In [4]:
```
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
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 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
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os
import pickle
```

In [5]:
```
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32

DIRECTORY = r"dataset"
CATEGORIES = ["with_mask", "without_mask"]

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")

data = []
labels = []

for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = preprocess_input(image)

        data.append(image)
        labels.append(category)
```

```
[INFO] loading images...
```

```
C:\Anaconda3\envs\tg\lib\site-packages\PIL\Image.py:996: UserWarning: Palette images
with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
```

In [6]:
```
data[1]
```

Out[6]:
```
array([[[ 0.32549024,  0.30196083,  0.24705887],
        [ 0.32549024,  0.30196083,  0.24705887],
        [ 0.34901965,  0.32549024,  0.27058828],
        ...,
        [ 0.35686278,  0.2941177 ,  0.27058828],
        [ 0.39607847,  0.33333337,  0.30980396],
        [ 0.39607847,  0.33333337,  0.30980396]],

       [[ 0.32549024,  0.30196083,  0.24705887],
        [ 0.32549024,  0.30196083,  0.24705887],
        [ 0.34901965,  0.32549024,  0.27058828],
        ...,
        [ 0.35686278,  0.2941177 ,  0.27058828],
        [ 0.39607847,  0.33333337,  0.30980396],
        [ 0.39607847,  0.33333337,  0.30980396]],

       [[ 0.3176471 ,  0.2941177 ,  0.23921573],
        [ 0.3176471 ,  0.2941177 ,  0.23921573],
        [ 0.3411765 ,  0.3176471 ,  0.26274514],
        ...,
        [ 0.34901965,  0.28627455,  0.26274514],
        [ 0.37254906,  0.30980396,  0.28627455],
        [ 0.37254906,  0.30980396,  0.28627455]],

       ...,

       [[-0.45098037, -0.47450978, -0.41960782],
        [-0.45098037, -0.47450978, -0.41960782],
        [-0.4823529 , -0.5058824 , -0.45098037],
        ...,
        [ 0.082353  ,  0.04313731,  0.09019613],
        [ 0.10588241,  0.06666672,  0.11372554],
        [ 0.10588241,  0.06666672,  0.11372554]],

       [[-0.56078434, -0.58431375, -0.5294118 ],
        [-0.56078434, -0.58431375, -0.5294118 ],
        [-0.6       , -0.6392157 , -0.58431375],
        ...,
        [ 0.09803927,  0.05882359,  0.10588241],
        [ 0.11372554,  0.07450986,  0.12156868],
        [ 0.11372554,  0.07450986,  0.12156868]],

       [[-0.56078434, -0.58431375, -0.5294118 ],
        [-0.56078434, -0.58431375, -0.5294118 ],
        [-0.6       , -0.6392157 , -0.58431375],
        ...,
        [ 0.09803927,  0.05882359,  0.10588241],
        [ 0.11372554,  0.07450986,  0.12156868],
        [ 0.11372554,  0.07450986,  0.12156868]]], dtype=float32)
```

In [7]:
```
labels[1]
```

Out[7]:
```
'with_mask'
```

In [8]:
```
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

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

In [9]:
```
labels
```

Out[9]:
```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [0., 1.],
       [0., 1.],
       [0., 1.]], dtype=float32)
```

In [ ]:
```
```

In [10]:
```
(trainX, testX, trainY, testY) = train_test_split(data, labels,
        test_size=0.20, stratify=labels, random_state=42)
```

In [11]:
```
# 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")
```

In [12]:
```
import tensorflow as tf
```

In [13]:
```
# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
        input_tensor=Input(shape=(224, 224, 3)))

# 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

# compile our model
print("[INFO] compiling model...")
opt = tf.keras.optimizers.legacy.Adam(learning_rate=INIT_LR, decay=INIT_LR / EPOCHS
model.compile(loss="binary_crossentropy", optimizer=opt,
        metrics=["accuracy"])
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [9
6, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the def
ault.
[INFO] compiling model...
```

In [14]:
```
# 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=5)
```

```
[INFO] training head...
Epoch 1/5
95/95 [==============================] - 125s 1s/step - loss: 0.4098 - accuracy: 0.8
537 - val_loss: 0.1423 - val_accuracy: 0.9817
Epoch 2/5
95/95 [==============================] - 89s 938ms/step - loss: 0.1476 - accuracy:
0.9657 - val_loss: 0.0744 - val_accuracy: 0.9857
Epoch 3/5
95/95 [==============================] - 90s 940ms/step - loss: 0.1006 - accuracy:
0.9720 - val_loss: 0.0536 - val_accuracy: 0.9896
Epoch 4/5
95/95 [==============================] - 87s 910ms/step - loss: 0.0860 - accuracy:
0.9763 - val_loss: 0.0452 - val_accuracy: 0.9896
Epoch 5/5
95/95 [==============================] - 85s 896ms/step - loss: 0.0751 - accuracy:
0.9782 - val_loss: 0.0429 - val_accuracy: 0.9883
```

In [15]:
```
print(tf.config.list_physical_devices('GPU'))
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

In [16]:
```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# 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)
```

```
[INFO] evaluating network...
24/24 [==============================] - 10s 354ms/step
```

In [17]:
```
# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
```

```
        target_names=lb.classes_))
```

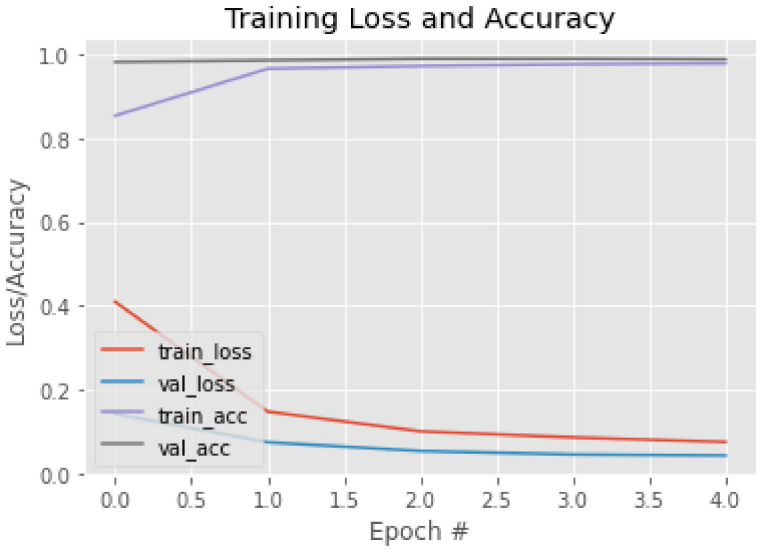
```
              precision    recall  f1-score   support

   with_mask       0.98      0.99      0.99       383
without_mask       0.99      0.98      0.99       384

    accuracy                           0.99       767
   macro avg       0.99      0.99      0.99       767
weighted avg       0.99      0.99      0.99       767
```

In [ ]:

In [18]:
```
# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save("mask_detector.model", save_format="h5")
```

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

In [21]:
```
# plot the training loss and accuracy
N = 5
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")
plt.savefig("plot.png")
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: