



PROJECT REPORT

FACE RECOGNITION OF EMOTIONS

ADVANCED SOFTWARE ENGINEERING PROJECT
CS 5394
SUMMER II 2020

BY GROUP 5394_1

PUNITHA VALLI DEVARAJ - P_D120
SONALI BANTE – S_B611
RISHITA SHARMA- R_S607

TABLE OF CONTENTS

1. PROJECT DESCRIPTION	3
2. UML CLASS DIAGRAM	4
3. SYSTEM STATE CHART DIAGRAM	5
4. DESCRIPTION OF EXECUTION OF ACCEPTANCE TEST CASES	6
5. ALGORITHMS AND TECHNIQUES	20
5.1 MINI BATCH GRADIENT DESCENT ALGORITHM	20
5.2 DATA AUGUMENTATION TECHNIQUE	21
5.3 CNN MODEL	22

1.PROJECT DESCRIPTION:

Facial Emotions are one of the most important aspects of human communication. Facial expression recognition (FER) provides machines a way of sensing emotions that can be considered one of the mostly used Artificial Intelligence and pattern analysis applications. One of the relatively new and promising trends in using facial expressions to classify learners' emotions is the development and use of software programs that automate the process of coding using advanced machine learning technologies.

There are many different types of emotions that have an influence on how we live and interact with others. Among those, the 7 important emotions are Happiness, Sadness, Fear, Disgust, Anger, Surprise and Neutral.

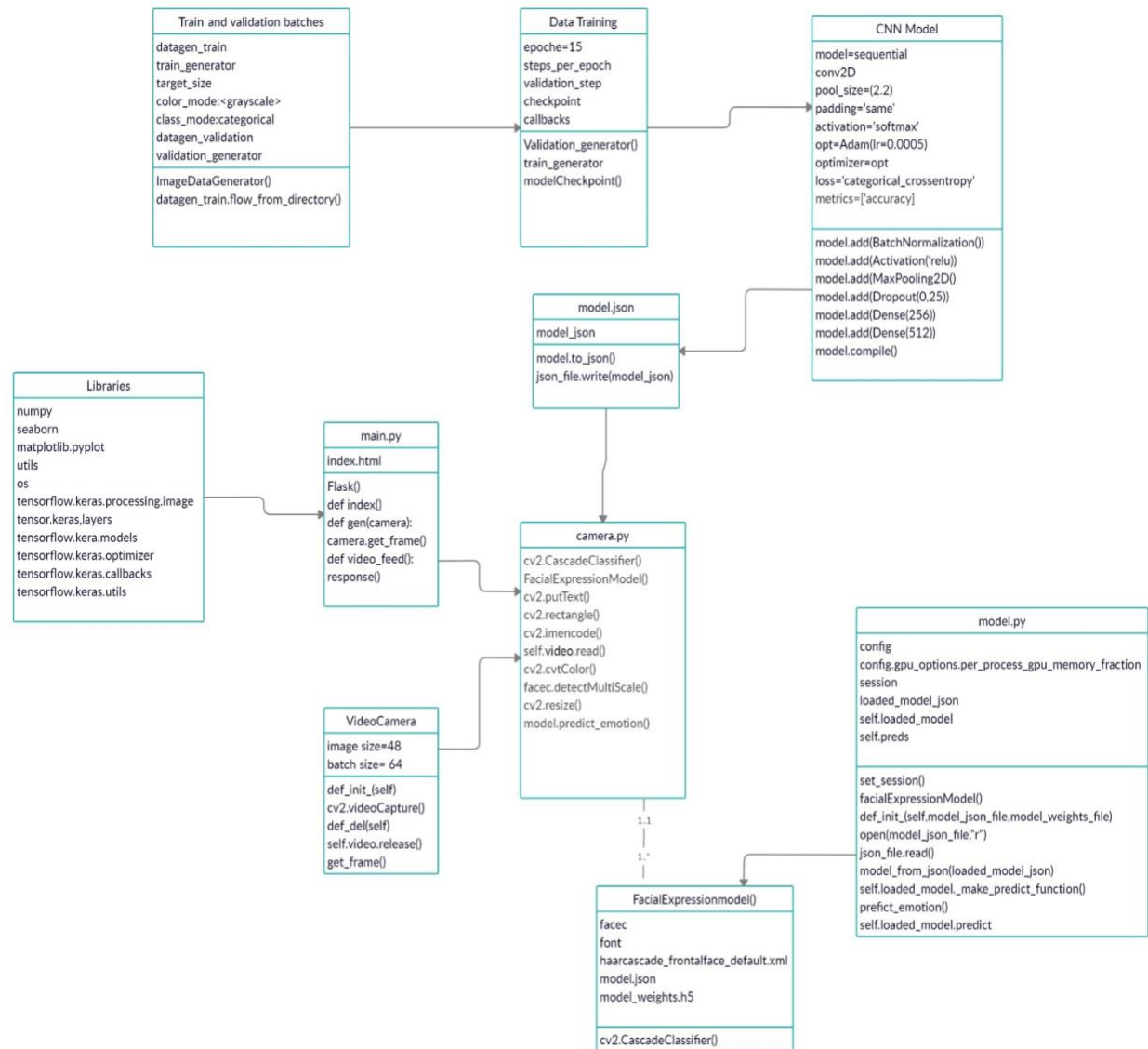
The aim of the project is to build efficient face recognition with keras by Deep Learning. We train and design a deep Convolutional Neural Networks (CNN) using tf.keras which is TensorFlow's high level API for building and training deep learning models.CNN is primarily used because, it is one of the popular Deep Artificial Neural Networks which are majorly used in image recognition, image clustering and classification, object detection.Face detection part is done using OpenCV(Open Source Computer Vision) where the face detection classifiers automatically detects faces and draws boundary boxes around them to analyze.We train the network using the dataset from Kaggle – facial expression dataset. Once the model is trained and saved, then it is deployed with a web interface using Flask to make predictions for inference and make it functional using Web camera - through live detection of images and sensing emotions. Also, it can be used to detect Facial emotions in a video.

PLATFORM: ANACONDA NAVIGATOR/GOOGLE COLAB

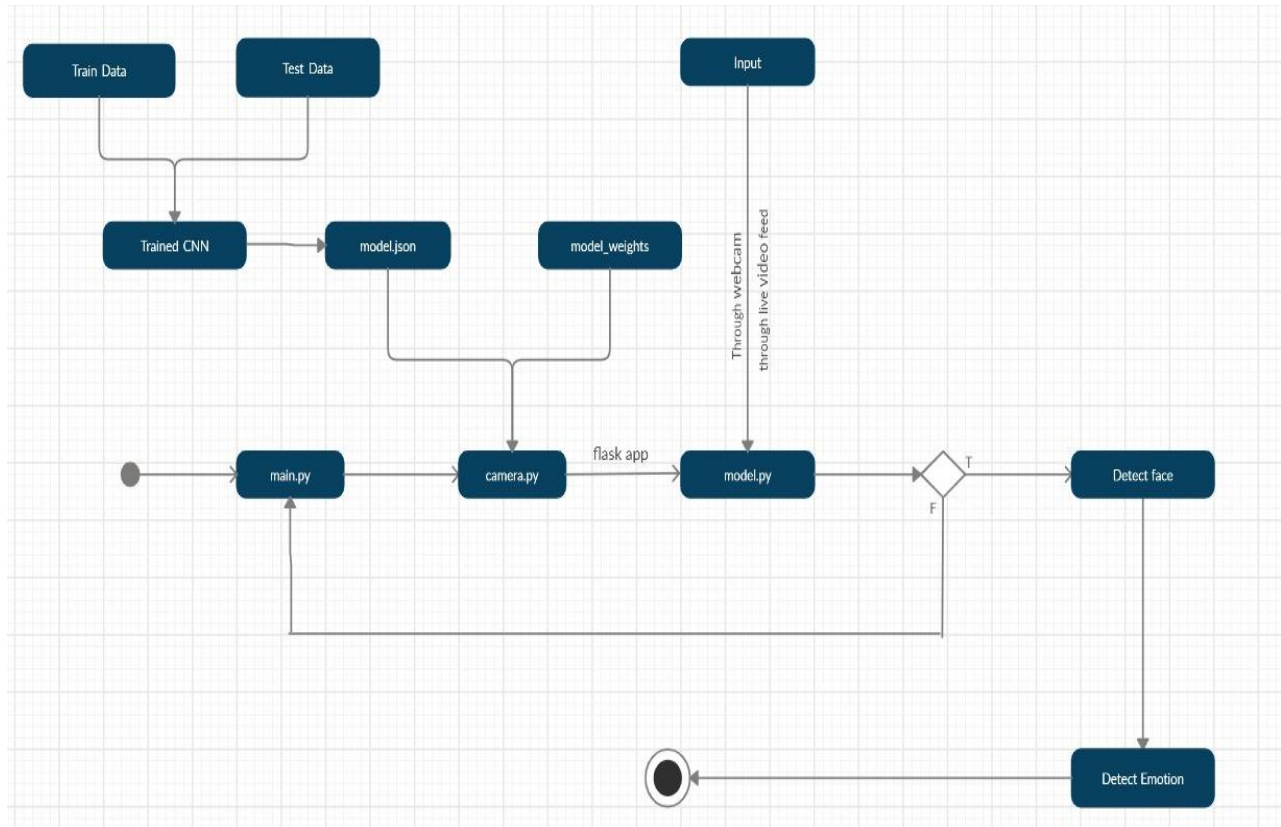
IMPORT LIBRARIES: TENSORFLOW, KERAS, OPENCV, FLASK, MATPLOTLIB, LIVELOSSPLOT, NUMPY, PANDAS,SEABORN

API: TensorFlow's High level API – tf.keras

2.UML CLASS DIAGRAM:

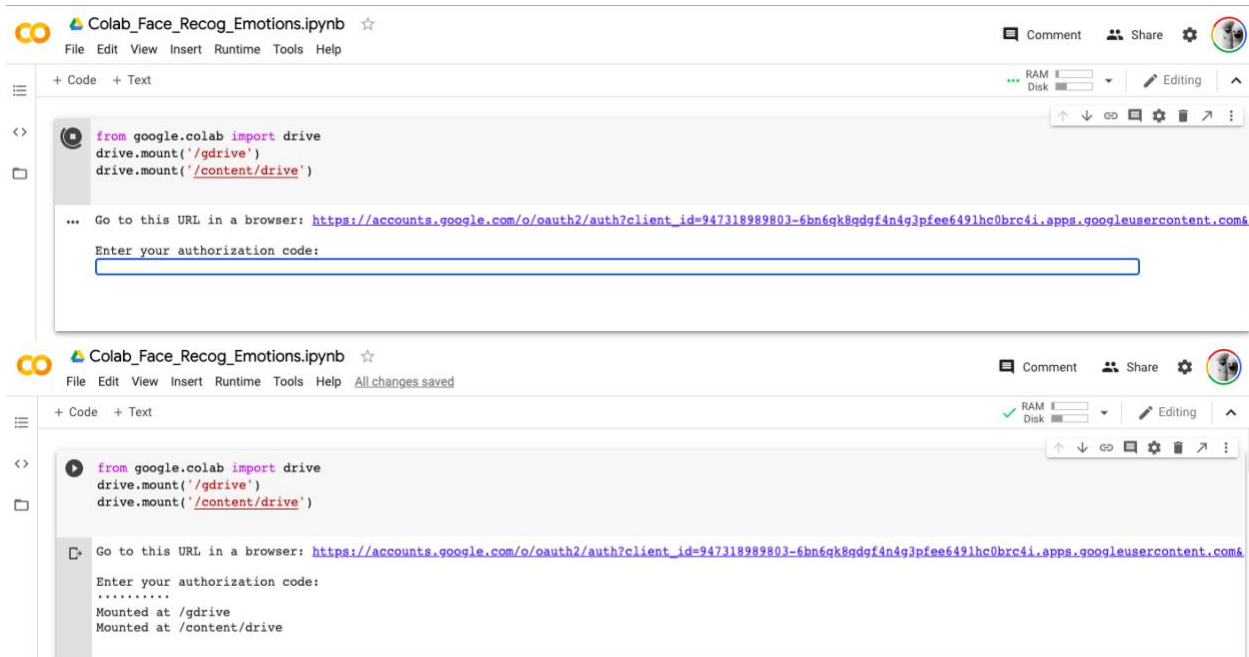


3. SYSTEM STATE CHART DIAGRAM:



4. DESCRIPTION OF EXECUTION OF ACCEPTANCE TESTCASES:

TEST CASE 4.1: IMPORTING GOOGLE DRIVE:



```
from google.colab import drive
drive.mount('/gdrive')
drive.mount('/content/drive')
```

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&...

Enter your authorization code:

```
from google.colab import drive
drive.mount('/gdrive')
drive.mount('/content/drive')
```

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&...

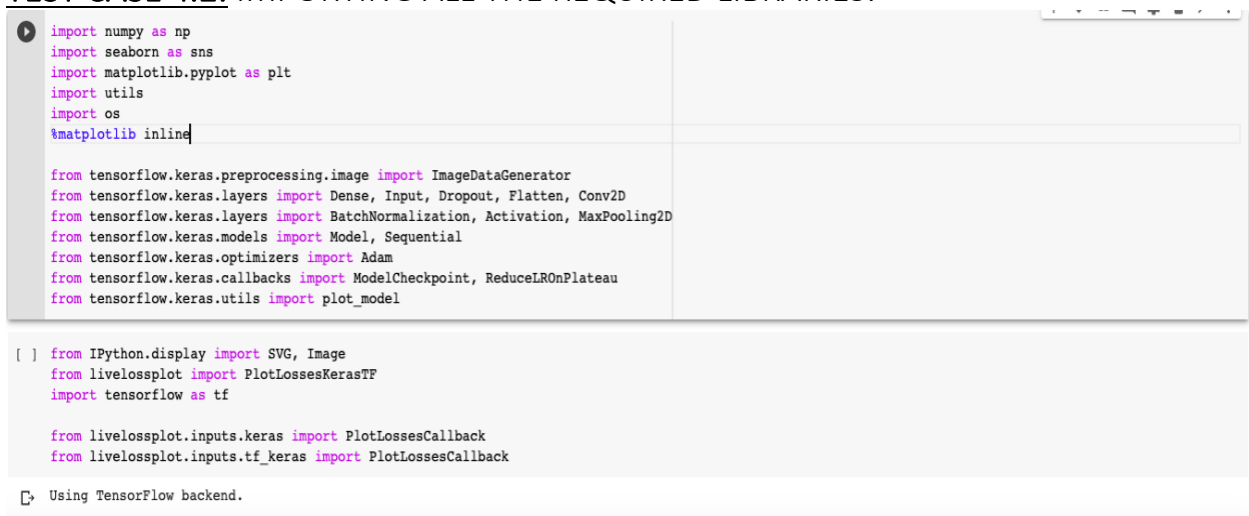
Enter your authorization code:

Mounted at /gdrive

Mounted at /content/drive

RESULT: Google drive is successfully mounted and the platform is set for the project.

TEST CASE 4.2: IMPORTING ALL THE REQUIRED LIBRARIES:



```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import utils
import os
%matplotlib inline

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Input, Dropout, Flatten, Conv2D
from tensorflow.keras.layers import BatchNormalization, Activation, MaxPooling2D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.utils import plot_model

[ ] from IPython.display import SVG, Image
from livelossplot import PlotLossesKerasTF
import tensorflow as tf

from livelossplot.inputs.keras import PlotLossesCallback
from livelossplot.inputs.tf_keras import PlotLossesCallback
```

Using TensorFlow backend.

RESULT: All the required libraries are imported and the Tensorflow is running at the backend mounting the Keras on top.

TEST CASE 4.3: TENSORFLOW VERSION CHECK

```
print("Tensorflow version: ",tf.__version__)
```

```
Tensorflow version: 2.3.0
```

RESULT: Tensorflow version is atleast above 2.0.0

TEST CASE 4.4: GPU AVAILABILITY TO TRAIN THE MODEL

```
print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))
```

```
Num GPUs Available: 1
```

RESULT: GPU/TPU is available for this project in order to Train the Model. Here Google Colab's GPU Cloud is accessed.

TEST CASE 4.5: SAMPLE IMAGES FROM KAGGLE IMPORTED TO OUR PLATFORM



RESULT: Sample Images from 7 emotion categories are displayed successfully from the Kaggle Dataset

TEST CASE 4.6: IMPORTING TEST AND TRAIN DATA IMAGES INTO PLATFORM

```
+ Code + Text RAM Disk Editing
# for training folder
for expression in os.listdir("/content/drive/My Drive/Face Recog Emotions/train/"):
    print(str(len(os.listdir("/content/drive/My Drive/Face Recog Emotions/train/" + expression))) + " " + expression + " images")

7234 happy images
4965 neutral images
3171 surprise images
4850 sad images
4097 fear images
436 disgust images
3995 angry images

[18] # for testing folder.
for expression in os.listdir("/content/drive/My Drive/Face Recog Emotions/test/"):
    print(str(len(os.listdir("/content/drive/My Drive/Face Recog Emotions/test/" + expression))) + " " + expression + " images")

1247 sad images
831 surprise images
1774 happy images
1024 fear images
958 angry images
1233 neutral images
111 disgust images
```

RESULT: Successfully imported all the train and test data into the platform. It also categorizes how many images are there under each emotion.

TEST CASE 4.7: CREATING CNN MODEL WITH 4 CONVOLUTIONAL LAYERS AND 2 FULLY CONNECTED LAYERS

```
+ Code + Text Reconnect Editing
# Using sequential model
model = Sequential() # initialize CNN

# Creating 1st convolution
model.add(Conv2D(64, (3,3), padding='same', input_shape=(48,48,1)))

model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Creating 2nd convolution
model.add(Conv2D(128, (5,5), padding='same')) # increasing the number of filters
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Creating 3rd convolution
model.add(Conv2D(512, (3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```


+ Code + Text

Reconnect ▾

Editing



Creating 4th convolution

```
model.add(Conv2D(512, (3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```

Now we are flattening the volume, so we can pass it to through our fully connected layers

```
model.add(Flatten())
```

Creating fully connected 1st layer

```
model.add(Dense(256)) # creating dense layer with 256 nodes
# adding batch normalization, relu activation and dropout
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
```

Creating fully connected 2nd layer

```
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
```

Creating output thereby adding last dense layer

```
model.add(Dense(7, activation='softmax'))
```

```
opt = Adam(lr=0.0005)
```

Compiling the model now.

```
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

to see summary

```
model.summary()
```

Creation Model Summary Output

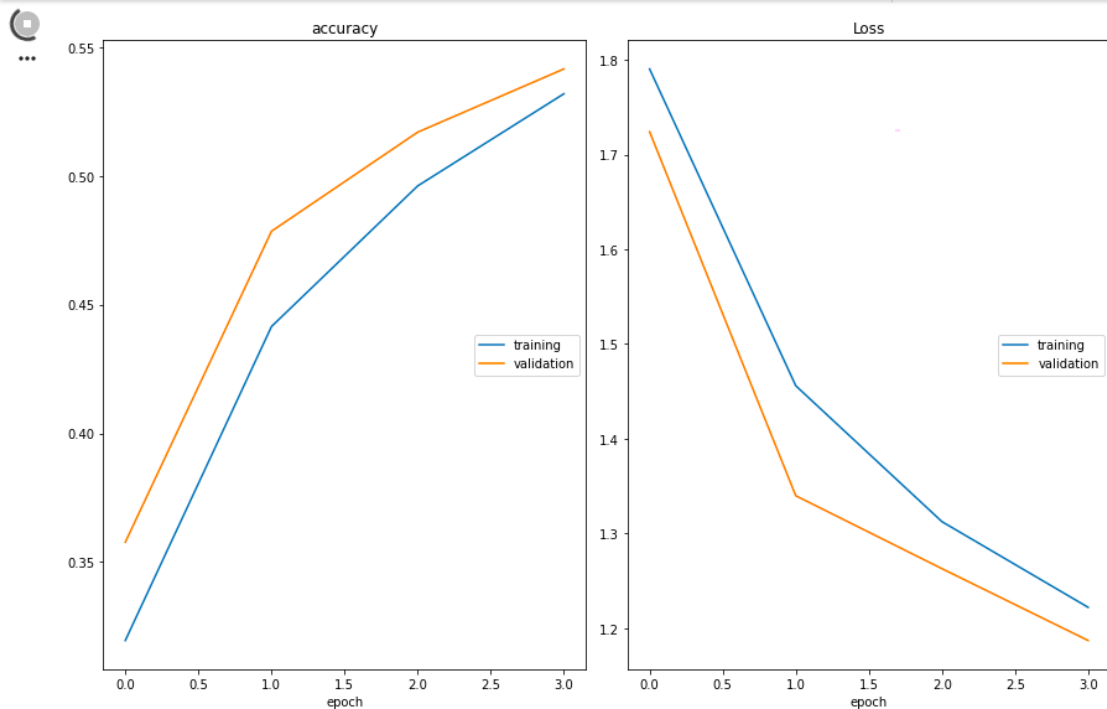
+ Code	+ Text	Reconnect	Editing	^
Model: "sequential"				
Layer (type)	Output Shape	Param #		
conv2d (Conv2D)	(None, 48, 48, 64)	640		
batch_normalization (Batch Normalization)	(None, 48, 48, 64)	256		
activation (Activation)	(None, 48, 48, 64)	0		
max_pooling2d (MaxPooling2D)	(None, 24, 24, 64)	0		
dropout (Dropout)	(None, 24, 24, 64)	0		
conv2d_1 (Conv2D)	(None, 24, 24, 128)	204928		
batch_normalization_1 (Batch Normalization)	(None, 24, 24, 128)	512		
activation_1 (Activation)	(None, 24, 24, 128)	0		
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 128)	0		
dropout_1 (Dropout)	(None, 12, 12, 128)	0		
conv2d_2 (Conv2D)	(None, 12, 12, 512)	590336		
batch_normalization_2 (Batch Normalization)	(None, 12, 12, 512)	2048		
activation_2 (Activation)	(None, 12, 12, 512)	0		
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 512)	0		
dropout_2 (Dropout)	(None, 6, 6, 512)	0		
conv2d_3 (Conv2D)	(None, 6, 6, 512)	2359808		
batch_normalization_3 (Batch Normalization)	(None, 6, 6, 512)	2048		
activation_3 (Activation)	(None, 6, 6, 512)	0		
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 512)	0		
dropout_3 (Dropout)	(None, 3, 3, 512)	0		
flatten (Flatten)	(None, 4608)	0		
dense (Dense)	(None, 256)	1179904		
batch_normalization_4 (Batch Normalization)	(None, 256)	1024		
activation_4 (Activation)	(None, 256)	0		
dropout_4 (Dropout)	(None, 256)	0		
dense_1 (Dense)	(None, 512)	131584		
batch_normalization_5 (Batch Normalization)	(None, 512)	2048		
activation_5 (Activation)	(None, 512)	0		
dropout_5 (Dropout)	(None, 512)	0		
dense_2 (Dense)	(None, 7)	3591		
Total params: 4,478,727				
Trainable params: 4,474,759				
Non-trainable params: 3,968				

RESULT: A Sequential CNN model is created by applying batch normalization, RELU(Non Linearity), Max pooling and Dropout Regularization. CNN Summary is generated. After each conv block, the volume(height*width) is reduced (by a factor of 2) and the number of channels are doubled. After the 4 conv blocks, we flatten the output and its passed to the 2 fully connected layers. Lastly, we use dense

layer, with Softmax activation functions to predict the output label which can take on one of 7 classes.

TEST CASE 4.8: TRAINING THE MODEL. EPOCH = 15

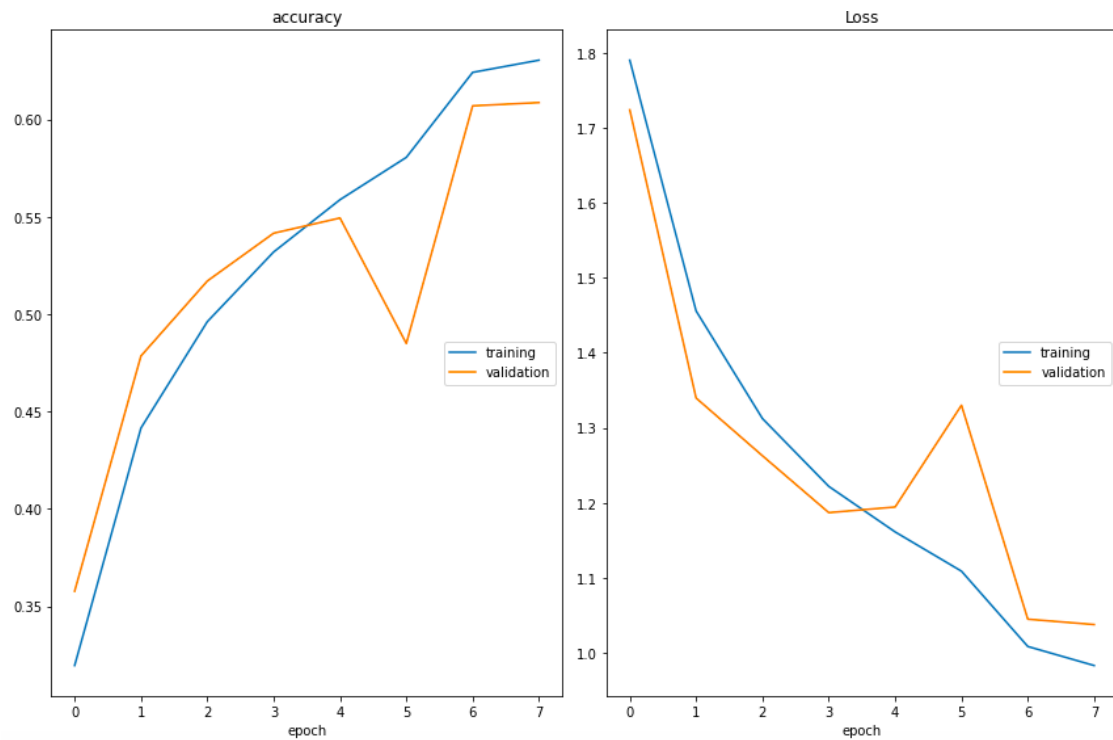
EPOCH = 5/15



```
accuracy
epoch
training (min: 0.320, max: 0.532, cur: 0.532)
validation (min: 0.358, max: 0.542, cur: 0.542)
Loss
epoch
training (min: 1.222, max: 1.790, cur: 1.222)
validation (min: 1.187, max: 1.724, cur: 1.187)

Epoch 00004: saving model to model_weights.h5
449/449 [=====] - 1450s 3s/step - loss: 1.2219 - accuracy: 0.5319 - val_loss: 1.1869 - val_accuracy: 0.5416 - lr: 5.0000e-04
Epoch 5/15
10/449 [.....] - ETA: 19:48 - loss: 1.1810 - accuracy: 0.5797
```

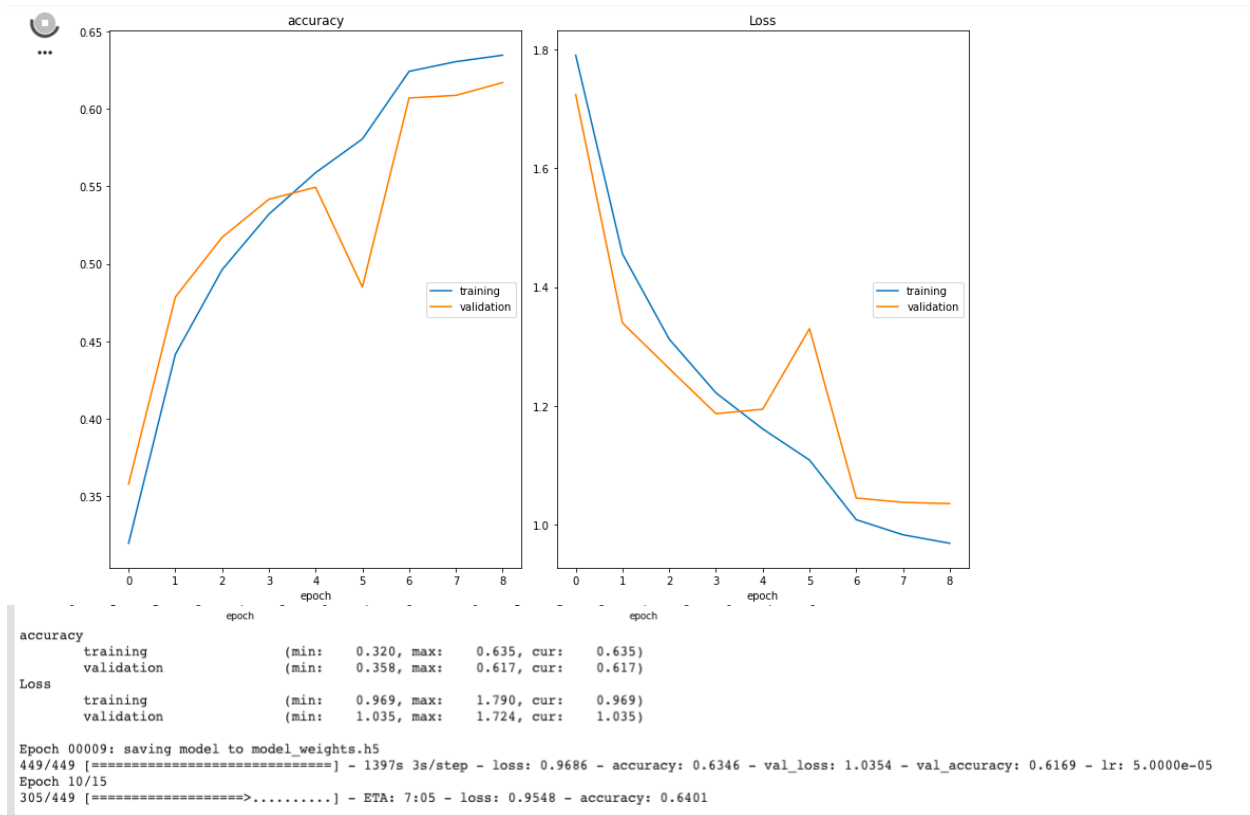
EPOCH = 9/15



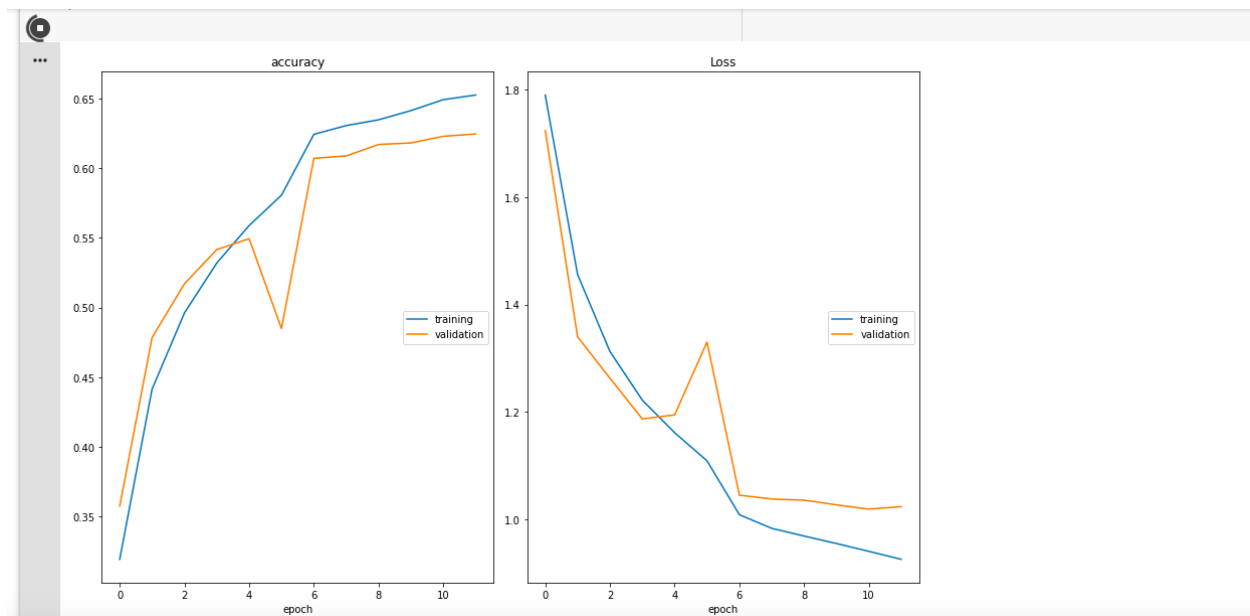
```
accuracy
  training (min: 0.320, max: 0.630, cur: 0.630)
  validation (min: 0.358, max: 0.609, cur: 0.609)
loss
  training (min: 0.983, max: 1.790, cur: 0.983)
  validation (min: 1.038, max: 1.724, cur: 1.038)
```

```
Epoch 00008: saving model to model_weights.h5
449/449 [=====] - 1374s 3s/step - loss: 0.9830 - accuracy: 0.6305 - val_loss: 1.0376 - val_accuracy: 0.6087 - lr: 5.0000e-05
Epoch 9/15
127/449 [=====>.....] - ETA: 15:26 - loss: 0.9569 - accuracy: 0.6455
```

EPOCH = 10/15



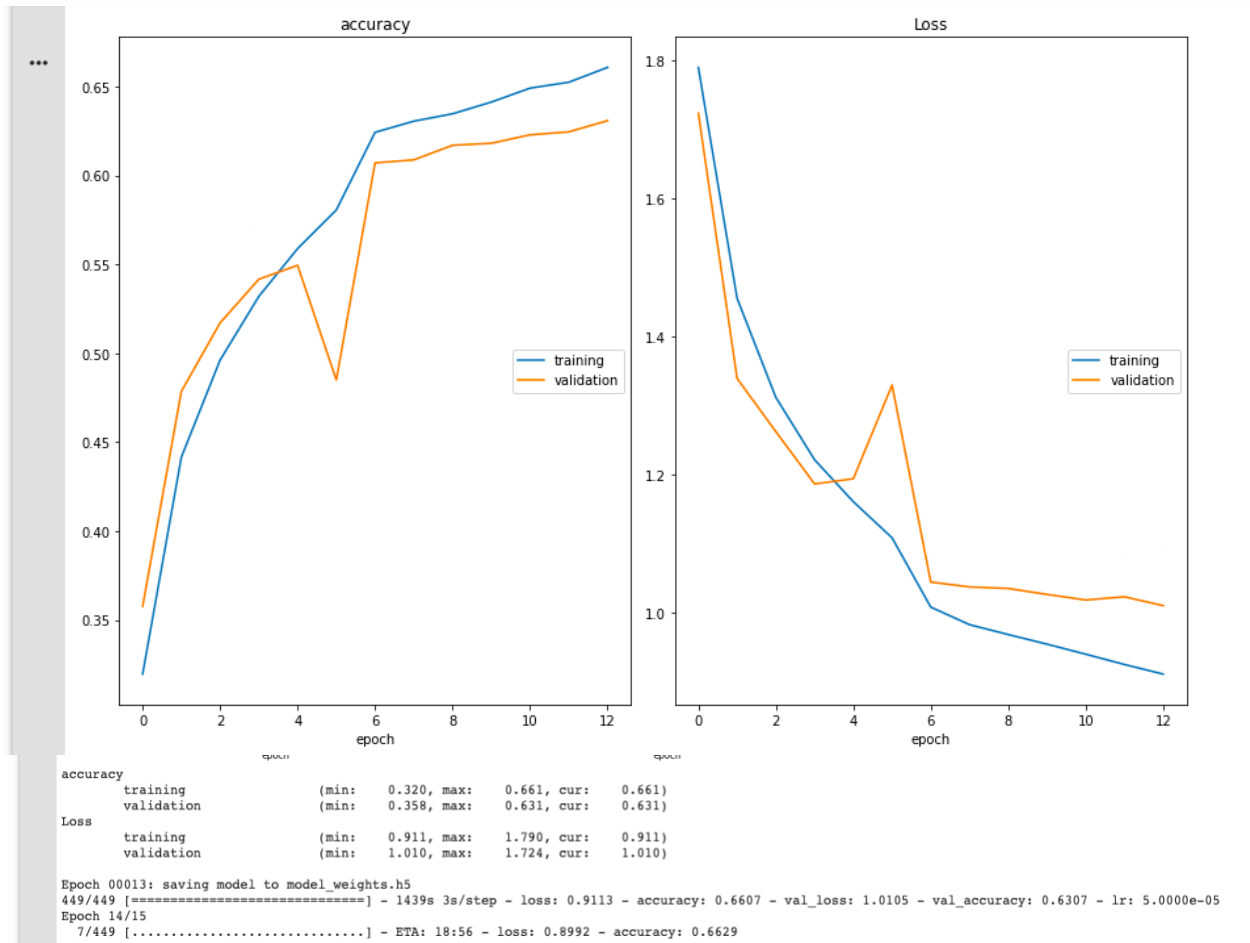
EPOCH = 13/15



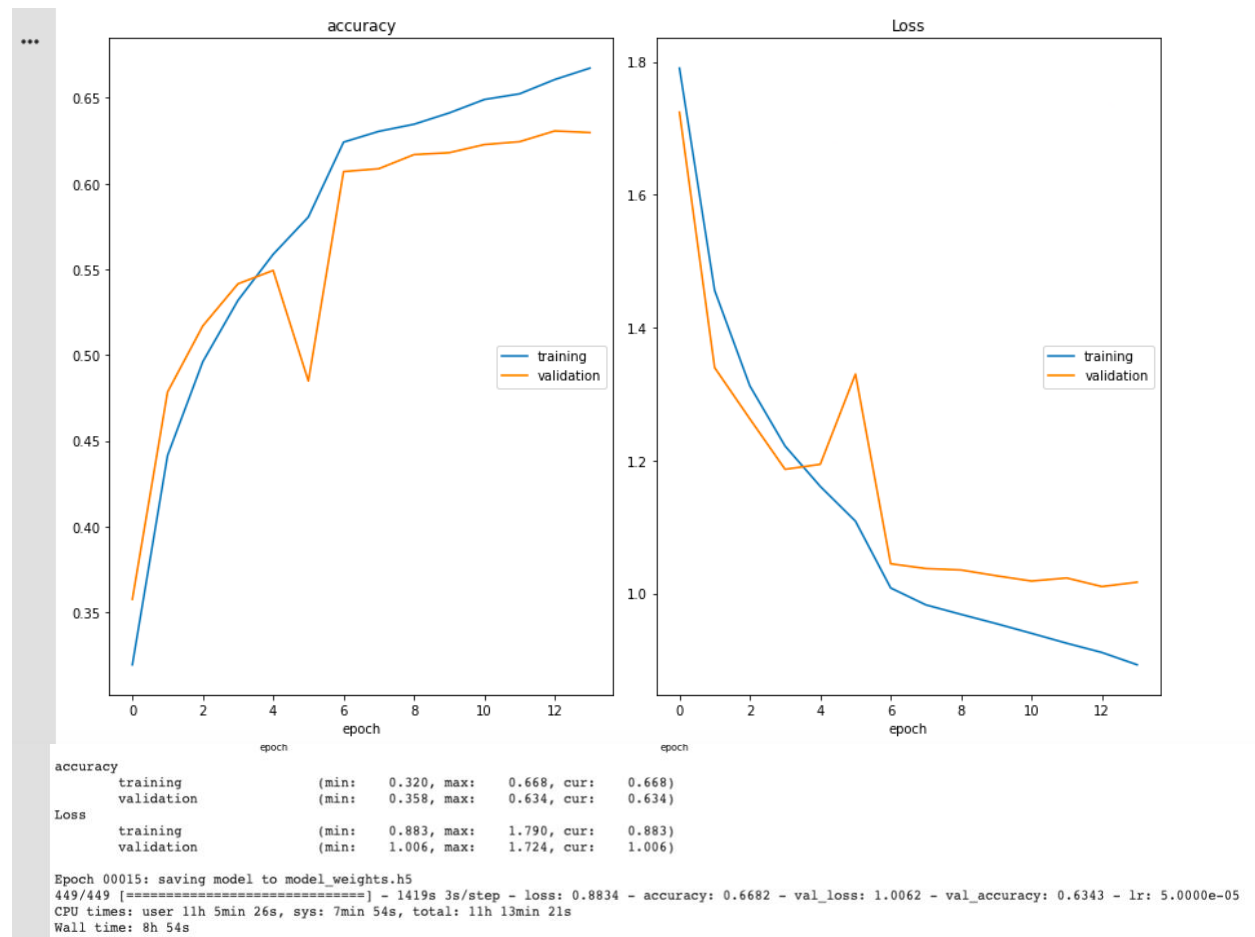
	epoch		epoch
accuracy	training	(min: 0.320, max: 0.652, cur: 0.652)	
	validation	(min: 0.358, max: 0.624, cur: 0.624)	
Loss	training	(min: 0.925, max: 1.790, cur: 0.925)	
	validation	(min: 1.019, max: 1.724, cur: 1.023)	

Epoch 00012: saving model to model_weights.h5
 449/449 [=====] - 1446s 3s/step - loss: 0.9252 - accuracy: 0.6523 - val_loss: 1.0234 - val_accuracy: 0.6244 - lr: 5.0000e-05
 Epoch 13/15
 16/449 [>.....] - ETA: 21:11 - loss: 0.9687 - accuracy: 0.6514

EPOCH = 14/15



EPOCH = 15/15 – FINAL RESULT



RESULT: We see that Training loss is decreased and Accuracy is increased. Because of high learning rate, we see extreme jumps in the curve. Validation accuracy reaches 63.4%.

TEST CASE 4.9: CODE SHOULD BE ABLE TO FLIP BETWEEN LIVE WEBCAM AND VIDEO FEED

```
class VideoCamera(object):
    def __init__(self):
        self.video = cv2.VideoCapture('/Users/punithabragadish/Desktop/Face_Recog_Emotions/videos/office.mp4')
        #self.video = cv2.VideoCapture(0)

    def __del__(self):
        self.video.release()

    # returns camera frames along with bounding boxes and predictions
    def get_frame(self):
        _, fr = self.video.read()
        gray_fr = cv2.cvtColor(fr, cv2.COLOR_BGR2GRAY)
        faces = facec.detectMultiScale(gray_fr, 1.3, 5)

        for (x, y, w, h) in faces:
            fc = gray_fr[y:y+h, x:x+w]

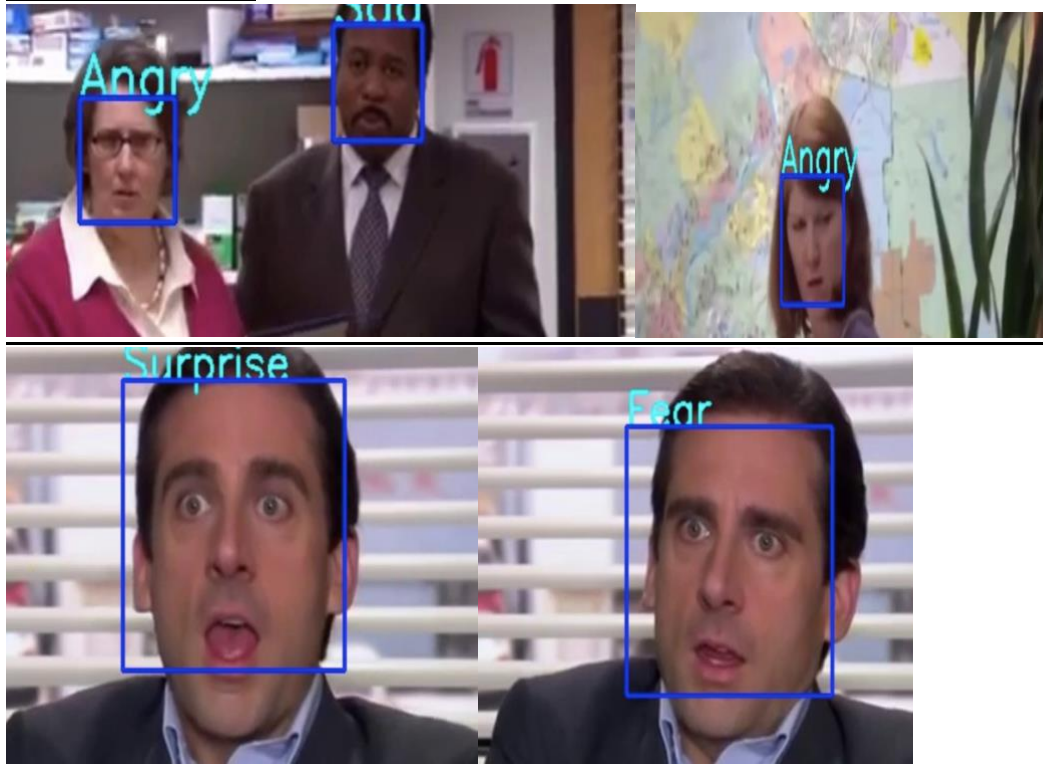
            roi = cv2.resize(fc, (48, 48))
            pred = model.predict_emotion(roi[np.newaxis, :, :, np.newaxis])

            cv2.putText(fr, pred, (x, y), font, 1, (255, 255, 0), 2)
            cv2.rectangle(fr, (x,y),(x+w,y+h),(255,0,0),2)

        _, jpeg = cv2.imencode('.jpg', fr)
        return jpeg.tobytes()
```

RESULT: It is achieved in camera.py file where for live webcam, we give VideoCapture(0) and for videofeed/video file we give that particular path.

TEST CASE 4.10: EMOTION DETECTION THROUGH VIDEO FEED

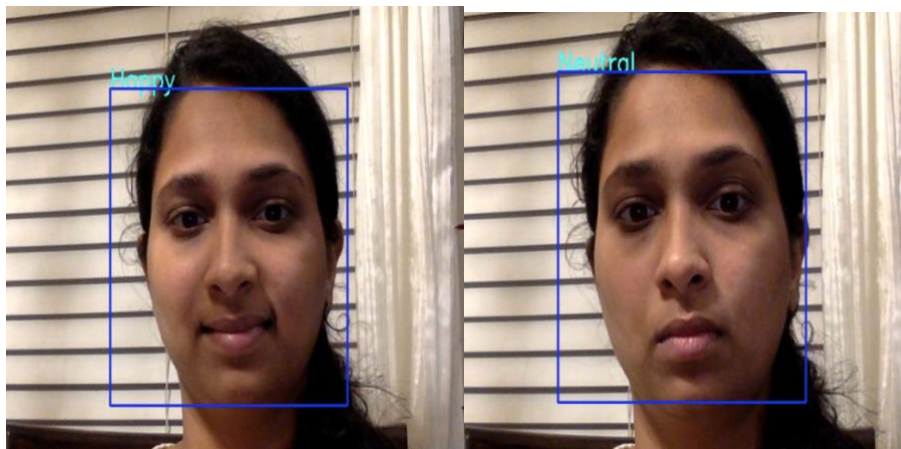


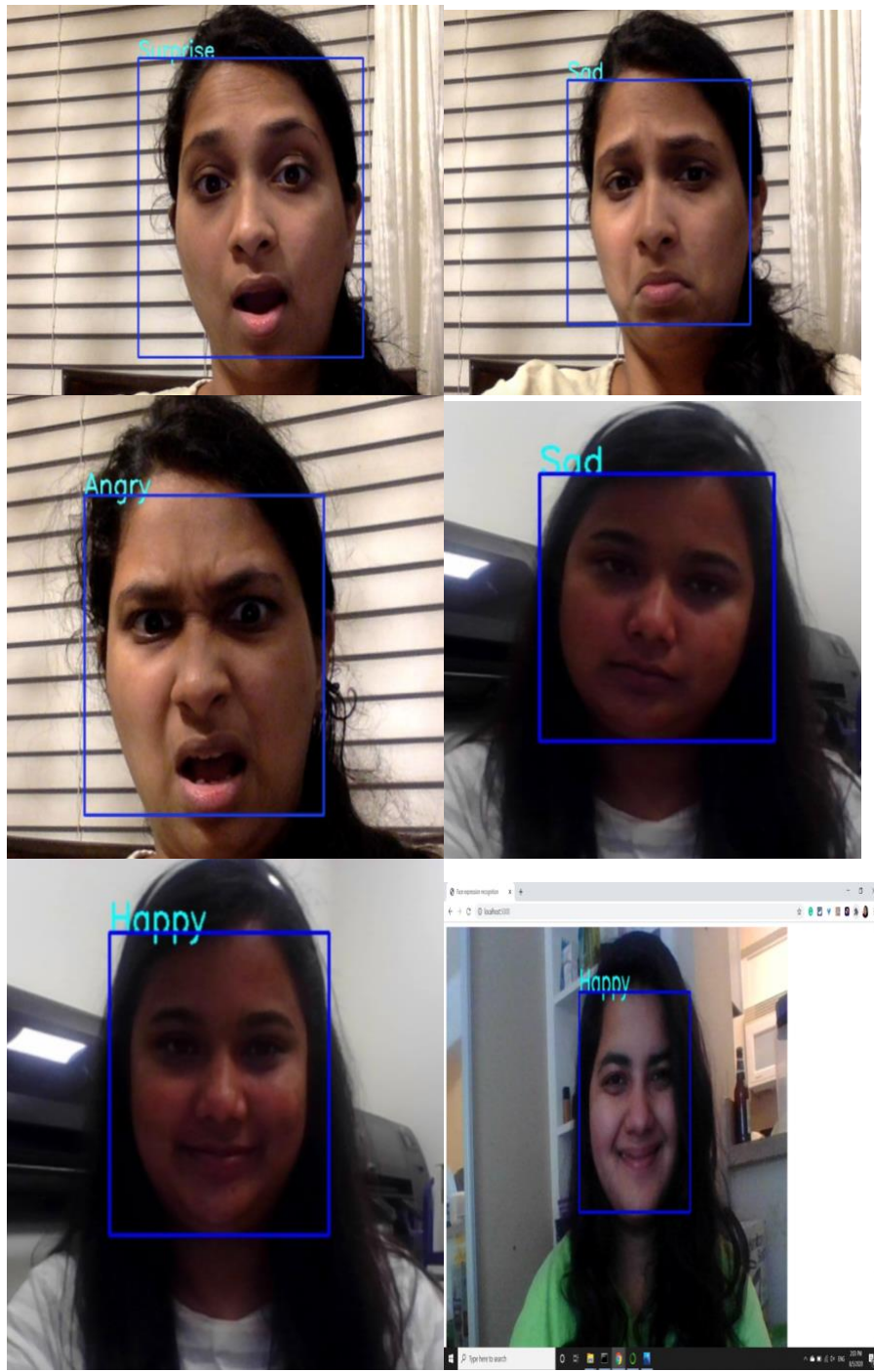


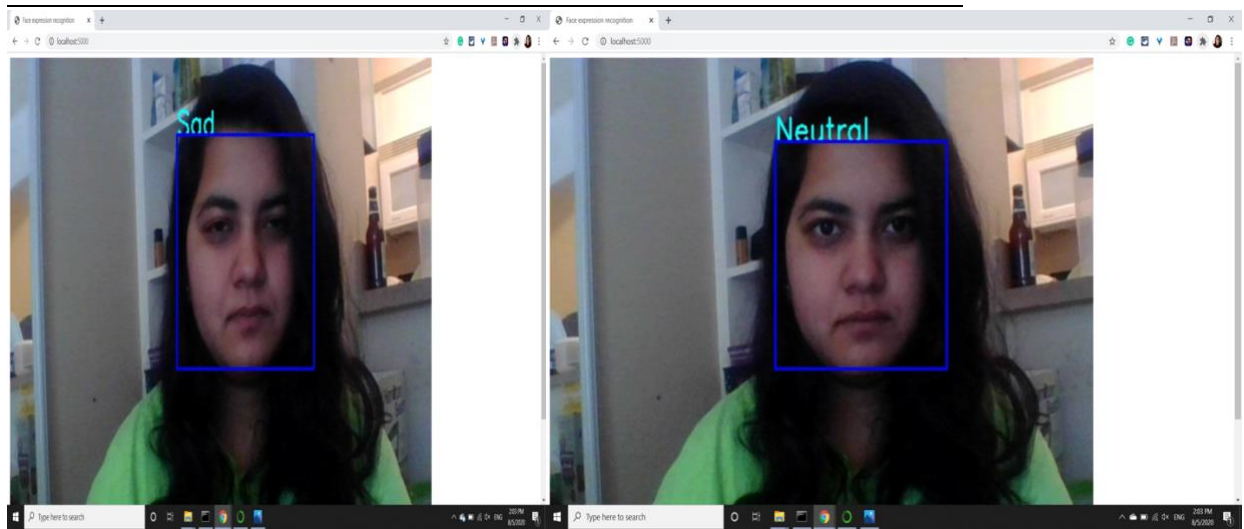
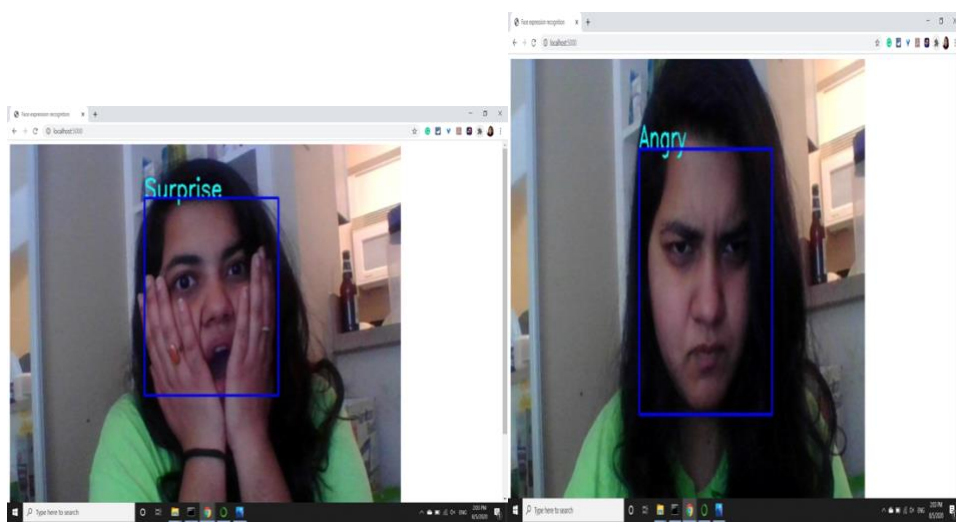
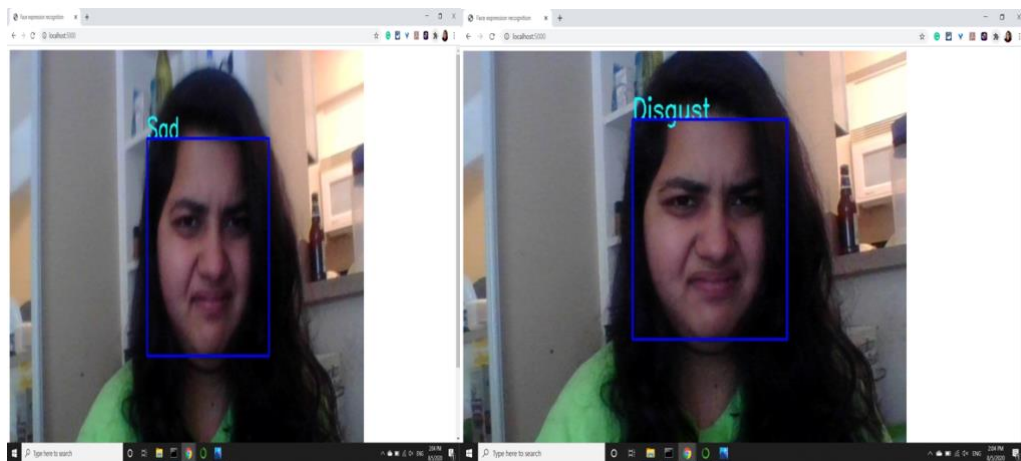


RESULT: All the emotions are captured correctly from the video feed for single face as well as multiple faces.

TEST CASE 4.11: EMOTION DETECTION THROUGH WEBCAM







RESULT: All the emotions are captured correctly from live webcam.

5.ALGORITHMS AND TECHNIQUES:

5.1 MINIBATCH GRADIENT DESCENT ALGORITHM:

→ Gradient descent is an optimization algorithm often used for finding the weights or coefficients of machine learning algorithms, such as artificial neural networks and logistic regression. It works by having the model make predictions on training data and using the error on the predictions to update the model in such a way as to reduce the error.

→ Mini-batch gradient descent is a variation of the gradient descent algorithm splits the training dataset into **small batches**, that calculates the error for each example in the training dataset, but only updates the model coefficients after all training examples have been evaluated.

→ Implementations may choose to sum the gradient over the mini batch which further reduces the variance of the gradient.

→ In our project, because of this, we train and validated the data into batches with batch_size=64. It is implemented in below step.

Step 3: Generate Training and Validation batches

```
In [9]: # Defining the parameters for the data loader.
# from tensorflow.keras.preprocessing.image import ImageDataGenerator - > This library makes it so simpler to use keras
# load the data for preprocessing images

# Generally we go by setting the height and width of the images, but since we already know the image size which is 48*48
# we are directly setting the image size to be 48.

img_size = 48
batch_size = 64 # here 64 is chosen for the convenient manner. later we can experiment changing the number of it.

# now we are creating the datagenerator object one for training set and one for the validation set.
# FOR TRAINING SET
datagen_train = ImageDataGenerator()
```

→ They are fixed number of training examples that are used for single one step of gradient descent. So after training the mini batches of the fixed size, we are going to perform series of steps in single epoch.

- Step 1: First is picking a minibatch
- Step 2: We feed minibatch to the NN
- Step 3: Calculate the mean gradient of the mini batch.

- Step 4: Use the mean gradient to update the weights
- Step 5: Repeat this process until we come to a local optimum result.

5.2 DATA AUGUMENTATION TECHNIQUE:

→ Data Image augmentation is a very powerful technique used to artificially create variations in existing images to expand an existing image data set. This creates new and different images from the existing image data set that represents a comprehensive set of possible images.

→ This is done by applying different transformation techniques like zooming the existing image, rotating the existing image by a few degrees, shearing or cropping the existing set of images etc.

→ A CNN, due to its in-variance property, can classify objects even when visible in different sizes, orientations or different illumination. Hence, we can take the small dataset of images and transform the objects to different sizes by zooming in or zooming out, flipping them vertically or horizontally or changing the brightness whatever makes sense for the object. This way we create a rich, diverse data set with variations.

→ Augmentation is applied as a pre-processing step to increase the size of the data set. This is usually done when we have a small training data set that we want to expand.

Basic data augmentation techniques

- **Flipping:** flipping the image vertically or horizontally
- **Rotation:** rotates the image by a specified degree.
- **Shearing:** shifts one part of the image like a parallelogram
- **Cropping:** object appear in different positions in different proportions in the image
- **Zoom in, Zoom out**

- Changing brightness or contrast

→ In our project, since Disgust Images are low in numbers, we are using Data augmentation technique – Flipping horizontally

```
In [7]: # Examining the class balance for specifically the training set.
# here for first iteration, it goes to train folder picks up happy, then angry, disgust fear and so on.
for expression in os.listdir("train/"):
    print(str(len(os.listdir("train/" + expression))) + " " + expression + " images")

# this function prints the number of images present for each emotion along with their emotion names.
7214 happy images
4830 sad images
4097 fear images
3171 surprise images
4965 neutral images
3995 angry images
436 disgust images

In [8]: # for testing folder.
for expression in os.listdir("test/"):
    print(str(len(os.listdir("test/" + expression))) + " " + expression + " images")
1774 happy images
1247 sad images
1024 fear images
831 surprise images
1233 neutral images
958 angry images
111 disgust images
```

• If we see the above data, it is almost balanced here except for disgust images. we can balance it by data augmentation method.

5.3 CNN MODEL

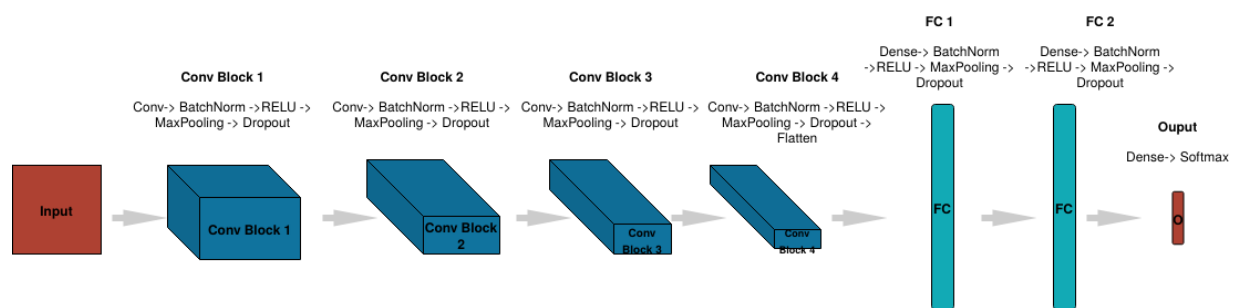


Fig 1: CNN Model

In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc.

CNN image classifications takes an input image, process it and classify it under certain emotion categories. Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully

connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1.

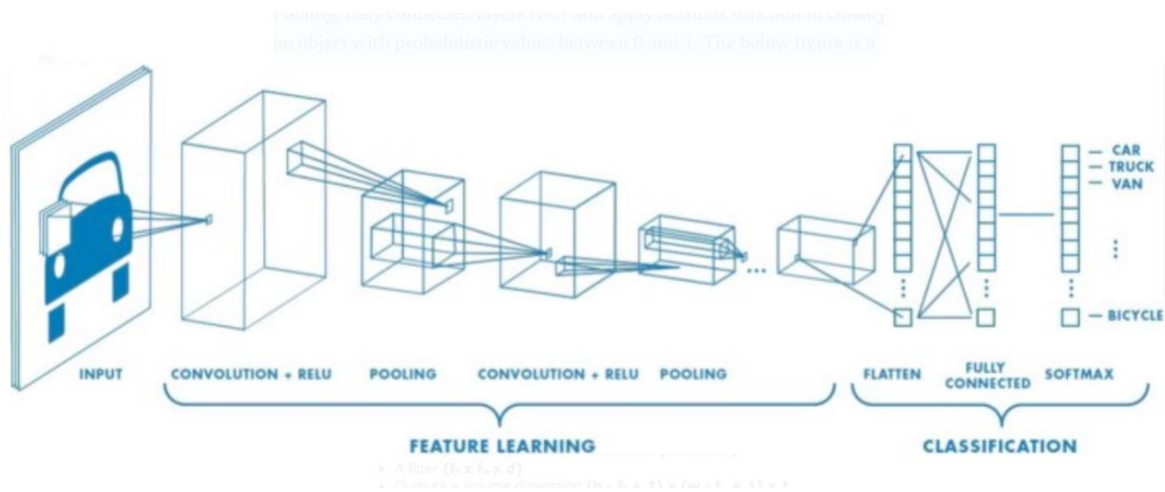


Fig 2: CNN Layers

Non Linearity (ReLU):

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = \max(0, x)$. ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.

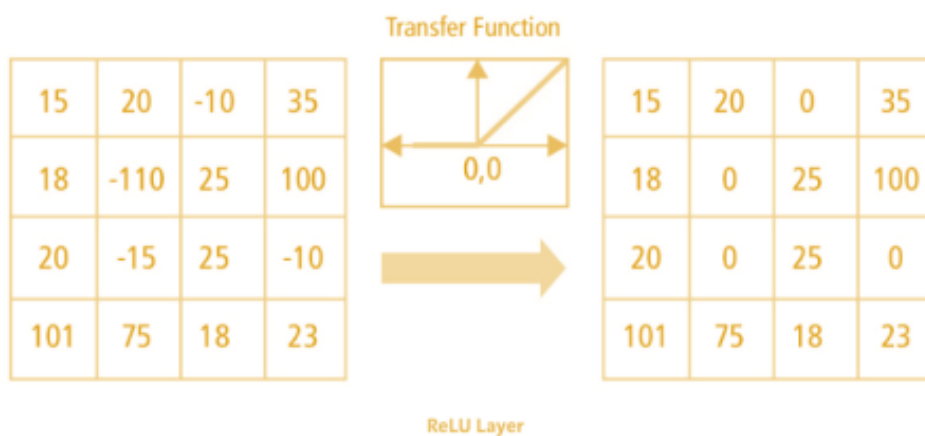


Fig 3: ReLU operation

There are other non linear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance wise ReLU is better than the other two.

Max Pooling:

Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

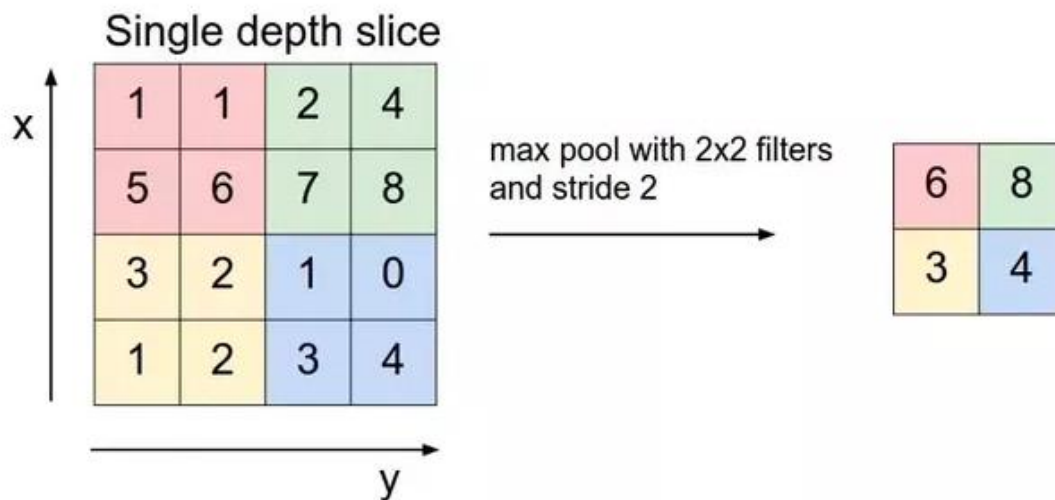


Fig 4: Max Pooling – featured matrix

Fully Connected Layer:

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.

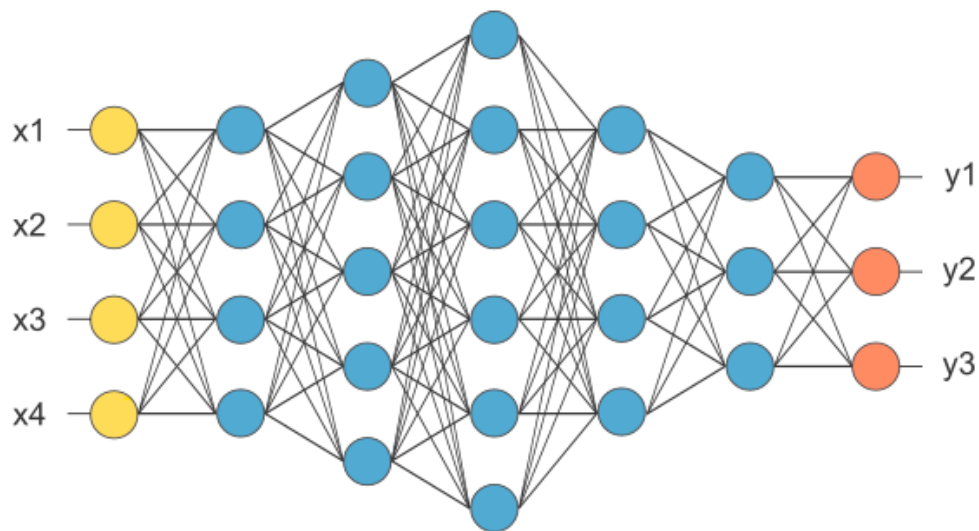


Fig 5: After Pooling, Flattened as FC layer

In the above diagram, the feature map matrix will be converted as vector (x_1, x_2, x_3, \dots). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as SoftMax or sigmoid to classify the outputs as happy, angry, sad, fear, neutral, disgust and surprise.