Masked Faces Detection Models

Jyväskylä University of Applied Sciences

Master of enginerring

2020 Bassam Al-Asadi

```
#import the libraries
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
import os
import keras

from tensorflow.keras.preprocessing.image import ImageDataGenerator
#import the VGG16  pre-traiend model
from tensorflow.keras.applications import VGG16
#import the Inception pre-traiend model
from tensorflow.keras.applications import InceptionV3
#import the MobileNetV2 pre-traiend model
from tensorflow.keras.applications import MobileNetV2



from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout,BatchNormalization
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 img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import cv2
import random as rand


# The generate_box function will obtain the face coordinates from the annotations fils
def generate_box(obj):
    xmin = int(obj.find('xmin').text)
    ymin = int(obj.find('ymin').text)
    xmax = int(obj.find('xmax').text)
    ymax = int(obj.find('ymax').text)
```

```
    return [xmin, ymin, xmax, ymax]
# The generate_label function will assign label associated with each label and return numb
def generate_label(obj):
    if obj.find('name').text == "with_mask":
        return 1
    elif obj.find('name').text == "mask_weared_incorrect":
        return 2
    return 0
# The generate_target function will parse the annotations file and obtain the objects from
def generate_target(image_id, file):
    with open(file) as f:
        data = f.read()
        soup = BeautifulSoup(data, 'xml')
        objects = soup.find_all('object')

        num_objs = len(objects)
        boxes = []
        labels = []
        for i in objects:
            boxes.append(generate_box(i))
            labels.append(generate_label(i))

        boxes=np.array(boxes)
        labels=np.array(labels)

        img_id = np.array(image_id)
    # Annotation is in dictionary format
        target = {}
        target["boxes"] = boxes
        target["labels"] = labels

        return (target,num_objs)


imgs = list(sorted(os.listdir("/content/drive/My Drive/faceMask/images")))
len(imgs)
```

853

```
labels = list(sorted(os.listdir("/content/drive/My Drive/faceMask/annotations/")))


# Here we store the number of faces in each image and their coordinates
targets=[]# face coordinates
numobjs=[]# number of faces in each image
for i in range(853):
    file_image = 'maksssksksss'+ str(i) + '.png'
    file_label = 'maksssksksss'+ str(i) + '.xml'
    img_path = os.path.join("/content/drive/My Drive/faceMask/images/", file_image)
    label_path = os.path.join("/content/drive/My Drive/faceMask/annotations/", file_label)
```

```
    #Generate Label
    target,numobj = generate_target(i, label_path)
    targets.append(target)
    numobjs.append(numobj)


#We continue carrying forward to obtain the faces from the images by using the extract coo
import cv2
import matplotlib.pyplot as plt
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
face_images=[]
face_labels=[]
for i in range(853):
    img_path = r"/content/drive/My Drive/faceMask/images/maksssksksss{}.png".format(i)
    img = cv2.imread(img_path)
    for j in range(numobjs[i]):
        locs=(targets[i]['boxes'][j])
        img1=img[locs[1]:locs[3],locs[0]:locs[2]]
        img1 = cv2.resize(img1, (224, 224))
        img1 = img_to_array(img1)
        img1 = preprocess_input(img1)
        face_images.append(img1)
        face_labels.append(targets[i]['labels'][j])

face_images= np.array(face_images, dtype="float32")
face_labels = np.array(face_labels)


# How many faces we have in 853 images
len(face_labels)
```

```
4072
```

```
# The categories distribution
unique, counts = np.unique(face_labels, return_counts=True)
dict(zip(unique, counts))
```

```
{0: 717, 1: 3232, 2: 123}
```

```
# Encoding the labels to 2D tensor
lb = LabelEncoder()
labels = lb.fit_transform(face_labels)
labels = to_categorical(labels)
labels
```

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

```
# Perform data augmentation.
aug = ImageDataGenerator(
    zoom_range=0.1,
    rotation_range=25,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest"
    )


# Split the data into train sets and test sets
(trainX, testX, trainY, testY) = train_test_split(face_images, labels,
  test_size=0.2, stratify=labels, random_state=42)


# Define the learning rate parameter, number of epochs, and the bunch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32
```

Inception3

```
# Download and define the inceptionV3 pre-trained model
InceptionModel = InceptionV3(weights="imagenet", include_top=False,
  input_shape=(224, 224, 3))

# Construct the model to fit the pre-trained base model
headModel = InceptionModel.output
headModel = AveragePooling2D(pool_size=(5, 5))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(256, activation="relu")(headModel)
headModel = Dropout(0.25)(headModel)
headModel = Dense(3, activation="softmax")(headModel)

model1 = Model(inputs=InceptionModel.input, outputs=headModel)

for layer in InceptionModel.layers:
  layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/ir
87916544/87910968 [==============================] - 1s 0us/step
```

```
#Complie the model and train it
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model1.compile(loss="categorical_crossentropy", optimizer=opt,
  metrics=["accuracy"])
```

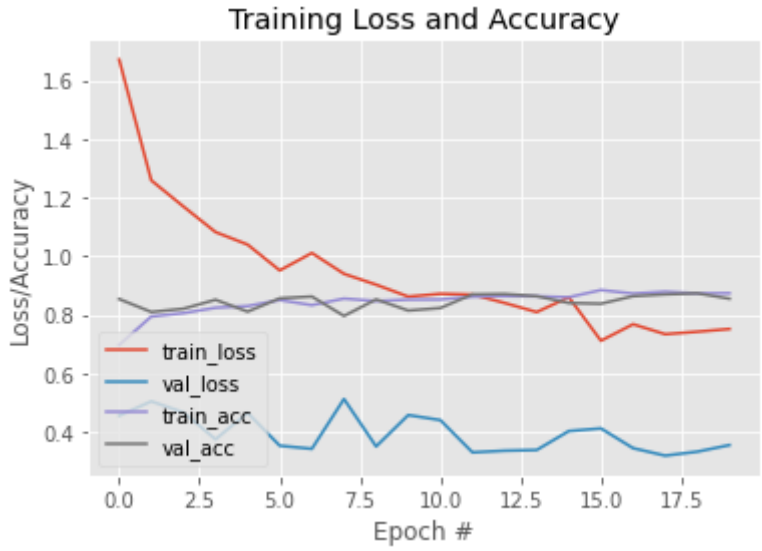
```
I = model1.fit(
  aug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS,
class_weight = {0:5 , 1:1, 2:10})
```

```
Epoch 1/20
101/101 [==============================] - 50s 499ms/step - loss: 1.6708 - accuracy:
Epoch 2/20
101/101 [==============================] - 46s 454ms/step - loss: 1.2589 - accuracy:
Epoch 3/20
101/101 [==============================] - 46s 451ms/step - loss: 1.1698 - accuracy:
Epoch 4/20
101/101 [==============================] - 45s 450ms/step - loss: 1.0825 - accuracy:
Epoch 5/20
101/101 [==============================] - 46s 451ms/step - loss: 1.0397 - accuracy:
Epoch 6/20
101/101 [==============================] - 45s 449ms/step - loss: 0.9516 - accuracy:
Epoch 7/20
101/101 [==============================] - 45s 449ms/step - loss: 1.0111 - accuracy:
Epoch 8/20
101/101 [==============================] - 45s 449ms/step - loss: 0.9400 - accuracy:
Epoch 9/20
101/101 [==============================] - 45s 445ms/step - loss: 0.9030 - accuracy:
Epoch 10/20
101/101 [==============================] - 45s 444ms/step - loss: 0.8622 - accuracy:
Epoch 11/20
101/101 [==============================] - 45s 444ms/step - loss: 0.8718 - accuracy:
Epoch 12/20
101/101 [==============================] - 45s 445ms/step - loss: 0.8692 - accuracy:
Epoch 13/20
101/101 [==============================] - 45s 443ms/step - loss: 0.8406 - accuracy:
Epoch 14/20
101/101 [==============================] - 44s 440ms/step - loss: 0.8095 - accuracy:
Epoch 15/20
101/101 [==============================] - 45s 442ms/step - loss: 0.8602 - accuracy:
Epoch 16/20
101/101 [==============================] - 45s 441ms/step - loss: 0.7118 - accuracy:
Epoch 17/20
101/101 [==============================] - 44s 438ms/step - loss: 0.7680 - accuracy:
Epoch 18/20
101/101 [==============================] - 44s 438ms/step - loss: 0.7341 - accuracy:
Epoch 19/20
101/101 [==============================] - 44s 439ms/step - loss: 0.7425 - accuracy:
Epoch 20/20
101/101 [==============================] - 44s 440ms/step - loss: 0.7518 - accuracy:
```

```
#Evaluate the model
print("[INFO] evaluating network...")
predIdxs = model1.predict(testX, batch_size=32)

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

```
# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,))

# plot the training loss and accuracy
E1 = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, E1), I.history["loss"], label="train_loss")
plt.plot(np.arange(0, E1), I.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, E1), I.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, E1), I.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()
```

```
[INFO] evaluating network...
              precision    recall  f1-score   support

           0       0.63      0.98      0.77       143
           1       0.98      0.84      0.90       647
           2       0.36      0.60      0.45        25

    accuracy                           0.86       815
   macro avg       0.66      0.81      0.71       815
weighted avg       0.90      0.86      0.87       815
```



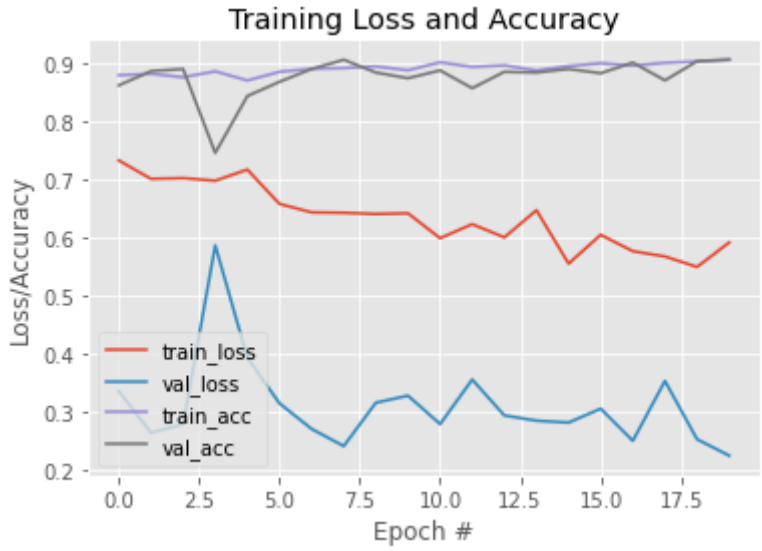

```
#save the model
model1.save('Inception.h5')


#load the saved model
model1 = keras.models.load_model('Inception.h5')
```

```
#train the saved model again
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model1.compile(loss="categorical_crossentropy", optimizer=opt,
  metrics=["accuracy"])

print("[INFO] training head...")
I = model1.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,
class_weight = {0:5 , 1:1, 2:10})
```

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/20
101/101 [==============================] - 47s 463ms/step - loss: 0.7330 - accuracy:
Epoch 2/20
101/101 [==============================] - 45s 445ms/step - loss: 0.7013 - accuracy:
Epoch 3/20
101/101 [==============================] - 45s 444ms/step - loss: 0.7026 - accuracy:
Epoch 4/20
101/101 [==============================] - 45s 445ms/step - loss: 0.6981 - accuracy:
Epoch 5/20
101/101 [==============================] - 45s 442ms/step - loss: 0.7174 - accuracy:
Epoch 6/20
101/101 [==============================] - 45s 441ms/step - loss: 0.6581 - accuracy:
Epoch 7/20
101/101 [==============================] - 45s 441ms/step - loss: 0.6436 - accuracy:
Epoch 8/20
101/101 [==============================] - 44s 440ms/step - loss: 0.6428 - accuracy:
Epoch 9/20
101/101 [==============================] - 44s 439ms/step - loss: 0.6408 - accuracy:
Epoch 10/20
101/101 [==============================] - 44s 438ms/step - loss: 0.6420 - accuracy:
Epoch 11/20
101/101 [==============================] - 44s 438ms/step - loss: 0.5990 - accuracy:
Epoch 12/20
101/101 [==============================] - 44s 437ms/step - loss: 0.6232 - accuracy:
Epoch 13/20
101/101 [==============================] - 44s 436ms/step - loss: 0.6004 - accuracy:
Epoch 14/20
101/101 [==============================] - 44s 438ms/step - loss: 0.6473 - accuracy:
Epoch 15/20
101/101 [==============================] - 44s 437ms/step - loss: 0.5552 - accuracy:
Epoch 16/20
101/101 [==============================] - 44s 437ms/step - loss: 0.6049 - accuracy:
Epoch 17/20
101/101 [==============================] - 44s 438ms/step - loss: 0.5766 - accuracy:
Epoch 18/20
101/101 [==============================] - 44s 440ms/step - loss: 0.5675 - accuracy:
Epoch 19/20
101/101 [==============================] - 44s 437ms/step - loss: 0.5494 - accuracy:
Epoch 20/20
101/101 [==============================] - 44s 436ms/step - loss: 0.5916 - accuracy:
```

```
#Evaluate the model again
print("[INFO] evaluating network...")
predIdxs = model1.predict(testX, batch_size=32)

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

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,))

# plot the training loss and accuracy
E1 = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, E1), I.history["loss"], label="train_loss")
plt.plot(np.arange(0, E1), I.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, E1), I.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, E1), I.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()
```

```
[INFO] evaluating network...
              precision    recall  f1-score   support

           0       0.75      0.93      0.83       143
           1       0.97      0.92      0.94       647
           2       0.50      0.44      0.47        25

    accuracy                           0.91       815
   macro avg       0.74      0.76      0.75       815
weighted avg       0.91      0.91      0.91       815
```




```
#save the Inception model
model1.save('face_mask_Inception.h5')
```

MobileNetV2

```
# Download and define the ModileNetV2 pre-trained model

MobileNetModel = MobileNetV2(weights="imagenet", include_top=False,
  input_shape=(224, 224, 3))

# Construct the model to fit the pre-trained base model
headModel = MobileNetModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(256, activation="relu")(headModel)
headModel = Dropout(0.25)(headModel)
headModel = Dense(3, activation="softmax")(headModel)

model2 = Model(inputs=MobileNetModel.input, outputs=headModel)

for layer in MobileNetModel.layers:
  layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mo
9412608/9406464 [==============================] - 0s 0us/step
```

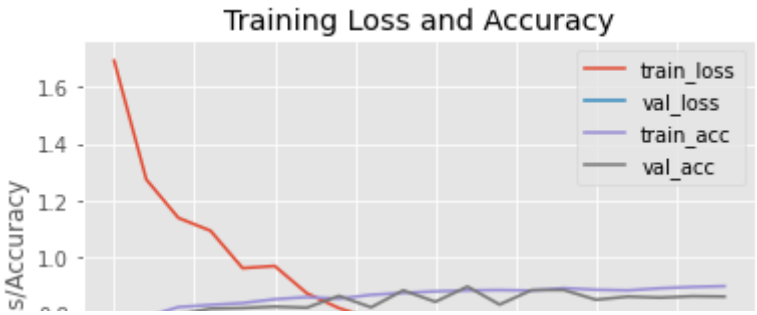
```
#Complie the model and train it
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model2.compile(loss="categorical_crossentropy", optimizer=opt,
  metrics=["accuracy"])

M = model2.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,
class_weight = {0:5 , 1:1, 2:10})
```

```
Epoch 1/20
101/101 [==============================] - 42s 412ms/step - loss: 1.6940 - accuracy:
Epoch 2/20
101/101 [==============================] - 40s 399ms/step - loss: 1.2738 - accuracy:
Epoch 3/20
101/101 [==============================] - 40s 396ms/step - loss: 1.1374 - accuracy:
Epoch 4/20
101/101 [==============================] - 40s 392ms/step - loss: 1.0920 - accuracy:
Epoch 5/20
101/101 [==============================] - 39s 390ms/step - loss: 0.9596 - accuracy:
Epoch 6/20
101/101 [==============================] - 39s 388ms/step - loss: 0.9671 - accuracy:
Epoch 7/20
101/101 [==============================] - 39s 386ms/step - loss: 0.8706 - accuracy:
Epoch 8/20
101/101 [==============================] - 40s 396ms/step - loss: 0.8181 - accuracy:
Epoch 9/20
```

```
101/101 [==============================] - 40s 399ms/step - loss: 0.7808 - accuracy:
Epoch 10/20
101/101 [==============================] - 40s 395ms/step - loss: 0.7590 - accuracy:
Epoch 11/20
101/101 [==============================] - 40s 392ms/step - loss: 0.7769 - accuracy:
Epoch 12/20
101/101 [==============================] - 39s 391ms/step - loss: 0.7387 - accuracy:
Epoch 13/20
101/101 [==============================] - 39s 388ms/step - loss: 0.7081 - accuracy:
Epoch 14/20
101/101 [==============================] - 40s 397ms/step - loss: 0.6949 - accuracy:
Epoch 15/20
101/101 [==============================] - 40s 395ms/step - loss: 0.6757 - accuracy:
Epoch 16/20
101/101 [==============================] - 40s 397ms/step - loss: 0.6433 - accuracy:
Epoch 17/20
101/101 [==============================] - 39s 391ms/step - loss: 0.6448 - accuracy:
Epoch 18/20
101/101 [==============================] - 39s 382ms/step - loss: 0.6354 - accuracy:
Epoch 19/20
101/101 [==============================] - 39s 391ms/step - loss: 0.6199 - accuracy:
Epoch 20/20
101/101 [==============================] - 41s 404ms/step - loss: 0.6127 - accuracy:
```

```
#Evaluate the model
print("[INFO] evaluating network...")
predIdxs = model2.predict(testX, batch_size=32)

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

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,))

# plot the training loss and accuracy
E2 = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, E2), M.history["loss"], label="train_loss")
plt.plot(np.arange(0, E2), M.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, E2), M.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, E2), M.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="upper right")
plt.show()
```

```
[INFO] evaluating network...
              precision    recall  f1-score   support

           0       0.65      0.93      0.76       143
           1       0.97      0.85      0.91       647
           2       0.35      0.56      0.43        25

    accuracy                           0.86       815
   macro avg       0.66      0.78      0.70       815
weighted avg       0.90      0.86      0.87       815
```




```
#save the model.
model2.save('MobileNet.h5')
```

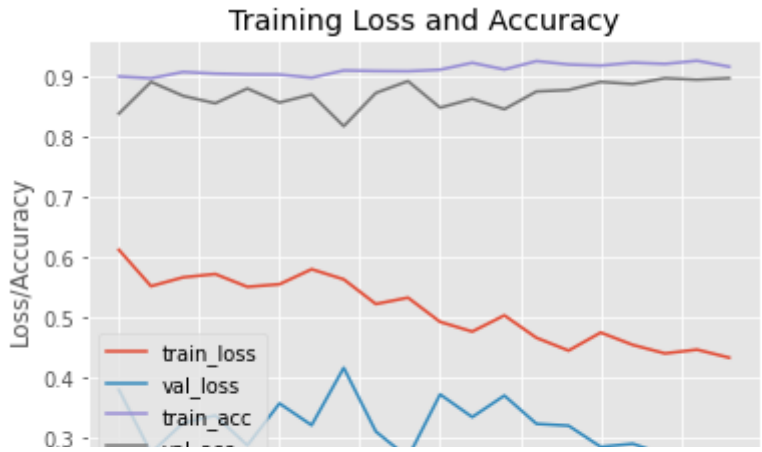
```
#load the saved model
model2 = keras.models.load_model('MobileNet.h5')


#train the saved model again
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model2.compile(loss="categorical_crossentropy", optimizer=opt,
  metrics=["accuracy"])

print("[INFO] training head...")
M = model2.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,
class_weight = {0:5 , 1:1, 2:10})
```

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/20
101/101 [==============================] - 42s 415ms/step - loss: 0.6118 - accuracy:
Epoch 2/20
101/101 [==============================] - 41s 406ms/step - loss: 0.5515 - accuracy:
Epoch 3/20
101/101 [==============================] - 41s 402ms/step - loss: 0.5662 - accuracy:
Epoch 4/20
101/101 [==============================] - 41s 402ms/step - loss: 0.5716 - accuracy:
Epoch 5/20
101/101 [==============================] - 41s 401ms/step - loss: 0.5504 - accuracy:
Epoch 6/20
101/101 [==============================] - 40s 399ms/step - loss: 0.5547 - accuracy:
Epoch 7/20
```

```
101/101 [==============================] - 41s 408ms/step - loss: 0.5796 - accuracy:
Epoch 8/20
101/101 [==============================] - 41s 404ms/step - loss: 0.5628 - accuracy:
Epoch 9/20
101/101 [==============================] - 41s 403ms/step - loss: 0.5219 - accuracy:
Epoch 10/20
101/101 [==============================] - 41s 402ms/step - loss: 0.5324 - accuracy:
Epoch 11/20
101/101 [==============================] - 41s 401ms/step - loss: 0.4922 - accuracy:
Epoch 12/20
101/101 [==============================] - 40s 397ms/step - loss: 0.4761 - accuracy:
Epoch 13/20
101/101 [==============================] - 41s 407ms/step - loss: 0.5030 - accuracy:
Epoch 14/20
101/101 [==============================] - 41s 402ms/step - loss: 0.4657 - accuracy:
Epoch 15/20
101/101 [==============================] - 41s 402ms/step - loss: 0.4445 - accuracy:
Epoch 16/20
101/101 [==============================] - 40s 398ms/step - loss: 0.4745 - accuracy:
Epoch 17/20
101/101 [==============================] - 40s 399ms/step - loss: 0.4540 - accuracy:
Epoch 18/20
101/101 [==============================] - 40s 395ms/step - loss: 0.4398 - accuracy:
Epoch 19/20
101/101 [==============================] - 41s 406ms/step - loss: 0.4461 - accuracy:
Epoch 20/20
101/101 [==============================] - 41s 402ms/step - loss: 0.4327 - accuracy:
```

```
#Evaluate the model again
print("[INFO] evaluating network...")
predIdxs = model2.predict(testX, batch_size=32)

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

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,))

# plot the training loss and accuracy
E2 = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, E2), M.history["loss"], label="train_loss")
plt.plot(np.arange(0, E2), M.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, E2), M.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, E2), M.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()
```

```
[INFO] evaluating network...
              precision    recall  f1-score   support

           0       0.74      0.92      0.82       143
           1       0.97      0.91      0.94       647
           2       0.41      0.44      0.42        25

    accuracy                           0.90       815
   macro avg       0.70      0.76      0.73       815
weighted avg       0.91      0.90      0.90       815
```




```
#save the Mobilenet model
model2.save('face_mask_mobilenet.h5')
```

VGG16

```
# Download and define the VGG16 pre-trained model
ResNetModel = VGG16(weights="imagenet", include_top=False,
  input_shape=(224, 224, 3))

# Construct the model to fit the pre-trained base model
headModel = ResNetModel.output
headModel = AveragePooling2D(pool_size=(5, 5))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(256, activation="relu")(headModel)
headModel = Dropout(0.25)(headModel)
headModel = Dense(3, activation="softmax")(headModel)

model3 = Model(inputs=ResNetModel.input, outputs=headModel)

for layer in ResNetModel.layers:
  layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vg
58892288/58889256 [==============================] - 0s 0us/step
```

```
#Complie the model and train it
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model3.compile(loss="categorical_crossentropy", optimizer=opt,
  metrics=["accuracy"])
```

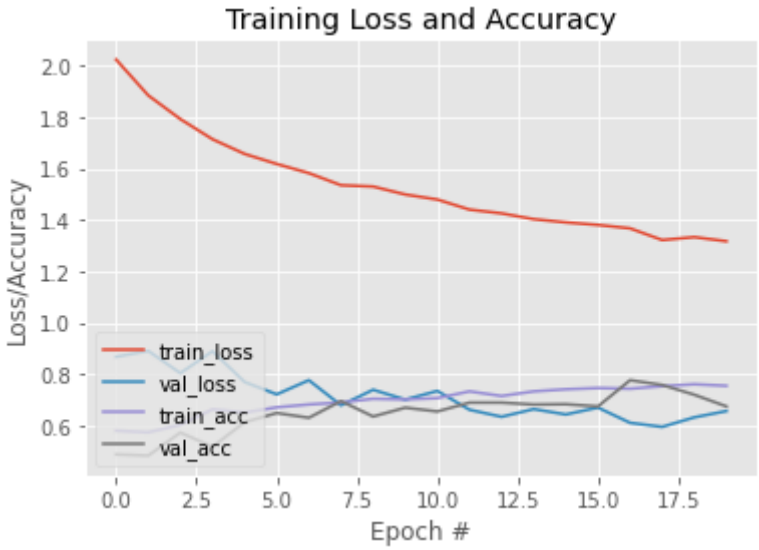
```
R = model3.fit(
  aug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS,
class_weight = {0:5 , 1:1, 2:10})
```

```
Epoch 1/20
101/101 [==============================] - 59s 587ms/step - loss: 2.0235 - accuracy:
Epoch 2/20
101/101 [==============================] - 51s 507ms/step - loss: 1.8843 - accuracy:
Epoch 3/20
101/101 [==============================] - 51s 505ms/step - loss: 1.7922 - accuracy:
Epoch 4/20
101/101 [==============================] - 51s 508ms/step - loss: 1.7139 - accuracy:
Epoch 5/20
101/101 [==============================] - 51s 508ms/step - loss: 1.6568 - accuracy:
Epoch 6/20
101/101 [==============================] - 51s 503ms/step - loss: 1.6177 - accuracy:
Epoch 7/20
101/101 [==============================] - 51s 501ms/step - loss: 1.5821 - accuracy:
Epoch 8/20
101/101 [==============================] - 51s 501ms/step - loss: 1.5351 - accuracy:
Epoch 9/20
101/101 [==============================] - 51s 501ms/step - loss: 1.5299 - accuracy:
Epoch 10/20
101/101 [==============================] - 51s 501ms/step - loss: 1.4990 - accuracy:
Epoch 11/20
101/101 [==============================] - 50s 499ms/step - loss: 1.4795 - accuracy:
Epoch 12/20
101/101 [==============================] - 50s 497ms/step - loss: 1.4403 - accuracy:
Epoch 13/20
101/101 [==============================] - 50s 496ms/step - loss: 1.4256 - accuracy:
Epoch 14/20
101/101 [==============================] - 50s 495ms/step - loss: 1.4031 - accuracy:
Epoch 15/20
101/101 [==============================] - 50s 496ms/step - loss: 1.3905 - accuracy:
Epoch 16/20
101/101 [==============================] - 50s 498ms/step - loss: 1.3807 - accuracy:
Epoch 17/20
101/101 [==============================] - 50s 497ms/step - loss: 1.3674 - accuracy:
Epoch 18/20
101/101 [==============================] - 50s 495ms/step - loss: 1.3219 - accuracy:
Epoch 19/20
101/101 [==============================] - 50s 495ms/step - loss: 1.3332 - accuracy:
Epoch 20/20
101/101 [==============================] - 50s 496ms/step - loss: 1.3172 - accuracy:
```

```
#Evaluate the model
print("[INFO] evaluating network...")
predIdxs = model3.predict(testX, batch_size=32)

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

# show a nicely formatted classification report
```

```
print(classification_report(testY.argmax(axis=1), predIdxs,))

# plot the training loss and accuracy
E3 = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, E3), R.history["loss"], label="train_loss")
plt.plot(np.arange(0, E3), R.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, E3), R.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, E3), R.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()
```

```
[INFO] evaluating network...
              precision    recall  f1-score   support

           0       0.37      0.99      0.54       143
           1       0.96      0.63      0.76       647
           2       0.27      0.12      0.17        25

    accuracy                           0.67       815
   macro avg       0.54      0.58      0.49       815
weighted avg       0.84      0.67      0.70       815
```



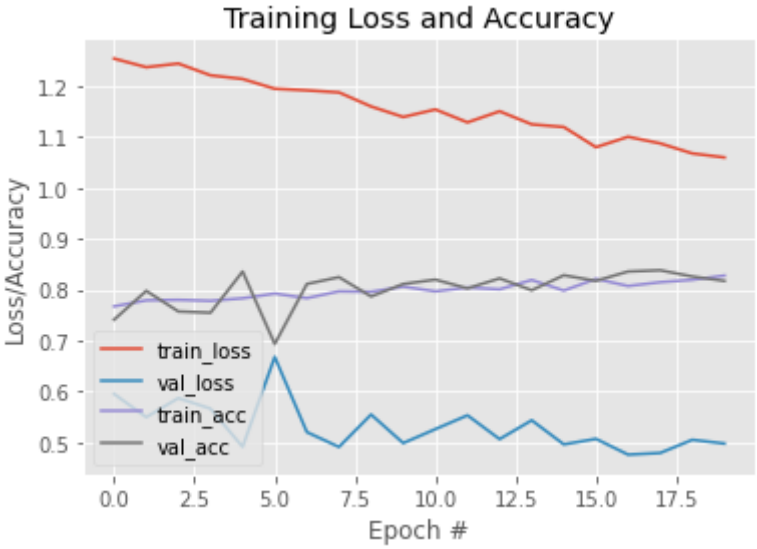

```
#save the model.
model3.save('VGG16.h5')


#load the saved model
model3 = keras.models.load_model('VGG16.h5')
```

```
#train the saved model again
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model3.compile(loss="categorical_crossentropy", optimizer=opt,
  metrics=["accuracy"])
print("[INFO] training head...")
R = model3.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,
  class_weight = {0:5 , 1:1, 2:10})
```

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/20
101/101 [==============================] - 51s 510ms/step - loss: 1.2538 - accuracy:
Epoch 2/20
101/101 [==============================] - 51s 507ms/step - loss: 1.2370 - accuracy:
Epoch 3/20
101/101 [==============================] - 51s 505ms/step - loss: 1.2441 - accuracy:
Epoch 4/20
101/101 [==============================] - 51s 505ms/step - loss: 1.2210 - accuracy:
Epoch 5/20
101/101 [==============================] - 51s 503ms/step - loss: 1.2138 - accuracy:
Epoch 6/20
101/101 [==============================] - 51s 503ms/step - loss: 1.1946 - accuracy:
Epoch 7/20
101/101 [==============================] - 51s 504ms/step - loss: 1.1915 - accuracy:
Epoch 8/20
101/101 [==============================] - 50s 499ms/step - loss: 1.1874 - accuracy:
Epoch 9/20
101/101 [==============================] - 50s 496ms/step - loss: 1.1598 - accuracy:
Epoch 10/20
101/101 [==============================] - 50s 498ms/step - loss: 1.1391 - accuracy:
Epoch 11/20
101/101 [==============================] - 50s 491ms/step - loss: 1.1539 - accuracy:
Epoch 12/20
101/101 [==============================] - 50s 492ms/step - loss: 1.1286 - accuracy:
Epoch 13/20
101/101 [==============================] - 50s 497ms/step - loss: 1.1503 - accuracy:
Epoch 14/20
101/101 [==============================] - 50s 499ms/step - loss: 1.1247 - accuracy:
Epoch 15/20
101/101 [==============================] - 50s 494ms/step - loss: 1.1194 - accuracy:
Epoch 16/20
101/101 [==============================] - 50s 496ms/step - loss: 1.0798 - accuracy:
Epoch 17/20
101/101 [==============================] - 50s 499ms/step - loss: 1.1003 - accuracy:
Epoch 18/20
101/101 [==============================] - 50s 497ms/step - loss: 1.0872 - accuracy:
Epoch 19/20
101/101 [==============================] - 51s 500ms/step - loss: 1.0676 - accuracy:
Epoch 20/20
101/101 [==============================] - 51s 500ms/step - loss: 1.0597 - accuracy:
```

```
#Evaluate the model again
print("[INFO] evaluating network...")
predIdxs = model3.predict(testX, batch_size=32)

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

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,))

# plot the training loss and accuracy
E3 = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, E3), R.history["loss"], label="train_loss")
plt.plot(np.arange(0, E3), R.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, E3), R.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, E3), R.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()
```

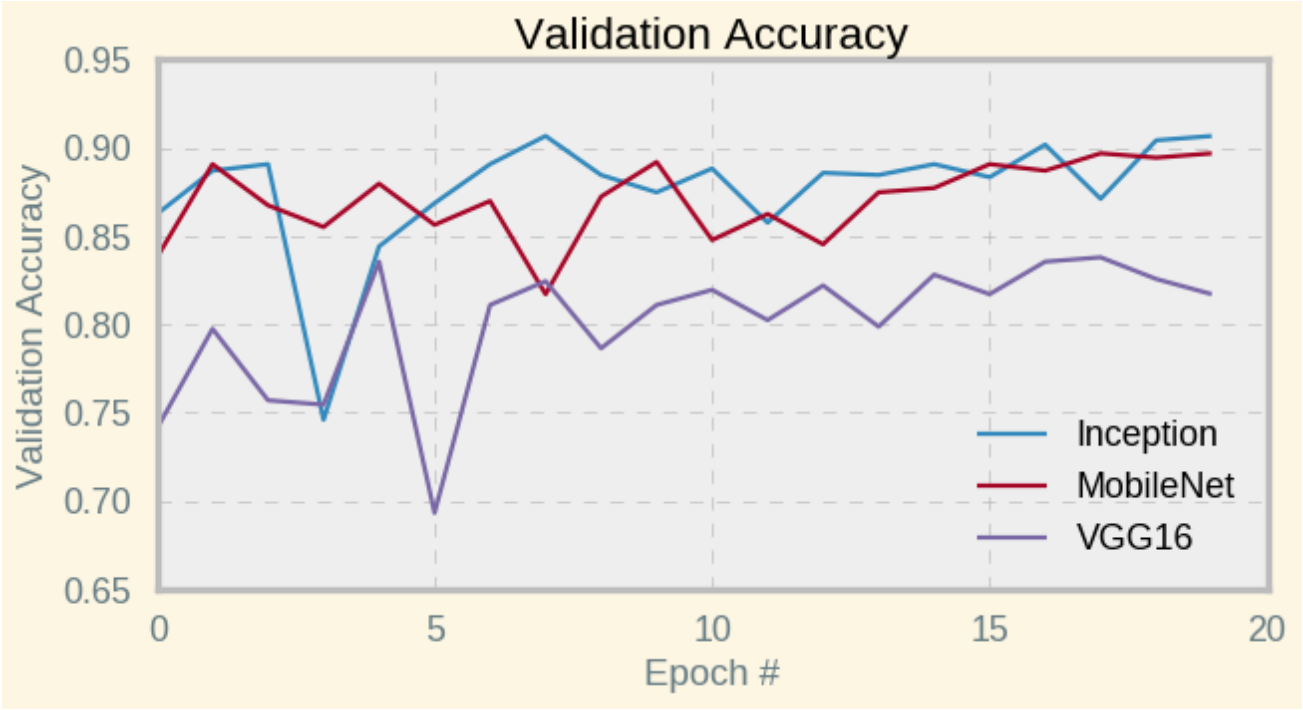
```
[INFO] evaluating network...
              precision    recall  f1-score   support

           0       0.53      0.99      0.69       143
           1       0.98      0.80      0.88       647
           2       0.32      0.24      0.27        25

    accuracy                           0.82       815
   macro avg       0.61      0.68      0.62       815
weighted avg       0.88      0.82      0.83       815
```



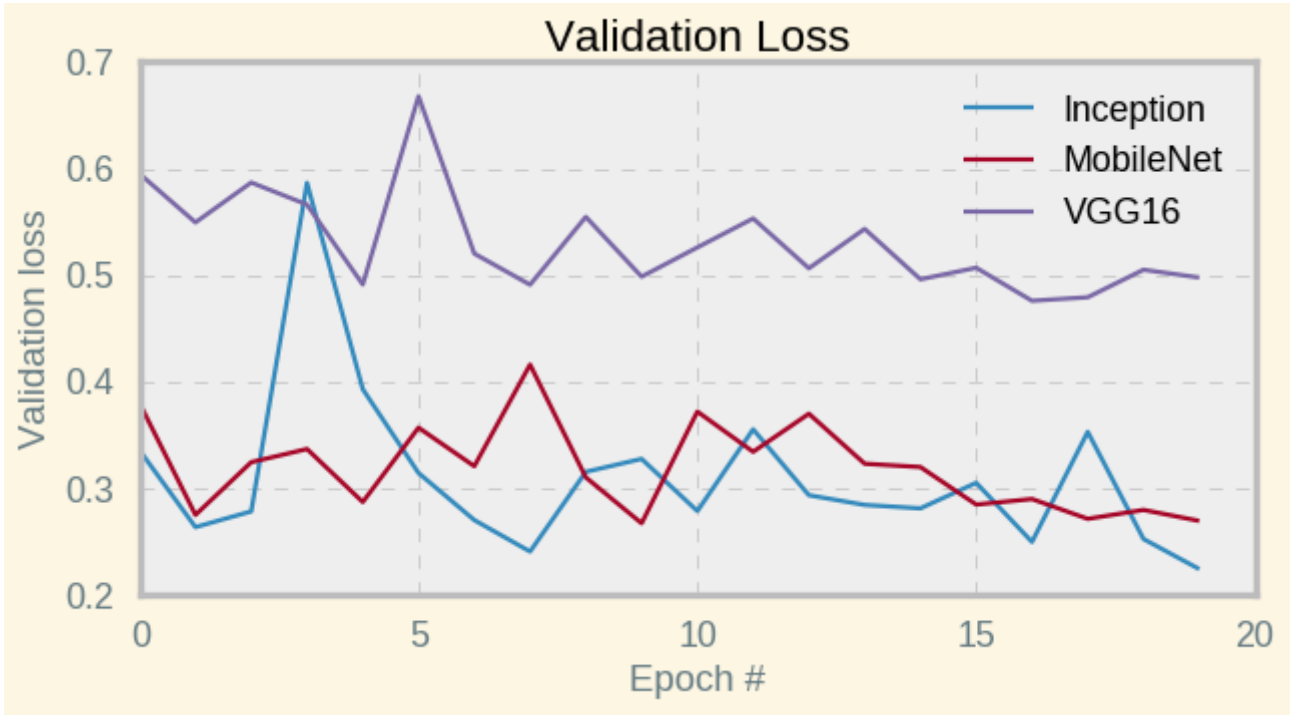

```
#save the resnet model.
model3.save('face_mask_VGG16.h5')
```

visual

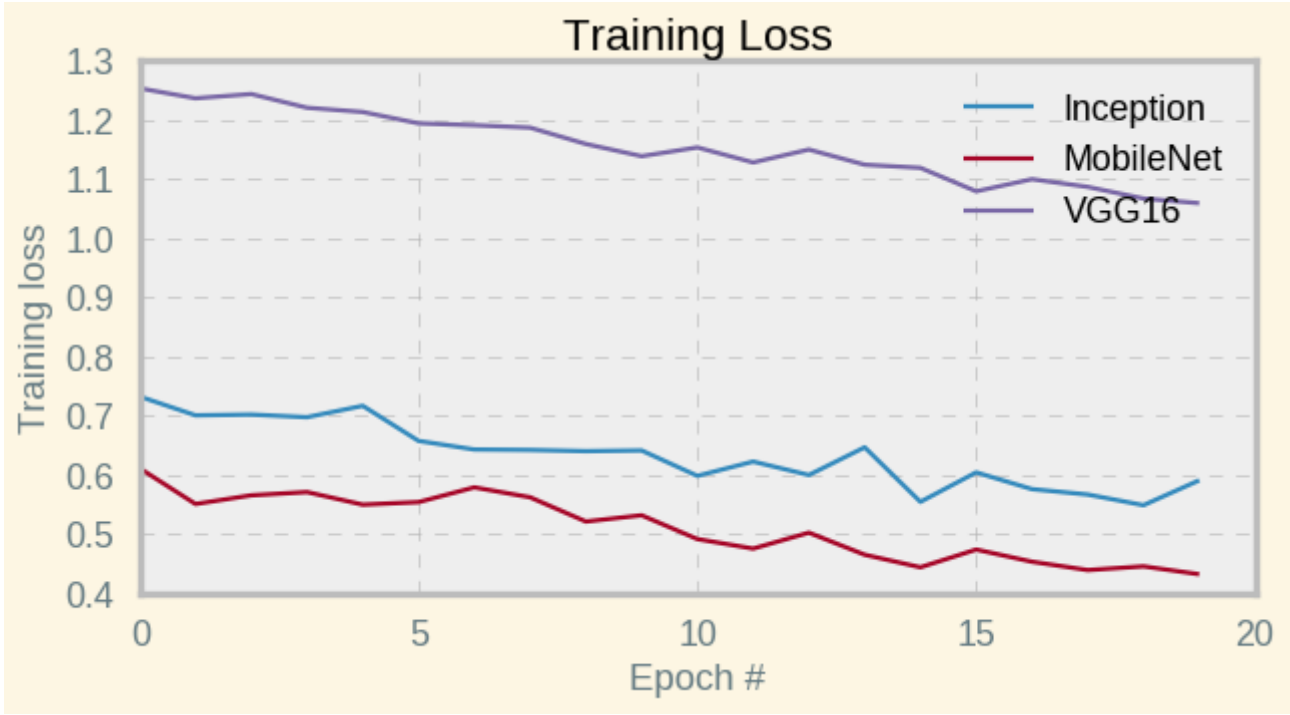
```
plt.style.use("bmh")
plt.figure(figsize=(8, 4))
plt.plot(np.arange(0, E1), I.history["val_accuracy"], label="Inception",linewidth=2, marke
plt.plot(np.arange(0, E2), M.history["val_accuracy"], label="MobileNet",linewidth=2, marke
plt.plot(np.arange(0, E3), R.history["val_accuracy"], label="VGG16",linewidth=2, markersiz
plt.title("Validation Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Validation Accuracy")
plt.legend(loc="lower right")
plt.show()
```



```
plt.style.use("bmh")
plt.figure(figsize=(8, 4))
plt.plot(np.arange(0, E1), I.history["val_loss"], label="Inception")
plt.plot(np.arange(0, E2), M.history["val_loss"], label="MobileNet")
plt.plot(np.arange(0, E3), R.history["val_loss"], label="VGG16")
plt.title("Validation Loss")
plt.xlabel("Epoch #")
plt.ylabel("Validation loss")
plt.legend(loc="upper right")
plt.show()
```



```
plt.style.use("bmh")
plt.figure(figsize=(8, 4))
plt.plot(np.arange(0, E1), I.history["loss"], label="Inception")
plt.plot(np.arange(0, E2), M.history["loss"], label="MobileNet")
plt.plot(np.arange(0, E3), R.history["loss"], label="VGG16")
plt.title("Training Loss")
plt.xlabel("Epoch #")
plt.ylabel("Training loss")
plt.legend(loc="upper right")
plt.show()
```



```
plt.style.use("bmh")
plt.figure(figsize=(8, 4))
plt.plot(np.arange(0, E1), I.history["accuracy"], label="Inception")
plt.plot(np.arange(0, E2), M.history["accuracy"], label="MobileNet")
plt.plot(np.arange(0, E3), R.history["accuracy"], label="VGG16")
plt.title("Training Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Training Accuracy")
plt.legend(loc="lower right")
plt.show()
```