CSCE 5222: Feature Engineering
Project Report

Project: Facial Emotion Detection

Team Members:

Gayatri Annapurna Vadlani

Mohammed Jhony Shaik

GitLink: https://github.com/Gayatri345/CSCE5222.git

Problem statement:

Rather than articulating, Human facial emotions convey a lot of information. Automatic facial expression recognition systems are having many applications. We can measure the effects that content and services have on the audience/users through facial emotion recognition. For example, these metrics can be used in retail, entertainment and gaming industries to evaluate audience emotions on the services.

Recognition of facial expression with high accuracy is a challenging task. Facial Expression For performing facial recognition we generally follow stages like pre-processing of input data, detecting face from the input, extraction of facial feature, and finally classify.

In this project we developed a deep neural network to detect facial emotions from a video. We used Haarcascade algorithm to extract and detect facial features. For developing Deep neural network, we used convolution layers, Dropout layers, Pooling layers, for convolution we used 3x3 filters. We also used optimizers like Adam and RMSprop. We developed a deep neural network to detect facial emotions like angry, disgust, fear, happiness, sadness, surprise, and neutral. We aim to implement facial emotion recognition from a video.

Fig.1 Sample human facial emotions

Method and Implementation:

We designed a deep learning neural network that makes machine to detect various emotions.

We implemented our project to detect facial emotions from video which uses Open CV haarcascades algorithm on video for face detection, extract the facial key features and input it to the Deep Learning model to detect the emotion from the video. We are planning to work on emotions like angry, disgust, happy, sad, surprise and neutral.

Method:

Step 1: Dataset

Step 2: Model for facial feature detection and extraction

Step 3: Design a deep neural network model for Emotion recognition

Step 4: Evaluate/Compare models on test data and custom data

Step 5: Developing a code for video input

Step 6: Testing/Deployment

Step 7: Generating outputs for the emotion

Step 1: Dataset

There are several datasets available for emotion recognition task like AffectNet, Emotic, FER-2013, etc.

We choose to use FER-2013 dataset because of the fact that it is a realistic data gathered from google search. This dataset has total 32,398 images in which 28,709 are for training set and 3,589 are for test set. All the images consists of faces in gray scale format with 48x48 pixels. The data set is annotated as 0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral.

Fig. 2 Sample images from FER-2013 dataset

Step 2: Model for facial feature detection and extraction

After some research during the initial stage of project, we decide to use haarcascade algorithm. This is a famous object detection algorithm. We used Haarcascade frontal face detection algorithm. From the papers, this algorithm uses famous Voila and Jones proposed edge or line detection techniques. This algorithm uses haar features to detect edges or lines in the image. They help to pick the line where there is sudden change in intensity levels to detect. We use haarcascade_frontalface_alt.xml file which is pretrained face detector and provided in OpenCv library. This algorithm uses:

1. Calculating Haar Feature:

   The objective is to find the sum of all image pixels lying in the darker and lighter areas of the haar feature and finding out the difference. Now we have the edge that separates darker and lighter regions.

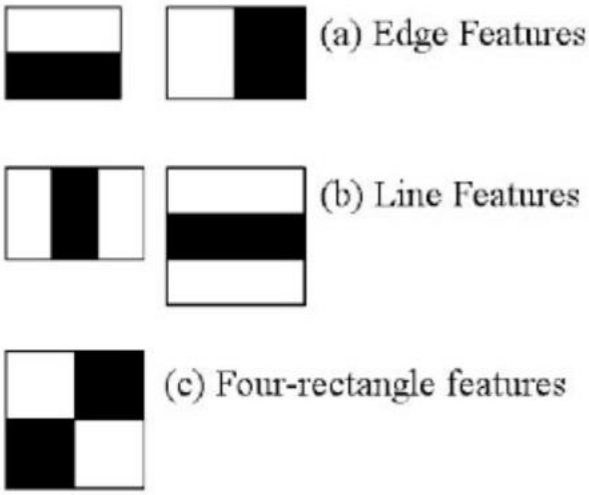


Fig 3. Sample of Haar features published in the original research paper by viola and jones

2. Creating Integral Images

   Instead of computing at each pixels, creating sub rectangles and array references for each of the sub rectangles.

3. Using Ada Boost

   In this step we use best features to train the classifier. It uses combination of weak classifiers to create strong classifier .

4. Implementing cascading classifiers

The cascade classifier is made up of a series of stages, where each stage is a collection of weak learners. Weak learners are trained using boosting, which allows for a highly accurate classifier from the mean prediction of all weak learners. Based on this prediction, the classifier either decides to indicate an object was found (positive) or move on to the next region (negative).

Implementation of HaarCascade algorithm:

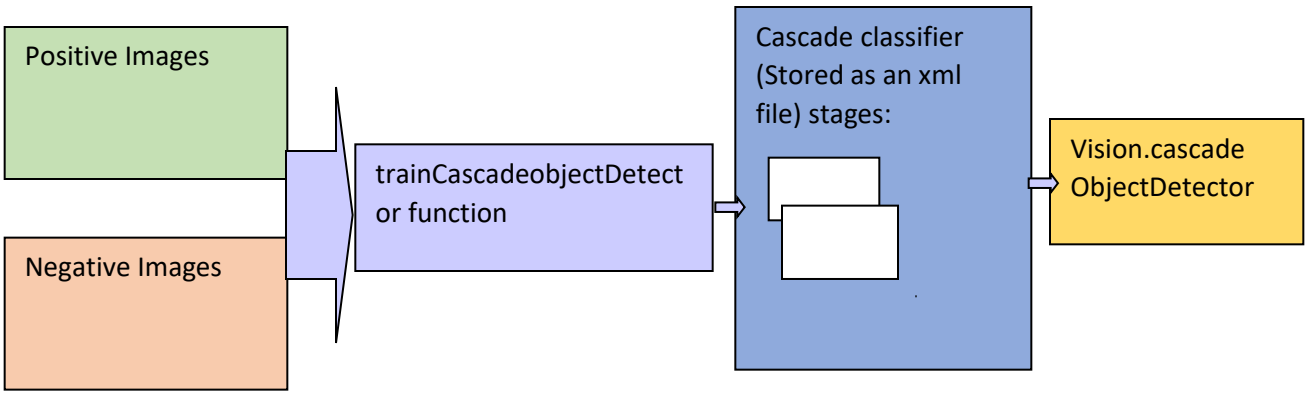


Fig. 3 Implementation of Haarcascade algorithm

This helps us to detect faces in images or videos. These features extracted from this algorithm are further used to predict emotions. In our project we have decided to use readily available Haar Cascade algorithm provided in the OpenCv library package of python. So, we are implementing facial detection and extraction with OpenCV haarcascade algorithm in python.

Step 3: Design a deep neural network model for Emotion recognition

We choose to design a deep neural network to recognize emotion from a video/image. Deep learning is the most popular technique in computer vision. For our model architecture we choose to use Convolution Neural Network.

A typical architecture of CNN has input layer, convolution layers, fully connected layers and output layer.

We planned to use **keras** library in python for creating our model. In keras we use sequential() as model and keep adding our layers required.

For this we want to use layers like

1. Input Layer

   Input layer is predetermined with fixed dimensions, so the input needs to be preprocessed. Every input is fed in to input layer as numpy array.

2. Convolution layers

   This numpy array is passed in to the convolution layer. In this layer we hypertune number of filters required. This generates the feature maps. For this we use **Conv2D** layer in keras sequential model

3. Pooling layers

   Pooling is applied after one or number of convolution layers. For this we use **Maxpolling2D** layer that uses (2,2) across the feature maps that keeps only maximum pixel value.

4. Dense layers

   This layer is connected deeply with number of neurons. This layer is used to changing the dimensions of input vector. In keras we have **Dense()** for adding dense layer

5. Dropout layers

   It is layer for regularization.We use dropout layer to avoid overfitting of input data. We have **Dropout** in keras to add dropout layer.

6. Batch Normalization Layer

   It normalizes the mini-batch of data across all observations for each channel independently. We add this layer in keras by using **BatchNormalization()** in sequential model.

7. Output layer

   In the output layer we use **softmax** activation function. As we need to classify our input into any of the 7 features. This helps us to give the probability of the class.

   Apart from this we used **Adam optimizer with hyperparameters like beta, epsilon and RMSProp optimizer, learning rate** and **sparse categorical crossentropy** loss function for compiling the model. We also hyper tuned

the model with **learning_rates**, **batch size** and number of **epochs** to fit the data properly.

Below image shows the basic architecture of neural network.

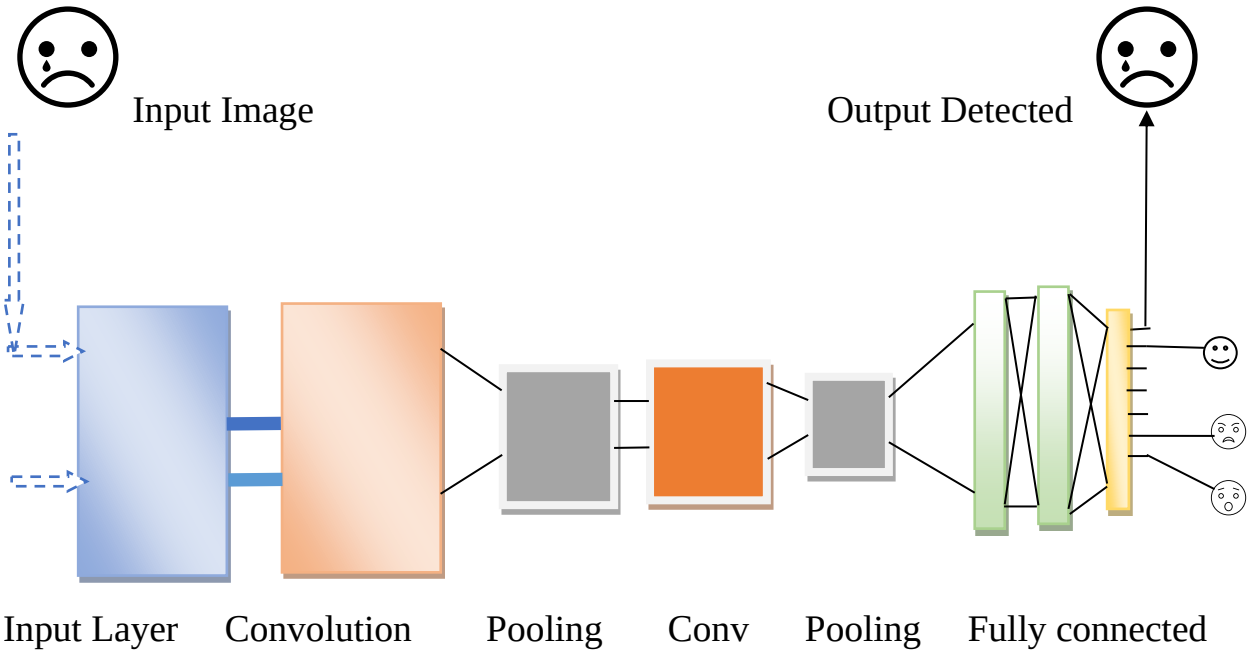


Fig 4. Block diagram of neural network design

Step 4: Evaluate/Compare models on test data and custom data:

To evaluate our model we divided training set into train and validation sets.

For evaluation of our model we used metrics like accuracy, validation accuracy, we plot confusion matrix, plot graphs between validation and training accuracies, classification report.

Confusion Matrix:

This is a performance metrics to evaluate performance of classification model on test data. It is tabular representation of predictions. For this we use metrics of confusion matrix from **sklearn** library.

Accuracy:

This is calculated from **sklearn** library with metrics of accuracy score. We also generate classification report.

Plot Graphs:

For plotting graphs we use **matplotlib.pyplot** library in python.

And finally we generate sample predictions from test data set and also custom images.

Step 5: Developing a code for video input

Google Colab environment:

For this we used pure python code and choose to run this Google colab environment to avoid conflicts with jupyter lab while running the webcam. We took this helper code for web cam from the following link https://github.com/theAIGuysCode/colab-webcam. We used some modules of this code which is helpful for our project and developed few more lines which helps us to add model for emotion recognition. We then updated this code with the developed model to predict emotions.

Jupyter lab environment:

In jupyter lab we have developed this code using **import os** libraries which is much simpler. But for final execution of our code Google colab is more flexible.

Step 6: Testing / Step 7: Generating outputs for the emotion

For testing of the model and final project execution we used several inputs of custom images, several emotion videos, test data images and tried executing the code in different environments like **jupyterlab** and **Google colab**. Used **Github** to configure entire project. We further documented most of the outputs of the emotions generated.

Implementation:

Tools: Anaconda

Languages: Python

Libraries: OpenCV, DeepFace, Tensorflow

Environment/IDE's: Google colab, JupyterLab

1. **DataSet**
2. **Preprocessing Data**
3. **Facial Feature Extraction**
4. **Developing Model for emotion recognition**
    - **Compile the model**
    - **Fit the model**
    - **Predict using the model**

Below image shows the low level implementation of our entire project.

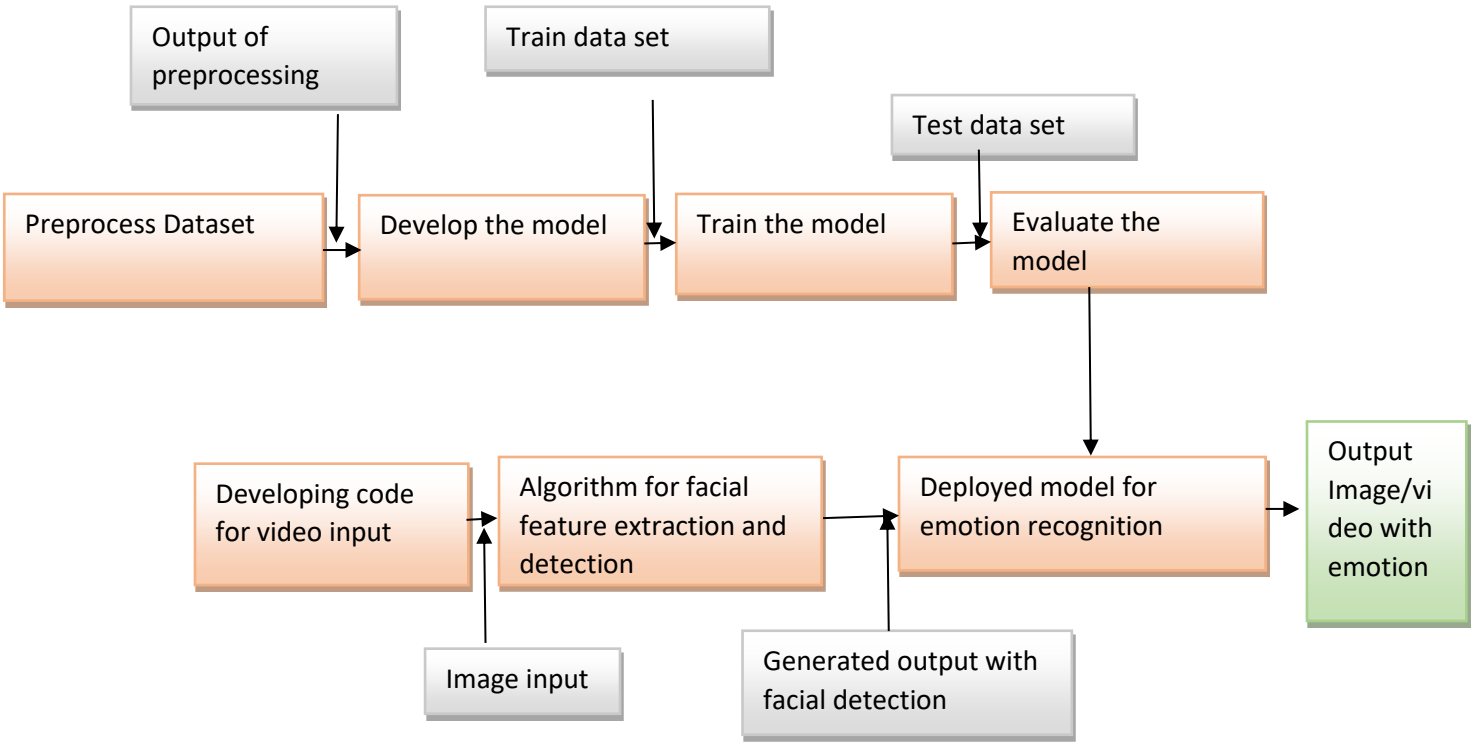


Fig. 5 Project implementation architecture

1. **DataSet:**

   The dataset we used is **FER-2013** dataset. The source we used to extract this dataset is **Kaggle**.

   https://www.kaggle.com/msambare/fer2013

   We started analysing the dataset has test and train folders and subfolders have the names of all the classes of data available. This dataset has 28,709 train images and 3,589 test images.

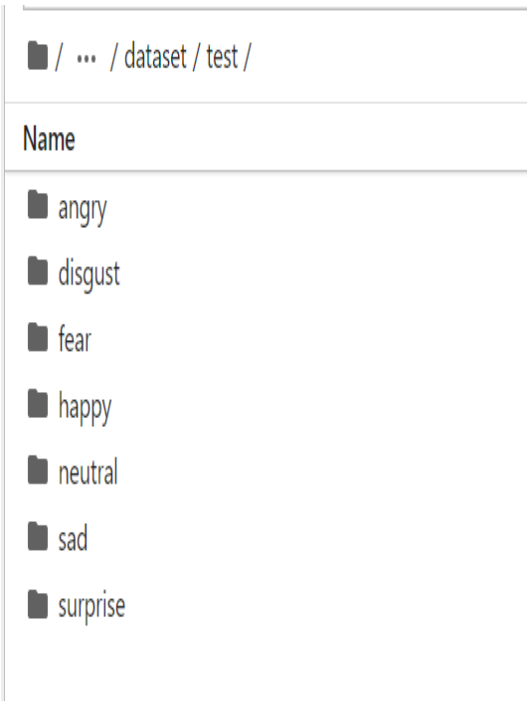


Fig. 6 Sample folders inside the dataset.

To read the images we used **open cv2** library in python, to plot the images we used **matlabplotlib** library in python, to navigate through folders we used **os** library in python.The initial size of the image we read is **48x48.**

```
]: #Reading an image from train subfolder and a image from happy directory in train subfolder.
image = cv2.imread("dataset/train/happy/Training_10019449.jpg")

#Plotting the image read
plt.imshow(image)
```

```
]: <matplotlib.image.AxesImage at 0x17a7f883520>
```

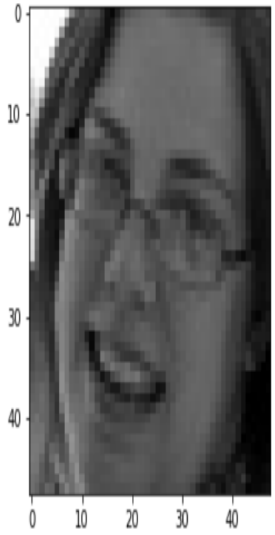


Fig.7 Sample Image read from the dataset

Now using **pandas** dataframe in python, we converted all our images in to corresponding pixel values and read this information of the image in to a dataframe. This dataframe will have columns of pixelvalue, corresponding class, datatype.

[283]:

	Emotion	PixelValue	DataType
0	angry	[46, 34, 26, 23, 12, 32, 35, 26, 26, 38, 71, 9...	PrivateTest
1	angry	[55, 33, 54, 32, 26, 21, 50, 29, 38, 45, 79, 7...	PrivateTest
2	angry	[123, 123, 126, 131, 124, 69, 109, 149, 159, 1...	PrivateTest
3	angry	[255, 255, 255, 255, 255, 255, 255, 255, 255, ...	PrivateTest
4	angry	[127, 121, 124, 137, 123, 118, 120, 111, 111, ...	PrivateTest
5	angry	[248, 243, 254, 255, 255, 252, 253, 254, 255, ...	PrivateTest
6	angry	[49, 36, 46, 53, 52, 48, 56, 40, 47, 72, 57, 7...	PrivateTest
7	angry	[225, 230, 230, 232, 216, 197, 213, 213, 208, ...	PrivateTest
8	angry	[118, 12, 17, 12, 5, 7, 11, 6, 6, 21, 59, 70, ...	PrivateTest
9	angry	[255, 255, 255, 255, 255, 255, 255, 255, 255, ...	PrivateTest

Fig. 8 Dataframe of the dataset

We generated a histogram, to help us in visualizing the data.

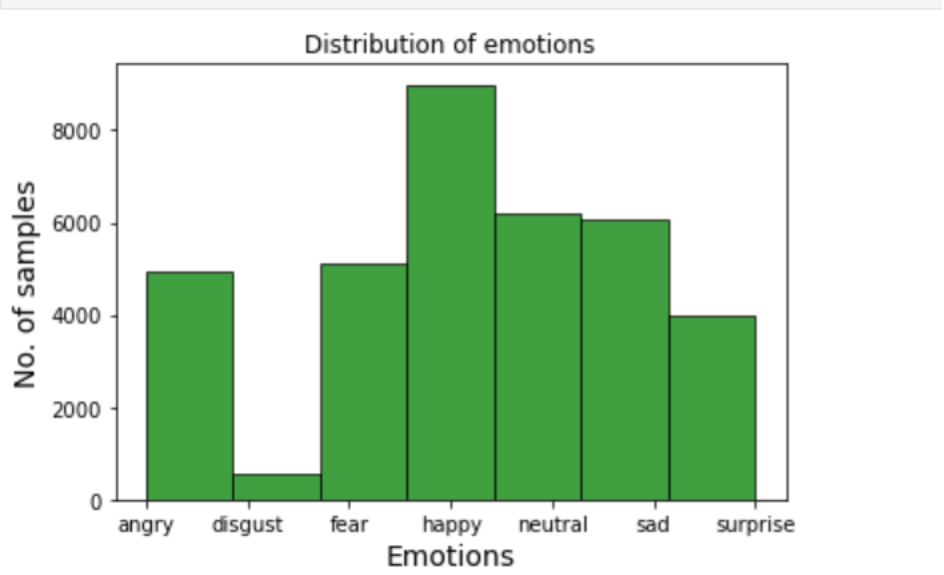


Fig. 9 Histogram of the data

This data looks like not balanced. Disgust is the emotion which is having least number of samples. Happy is the highest samples available class in this dataset.

We may have complexity while generating the output in detecting certain class of emotions.

We encoded the emotion columns to corresponding integer representation using **sklearn label encoder** model.

	Emotion	PixelValue	DataType
0	0	[46, 34, 26, 23, 12, 32, 35, 26, 26, 38, 71, 9...	PrivateTest
1	0	[55, 33, 54, 32, 26, 21, 50, 29, 38, 45, 79, 7...	PrivateTest
2	0	[123, 123, 126, 131, 124, 69, 109, 149, 159, 1...	PrivateTest
3	0	[255, 255, 255, 255, 255, 255, 255, 255, 255, ...	PrivateTest
4	0	[127, 121, 124, 137, 123, 118, 120, 111, 111, ...	PrivateTest
5	0	[248, 243, 254, 255, 255, 252, 253, 254, 255, ...	PrivateTest
6	0	[49, 36, 46, 53, 52, 48, 56, 40, 47, 72, 57, 7...	PrivateTest
7	0	[225, 230, 230, 232, 216, 197, 213, 213, 208, ...	PrivateTest
8	0	[118, 12, 17, 12, 5, 7, 11, 6, 6, 21, 59, 70, ...	PrivateTest
9	0	[255, 255, 255, 255, 255, 255, 255, 255, 255, ...	PrivateTest

Fig. 10 Dataframe after encoding

We then exported all our dataframes of test, train and whole dataset in to csv files. We use this csv file for further development of our project. The details of dataset experiment are given in “Dataset.ipynb file”.

```
[293]: df_train.to_csv('dataset/train.csv')

[294]: df_test.to_csv('dataset/test.csv')

[295]: df_encoded.to_csv('dataset/datasetFER_2013.csv')
```

Fig. 11 Dataset csv files

Output of sample images after reading array pixel values from the csv file

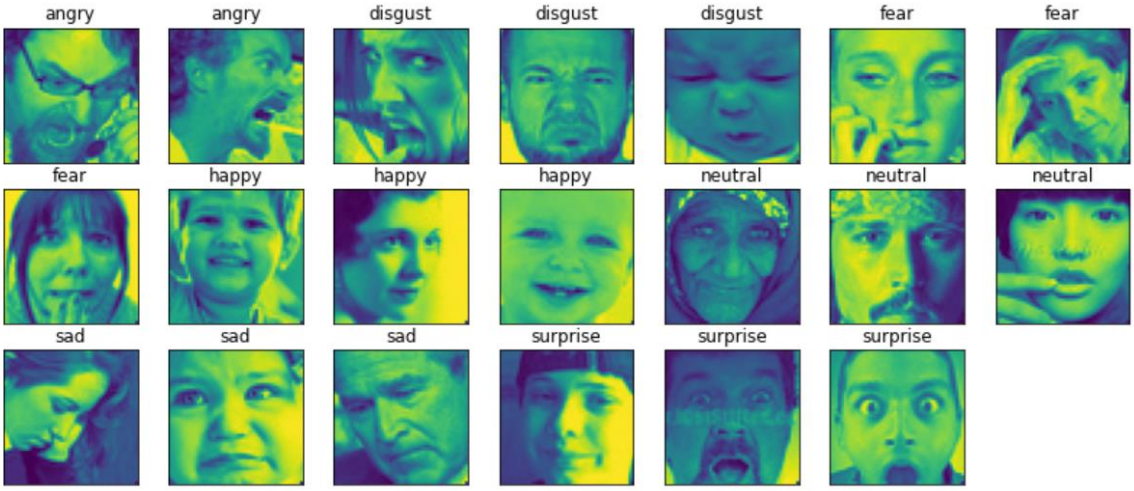


Fig. 13 Sample images from csv file output

2. Preprocessing:

As our model trained with images of size 48x48. All the inputs should be converted to 48x48, so that input is compatible with the models developed. So all the photos/ videos which we input should be preprocessed in order to fed to out model.

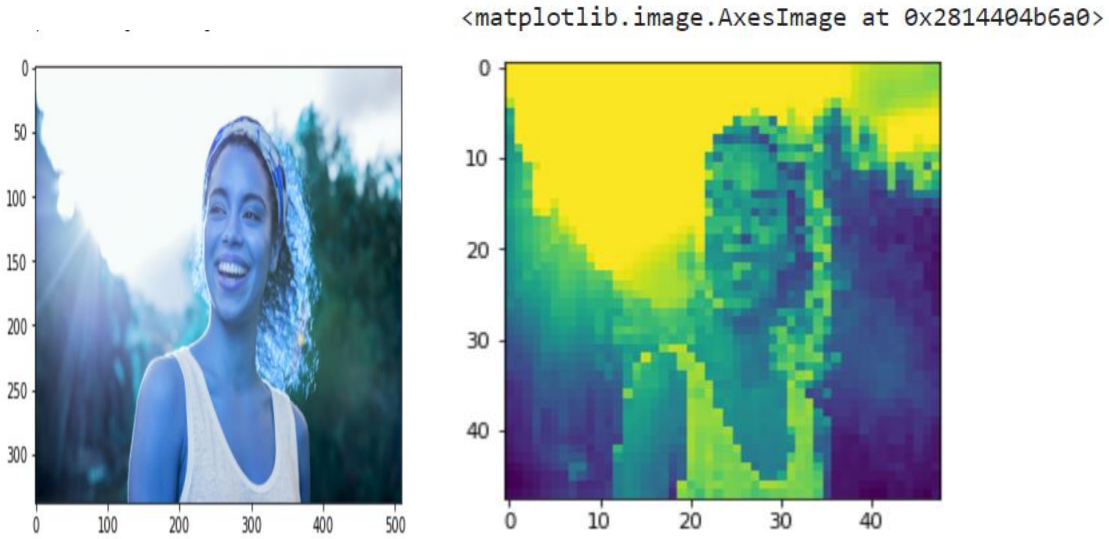


Fig. 12 Sample of original image vs resized image

3. Facial Feature Extraction

Step 1:

Reading the input image and convert it into gray scale image.

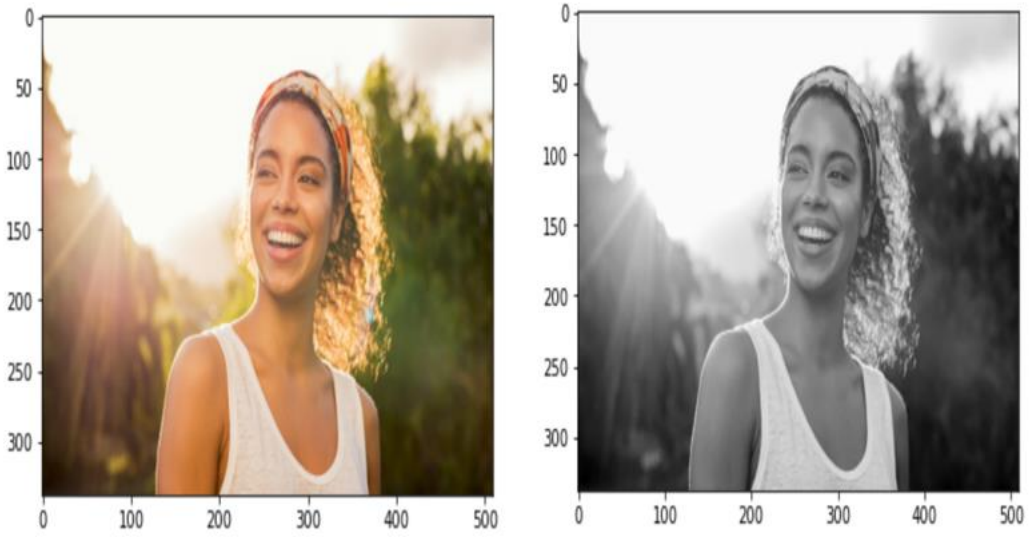


Fig. 13 Input image vs grayscale image

Step:3

Import **cv2.CascadeClassifier**. For this we also need haarcascade_frontalface_alt.xml file in the workspace. Input the gray scale image to the classifier and draw a bounding box around the faces detected. This is how the output of our classifier looks like.

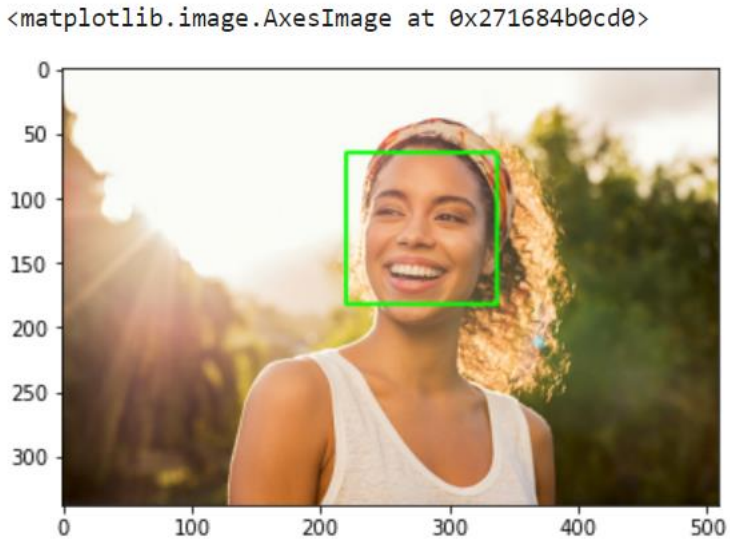


Fig. 14 output of cascade classifier

Same way we can detect multiple faces in an image with it.

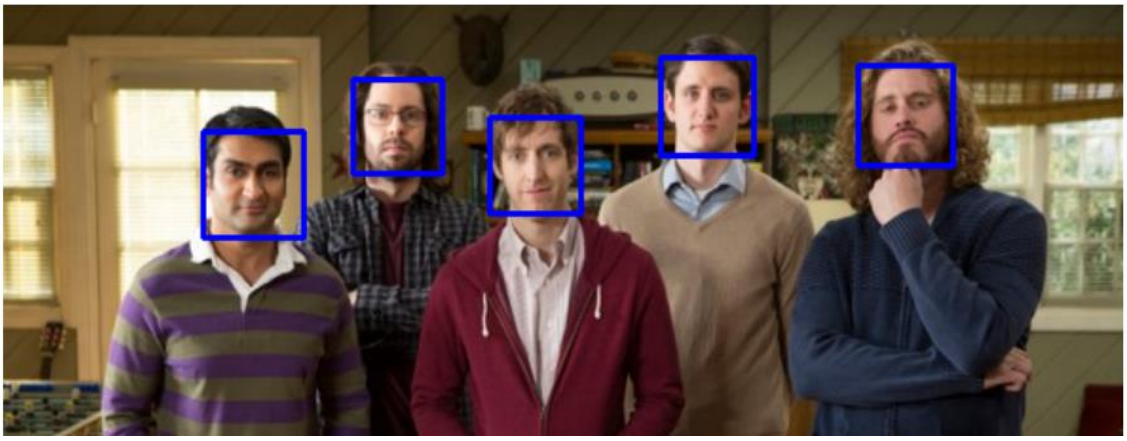


Fig. 15 Detecting multiple faces

So before developing our neural network, we used DeepFace pre trained model available to detect the sample emotions of the features extracted from the classifier. So, to make sure our facial feature extracting model is working. Below is the output generated by **DeepFace** model with predictions for the extracted facial feature. It is predicting the face extracted as 100% happy.

Fig. 16 Predicted face from DeepFace model

We have successfully extracted facial features necessary for our emotion recognition model.

4. **Developing Model for emotion recognition:**

   We used deep neural network with multiple layers. For this we used keras library which has sequential model.

   tf.keras.layers.Conv2D,

   tf.keras.layers.MaxPool2D,

   tf.keras.layers.BatchNormalization,

   tf.keras.layers.Dropout

   tf.keras.layers.flatten

   tf.keras.layers.Dense

The model was built and trained on a convolutional neural network. Our architecture consists of four Conv 2D layers, each of the layer is followed by Pooling layer with (2,2) kernel, batch normalization, relu layer, droput layer. Conv 2D layer starts with 32 filters of size (3,3) each. Input shape of the model is (46,46,1). The first convolution layer consists of 32 filters with (3,3) kernels, relu

unit. The second convolution has 64 filters with (3,3) kernels and third has 128 filters. Finally, the fourth layer consists of 256 filters. Then we have a one dense layer and one flatten layer, where the dense layer has 128 neurons and a relu unit. Dense layer will be followed by Dropout layer with a value of 0.5. Finally output of the model is fed to softmax layer that assigns the probabilities for the classes. For training our model we used 90% of the entire data to train and 10% of the data to test.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 23, 23, 32)        128
_________________________________________________________________
dropout (Dropout)            (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 10, 10, 64)        256
_________________________________________________________________
dropout_1 (Dropout)          (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 128)         512
_________________________________________________________________
```

Fig. 17 Model architecture

Compile and fit the model:

The model was compiled using RMSProp optimizer with learning rate of 0.0001, sparse categorical cross entropy loss function. We used callback and model checkpoints to save the model. We created learning curves for loss and accuracy of training and validation data. Trained the model by using fit method, where we used 100 epochs and batch_size as 32. The model generated an accuracy of 63% which is good accuracy for dataset like FER-2013.

```
[55]: #Predicting the accuracy of the model

      print("Accuracy for the model3",metrics.accuracy_score(y_test,Pred_Emo))

      Accuracy for the model3 0.6260796879353581
```

Fig. 18 Accuracy of trained model

Prediction from the model:

The below screenshot shows our model predicting on our test set, all these predictions are read into y_pred set. Looking at the first prediction array shows that predicted emotion from the test set matches the original emotion in the test set.

```
[56]: loaded_model = load_model('checkpoint/Model3.h5')

[57]: y_pred = loaded_model.predict(X_test)

[64]: y_pred[0]

[64]: array([0.10357963, 0.00287146, 0.22825462, 0.00709566, 0.24733329,
             0.14566442, 0.265201  ], dtype=float32)

[68]: np.argmax(y_pred[1])

[68]: 4

[70]: y_test[1]

[70]: 4
```

Fig. 16 Predictions from model

Experimental results and Discussions:

1. **Evaluate model**
2. **Validation on test data set**
3. **Emotion recognition using images**
4. **Emotion recognition using videos**

1. **Evaluate model:**

   The following figure shows the accuracy and loss for training and validation with the epochs. The validation accuracy looks almost near to testing accuracy up to 100 epochs.

   Learning Curves:

```
[66]: # summarize history for accuracy
      plt.plot(history.history['accuracy'])
      plt.plot(history.history['val_accuracy'])
      plt.title('model accuracy')
      plt.ylabel('accuracy')
      plt.xlabel('epoch')
      plt.legend(['train', 'valid'], loc='upper left')
      plt.show()
```

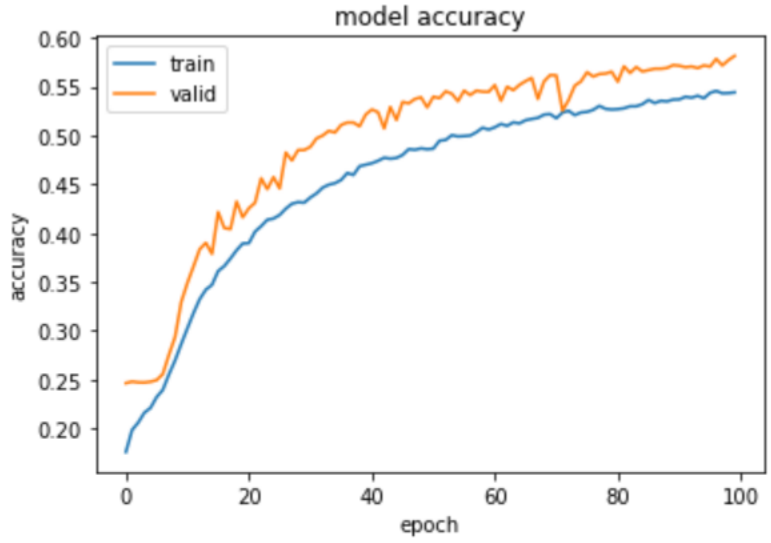


Fig. 18 Accuracy learning curve

```
[67]: # summarize history for loss
      plt.plot(history.history['loss'])
      plt.plot(history.history['val_loss'])
      plt.title('model loss')
      plt.ylabel('loss')
      plt.xlabel('epoch')
      plt.legend(['train', 'valid'], loc='upper left')
      plt.show()
```

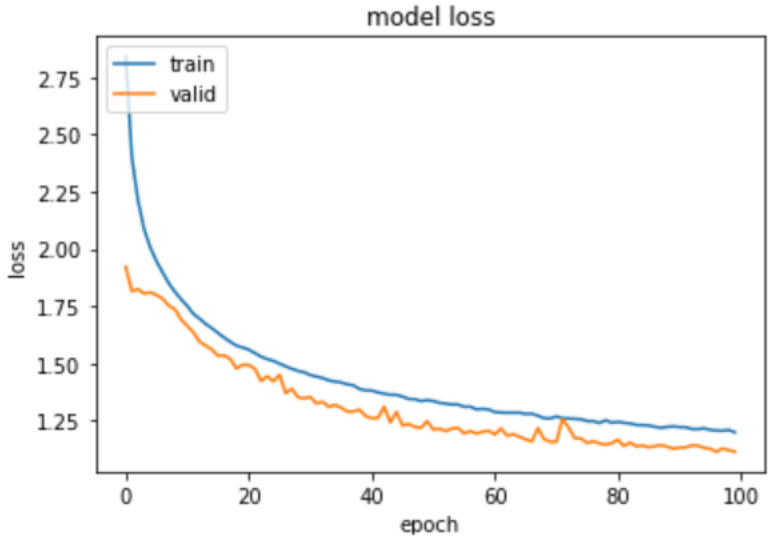


Fig. 19 Loss learning curve

Classification Report:

The classification report shows precision, F1-score, recall and support for each emotion from 0 to 6 which are annotated as “anger”, “disgust”, “fear”, “happy”, “neutral”, “sad”, “surprise” respectively. According to the report emotion “happy” has highest precision and f-1 score which is above 80%. All emotions are having different support values, but “disgust” has the least number of support value. The accuracy on the test set is 63% and weighted average ranges from 57-62% which is almost same for precision, recall and F-1 score. “Fear” even though it has 512 support values its recall and F-1 scores are very low. Because of which it returns very few results. This can be seen in the confusion matrix of “fear” which returns few results.

[51]:
```
print("Classification report for trained model3\n\n")
print(classification_report(y_test,Pred_Emo))
```

```
Classification report for trained model3


              precision    recall  f1-score   support

           0       0.55      0.54      0.55       476
           1       0.57      0.40      0.47        72
           2       0.64      0.15      0.24       512
           3       0.82      0.85      0.84       882
           4       0.56      0.69      0.62       605
           5       0.48      0.64      0.54       660
           6       0.71      0.78      0.74       382

    accuracy                           0.63      3589
   macro avg       0.62      0.58      0.57      3589
weighted avg       0.64      0.63      0.61      3589
```

Confusion Matrix:

The below figure shows the confusion matrix of our model. The confusion matrix shows “disgust” which is represented by ‘1’. The number of samples used to predict are very low because of the imbalance in the dataset. This is because of the small set of images disgust class in our dataset. According to this matrix we can observe 3 which is “happy” is predicted with highest number of true positives 754 times. That is “happy” is predicted well in our model. “Neutral” and “Sad” (4,5 respectively) also have good true predictions that is 416 times and 420 times respectively. Again, “Surprise” and “angry” which are 6 and 1 respectively are predicted very a smaller number of times compared to happy, neutral, and sad. This is because of the smaller number of images the dataset is having. Overall, the dominating set of emotions are ‘Happy”, “Neutral” and “Sad” and next follows “angry” and “surprise” which are not no dominating. The least emotions classified in the predictions is “disgust” and also “fear”.

Fig. 20 Confusion matrix

2. Validation on test data set:

We used our model to predict the test set images. Below are the sample predictions made by the model. As Happy and Surprise are dominant emotions they are predicted with good percentage of prediction. Fear which is predicted as anger. Because of the less number of samples of fear and anger, looks like the predictions are confused.

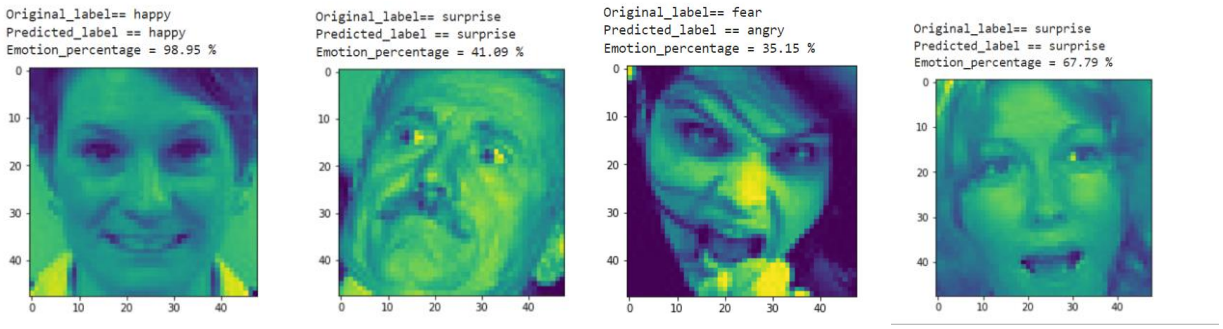


Fig. Emotions predicted on test data

3. Emotion recognition using real time images:

Finally, our model is ready to test on real time data of images. We also developed a code which inputs video from web cam to the model. Below are the sample predictions on the real time images. We need to preprocess the data and reshape

each image in to (48,48,1) to fit in to our model. We use classifier for detecting faces from the image. These features are then fed to the trained model and the final predictions are generated. In the below predictions our classifier is detecting facial feature and model is predicting the emotion. Below is the set of images where the prediction for happy is giving good predicted percentage. Neutral is detected as neutral, though with less amount of prediction, but the other emotions should have more lesser prediction percentage. Though the confidence level of this prediction class is low, it predicted correctly. For surprise it predicted as neutral. This may be because of the smaller number of images in the surprise class from our dataset. So, it is naturally dominated by the dominant emotion available. As happy is dominant emotion of images in our dataset its prediction is higher.

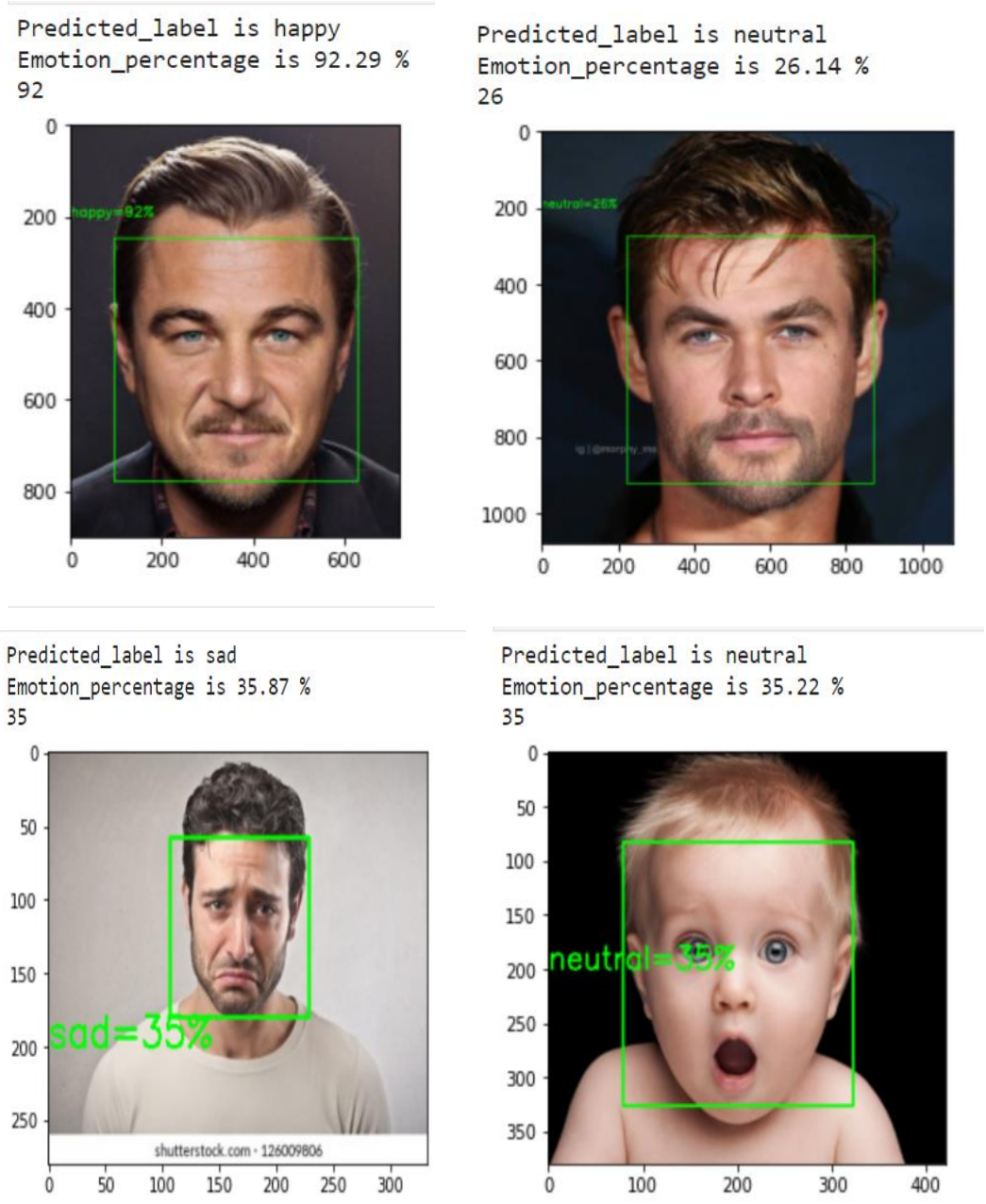


Fig. 22 Emotions prediction on real time data

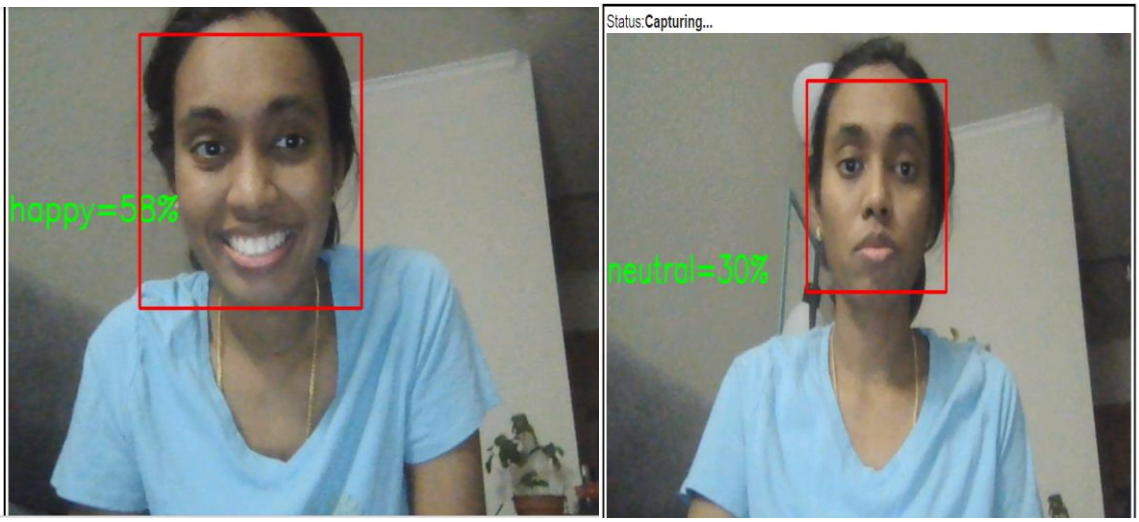
4. Emotion recognition using videos:

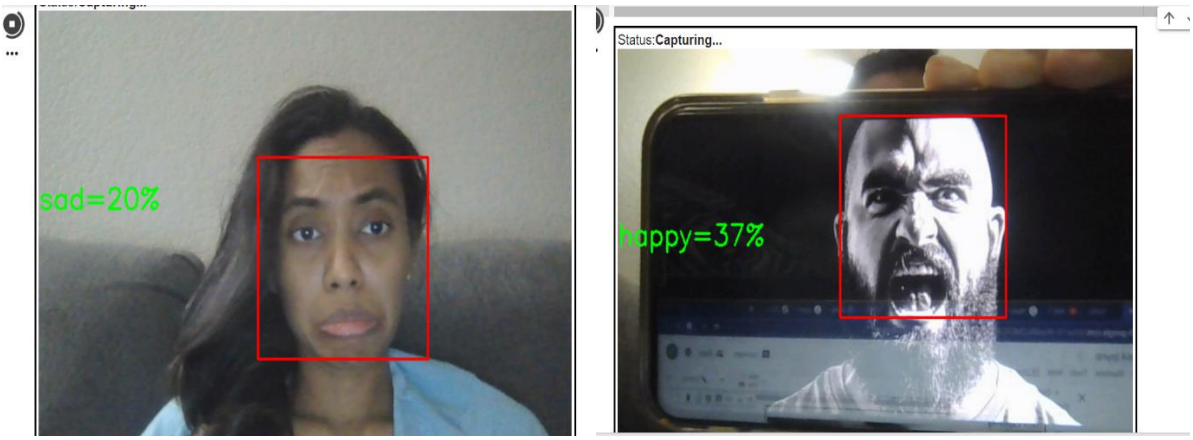
So, we used openCv for video input in jupyter lab, we took the helpin code for giving video input in google colab. Because of the flexibility we choose to use

Colab for this part of execution rather than Jupyter lab. So, for colab, we need to use a helping code for inputting video from the webcam. This is taken from the https://github.com/theAIGuysCode/colab-webcam reference.

We took the necessary modules for our project from this code to use webcam and updated this with using our trained model to detect the faces in the video and detect the emotion.

Below are the screenshots from our working model to recognize emotions from the webcam. We also tried giving input some good expressive photos from mobile. But except happy, neutral and sad emotions other emotions are difficult to predict by the model, this is because of the dataset problem like imbalance between the classes. The emotion angry is detected as happy. From the distribution of the classes in the dataset and confusion matrix the number images of angry class for this model are less and happy has the dominant level of images, because of which it is predicting as happy instead of angry.





Conclusion:

- We used FER-2013 dataset for training and testing our model.
- Every input is preprocessed into the size that can be fed in to our trained model.
- We used haarcascade facial detection algorithm to extract facial key features.
- We developed a deep neural network to predict emotions from the video. As of now the scope of this project is limited to detecting faces from the image/video and analyzing the emotion and classifying it into one of the anger, disgust, fear, happy, neutral, sad, surprise classes.
- We used keras library and helping libraries in python for our CNN architecture.
- From the confusion matrix the least accurate prediction is “disgust” and most accurate prediction is “Happy”.
- OpenCv is used to capture video from the webcam and detect the faces.
- We then integrated our training model to detect faces and recognize the emotion.
- We analyzed the model and evaluated it on test data and real-time data.

- Given the time we extended our efforts in training and testing different models developed with different hyper parameters, epochs and predicting the results.
- We developed models with RMSprop optimizer and Adam optimizer, where the model with hyper parameters of Adam optimizer gave only 51% of accuracy. The model with RMSprop optimizer gave accuracy of 63%.

References:

- https://ieeexplore.ieee.org/document/990517
- https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08
- https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148