. 180 3x3 filters are applied to the input layer along with a ReLU activation function. We used conv2d() method to create the layers. To illustrate, (200,kernel_size=(3,3),activation='relu')[28].
We can keep two types of value in the padding argument. If we use valid padding, it means there is no padding. We used padding ="same" which generates padding evenly to the left or right or up or down of the input. Thus we maintained the same height or width dimension for the output as the input[49].

Figure 4.6 shows the convolved feature map of our very first convolutional layer.

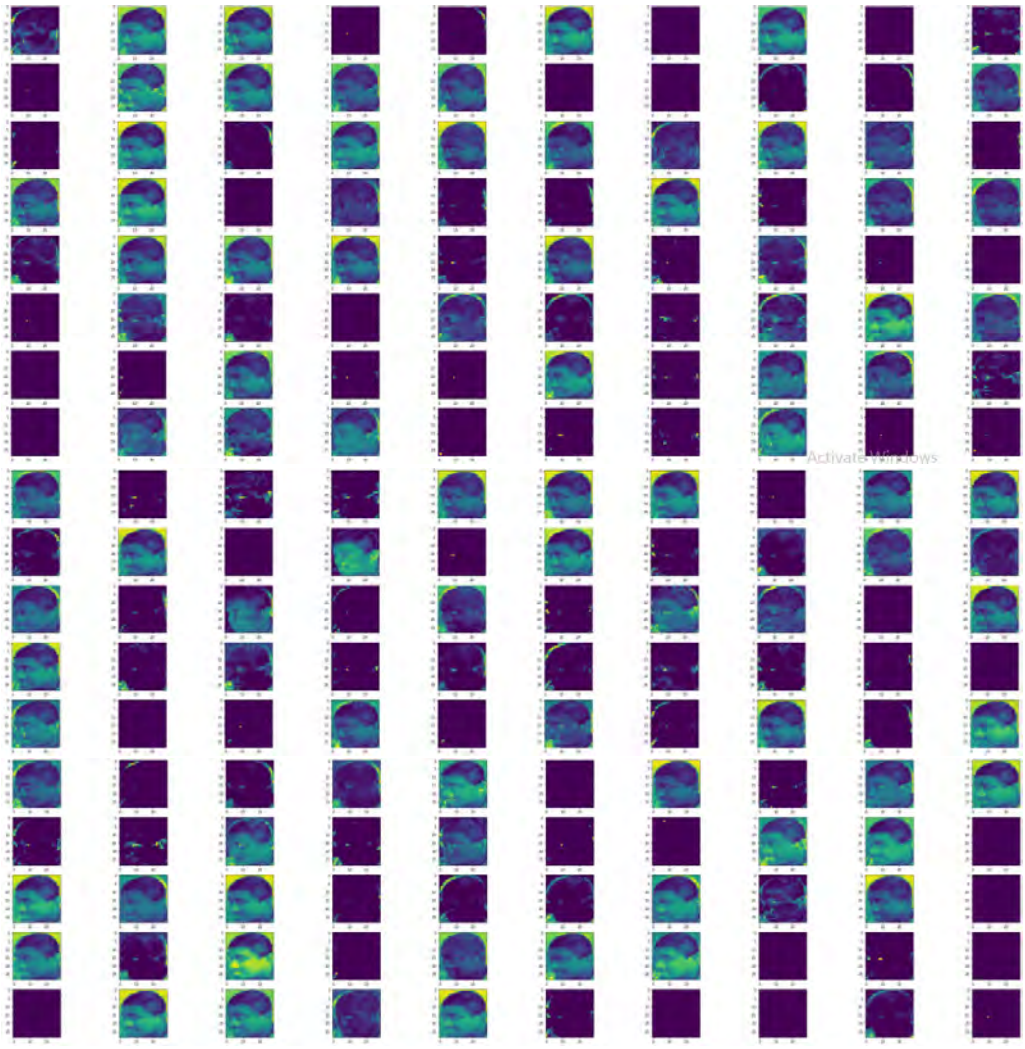


Figure 4.7: Feature map of pooling layer 1.

Pooling Layer 1: For pooling layer 1, we used Max_pooling2d () method. This creates a layer of max pooling with 5x5 filter. Furthermore, we kept the default value of strides which is (1, 1). This default value worked well for our system performance. The method we used with parameters was MaxPool2D(5,5) [28].
In figure 4.7, we can notice the difference of image output after it was pass through pooling layer 1.

Convolutional Layer 2: A second convolutional layer was used in our CNN model with the help of conv2d(). We incorporated 180 3x3 filters with the ReLU activation function. There were 180 channels. We kept the padding ="same" as before.
The feature map of second convolutional layer is given in figure 4.8.

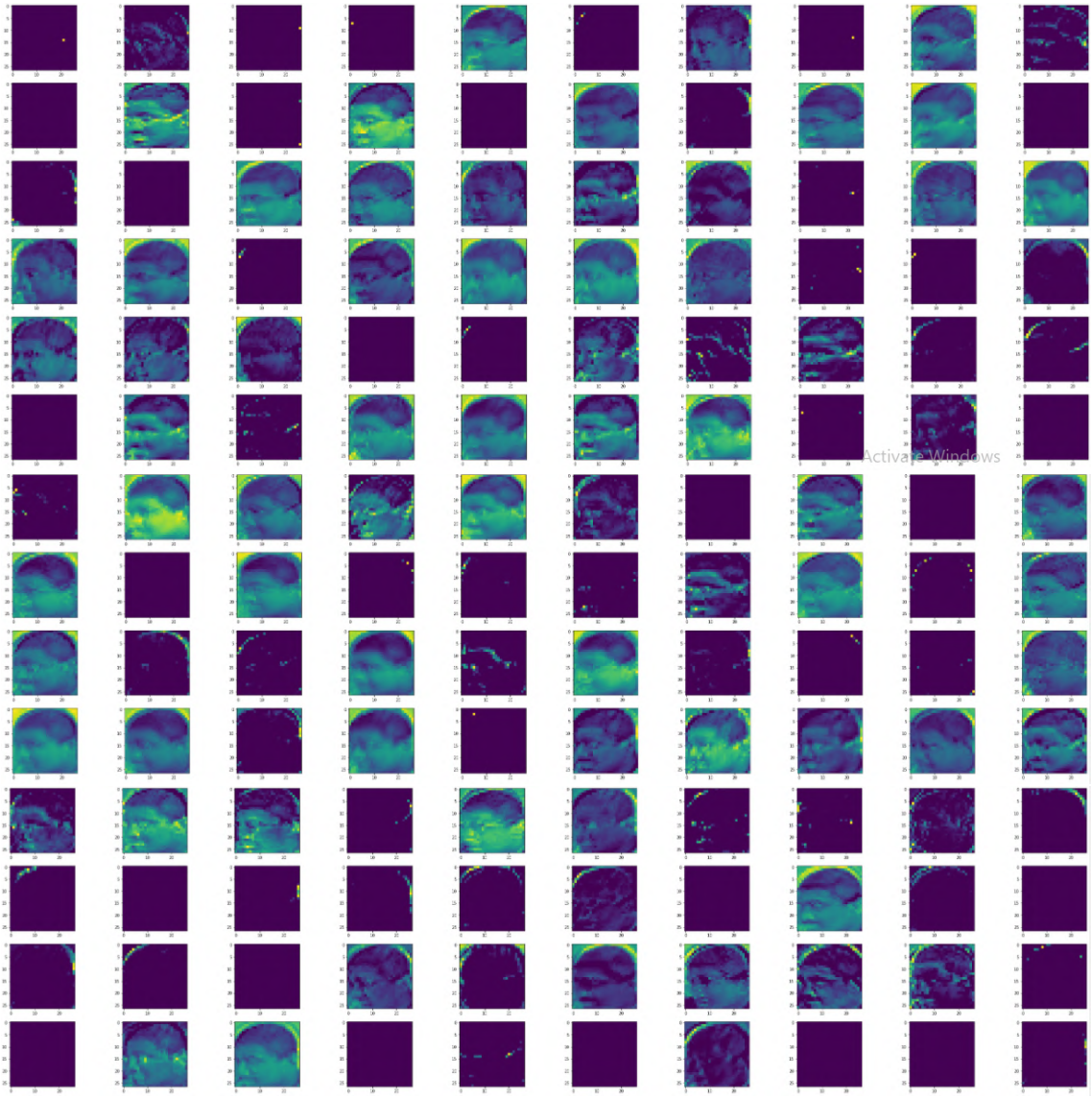


Figure 4.8: Feature map of convolutional layer 2.

Convolutional Layer 3: We used third convolutional layer. This time we used 140 3x3 filters with the ReLU activation function again. Other parameters were the same as before. Increasing the number of layers helps our model to learn better. Figure 4.9 refers to the output after third convolutional layer.

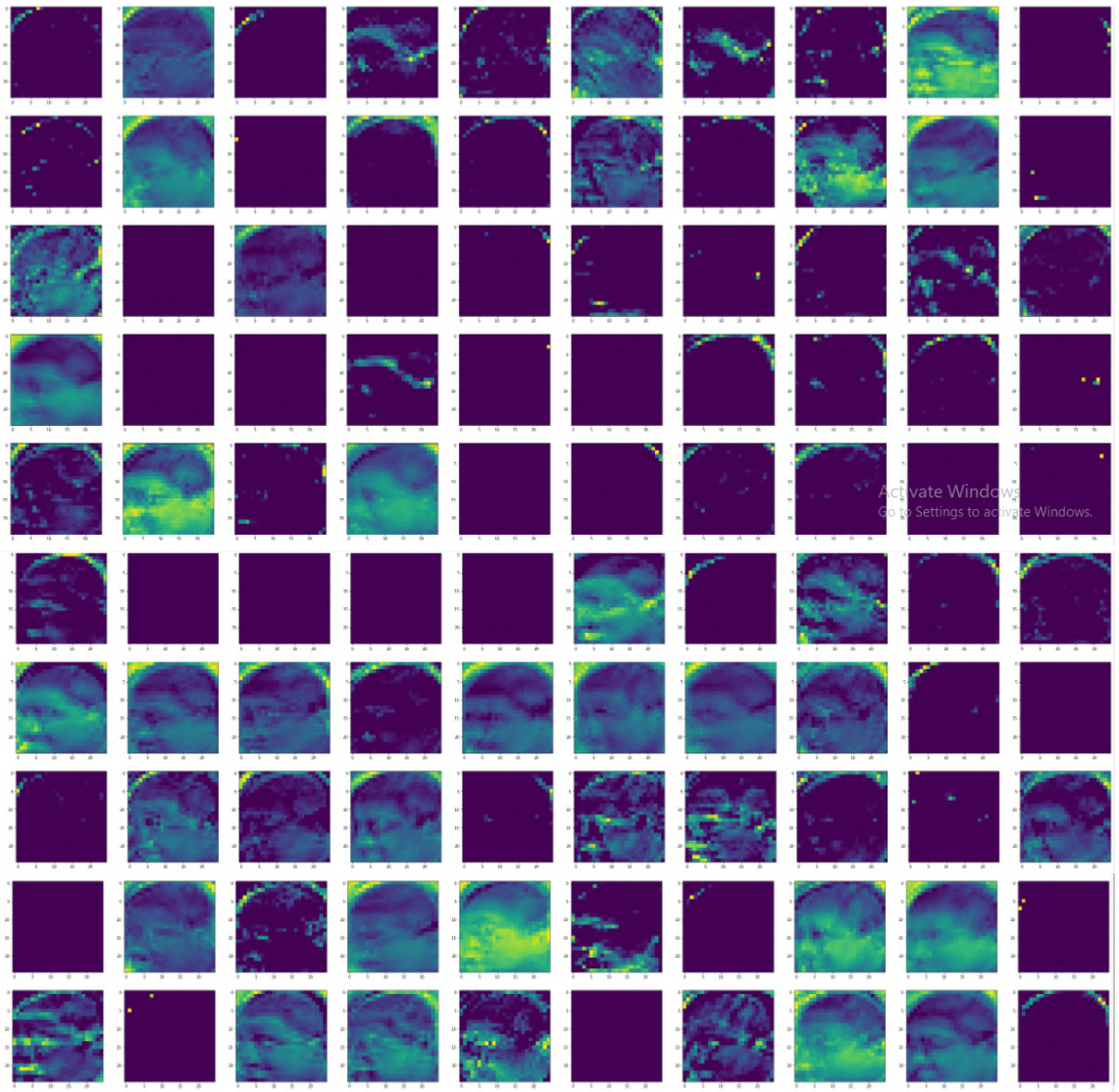


Figure 4.9: Feature map of convolutional layer 3.

Convolutional Layer 4: For fourth convolutional layer, 100 3x3 filters with the ReLU activation function were applied. The weights in the filters were learned during the training of the layer. The filters that operated on the output of the third line layer was used to extract features that are combinations of lower-level features. To illustrate, features that contained multiple lines to demonstrate shapes [39].The image data after fourth convolutional layer can be seen in figure 4.10.

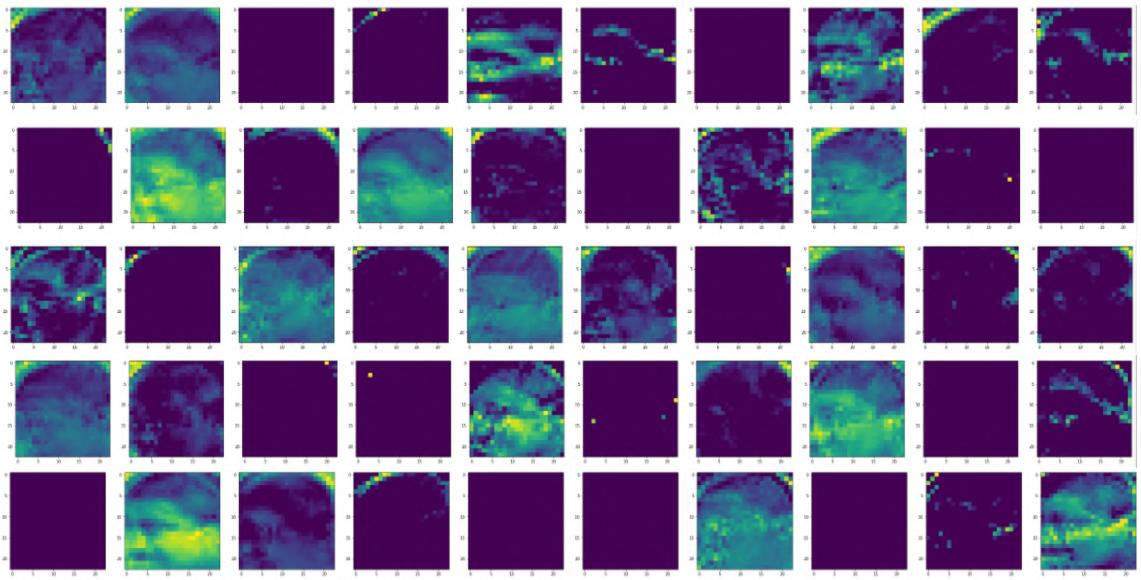


Figure 4.10: Feature map of convolutional layer 4.

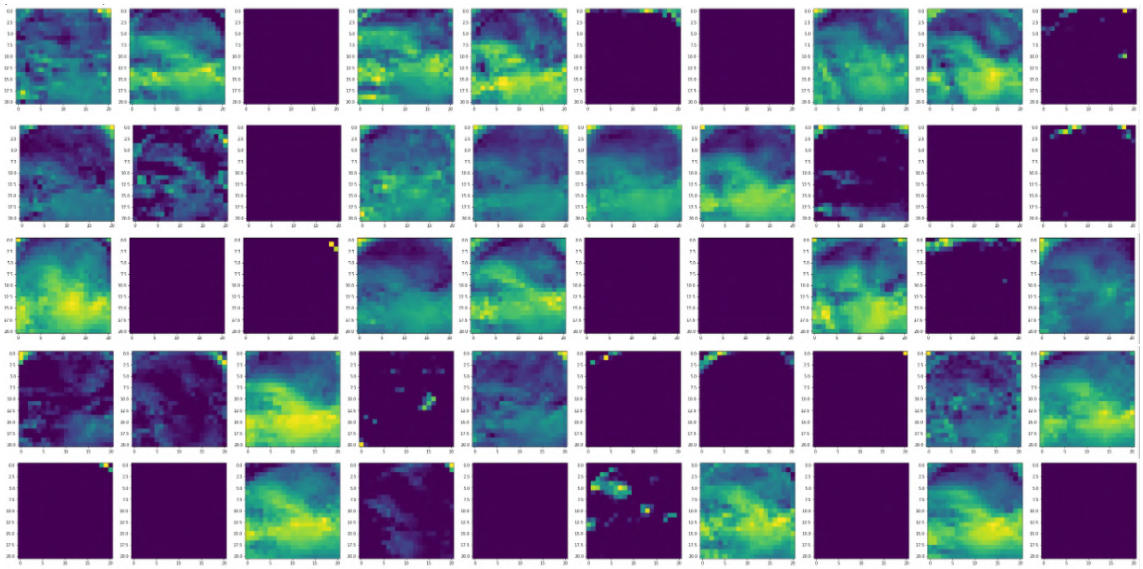


Figure 4.11: Feature map of convolutional layer 5.

Convolutional Layer 5: We applied 50 3x3 filters with the ReLU activation function for the last convolutional layer. This process of multiple layering is called deep learning. This improves model performance.Finally, we can observe the feature map of our last convolutional layer in figure 4.11.

Pooling Layer 2: The same 5x5 max pooling filter was implemented for pooling layer 2. The last convolutional layer values was given to the second pooling layer as input. This reduced the number of parameters. That shortened the training time and prevented overfitting. Pooling layer 2 was used to downsample each feature map separately. Thus the height and width were reduced. But the depth was intact [21].The output of second pooling layer is shown in figure 4.12.

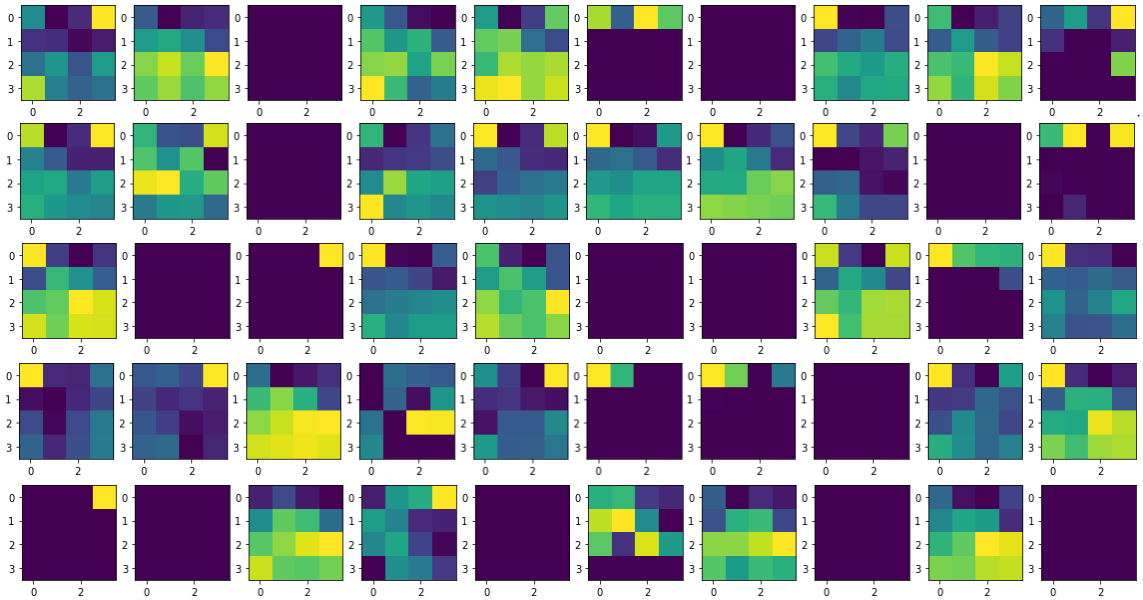


Figure 4.12: Feature map of pooling layer 2.

Flatten layers: 'Flatten' layers lie between the convolutional layer and the FC layer. The basic activity of flattening is to convert a two-dimensional matrix of features into a vector. After that, we fed the vector to a fully connected neural network classifier [51]. We used flattening to reshape the data into a 1-dimensional array. Then we gave it to the following layer as input. A single long feature vector was created when we flattened the output of the convolutional layers. In this way, connections were created to the final FC layer [32].

Dense Layer and Dropout layer: Dense layer is used for linear operation. This connects every input to every output using a weight [17]. A neural network was formed at the end in dense layers. This was required to classify the features [16]. We added four dense layers using Dense() method. An activation function ReLU was used as parameter. Our final dense layer was implemented with an activation function softmax. Softmax was applied because it converted the output of the last layer in the neural network into a probability distribution by normalization. For example: [Dense (4,activation='softmax')]. Dropout layer prevents the model from overfitting. In dropout layer, the outputs were randomly subsampled. This eventually results in reducing the capacity or thinning the network at the time of training [30]. To illustrate: [Dropout(rate=0.5)]. We kept the drop rate 0.5 to determine the odds of dropping out neurons instead of keeping them [31].

```
[array([[ 0.01247323,  0.03656624, -0.03951846, ..., -0.06687342,
         -0.00397734,  0.06239998],
        [ 0.02887394, -0.06940307,  0.0096105 , ..., -0.03136519,
          0.06056922,  0.07641121],
        [ 0.01515743,  0.0149968 , -0.04011446, ...,  0.0191003 ,
         -0.0329225 ,  0.02558659],
        ...,
        [-0.02453398,  0.02094942,  0.0406818 , ..., -0.0737022 ,
         -0.0327484 ,  0.05587298],
        [ 0.03618764, -0.02595341,  0.04822545, ..., -0.06919394,
         -0.0740575 ,  0.02310491],
        [-0.07335329, -0.06493972,  0.0428552 , ...,  0.06275963,
          0.03550147, -0.05568036]], dtype=float32),
 array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)]
```

Figure 4.13: Values of Dense layer 1.

```
[array([[-0.06670126, -0.12944177,  0.11113489, ..., -0.10956965,
         -0.03445061,  0.08912279],
        [ 0.0948022 ,  0.04638065,  0.01399398, ...,  0.00844015,
         -0.08424556, -0.00961949],
        [-0.13648848, -0.14287192,  0.13466755, ...,  0.08911936,
         -0.13904008, -0.12065132],
        ...,
        [-0.12489688, -0.13312158,  0.05694942, ..., -0.04934772,
          0.0619666 ,  0.03119949],
        [-0.06209126, -0.12218818,  0.09419902, ..., -0.1056271 ,
         -0.04496025, -0.05593966],
        [-0.06231131,  0.10956839,  0.06951423, ..., -0.14375362,
         -0.07497197, -0.13169551]], dtype=float32),
 array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       dtype=float32)]
```

Figure 4.14: Values of Dense layer 2.

```
[array([[ 0.06124763,  0.11541028,  0.09529711, ...,  0.17447175,
         -0.10992356, -0.15022445],
        [-0.04324779, -0.08203912,  0.05857195, ..., -0.1868215 ,
          0.0364666 ,  0.01264843],
        [-0.04832034, -0.10512862,  0.13055353, ..., -0.1518012 ,
          0.04717055, -0.0219336 ],
        ...,
        [-0.12932521, -0.05783692,  0.0730405 , ...,  0.00382905,
         -0.08819313, -0.0044506 ],
        [ 0.09801425,  0.17639951,  0.02432017, ...,  0.11899294,
         -0.11503744,  0.16017283],
        [ 0.18317752, -0.14154692, -0.19046846, ..., -0.12797737,
         -0.07622981,  0.19136916]], dtype=float32),
 array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       dtype=float32)]
```

Figure 4.15: Values of Dense layer 3.

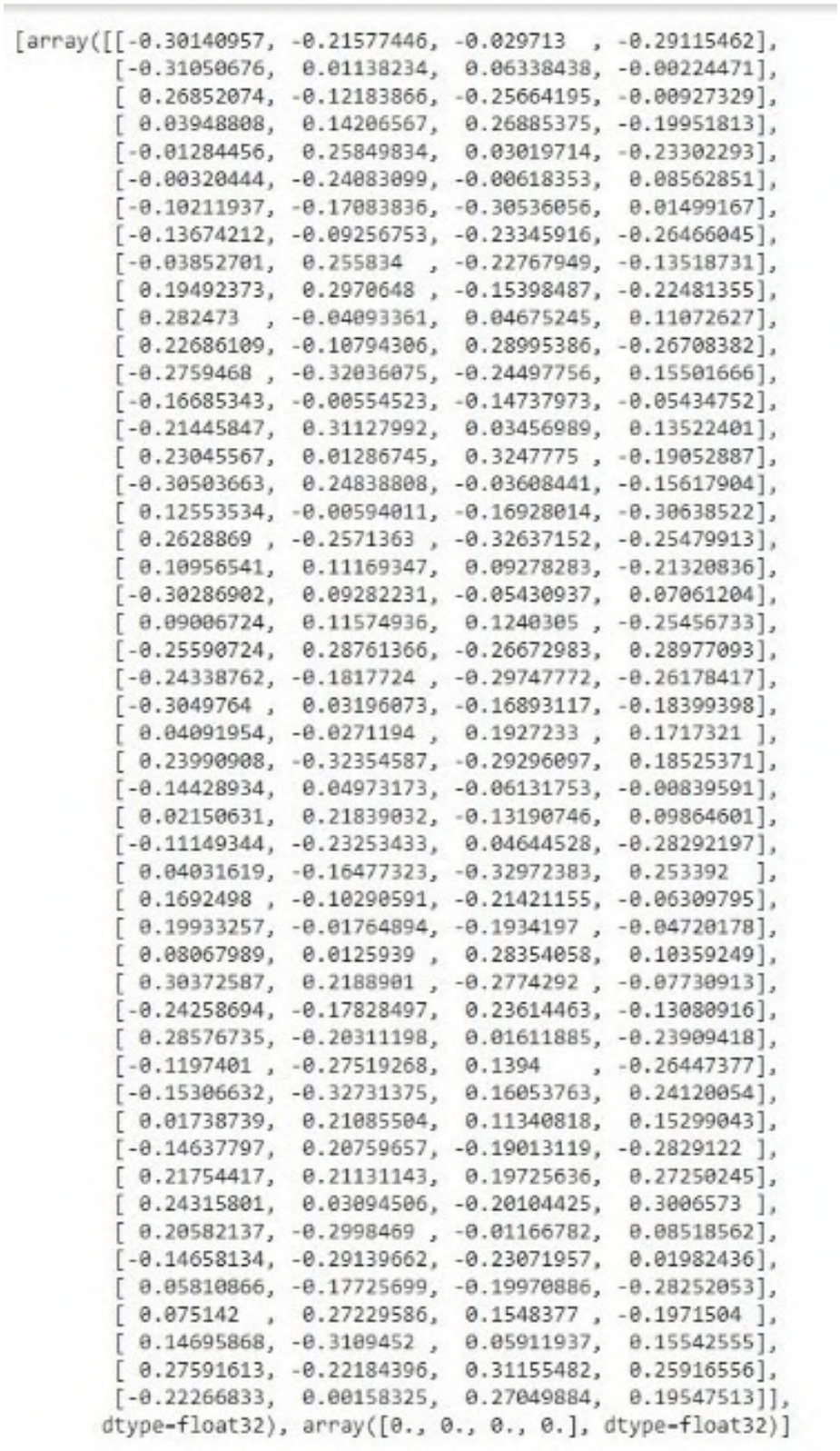


Figure 4.16: Values of Dense Layer 4.

4.2 Inception v3

A convolutional neural network which assists in analysis of images and detection of objects is called inception v3. It was started as a module for Googlenet. The model is the third edition of Google's Inception Convolutional Neural Network. Inception v3 was presented during the ImageNet Recognition Challenge originally. ImageNet is focused on a database of classified visual objects. On the other hand, Inception is used for classification of objects [40]. The usefulness of Inception v3 is it burns less computational power by improving the previous Inception architectures. Different techniques for optimizing the network are attempted for easier model adaptation in the inception v3 model. The techniques include factorized convolutions, regularization, dimension reduction, and parallelized computations [41]. The techniques used in inception v3 architecture are explained below:

- **Factorized Convolutions**: This technique is used for reducing the number of parameters involved in a network. This eventually decreases the computational efficiency. It also examines the network efficiency[41].
- **Smaller convolutions**: The technique involves replacing larger convolutions with smaller convolutions that leads to faster training[41].
- **Asymmetric convolutions**: In this technique, a 3×3 convolution is replaced by a 1×3 convolution followed by a 3×1 convolution. The number of parameters would be a bit greater than the asymmetric convolution proposed if a 3×3 convolution is replaced by a 2×2 convolution[41].
- **Auxiliary classifier**: During training, an auxiliary classifier is a small CNN placed between layers. The loss ocurred is added to the main network loss. Auxiliary classifiers were applied for a deeper network in GoogLeNet. On the other hand, an auxiliary classifier performs as a regularizer in inception v3 [41].
- **Grid size reduction**: Pooling operations are applied in this reduction technique[41].

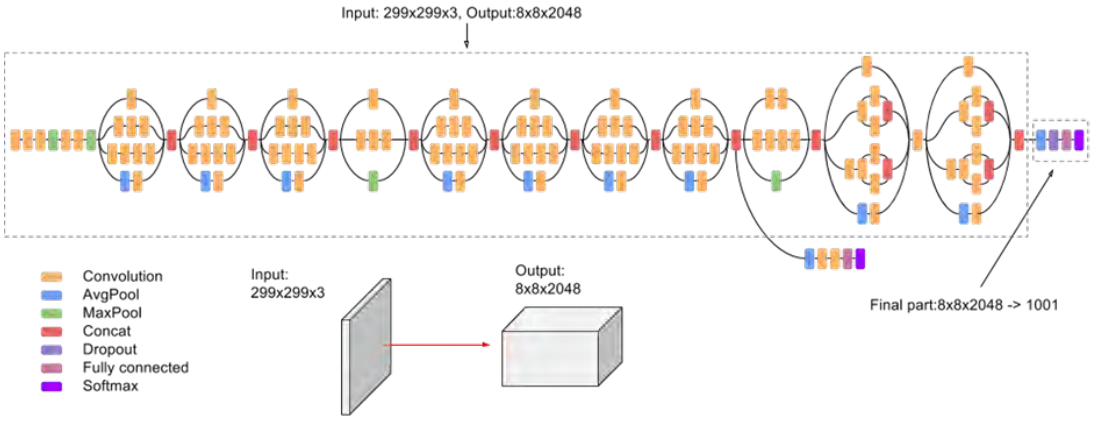


Figure 4.17: Inception v3 Architecture.

We kept the input shape (75, 75, 3). For weights we passed 'imagenet' as parameter. This was applied for pre training on ImageNet. For example, InceptionV3(input shape = (75, 75, 3), include top = False, weights = 'imagenet')[49]. Figure 4.18 shows the activation maps from first convolutional layer of our research. Here, some of the filters detect edge, some of them detect background of the image. Other filters focus on outer boundary of the image.

Figure 4.18: Activation maps from first convolutional layer.

In figure 4.19 , we can observe that some filters work on activation of surface texture. Filters have started to recognize the edges properly.

Figure 4.19: Activation maps from third convolutional layer.

Max pooling operation is done on the images here figure 4.20.

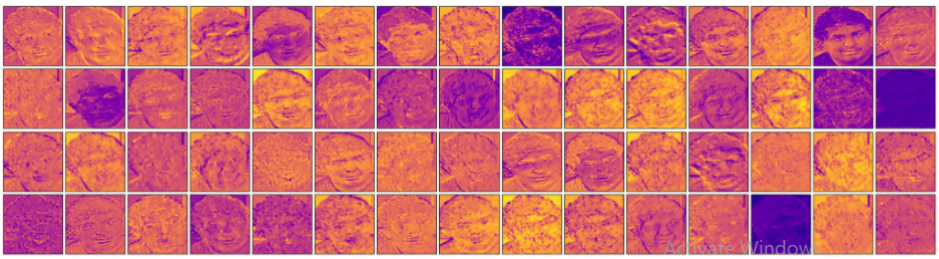


Figure 4.20: Activation maps fourth convolutional layer.

Image becomes blurry in this layer for pooling operation as shown in figure 4.21 Concatenation operation is done here. Figure **??** refer to the activation maps after concatenation of ReLU activated outputs from four convolution layers.

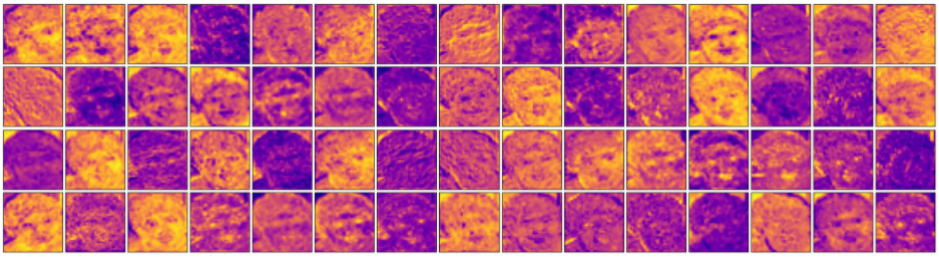


Figure 4.21: Activation maps from next convolutional layer.

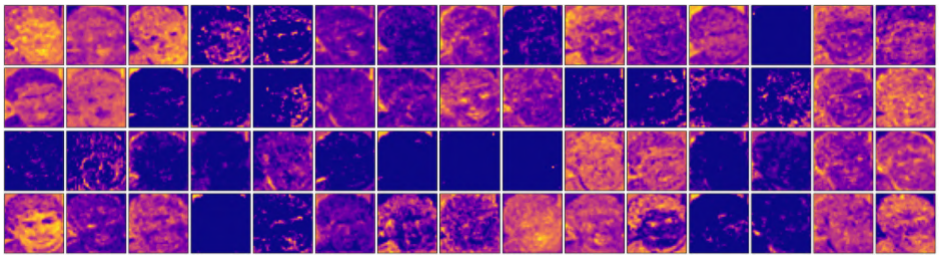


Figure 4.22: Activation maps from first concatenate layer.

4.3 VGG 16

VGG 16 is a convolutional neural network (CNN) architecture which was put forward by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University in order to win ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) competition in 2014 in their paper named "Very Deep Convolutional Networks for Large-Scale Image Recognition". It is regarded to be one of the finest vision model architecture outperforming every other architecture by miles. This model brings about 92.7% top-5 test accuracy on ImageNet dataset which consists of 14 million images originating from 1000 classes. VGG16 is different from the others because in place of a huge number of hyper parameters, focus is given on 3x3 filter convolutional layers with a stride 1 and using same padding and maxpool layer of 2x2 filter with stride 2. This pattern is followed entirely throughout the architecture. In conclusion we have 2 fully connected layers which are then lead by a softmax layer for the result. The name is VGG16 here because the 16 in the name denotes the 16 layers which contain weights. The 13 convolutional layers are all bunched up one after the other serially and the 3 other dense layers are stacked for classification. In the convolutional layers the filers are set up in an increasing order which is identical to decoder architecture of autoencoder. The network is of massive size which may contain up to 138 million (approx.) parameters.

The design of VGG is so followed by 3 simple rules which are:

1. 3x3 convolutional filters are used to increase the representational capacity of the CovNet with the equal number of parameters.

2. To prevent and avoid bottlenecks the number of convolutional filters is doubled each time the half the spatial resolution of the input.

3. To avoid the problem of saturation we implement the Rectified Linear Units (ReLU) for the activation function.

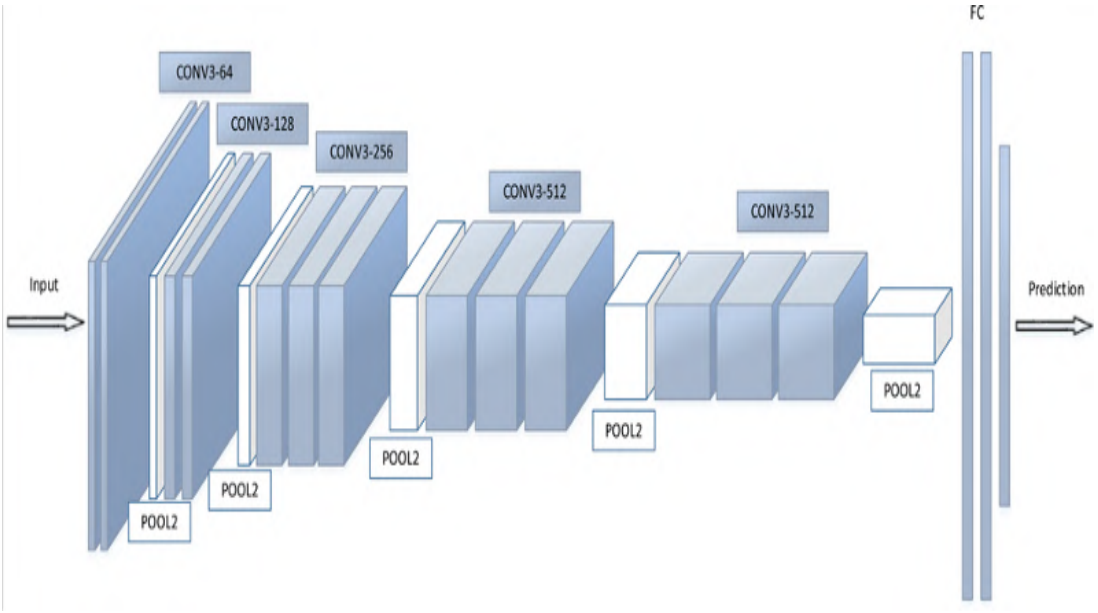


Figure 4.23: Vgg 16 Architecture [37].

The fundamental method to get transfer learning implemented is to get a pre-trained model which consists of the weights loaded and eliminate the last fully-connected layers of that model after that the rest of the portion of the model is used as a feature extractor for our very own data set which is small-scale. Bottleneck features are the extracted features which are the final activation maps before the fully-connected layers in the original model. Then a tiny fully-connected network on those extracted bottleneck features is trained in order to get the classes we need as results for our problem. To implement the VGG16 from scratch we need to firstly import all the required libraries. The sequential method is to be used to make a sequential model which denotes all the layers being arrange in order or sequence one after the other to implement the VGG16 fully. The 'ImageDataGenerator' class is imported from 'keras.preprocessing' so that the image files can be easily imported along with the labels into the model. This class is very helpful as it capable of rescale, rotate, zoom, flip and many other functions all in one. Not only that, it also does not effect the data on the disk. The ImageDataGenerator object is used to both in training and testing phase. The folder structure of the data is as follows:

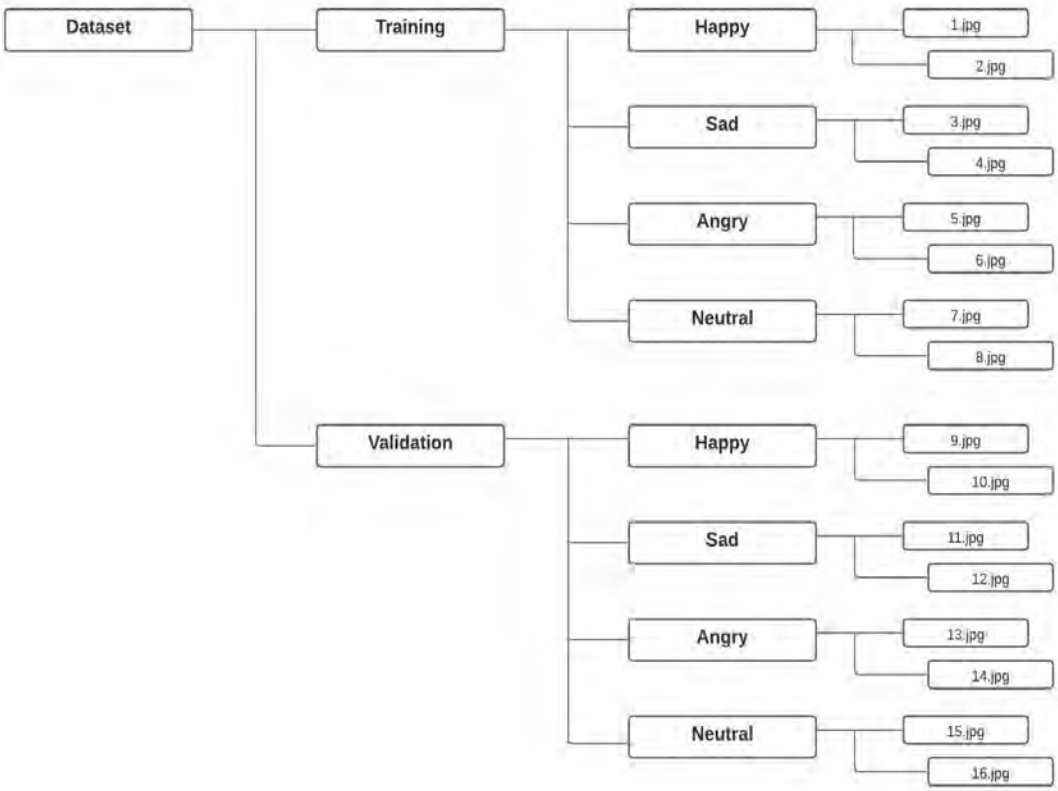


Figure 4.24: Folder Structure.

After initializing the model and setting up all the convolutional layers we attach ReLU (Rectified Linear Unit) activation to every layer so that every non positive value is not passed forward on to the next layer. Furthermore, the data is passed on to the dense layer which requires flattening of the vector. Again, ReLU activation is used so that the non-positive values are not carried forward between the network. A 4- unit dense layer is created finally along with softmax because we have 4 classes to forecast the result from in the end which are 'happy', 'sad', 'angry' and 'neutral'. We also used the Adam optimizer to reach the global minima throughout the training model. The summary of the model gives us:

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, None, None, 256)   295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, None, None, 256)   0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, None, None, 512)   0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.25: Model Summary.

We used the model.fit_generator because we used the ImageDataGenerator class to pass the data to the model. So after the train and test data are passed to fit_generator the steps_per_epoch sets the batch size according to pass the training data to the model and for the test data the validation_steps will do the same. After the model is done training we can see the accuracy and loss of our training and validation. After fully implementing we could achieve an accuracy of 98%. This model significantly outperforms the earlier generation of models. The disadvantages of VGG16 are that it is very slow to train as it is a very lengthy process, the original VGG model was trained using an NVDIA Titan GPU for 25-20days. Also, after the training the trained ImageNet weighted up to almost 528MB which means it takes up a huge amount of disk space and bandwidth resulting in its inefficiency.
From figure 4.26, we can observe all 64 feature maps. The figure refer to the output of first convolutional layer of VGG16 model after applying filters. The figure shows the same image multiple times highlighting different features of the input image. We can notice that some of the plots focus on highlighting lines. Other plots worked on background or foreground of the same image.

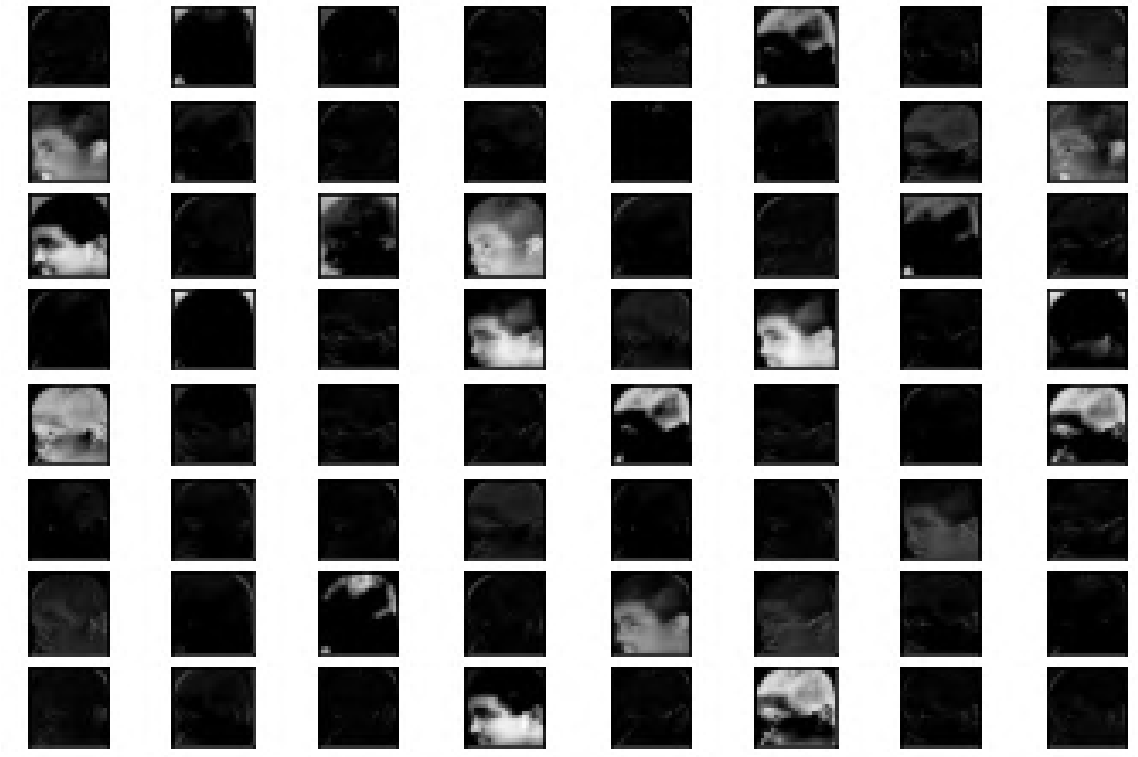


Figure 4.26: Feature Maps from the Convolutional Layer 1 (VGG16).

The figures given below are the feature maps of each block in VGG16 model. We can observe that, fine details are shown at first. After that, as we go deeper into the model, the model starts focusing on less details.

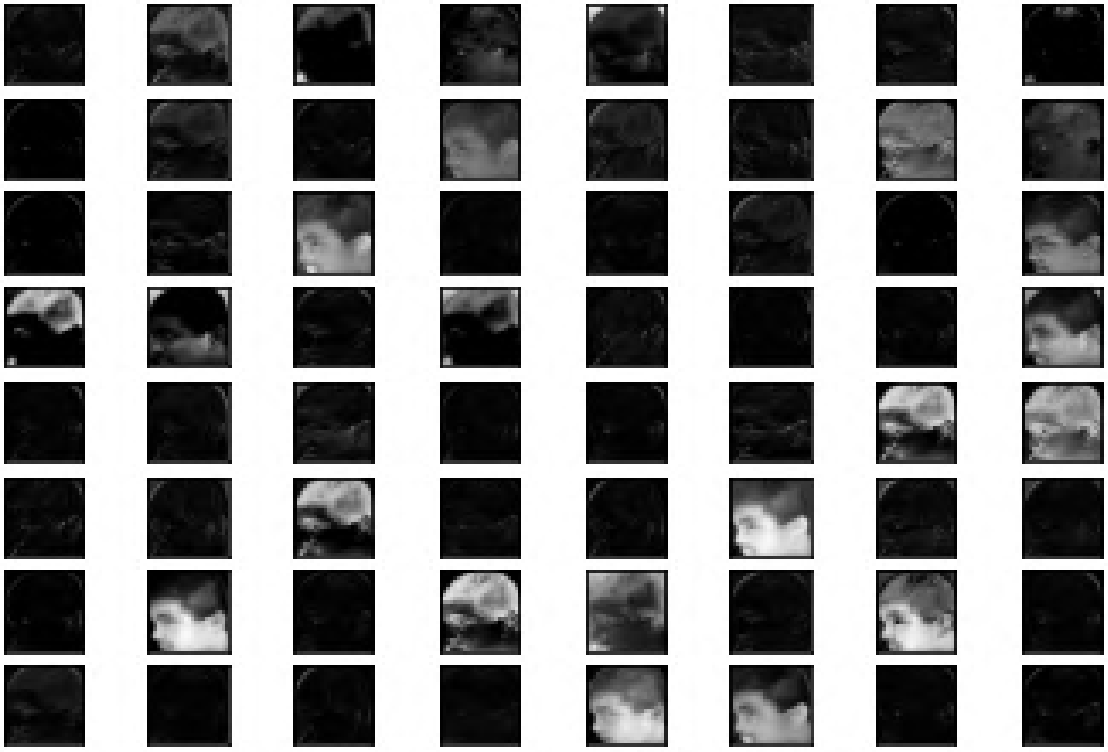


Figure 4.27: Feature Maps from Block 1 (VGG16).

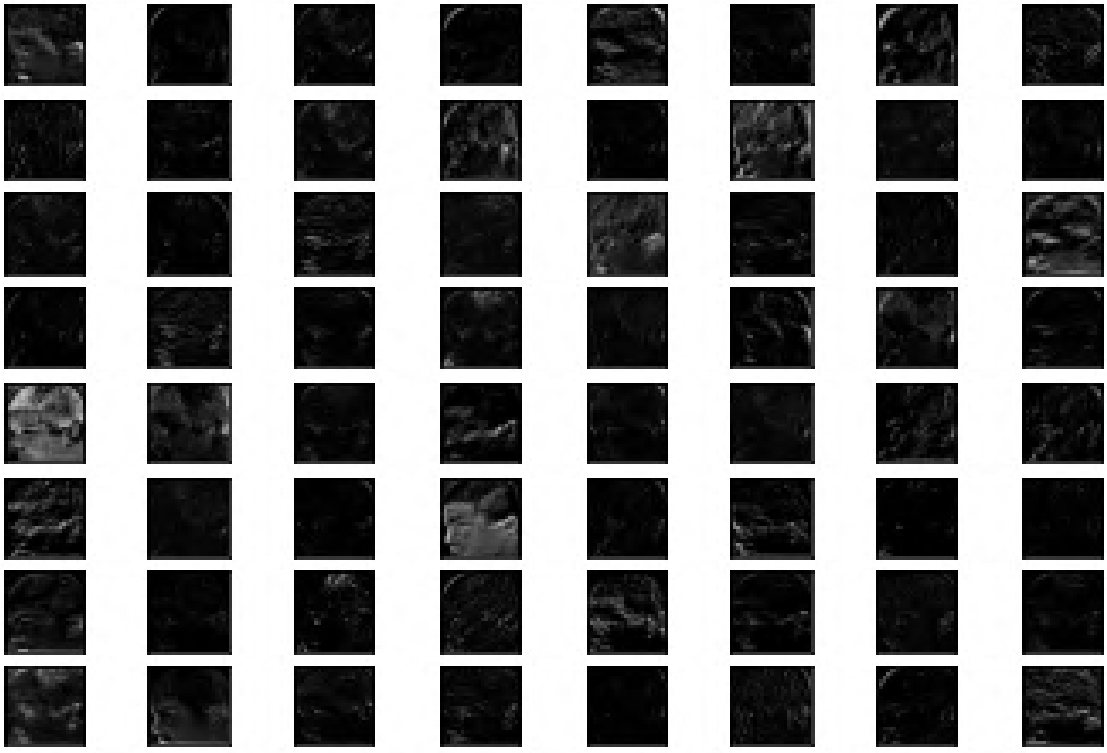


Figure 4.28: Feature Maps from Block 2 (VGG16).

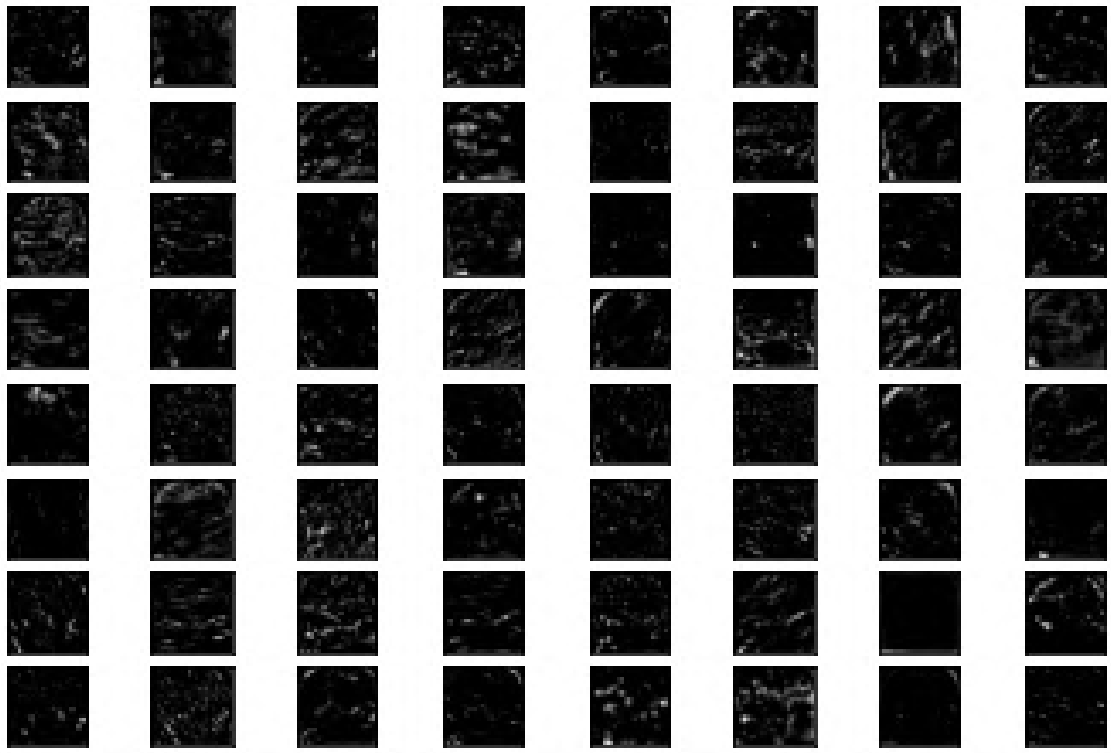


Figure 4.29: Feature Maps from Block 3 (VGG16).

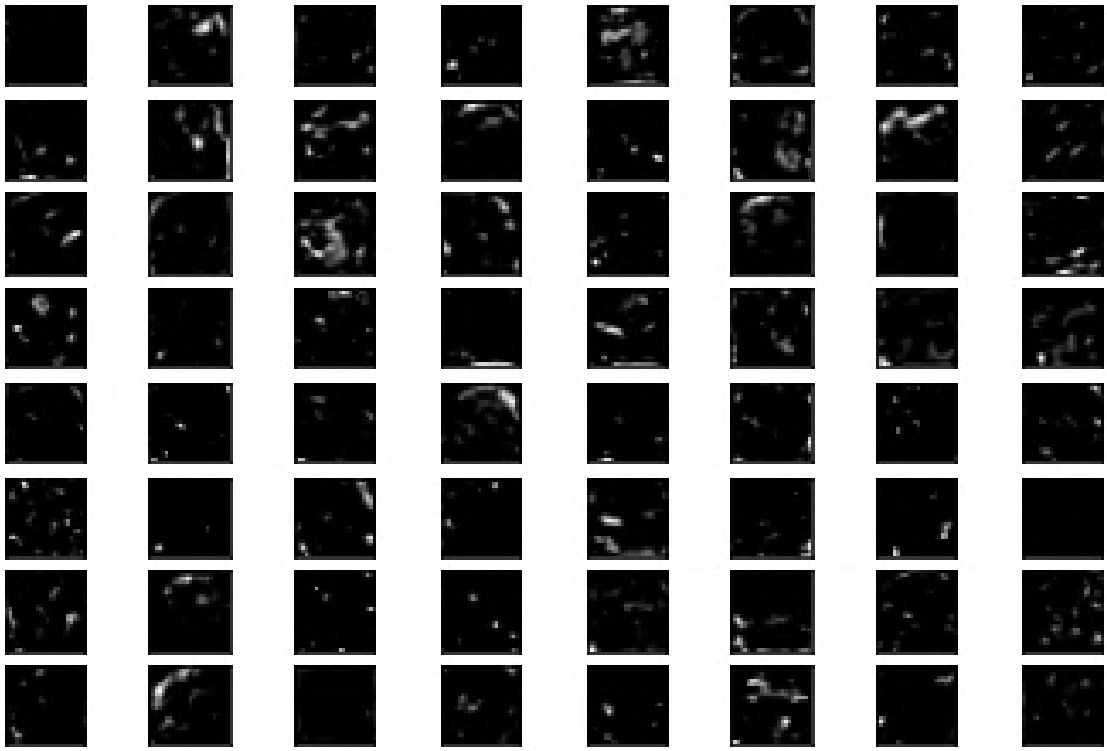


Figure 4.30: Feature Maps from Block 4 (VGG16).

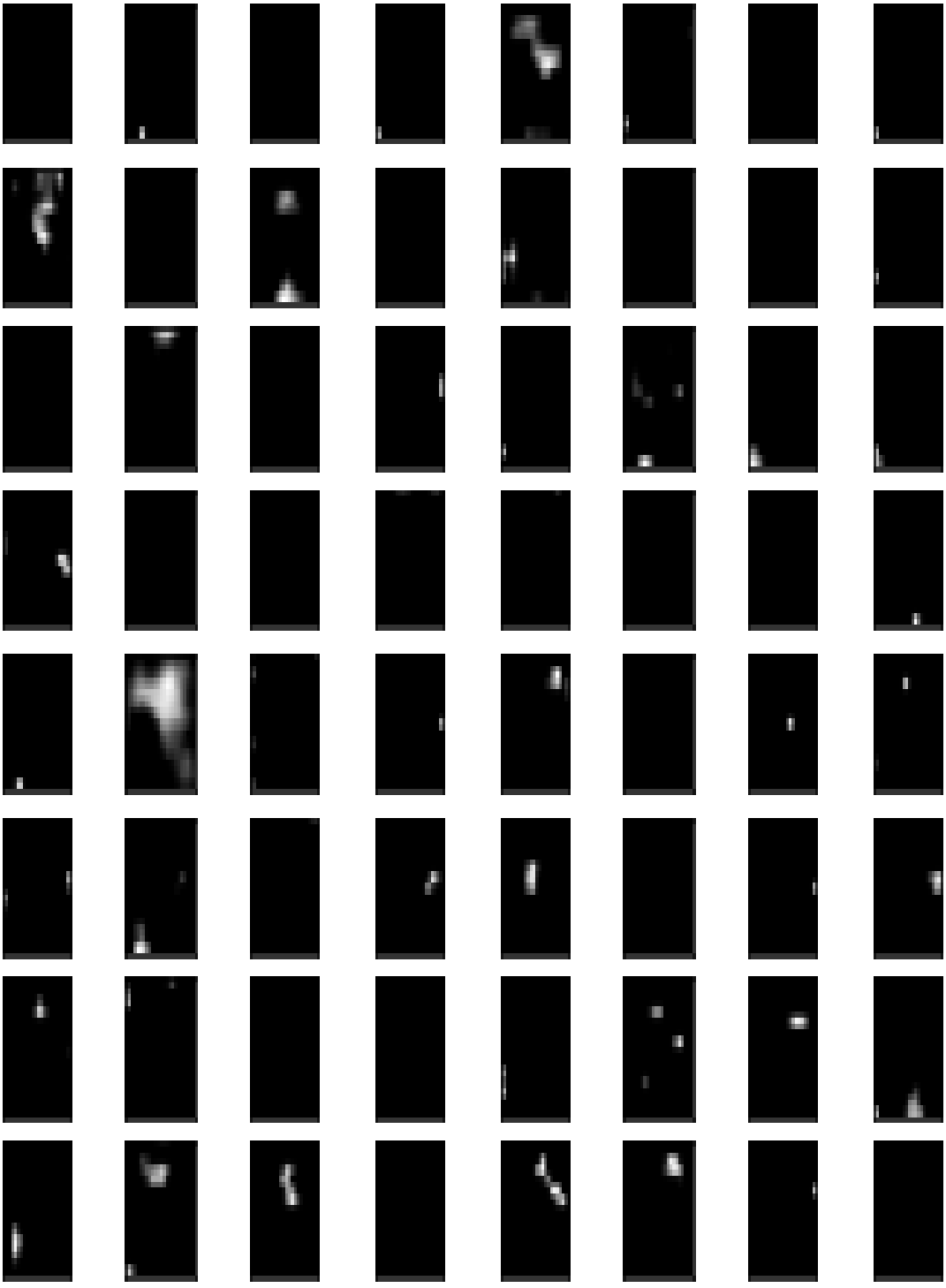


Figure 4.31: Feature Maps from Block 5 (VGG16).

4.4 SVM Classifier

Support Vector Machines (SVM) is a supervised machine learning algorithm, which is used in problems imvolving regression and classification [23]. It is most commonly used in classification if compared in general. It tries to find a fine line of segregation in the dataset, and tries to divide the data along a plane known as the hyperplane. The hyperplane divides the points on an x dimensional space, where x is the number of features or categories. This helps to categorize the data points in an organized manner. There can be many possible hyperplanes to split the data, but we need to find the one that created the greatest distance between the data points. The distance between the nearest two sample points is known as the margin, and the points are the support vector.
After using the SVM classifier, we got an accuracy of 60.48% for the emotion "sad", and this was the highest among all the other emotions.
We then applied PCA feature extraction technique with SVM classifier, and the results we obtained were as follows: 71.52% for "angry", 68.99% for "happy", 69.94% for normal, 72.15% for "sad" emotions respectively. In a nutshell, applying a feature extraction technique has vastly improved the accuracy of the results.

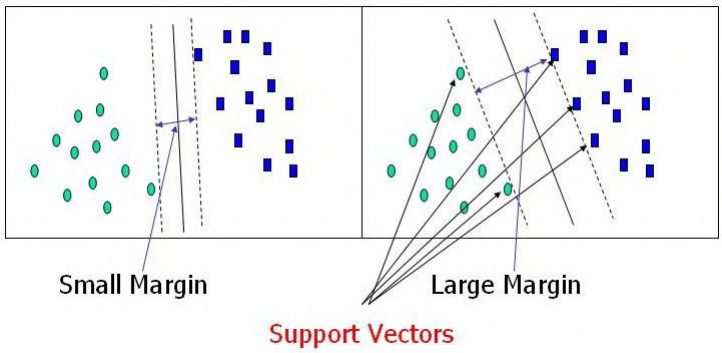


Figure 4.32: Support Vectors [23].

The hypothesis function h is defined as:

$$\mathrm{h}(\mathrm{x}_i) = \begin{cases} +1 & \text{if } w.x + b \geq 0 \\ -1 & \text{if } w.x + b < 0 \end{cases}$$

The point above or on the hyperplane will be classified as class +1, and the point below the hyperplane will be classified as class -1.
Computing the (soft-margin) SVM classifier amounts to minimizing an expression of the form:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(w.x_i - b))\right] + \lambda\|w\|^2$$

Here, choosing a sufficiently small value for lambda yields the hard-margin classifier for linearly-classifiable input data

4.5 Logistic Regression

A machine learning algorithm known as logistic regression is used for classification problems which analyzes a dataset which contains a variable that is dependent and one or more variables that are independent to project the output or result in a binary variable. In other words, it predicts the probability of a categorical dependent variable. This means it has only two outcomes: '1' for true/success/yes '0' for false/failure/no Binary classes can be forecasted using by a statistical method known as logistic regression[14]. In this algorithm the target variable is an instance of linear regression where it is categorical. It utilizes the logit function to predict the outcome of an event. The linear regression equation is as:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n \tag{4.3}$$

Here y is the dependent variable. X1, X2,... Xn are the independent variables which is used to predict the dependent value, the symbol 0 is the Y-intercept and 1, 2,... n is the slope of the line which can be positive or negative depending on the relationship between the dependent and independent variables respectively[50]. And the sigmoid function:

$$p = \frac{1}{1 + e^{-y}} \tag{4.4}$$

This gives us an 's' shaped curve which takes a real number and maps it to a binary value of 0 or 1. The value of y will become 1 if the curve is leading towards positive infinity, and if the value of y will become 0 then the curve leans towards negative infinity. If the output is more than 0.5 it is considered as 1 or 'yes/true' and if it is less than 0.5 then it is 0 or 'no/false'. Now, on linear regression we put the sigmoid function, we get the logistic regression equation:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n)}} \tag{4.5}$$

In the logistic regression algorithm, Bernoulli Distribution leads the dependent variable and estimations are made through maximum similarity. There are three categories of logistic regression, they are:

- Binary Logistic Regression: The target variable has only two possible outcomes that is 0 or 1.
- Multinomial Logistic Regression: The target variable has three or more nominal categories
- Ordinal Logistic Regression: the target variable has three or more ordinal categories

In order to implement the logistic regression model is to import and install all the required packages in Python. These include pandas, sklearn, tensorflow and matplotlib. Secondly, we need to collect the data. The dataset is loaded from the disk and then the features are selected by dividing the dataset to dependent (target) variable which is represented as 'y' and independent (feature) variables represented by 'X'. Then we split the dataset to training set and test set using train test split method. We used 75% of our dataset for training and the rest 25% as test set and

set random state = 42. In order to train our model, we need to run the fit method on the 'LogisticRegression' object and input our xtrainingdata and ytrainingdata variables [48]. Now, for the prediction of the model to be executed the logistic regression function is implemented through importing the logistic regression model in the sklearn module. Thus, the model will be trained and will be ready for testing phase. After testing phase we achieve an accuracy of 56%. This model does not require high computation power and is easy to interpret and implement because of its efficiency and simple nature. Another advantage of logistic regression does not need scaling of features [50]. On the contrary, the model cannot deal with huge number of categorical variables and is vulnerable to overfitting. And if the independent variables are not correlated or similar to the target variable then this model does not perform well. This model is widely used by researchers, statisticians, data scientists in predictive analytics [50].

4.6 Decision Tree Classifier

Decision Tree is a supervised machine learning algorithm that is one of the easiest and simplest classification algorithms to learn and interpret. A decision tree is like a flowchart tree structure. Here an interior node denotes feature or attribute, the branch denotes a decision rule, and the leaf node denotes the result[13]. The main purpose of a Decision Tree is to make a training model that is used to predict the class of the result variable by learning simple decision rules that were found from previous training data. The decision tree the class labels are predicted by firstly starting from the root of the entire tree and gradually comparing the values of the root feature to the record's feature. It is a bit of a challenge to initialize what attributes to pick as root node for that reason, there are various different attribute selection measures to identify what to initialize as root node at each level. As a result of the comparison the branch corresponding to the class label is selected and move further on to the next node. The decision tree uses the Sum of Products (SOP) which is also known as Disjunctive Normal Form[10]. Each branch from the root to a leaf node of similar class is conjunction i.e. product of values and dissimilar branches that end in that class form disjunction i.e. summation. Decision trees are the closest to human level thinking so it makes more accurate interpretations. This machine learning algorithm is a white-box type of algorithm that shares internal decision-making logic unlike other black-box algorithms like SVM or Neural Networks where the logic for data interpretation is not visible[19]. It is non-parametric and without any distribution that is independent of probability distribution assumptions. Decision trees are able to cope with huge dimension of data with high accuracy. There are two types of decision trees based on the target variable which are Categorical Variable Decision Tree where the tree that has a categorical target variable and the other being Continuous Variable Decision Tree where the tree has a continuous target variable [44].

While implementing a decision tree some assumptions are required to be made. Firstly, the entire training data set is regarded as the root. Secondly, the feature values are preferred to be categorical but if they are continuous then they are represented using discrete values or quantities. The records are then spread out recursively in order of attributes. Then for organizing attributes as roots different statistical processes are used. There are two widely known attribute selection mea-

sure (ASM) which are Information Gain and Gini Index, through which the best attribute for the node can be chosen of a tree[34]. We used information gain to build our decision tree algorithm.

Information Gain: It is the calculation of change in entropy after splitting the dataset on the basis of attributes. After calculating the value of information gain, we can segment the node and construct the tree. The attribute with the highest information gain is selected. Entropy is the measure of randomness or impurity in the input dataset. The decrease in entropy is basically information gain. That is why a decision tree always attempts to maximize the value of information gain. Information gain is calculated using the formula:

$$InformationGain = Entropy(S) - [(WeightedAvg) * Entropy(eachfeature) \quad (4.6)$$

And entropy is calculated as:

$$Entropy(s) = \sum_{i=1}^{c} -p_i \log_2 p_i \quad (4.7)$$

The decision tree algorithm was implemented using python by firstly installing all the necessary packages in python and then loading the dataset from the existing directory. We upload the zip file and extract all the images from the compressed folder. Then we set all the four categories of emotion that are 'happy', 'sad', 'angry' and 'neutral'. Furthermore, the images are all converted to grayscale and then resized at width = 150 and height = 150. The dataset is then split into training and test data where the training data is 75% and the test data is the rest of the 25%. The training data is also shuffled so that there is no biasedness[29]. Then we fit the model to the training set using the 'DesicionTreeClassifier' class where we pass the two parameters criterion and random state[30]. Here the criterion calculates the quality of split and the other one produces random states. And finally, the test data is used to predict the outcome using a new prediction vector. After implementing the algorithm, we obtained an accuracy of 46% for our decision tree classifier.
The advantages of decision tree are that it is really simple to understand and it follows the same process as of the human thinking to make decisions. It also figures out all possible outcomes from a problem and it requires relatively less data compared to other algorithms. While there are various advantages, there are some disadvantages to which include, lots of layers making it complex and bulky and the overfitting issues. Also, if the class labels increase then the computational complexity increases greatly.

4.7 Random Forest Classifier

Random forest algorithm is a supervised machine learning classification algorithm. Random Forest, or simply RF, is a combination of multiple decision trees, where each tree provides a classification, and the final output label from the RF is the

modal result that is given by all of the decision trees, or sometimes it is even the mean value from all the decision trees. It consists of a very high discrimination performance, training classification and higher recognition rate with a comparatively low cost of computation, and also reduces overfitting as a result [11]. Random forest is demonstrated to be giving better results as compared with those of the most well-known classifiers when it comes to accuracy in several tasks, including facial recognition, as it also involves cross validation [12]. This algorithm can be used both for classification and problems that involve regression. The higher the number of decision trees, the better accuracy is obtained. For the classification process, suitable feature vectors that are able to characterize are highly suggested to be used. In this paper, we implement a blueprint for facial expression classification, which utilizes the random forest as the classifier, and we compared it on the basis of features gathered from principal component analysis, and also compared with Local Binary Pattern [8]. Although we know that PCA has drawbacks, the first one is that it is hard to assume the covariance matrix and secondly, it is computationally restrictive to extract the eigenvectors. So, we decided to use a random forest classifier to further improve the accuracy of our results. Some advantages of using RF as a classifier include straightforward learning via multi step decisions, parallelization, and faster training time [20].
The parameters in the RF code segment include estimators that represent the number of decision trees. The function to measure the quality of split is Gini index, which calculates the probability of a variable to be classified incorrectly if it is chosen after shuffling. The term entropy is used to denote the information gain[45]. Bootstrap samples are used when building trees, and if it is set to false, the entire dataset is used to build each tree.
When we used our dataset of ASD children and implemented random forest classifier, we obtained accuracies as follows:
The highest accuracy was obtained for the expression labelled “normal”, which had an accuracy of 68.28%. The other expressions had an accuracy of 63.44% for being “angry”, 63.44% for being ”happy” and 66.13% for being ”sad” respectively.
When we included the PCA feature extraction to our previous random forest classifier, we could see a slight change in the accuracy of our results as compared to the previous one. The results obtained were 62.10% for the emotion labelled “angry”, 64.52% for “happy”, 62.63% for “normal”, and 56.72% for “sad” respectively.

4.8 K-neighbours Classifier

The KNN which stands for k-nearest neighbor algorithm is one of the straightforward and easy-to-implement supervised machine learning algorithm. The algorithm can figure out classification and regression problems but it is mostly used in the industry to resolve classification problems. In the algorithm it is supposed that everything co-exists in close proximity, more briefly things that are ‘similar’ exists near each other. The KNN algorithm predicts on the basis of how closely the new data point matches the training dataset points. This algorithm is so simple because it contains only two parameters which are the distance metric (e.g. Euclidian, Manhattan, etc. distances) and the value of ‘k’ (e.g. 1, 2, 3...). The distance metric is needed to be defined in order to apply the KNN algorithm. Some of the most commonly used distance metric include:

$$\sqrt{\sum_{i=1}^{k} (x_i - y_i)^2} \tag{4.8}$$

$$\sum_{i=1}^{k} |x_i - y_i| \tag{4.9}$$

$$\left(\sum_{i=1}^{k} (|x_i - y_i|)^q \right)^{\frac{1}{q}} \tag{4.10}$$

The equation 4.8 is for euclidian distance, the equation 4.9 is for Manhattan distance and the equation 4.10 is for Minkowski series distance.

The algorithm is implemented firstly, by importing all the required Python packages or installing the packages if not already installed. Next, we finish loading the training and test data from the train and test directory. Then, all the images are resized to a certain dimension of width = 150 and height = 150 and then flattened[39]. The images are converted to a feature vector which is basically a list of numbers that denote the details of the images. This results in transforming the images to 150 x 150 pixels, and given 3 channels for each RGB component respectively[43]. The feature vector should contain a list of 150 * 150 * 3 = 67500 numbers. The images are then transformed to the HSV (Hue, Saturation, Value) color space in order to build a color histogram to describe the color spread of the images. The value of 'k' (an integer) which is the closest data points is selected, in our algorithm the value of k has been set to 1 where it chooses the nearest data point. We need to be careful that the value of k is an odd number to avoid a tiebreaker. We take our dataset and split it into two division which are training and testing. We split it such that 75% is used for the training phase and the rest 25% is used for testing phase. In the test data for every point and row of training data we measure the distance metric using the Euclidean distance formula and sort them in an ascending order [43]. Now in the array which is sorted the upper k rows are chosen and assigned a class to the test point on the basis of the nearest class that could be found. After the execution of the algorithm is complete, the accuracy we got was 72%. This was the highest accuracy we obtained after implementing all the seven machine learning algorithms. There are various advantages of using this algorithm as it does not require building models or setting various parameters. Basically, this machine learning algorithm in fact does not learn anything. And, as there is no presuming of data it is good for non-linear data. This algorithm usually yields high accuracy but there are some supervised learning models which surpass the KNN algorithm. Some of the disadvantages of this algorithm include it being computationally expensive as all the training data are stored on the disk, and it gets evidently slower with the increased number of predictors or variables.

4.9 Naïve Bayes Classifier

One of the most widely known statistical classification technique is based on Bayes' Theorem which assumes an independence among the predictors. Naïve Bayes techniques are a collection of supervised learning algorithms with the assumption of

conditional independence between each feature where the values of class variables are given. Basically, Naïve Bayes' classifier presumes that the existence of a certain feature or quality in a class is not related to the existence of any other feature or quality. This is one of the simplest supervised learning algorithms that is fast, accurate and stable. The algorithm works even more efficiently by increasing accuracy and speed on large datasets. The Naïve Bayes' classifiers are implemented on Bayesian classification techniques. The Bayes theorem:
The above equation explains the relationship of conditional probabilities of statistical quantities. Here P(h) is the possibility of 'h' being true regardless of the other data and it is called 'prior probability of h'. Secondly, the probability of the data regardless of the hypothesis is known as prior probability. Then P(h|D) is the probability of hypothesis given that the data 'D' occurs which is known as posterior probability. The classification algorithm Naïve Bayes' is for two-class (binary) and multiclass classification. It does not calculate the probabilities of each attribute value rather supposes that the attributes are conditionally independent. Naïve Bayes' itself being so simplified and is used in many real-life situations such as document classification, news categorization, medical diagnosis, weather prediction and spam filtering. The classifier can be very fast compared to other classification techniques. There are a few types of Naïve Bayes' classifiers. They are:

- **Multinomial Naive Bayes**: The features or predictors used by the classifier are frequency of words existing in a document. This is usually used for classification problem of documents such as whether a document is of category entertainment, politics or technology etc.
- **Bernoulli Naive Bayes**: The variables of the classifier are Boolean variables but very similar to the multinomial naive bayes classifier. The parameters here are only yes/true or no/false.
- **Gaussian Naive Bayes**: The predictors are continuous values and are not discrete which are sampled from gaussian distribution. The values existing in the dataset varies so the conditional probability changes accordingly.

In order to implement the Naïve Bayes' classifier, we firstly import the required packages namely NumPy, Pandas, GaussianNB, train_test_split, accuracy_score and Matplotlib. Then load all the dataset from our disk. After loading all the data, we can manually choose to read the files if we wish to. We require doing some preprocessing before feeding the data to the Naïve Bayes' classifier model by creating the two required variables x and y from the dataset. We then split the training and test dataset into two parts of 75% and 25% respectively using the train_test_split function of scikit-learn. We then summarize the training data by calculating the mean and standard deviation of each feature or attribute by the given class value. They are required in order to make predictions to calculate the probability of certain feature or attribute values belonging to each class value. The 'GaussianNB ()' method is used along with the model 'fit' to create our model. Then we can finally make predictions using the summaries created from our training data. After finishing training, the model, can successfully make predictions. We can use the predict method on the model and pass x_test as a parameter to get the output as y_pred. The accuracy may vary according to the training and test split ratio. The obtained accuracy

from our algorithm was 46%. The Naïve Bayes' classifier is not only simple but also really fast and accurate under such low computational cost. Also, it can easily work on a large dataset. Furthermore, it can run well in discrete response variable compared to continuous variable. It is versatile as it can also handle multiple class prediction. On the contrary the assumption of independent features is impossible as it is not realistic to obtain a set of completely independent predictors. The model can fail sometimes to make prediction if there are no training tuples of a certain class, this problem is known as Zero Probability/Frequency Problem, this hinders the performance of the classifier.

4.10 Linear Discriminant Analysis

Linear discriminant analysis or LDA is a supervised machine learning technique which identifies a linear sequence of attributes that distinguishes two or more classes of objects or occurrences. This can be used to model a separation in different groups or separating two or more classes [16]. The separation of classes is completed by calculating the linear discriminants that denote the axis which intensifies the separation of different classes. The technique was initially named Linear Discriminant or Fisher's Discriminant Analysis in 1936 by Ronald A. Fisher which was a two-class technique[3]. Later on, a multiple class variant was implemented known as multiple discriminant analysis by C.R Rao[3]. Among the different classification techniques of data, linear discriminant analysis (LDA) is one of them. Just like Principal Component Analysis (PCA) technique LDA is also another commonly used method to classify data and reduce dimension. Linear discriminant analysis basically casts the dataset from a higher dimension-space to a lower dimensional-space with a class of distinguishable features that minimize overfitting and computational costs. LDA can be applied in various fields for example it can be used for face recognition, disease state of patients, identification of customers and various other predictions and learning.
For converting the features from higher dimensional space to lower dimensional space we need to follow a few steps. The class variance among the different classes is the distance between the mean of the different classes which needs to be calculated[1]. It is known as between-class variance and is calculated by:

$$S_b = \sum_{i=1}^{g} N_i \left(\overline{x_i} - \overline{x}\right)\left(\overline{x_i} - \overline{x}\right)^T \tag{4.11}$$

Then we determine the distance between the mean and sample of every class which is known as within-class variance. It can be calculated using:

$$S_w = \sum_{i=1}^{g} (N_i - 1) S_i = \sum_{i=1}^{g} \sum_{j=1}^{N_i} \left(x_{i,j} - \overline{x}_i\right)\left(x_{i,j} - \overline{x_i}\right)^T \tag{4.12}$$

Then we finally get the lower-dimension space which increases the between-class variance and decreases the within-class variance. Here, P is supposed to be the lower-dimension space projection or Fisher's criterion[2]. It is written as:

$$P_{lda} = \arg\max_{P} \frac{\left|P^T S_b P\right|}{\left|P^T S_w P\right|} \tag{4.13}$$

Linear discriminant analysis presumes that every feature of the dataset is a gaussian distribution which is more of a bell-shaped curve[5]. Also, every feature consist similar variance and each feature is supposed to be randomly sampled. Linear discriminant analysis model is based on Bayes' Theorem to estimate probabilities[9]. The predictions are made on the basis of the probability that every new input dataset corresponds to each class. The highest probability class is chosen as the output class and thus it makes a prediction. Linear Discriminant Analysis model can be implemented by using Python firstly by importing all the necessary libraries. We import and load the dataset from the data directory and separate the dependent and independent variables. Our dependent variable has got four classes, 'happy', 'sad', 'angry' and 'neutral'. Then feature vectors are created for each class and saved. After loading the data, we need to divide the dataset to the features and corresponding labels. Then it is divided to training and test set. Just like the principal component analysis model we require to do feature scaling too. Now, using scikit-learn we perform the linear discriminant analysis model from the 'LinearDiscriminantAnalysis' class of the 'sklearn.discriminant_analysis' library in Python[46]. We run the 'fit' and 'transform' methods to obtain the linear discriminants. The 'transform' method requires two parameters which are 'X_train' and the 'y_train' unlike the principal component analysis model where it requires only one parameter 'X-train'. The algorithm achieved an accuracy of 52.419% after evaluating the performance.

4.11 Feature Extraction

Feature extraction improves accuracy, reduces overfitting risk in Machine Learning. This also speeds up the training time of the model. Feature Extraction reduces the number of features in a dataset. Then new features are created from the existing ones. This new decreased set of features is the summary of the most of the information which was in the original set of features.

4.11.1 Principle Components Analysis (PCA)

A method that is used to decrease the dimension of a very big dataset by converting the huge set of variables or attributes to a much smaller dataset is called Principal Component Analysis. This process does not lose any information in the huge dataset. As we know that large datasets can be a huge and difficult task to interpret, this can be used to minimize the problems[6]. By increasing the variance, it makes new uncorrelated variables or attributes. PCA mitigates various problems such as computational cost and increased error rate from multiple test correction. PCA is an unsupervised learning method. The method is very much similar to clustering[4]. Reducing the data by geometrically projecting them to a lower dimension known as principal components with a view to find the best synopsis of the given data by using a limited minimum number of principal components is the main goal of PCA. We used Pandas DataFrame to make the tabular data structure two dimensional with labeled axes consisting of rows and columns. Pandas DataFrame contains three principle components, the data, rows, and columns[47]. We kept whiten= False. This improves the system accuracy sometimes [28]. Implementation of Machine Learning Algorithms after demonstrating PCA: We first converted the train

and test dataset into DataFrame. Then, we created the new model using PCA.fit on training dataset. Finally, this was passed through the algorithms that we implemented. The accuracy from Logistic Regression using PCA was 53%. We achieved the highest accuracy from SVM using PCA which was 67%. Furthermore, PCA improved the accuracy of K-Nearest Neighbors Classifier which was 55%. In addition, the accuracies of Linear Discrimination Analysis and Decision Tree Classifier were also improved while using PCA. On the other hand, accuracy were reduced in case of Gaussian Naive Bayes and Random Forest Classifier. PCA explained variance is shown in figure 4.33.

```
array([1.41231949e+07, 7.98463135e+06, 5.37448173e+06, ...,
       2.80907513e+02, 4.67071316e-25, 2.11300723e-27])
```

Figure 4.33: PCA Explained Variance.

Figure 4.34 is the distribution plot of PCA explained variance. X and Y axis represent the decomposed value. The figure is a scatter plot of PCA components.

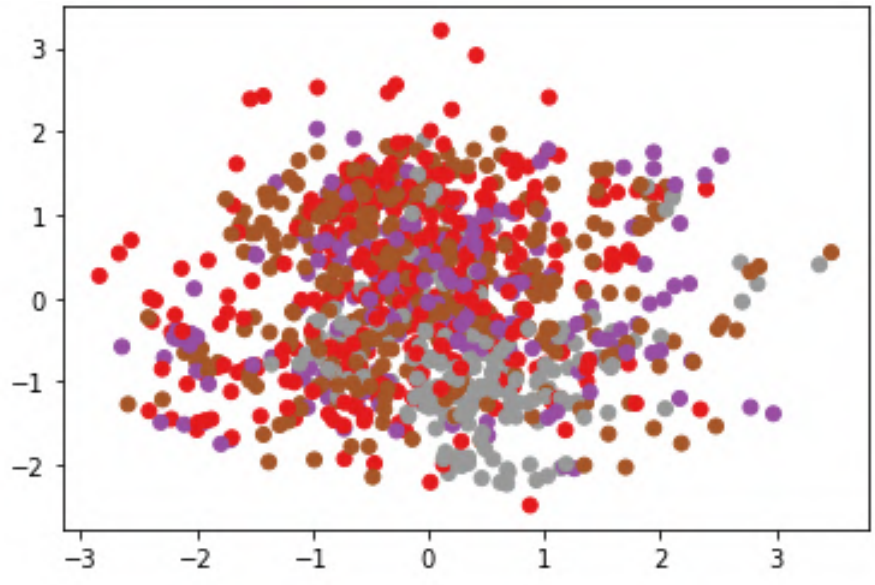


Figure 4.34: Distribution of PCA components.

Chapter 5

Performance Analysis

The algorithms that we implemented separately are- Random Forest Classifier, Support Vector Machine, Decision Tree Classifier, K-Nearest Neighbors Classifier, Logistic Regression, Gaussian Naïve Bayes and Linear Discriminant Analysis. At first, our collected and labeled data was used for training and testing. After that, we implemented all the mentioned algorithms and classifiers. We used PCA for feature extraction and compared the performance with the raw algorithms [28].

5.1 CNN

From our research, we came to a conclusion that if we increase the layers of CNN, we get the best results. That is why, we applied 5 convolutional layers, 2 pooling layers, 4 dense layers and one dropout layer in creation of the model CNN and got accuracy of 89%. We found that CNN model took more time to execute than other Convolutional Neural Networks. Figure 5.1 and figure 5.2 show the internal accuracy and loss graph of CNN.

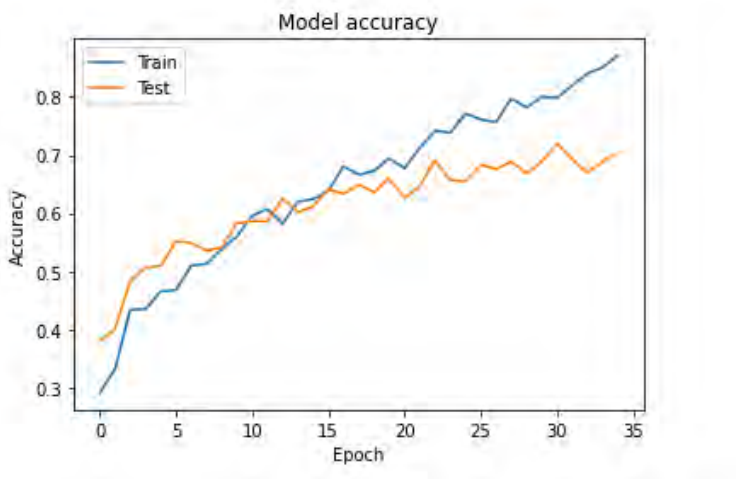


Figure 5.1: The internal Accuracy in every Epochs

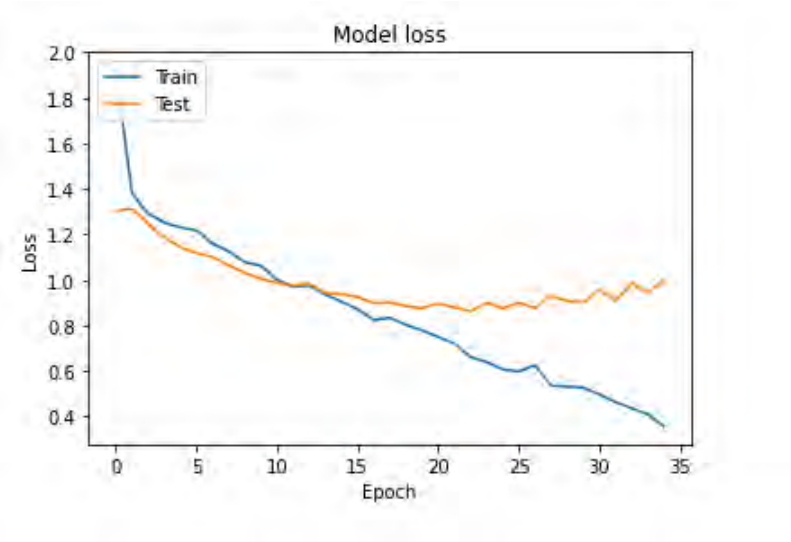


Figure 5.2: The internal loss in every Epochs

From the mentioned figures, we can see that the decrease of loss and increase of accuracy with every epoch. The testing accuracy is little bit less than training accuracy when epochs increases. This means, there might be little overfitting. We kept 25 iteration to fit the model. We showed the internal values here. The performance of CNN was satisfactory. This is because CNN can learn appropriate features from an image at different levels. This is much similar to human brain. Moreover, CNN model shares weights. CNN takes less memory. The model requires less pre-processing. These are the reasons behind efficient performance of CNN.

5.2 VGG 16

As our CNN model took a lot of time to execute so we tried improved versions of CNN. One of these are VGG16. VGG16 has a combination of 16 layers consists of convolution and fully connected layers. We trained our dataset and created the VGG model with it. Here we got the accuracy of 90.97%. The performance of VGG16 was better than CNN model that we mentioned earlier. The model also took less time to execute than CNN. We chose this model in our research because it is considered to be one of the finest vision model architecture outperforming every other architecture by miles. The internal accuracy and loss graph are shown in figure 5.3 and figure 5.4.

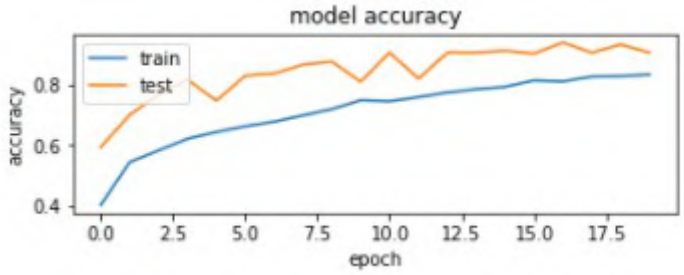


Figure 5.3: The internal Accuracy in every Epochs

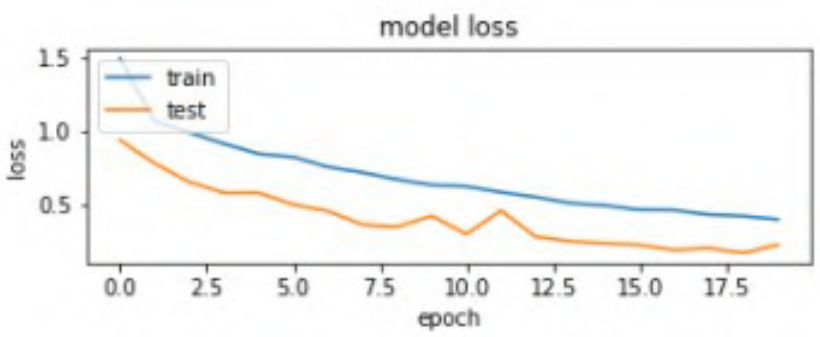


Figure 5.4: The internal loss in every Epochs.

The increase of training and testing accuracy and decrease of training and testing loss are noticed in the mentioned figures with the increased epochs. The distance between training and testing graph is too small. So we can say that the model we created was ideal. The dominance of overfitting was not there. VGG16 is an efficient architecture for benchmarking considering a particular task. VGG16 has good feature representation. That is why, we can observe that VGG16 gave us improved accuracy than CNN model. However, the model took more memory than previously mentioned CNN model.

5.3 Inception v3

Inception v3 is much more improved inception architecture of CNN than the previous versions. The model takes less computational power. We used ImageNet image database for pre training. For this reason, the model performed better and we got the accuracy of 89.17%. We found that Inception v3 took much less time to run compared to other convolutional neural networks that we used before. The model also took less memory than VGG16 network. Inception v3 is 42 layers deep. The model required fewer parameters. Besides, it had lower error rate in our case. Thus, it led us to achieve satisfactory result.

5.4 Comparison among CNN, Inception v3, and VGG 16

Figure 5.5 shows the comparison graph among these three convolutional neural network models.Comparing the three convolutional neural networks, we got the highest accuracy from VGG16 which was 90.97% and the lowest from CNN which was 89%. VGG16 and Inception v3 are deeper networks than previous models of CNN. That is why, VGG16 and inception v3 gave us the better results.

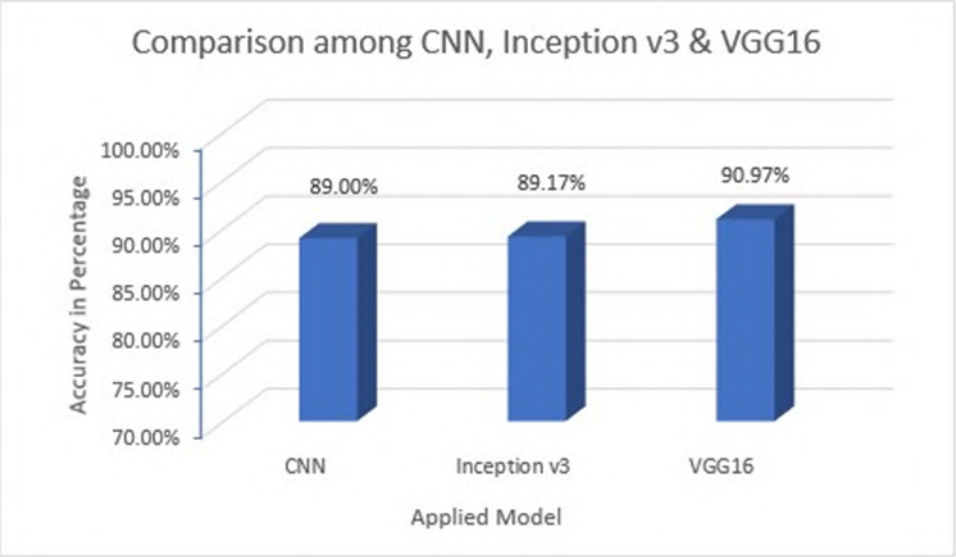


Figure 5.5: Comparison among Neural Networks.

5.5 Random Forest Classifier

Random Forest Classifier showed us the high performance in our research with 68% accuracy. The reason behind better performance of Random forest is that it can handle a large proportion of dataset which has higher dimensionality. Moreover, the algorithm prevents overfitting trees in the model when the number of trees increase. Figure 5.6, figure 5.7 and figure 5.8 show the confusion matrix, normalized confusion matrix and classification report of Random Forest Classifier.

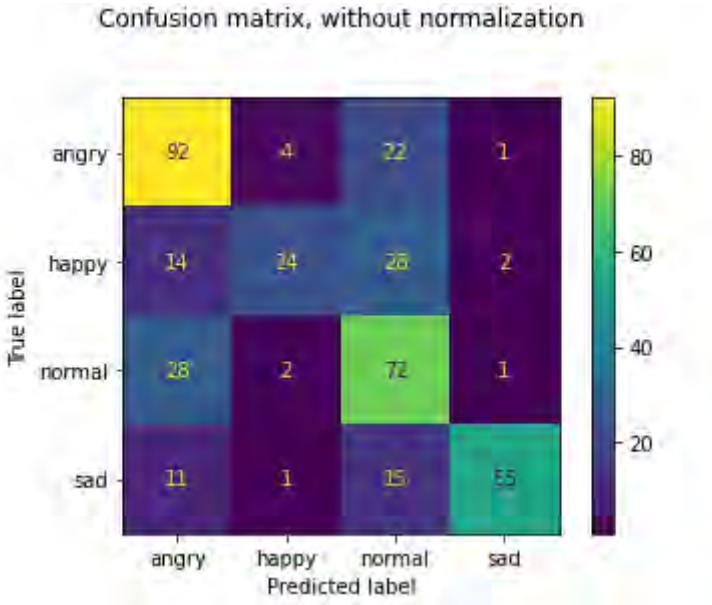


Figure 5.6: Confusion Matrix of Random Forest Classifier.

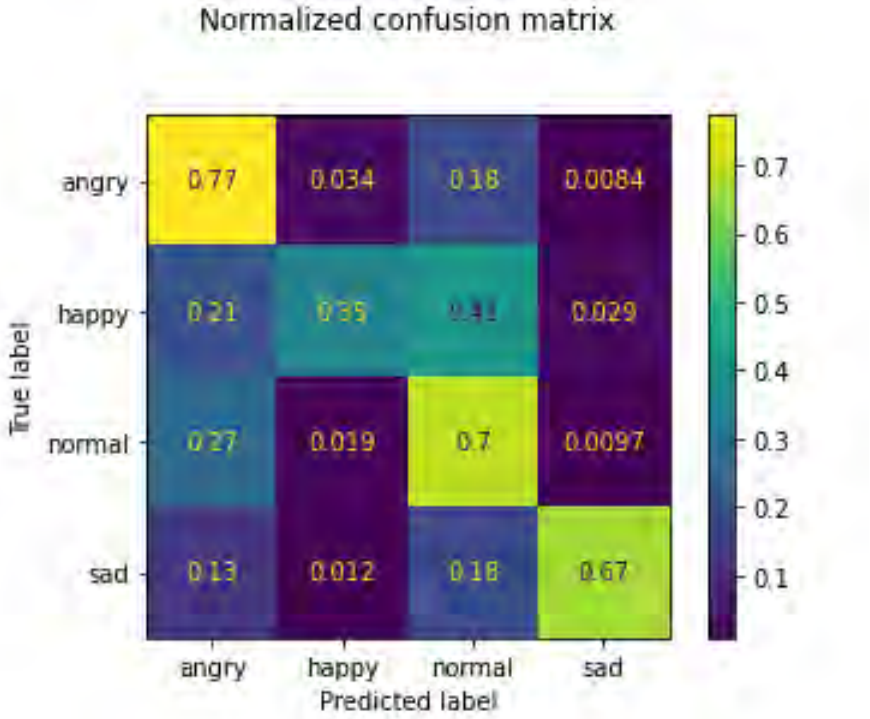


Figure 5.7: Normalized Confusion Matrix of Random Forest Classifier.

	precision	recall	f1-score	support
0	0.64	0.83	0.72	111
1	0.68	0.36	0.47	76
2	0.62	0.69	0.65	115
3	0.95	0.81	0.88	70
accuracy			0.69	372
macro avg	0.72	0.67	0.68	372
weighted avg	0.70	0.69	0.68	372

Figure 5.8: Classification Report of Random Forest Classifier.

5.6 SVM Classifier

SVM Classifier was another algorithm which gave us better performance. The accuracy that we obtained from this classifier was 61%. We chose this algorithm for its better performance when there was clear margin of separation between classes. In addition, the classifier was memory efficient in our case. On the other hand, Support Vector Machine took a long time for training.

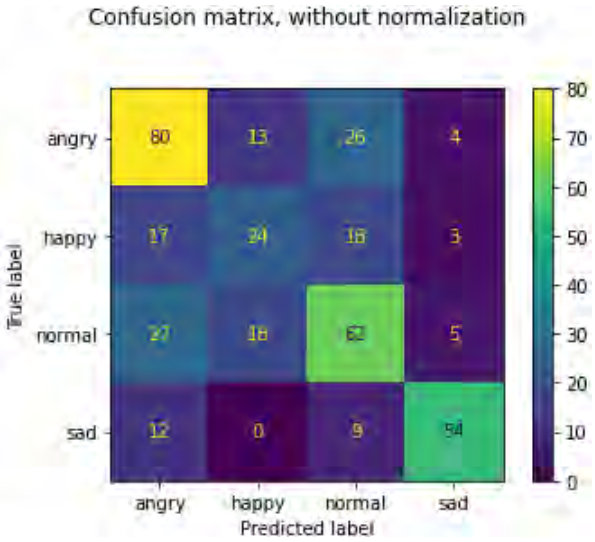


Figure 5.9: Confusion Matrix of SVM Classifier.

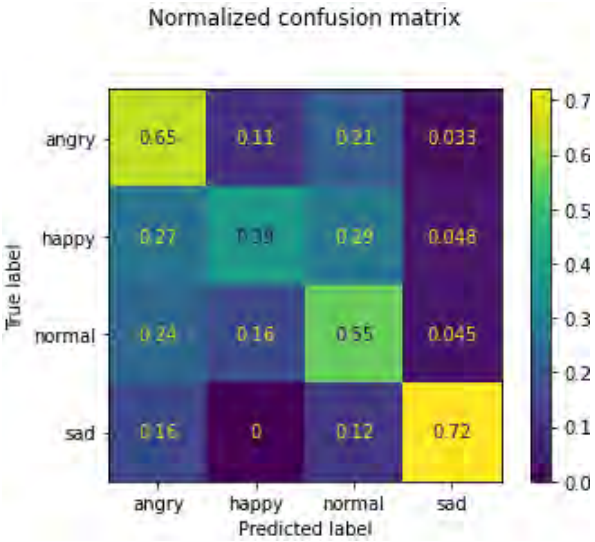


Figure 5.10: Normalized Confusion Matrix of SVM Classifier.

Figure 5.9, 5.10 and figure 5.11 show the confusion matrix, normalized confusion matrix and classification report of SVM Classifier. SVM classifier was another efficient algorithm in image classification for our research.

	precision	recall	f1-score	support
0	0.56	0.64	0.60	112
1	0.63	0.56	0.59	142
2	0.53	0.55	0.54	110
3	0.78	0.75	0.76	71
accuracy			0.61	435
macro avg	0.63	0.62	0.62	435
weighted avg	0.61	0.61	0.61	435

Figure 5.11: Classification Report of SVM Classifier.

5.7 Logistic Regression Analysis

Logistic Regression Analysis resulted in 56% accuracy. The algorithm was easy to implement. We trained the model efficiently. However, the model led to overfitting sometimes. Thus, it gave us a moderate result. We imported GridSearchCV from sklearn library to find the best hypermeters. Figure 5.12 shows the confusion matrix of Logistic Regression Analysis.

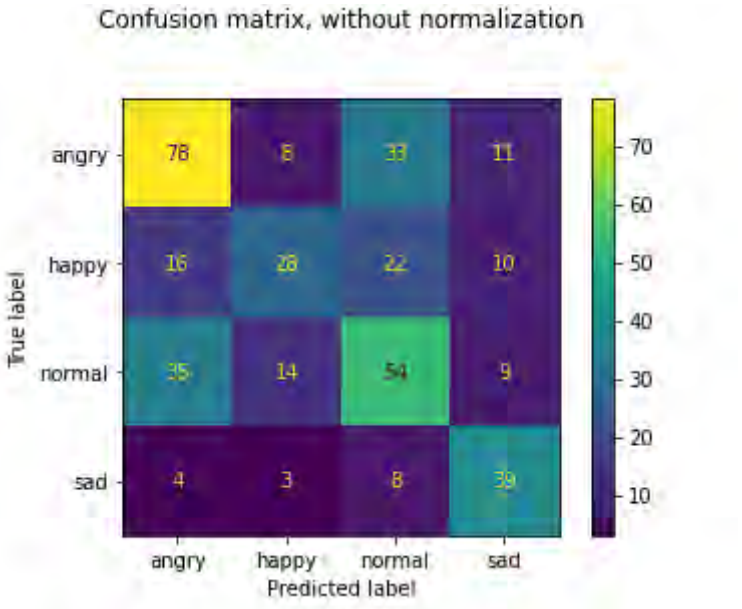


Figure 5.12: Confusion Matrix of Logistic Regression Analysis.

Logistic Regression Analysis took a lot of time to run in our research. We kept the learning rate 0.25 for better result to create the model. There were two separate methods for backpropagation and forward propagation. We also used multiple methods for specific task while implementing this algorithm. For example, there was a method for calculating sigmoid function and another one was for mean squared loss. Normalized confusion matrix for the algorithm is shown in figure 5.13.

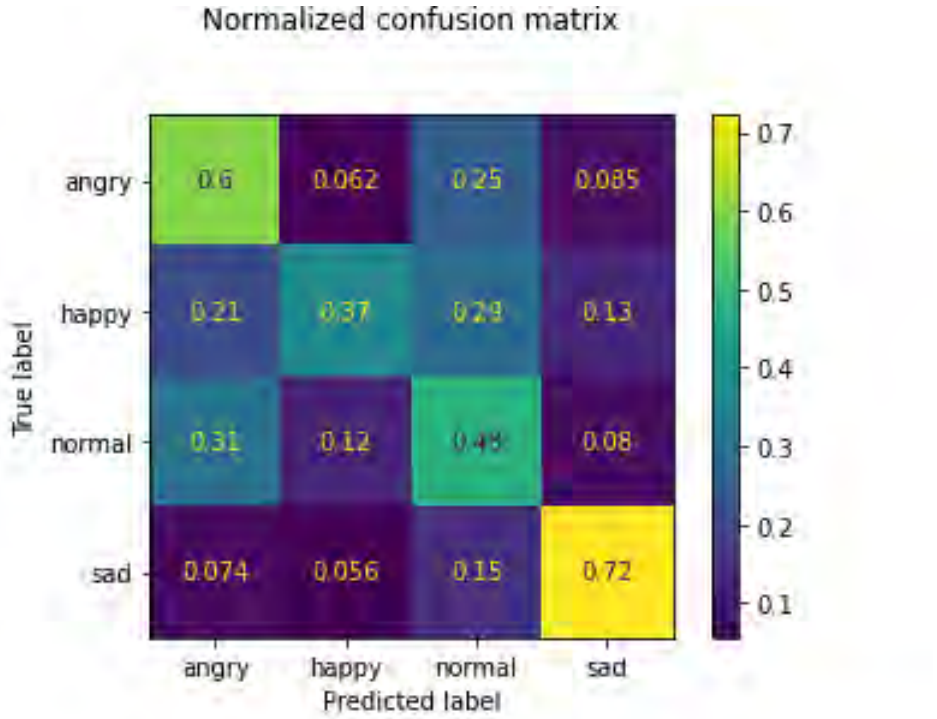


Figure 5.13: Normalized Confusion Matrix of Logistic Regression Analysis.

5.8 K-Nearest Neighbors Classifier

We achieved the highest performance from KNN classifier. The algorithm gave us 72% accuracy. The implementation of this classifier was very simple. The algorithm robusted to noisy training data. We kept the K value 1 as we can observe from the figure 5.14 that when K value is 1 the mean error is zero.

As the mean error was zero, the algorithm gave us highest accuracy. The execution time of this algorithm was lesser than other algorithms. During our research we observed that, KNN algorithm stored the training dataset. While making real time predictions, the classifier learnt from it. That is why, KNN performed as a faster algorithm comparing other algorithms. Figure 5.15, 5.16 and figure 5.17 refer to the confusion matrix, normalized confusion matrix and classification report of KNN Classifier.

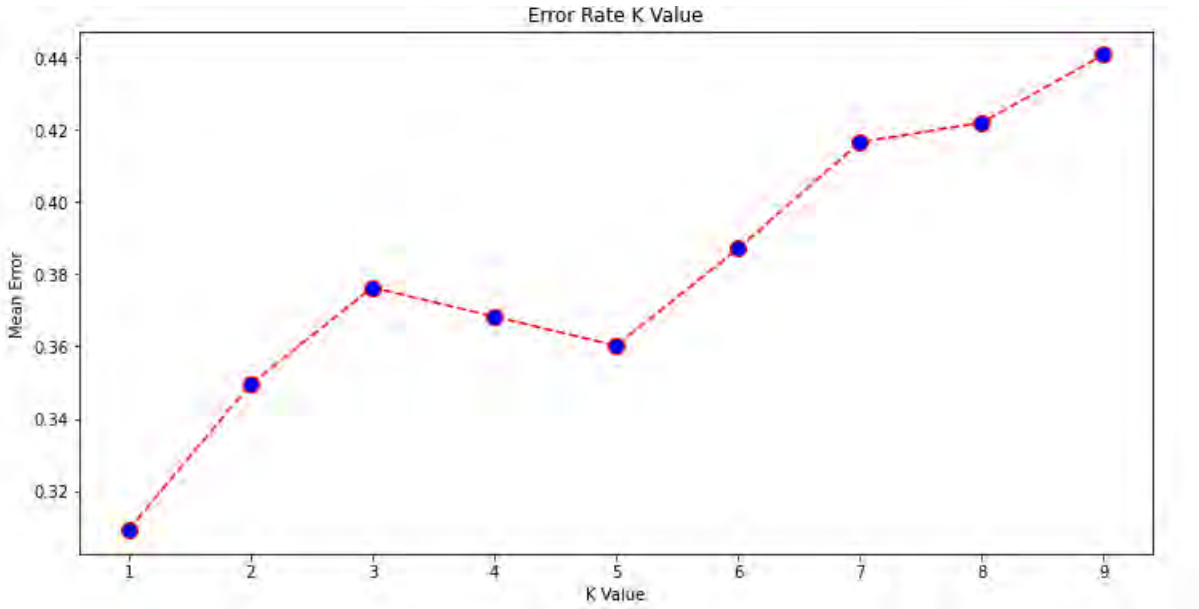


Figure 5.14: Graph of Error rate for K Value.

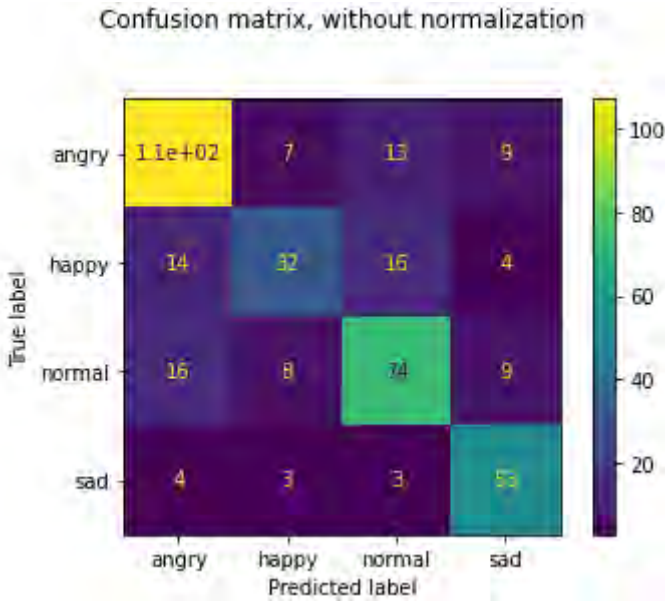


Figure 5.15: Confusion Matrix of KNN Classifier.

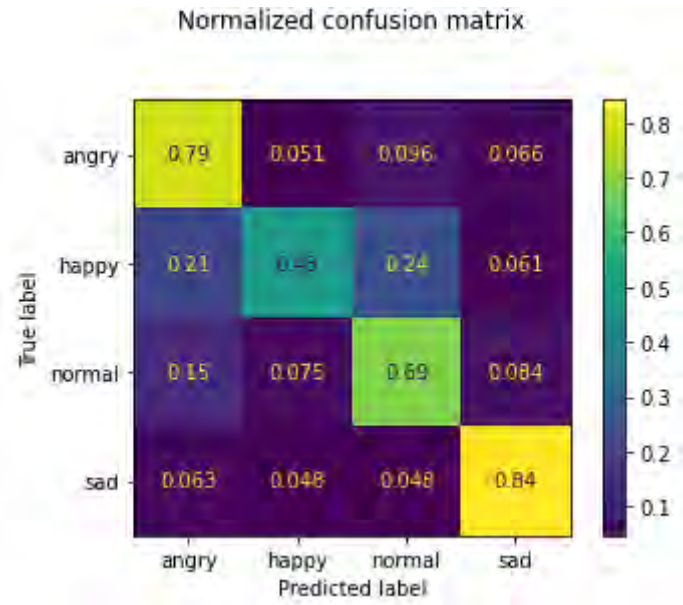


Figure 5.16: Normalized Confusion Matrix of KNN Classifier.

	precision	recall	f1-score	support
0	0.76	0.79	0.77	136
1	0.64	0.48	0.55	66
2	0.70	0.69	0.69	107
3	0.71	0.84	0.77	63
accuracy			0.72	372
macro avg	0.70	0.70	0.70	372
weighted avg	0.71	0.72	0.71	372

Figure 5.17: Classification Report of KNN Classifier.

5.9 Linear Discriminant Analysis

The accuracy that we gained from Linear Discriminant Analysis was 52%. The algorithm minimized overfitting in our case. LDA made strong speculation about the distribution of each input variable. Figure 5.18, 5.19 and figure 5.20 show the confusion matrix, normalized confusion matrix and classification report of Linear Discriminant Analysis.

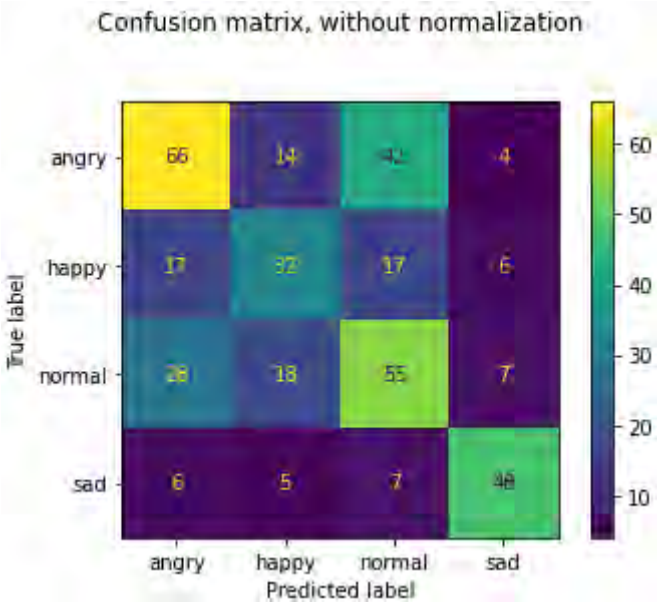


Figure 5.18: Confusion Matrix of Linear Discriminant Analysis.

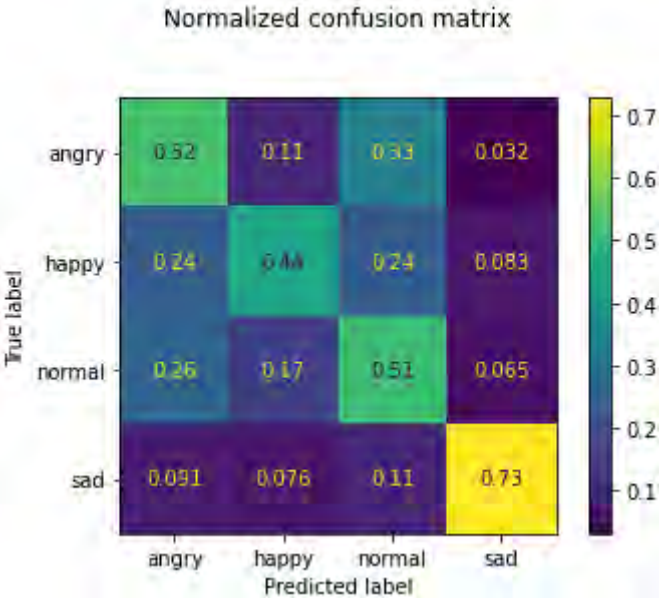


Figure 5.19: Normalized Confusion Matrix of Linear Discriminant Analysis.

	precision	recall	f1-score	support
0	0.47	0.65	0.54	105
1	0.54	0.41	0.46	81
2	0.49	0.42	0.45	113
3	0.69	0.63	0.66	73
accuracy			0.52	372
macro avg	0.55	0.53	0.53	372
weighted avg	0.53	0.52	0.52	372

Figure 5.20: Classification Report of Linear Discriminant Analysis.

5.10 Decision Tree Classifier

The lowest accuracy that we got was 46% from Decision Tree Classifier. Decision Tree Classifier was insufficient in case of regression. It had more complexity than other algorithms. We used 'entropy' as criterion parameter for information gain method to measure the quality of split. The algorithm has high probability of overfitting that might result in lower accuracy in our research. Besides, it might cause sampling errors. So, these are the reasons behind poor performance of Decision Tree Classifier in our research.

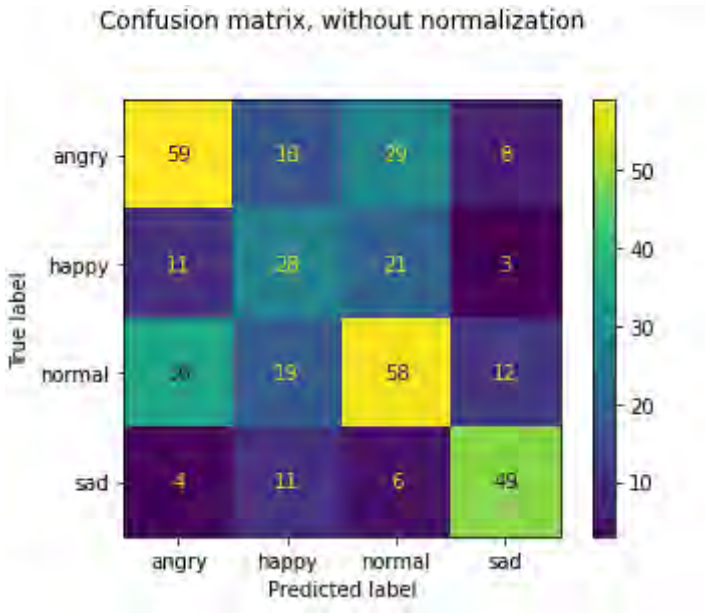


Figure 5.21: Confusion Matrix of Decision Tree Classifier.

Here, we can examine confusion matrix, normalized confusion matrix and classification report of Decision Tree Classifier in figure 5.21, 5.22 and figure 5.23 accordingly.

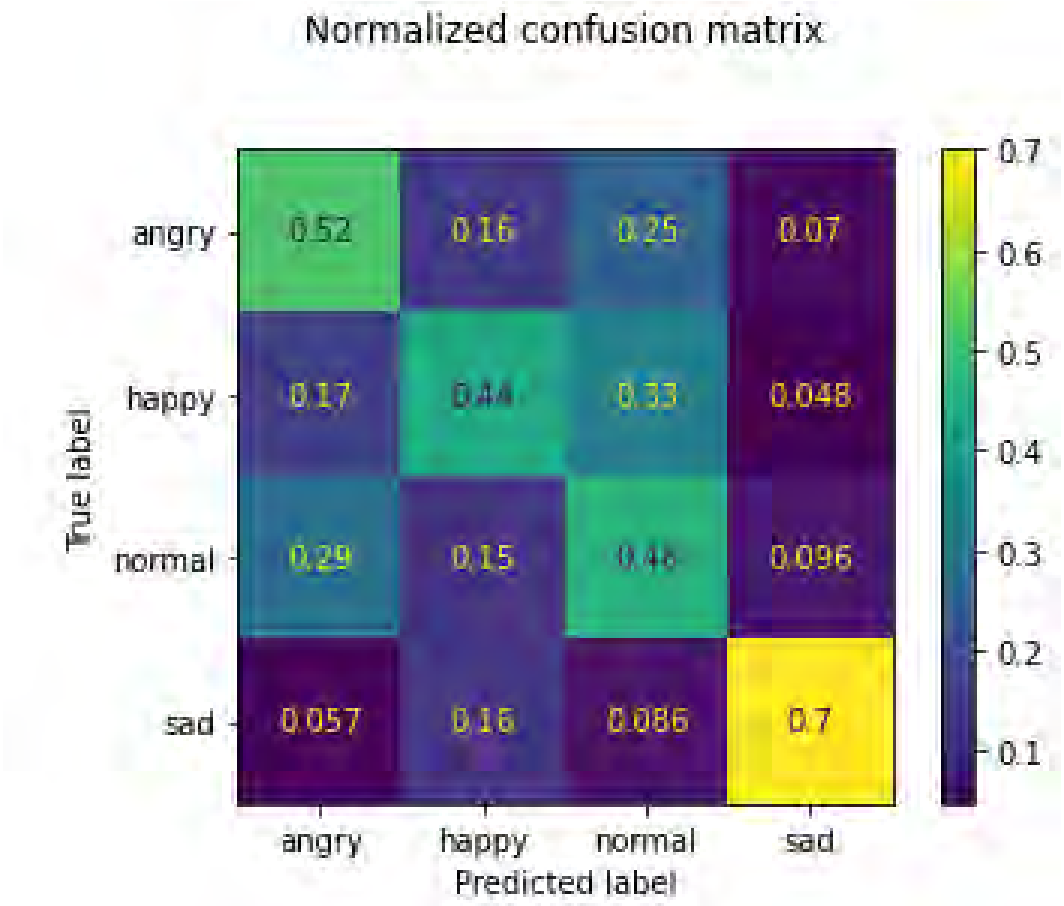


Figure 5.22: Normalized Confusion Matrix of Decision Tree Classifier.

	precision	recall	f1-score	support
0	0.45	0.45	0.45	132
1	0.52	0.40	0.45	139
2	0.38	0.42	0.40	111
3	0.55	0.77	0.64	53
accuracy			0.46	435
macro avg	0.47	0.51	0.48	435
weighted avg	0.47	0.46	0.46	435

Figure 5.23: Classification Report of Decision Tree Classifier.

5.11 Gaussian Naïve Bayes Classifier

Decision Tree Classifier and Gaussian Naïve Bayes Classifier resulted in the same accuracy of 46%. Gaussian Na¨ıve Bayes Classifier had lack of interactions among features [3]. The algorithm has zero frequency problem. It considers all attributes as independent. These deficiencies might cause us lower accuracy. Figure 5.24, figure 5.25 and figure 5.26 show the confusion matrix, normalized confusion matrix and classification report of Gaussian Naïve Bayes Classifier.

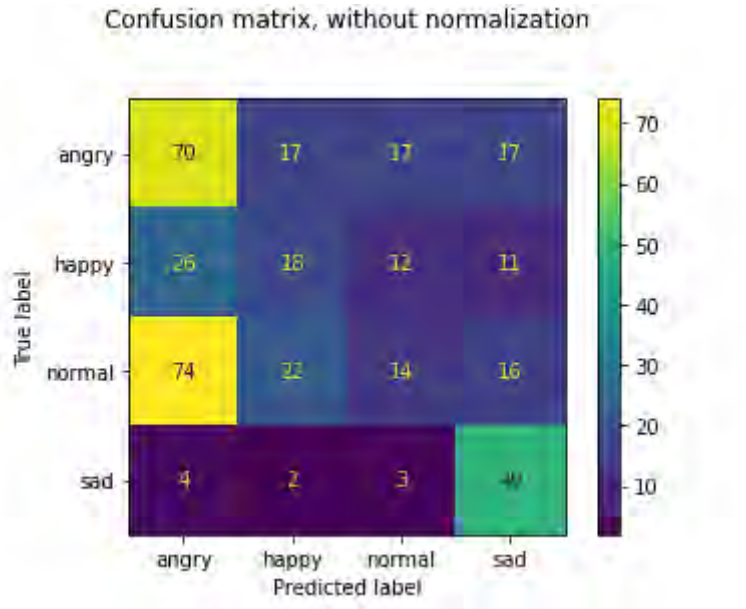


Figure 5.24: Confusion Matrix of Gaussian Naïve Bayes Classifier.

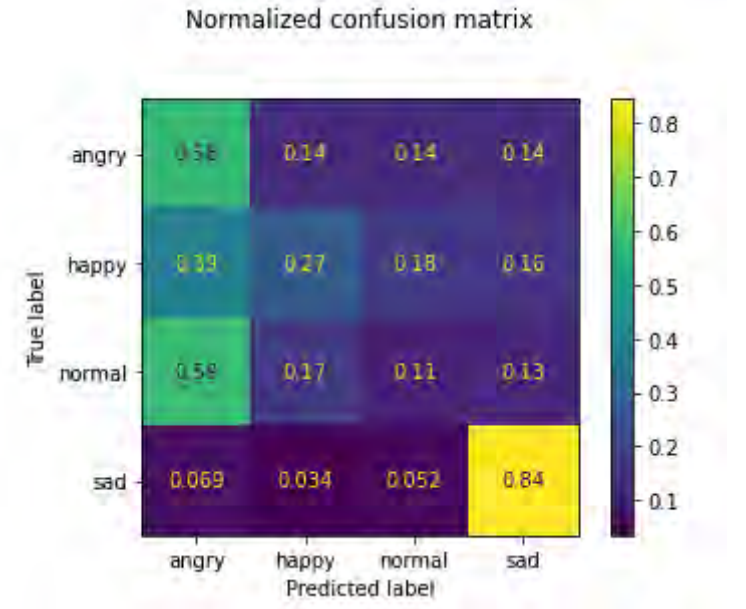


Figure 5.25: Normalized Confusion Matrix of Gaussian Naïve Bayes Classifier.

	precision	recall	f1-score	support
0	0.43	0.57	0.49	127
1	0.47	0.41	0.44	125
2	0.36	0.18	0.24	116
3	0.55	0.81	0.65	67
accuracy			0.46	435
macro avg	0.45	0.49	0.45	435
weighted avg	0.44	0.46	0.43	435

Figure 5.26: Classification Report of Gaussian Naïve Bayes Classifier.

5.12 Algorithms without Feature Extraction

After running all of the seven algorithms separately, we got the highest accuracy from KNN Classifier which was 72%. In contrast, the lowest accuracy that we got was 46% Gaussian Naive Bayes and Decision Tree Classifier.

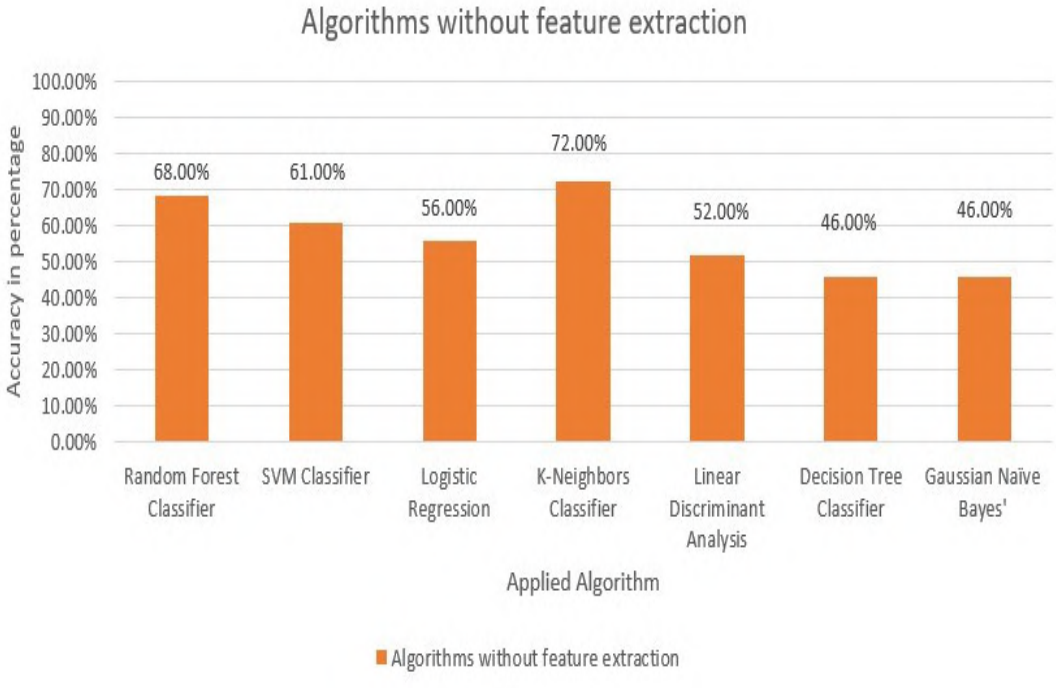


Figure 5.27: Algorithms without Feature Extraction.

5.13 Implementing Random Forest Classifier along with PCA

The below figure 5.28 , figure 5.29 and figure 5.30 show the confusion matrix and normalized confusion matrix of PCA with Random Forest accordingly. PCA decreased the accuracy of the algorithm from 68% to 61%.

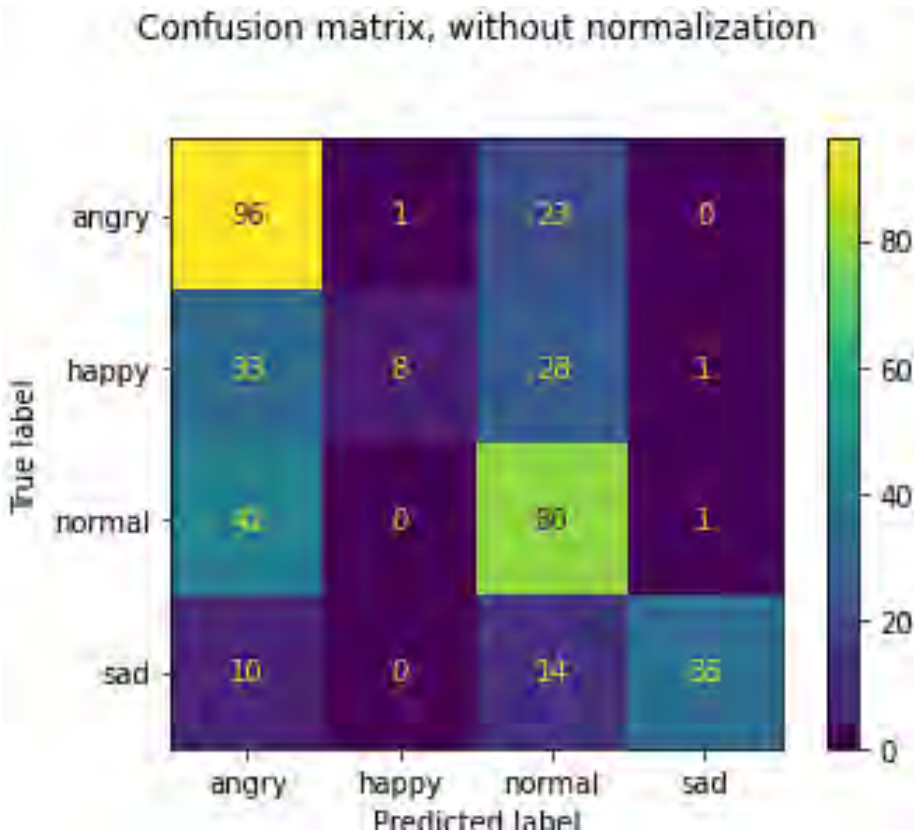


Figure 5.28: Confusion Matrix of Random Forest Classifier with PCA.

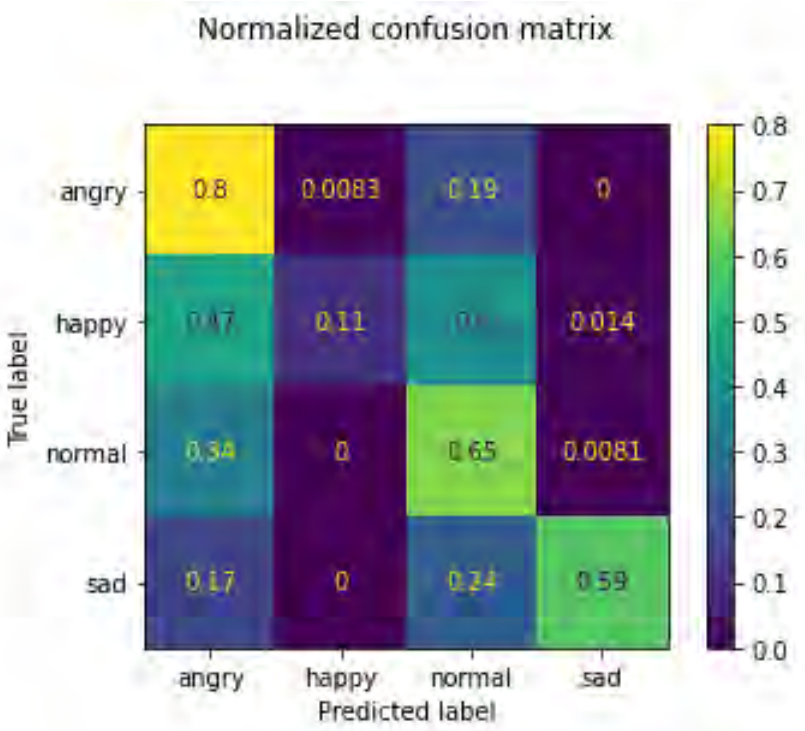


Figure 5.29: Normalized Confusion Matrix of Random Forest Classifier with PCA.

	precision	recall	f1-score	support
0	0.59	0.59	0.59	128
1	0.53	0.68	0.60	128
2	0.57	0.49	0.53	113
3	1.00	0.70	0.82	66
accuracy			0.61	435
macro avg	0.67	0.61	0.63	435
weighted avg	0.63	0.61	0.61	435

Figure 5.30: Classification Report of Random Forest Classifier with PCA.

5.14 Implementing SVM Classifier along with PCA

Our implemented SVM classifier had the accuracy of 61% whereas SVM along with PCA gave us the improved accuracy of 67%. The confusion matrix, normalized confusion matrix and classification report of SVM with PCA feature extraction are given below in figure 5.31, figure 5.32 and figure 5.33 accordingly.

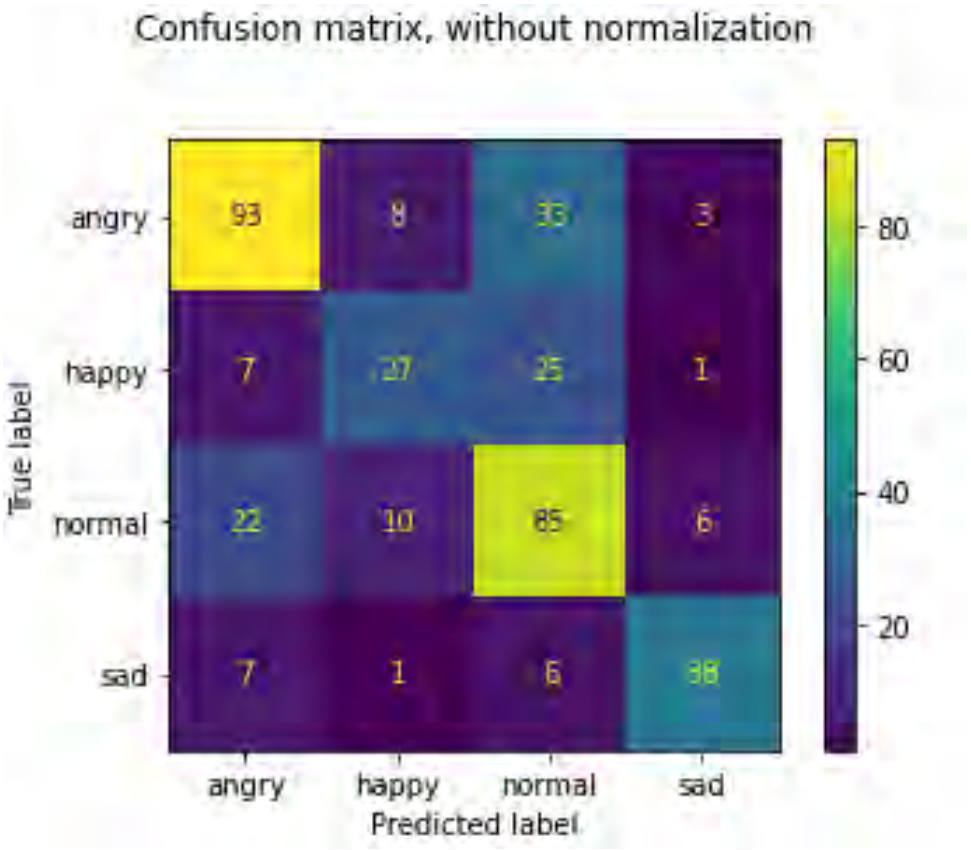


Figure 5.31: Confusion Matrix of Support Vector Machine Classifier with PCA.

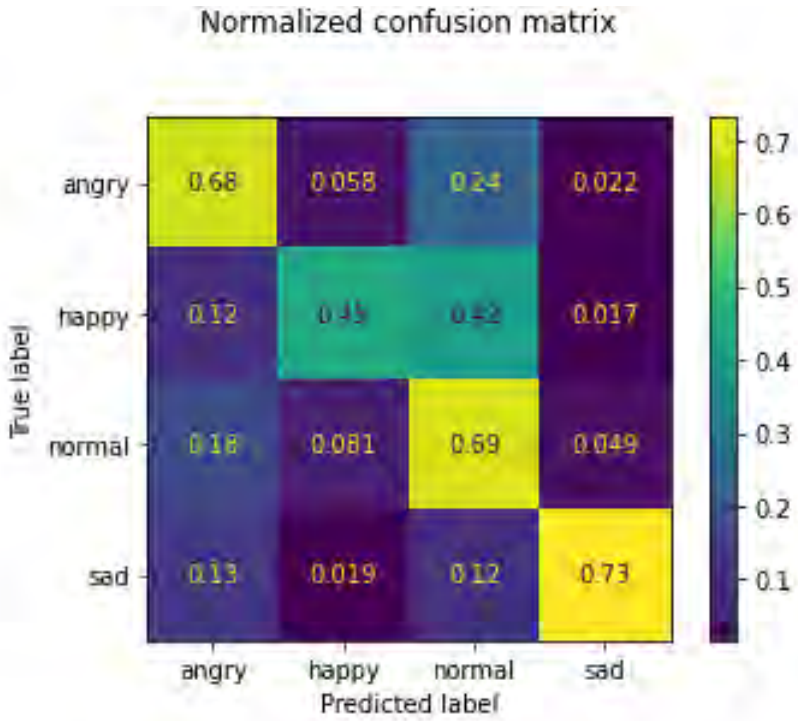


Figure 5.32: Normalized Confusion Matrix of Support Vector Machine Classifier with PCA.

	precision	recall	f1-score	support
0	0.66	0.71	0.68	129
1	0.62	0.60	0.61	121
2	0.62	0.62	0.62	123
3	0.88	0.82	0.85	62
accuracy			0.67	435
macro avg	0.69	0.69	0.69	435
weighted avg	0.67	0.67	0.67	435

Figure 5.33: Classification Report of Support Vector Machine Classifier with PCA.

5.15 Implementing Logistic Regression along with PCA

Logistic Regression alone gave us the accuracy of 56.00% and along with PCA, it had an accuracy of 53%. Figure **??** shows the PCA explained variance ratio that we implemented for logistic regression.

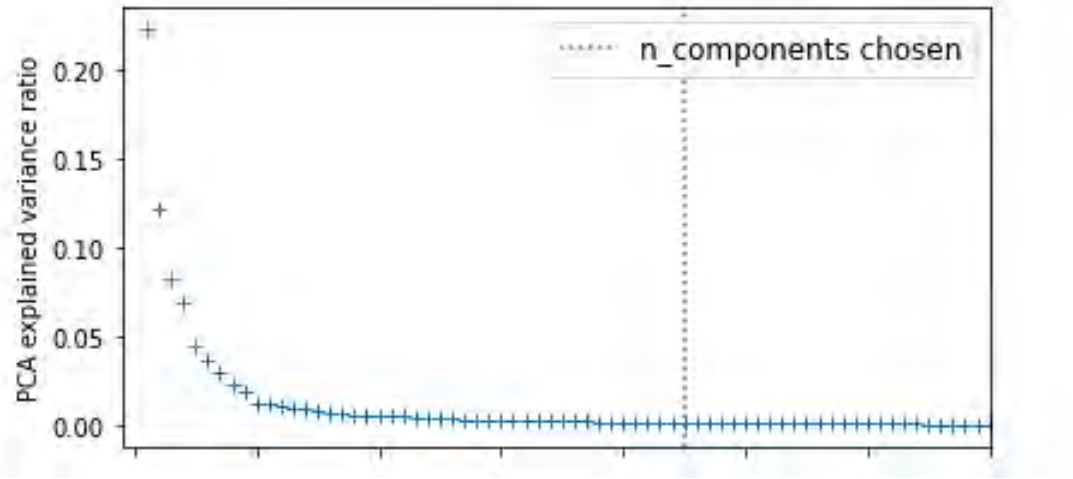


Figure 5.34: PCA explained variance ratio for Logistic Regression.

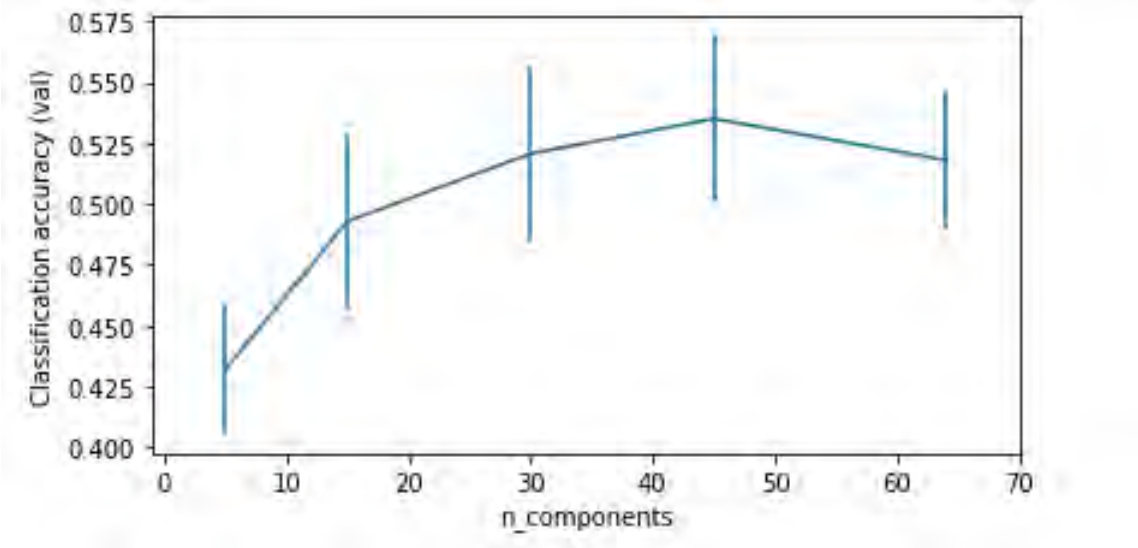


Figure 5.35: Classification Accuracy of Logistic Regression for increasing n_components of PCA.

From figure 5.35, we can observe that accuracy is highest when n_components of PCA is 45. That is why, we kept the value of n_components 45 while implementing PCA for Logistic Regression. Figure 5.36, figure 5.37 and figure 5.38 show the confusion matrix, normalized confusion matrix and classification report of Logistic Regression with PCA.

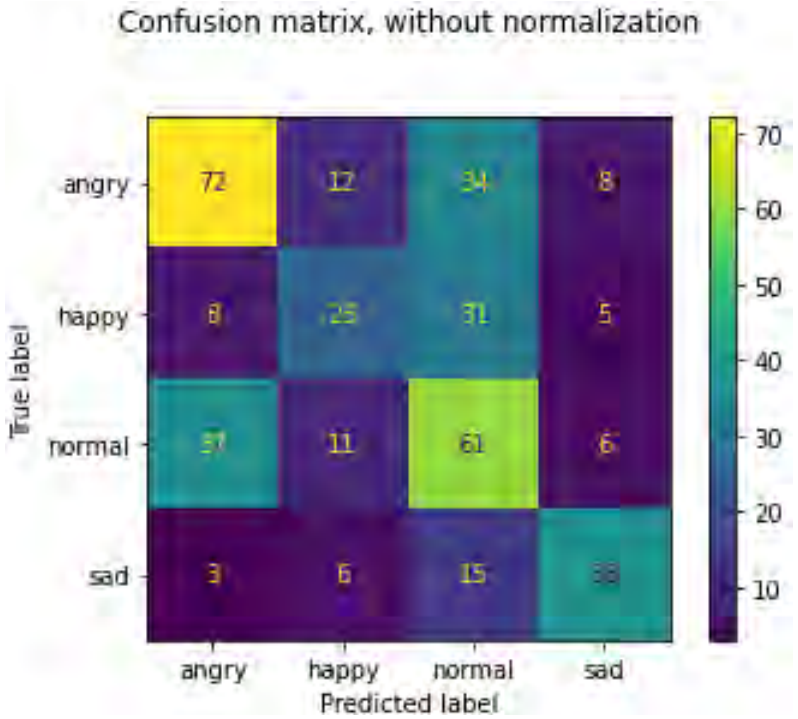


Figure 5.36: Confusion Matrix of Logistic Regression with PCA.

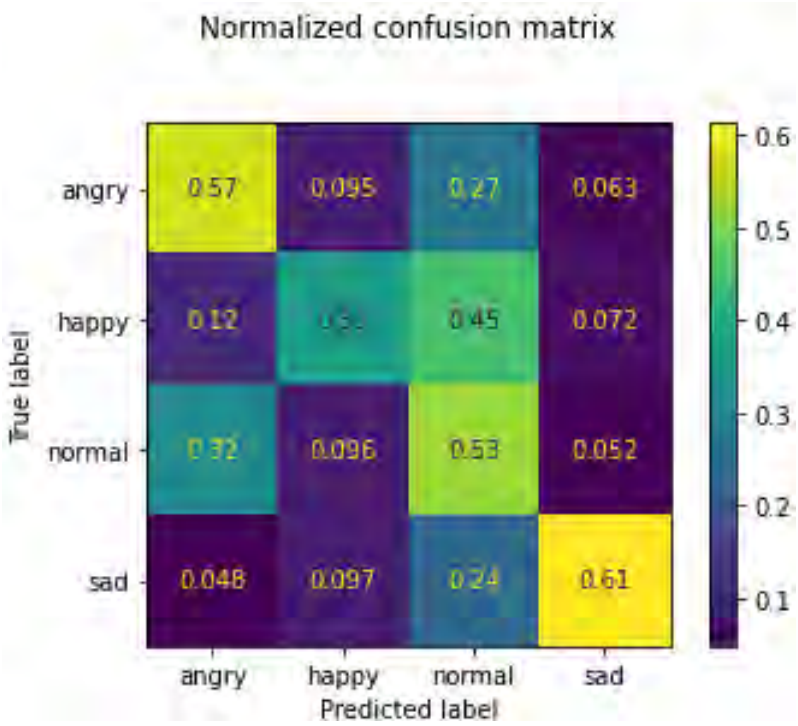


Figure 5.37: Normalized Confusion Matrix of Logistic Regression with PCA.

	precision	recall	f1-score	support
0	0.60	0.57	0.59	126
1	0.46	0.36	0.41	69
2	0.43	0.53	0.48	115
3	0.67	0.61	0.64	62
accuracy			0.53	372
macro avg	0.54	0.52	0.53	372
weighted avg	0.53	0.53	0.53	372

Figure 5.38: Classification Report of Logistic Regression with PCA.

5.16 Implementing K-Neighbors Classifier along with PCA

The accuracy of 72% was given by K-Neighbors Classifier whereas with PCA the accuracy went down to 55%. Figure 5.39, figure 5.40 and figure 5.41 refer to the confusion matrix, normalized confusion matrix and classification report of KNN along with PCA.

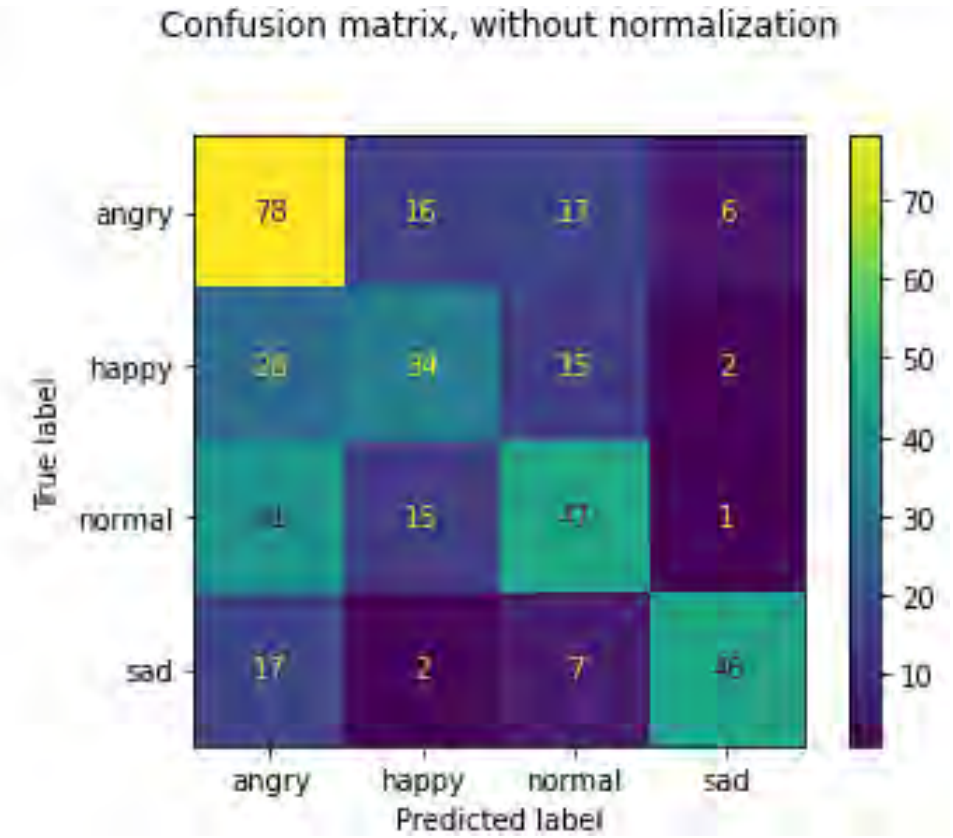


Figure 5.39: Confusion Matrix of KNN with PCA.

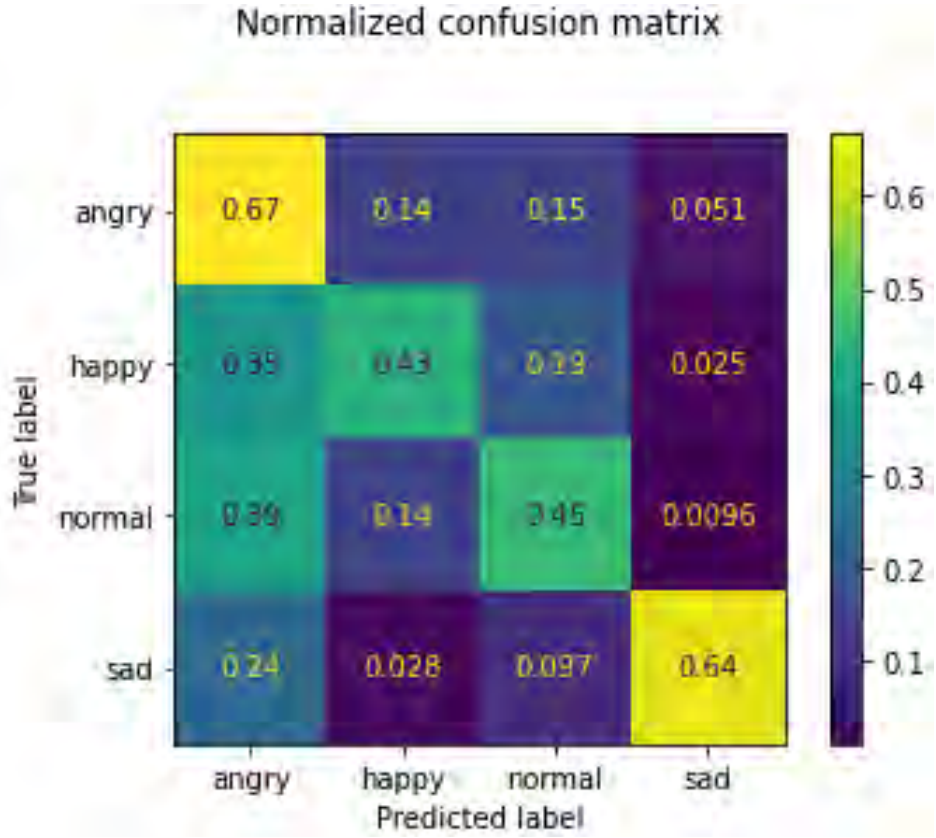


Figure 5.40: Normalized Confusion Matrix of KNN with PCA.

	precision	recall	f1-score	support
0	0.48	0.67	0.56	117
1	0.51	0.43	0.47	79
2	0.55	0.45	0.49	104
3	0.84	0.64	0.72	72
accuracy			0.55	372
macro avg	0.59	0.55	0.56	372
weighted avg	0.57	0.55	0.55	372

Figure 5.41: Classification Report of KNN with PCA.

5.17 Implementing Linear Discriminant Analysis along with PCA

Linear Discriminant Analysis, we got the lower accuracy of 52%. In contrast, the highest accuracy was 54% along with PCA. The confusion matrix, normalized confusion matrix and classification report of Linear Discriminant Analysis is shown in figure 5.42, figure 5.43 and figure 5.44 accordingly.

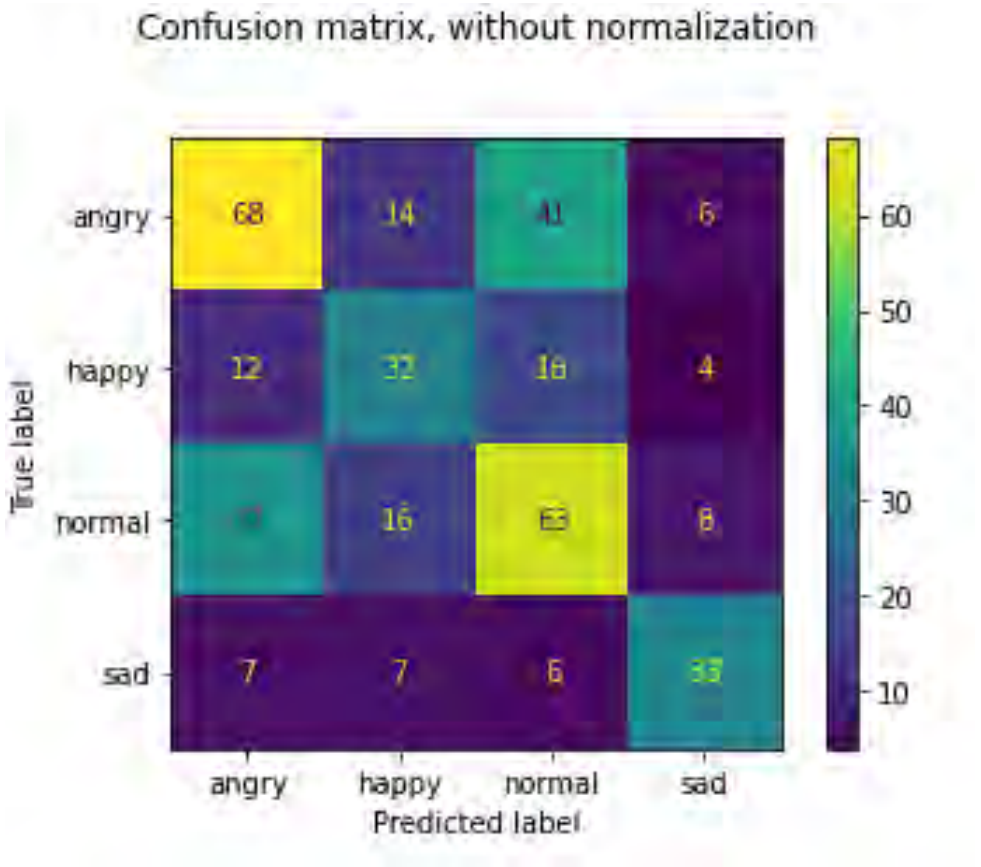


Figure 5.42: Confusion Matrix of Linear Discriminant Analysis with PCA.

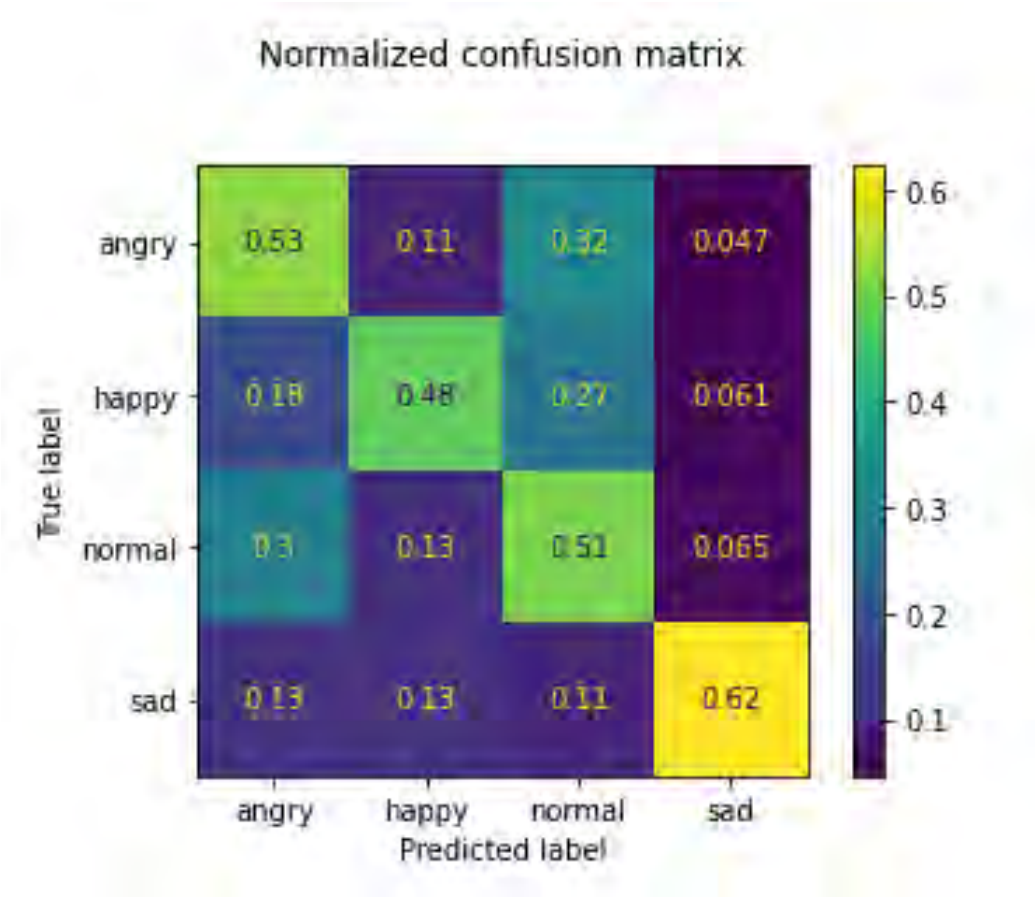


Figure 5.43: Normalized Confusion Matrix of Linear Discriminant Analysis with PCA.

	precision	recall	f1-score	support
0	0.53	0.61	0.56	117
1	0.47	0.39	0.43	66
2	0.50	0.48	0.49	122
3	0.68	0.66	0.67	67
accuracy			0.54	372
macro avg	0.54	0.54	0.54	372
weighted avg	0.54	0.54	0.54	372

Figure 5.44: Classification Report of Linear Discriminant Analysis with PCA.

5.18 Implementing Decision Tree Classifier along with PCA

While implementing Decision Tree Classifier, the accuracy we obtained was 46% and with PCA it was 50%. We plotted the confusion matrix, normalized confusion matrix and classification report of the algorithm in figure 5.45, figure 5.46 and figure 5.47.

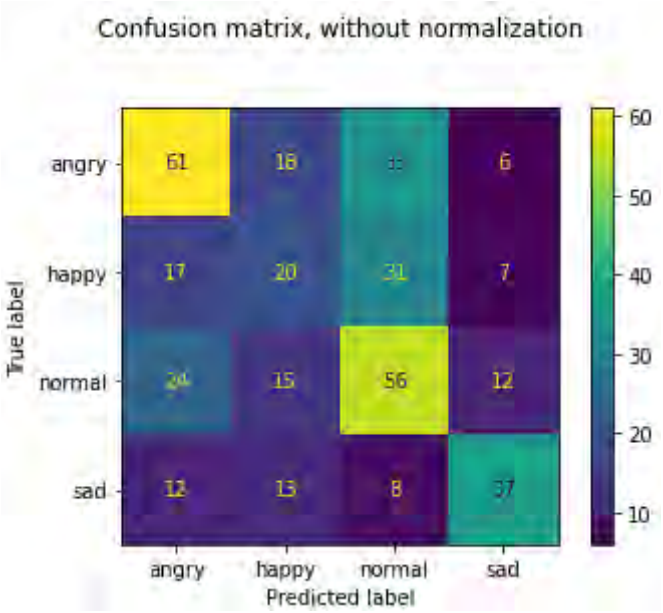


Figure 5.45: Confusion Matrix of Decision Tree Classifier with PCA.

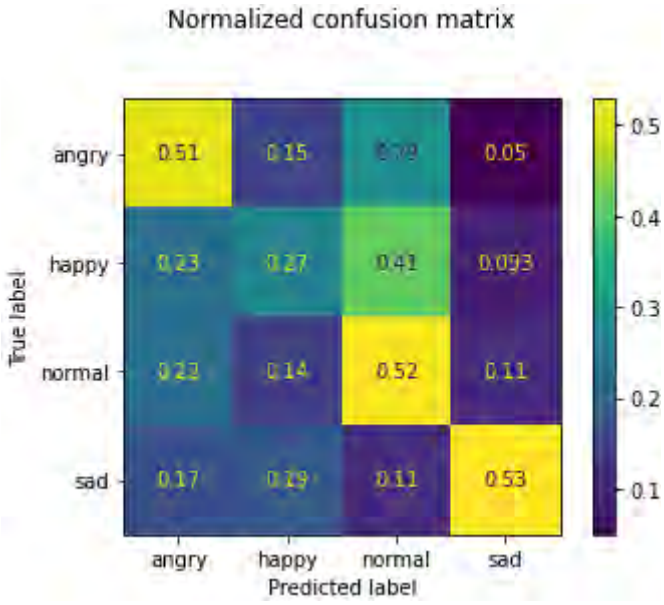


Figure 5.46: Normalized Confusion Matrix of Decision Tree Classifier with PCA.

	precision	recall	f1-score	support
0	0.54	0.46	0.49	127
1	0.48	0.45	0.46	134
2	0.39	0.45	0.42	101
3	0.62	0.74	0.68	73
accuracy			0.50	435
macro avg	0.51	0.52	0.51	435
weighted avg	0.50	0.50	0.50	435

Figure 5.47: Classification Report of Decision Tree Classifier with PCA.

5.19 Implementing Gaussian NB (Naïve Bayes) along with PCA

Gaussian NB gave us the accuracy of 46%; with PCA the accuracy changed into 38%. This was the lowest accuracy comparing all the algorithms that we implemented. Here, we can observe the confusion matrix, normalized confusion matrix and classification report of the algorithm in figure 5.48, figure 5.49 and figure 5.50.

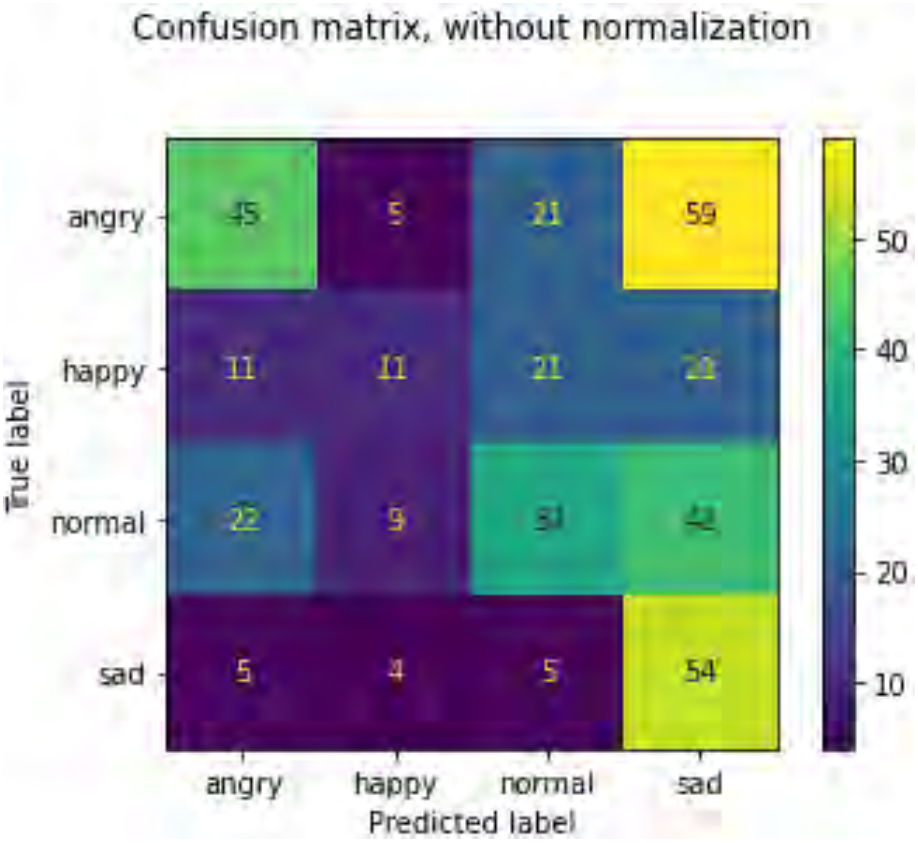


Figure 5.48: Confusion Matrix of Gaussian Naïve Bayes Classifier with PCA.

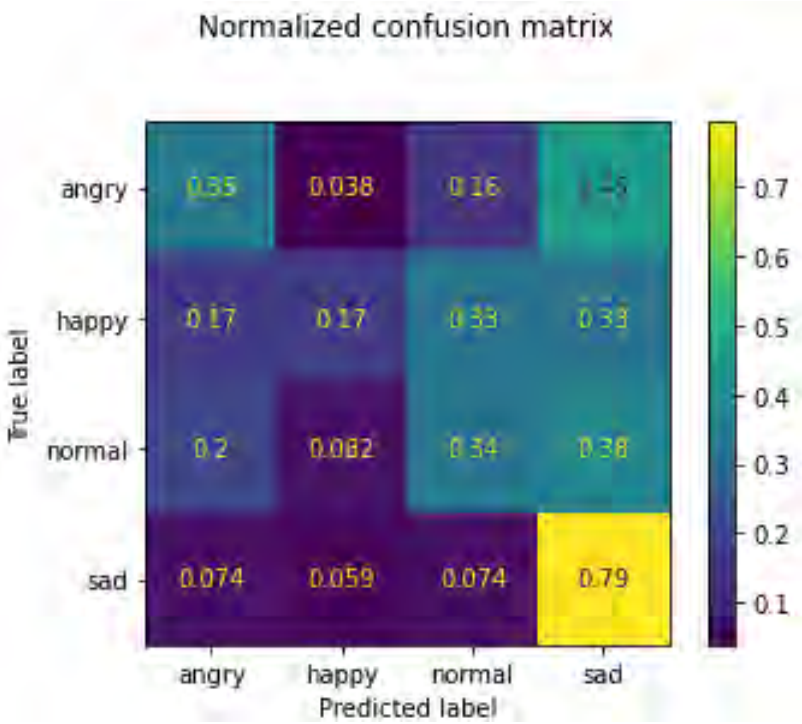


Figure 5.49: Normalized Confusion Matrix of Gaussian Naïve Bayes Classifier with PCA.

	precision	recall	f1-score	support
0	0.40	0.46	0.43	127
1	0.46	0.26	0.34	125
2	0.46	0.23	0.31	126
3	0.29	0.79	0.43	57
accuracy			0.38	435
macro avg	0.40	0.44	0.38	435
weighted avg	0.42	0.38	0.37	435

Figure 5.50: Classification Report of Gaussian Naïve Bayes with PCA.

5.20 Principal Component Analysis (PCA) with all seven algorithms

Now that, we implemented all of the seven algorithms and classifiers along with PCA, we may notice that in the figure 5.51, we got the best accuracy from SVM which was 67% and the lowest was 38% from Gaussian Naïve Bayes.

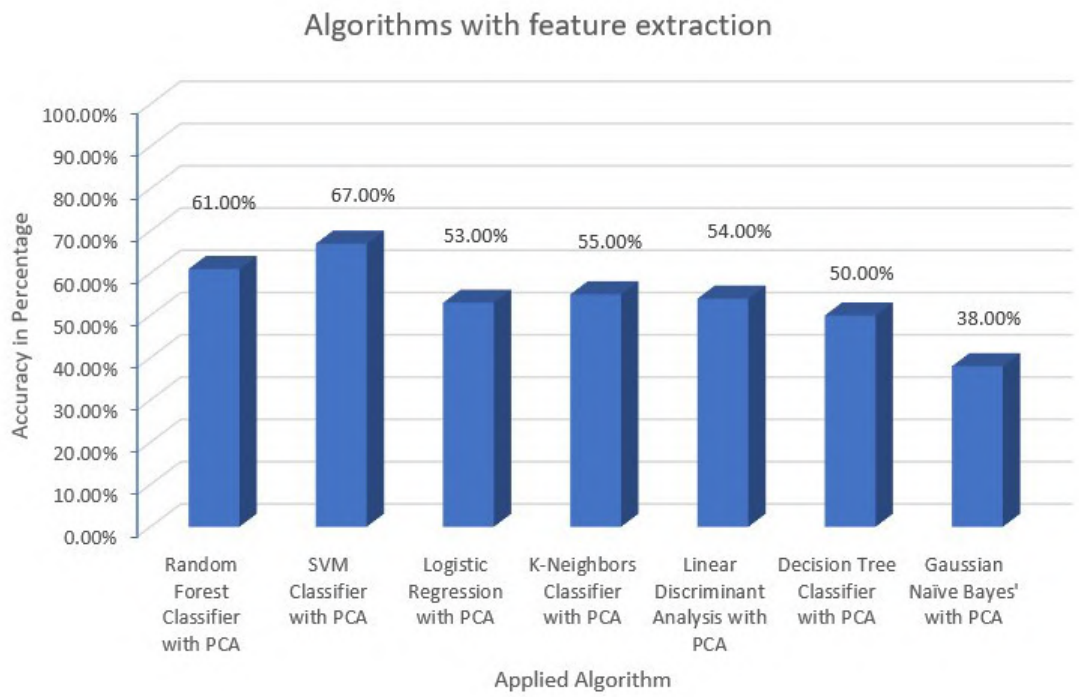


Figure 5.51: PCA Feature Extraction along with all seven algorithms.

PCA reduces the size of feature vectors which leads to computational efficiency. The reason behind improved accuracy was that it reduced the complexity of the model. This might help to reduce overfitting. While experimenting with the algorithms, we found that it removed the least beneficial features. PCA did not examine the labels of the training data. PCA considered features important when they had large variance. But higher variance did not contribute to prediction target in some cases. In this way, PCA removed important features and created unnecessary features in the algorithms in which we got lower accuracy

5.21 Overall Performance Analysis Comparison

Finally, we came to a conclusion that, we got the highest accuracy from K-Nearest Neighbors Classifier which was 72%. On the other hand, the lowest accuracy we got was from Gaussian Naïve Bayes along with PCA which was 38% after comparing all the algorithms. From figure 5.52, we can view the improved and decreased accuracy before and after implementing the mentioned algorithms with PCA.
While implementing the algorithms, we found that, if we increase the dataset, accuracy decreases. This happened because autistic children do not have differentiable and unique facial expression. For this reason, the number of similar facial expressions increased which might lead to wrong classification. So, the dataset became complex and it caused lower accuracy. In the end, we get the result as shown in the table 5.1. Our goal was to predict emotion using machine learning algorithms and neural networks with an improved accuracy. Finally, we can say that, we have reached our goal successfully from the above discussion.

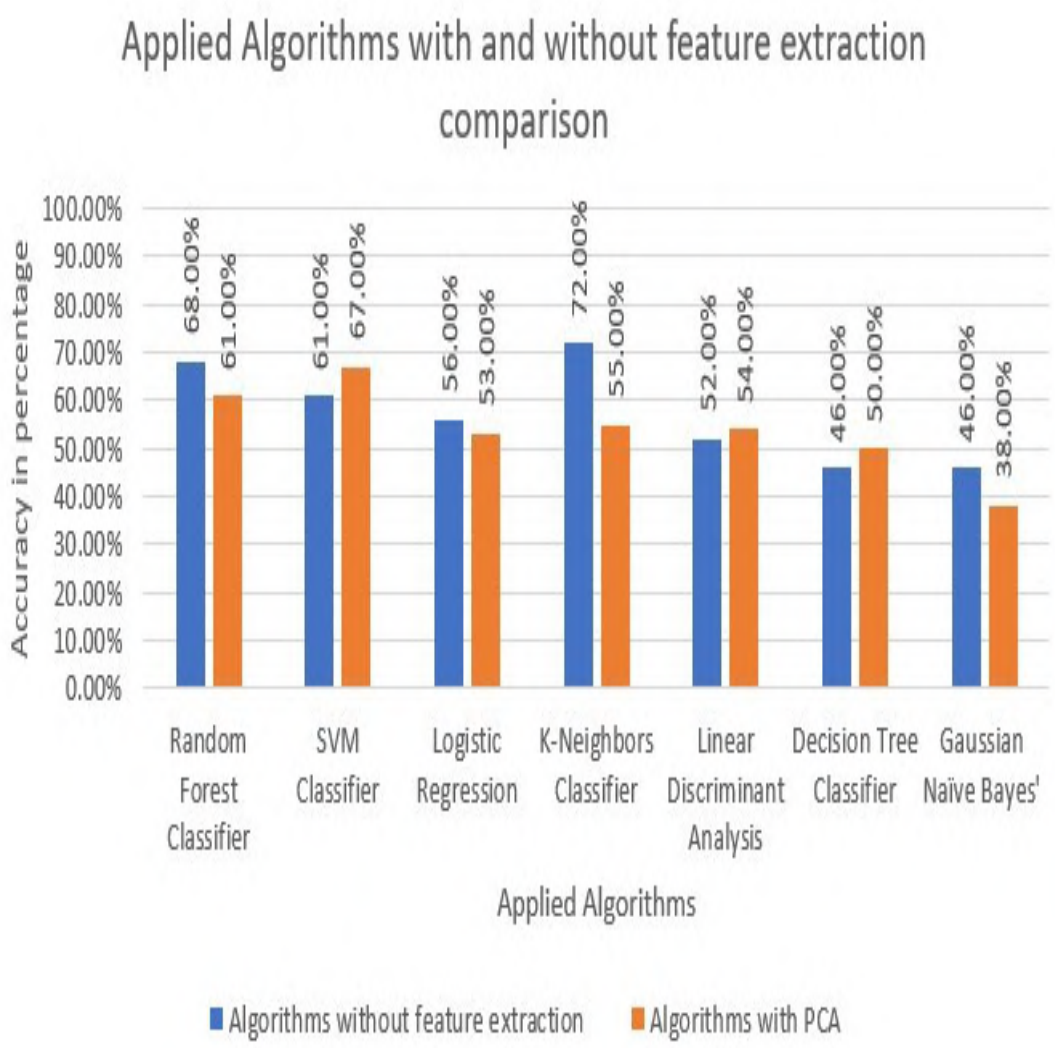


Figure 5.52: Comparisons of all algorithms along with PCA.

Feature Extraction Techniques	Algorithms without Feature Extraction Techniques	Algorithms with PCA
Random Forest Classifier	68%	61%
SVM Classifier	61%	67%
Logistic Regression	56%	53%
K-Nearest Neighbors Classifier	72%	55%
Linear Discriminant Analysis	52%	54%
Decision Tree Classifier	46%	50%
Gaussian Naïve Bayes	46%	38%

Table 5.1: Computed Final Scores in Percentage

Chapter 6

Conclusion

Facial expression recognition using machine learning nowadays is getting more popular as more and more works are being done in this field. Autistic children are different than the normal children as they have special needs and their way of communication with others is different. As we are working with autistic children in our project, we are one of The pioneers in this field. Being the pioneers we had to face different complications throughout our journey. Also, we enjoyed the challenges and got new experiences. As mentioned earlier, autistic children express themselves differently and this is where we wanted to work from the very first of our thesis journey. Our vision was to try and find out if there is any pattern in their facial expressions through which we could predict their emotions. As machine learning is all about teaching and predicting via patterns, we tried using that in our case and we came up with best possible results. We tried different algorithms and models to feed in our self-collected dataset and train them to give us results. We used grey-scale images for most of these models. At this point of our work, we think we have come up with a new dimension in the research field of image processing and emotion recognition using facial expression. Another thing, during our thesis work, COVID-19 hit badly worldwide and same goes for our country. For this reason we could not collect more data as we planned, as all the institutions were closed. We have progressed a lot in this field, so, we strongly believe that further works in this field can refine this work of ours and our work will be of great use for that. This will help everyone to predict the autistic children's emotional state and take care of them properly.

Bibliography

[1] W. Zhao, A. Krishnaswamy, R. Chellappa, D. L. Swets, and J. Weng, "Discriminant analysis of principal components for face recognition," in *Face Recognition*, Springer, 1998, pp. 73–85.

[2] J. Yang, H. Yu, and W. Kunz, "An efficient lda algorithm for face recognition," in *Proceedings of the International Conference on Automation, Robotics, and Computer Vision (ICARCV 2000)*, Citeseer, 2000, pp. 34–47.

[3] J. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "Face recognition using lda-based algorithms," *IEEE Transactions on Neural networks*, vol. 14, no. 1, pp. 195–200, 2003.

[4] J. Yang, D. Zhang, A. F. Frangi, and J.-y. Yang, "Two-dimensional pca: A new approach to appearance-based face representation and recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 1, pp. 131–137, 2004.

[5] A. Eleyan and H. Demirel, "Pca and lda based neural networks for human face recognition," *Face Recognition*, 2007. DOI: 10.5772/4833.

[6] Y. V. Lata, C. K. B. Tungathurthi, H. R. M. Rao, A. Govardhan, and L. Reddy, "Facial recognition using eigenfaces by pca," *International Journal of Recent Trends in Engineering*, vol. 1, no. 1, p. 587, 2009.

[7] K. M. Rump, J. L. Giovannelli, N. J. Minshew, and M. S. Strauss, "The development of emotion recognition in individuals with autism," *Child development*, vol. 80, no. 5, pp. 1434–1447, 2009.

[8] L. WANG, X. DING, and C. FANG, "Accurate localization of facial feature points based on random forest classifier," *Journal of Tsinghua University (Science and Technology)*, vol. 4, p. 021, 2009.

[9] F. Ye, Z. Shi, and Z. Shi, "A comparative study of pca, lda and kernel lda for image classification," *2009 International Symposium on Ubiquitous Virtual Reality*, 2009. DOI: 10.1109/isuvr.2009.26.

[10] D. Maturana, D. Mery, and A. Soto, "Face recognition with decision tree-based local binary patterns," in *Asian Conference on Computer Vision*, Springer, 2010, pp. 618–629.

[11] B. O'Connor and K. Roy, "Facial recognition using modified local binary pattern and random forest," *International Journal of Artificial Intelligence & Applications*, vol. 4, no. 6, p. 25, 2013.

[12] M. K. Abd El Meguid and M. D. Levine, "Fully automated recognition of spontaneous facial expressions in videos using random forest classifiers," *IEEE Transactions on Affective Computing*, vol. 5, no. 2, pp. 141–154, 2014.

[13] Y. Lv, Z. Feng, and C. Xu, "Facial expression recognition via deep learning," in *2014 International Conference on Smart Computing*, IEEE, 2014, pp. 303–308.

[14] C. Zhou, L. Wang, Q. Zhang, and X. Wei, "Face recognition based on pca and logistic regression analysis," *Optik*, vol. 125, no. 20, pp. 5916–5919, 2014.

[15] S. Anwar and M. Milanova, "Real time face expression recognition of children with autism," in *Proc. IAEMR*, 2016.

[16] E. J. HARFASH, "Face recognition system using pca, lda, kernel pca and kernel lda," *International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR)*, vol. 6, no. 5, pp. 9–20, 2016.

[17] J. Howard, *Dense vs convolutional vs fully connected layers*, Nov. 2016. [Online]. Available: https://forums.fast.ai/t/dense-vs-convolutional-vs-fully-connected-layers/191/3.

[18] C. Pramerdorfer and M. Kampel, "Facial expression recognition using convolutional neural networks: State of the art," *arXiv preprint arXiv:1612.02903*, 2016.

[19] F. Z. Salmam, A. Madani, and M. Kissi, "Facial expression recognition using decision trees," in *2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV)*, IEEE, 2016, pp. 125–130.

[20] F. Boyles and F. Boyles, *Random forest in python*, Jul. 2017. [Online]. Available: https://www.blopig.com/blog/2017/07/using-random-forests-in-python-with-scikit-learn.

[21] A. Dertat, *Applied deep learning - part 4: Convolutional neural networks*, Nov. 2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

[22] J. Hu, "A personal facial expression monitoring system using deep learning," Ph.D. dissertation, 2017.

[23] R. Gandhi, *Support vector machine - introduction to machine learning algorithms*, Jul. 2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[24] M. I. U. Haque and D. Valles, "A facial expression recognition approach using dcnn for autistic children to identify emotions," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, IEEE, 2018, pp. 546–551.

[25] T. Kalsum, S. M. Anwar, M. Majid, B. Khan, and S. M. Ali, "Emotion recognition from facial expressions using hybrid feature descriptors," *IET Image Processing*, vol. 12, no. 6, pp. 1004–1012, 2018.

[26] N. Raut, "Facial emotion recognition using machine learning," 2018.

[27] S. Saha, *A comprehensive guide to convolutional neural networks-the eli5 way*, Dec. 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[28] N. Tasnim, A. Khwaja, and H. Rashid, "Emotion recognition from facial expression of autism spectrum disordered children using image processing and machine learning algorithms," Ph.D. dissertation, BRAC University, 2018.

[29] A. S. 4. out this Author's contributed articles., A. S. 44, and C. out this Author's contributed articles., *Decision tree implementation using python*, Nov. 2019. [Online]. Available: https://www.geeksforgeeks.org/decision-tree-implementation-python/.

[30] J. Brownlee, *A gentle introduction to dropout for regularizing deep neural networks*, Aug. 2019. [Online]. Available: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/.

[31] Chris, *How to use dropout with keras?* Dec. 2019. [Online]. Available: https://www.machinecurve.com/index.php/2019/12/18/how-to-use-dropout-with-keras/.

[32] J. Jeong, *The most intuitive and easiest guide for cnn*, Jul. 2019. [Online]. Available: https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480.

[33] D. Liliana, "Emotion recognition from facial expression using deep convolutional neural network," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1193, 2019, p. 012 004.

[34] C. Maklin, *Decision tree in python*, Jul. 2019. [Online]. Available: https://towardsdatascience.com/decision-tree-in-python-b433ae57fb93.

[35] S. Minaee and A. Abdolrashidi, "Deep-emotion: Facial expression recognition using attentional convolutional network," *arXiv preprint arXiv:1902.01019*, 2019.

[36] R. Prabhu, *Understanding of convolutional neural network (cnn) - deep learning*, Nov. 2019. [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

[37] R. Thakur, *Transfer learning from scratch using keras*, Aug. 2019. [Online]. Available: https://medium.com/@1297rohit/transfer-learning-from-scratch-using-keras-339834b153b9.

[38] Y. Zhu, *Automated supply-chain quality inspection using image analysis and machine learning*, 2019.

[39] J. Brownlee, *How do convolutional layers work in deep learning neural networks?* Apr. 2020. [Online]. Available: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

[40] *Inceptionv3*, Feb. 2020. [Online]. Available: https://en.wikipedia.org/wiki/Inceptionv3.

[41] V. Kurama, *A guide to resnet, inception v3, and squeezenet*, Jun. 2020. [Online]. Available: https://blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/.

[42] M. Walia, *Keras.conv2d class*, May 2020. [Online]. Available: https://www.geeksforgeeks.org/keras-conv2d-class/.

[43] *3.2.4.3.1. sklearn.ensemble.randomforestclassifier*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[44] *Convolutional neural networks in python*. [Online]. Available: https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python.

[45] P. Durham, *Random forests® in python*. [Online]. Available: https://www.kdnuggets.com/2016/12/random-forests-python.html.

[46] D. Patel, *Linear discriminant analysis*. [Online]. Available: https://github.com/topics/linear-discriminant-analysis?o=desc.

[47] *Python: Pandas dataframe*. [Online]. Available: https://www.geeksforgeeks.org/python-pandas-dataframe/.

[48] *Sklearn.linear$_m$odel.logisticregression*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[49] K. Team, *Keras documentation: Inceptionv3*. [Online]. Available: https://keras.io/api/applications/inceptionv3/.

[50] L. Thomas. [Online]. Available: https://datatofish.com/logistic-regression-python/.

[51] *Using the keras flatten operation in cnn models with code examples*. [Online]. Available: https://missinglink.ai/guides/keras/using-keras-flatten-operation-cnn-models-code-examples/.