```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)

     Mounted at /content/drive
```

```
#Extracting files from zip folder in temporary folder
import zipfile
file="/content/drive/MyDrive/data.zip"
with zipfile.ZipFile(file, "r") as z:
  z.extractall()
  print("done")

     done
```

```
import tensorflow as tf
import tensorflow_hub as hub
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import resnet50
from tensorflow.keras.layers import Dropout, MaxPooling2D, AveragePooling2D, Dense, Flatten, Input, Conv2D, add
from keras.utils.vis_utils import plot_model
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Sequential , Model , load_model
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam

from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report, f1_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer

from tensorflow.keras.preprocessing.image import load_img , img_to_array
from PIL import Image
import matplotlib.pyplot as plt
import pandas as pd

import cv2
```

Saved successfully! ×

```
import os
import time
import warnings
warnings.filterwarnings("ignore")
```

```
#finding all categories of pictures and labeling it
data_path='/content/data'
categories=os.listdir(data_path)
labels=[i for i in range(len(categories))]

label_dict=dict(zip(categories,labels))

print(label_dict)
print(categories)
print(labels)

     {'without': 0, 'with': 1}
     ['without', 'with']
     [0, 1]
```

```
# preprocessing each readable image using resnet50 and cv2 and labeling it
img_size=100
data=[]
target=[]


for category in categories:
    folder_path=os.path.join(data_path,category)
    img_names=os.listdir(folder_path)

    for img_name in img_names:
        img_path=os.path.join(folder_path,img_name)
        img=cv2.imread(img_path)
        if img is not None:
            lab = img_path.split(os.path.sep)[-2]
            target.append(label_dict[lab])
            img = img.astype("float")
            img = resnet50.preprocess_input(img)
            img = cv2.resize(img, (64, 64))
            data.append(img)
# convert the data into a NumPy array, then preprocess it by scaling
# all pixel intensities to the range [0, 1]
data = np.array(data, dtype="float32")/255.0
target = np.array(target)
# doing one hot encoding
lb = LabelBinarizer()
target = lb.fit_transform(target)
target=to_categorical(target)
# partition the data into training and testing splits using 70% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, target,stratify=target, test_size=0.20, random_state=42)

# construct the training image generator for data augmentation
aug = ImageDataGenerator(rotation_range=20,
                         zoom_range=0.15,
                                                width_shift_range=0.2,
                                                height_shift_range=0.2,
                                                shear_range=0.15,
                                                horizontal_flip=True,
                                                fill_mode="nearest")
data[0].shape
```

```
    (64, 64, 3)
```

```
baseModel = resnet50.ResNet50(weights="imagenet", include_top=False,input_tensor=Input(shape=(64, 64, 3)))
# the base model
headModel = baseModel.output
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(len(lb.classes_), activation="softmax")(headModel)
                                  the base model (this will become
```

Saved successfully! ×

```
                                       , 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


learningrate = [.0001,.001,.01,.05]
BS = 15
EPOCH = [1,10,15,20,25]
score=0
final_lr=0
final_epoch=0
for INIT_LR in learningrate:
    for EPOCHS in EPOCH:
            opt = Adam(lr=INIT_LR)
            model.compile(loss="binary_crossentropy", optimizer=opt,
                metrics=["accuracy"])
            # train the head of the network
            print("[INFO] training head...")
            model.fit(
                aug.flow(trainX, trainY, batch_size=BS),
                steps_per_epoch=len(trainX) // BS,
                validation_data=(testX, testY),
                validation_steps=len(testX) // BS,
                epochs=EPOCHS)
            print("[INFO] evaluating network...")
            predictions = model.predict(testX, batch_size=BS)
            t=f1_score(testY.argmax(axis=1),predictions.argmax(axis=1))
            print(t)
            print('/n')
```

```
/n
[INFO] training head...
Epoch 1/15
534/534 [==============================] - ETA: 0s - loss: 0.5349 - accuracy: 0.7272WARNING:tensorflow:Your input ran out of data; in
534/534 [==============================] - 197s 362ms/step - loss: 0.5349 - accuracy: 0.7272 - val_loss: 0.4842 - val_accuracy: 0.7614
Epoch 2/15
534/534 [==============================] - 153s 287ms/step - loss: 0.5371 - accuracy: 0.7196
Epoch 3/15
534/534 [==============================] - 153s 286ms/step - loss: 0.5306 - accuracy: 0.7326
Epoch 4/15
534/534 [==============================] - 153s 287ms/step - loss: 0.5253 - accuracy: 0.7330
Epoch 5/15
534/534 [==============================] - 154s 289ms/step - loss: 0.5235 - accuracy: 0.7361
Epoch 6/15
534/534 [==============================] - 155s 290ms/step - loss: 0.5159 - accuracy: 0.7476
Epoch 7/15
534/534 [==============================] - 155s 291ms/step - loss: 0.5243 - accuracy: 0.7367
Epoch 8/15
534/534 [==============================] - 154s 288ms/step - loss: 0.5200 - accuracy: 0.7402
Epoch 9/15
534/534 [==============================] - 154s 289ms/step - loss: 0.5213 - accuracy: 0.7409
Epoch 10/15
534/534 [==============================] - 153s 286ms/step - loss: 0.5147 - accuracy: 0.7448
Epoch 11/15
534/534 [==============================] - 152s 284ms/step - loss: 0.5189 - accuracy: 0.7406
Epoch 12/15
534/534 [==============================] - 152s 284ms/step - loss: 0.5120 - accuracy: 0.7481
Epoch 13/15
534/534 [==============================] - 152s 285ms/step - loss: 0.5124 - accuracy: 0.7414
Epoch 14/15
534/534 [==============================] - 152s 285ms/step - loss: 0.5092 - accuracy: 0.7486
Epoch 15/15
534/534 [==============================] - 153s 287ms/step - loss: 0.5040 - accuracy: 0.7541
[INFO] evaluating network...
134/134 [==============================] - 35s 251ms/step
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Ada
0.807142857142857
/n
[INFO] training head...
Epoch 1/20
534/534 [==============================] - ETA: 0s - loss: 0.5072 - accuracy: 0.7454WARNING:tensorflow:Your input ran out of data; in
534/534 [==============================] - 184s 338ms/step - loss: 0.5072 - accuracy: 0.7454 - val_loss: 0.4389 - val_accuracy: 0.794
                                  ========] - 143s 268ms/step - loss: 0.5049 - accuracy: 0.7538

534/534 [==============================] - 145s 271ms/step - loss: 0.5111 - accuracy: 0.7508
Epoch 4/20
534/534 [==============================] - 144s 269ms/step - loss: 0.5094 - accuracy: 0.7478
Epoch 5/20
534/534 [==============================] - 143s 268ms/step - loss: 0.5134 - accuracy: 0.7377
```

Saved successfully!

```
opt = Adam(lr=.01)
BS=15
model.compile(loss="binary_crossentropy", optimizer=opt,
                metrics=["accuracy"])
# train the head of the network
print("[INFO] training head...")
H=model.fit(
                aug.flow(trainX, trainY, batch_size=BS),
                steps_per_epoch=len(trainX) // BS,
                validation_data=(testX, testY),
                validation_steps=len(testX) // BS,
                epochs=30)
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=BS)
t=f1_score(testY.argmax(axis=1),predictions.argmax(axis=1))
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Ada
[INFO] training head...
Epoch 1/30
534/534 [==============================] - ETA: 0s - loss: 0.4614 - accuracy: 0.7775WARNING:tensorflow:Your input ran out of data; in
534/534 [==============================] - 209s 382ms/step - loss: 0.4614 - accuracy: 0.7775 - val_loss: 0.3644 - val_accuracy: 0.832
Epoch 2/30
534/534 [==============================] - 158s 296ms/step - loss: 0.4609 - accuracy: 0.7824
Epoch 3/30
534/534 [==============================] - 157s 293ms/step - loss: 0.4655 - accuracy: 0.7754
Epoch 4/30
534/534 [==============================] - 157s 293ms/step - loss: 0.4615 - accuracy: 0.7775
Epoch 5/30
534/534 [==============================] - 155s 290ms/step - loss: 0.4576 - accuracy: 0.7742
Epoch 6/30
```

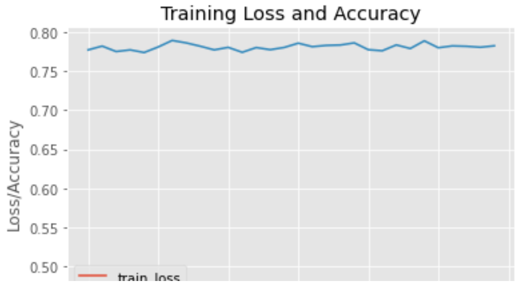
```
534/534 [==============================] - 155s 290ms/step - loss: 0.4634 - accuracy: 0.7813
Epoch 7/30
534/534 [==============================] - 158s 296ms/step - loss: 0.4522 - accuracy: 0.7896
Epoch 8/30
534/534 [==============================] - 159s 297ms/step - loss: 0.4533 - accuracy: 0.7865
Epoch 9/30
534/534 [==============================] - 157s 294ms/step - loss: 0.4621 - accuracy: 0.7823
Epoch 10/30
534/534 [==============================] - 158s 295ms/step - loss: 0.4628 - accuracy: 0.7775
Epoch 11/30
534/534 [==============================] - 157s 295ms/step - loss: 0.4602 - accuracy: 0.7808
Epoch 12/30
534/534 [==============================] - 160s 299ms/step - loss: 0.4680 - accuracy: 0.7744
Epoch 13/30
534/534 [==============================] - 157s 294ms/step - loss: 0.4628 - accuracy: 0.7805
Epoch 14/30
534/534 [==============================] - 156s 291ms/step - loss: 0.4740 - accuracy: 0.7778
Epoch 15/30
534/534 [==============================] - 158s 295ms/step - loss: 0.4592 - accuracy: 0.7806
Epoch 16/30
534/534 [==============================] - 155s 290ms/step - loss: 0.4503 - accuracy: 0.7861
Epoch 17/30
534/534 [==============================] - 156s 293ms/step - loss: 0.4568 - accuracy: 0.7816
Epoch 18/30
534/534 [==============================] - 157s 293ms/step - loss: 0.4672 - accuracy: 0.7833
Epoch 19/30
534/534 [==============================] - 157s 294ms/step - loss: 0.4577 - accuracy: 0.7838
Epoch 20/30
534/534 [==============================] - 157s 294ms/step - loss: 0.4522 - accuracy: 0.7865
Epoch 21/30
534/534 [==============================] - 157s 294ms/step - loss: 0.4609 - accuracy: 0.7778
Epoch 22/30
534/534 [==============================] - 155s 291ms/step - loss: 0.4695 - accuracy: 0.7764
Epoch 23/30
534/534 [==============================] - 157s 294ms/step - loss: 0.4545 - accuracy: 0.7839
Epoch 24/30
534/534 [==============================] - 156s 292ms/step - loss: 0.4640 - accuracy: 0.7793
Epoch 25/30
534/534 [==============================] - 155s 290ms/step - loss: 0.4537 - accuracy: 0.7891
Epoch 26/30
534/534 [==============================] - 156s 292ms/step - loss: 0.4629 - accuracy: 0.7803
Epoch 27/30
```

Saved successfully!

```
0.8671462829736212
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 30), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 30), H.history["accuracy"], label="train_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
```

```
<matplotlib.legend.Legend at 0x7f6a54b50730>
```



Saved successfully!