EMOTION TO EMOJI

PROJECT AIM:

Our project aims to develop a model which takes the video of a person as an input and displays the emotion using an emoji. This model uses deep learning concepts to recognize the emotion of one or multiple person at a time.

BRIEF WALKTHROUGH:

- LOADING THE APPROPRIATE DATA SET CONTAINING SUFFICIENT IMAGES OF ALL 7 TYPES OF EMOTIONS
- DATA AUGUMENTATION AND IMAGE PRE-PROCESSING USING “ImageGenerator” and “.flow_from_directory”
- BUILDING THE CNN ARCHITECTURE
- COMPILING OUR MODEL USING LOSS, OPTIMIZERS (Adam) & METRICS
- TRAINING OUR MODEL WITH TRAINING SET AND SAVING OUR MODEL
- EVALUATING TESTING ACCURACY
- USING OPENCV WHICH IS A REAL TIME COMPUTER VISION LIBRARY TO CAPTURE THE VIDEO USING APPROPRIATE FRAME
- MAPPING THE EMOJIS WITH THE HIGHEST PROBABLE EMOTION
- USING FLASK FRAMEWORK TO BUILD A WEBPAGE

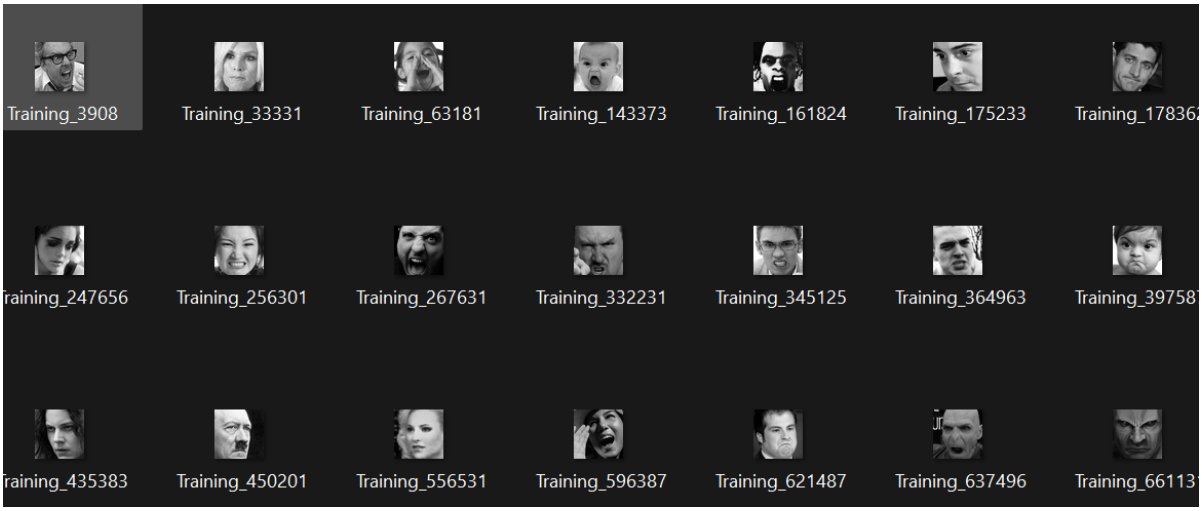
TECH USED:

Jupyter Notebook
Anaconda
Google Colab
Python
Flask framework
Numpy, pandas, opencv ,etc
Kaggle FER2013 DATASET

WORKING

DATACOLLECTION:

For this we used fer2013 dataset which contains black and white images of 7 different types of emotions



MODEL:
We used 2 different models - one is our coustomized model , we did some changes from a research paper to get maximum accuracy and other is vgg16 (pretrained complex cnn model) , we trained on both , vgg16 being complex and had more layers took a lot of time for even 20 epochs , but had almost same accuracy as our coustomized model.

vgg16:-

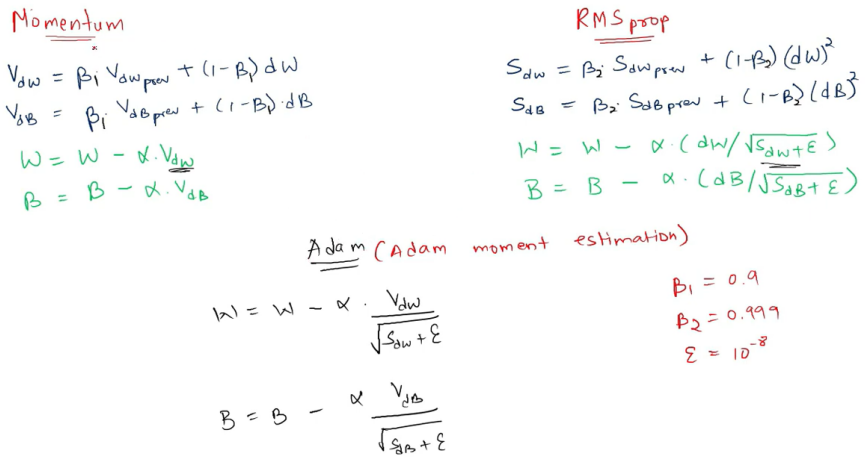
```
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(48,48,1)))
model.add(Convolution2D(64, kernel_size=(3,3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, kernel_size=(3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, kernel_size=(3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, kernel_size=(3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, kernel_size=(3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, kernel_size=(3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))


model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512,kernel_size=(3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, kernel_size=(3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
```

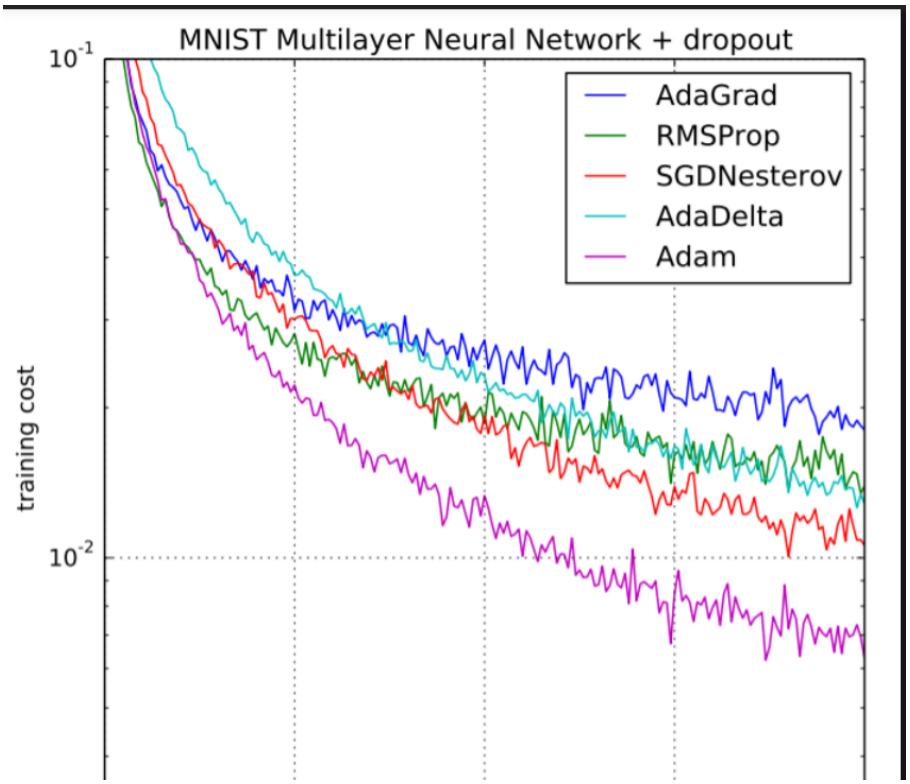
Our model:-

```
emotion_model=Sequential()

emotion_model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(48,48,1)))

emotion_model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))

emotion_model.add(MaxPooling2D(2,2))

emotion_model.add(Dropout(.25))

emotion_model.add(Conv2D(128, kernel_size=(3,3), activation='relu'))

emotion_model.add(MaxPooling2D(2,2))

emotion_model.add(Conv2D(128, kernel_size=(3,3), activation='relu'))

emotion_model.add(MaxPooling2D(2,2))

emotion_model.add(Dropout(.25))

emotion_model.add(Flatten())

emotion_model.add(Dense(1024, activation='relu'))

emotion_model.add(Dropout(.5))

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 46, 46, 32)        320

 conv2d_1 (Conv2D)           (None, 44, 44, 64)        18496

 max_pooling2d (MaxPooling2D  (None, 22, 22, 64)       0
 )

 dropout (Dropout)           (None, 22, 22, 64)        0

 conv2d_2 (Conv2D)           (None, 20, 20, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 10, 10, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_1 (Dropout)         (None, 4, 4, 128)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 1024)              2098176

 dropout_2 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 7)                 7175

=================================================================
Total params: 2,345,607
Trainable params: 2,345,607
Non-trainable params: 0
```

We than used adam as our optimizer as it combines both rmsprop and momentum and provides the best results.



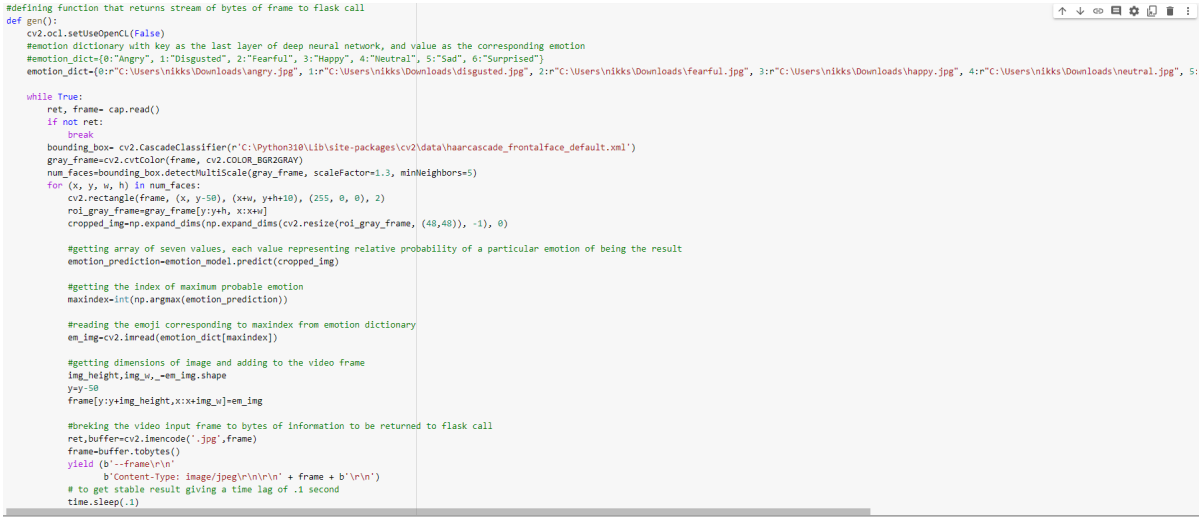


After compiling our model we train it by passing the training dataset around 20-25 times , epoch=20 to 25
Since training again and again is time consuming thus after training once we saved our model's weights and use it again without running the epochs and directly loading the saved pretrained weights (the files will be uploaded , two files , one vgg16 and other our coustomized one will be uploaded in the end of this file so you can directly download and use our model)

REAL TIME FACE RECOGNITION:

- Opencv is used for real time computer vision
- We take live video via web cam and the input image is processed to our model

- Then the trained model recognizes the emotion and maps it with the emoji to display it in the output
- The frame is adjusted and BGR image is converted to gray as our training set was also in grayscale
- getting array of seven values, each value representing relative probability of a particular emotion of being the result and getting the index of maximum probable emotion
- Reading the emoji corresponding to maxindex emotion dictionary
- Getting the dimensions of video and image to the frame
- Displaying the video and the output with a time lag of .1 seconds and emojis are aligned according to the frame(top left)



```
#defining function that returns stream of bytes of frame to flask call
def gen():
    cv2.ocl.setUseOpenCL(False)
    #emotion dictionary with key as the last layer of deep neural network, and value as the corresponding emotion
    #emotion_dict={0:"Angry", 1:"Disgusted", 2:"Fearful", 3:"Happy", 4:"Neutral", 5:"Sad", 6:"Surprised"}
    emotion_dict={0:r"C:\Users\nikks\Downloads\angry.jpg", 1:r"C:\Users\nikks\Downloads\disgusted.jpg", 2:r"C:\Users\nikks\Downloads\fearful.jpg", 3:r"C:\Users\nikks\Downloads\happy.jpg", 4:r"C:\Users\nikks\Downloads\neutral.jpg", 5:

    while True:
        ret, frame= cap.read()
        if not ret:
            break
        bounding_box= cv2.CascadeClassifier(r'C:\Python310\Lib\site-packages\cv2\data\haarcascade_frontalface_default.xml')
        gray_frame=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        num_faces=bounding_box.detectMultiScale(gray_frame, scaleFactor=1.3, minNeighbors=5)
        for (x, y, w, h) in num_faces:
            cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (255, 0, 0), 2)
            roi_gray_frame=gray_frame[y:y+h, x:x+w]
            cropped_img=np.expand_dims(np.expand_dims(cv2.resize(roi_gray_frame, (48,48)), -1), 0)

            #getting array of seven values, each value representing relative probability of a particular emotion of being the result
            emotion_prediction=emotion_model.predict(cropped_img)

            #getting the index of maximum probable emotion
            maxindex=int(np.argmax(emotion_prediction))

            #reading the emoji corresponding to maxindex from emotion dictionary
            em_img=cv2.imread(emotion_dict[maxindex])

            #getting dimensions of image and adding to the video frame
            img_height,img_w,_=em_img.shape
            y=y-50
            frame[y:y+img_height,x:x+img_w]=em_img

            #breking the video input frame to bytes of information to be returned to flask call
            ret,buffer=cv2.imencode('.jpg',frame)
            frame=buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
            # to get stable result giving a time lag of .1 second
            time.sleep(.1)
```

MAKING A GUI:

After training the model and saving the labels in a .h5 file, we worked towards integrating the ML model with an interface like an app or webpage. We used flask framework for creating web application in python, to integrate the I model to web app.

```
            #breking the video input frame to bytes of information to be returned to flask call
            ret,buffer=cv2.imencode('.jpg',frame)
            frame=buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
            # to get stable result giving a time lag of .1 second
            time.sleep(.1)
```

```
@app.route('/')
def home():
  #error handling if the homepage of html file is not shown up
    try:
        return render_template(r'C:\Users\nikks\Downloads\mainpage.html')#render template helps in showing a separate html file in flask
    except Exception as e:
        return str(e)
@app.route('/d')
def d():
    return Response(gen(), mimetype='multipart/x-mixed-replace; boundary=frame')# Response helps in returning stream of bytes of frame
if __name__=="__main__":
  #running application
    Flask.run(app)
```

After adding corresponding emotion's emoji to the frame, using imencode, we converted the image format of the frame to a streaming data format in form of bytes of data to be

transferred over the net, then to return the frame to main function call in the flask application, we used yield keyword that is used to return from a function without destroying the states of its local variable and when the function is called, the execution starts from the last yield statement.

Home function calls the mainpage.html file where the html framework is stored, which has a button that calls the function to start the camera view and emotion to emoji service, by getting the image view from the ‘d’ function defined using the flask fraework.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div class="container">
        <div class="row">
            <div class="col-lg-8 offset-lg-2">
                <h3 class="mt-5">WANT TO KNOW YOUR EMOTION CLICK BELOW</h3>
                <input type="button" onclick ="get_img()"  value="click here">

            </div>
        </div>
    </div>
    <script>
        function get_img(){
            var a= document.createElement("img");
            a.src="{{url_for('d')}}";
            document.body.appendChild(a);
        }

    </script>
</body>
</html>
```

REFERENCES:

https://analyticsindiamag.com/my-first-cnn-project-emotion-detection-using-convolutional-neural-network-with-tpu/

https://www.youtube.com/watch?v=mzX5oqd3pKA&t=511s

https://drive.google.com/file/d/1C4hEXjMeeXOX3Jd-DHuv4pAlUs1ekgqG/view?usp=sharing(pre-trained weights of coustomized model)

https://drive.google.com/file/d/1PVklZAUiW7durtLhzdZoRtgr67LcZnaK/view?usp=sharing (pre-trained weights of vgg16)