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

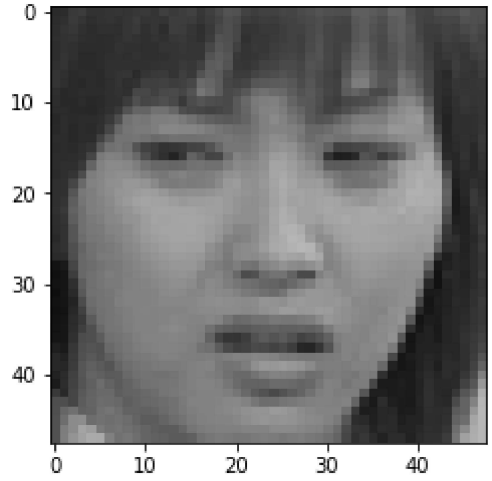
```
Mounted at /content/drive
```

```
import tensorflow as tf
import numpy as np
import cv2
import matplotlib.pyplot as plt
import pickle
import os
```

```
img = cv2.imread("/content/drive/MyDrive/IMAGES/IMAGES_FEC/train/1/1073.jpg")
```

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7ff95903ce10>
```




```
dataDirectory = "/content/drive/MyDrive/IMAGES/IMAGES_FEC/train" # training images are saved
```

```
classes = ["0","1","2","3","4","5","6"] #0-angry 1-disgust 2-fear 3-happy 4-neutral 5-sad 6-s
```

```
img_size = 224 ## Imagenet --> 224 x 224
```

read all the images and convert it to an array

```
training_data = [] #data array

def create_training_data():
    for category in classes:
        path = os.path.join(dataDirectory , category)
        class_num = classes.index(category)
```

```
            class_num = classes.index(category)
            c = 0
            for img in os.listdir(path):
                if c > 3500:
                  break;
                try:
                    img_array = cv2.imread(os.path.join(path , img))
                    new_array = cv2.resize(img_array, (img_size,img_size))
                    training_data.append([new_array, class_num])
                    c= c+1
                except Exception as e:
                        pass
        file_name = "image_data.pkl"
        open_file = open(file_name, "wb")
        pickle.dump(training_data, open_file)
        open_file.close()


    #file_name = "image_data3.pkl"
    open_file = open(file_name, "wb")
    pickle.dump(training_data, open_file)
    open_file.close()


    create_training_data()
```

```
    ---------------------------------------------------------------------------
    KeyboardInterrupt                         Traceback (most recent call last)
    <ipython-input-12-be131b356c3a> in <module>()
    ----> 1 create_training_data()

    <ipython-input-10-930617282105> in create_training_data()
         10                break;
         11            try:
    ---> 12                img_array = cv2.imread(os.path.join(path , img))
         13                new_array = cv2.resize(img_array, (img_size,img_size))
         14                training_data.append([new_array, class_num])

    KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
    training_data = []
    file_name = "image_data.pkl"

    open_file = open(file_name, "rb")
    training_data = pickle.load(open_file)
    open_file.close()


    #print(len(training_data))
    print(len(training_data))
```

```
21257
```

```
import random
random.shuffle(training_data) #for making the model more dynamic
```

```
X = [] #data
Y = [] #lable

for data,lable in training_data:
    X.append(data)
    Y.append(lable)
```

```
X = np.array(X).reshape(-1, img_size ,img_size ,3) #we need 4 dimesions
```

```
X = X/255.0   #normalizing the data
```

```
X.shape
```

```
(21257, 224, 224, 3)
```

```
Y = np.array(Y)
```

```
Y.shape
```

```
(21257,)
```

```
from tensorflow import keras
from tensorflow.keras import layers
```

```
# MobileNet-v2 is a convolutional neural network that is 53 layers deep.
# This network trained on more than a million images from the ImageNet database
# The pretrained network can classify images into 1000 object categories,
model = tf.keras.applications.MobileNetV2()  #pre-trained model
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobi
14540800/14536120 [==============================] - 0s 0us/step
```

```
model.summary()
```

transfer learning -Tuning, weights will start from last checkpoint

```
base_input = model.layers[0].input #input to our model
```

```
base_output = model.layers[-2].output
```

```
base_output
```

```
<KerasTensor: shape=(None, 1280) dtype=float32 (created by layer 'global_average_poolin
```

```
final_output = layers.Dense(128)(base_output)  #adding new layer, after the output of global
final_output = layers.Activation('relu')(final_output) #activation function
final_output = layers.Dense(64)(final_output)
final_output = layers.Activation('relu')(final_output)
final_output = layers.Dense(7, activation ='softmax')(final_output) #we are classifing 7 clas
```

```
final_output #output of our model
```

```
new_model = keras.Model(inputs = base_input , outputs = final_output)
```

```
new_model.summary()
```

```
new_model.compile(loss = "sparse_categorical_crossentropy" ,optimizer ="adam", metrics =["acc
```

```
new_model.fit(X,Y , epochs =25)
```

```
Epoch 1/25
665/665 [==============================] - 93s 108ms/step - loss: 1.4994 - accuracy: 0.
Epoch 2/25
665/665 [==============================] - 73s 109ms/step - loss: 1.1579 - accuracy: 0.
Epoch 3/25
665/665 [==============================] - 71s 106ms/step - loss: 1.0627 - accuracy: 0.
Epoch 4/25
665/665 [==============================] - 71s 107ms/step - loss: 0.9952 - accuracy: 0.
Epoch 5/25
665/665 [==============================] - 72s 108ms/step - loss: 0.9205 - accuracy: 0.
Epoch 6/25
665/665 [==============================] - 71s 107ms/step - loss: 0.8586 - accuracy: 0.
Epoch 7/25
665/665 [==============================] - 71s 107ms/step - loss: 0.8109 - accuracy: 0.
Epoch 8/25
665/665 [==============================] - 72s 108ms/step - loss: 0.7526 - accuracy: 0.
Epoch 9/25
665/665 [==============================] - 72s 109ms/step - loss: 0.6825 - accuracy: 0.
Epoch 10/25
665/665 [==============================] - 72s 108ms/step - loss: 0.6197 - accuracy: 0.
Epoch 11/25
665/665 [==============================] - 72s 108ms/step - loss: 0.5748 - accuracy: 0.
Epoch 12/25
665/665 [==============================] - 70s 105ms/step - loss: 0.5307 - accuracy: 0.
Epoch 13/25
```

```
665/665 [==============================] - 71s 107ms/step - loss: 0.4777 - accuracy: 0.
Epoch 14/25
665/665 [==============================] - 70s 106ms/step - loss: 0.4262 - accuracy: 0.
Epoch 15/25
665/665 [==============================] - 71s 107ms/step - loss: 0.3650 - accuracy: 0.
Epoch 16/25
665/665 [==============================] - 73s 109ms/step - loss: 0.3375 - accuracy: 0.
Epoch 17/25
665/665 [==============================] - 73s 110ms/step - loss: 0.3098 - accuracy: 0.
Epoch 18/25
665/665 [==============================] - 73s 110ms/step - loss: 0.2795 - accuracy: 0.
Epoch 19/25
665/665 [==============================] - 72s 108ms/step - loss: 0.2578 - accuracy: 0.
Epoch 20/25
665/665 [==============================] - 72s 108ms/step - loss: 0.2347 - accuracy: 0.
Epoch 21/25
665/665 [==============================] - 72s 108ms/step - loss: 0.2103 - accuracy: 0.
Epoch 22/25
665/665 [==============================] - 72s 108ms/step - loss: 0.1982 - accuracy: 0.
Epoch 23/25
665/665 [==============================] - 72s 108ms/step - loss: 0.1846 - accuracy: 0.
Epoch 24/25
665/665 [==============================] - 72s 108ms/step - loss: 0.1903 - accuracy: 0.
Epoch 25/25
665/665 [==============================] - 72s 108ms/step - loss: 0.1703 - accuracy: 0.
<tensorflow.python.keras.callbacks.History at 0x7ff95012d310>
```

```
new_model.save('pro_model_v1.h5')
```

```
#Hierarchical Data Format 5. It is an open-source file which comes in handy to store large an
new_model = tf.keras.models.load_model('pro_model_v1.h5')
```

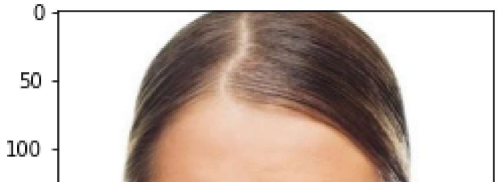
```
test_img = cv2.imread("disgust.jpg")
```

```
test_img.shape
```

```
(320, 319, 3)
```

```
#cv2. cvtColor() method is used to convert an image from one color space to another
plt.imshow(cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7ff8e43f5850>
```



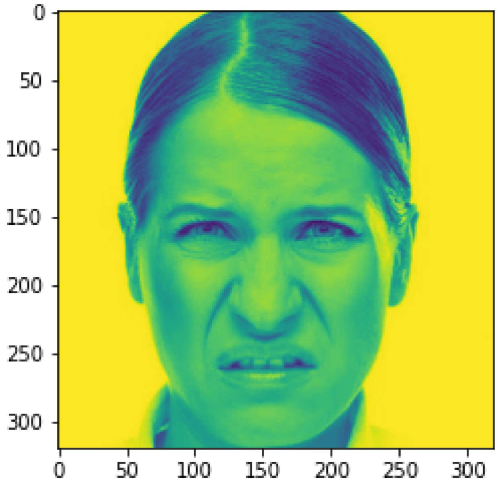
We need face detection algorithm

```
faceCascade = cv2.CascadeClassifier('facedetection_algorithm.xml')
```

```
gray = cv2.cvtColor(test_img , cv2.COLOR_BGR2GRAY) #BGR to GRAY color

plt.imshow(gray)
```

```
<matplotlib.image.AxesImage at 0x7ff8e4344290>
```



```
gray.shape
```

```
(320, 319)
```

```
faces = faceCascade.detectMultiScale(gray, 1.1,4)
for x,y,w,h in faces:
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = test_img[y:y+h, x:x+w]
    cv2.rectangle(test_img, (x,y), (x+w ,y+h), (255,0,0,0), 2)
    facess = faceCascade.detectMultiScale(roi_gray)
    if len(facess) ==0:
        print("face not detected")
    else:
        for (ex,ey,ew,eh) in facess:
            face_roi = roi_color[ey: ey+eh, ex:ex+ew]

plt.imshow(cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7ff8e4332c10>
```

```
plt.imshow(cv2.cvtColor(face_roi, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7ff8e42a24d0>
```

```
final_image = cv2.resize(face_roi, (224,224))
final_image = np.expand_dims(final_image, axis =0)#need 4th dimensions
final_image = final_image #normalization


prediction = new_model.predict(final_image)


prediction[0]
```

```
array([1.4423076e-03, 9.9528956e-01, 2.2711162e-03, 1.4332406e-05,
       5.4088596e-05, 2.6412756e-04, 6.6459784e-04], dtype=float32)
```

```
result = np.argmax(prediction)
if(result == 0):
    print("angry")
elif(result == 1):
    print("disgust")
```

```
    elif(result == 2):
        print("fear")
    elif(result == 3):
        print("happy")
    elif(result == 4):
        print("neutral")
    elif(result == 5):
        print("sad")
    elif(result == 6):
        print("surprise")
```

```
     disgust
```

✓ 0s completed at 4:33 AM