MANIPAL ACADEMY of HIGHER EDUCATION
(Deemed to be University under Section 3 of the UGC Act, 1956)

MANIPAL SCHOOL OF INFORMATION SCIENCES

(A Constituent unit of MAHE, Manipal)

Facial Expression Recognition Using Convolutional Neural Network

Reg. Number	Name	Branch
201046019	Swathi M Huttada	Big Data and Data Analytics
201046020	Rakshita Raghava Shetty	Big Data and Data Analytics
201046037	Bonagiri Sri Pranathi	Big Data and Data Analytics

Under the guidance of

Dr. Harishchandra Hebbar

Manipal School of Information Sciences,

MAHE, MANIPAL

25/01/2021

MANIPAL SCHOOL OF INFORMATION SCIENCES
MANIPAL
(A constituent unit of MAHE, Manipal)

DECLARATION

We declare that this mini project, submitted for the evaluation of course work of Mini Project to Manipal School of Information Sciences, is extending an existing idea available at (**https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9031283**)), conducted under the supervision of my guide and panel members, **Prof. Samarendranath Bhattacharya, Prof. Raghudatesh G P**. References, help and material obtained from other sources have been duly acknowledged.

ABSTRACT

Emojis or avatars are ways to indicate nonverbal cues. These cues have become an important part of online chatting, product review, brand emotion, and many more. Human emotions and intentions are expressed through facial expressions, which play a communicative role in interpersonal relations.

The objective is to implement Convolutional Neural Networks for classification of facial expressions. Facial images are classified into seven facial expression categories namely Happy, Sad, Surprise, Anger, Fear, Disgust, and Neutral. Kaggle dataset is used to train and test the classifier.

Keywords

Facial Expression Recognition, Convolutional Neural Network

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

CNN	Convolution Neural Network
FER	Facial Expression Recognition
GPU	Graphics Processing Unit
ReLU	Rectified Linear Unit

1. INTRODUCTION

Overview

Human beings communicate with each other in the form of speech, gestures and emotions. As such systems that can recognize the same are in great demand in many fields. With respect to artificial intelligence, a computer will be able to interact with humans much more naturally if they are capable of understanding human emotion. However not all emotions can be inferred just by looking at someone.

Emotions can be inferred from a person's actions, speech, writing and facial expressions. In terms of facial emotion recognition, one major challenge lies in the data collected. Most datasets contain labelled images which are generally posed.

The objective of the project is to classify human faces into one of the seven universal emotions using convolution neural network. Many papers have been published using deep learning for facial emotion recognition [6] [4] [3].For real-time purposes, facial emotion recognition has a number of applications.

1.1 Applications of Facial expressions:-

1. **Gaming Industry -** Emotion-aware games can be developed which could vary the difficulty of a level depending on the player's emotions.
2. **Emotional support** - Nurse bots can remind older patients to take their medication and 'talk' with them every day to monitor their overall wellbeing.
3. **Mental health treatment** - Emotion AI-powered chatbots can imitate a therapist or a counsellor and help automate talk therapy and accessibility. There are also mood tracking apps like Woebot that help people manage mental health through short daily chat conversations, mood tracking, games, curated videos, etc.
4. **AI as medical assistants** -Emotion AI can assist doctors with diagnosis and intervention and provide better care. Applications like Affectiva allow measuring a patient's heart-rate without the need of wearing a sensor: the program can track color changes in the person's face with every heartbeat.
5. **Advertising research -** Emotion is the core of effective advertising: the shift from negative to positive emotions can ultimately increase sales. Emotional AI-powered solutions like Affdex by Affectiva allow marketers to remotely measure consumer emotional responses to ads, videos, and TV shows and better evaluate their relevance.
6. **Personalization -** A better understanding of human emotional responses to marketing campaigns and the ability to deliver the right content through the right channel at the right time.
7. **Law enforcement -** Emotion detection technology allows performing real-time analysis of the reactions in the suspects during interrogations, and analysing audio and video recordings.

1.2 Problem definition

Human emotions and intentions are expressed through facial expressions and deriving an efficient and effective feature is the fundamental component of facial expression system. Facial expressions convey non-verbal cues, which play an important role in interpersonal relations. Automatic recognition of facial expressions can be an important component of natural human-machine interfaces; it may also be used in behavioural science and in clinical practice. An automatic Facial Expression Recognition system needs to solve the following problems: detection and location of faces in a cluttered scene, facial feature extraction, and facial expression classification.

1.3 Scope of the Project

In this project facial expression recognition system is implemented using convolution neural network. Facial images are classified into seven facial expression categories namely Anger, Disgust, Fear, Happy, Sad, Surprise and 'Neutral. Kaggle dataset is used to train and test the classifier.

2. LITERATURE REVIEW

Two different approaches are used for facial expression recognition, both of which include two different methodologies, exist [6]. Dividing the face into separate action units or keeping it as a whole for further processing appears to be the first and the primary distinction between the main approaches. In both of these approaches, two different methodologies, namely the ‘Geometric based’ and the ‘Appearance-based’ parameterizations can be used.

Making use of the whole frontal face image and processing it in order to end up with the classifications of 6 universal facial expression prototypes: disgust, fear, joy, surprise, sad ness and anger; outlines the first approach. Here, it is assumed that each of the above mentioned emotions have characteristic expressions on face and that’s why recognition of them is necessary and sufficient. Instead of using the face images as a whole, dividing them into some sub-sections for further processing forms up the main idea of the second approach for facial expression analysis. As expression is more related with subtle changes of some discrete features such as eyes, eyebrows and lip corners; these fine-grained changes are used for analyzing automated recognition.

There are two main methods that are used in both of the above explained approaches. Geometric Based Parameterization is an old way which consists of tracking and processing the motions of some spots on image sequences, firstly presented by Suwa et al to recognize facial expressions [7]. Cohn and Kanade later on tried geometrical modeling and tracking of facial features by claiming that each AU is presented with a specific set of facial muscles [8]. The disadvantages of this method are the contours of these features and components have to be adjusted manually in this frame, the problems of robustness and difficulties come out in cases of pose and illumination changes while the tracking is applied on images, as actions & expressions tend to change both in morphological and in dynamical senses, it becomes hard to estimate general parameters for movement and displacement. Therefore, ending up with robust decisions for facial actions under these varying conditions becomes to be difficult.

Rather than tracking spatial points and using positioning and movement parameters that vary within time, color (pixel) information of related regions of face are processed in Appearance Based Parameterizations; in order to obtain the parameters that are going to form the feature 8 vectors. Different features such as Gabor, Haar wavelet coefficients, together with feature extraction and selection methods such as PCA, LDA, and Adaboost are used within this framework.

For classification problem, algorithms like Machine learning, Neural Network, Support Vector Machine, Deep learning, Naive Bayes are used.

Raghuvanshi A. et al have built a Facial expression recognition system upon recent research to classify images of human faces into discrete emotion categories using convolutional neural networks [9]. Alizadeh, Shima, and AzarFazel have developed Facial Expression Recognition system using Convolutional Neural Networks based on Torch model [10].

3. COMPUTER VISION

Computer vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos.

From the perspective of engineering, it seeks to automate tasks that the human visual system can do. Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems.

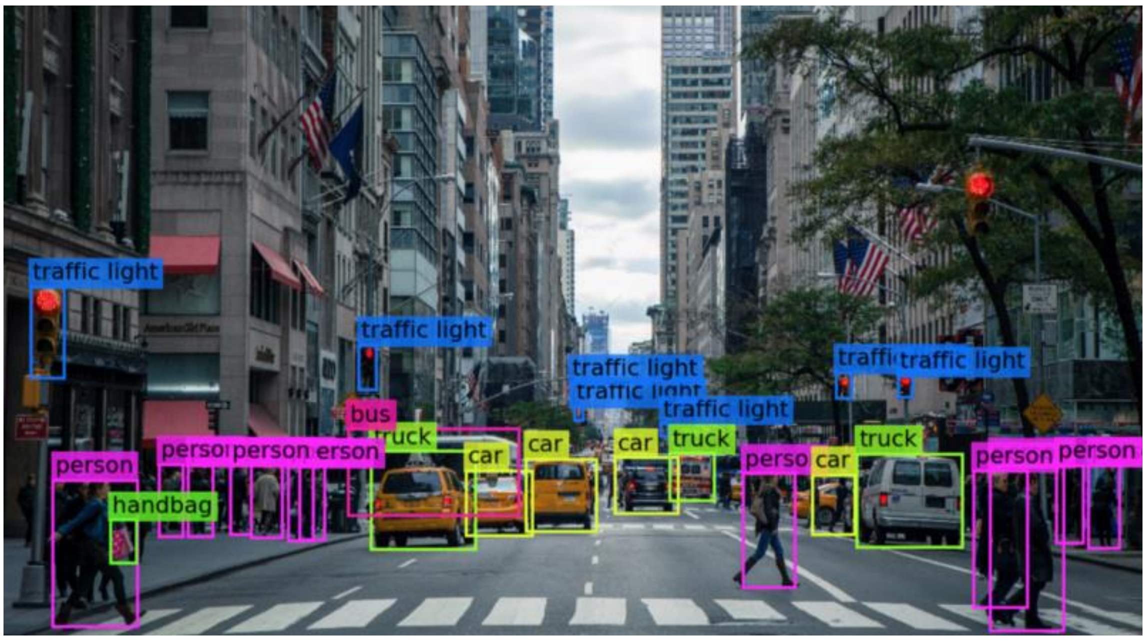


Figure1. Computer vision

In the figure1, as human eye see various objects such as car, track, traffic light, person, etc. Similarly computer vision technique is used to make computer to gain the information that are visible to human eye.

3.1 Uses of Computer vision:-

1. **Automatic inspection** - example in manufacturing applications;
2. **Assisting humans in identification tasks** - example a species identification system.
3. **Controlling processes**- example an industrial robot.
4. **Detecting events**- example for visual surveillance or people counting example in the restaurant industry.
5. **Interaction** - example as the input to a device for computer-human interaction.
6. **Modelling objects or environments** - example medical image analysis or topographical modelling
7. **Navigation** - example by an autonomous vehicle or mobile robot.
8. **Organizing information** - for indexing databases of images and image sequences.

4. CONVOLUTION NEURAL NETWORK

A convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analysing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

4.1 Architecture

Convolution is one of the main building blocks of a CNN. The term convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information.

A Convolutional Neural Network/ConvNet is a feed-forward neural network that is generally used to analyze visual images by processing data with grid-like topology. It is used to detect and classify objects in an image.

Convolutional Neural Networks have a different architecture than regular Neural Networks. Regular Neural Networks transform an input by putting it through a series of hidden layers. Every layer is made up of a set of neurons, where each layer is fully connected to all neurons in the layer before. Finally, there is a last fully-connected layer — the output layer — that represent the predictions.

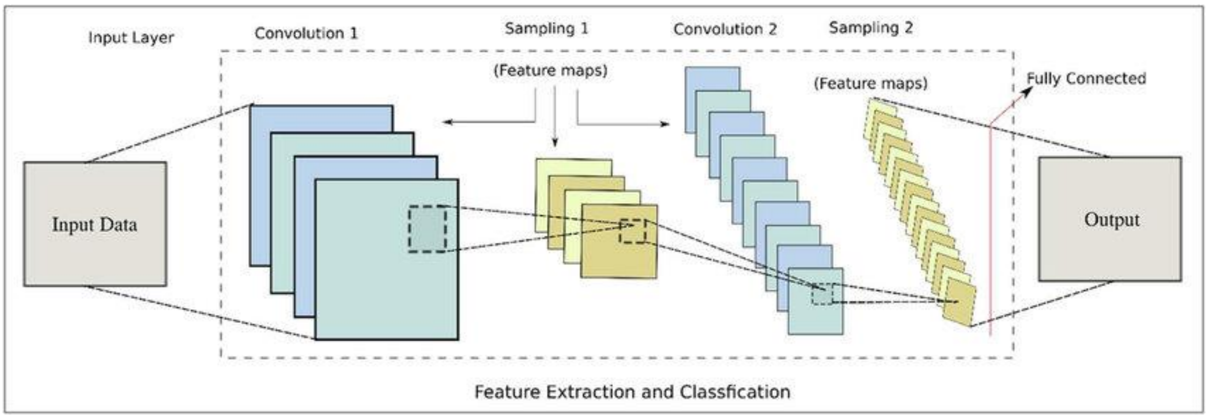


Figure 2 - Diagrammatic Representation of convolution neural network

Convolutional Neural Networks are a bit different. First of all, the layers are organised in 3 dimensions: width, height and depth. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension.

4.2Layers of Convolution Neural Network

CNNs have two components:

1. The Hidden layers/Feature extraction part

   In this part, the network will perform a series of convolutions and pooling operations during which the features are detected.

2. The Classification part

   Here, the fully connected layers will serve as a classifier on top of these extracted features. They will assign a probability for the object on the image being what the algorithm predicts it is.

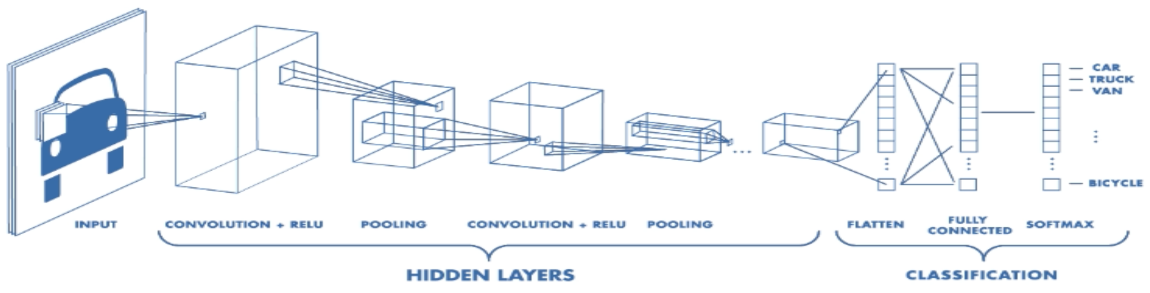
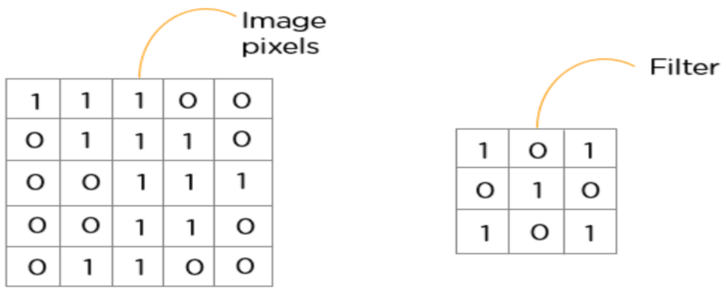


Figure 3 – Diagrammatic Representation of hidden layers and Classification in CNN

Feature extraction

1. **Convolution layer** - The convolution is performed on the input data with the use of a filter or kernel (these terms are used interchangeably) to then produce a feature map.We execute a convolution by sliding the filter over the input. At every location, a matrix multiplication is performed and sums the result onto the feature map.The area of our filter is also called the receptive field, named after the neuron cells! The size of this filter is 3x3.

   Kernel - or mask is a small matrix used for blurring, sharpening, embossing, edge detection, and more. This is accomplished by doing a convolution between a kernel and an image.



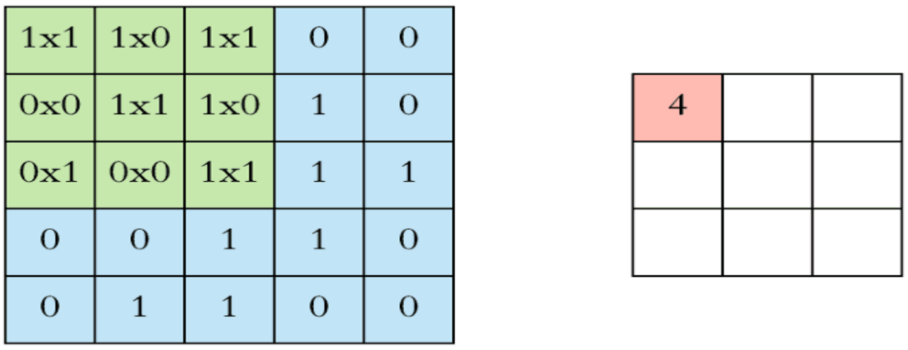
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	0	1
0	1	0
1	0	1

-

Figure 4 - Representation of feature extraction

Filter (the green square) is sliding over our input (the blue square) and the sum of the convolution goes into the feature map (the red square).



1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

Figure 5 -Representation of filter

Stride is the size of the step the convolution filter moves each time. A stride size is usually 1, meaning the filter slides pixel by pixel. The animation shows stride size 1 in action.

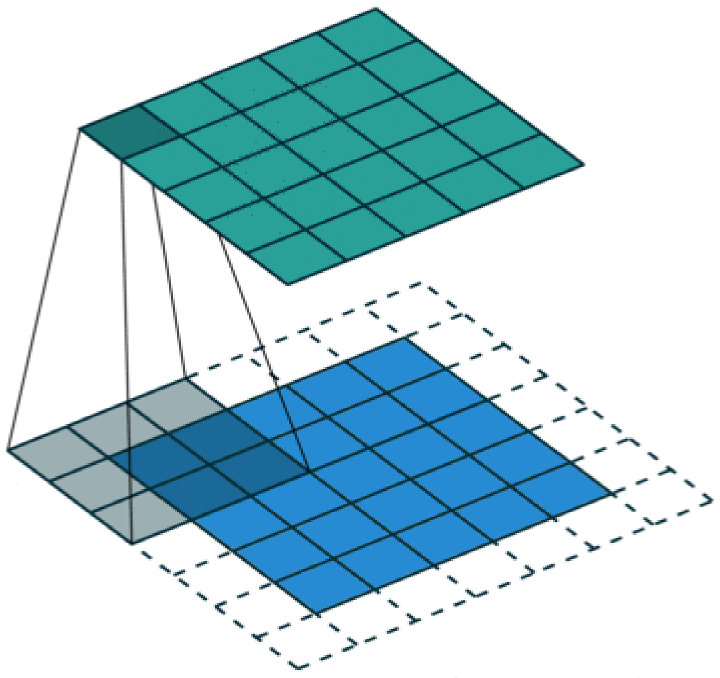


Figure 6 –Representation of Stride

Because the size of the feature map is always smaller than the input, we have to do something to prevent our feature map from shrinking. This is where we use **padding**. A layer of zero-value pixels is added to surround the input with zeros, so that our feature map will not shrink. In addition to keeping the spatial size constant after performing convolution, padding also improves performance and makes sure the kernel and stride size will fit in the input.

2. **ReLu (Rectified Linear Unit) Layer**–The convolution layer generates a matrix that is much smaller in size than the original image. This matrix runs through an activation layer, which introduces non-linearity to allow the network to train itself.

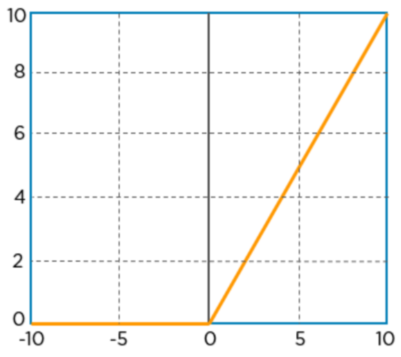


Figure 7 – Diagrammatic Representation ofRelu

Weights–It decides how much influence the input will have on the output.

Biases - Bias unit guarantees that even when all the inputs are zeros there will still be activation in the neuron.

$$Y = \sum (weight * input) + bias$$

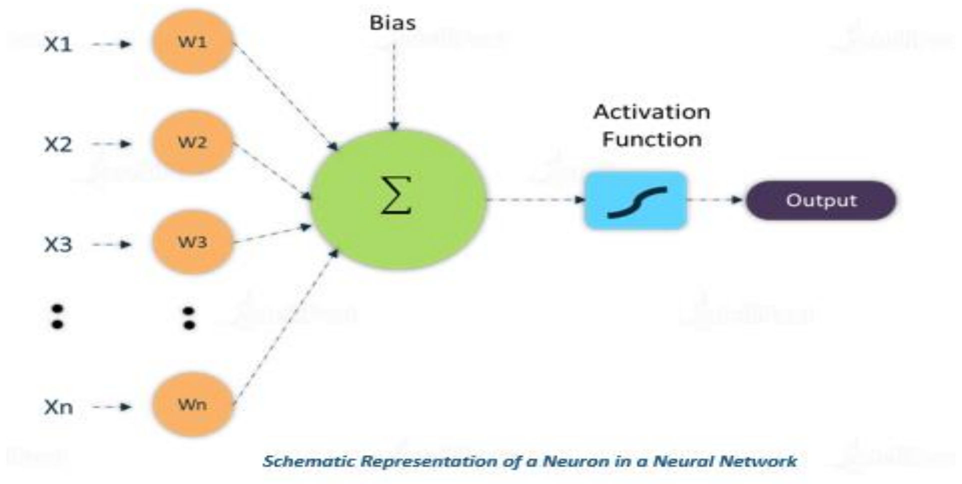


Figure 8 –Schematic Representation of a neuron in a neural network

Normalization is the process of subtracting the mean and dividing by the standard deviation.

Batch Normalization helps making the network output more stable predictions, speeds up training by an order of magnitude

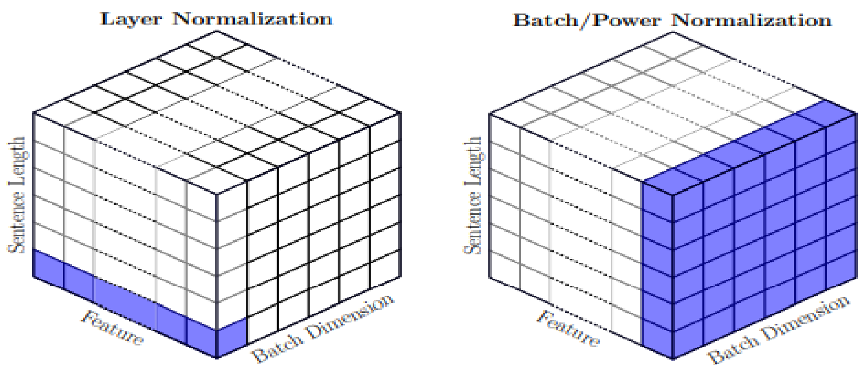


Figure 9 - Representation of layer normalization and batch normalization

3. **Pooling layer -** After a convolution and ReLulayer, it is common to add a pooling layer in between CNN layers. The function of pooling is to continuously reduce the dimensionality to reduce the number of parameters and computation in the network. This shortens the training time and controls over fitting.

   The three types of pooling operations are:

   - **Max pooling**: The maximum pixel value of the batch is selected.
   - **Min pooling**: The minimum pixel value of the batch is selected.
   - **Average pooling**: The average value of all the pixels in the batch is selected.

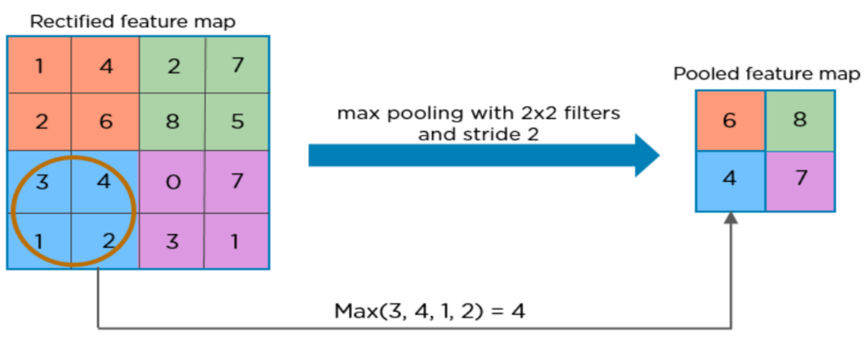


Figure 10 - Diagrammatic Representation of max pooling

The most frequent type of pooling is **max pooling**, which takes the maximum value in each window. These window sizes need to be specified beforehand. This decreases the feature map size while at the same time keeping the significant information.
Thus when using a CNN, the four important hyperparameters we have to decide on are:

- The kernel size
- The filter count
- Stride
- Padding

Classification

4. **Fully Connected Layers -** After the convolution and pooling layers, our classification part consists of a few fully connected layers. However, these fully connected layers can only accept 1 Dimensional data. To convert our 3D data to 1D, we use the function flatten in Python. This essentially arranges our 3D volume into a 1D vector.
   The last layers of a Convolutional Neural Network are fully connected layers. Neurons in a fully connected layer have full connections to all the activations in the previous layer. This part is in principle the same as a regular Neural Network.

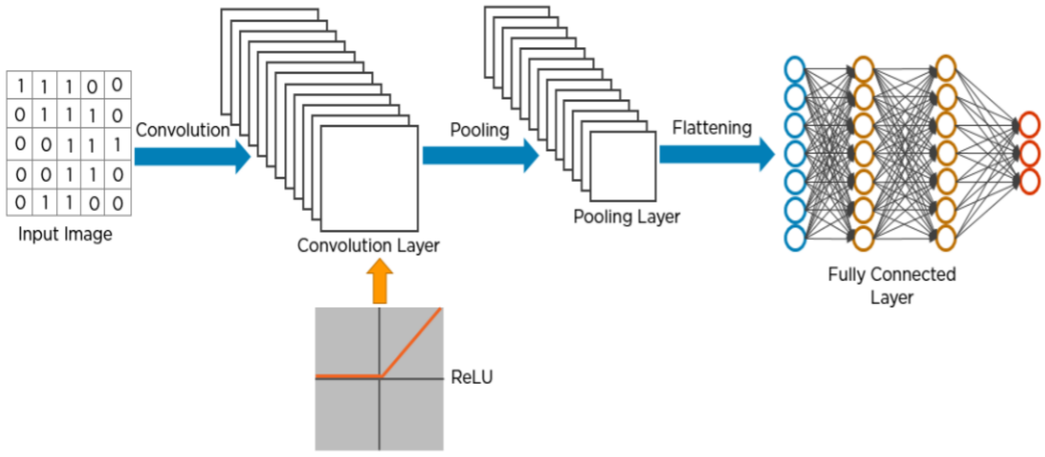


Figure 11–Diagrammatic Representation of Fully connected layer

Training

Training a CNN works in the same way as a regular neural network, using back-propagation or gradient descent. However, here this is a bit more mathematically complex because of the convolution operations**.**In order to reach the optimal weights and biases that will give us the desired output, we will have to train our neural network.

Neural network will have to undergo many epochs or iterations to give us an accurate prediction. We specify a learning rate for the model. The learning rate is the multiplier to update the parameters. It determines how rapidly they can change.

Learning rate -

$$a := a - \alpha * \frac{dL(w)}{da}$$

- := means that this is a definition, not an equation or proven statement.
- *a* is the learning rate called *alpha*
- dL(w) is the derivative of the total loss with respect to our weight w
- da is the derivative of alpha

5. METHODOLGY

5.1 Phases in CNN

1. Training Phase - The machine learns all the features from the input dataset using certain algorithms and stores it in a database as a model.
2. Test Phase - This learned information/model of the machine is then again tested for further validations.

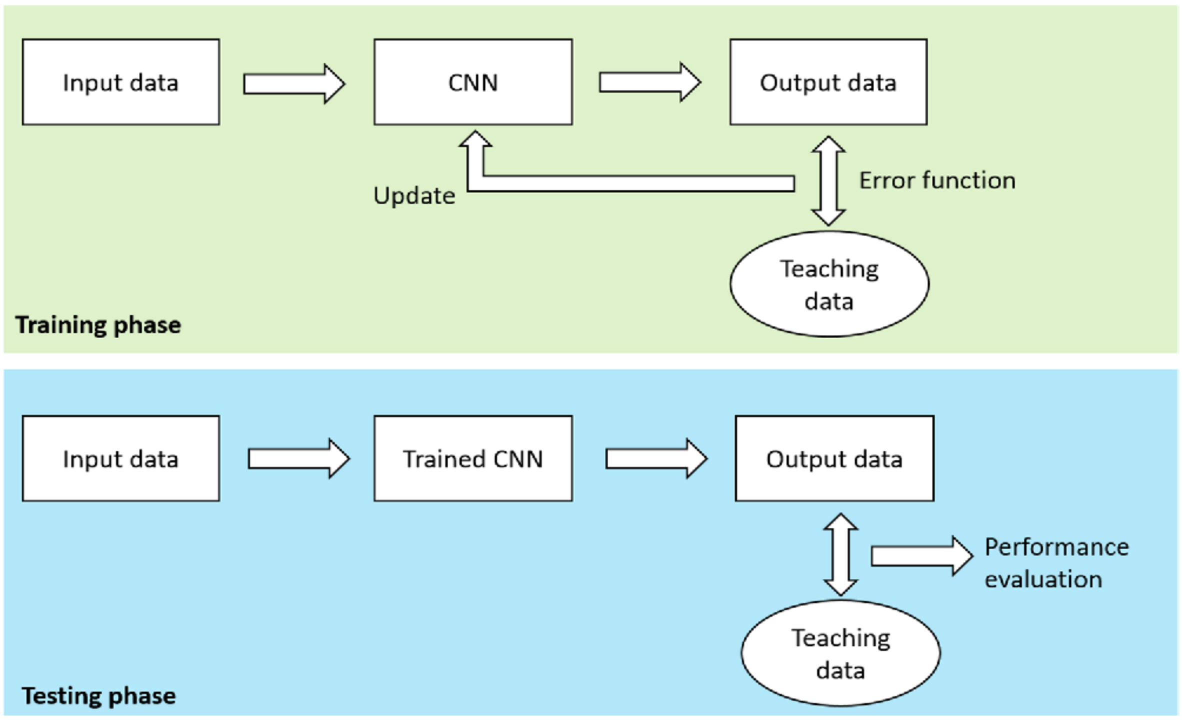


Figure 12 – Training and Testing phases

During training, the system received a training data comprising grayscale images of faces with their respective expression label and learns a set of weights for the network. The training step took as input an image with a face. Thereafter, intensity normalization is applied to the image. The normalized images are used to train the Convolutional Network. To ensure that the training performance is not affected by the order of presentation of the examples, validation dataset is used to choose the final best set of weights out of a set of trainings performed with samples presented in different orders. The output of the training step is a set of weights that achieve the best result with the training data.

During test, the system received a grayscale image of a face from test dataset, and output the predicted expression by using the final network weights learned during training. Its output is a single number that represents one of the seven basic expressions.

6. IMPLEMENTATION

1. Install dependencies.
    - **Tensorflow**– It is a open source library for numerical computation and large scale machine. It has a comprehensive, flexible ecosystem of tools, libraries and community resources.
    - **Tflearn**–It is a modules and transparent deep learning library built on top off tensorflow. It was designed to speed up experiment**.**
    - **Numpy**–It is a python library used for working with arrays.
    - **Keras**– It is free open source of python library for developing and evaluating deep learning model.
    - **opencv 3**– It is a cross platform library using which we can develop real time computer vision application. It mainly focus on image processing, video capture and analysis including features like face detection and object detection.
    - **scipy**–It is used to solve scientific and mathematical problems built on numpy extension and allows to manipulate and visualize data with a wide range of high level commands.
    - **pandas**–It is used for data manipulating and analysis. Data is messy in real world and pandas is helpful when it comes to cleaning, transforming, manipulating and analyzing data.
    - **skimage**–It is an open source python package designed for image preprocessing.
    - We used **Google Colab**with runtime **GPU.**

2. Download and prepare the data.

` The dataset from a Kaggle Facial Expression Recognition Challenge (FER2013) is used for the training and testing. It comprises pre-cropped, 48-by-48-pixel grayscale images of faces each labelled with one of the 7 emotion classes: anger, disgust, fear, happiness, sadness, surprise, and neutral. Dataset has training set of 35887 facial images with facial expression labels. The dataset has class imbalance issue, since some classes have large number of examples while some has few. The dataset is balanced using oversampling, by increasing numbers in minority classes.

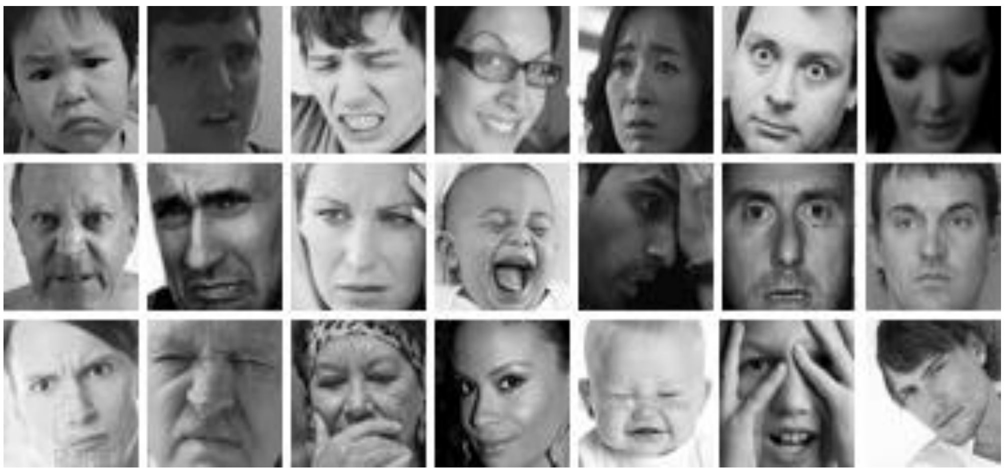
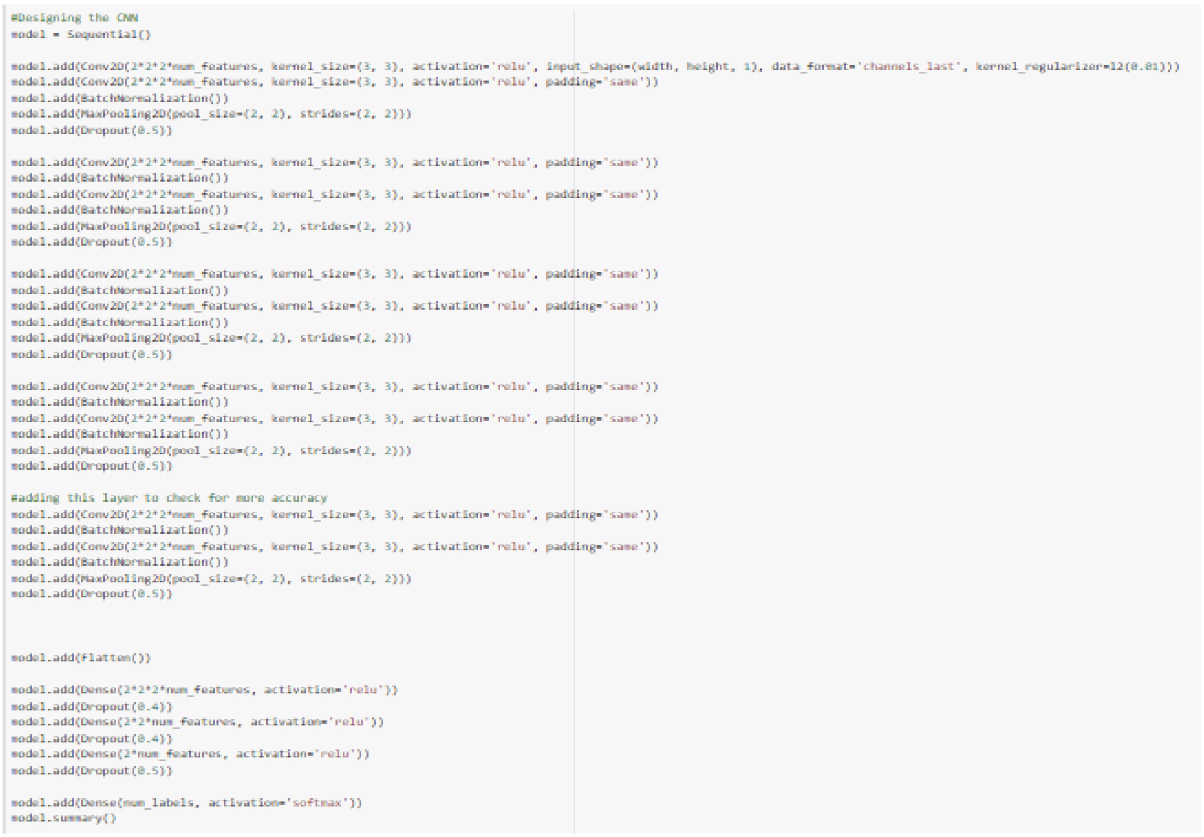


Figure 13:Example images from the FER2013. Images in the same column depict identical expressions, namely anger, disgust, fear, happiness, sadness, surprise, as well as neutral.

3. Train the data.

This step is the most important part of the entire process as we design the CNN through which we will pass our features to train the model and eventually test it using the test features. We have used a combination of several different functions to construct CNN which we will discuss one by one.



```
#Designing the CNN
model = Sequential()

model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', input_shape=(width, height, 1), data_format='channels_last', kernel_regularizer=l2(0.01)))
model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))

#adding this layer to check for more accuracy
model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))



model.add(Flatten())

model.add(Dense(2*2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*num_features, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(num_labels, activation='softmax'))
model.summary()
```

Figure 14: training of CNN Model

1. Sequential() - A sequential model is just a linear stack of layers which is putting layers on top of each other as we progress from the input layer to the output layer. You can read more about this here.

2. model.add(Conv2D()) - This is a 2D Convolutional layer which performs the convolution operation as described at the beginning of this post. To quote Keras Documentation " This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs." Here we are using a 3x3 kernel size and Rectified Linear Unit (ReLU) as our activation function.

3. model.add(BatchNormalization()) - It performs the batch normalization operation on inputs to the next layer so that we have our inputs in a specified scale say 0 to 1 instead of being scattered all over the place.

4. model.add(MaxPooling2D()) - This function performs the pooling operation on the data as explained at the beginning of the post. We are taking a pooling window of 2x2 with 2x2 strides in this model.

5. model.add(Dropout()) - As explained above Dropout is a technique where randomly selected neurons are ignored during the training. They are "dropped out" randomly. This reduces overfitting.

6. model.add(Flatten()) - This just flattens the input from ND to 1D and does not affect the batch size.

7. model.add(Dense()) - According to Keras Documentation, Dense implements the operation: output = activation(dot(input, kernel)where activationis the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer. In simple words, it is the final nail in the coffin which uses the features learned using the layers and maps it to the label. During testing, this layer is responsible for creating the final label for the image being processed.

After the model.summary() function is executed, the output looks something like this –

```
Model: "sequential"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

conv2d (Conv2D)              (None, 46, 46, 512)       5120

_________________________________________________________________

conv2d_1 (Conv2D)            (None, 46, 46, 512)       2359808
```

```
_________________________________________________________________

batch_normalization (BatchNo (None, 46, 46, 512)       2048

_________________________________________________________________

max_pooling2d (MaxPooling2D) (None, 23, 23, 512)       0

_________________________________________________________________

dropout (Dropout)            (None, 23, 23, 512)       0

_________________________________________________________________

conv2d_2 (Conv2D)            (None, 23, 23, 512)       2359808

_________________________________________________________________

batch_normalization_1 (Batch (None, 23, 23, 512)       2048

_________________________________________________________________

conv2d_3 (Conv2D)            (None, 23, 23, 512)       2359808

_________________________________________________________________

batch_normalization_2 (Batch (None, 23, 23, 512)       2048

_________________________________________________________________

max_pooling2d_1 (MaxPooling2 (None, 11, 11, 512)       0

_________________________________________________________________

dropout_1 (Dropout)          (None, 11, 11, 512)       0

_________________________________________________________________

conv2d_4 (Conv2D)            (None, 11, 11, 512)       2359808

_________________________________________________________________

batch_normalization_3 (Batch (None, 11, 11, 512)       2048

_________________________________________________________________

conv2d_5 (Conv2D)            (None, 11, 11, 512)       2359808

_________________________________________________________________

batch_normalization_4 (Batch (None, 11, 11, 512)       2048

_________________________________________________________________

max_pooling2d_2 (MaxPooling2 (None, 5, 5, 512)         0

_________________________________________________________________

dropout_2 (Dropout)          (None, 5, 5, 512)         0

_________________________________________________________________

conv2d_6 (Conv2D)            (None, 5, 5, 512)         2359808
```

```
_________________________________________________________________

batch_normalization_5 (Batch (None, 5, 5, 512)         2048

_________________________________________________________________

conv2d_7 (Conv2D)            (None, 5, 5, 512)         2359808

_________________________________________________________________

batch_normalization_6 (Batch (None, 5, 5, 512)         2048

_________________________________________________________________

max_pooling2d_3 (MaxPooling2 (None, 2, 2, 512)         0

_________________________________________________________________

dropout_3 (Dropout)          (None, 2, 2, 512)         0

_________________________________________________________________

conv2d_8 (Conv2D)            (None, 2, 2, 512)         2359808

_________________________________________________________________

batch_normalization_7 (Batch (None, 2, 2, 512)         2048

_________________________________________________________________

conv2d_9 (Conv2D)            (None, 2, 2, 512)         2359808

_________________________________________________________________

batch_normalization_8 (Batch (None, 2, 2, 512)         2048

_________________________________________________________________

max_pooling2d_4 (MaxPooling2 (None, 1, 1, 512)         0

_________________________________________________________________

dropout_4 (Dropout)          (None, 1, 1, 512)         0

_________________________________________________________________

flatten (Flatten)            (None, 512)               0

_________________________________________________________________

dense (Dense)                (None, 512)               262656

_________________________________________________________________

dropout_5 (Dropout)          (None, 512)               0

_________________________________________________________________

dense_1 (Dense)              (None, 256)               131328

_________________________________________________________________

dropout_6 (Dropout)          (None, 256)               0
```

```
_________________________________________________________________

dense_2 (Dense)              (None, 128)               32896

_________________________________________________________________

dropout_7 (Dropout)          (None, 128)               0

_________________________________________________________________

dense_3 (Dense)              (None, 7)                 903

=================================================================

Total params: 21,689,607

Trainable params: 21,680,391

Non-trainable params: 9,216
```

4.Evaluate a trained model.

The model is trained for 100 epochs and we have used Adam optimiser with default values of learning rate(lr) = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e-7. We have saved the trained model in a .json file and its respective weights in .h5 file for further use. We achieved an accuracy of 66.36% for the dataset we have chosen as training dataset.

5.Recognizing facial expressions from an image file.

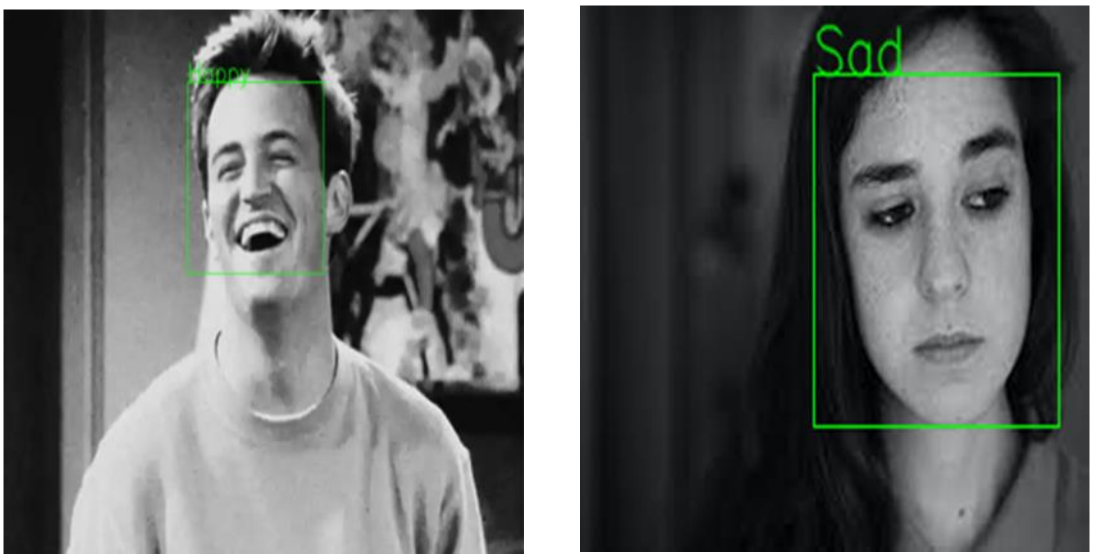


Figure 15:classified correctly

The above pictures is classified correctly when the input image file is given.

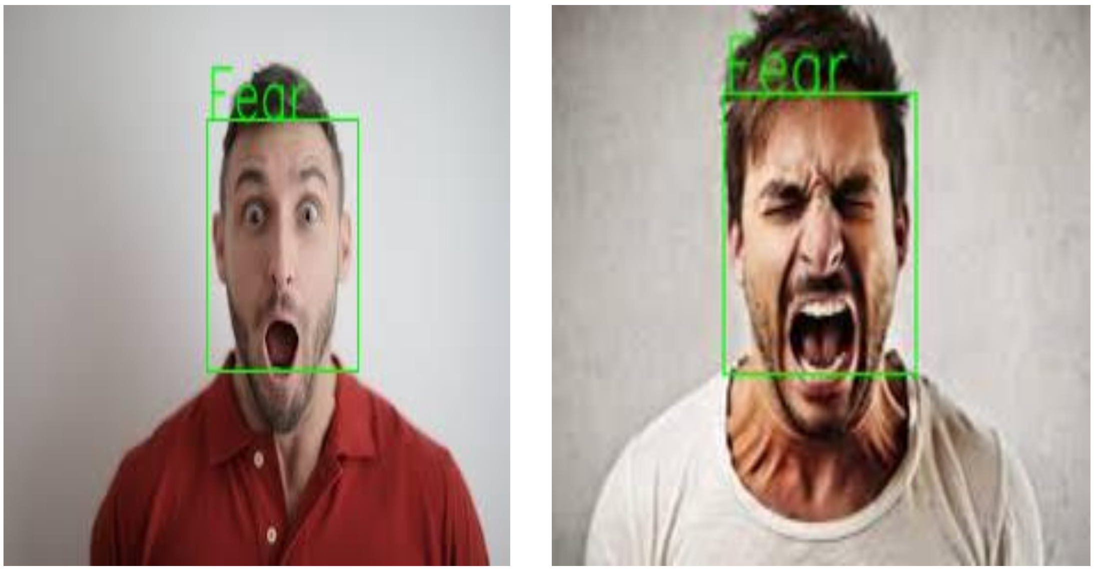


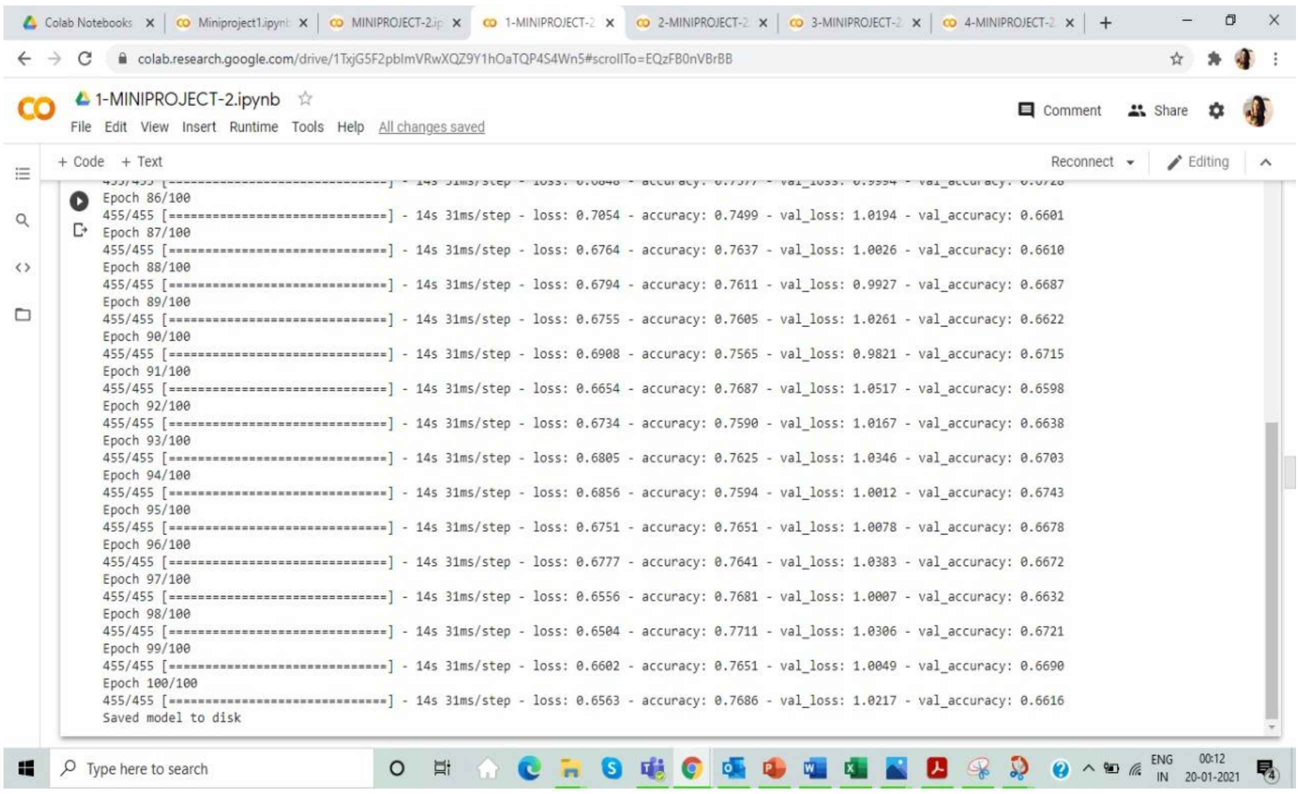
Figure 16: misclassified image

The above picture is misclassified when the input image file is given.

7. RESULTS AND ANALYSIS

CNN architecture for facial expression recognition as mentioned above was implemented in Python. Validation set was used to validate the training process. In last batch of every epoch in validation cost, validation error, training cost, training error is calculated.Input parameters for training are image set and corresponding output labels. The training process updated the weights of feature maps and hidden layers based on hyper-parameters such as learning rate, momentum, regularization and decay

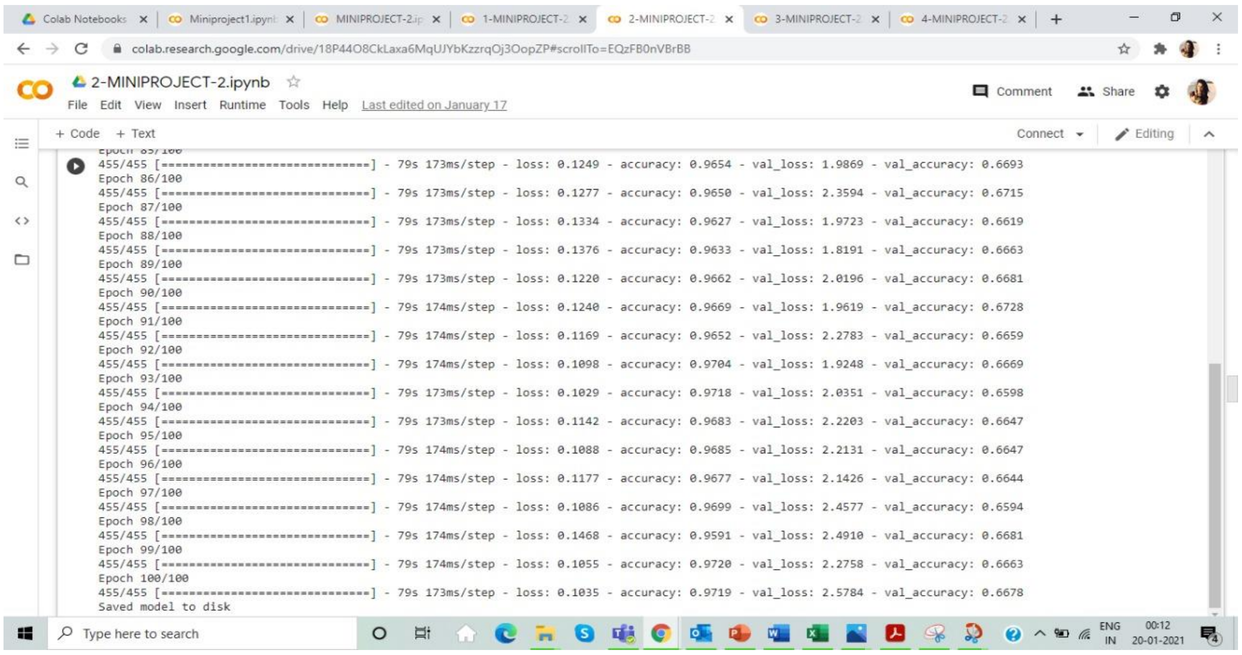
We design the CNN through which we will pass our features to train the model and eventually test it using the test features. We trained every modification for 100 epochs of fixed batch size 64.



```
Epoch 86/100
455/455 [==============================] - 14s 31ms/step - loss: 0.7054 - accuracy: 0.7499 - val_loss: 1.0194 - val_accuracy: 0.6601
Epoch 87/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6764 - accuracy: 0.7637 - val_loss: 1.0026 - val_accuracy: 0.6610
Epoch 88/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6794 - accuracy: 0.7611 - val_loss: 0.9927 - val_accuracy: 0.6687
Epoch 89/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6755 - accuracy: 0.7605 - val_loss: 1.0261 - val_accuracy: 0.6622
Epoch 90/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6908 - accuracy: 0.7565 - val_loss: 0.9821 - val_accuracy: 0.6715
Epoch 91/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6654 - accuracy: 0.7687 - val_loss: 1.0517 - val_accuracy: 0.6598
Epoch 92/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6734 - accuracy: 0.7590 - val_loss: 1.0167 - val_accuracy: 0.6638
Epoch 93/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6805 - accuracy: 0.7625 - val_loss: 1.0346 - val_accuracy: 0.6703
Epoch 94/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6856 - accuracy: 0.7594 - val_loss: 1.0012 - val_accuracy: 0.6743
Epoch 95/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6751 - accuracy: 0.7651 - val_loss: 1.0078 - val_accuracy: 0.6678
Epoch 96/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6777 - accuracy: 0.7641 - val_loss: 1.0383 - val_accuracy: 0.6672
Epoch 97/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6556 - accuracy: 0.7681 - val_loss: 1.0007 - val_accuracy: 0.6632
Epoch 98/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6504 - accuracy: 0.7711 - val_loss: 1.0306 - val_accuracy: 0.6721
Epoch 99/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6602 - accuracy: 0.7651 - val_loss: 1.0049 - val_accuracy: 0.6690
Epoch 100/100
455/455 [==============================] - 14s 31ms/step - loss: 0.6563 - accuracy: 0.7686 - val_loss: 1.0217 - val_accuracy: 0.6616
Saved model to disk
```

Figure 17 : Training of modification-1 for 100 epochs

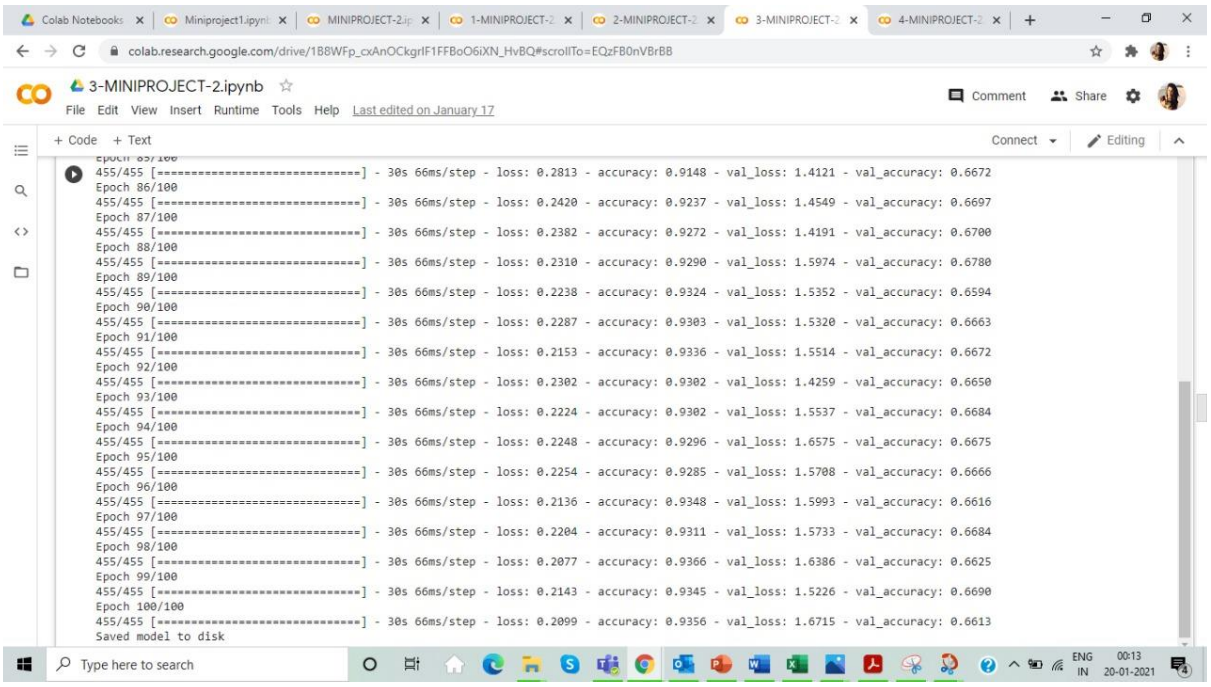
In the above figure shows that we trained for 100 epochs for miniproject1 and we have got validation accuracy as 65.05%.



```
455/455 [==============================] - 79s 173ms/step - loss: 0.1249 - accuracy: 0.9654 - val_loss: 1.9869 - val_accuracy: 0.6693
Epoch 86/100
455/455 [==============================] - 79s 173ms/step - loss: 0.1277 - accuracy: 0.9650 - val_loss: 2.3594 - val_accuracy: 0.6715
Epoch 87/100
455/455 [==============================] - 79s 173ms/step - loss: 0.1334 - accuracy: 0.9627 - val_loss: 1.9723 - val_accuracy: 0.6619
Epoch 88/100
455/455 [==============================] - 79s 173ms/step - loss: 0.1376 - accuracy: 0.9633 - val_loss: 1.8191 - val_accuracy: 0.6663
Epoch 89/100
455/455 [==============================] - 79s 173ms/step - loss: 0.1220 - accuracy: 0.9662 - val_loss: 2.0196 - val_accuracy: 0.6681
Epoch 90/100
455/455 [==============================] - 79s 174ms/step - loss: 0.1240 - accuracy: 0.9669 - val_loss: 1.9619 - val_accuracy: 0.6728
Epoch 91/100
455/455 [==============================] - 79s 174ms/step - loss: 0.1169 - accuracy: 0.9652 - val_loss: 2.2783 - val_accuracy: 0.6659
Epoch 92/100
455/455 [==============================] - 79s 174ms/step - loss: 0.1098 - accuracy: 0.9704 - val_loss: 1.9248 - val_accuracy: 0.6669
Epoch 93/100
455/455 [==============================] - 79s 173ms/step - loss: 0.1029 - accuracy: 0.9718 - val_loss: 2.0351 - val_accuracy: 0.6598
Epoch 94/100
455/455 [==============================] - 79s 173ms/step - loss: 0.1142 - accuracy: 0.9683 - val_loss: 2.2203 - val_accuracy: 0.6647
Epoch 95/100
455/455 [==============================] - 79s 174ms/step - loss: 0.1088 - accuracy: 0.9685 - val_loss: 2.2131 - val_accuracy: 0.6647
Epoch 96/100
455/455 [==============================] - 79s 174ms/step - loss: 0.1177 - accuracy: 0.9677 - val_loss: 2.1426 - val_accuracy: 0.6644
Epoch 97/100
455/455 [==============================] - 79s 174ms/step - loss: 0.1086 - accuracy: 0.9699 - val_loss: 2.4577 - val_accuracy: 0.6594
Epoch 98/100
455/455 [==============================] - 79s 174ms/step - loss: 0.1468 - accuracy: 0.9591 - val_loss: 2.4910 - val_accuracy: 0.6681
Epoch 99/100
455/455 [==============================] - 79s 174ms/step - loss: 0.1055 - accuracy: 0.9720 - val_loss: 2.2758 - val_accuracy: 0.6663
Epoch 100/100
455/455 [==============================] - 79s 173ms/step - loss: 0.1035 - accuracy: 0.9719 - val_loss: 2.5784 - val_accuracy: 0.6678
Saved model to disk
```

Figure 18 : Training of modification-2 for 100 epochs

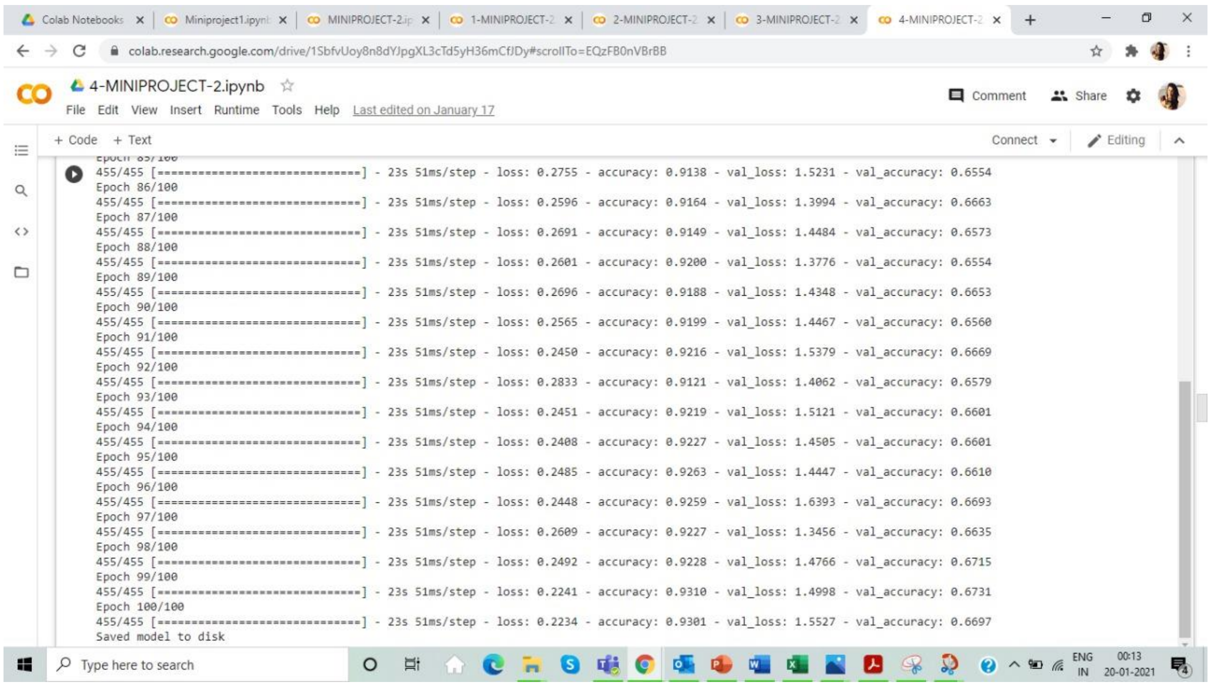
In the above figure shows that we trained for 100 epochs for miniproject2 and we have got validation accuracy as 66.36%.



```
455/455 [==============================] - 30s 66ms/step - loss: 0.2813 - accuracy: 0.9148 - val_loss: 1.4121 - val_accuracy: 0.6672
Epoch 86/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2420 - accuracy: 0.9237 - val_loss: 1.4549 - val_accuracy: 0.6697
Epoch 87/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2382 - accuracy: 0.9272 - val_loss: 1.4191 - val_accuracy: 0.6700
Epoch 88/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2310 - accuracy: 0.9290 - val_loss: 1.5974 - val_accuracy: 0.6780
Epoch 89/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2238 - accuracy: 0.9324 - val_loss: 1.5352 - val_accuracy: 0.6594
Epoch 90/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2287 - accuracy: 0.9303 - val_loss: 1.5320 - val_accuracy: 0.6663
Epoch 91/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2153 - accuracy: 0.9336 - val_loss: 1.5514 - val_accuracy: 0.6672
Epoch 92/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2302 - accuracy: 0.9302 - val_loss: 1.4259 - val_accuracy: 0.6650
Epoch 93/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2224 - accuracy: 0.9302 - val_loss: 1.5537 - val_accuracy: 0.6684
Epoch 94/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2248 - accuracy: 0.9296 - val_loss: 1.6575 - val_accuracy: 0.6675
Epoch 95/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2254 - accuracy: 0.9285 - val_loss: 1.5708 - val_accuracy: 0.6666
Epoch 96/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2136 - accuracy: 0.9348 - val_loss: 1.5993 - val_accuracy: 0.6616
Epoch 97/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2204 - accuracy: 0.9311 - val_loss: 1.5733 - val_accuracy: 0.6684
Epoch 98/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2077 - accuracy: 0.9366 - val_loss: 1.6386 - val_accuracy: 0.6625
Epoch 99/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2143 - accuracy: 0.9345 - val_loss: 1.5226 - val_accuracy: 0.6690
Epoch 100/100
455/455 [==============================] - 30s 66ms/step - loss: 0.2099 - accuracy: 0.9356 - val_loss: 1.6715 - val_accuracy: 0.6613
Saved model to disk
```

Figure 19: Training of modification-3 for 100 epochs

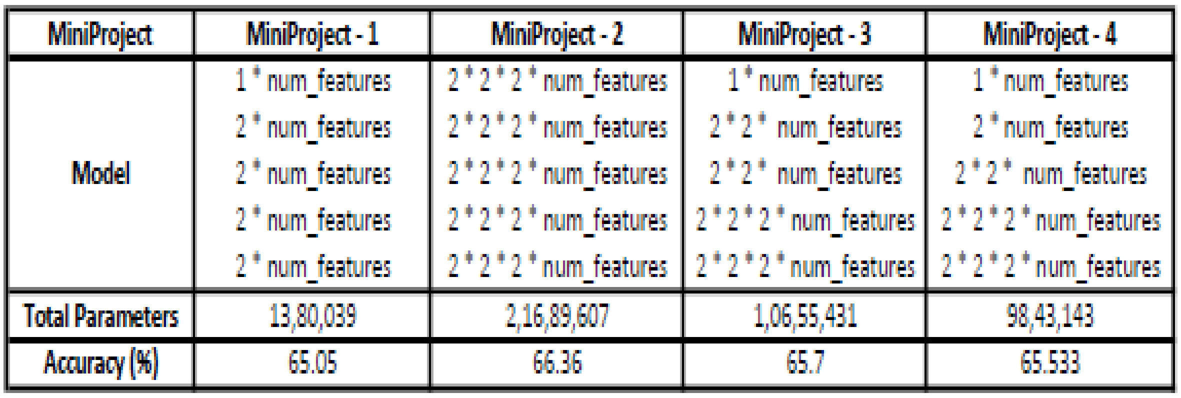
In the above figure shows that we trained for 100 epochs for miniproject3 and we have got validation accuracy as 65.7%.


4-MINIPROJECT-2.ipynb
File Edit View Insert Runtime Tools Help Last edited on January 17
Comment Share
+ Code + Text
Connect
Editing

```
455/455 [==============================] - 23s 51ms/step - loss: 0.2755 - accuracy: 0.9138 - val_loss: 1.5231 - val_accuracy: 0.6554
Epoch 86/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2596 - accuracy: 0.9164 - val_loss: 1.3994 - val_accuracy: 0.6663
Epoch 87/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2691 - accuracy: 0.9149 - val_loss: 1.4484 - val_accuracy: 0.6573
Epoch 88/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2601 - accuracy: 0.9200 - val_loss: 1.3776 - val_accuracy: 0.6554
Epoch 89/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2696 - accuracy: 0.9188 - val_loss: 1.4348 - val_accuracy: 0.6653
Epoch 90/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2565 - accuracy: 0.9199 - val_loss: 1.4467 - val_accuracy: 0.6560
Epoch 91/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2450 - accuracy: 0.9216 - val_loss: 1.5379 - val_accuracy: 0.6669
Epoch 92/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2833 - accuracy: 0.9121 - val_loss: 1.4062 - val_accuracy: 0.6579
Epoch 93/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2451 - accuracy: 0.9219 - val_loss: 1.5121 - val_accuracy: 0.6601
Epoch 94/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2408 - accuracy: 0.9227 - val_loss: 1.4505 - val_accuracy: 0.6601
Epoch 95/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2485 - accuracy: 0.9263 - val_loss: 1.4447 - val_accuracy: 0.6610
Epoch 96/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2448 - accuracy: 0.9259 - val_loss: 1.6393 - val_accuracy: 0.6693
Epoch 97/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2609 - accuracy: 0.9227 - val_loss: 1.3456 - val_accuracy: 0.6635
Epoch 98/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2492 - accuracy: 0.9228 - val_loss: 1.4766 - val_accuracy: 0.6715
Epoch 99/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2241 - accuracy: 0.9310 - val_loss: 1.4998 - val_accuracy: 0.6731
Epoch 100/100
455/455 [==============================] - 23s 51ms/step - loss: 0.2234 - accuracy: 0.9301 - val_loss: 1.5527 - val_accuracy: 0.6697
Saved model to disk
```

Figure 20: Training of modification-4 for 100 epochs

In the above figure shows that we trained for 100 epochs for miniproject4 and we have got validation accuracy as 65.53%.

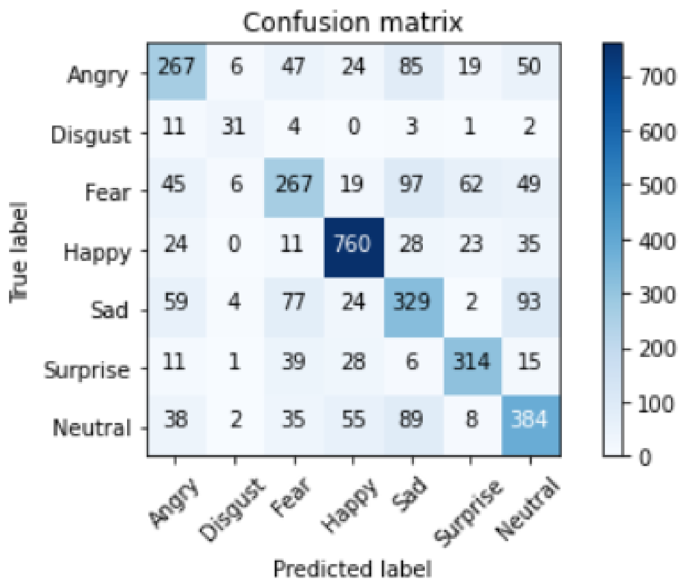


MiniProject	MiniProject - 1	MiniProject - 2	MiniProject - 3	MiniProject - 4
Model	1 * num_features	2 * 2 * 2 * num_features	1 * num_features	1 * num_features
	2 * num_features	2 * 2 * 2 * num_features	2 * 2 * num_features	2 * num_features
	2 * num_features	2 * 2 * 2 * num_features	2 * 2 * num_features	2 * 2 * num_features
	2 * num_features	2 * 2 * 2 * num_features	2 * 2 * 2 * num_features	2 * 2 * 2 * num_features
	2 * num_features	2 * 2 * 2 * num_features	2 * 2 * 2 * num_features	2 * 2 * 2 * num_features
Total Parameters	13,80,039	2,16,89,607	1,06,55,431	98,43,143
Accuracy (%)	65.05	66.36	65.7	65.533

Figure 21: Accuracy Comparision between the layers of CNN

The above table is the comparsion between various modifications named miniproject - 1, miniproject - 2, miniproject - 3, and miniproject - 4 and tells about how many layers we have added in each modification and here we have added 5 layers in each of them with different arrangement of parameters.

Confusion matrix for facial expression recognition



True label \ Predicted label	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
Angry	267	6	47	24	85	19	50
Disgust	11	31	4	0	3	1	2
Fear	45	6	267	19	97	62	49
Happy	24	0	11	760	28	23	35
Sad	59	4	77	24	329	2	93
Surprise	11	1	39	28	6	314	15
Neutral	38	2	35	55	89	8	384

Figure 22 : Confusion matrix of facial expression

The above table shows the true label and predicted label of confusion matrix for facial expression. The value of angry and fear is 267, disgust is the lowest accuracy with 31, sad is 329, surprise is 314, neutral is 384 and highest accuacyr confusion matrix is happy with 760.

8. Conclusion

In this project, convolution neural network is implemented to classify human facial expressions i.e. happy, sad, surprise, fear, anger, disgust, and neutral with different layers of CNN. The model gives us Accuracy of 66.36.

Enhancement

In the future work, the model can be extended to color images. This will allow to investigate the efficiency of pre-trained models such as AlexNet[11] or VGGNet [12] for facial emotion recognition.

9. REFERENCES

[1] Shan, C., Gong, S., & McOwan, P. W. (2005, September). Robust facial expression recognition using local binary patterns. In Image Processing, 2005. ICIP 2005. IEEE International Conference on (Vol. 2, pp. II-370). IEEE.

[2] Chibelushi, C. C., & Bourel, F. (2003). Facial expression recognition: A brief tutorial overview. CVonline: On-Line Compendium of Computer Vision, 9.

[3] "Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation". DeepLearning 0.1. LISA Lab. Retrieved 31 August 2013.

[4] Simonyan, Karen, and Andrew Zisserman. ”Very deep convolutional networks for large-scale image recognition.” arXiv preprint arXiv:1409.1556 (2014).

[5][Fer2013] Dataset from Kaggle [https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression- recognition-challenge/data]

[6] C. Zor, “Facial expression recognition,” Master’s thesis, University of Surrey, Guildford, 2008.

[7] Suwa, M.; Sugie N. and Fujimora K. A Preliminary Note on Pattern Recognition of Human Emotional Expression, Proc. International Joint Conf, Pattern Recognition, pages 408-410, 1978

[8] Recognizing action units for facial expression analysis YI Tian, T Kanade, JF Cohn IEEE Transactions on pattern analysis and machine intelligence 23 (2), 97-115

[9] Raghuvanshi, Arushi, and Vivek Choksi. "Facial Expression Recognition with Convolutional Neural Networks." Stanford University, 2016

[10] Alizadeh, Shima, and Azar Fazel. "Convolutional Neural Networks for Facial Expression Recognition." Stanford University, 2016

[11] S. Li and W. Deng, “Deep facial expression recognition: A survey,” arXiv preprint arXiv:1804.08348, 2018.

[12] Pramerdorfer, C., Kampel, M.: Facial expression recognition using convolutional neural networks: state of the art. Preprint arXiv:1612.02903v1, 2016.

[13] I. J. Goodfellow, D. Erhan, P. L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee et al., “Challenges in representation learning: A report on three machine learning contests,” in International Conference on Neural Information Processing. Springer, 2013, pp. 117–124.

[14] Y. Tang, “Deep Learning using Support Vector Machines,” in International Conference on Machine Learning (ICML) Workshops, 2013.

[15] Z. Zhang, P. Luo, C.-C. Loy, and X. Tang, “Learning Social Relation Traits from Face Images,” in Proc. IEEE Int. Conference on Computer Vision (ICCV), 2015, pp. 3631–3639.

[16] B.-K. Kim, S.-Y. Dong, J. Roh, G. Kim, and S.-Y. Lee, “Fusing Aligned and Non-Aligned Face Information for Automatic Affect Recognition in the Wild: A Deep Learning

Approach," in IEEE Conf. Computer Vision and Pattern Recognition (CVPR) Workshops, 2016, pp. 48–57.

[17] https://en.wikipedia.org/wiki/Convolutional_neural_network