IMPLEMENTATION

Face mask :

MobilenetV2 :

Training :

```
# import the necessary packages
# Main modules - keras(for deep learning) , tensorflow(deep learning) , numpy(math , matrix) , sklearn (prediction)
import numpy as np
import os
import matplotlib.pyplot as plt
from imutils import paths

from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```
# importing the dataset form machine

dataset='/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/dataset'
imagePaths=list(paths.list_images(dataset))

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
```

```
imagePaths
```

```
data=[]
labels=[]
```

```
for i in imagePaths:
    label=i.split(os.path.sep)[-2]

data=np.array(data,dtype='float32')
labels=np.array(labels)
```

```
data
```

```
array([[[[-0.654902  , -0.7647059 , -0.7882353 ],
         [-0.654902  , -0.7647059 , -0.7882353 ],
         [-0.654902  , -0.7647059 , -0.7882353 ],
         ...,
         [-0.60784316, -0.6862745 , -0.69411767],
         [-0.60784316, -0.6862745 , -0.69411767],
         [-0.6156863 , -0.69411767, -0.7019608 ]],

        [[-0.6627451 , -0.77254903, -0.79607844],
         [-0.6627451 , -0.77254903, -0.79607844],
         [-0.6627451 , -0.77254903, -0.79607844],
         ...,
         [-0.6156863 , -0.69411767, -0.7019608 ],
         [-0.6156863 , -0.69411767, -0.7019608 ],
         [-0.62352943, -0.7019608 , -0.70980394]],
```

```
labels
```

```
array(['with_mask', 'with_mask', 'with_mask', ..., 'without_mask',
       'without_mask', 'without_mask'], dtype='<U12')
```

```
# perform one-hot encoding on the labels

lb=LabelBinarizer()
labels=lb.fit_transform(labels)
labels=to_categorical(labels)
```

```
labels
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [0., 1.],
       [0., 1.],
       [0., 1.]], dtype=float32)
```

```
train_X,test_X,train_Y,test_Y=train_test_split(data,labels,test_size=0.20,stratify=labels,r
```

```
train_X
```

```
array([[[[-0.17647058, -0.14509803, -0.05098039],
         [-0.17647058, -0.14509803, -0.05098039],
         [-0.15294117, -0.12156862, -0.02745098],
         ...,
         [-0.18431371, -0.15294117, -0.05882353],
         [-0.20784312, -0.17647058, -0.08235294],
         [-0.20784312, -0.17647058, -0.08235294]],

        [[-0.17647058, -0.14509803, -0.05098039],
         [-0.17647058, -0.14509803, -0.05098039],
         [-0.15294117, -0.12156862, -0.02745098],
         ...,
         [-0.18431371, -0.15294117, -0.05882353],
         [-0.20784312, -0.17647058, -0.08235294],
         [-0.20784312, -0.17647058, -0.08235294]],

        [[-0.16862744, -0.1372549 , -0.04313725],
         [-0.16862744, -0.1372549 , -0.04313725],
         [-0.12941176, -0.09803921, -0.00392157],
         ...,
```

```
test_X
```

```
array([[[[-0.4352941 , -0.31764704,  0.12941182],
         [-0.4588235 , -0.34117645,  0.10588241],
         [-0.4588235 , -0.34117645,  0.10588241],
         ...,
         [-0.7254902 , -0.6627451 , -0.27843136],
         [-0.7254902 , -0.6627451 , -0.27843136],
         [-0.70980394, -0.64705884, -0.26274508]],

        [[-0.4352941 , -0.31764704,  0.12941182],
         [-0.4588235 , -0.34117645,  0.10588241],
         [-0.4588235 , -0.34117645,  0.10588241],
         ...,
         [-0.7254902 , -0.6627451 , -0.27843136],
         [-0.7254902 , -0.6627451 , -0.27843136],
         [-0.7176471 , -0.654902  , -0.27058822]],

        [[-0.4352941 , -0.31764704,  0.12941182],
         [-0.45098037, -0.3490196 ,  0.082353  ],
         [-0.45098037, -0.3490196 ,  0.082353  ],
         ...,
```

```
train_X.shape
```

```
(3066, 224, 224, 3)
```

```
test_X.shape
```

```
(767, 224, 224, 3)
```

```
# construct the training image generator for data augmentation
# creates mupliple images from a single image

aug=ImageDataGenerator(rotation_range=20,
                       zoom_range=0.15,         # rotate , flip , mirror , zoom
                       width_shift_range=0.2,
                       height_shift_range=0.2,
                       shear_range=0.15,
                       horizontal_flip=True,
                       vertical_flip=True,
                       fill_mode='nearest')
```

```
baseModel = MobileNetV2(weights="imagenet", include_top=False,input_tensor=Input(shape=(224, 224, 3)))
```

```
print(baseModel.summary())
```

```
Model: "mobilenetv2_1.00_224"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
Conv1_pad (ZeroPadding2D)       (None, 225, 225, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
Conv1 (Conv2D)                  (None, 112, 112, 32) 864         Conv1_pad[0][0]
__________________________________________________________________________________________________
bn_Conv1 (BatchNormalization)   (None, 112, 112, 32) 128         Conv1[0][0]
__________________________________________________________________________________________________
Conv1_relu (ReLU)               (None, 112, 112, 32) 0           bn_Conv1[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise (Depthw (None, 112, 112, 32) 288         Conv1_relu[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise_BN (Bat (None, 112, 112, 32) 128         expanded_conv_depthwise[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise_relu (R (None, 112, 112, 32) 0           expanded_conv_depthwise_BN[0][0]
__________________________________________________________________________________________________
expanded_conv_project (Conv2D)  (None, 112, 112, 16) 512         expanded_conv_depthwise_relu[0][0
__________________________________________________________________________________________________
expanded_conv_project_BN (Batch (None, 112, 112, 16) 64          expanded_conv_project[0][0]
__________________________________________________________________________________________________


__________________________________________________________________________________________________
block_16_expand (Conv2D)        (None, 7, 7, 960)    153600      block_15_add[0][0]
__________________________________________________________________________________________________
block_16_expand_BN (BatchNormal (None, 7, 7, 960)    3840        block_16_expand[0][0]
__________________________________________________________________________________________________
block_16_expand_relu (ReLU)     (None, 7, 7, 960)    0           block_16_expand_BN[0][0]
__________________________________________________________________________________________________
block_16_depthwise (DepthwiseCo (None, 7, 7, 960)    8640        block_16_expand_relu[0][0]
__________________________________________________________________________________________________
block_16_depthwise_BN (BatchNor (None, 7, 7, 960)    3840        block_16_depthwise[0][0]
__________________________________________________________________________________________________
block_16_depthwise_relu (ReLU)  (None, 7, 7, 960)    0           block_16_depthwise_BN[0][0]
__________________________________________________________________________________________________
block_16_project (Conv2D)       (None, 7, 7, 320)    307200      block_16_depthwise_relu[0][0]
__________________________________________________________________________________________________
block_16_project_BN (BatchNorma (None, 7, 7, 320)    1280        block_16_project[0][0]
__________________________________________________________________________________________________
Conv_1 (Conv2D)                 (None, 7, 7, 1280)   409600      block_16_project_BN[0][0]
__________________________________________________________________________________________________
Conv_1_bn (BatchNormalization)  (None, 7, 7, 1280)   5120        Conv_1[0][0]
__________________________________________________________________________________________________
out_relu (ReLU)                 (None, 7, 7, 1280)   0           Conv_1_bn[0][0]
==================================================================================================
Total params: 2,257,984
Trainable params: 2,223,872
Non-trainable params: 34,112
__________________________________________________________________________________________________
None
```

```
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
```

```
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
```

```
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False
```

```
print(model.summary())
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
Conv1_pad (ZeroPadding2D)       (None, 225, 225, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
Conv1 (Conv2D)                  (None, 112, 112, 32) 864         Conv1_pad[0][0]
__________________________________________________________________________________________________
bn_Conv1 (BatchNormalization)   (None, 112, 112, 32) 128         Conv1[0][0]
__________________________________________________________________________________________________
Conv1_relu (ReLU)               (None, 112, 112, 32) 0           bn_Conv1[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise (Depthw (None, 112, 112, 32) 288         Conv1_relu[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise_BN (Bat (None, 112, 112, 32) 128         expanded_conv_depthwise[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise_relu (R (None, 112, 112, 32) 0           expanded_conv_depthwise_BN[0][0]

__________________________________________________________________________________________________
block_16_expand (Conv2D)        (None, 7, 7, 960)    153600      block_15_add[0][0]
__________________________________________________________________________________________________
block_16_expand_BN (BatchNormal (None, 7, 7, 960)    3840        block_16_expand[0][0]
__________________________________________________________________________________________________
block_16_expand_relu (ReLU)     (None, 7, 7, 960)    0           block_16_expand_BN[0][0]
__________________________________________________________________________________________________
block_16_depthwise (DepthwiseCo (None, 7, 7, 960)    8640        block_16_expand_relu[0][0]
__________________________________________________________________________________________________
block_16_depthwise_BN (BatchNor (None, 7, 7, 960)    3840        block_16_depthwise[0][0]
__________________________________________________________________________________________________
block_16_depthwise_relu (ReLU)  (None, 7, 7, 960)    0           block_16_depthwise_BN[0][0]
__________________________________________________________________________________________________
block_16_project (Conv2D)       (None, 7, 7, 320)    307200      block_16_depthwise_relu[0][0]
__________________________________________________________________________________________________
block_16_project_BN (BatchNorma (None, 7, 7, 320)    1280        block_16_project[0][0]
__________________________________________________________________________________________________
Conv_1 (Conv2D)                 (None, 7, 7, 1280)   409600      block_16_project_BN[0][0]
__________________________________________________________________________________________________
Conv_1_bn (BatchNormalization)  (None, 7, 7, 1280)   5120        Conv_1[0][0]
__________________________________________________________________________________________________
out_relu (ReLU)                 (None, 7, 7, 1280)   0           Conv_1_bn[0][0]
__________________________________________________________________________________________________
average_pooling2d (AveragePooli (None, 1, 1, 1280)   0           out_relu[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 1280)         0           average_pooling2d[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          163968      flatten[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 128)          0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 2)            258         dropout[0][0]
==================================================================================================
Total params: 2,422,210
Trainable params: 164,226
Non-trainable params: 2,257,984
__________________________________________________________________________________________________
```

```
# initialize the initial learning rate, number of epochs to train for,
# and batch size
learning_rate=0.001
Epochs=20
BS=12
```

```
# compile our model
print("[INFO] compiling model...")
opt=Adam(lr=learning_rate,decay=learning_rate/Epochs)
model.compile(loss='binary_crossentropy',optimizer=opt,metrics=['accuracy'])
```

```
[INFO] compiling model...
```

```
# train the head of the network
print("[INFO] training head...")
H=model.fit(
    aug.flow(train_X,train_Y,batch_size=BS),
    steps_per_epoch=len(train_X)//BS,
    validation_data=(test_X,test_Y),
    validation_steps=len(test_X)//BS,
    epochs=Epochs)
```

```
[INFO] training head...
Train for 255 steps, validate on 767 samples
Epoch 1/20
255/255 [==============================] - 130s 510ms/step - loss: 0.3702 - accuracy: 0.8386 - val_loss: 0.3579 - val_accuracy: 0.8677
Epoch 2/20
255/255 [==============================] - 126s 495ms/step - loss: 0.2593 - accuracy: 0.8946 - val_loss: 0.3319 - val_accuracy: 0.8730
Epoch 3/20
255/255 [==============================] - 124s 486ms/step - loss: 0.2015 - accuracy: 0.9172 - val_loss: 0.2699 - val_accuracy: 0.9048
Epoch 4/20
255/255 [==============================] - 124s 486ms/step - loss: 0.1923 - accuracy: 0.9204 - val_loss: 0.3203 - val_accuracy: 0.8929
Epoch 5/20
255/255 [==============================] - 124s 487ms/step - loss: 0.1858 - accuracy: 0.9227 - val_loss: 0.4504 - val_accuracy: 0.8664
Epoch 6/20
255/255 [==============================] - 125s 489ms/step - loss: 0.1885 - accuracy: 0.9221 - val_loss: 0.8745 - val_accuracy: 0.8188
Epoch 7/20
255/255 [==============================] - 124s 488ms/step - loss: 0.1668 - accuracy: 0.9391 - val_loss: 0.7256 - val_accuracy: 0.8135
Epoch 8/20
255/255 [==============================] - 125s 490ms/step - loss: 0.1551 - accuracy: 0.9388 - val_loss: 0.7927 - val_accuracy: 0.8188
Epoch 9/20
255/255 [==============================] - 125s 489ms/step - loss: 0.1587 - accuracy: 0.9391 - val_loss: 0.5994 - val_accuracy: 0.8611
Epoch 10/20
255/255 [==============================] - 125s 490ms/step - loss: 0.1698 - accuracy: 0.9332 - val_loss: 0.5402 - val_accuracy: 0.8770
Epoch 11/20
255/255 [==============================] - 126s 496ms/step - loss: 0.1501 - accuracy: 0.9453 - val_loss: 0.6855 - val_accuracy: 0.8519
Epoch 12/20
255/255 [==============================] - 129s 505ms/step - loss: 0.1436 - accuracy: 0.9434 - val_loss: 0.5045 - val_accuracy: 0.8836
Epoch 13/20
255/255 [==============================] - 125s 491ms/step - loss: 0.1525 - accuracy: 0.9453 - val_loss: 0.7014 - val_accuracy: 0.8519
Epoch 14/20
255/255 [==============================] - 125s 489ms/step - loss: 0.1486 - accuracy: 0.9430 - val_loss: 0.7949 - val_accuracy: 0.8347
Epoch 15/20
255/255 [==============================] - 126s 492ms/step - loss: 0.1302 - accuracy: 0.9519 - val_loss: 0.5703 - val_accuracy: 0.8704
Epoch 16/20
255/255 [==============================] - 126s 494ms/step - loss: 0.1296 - accuracy: 0.9502 - val_loss: 0.5905 - val_accuracy: 0.8624
Epoch 17/20
255/255 [==============================] - 125s 492ms/step - loss: 0.1294 - accuracy: 0.9473 - val_loss: 0.8137 - val_accuracy: 0.8307
Epoch 18/20
255/255 [==============================] - 125s 491ms/step - loss: 0.1294 - accuracy: 0.9476 - val_loss: 0.5925 - val_accuracy: 0.8558
Epoch 19/20
255/255 [==============================] - 126s 493ms/step - loss: 0.1346 - accuracy: 0.9479 - val_loss: 0.7837 - val_accuracy: 0.8439
Epoch 20/20
255/255 [==============================] - 126s 493ms/step - loss: 0.1251 - accuracy: 0.9565 - val_loss: 0.7262 - val_accuracy: 0.8452
```

```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(test_X, batch_size=BS)
```

```
[INFO] evaluating network...
```

```
# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)
```

```
print(classification_report(test_Y.argmax(axis=1),predIdxs,target_names=lb.classes_))
```

```
              precision    recall  f1-score   support

   with_mask       0.98      0.70      0.82       383
without_mask       0.77      0.99      0.87       384

    accuracy                           0.85       767
   macro avg       0.88      0.85      0.84       767
weighted avg       0.88      0.85      0.84       767
```

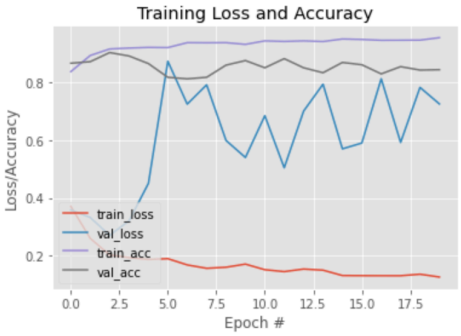
```
# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save("/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/mask_detector.model", save_format="h5")
```

```
[INFO] saving mask detector model...
```

```
# plot the training loss and accuracy
N = EPOCHS
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
```

```
<matplotlib.legend.Legend at 0x7ff02cb73250>
```




```
plt.savefig("/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/plot.png")
```

```
<Figure size 432x288 with 0 Axes>
```

```
# training of the machine is done now lets go for testing......
```

Testing :

```
# import the necessary packages

from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import cv2
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

```
def detect_and_predict_mask(frame, faceNet, maskNet):
    # grab the dimensions of the frame and then construct a blob
    # from it
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
        (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the face detections
    faceNet.setInput(blob)
    detections = faceNet.forward()
    print(detections.shape)

    # initialize our list of faces, their corresponding locations,
    # and the list of predictions from our face mask network
    faces = []
    locs = []
    preds = []

    # loop over the detections
    for i in range(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the detection
        confidence = detections[0, 0, i, 2]

        # filter out weak detections by ensuring the confidence is
        # greater than the minimum confidence
        if confidence > 0.5:
            # compute the (x, y)-coordinates of the bounding box for
            # the object
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # ensure the bounding boxes fall within the dimensions of
            # the frame
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
```

```
            # ensure the bounding boxes fall within the dimensions of
            # the frame
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

            # extract the face ROI, convert it from BGR to RGB channel
            # ordering, resize it to 224x224, and preprocess it
            face = frame[startY:endY, startX:endX]
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
            face = img_to_array(face)
            face = preprocess_input(face)

            # add the face and bounding boxes to their respective
            # lists
            faces.append(face)
            locs.append((startX, startY, endX, endY))

    # only make a predictions if at least one face was detected
    if len(faces) > 0:
        # for faster inference we'll make batch predictions on *all*
        # faces at the same time rather than one-by-one predictions
        # in the above `for` loop
        faces = np.array(faces, dtype="float32")
        preds = maskNet.predict(faces, batch_size=32)

    # return a 2-tuple of the face locations and their corresponding
    # locations
    return (locs, preds)
```

```
# load our serialized face detector model from disk

prototxtPath = "/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/face_detector/deploy.prototxt"
weightsPath = "/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/face_detector/res10_300x300_ssd_iter_140000.caffe
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
```

```
# load the face mask detector model from disk

maskNet = load_model("/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/pre trained file/mask_detector.model")
```

```
# initialize the video stream

print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
```

```
[INFO] starting video stream...
```

```
# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)

    # detect faces in the frame and determine if they are wearing a
    # face mask or not
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

    # loop over the detected face locations and their corresponding
    # locations
    for (box, pred) in zip(locs, preds):
        # unpack the bounding box and predictions
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred

        # determine the class label and color we'll use to draw
        # the bounding box and text
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

        # include the probability in the label
        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

        # display the label and bounding box rectangle on the output
        # frame
        cv2.putText(frame, label, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

    # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

# do a bit of cleanup

cv2.destroyAllWindows()
vs.stop()
```

VGG 16 :

Training :

```
from keras.layers import Input, Lambda, Dense, Flatten
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

```
IMAGE_SIZE = [224, 224] #for resizing all image
```

```
train_path = '/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/dataset/train'
valid_path = '/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/dataset/test'
```

```
# add preprocessing layer to the front of VGG
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 182s 3us/step
```

```
# don't train existing weights
for layer in vgg.layers:
  layer.trainable = False
```

```
 # useful for getting number of classes
folders = glob('/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/dataset/train/*')
```

```
# our layers - you can add more if you want
x = Flatten()(vgg.output)
```

```
# x = Dense(1000, activation='relu')(x)
prediction = Dense(len(folders), activation='softmax')(x)
```

```
# create a model object
model = Model(inputs=vgg.input, outputs=prediction)
```

```
# view the structure of the model
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 50178
=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

```
from keras.preprocessing.image import ImageDataGenerator
```

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
```

```
test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
training_set = train_datagen.flow_from_directory('/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/dataset/train',
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
```

```
Found 1178 images belonging to 2 classes.
```

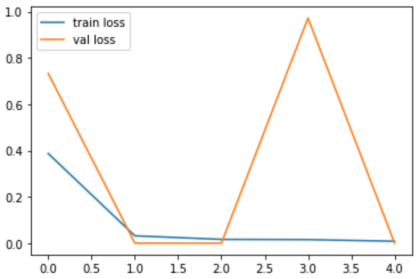
```
test_set = test_datagen.flow_from_directory('/Users/vishnudinesh/Desktop/6th sem btech project/face mask detection/dataset/test',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

```
Found 194 images belonging to 2 classes.
```

```
# fit the model
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=5,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

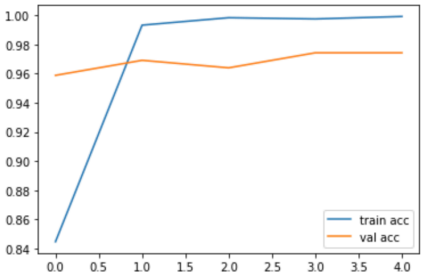
```
Epoch 1/5
37/37 [==============================] - 370s 10s/step - loss: 0.3855 - accuracy: 0.8447 - val_loss: 0.7329 - val_accuracy: 0.9588
Epoch 2/5
37/37 [==============================] - 366s 10s/step - loss: 0.0323 - accuracy: 0.9932 - val_loss: 7.9201e-04 - val_accuracy: 0.9691
Epoch 3/5
37/37 [==============================] - 365s 10s/step - loss: 0.0166 - accuracy: 0.9983 - val_loss: 2.3115e-04 - val_accuracy: 0.9639
Epoch 4/5
37/37 [==============================] - 366s 10s/step - loss: 0.0158 - accuracy: 0.9975 - val_loss: 0.9720 - val_accuracy: 0.9742
Epoch 5/5
37/37 [==============================] - 362s 10s/step - loss: 0.0087 - accuracy: 0.9992 - val_loss: 9.2983e-06 - val_accuracy: 0.9742
```

```
# loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')
```




```
<Figure size 432x288 with 0 Axes>
```

```
# accuracies
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```




```
<Figure size 432x288 with 0 Axes>
```

```
import tensorflow as tf
```

```
from keras.models import load_model
```

```
model.save('Mask_detector_model.h5')
```

Testing (image and video) :

```
import cv2
import numpy as np
from keras.models import load_model
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
```

```
Using TensorFlow backend.
```

```
# Function to load and prepare the image in right shape
def load_image(filename):
        # Load the image
    img = load_img(filename, target_size=(224, 224))
        # Convert the image to array
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    return img

# Load an image and predict the apparel class
filepath='/Users/vishnudinesh/Desktop/123.jpg'
img = load_image(filepath)
# Load the saved model
model = load_model('Mask_detector_model.h5')
# Predict the apparel class
prediction = model.predict(img)

#loading the cascades
face_cascade=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
image = cv2.imread(filepath)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    cv2.rectangle(image, (x,y), (x+w,y+h), (127,0,255), 2)

    if(prediction[0][0]>0.5):
        pred='Mask'
        #print(pred)
        cv2.putText(image,pred,(50,50),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
    else:
        pred='No Mask'
        #print(pred)
        cv2.putText(image,pred,(50,50),cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)
    cv2.imshow('Face Detection', image)
    cv2.waitKey(0) & 0xFF==ord('q')
#Map apparel category with the numerical class
'''if (prediction[0][0]>0.5):
  pred = "Mask"
else:
  pred = "No Mask"'''

print(pred)


cv2.destroyAllWindows()
```

```
import cv2
from keras.models import load_model
import numpy as np
from keras.preprocessing import image
from PIL import Image
```

```
#load the model
model=load_model('Mask_detector_model.h5')

#loading the cascades
face_cascade=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

#webcam face recognition
video_capture=cv2.VideoCapture(0)
while True:
    _,frame=video_capture.read()
    faces=face_cascade.detectMultiScale(frame,1.3,5)

    for (x,y,w,h) in faces:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,255),2)
        face=frame[y:y+h,x:x+w]
        cropped_face=face

        if type(face) is np.ndarray:
            face=cv2.resize(face,(224,224))
            im=Image.fromarray(face,'RGB')
            img_array=np.array(im)
            img_array=np.expand_dims(img_array,axis=0)
            pred=model.predict(img_array)
            print(pred)

            if(pred[0][0]>0.5):
                prediction='Mask'
                cv2.putText(cropped_face,prediction,(50,50),cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
            else:
                prediction='No Mask'
                cv2.putText(cropped_face,prediction,(50,50),cv2.FONT_HERSHEY_COMPLEX,1,(0,0,255),2)
        else:
            cv2.putText(frame,'No Face Found',(50,50),cv2.FONT_HERSHEY_COMPLEX,1,(255,0,0),2)

    cv2.imshow('Video',frame)
    if cv2.waitKey(1) & 0xFF==ord('q'):
        break
video_capture.release()
cv2.destroyAllWindows()
```

Facial emotion :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
from keras import models , Sequential
from keras.layers import Input, Conv2D, MaxPooling2D, Dropout, BatchNormalization, Dense, Flatten, Activation
from keras.optimizers import Adam
from keras.callbacks import Callback,EarlyStopping,ReduceLROnPlateau , ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from keras.utils.vis_utils import plot_model
import pydot
import graphviz
import visualkeras
```

```
data = pd.read_csv(r"D:\Project\fer2013.csv")
data.head()
```

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

```
data.shape
```

```
(35887, 3)
```

```
data["Usage"].value_counts()
```

```
Training       28709
PrivateTest     3589
PublicTest      3589
Name: Usage, dtype: int64
```

```
data["emotion"].value_counts()
```

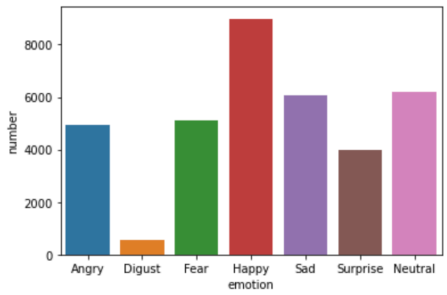
```
3    8989
6    6198
4    6077
2    5121
0    4953
5    4002
1     547
Name: emotion, dtype: int64
```

```
emotion_l = {0: 'Angry', 1: 'Digust', 2: 'Fear', 3: 'Happy', 4: 'Sad', 5: 'Surprise', 6: 'Neutral'}
emotion_counts = data['emotion'].value_counts(sort=False).reset_index()
emotion_counts.columns = ['emotion', 'number']
emotion_counts['emotion'] = emotion_counts['emotion'].map(emotion_l)
emotion_counts
```

	emotion	number
0	Angry	4953
1	Digust	547
2	Fear	5121
3	Happy	8989
4	Sad	6077
5	Surprise	4002
6	Neutral	6198

```
sns.barplot(emotion_counts['emotion'], emotion_counts['number'])
```

```
<AxesSubplot:xlabel='emotion', ylabel='number'>
```



```
training = data.loc[data["Usage"] == "Training"]
testing = data.loc[data["Usage"] == "PublicTest"]
validation = data.loc[data["Usage"] == "PrivateTest"]
```

```
training.shape , testing.shape , validation.shape
```

```
((28709, 3), (3589, 3), (3589, 3))
```

```
x_train = training["pixels"].str.split(" ").tolist()
x_train = np.uint8(x_train)
x_train = x_train.reshape((28709, 48, 48, 1))
x_train = x_train.astype("float32") / 255

y_train = training["emotion"]
y_train = to_categorical(y_train)

x_test = testing["pixels"].str.split(" ").tolist()
x_test = np.uint8(x_test)
x_test = x_test.reshape((3589, 48, 48, 1))
x_test = x_test.astype("float32") / 255

y_test = testing["emotion"]
y_test = to_categorical(y_test)

x_val = validation["pixels"].str.split(" ").tolist()
x_val = np.uint8(x_val)
x_val = x_val.reshape((3589, 48, 48, 1))
x_val = x_val.astype("float32") / 255

y_val = validation["emotion"]
y_val = to_categorical(y_val)
```

```
x_train.shape , y_train.shape
```

```
((28709, 48, 48, 1), (28709, 7))
```

```
x_test.shape , y_test.shape
```

```
((3589, 48, 48, 1), (3589, 7))
```

```
x_val.shape , y_val.shape
```

```
((3589, 48, 48, 1), (3589, 7))
```

```
x_test.dtype , y_test.dtype
```

```
(dtype('float32'), dtype('float32'))
```

```
plt.figure(0, figsize=(12,6))
for i in range(1, 13):
    plt.subplot(3,4,i)
    plt.imshow(x_train[i, :, :, 0], cmap="gray")

plt.tight_layout()
plt.show()
```

```
model = Sequential()

# Block-1: The First Convolutional Block

model.add(Conv2D(filters=32,kernel_size=(3,3),padding='same',kernel_initializer='he_normal',activation="relu",input_shape=(48,48, 1)))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same',kernel_initializer='he_normal',activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

# Block-2: The Second Convolutional Block

model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same',kernel_initializer='he_normal',activation="relu"))
model.add(BatchNormalization())
model.add(Conv2D(filters=64,kernel_size=(3,3),padding='same',kernel_initializer='he_normal',activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

# Block-3: The Third Convolutional Block

model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same',kernel_initializer='he_normal',activation="relu"))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same',kernel_initializer='he_normal',activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

# Block-4: The Fully Connected Block

model.add(Flatten())
model.add(Dense(64, activation="relu", kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Block-5: The Output Block

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

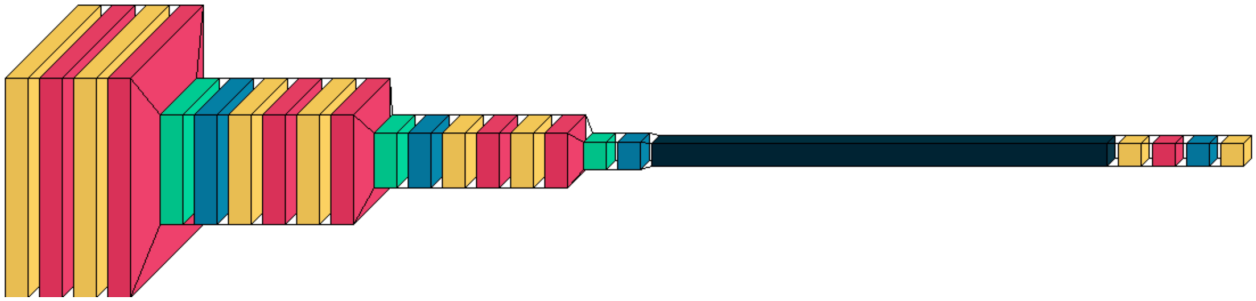
```
x_train.shape
```

```
(28709, 48, 48, 1)
```

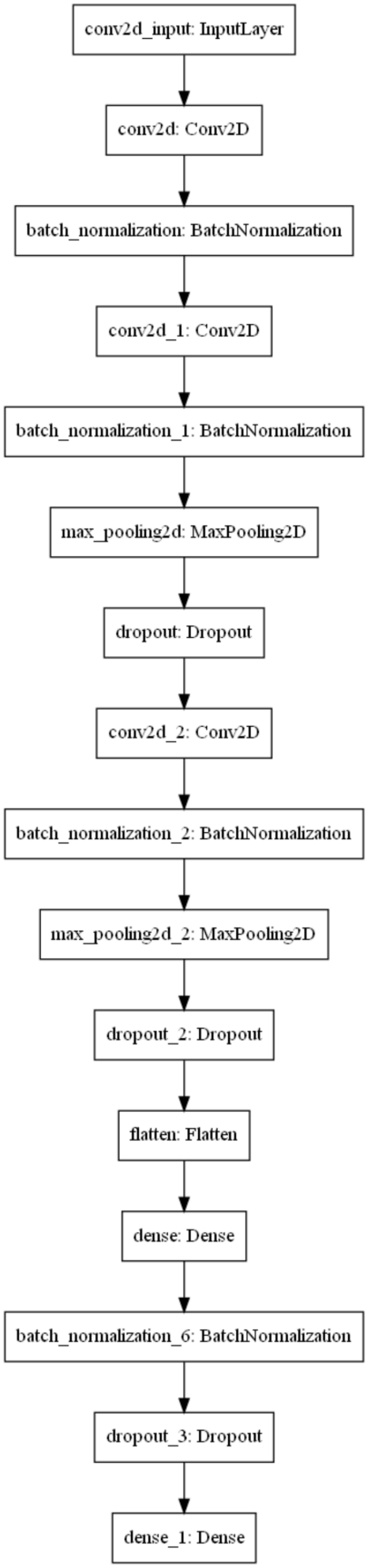
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 48, 48, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 48, 48, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 48, 48, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 24, 24, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 24, 24, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 24, 24, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 128)       147584
_________________________________________________________________
batch_normalization_5 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 64)                294976
_________________________________________________________________
batch_normalization_6 (Batch (None, 64)                256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 455
=================================================================
Total params: 583,911
Trainable params: 582,887
Non-trainable params: 1,024
```

```
visualkeras.layered_view(model)
```



```
plot_model(model, to_file='model.png', show_layer_names=True)
```



```
model.compile(optimizer = Adam(lr=0.001, beta_1=0.9,
                               beta_2=0.999, epsilon=1e-7),
              loss = "categorical_crossentropy", metrics = ["accuracy"])
```

```
lr_reducer = ReduceLROnPlateau(monitor='val_accuracy', factor=0.9, patience=3, verbose=1)
early_stopping = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=8, verbose=1, mode='auto')
checkpointer = ModelCheckpoint("emotions2.h5", monitor='val_accuracy', verbose=1, save_best_only=True)
```

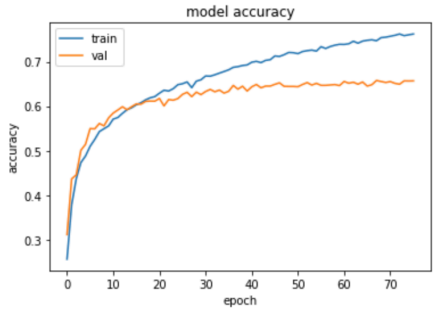
```
history = model.fit(x_train, y_train, batch_size = 64, epochs = 100,
                    verbose=1,validation_data = (np.array(x_test),np.array(y_test)) ,
                    callbacks = [early_stopping , lr_reducer,checkpointer] ,
                    shuffle=True)
```

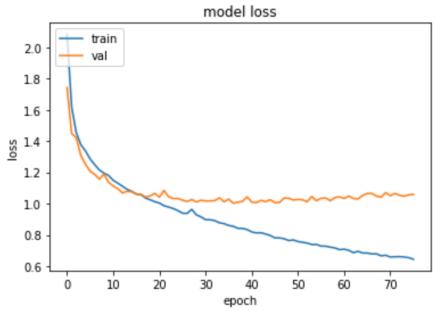
```
Epoch 1/100
449/449 [==============================] - 317s 703ms/step - loss: 2.3856 - accuracy: 0.2101 - val_loss: 1.7418 - val_accuracy: 0.3115

Epoch 00001: val_accuracy improved from -inf to 0.31151, saving model to emotions2.h5
Epoch 2/100
449/449 [==============================] - 317s 707ms/step - loss: 1.6747 - accuracy: 0.3637 - val_loss: 1.4481 - val_accuracy: 0.4366

Epoch 00002: val_accuracy improved from 0.31151 to 0.43661, saving model to emotions2.h5
Epoch 3/100
449/449 [==============================] - 322s 717ms/step - loss: 1.4753 - accuracy: 0.4276 - val_loss: 1.4211 - val_accuracy: 0.4458

Epoch 00003: val_accuracy improved from 0.43661 to 0.44581, saving model to emotions2.h5
Epoch 4/100
449/449 [==============================] - 315s 702ms/step - loss: 1.4014 - accuracy: 0.4599 - val_loss: 1.3073 - val_accuracy: 0.5010

Epoch 00004: val_accuracy improved from 0.44581 to 0.50098, saving model to emotions2.h5
Epoch 5/100
449/449 [==============================] - 320s 714ms/step - loss: 1.3605 - accuracy: 0.4767 - val_loss: 1.2520 - val_accuracy: 0.5143

Epoch 00005: val_accuracy improved from 0.50098 to 0.51435, saving model to emotions2.h5
Epoch 6/100
449/449 [==============================] - 313s 697ms/step - loss: 1.2816 - accuracy: 0.5115 - val_loss: 1.2072 - val_accuracy: 0.5497

Epoch 00006: val_accuracy improved from 0.51435 to 0.54974, saving model to emotions2.h5
Epoch 7/100
449/449 [==============================] - 322s 718ms/step - loss: 1.2507 - accuracy: 0.5250 - val_loss: 1.1876 - val_accuracy: 0.5492

Epoch 00007: val_accuracy did not improve from 0.54974
Epoch 8/100
449/449 [==============================] - 308s 686ms/step - loss: 1.2134 - accuracy: 0.5401 - val_loss: 1.1570 - val_accuracy: 0.5612

Epoch 00008: val_accuracy improved from 0.54974 to 0.56116, saving model to emotions2.h5
Epoch 9/100
449/449 [==============================] - 314s 700ms/step - loss: 1.1950 - accuracy: 0.5496 - val_loss: 1.1894 - val_accuracy: 0.5559

Epoch 00009: val_accuracy did not improve from 0.56116
Epoch 10/100
449/449 [==============================] - 305s 679ms/step - loss: 1.1762 - accuracy: 0.5584 - val_loss: 1.1366 - val_accuracy: 0.5737

Epoch 00010: val_accuracy improved from 0.56116 to 0.57370, saving model to emotions2.h5
Epoch 11/100
449/449 [==============================] - 302s 673ms/step - loss: 1.1523 - accuracy: 0.5686 - val_loss: 1.1128 - val_accuracy: 0.5846

Epoch 00011: val_accuracy improved from 0.57370 to 0.58456, saving model to emotions2.h5
Epoch 12/100
449/449 [==============================] - 319s 710ms/step - loss: 1.1249 - accuracy: 0.5757 - val_loss: 1.0956 - val_accuracy: 0.5915

Epoch 00012: val_accuracy improved from 0.58456 to 0.59153, saving model to emotions2.h5
Epoch 13/100
449/449 [==============================] - 320s 713ms/step - loss: 1.1160 - accuracy: 0.5784 - val_loss: 1.0683 - val_accuracy: 0.5988

Epoch 00013: val_accuracy improved from 0.59153 to 0.59877, saving model to emotions2.h5
Epoch 14/100
449/449 [==============================] - 341s 759ms/step - loss: 1.0890 - accuracy: 0.5949 - val_loss: 1.0790 - val_accuracy: 0.5921
```

••••••••••••••

```
Epoch 00070: val_accuracy did not improve from 0.65784
Epoch 71/100
449/449 [==============================] - 294s 656ms/step - loss: 0.6497 - accuracy: 0.7588 - val_loss: 1.0510 - val_accuracy: 0.6553

Epoch 00071: ReduceLROnPlateau reducing learning rate to 0.0003486784757114947.

Epoch 00071: val_accuracy did not improve from 0.65784
Epoch 72/100
449/449 [==============================] - 295s 658ms/step - loss: 0.6486 - accuracy: 0.7632 - val_loss: 1.0655 - val_accuracy: 0.6514

Epoch 00072: val_accuracy did not improve from 0.65784
Epoch 73/100
449/449 [==============================] - 301s 670ms/step - loss: 0.6631 - accuracy: 0.7591 - val_loss: 1.0533 - val_accuracy: 0.6495

Epoch 00073: val_accuracy did not improve from 0.65784
Epoch 74/100
449/449 [==============================] - 313s 697ms/step - loss: 0.6535 - accuracy: 0.7616 - val_loss: 1.0474 - val_accuracy: 0.6570

Epoch 00074: ReduceLROnPlateau reducing learning rate to 0.00031381062290165574.

Epoch 00074: val_accuracy did not improve from 0.65784
Epoch 75/100
449/449 [==============================] - 294s 655ms/step - loss: 0.6542 - accuracy: 0.7587 - val_loss: 1.0563 - val_accuracy: 0.6567

Epoch 00075: val_accuracy did not improve from 0.65784
Epoch 76/100
449/449 [==============================] - 294s 654ms/step - loss: 0.6423 - accuracy: 0.7637 - val_loss: 1.0594 - val_accuracy: 0.6570

Epoch 00076: val_accuracy did not improve from 0.65784
Epoch 00076: early stopping
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```





Testing :

```
from tensorflow.keras.models import load_model
import tensorflow as tf
from time import sleep
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import model_from_json
from tensorflow.keras.preprocessing import image
import cv2
import numpy as np
```

```
emotion_model =load_model('emotions2.h5')

face_haar_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
class_labels = ['Angry','Disgust','Fear','Happy','Sad', 'Surprise','Neutral']
cap = cv2.VideoCapture(0)
while True:
    # Grab a single frame of video
    ret, frame = cap.read()
    labels = []
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    faces = face_haar_cascade.detectMultiScale(gray,1.3,5)

    for (x,y,w,h) in faces:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h,x:x+w]
        roi_gray = cv2.resize(roi_gray,(48,48),interpolation=cv2.INTER_AREA)


        if np.sum([roi_gray])!=0:
            roi = roi_gray.astype('float')/255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi,axis=0)

            preds = emotion_model.predict(roi)[0]
            label=class_labels[preds.argmax()]
            percen = f'{round(np.max(preds)*100)}%'
            print("\nlabel:{0} {1}%".format(label,round(np.max(preds)*100)))
            label_position = (x,y-10)
        cv2.putText(frame,f'{label}:{percen}',label_position,cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)

    cv2.imshow('Emotion Detector',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Implementation for both face mask and face emotion (for mixing) :
Emotion :

```
import pickle
from tqdm import tqdm
import cv2
import os
from keras.preprocessing import image
import numpy as np
import pandas as pd
from tensorflow.keras.utils import to_categorical
from sklearn.utils import shuffle as sk_shuffle
from sklearn.model_selection import train_test_split
from keras.applications.mobilenet_v2 import preprocess_input
from keras.applications.vgg16 import preprocess_input as vgg16_preprocess
```

```
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import AveragePooling2D, Dropout, Flatten, BatchNormalization,Dense, Input, GlobalAveragePooling2D, Conv2D, MaxPooling2D
from keras import regularizers
from keras.models import Model, Sequential
from keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
import pickle
import os
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import cv2
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
```

```
def get_image_value(path, dim, bw, model_type):
    '''This function will read an image and convert to a specified version and resize depending on which algorithm is being used.  If edge is s
    img = image.load_img(path, target_size = dim)
    img = image.img_to_array(img)
    if bw == True:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        if model_type.upper() != 'Normal':
            img = np.stack((img,)*3, axis =-1)
        else:
            img = img.reshape(img.shape[0], img.shape[1],1)


    if model_type.upper() == 'MOBILENET':
        img = mobile_preprocess(img)
        return img
    elif model_type.upper() == 'VGG16':
        img = vgg16_preprocess(img)
        return img
    return img/255
```

```
def get_emotion_classes(class_type):
    angry_paths = [f'D:/Project/FER/{class_type}/angry/{i}' for i in os.listdir(f'D:/Project/FER/{class_type}/angry')]
    angry_labels = [0 for i in range(len(angry_paths))]

    happy_paths = [f'D:/Project/FER/{class_type}/happy/{i}' for i in os.listdir(f'D:/Project/FER/{class_type}/happy')]
    happy_labels = [1 for i in range(len(happy_paths))]

    neutral_paths = [f'D:/Project/FER/{class_type}/neutral/{i}' for i in os.listdir(f'D:/Project/FER/{class_type}/neutral')]
    neutral_labels = [2 for i in range(len(neutral_paths))]


    labels = np.array(angry_labels + happy_labels + neutral_labels)

    print(f'{class_type.upper()} Value Count')
    print(pd.Series(labels).value_counts())
    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    labels = to_categorical(labels)
    paths = np.array(angry_paths + happy_paths + neutral_paths)
    paths, labels = sk_shuffle(paths, labels)
    return paths, labels
#0: angry  || 1: happy || 2: neutral

def get_emotion_splits(dim, pick_name, model_type = 'normal',bw = False,max_values = 6000):

    #Train

    train_paths, train_labels = get_emotion_classes('train')
    test_paths, test_labels = get_emotion_classes('test')

    train_images = np.array([get_image_value(i, dim, bw, model_type) for i in tqdm(train_paths, desc = 'Getting Emotion Train Images')])
    test_images = np.array([get_image_value(i, dim, bw, model_type) for i in tqdm(test_paths, desc = 'Getting Emotion Test Images')])

    tts = (train_images, test_images, train_labels, test_labels)



    return tts
dim = (48,48)
tts = get_emotion_splits(dim, pick_name = 'normal', bw = False)
```

```
Getting Emotion Train Images:   2%||          | 252/16175 [00:00<00:06, 2502.10it/s]
TRAIN Value Count
1    7215
2    4965
0    3995
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
TEST Value Count
1    1774
2    1233
0     958
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Getting Emotion Train Images: 100%|██████████| 16175/16175 [00:06<00:00, 2676.19it/s]
Getting Emotion Test Images: 100%|██████████| 3965/3965 [00:01<00:00, 2623.77it/s]
```

```
x_train, x_test, y_train, y_test = get_emotion_splits(dim = (48,48), pick_name = 'normal', bw = True)
print(x_train.shape, x_test.shape)
print(y_train.shape, y_test.shape)
```

```
Getting Emotion Train Images:   3%|▏         | 462/16175 [00:00<00:07, 2184.28it/s]
TRAIN Value Count
1    7215
2    4965
0    3995
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
TEST Value Count
1    1774
2    1233
0     958
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Getting Emotion Train Images: 100%|██████████| 16175/16175 [00:07<00:00, 2271.50it/s]
Getting Emotion Test Images: 100%|██████████| 3965/3965 [00:01<00:00, 2214.82it/s]
(16175, 48, 48, 3) (3965, 48, 48, 3)
(16175, 3) (3965, 3)
```

```
 x_train.shape
```

```
(16175, 48, 48, 3)
```

```
x_train.shape[1],x_train.shape[2],x_train.shape[3]
```

```
(48, 48, 3)
```

```
def get_conv_model(dim):

    model = Sequential()

    model.add(Conv2D(32, kernel_size = (3,3), activation = 'relu', input_shape = dim))
    model.add(Conv2D(64, kernel_size = (3,3), activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))
    model.add(Dropout(.25))

    model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(3, activation = 'softmax', name = 'Output'))
    model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
    return model
```

```
early_stopping = EarlyStopping(monitor='val_loss', verbose = 1, patience=5, min_delta = .00075)
model_checkpoint = ModelCheckpoint(f'N_Emotions_only3.h5', verbose = 1, save_best_only=True,
                                   monitor = 'val_loss')
lr_plat = ReduceLROnPlateau(patience = 5, mode = 'min')
epochs = 2000
batch_size = 32
augment = False

dim = (x_train.shape[1], x_train.shape[2], x_train.shape[3])
cnn = get_conv_model(dim =dim)

if augment == True:
    train_aug =ImageDataGenerator(rotation_range = 40, width_shift_range = .2, height_shift_range = .2,
                                                        horizontal_flip = True, shear_range = .15,
                                  fill_mode = 'nearest', zoom_range = .15)
    # augmentation.fit(x_train)
    cnn_history = cnn.fit_generator(train_aug.flow(x_train, y_train, batch_size = batch_size),
                epochs = epochs,
         callbacks = [early_stopping, model_checkpoint, lr_plat], validation_data = (x_test, y_test), verbose= 1)

else:
    cnn_history = cnn.fit(x_train, y_train, batch_size = batch_size,
                epochs = epochs,
         callbacks = [early_stopping, model_checkpoint, lr_plat], validation_data = (x_test, y_test), verbose= 1)
```

```
Epoch 1/2000
506/506 [==============================] - 108s 188ms/step - loss: 1.0611 - accuracy: 0.4498 - val_loss: 0.8656 - val_accuracy: 0.5997

Epoch 00001: val_loss improved from inf to 0.86558, saving model to N_Emotions_only3.h5
Epoch 2/2000
506/506 [==============================] - 105s 207ms/step - loss: 0.8350 - accuracy: 0.6154 - val_loss: 0.6964 - val_accuracy: 0.6913

Epoch 00002: val_loss improved from 0.86558 to 0.69637, saving model to N_Emotions_only3.h5
Epoch 3/2000
506/506 [==============================] - 103s 203ms/step - loss: 0.7108 - accuracy: 0.6911 - val_loss: 0.6609 - val_accuracy: 0.7117

Epoch 00003: val_loss improved from 0.69637 to 0.66090, saving model to N_Emotions_only3.h5
Epoch 4/2000
506/506 [==============================] - 102s 201ms/step - loss: 0.6607 - accuracy: 0.7095 - val_loss: 0.5946 - val_accuracy: 0.7435

Epoch 00004: val_loss improved from 0.66090 to 0.59465, saving model to N_Emotions_only3.h5
Epoch 5/2000
506/506 [==============================] - 109s 216ms/step - loss: 0.6166 - accuracy: 0.7412 - val_loss: 0.5853 - val_accuracy: 0.7473
Epoch 00005: val_loss improved from 0.59465 to 0.58528, saving model to N_Emotions_only3.h5
Epoch 6/2000
506/506 [==============================] - 113s 223ms/step - loss: 0.5844 - accuracy: 0.7540 - val_loss: 0.5708 - val_accuracy: 0.7498

Epoch 00006: val_loss improved from 0.58528 to 0.57076, saving model to N_Emotions_only3.h5
Epoch 7/2000
506/506 [==============================] - 104s 206ms/step - loss: 0.5637 - accuracy: 0.7601 - val_loss: 0.5474 - val_accuracy: 0.7654

Epoch 00007: val_loss improved from 0.57076 to 0.54741, saving model to N_Emotions_only3.h5
Epoch 8/2000
506/506 [==============================] - 102s 201ms/step - loss: 0.5336 - accuracy: 0.7704 - val_loss: 0.5615 - val_accuracy: 0.7612

Epoch 00008: val_loss did not improve from 0.54741
Epoch 9/2000
506/506 [==============================] - 108s 213ms/step - loss: 0.5014 - accuracy: 0.7942 - val_loss: 0.5442 - val_accuracy: 0.7776

Epoch 00009: val_loss improved from 0.54741 to 0.54423, saving model to N_Emotions_only3.h5
Epoch 10/2000
506/506 [==============================] - 108s 214ms/step - loss: 0.4923 - accuracy: 0.7943 - val_loss: 0.5715 - val_accuracy: 0.7574

Epoch 00010: val_loss did not improve from 0.54423
Epoch 11/2000
506/506 [==============================] - 111s 220ms/step - loss: 0.4747 - accuracy: 0.8056 - val_loss: 0.5403 - val_accuracy: 0.7793

Epoch 00011: val_loss improved from 0.54423 to 0.54033, saving model to N_Emotions_only3.h5
Epoch 12/2000
506/506 [==============================] - 103s 203ms/step - loss: 0.4619 - accuracy: 0.8100 - val_loss: 0.5261 - val_accuracy: 0.7879

Epoch 00012: val_loss improved from 0.54033 to 0.52609, saving model to N_Emotions_only3.h5
Epoch 13/2000
506/506 [==============================] - 104s 206ms/step - loss: 0.4443 - accuracy: 0.8178 - val_loss: 0.5623 - val_accuracy: 0.7637

Epoch 00013: val_loss did not improve from 0.52609
Epoch 14/2000
506/506 [==============================] - 113s 223ms/step - loss: 0.4323 - accuracy: 0.8233 - val_loss: 0.5216 - val_accuracy: 0.7796

Epoch 00014: val_loss improved from 0.52609 to 0.52158, saving model to N_Emotions_only3.h5
Epoch 15/2000
506/506 [==============================] - 120s 238ms/step - loss: 0.4094 - accuracy: 0.8350 - val_loss: 0.5363 - val_accuracy: 0.7869

Epoch 00015: val_loss did not improve from 0.52158
Epoch 16/2000
506/506 [==============================] - 114s 225ms/step - loss: 0.3869 - accuracy: 0.8387 - val_loss: 0.5455 - val_accuracy: 0.7770

Epoch 00016: val_loss did not improve from 0.52158
Epoch 17/2000
506/506 [==============================] - 114s 225ms/step - loss: 0.3818 - accuracy: 0.8425 - val_loss: 0.5327 - val_accuracy: 0.7897

Epoch 00017: val_loss did not improve from 0.52158
Epoch 18/2000
506/506 [==============================] - 106s 210ms/step - loss: 0.3725 - accuracy: 0.8490 - val_loss: 0.5664 - val_accuracy: 0.7753

Epoch 00018: val_loss did not improve from 0.52158
Epoch 19/2000
506/506 [==============================] - 107s 212ms/step - loss: 0.3559 - accuracy: 0.8585 - val_loss: 0.5466 - val_accuracy: 0.7834

Epoch 00019: val_loss did not improve from 0.52158
Epoch 00019: early stopping
```

Mask :

```
import pickle
from tqdm import tqdm
import cv2
import os
from keras.preprocessing import image
import numpy as np
import pandas as pd
from tensorflow.keras.utils import to_categorical
from sklearn.utils import shuffle as sk_shuffle
from sklearn.model_selection import train_test_split
from keras.applications.mobilenet_v2 import preprocess_input
from keras.applications.vgg16 import preprocess_input as vgg16_preprocess
```

```
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import AveragePooling2D, Dropout, Flatten, BatchNormalization,Dense, Input, GlobalAveragePooling2D, Conv2D, MaxPooling2D
from keras import regularizers
from keras.models import Model, Sequential
from keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
import pickle
import os
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import cv2
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
#from PyFunctions import Viz
#from PyFunctions import Functions as func
```

```python
def get_image_value(path, dim, bw, model_type):
    '''This function will read an image and convert to a specified version and resize depending on which algorithm is being used.  If edge is
    img = image.load_img(path, target_size = dim)
    img = image.img_to_array(img)
    if bw == True:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = img.reshape(img.shape[0], img.shape[1],1)
    if model_type == 'mobilenet':
        img = preprocess_input(img)
        return img
    return img/255
```

```python
def get_mask_classes(class_type):
    mask_paths = [f'/Face_Mask_Dataset/{class_type}/WithMask/{i}' for i in os.listdir(f'/Face_Mask_Dataset/{class_type}/WithMask')]
    mask_labels = [1 for i in range(len(mask_paths))]

    nomask_paths = [f'/Face_Mask_Dataset/{class_type}/WithoutMask/{i}' for i in os.listdir(f'/Face_Mask_Dataset/{class_type}/WithoutMask')]
    nomask_labels = [0 for i in range(len(nomask_paths))]

    labels = np.array(mask_labels + nomask_labels)
    print(f'{class_type.upper()} Value Counts')
    print(pd.Series(labels).value_counts())
    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    paths = np.array(mask_paths + nomask_paths)
    labels = to_categorical(labels)
    paths, labels = sk_shuffle(paths, labels)
    return paths, labels

def get_mask_splits(dim, pick_name, model_type = 'Mobilenet', bw = False):

    #Train Set
    train_paths, train_labels = get_mask_classes('Train')
    train_images = np.array([get_image_value(i, dim, bw, model_type) for i in tqdm(train_paths, desc = 'Getting Train Images')])
    train_dict = dict(images = train_images, labels = train_labels)

    #Test Set
    test_paths, test_labels = get_mask_classes('Test')
    test_images = np.array([get_image_value(i, dim, bw, model_type) for i in tqdm(test_paths, desc = 'Getting Test Images')])
    test_images, test_labels = sk_shuffle(test_images, test_labels)

    #Validation Set
    val_paths, val_labels = get_mask_classes('Validation')
    val_images = np.array([get_image_value(i, dim, bw, model_type) for i in tqdm(val_paths, desc = 'Getting Validation Images')])
    val_images, val_labels = sk_shuffle(val_images, val_labels)

    tts = train_images, test_images, train_labels, test_labels, val_images, val_labels


    return tts


dim = (244,244)
tts = get_mask_splits(dim, pick_name = 'Normal1',model_type = 'Normal', bw = False )
```

```
Getting Train Images:   0%|          | 31/10000 [00:00<00:33, 301.37it/s]
TRAIN Value Counts
0    5000
1    5000
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Getting Train Images: 100%|██████████| 10000/10000 [00:27<00:00, 363.95it/s]
Getting Test Images:   4%|▍         | 36/992 [00:00<00:02, 353.24it/s]
TEST Value Counts
0    509
1    483
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Getting Test Images: 100%|██████████| 992/992 [00:02<00:00, 391.00it/s]
Getting Validation Images:   4%|▍         | 35/800 [00:00<00:02, 346.30it/s]
VALIDATION Value Counts
0    400
1    400
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Getting Validation Images: 100%|██████████| 800/800 [00:01<00:00, 424.51it/s]
```

```python
dim = (224,224)

x_train, x_test, y_train, y_test, x_val, y_val = get_mask_splits(dim,'Normal1',bw=True)
```

```
Getting Train Images:   0%|          | 39/10000 [00:00<00:25, 389.16it/s]
TRAIN Value Counts
0    5000
1    5000
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Getting Train Images: 100%|██████████| 10000/10000 [00:23<00:00, 422.09it/s]
Getting Test Images:   5%|▌         | 54/992 [00:00<00:01, 539.81it/s]
TEST Value Counts
0    509
1    483
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Getting Test Images: 100%|██████████| 992/992 [00:01<00:00, 636.20it/s]
Getting Validation Images:   7%|▋         | 58/800 [00:00<00:01, 568.37it/s]
VALIDATION Value Counts
0    400
1    400
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Getting Validation Images: 100%|██████████| 800/800 [00:01<00:00, 624.61it/s]
```

```
def get_mobilenet(dim):
    model = Sequential()
    optimizer = Adam(lr = .0005)
    baseModel = MobileNetV2(weights="imagenet", include_top=False,
        input_tensor=Input(shape=dim))

    model.add(baseModel)
    model.add(AveragePooling2D(pool_size=(7, 7)))
    model.add(Flatten(name="flatten"))
    model.add(Dense(256, activation="relu"))
    model.add(Dropout(0.6))
    model.add(Dense(2, activation="sigmoid", name = 'Output'))


    for layer in baseModel.layers:
        layer.trainable = False

    model.compile(loss = 'binary_crossentropy', optimizer = optimizer, metrics = ['accuracy'])
    return model
```

```
x_train.shape
```

```
(10000, 224, 224, 1)
```

```
x_train.shape[1] , x_train.shape[2],x_train.shape[3]
```

```
(224, 224, 1)
```

```
dim = (224,224)

x_train, x_test, y_train, y_test, x_val, y_val = func.get_mask_splits(dim)
```

```
early_stopping = EarlyStopping(monitor='val_loss', verbose = 1, patience=5, min_delta = .00075)
model_checkpoint = ModelCheckpoint('Mobilenet_Masks_1.h5', verbose = 1, save_best_only=True,
                                   monitor = 'val_loss')
lr_plat = ReduceLROnPlateau(patience = 5, mode = 'min')
epochs = 2000
batch_size = 64


dim = (x_train.shape[1], x_train.shape[2], x_train.shape[3])
mobilenet = get_mobilenet(dim=dim)


augmentation =ImageDataGenerator(rotation_range = 20, width_shift_range = .2, height_shift_range = .2,
                                                    horizontal_flip = True, shear_range = .15,
                                 fill_mode = 'nearest', zoom_range = .15)
augmentation.fit(x_train)
mobilenet_history = mobilenet.fit_generator(augmentation.flow(x_train, y_train, batch_size = batch_size),
            epochs = epochs,
     callbacks = [early_stopping, model_checkpoint, lr_plat], validation_data = (x_test, y_test), verbose= 1)
```

•••••••••

```
Epoch 00005: val_loss improved from 0.14802 to 0.13957, saving model to ModelWeights/Mobilenet_Masks.h5
Epoch 6/2000
157/157 [==============================] - 90s 573ms/step - loss: 0.0641 - acc: 0.9779 - val_loss: 0.2344 - val_acc: 0.9047

Epoch 00006: val_loss did not improve from 0.13957
Epoch 7/2000
157/157 [==============================] - 93s 592ms/step - loss: 0.0622 - acc: 0.9785 - val_loss: 0.1219 - val_acc: 0.9395

Epoch 00007: val_loss improved from 0.13957 to 0.12195, saving model to ModelWeights/Mobilenet_Masks.h5
Epoch 8/2000
157/157 [==============================] - 91s 578ms/step - loss: 0.0565 - acc: 0.9798 - val_loss: 0.0544 - val_acc: 0.9798

Epoch 00008: val_loss improved from 0.12195 to 0.05442, saving model to ModelWeights/Mobilenet_Masks.h5
Epoch 9/2000
157/157 [==============================] - 90s 574ms/step - loss: 0.0582 - acc: 0.9790 - val_loss: 0.0747 - val_acc: 0.9677

Epoch 00009: val_loss did not improve from 0.05442
Epoch 10/2000
157/157 [==============================] - 91s 580ms/step - loss: 0.0571 - acc: 0.9801 - val_loss: 0.0426 - val_acc: 0.9869

Epoch 00010: val_loss improved from 0.05442 to 0.04260, saving model to ModelWeights/Mobilenet_Masks.h5
Epoch 11/2000
157/157 [==============================] - 91s 580ms/step - loss: 0.0512 - acc: 0.9801 - val_loss: 0.0371 - val_acc: 0.9894

Epoch 00011: val_loss improved from 0.04260 to 0.03707, saving model to ModelWeights/Mobilenet_Masks.h5
Epoch 12/2000
157/157 [==============================] - 91s 581ms/step - loss: 0.0520 - acc: 0.9802 - val_loss: 0.0797 - val_acc: 0.9662

Epoch 00012: val_loss did not improve from 0.03707
Epoch 13/2000
157/157 [==============================] - 91s 577ms/step - loss: 0.0468 - acc: 0.9832 - val_loss: 0.0659 - val_acc: 0.9698

Epoch 00013: val_loss did not improve from 0.03707
Epoch 14/2000
157/157 [==============================] - 90s 575ms/step - loss: 0.0448 - acc: 0.9838 - val_loss: 0.0643 - val_acc: 0.9728

Epoch 00014: val_loss did not improve from 0.03707
Epoch 15/2000
157/157 [==============================] - 90s 576ms/step - loss: 0.0494 - acc: 0.9823 - val_loss: 0.0421 - val_acc: 0.9849

Epoch 00015: val_loss did not improve from 0.03707
Epoch 16/2000
157/157 [==============================] - 91s 579ms/step - loss: 0.0518 - acc: 0.9815 - val_loss: 0.0518 - val_acc: 0.9824

Epoch 00016: val_loss did not improve from 0.03707
Epoch 00016: early stopping
```

Testing :

```
import cv2
import numpy as np
from keras.models import load_model
from keras.applications.mobilenet_v2 import preprocess_input as mobile_preprocess
from keras.applications.vgg16 import preprocess_input as vgg_preprocess


#live video

classifier = cv2.CascadeClassifier(f'haarcascade_frontalface_default.xml')
mask_model = load_model(f'Mobilenet_Masks1.h5')
emotion_model = load_model(f'emotions/N_Emotions_only3.h5')

emotion_dim = (48,48)
mask_dim = (224,224)

mask_dict = {0: 'No Mask', 1: 'Mask'}
mask_dict_color = {0: (0,0,255), 1: (0,255,0)} #colors for bounding boxes

emotion_dict = {0: 'Angry', 1: 'Happy',2:'Neutral'}
vid_frames = []
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    frame = cv2.flip(frame, 1,1)
    clone = frame.copy()
    bboxes = classifier.detectMultiScale(clone)
    for i in bboxes:
        x, y, width, height = i[0], i[1], i[2], i[3]
        x2, y2 = x+ width, y+height
        mask_roi = clone[y:y2, x:x2]
        emotion_roi = mask_roi.copy()
        emotion_roi = cv2.resize(emotion_roi, emotion_dim, interpolation = cv2.INTER_CUBIC)
        mask_roi = cv2.resize(mask_roi, mask_dim, interpolation = cv2.INTER_CUBIC)

        #preprocess mask_input
        mask_roi= mobile_preprocess(mask_roi)

        #preprocess emotion input
        emotion_roi = emotion_roi/255

        #resize emotion and mask to feed into nn
        mask_roi = mask_roi.reshape(1, mask_roi.shape[0], mask_roi.shape[1], mask_roi.shape[2])
        emotion_roi = emotion_roi.reshape(1, emotion_roi.shape[0], emotion_roi.shape[1], emotion_roi.shape[2])


         #mask predictions
         mask_predict = mask_model.predict(mask_roi)[0]
         mask_idx = np.argmax(mask_predict)
         mask_conf = f'{round(np.max(mask_predict)*100)}%'
         mask_cat = mask_dict[mask_idx]
         mask_color = mask_dict_color[mask_idx]
         if mask_idx == 0: #if there is no mask detected --> move onto emotions
             #emotion predictions
             emotion_predict = emotion_model.predict(emotion_roi)[0]
             emotion_idx = np.argmax(emotion_predict)
             emotion_cat = emotion_dict[emotion_idx]
             emotion_conf = f'{round(np.max(emotion_predict)*100)}%'
             cv2.putText(clone, f'{mask_cat}: {mask_conf}         {emotion_cat}: {emotion_conf}', (x, y-15), cv2.FONT_HERSHEY_SIMPLEX, .5, mask_color, 2)
             cv2.rectangle(clone, (x,y), (x2,y2), mask_color, 1)
             continue
         cv2.putText(clone, f'{mask_cat}: {mask_conf}', (x, y-15), cv2.FONT_HERSHEY_SIMPLEX, .5, mask_color, 2)
         cv2.rectangle(clone, (x,y), (x2,y2), mask_color, 1)

    cv2.imshow('Face Mask Emotion Detector', clone)
    vid_frames.append(clone)

    if cv2.waitKey(1) & 0xFF ==ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

```
mask_roi.shape
```

```
(1, 224, 224, 3)
```

```
emotion_roi
```

```
array([[[[0.08627451, 0.0745098 , 0.08235294],
         [0.0745098 , 0.05490196, 0.05882353],
         [0.07843137, 0.05882353, 0.05490196],
         ...,
         [0.1254902 , 0.08235294, 0.07058824],
         [0.1372549 , 0.08235294, 0.0745098 ],
         [0.1372549 , 0.09803922, 0.09019608]],

        [[0.09411765, 0.0745098 , 0.07843137],
         [0.07843137, 0.05882353, 0.05882353],
         [0.09411765, 0.06666667, 0.05882353],
         ...,
         [0.15294118, 0.09411765, 0.08235294],
         [0.16862745, 0.10196078, 0.09019608],
         [0.1372549 , 0.09019608, 0.0745098 ]],

        [[0.10588235, 0.06666667, 0.09019608],
         [0.10196078, 0.0627451 , 0.07058824],
         [0.09803922, 0.08235294, 0.0627451 ],
         ...,
         [0.1372549 , 0.10588235, 0.08235294],
         [0.16078431, 0.12156863, 0.10980392],
         [0.15294118, 0.11764706, 0.10588235]],
```

```
emotion_roi.shape
```

```
(1, 48, 48, 3)
```

```
mask_roi.shape[0], mask_roi.shape[1], mask_roi.shape[2]
```

```
(1, 224, 224)
```

```
for idx, i in enumerate(vid_frames):
    cv2.imwrite(f'Test/{idx}.png', i)
```