In [2]:
```
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
```

In [4]:
```
INIT_LR = 1e-4
EPOCHS = 20
BS = 32

DIRECTORY = "C:/Users/Devanshi/Downloads/Face-Mask-Detection-master/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:\Users\Devanshi\anaconda3\lib\site-packages\PIL\Image.py:951: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
```

In [5]:
```
# 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)

(trainX, testX, trainY, testY) = train_test_split(data, labels,
        test_size=0.20, stratify=labels, random_state=42)
```

In [7]:
```
labels.shape
```

Out[7]:
```
(3833, 2)
```

In [8]:
```
# 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")

# 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)))
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the defa
ult.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9412608/9406464 [==============================] - 3s 0us/step
```

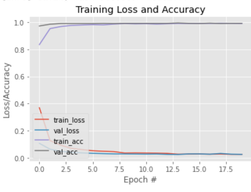
In [10]:
```
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
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
```

In [11]:
```
# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
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=EPOCHS)

# 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)

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

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/20
95/95 [==============================] - 80s 837ms/step - loss: 0.3692 - accuracy: 0.8352 - val_loss: 0.1065 - val_accuracy: 0.9726
Epoch 2/20
95/95 [==============================] - 83s 877ms/step - loss: 0.1270 - accuracy: 0.9532 - val_loss: 0.0631 - val_accuracy: 0.9857
Epoch 3/20
95/95 [==============================] - 78s 817ms/step - loss: 0.0931 - accuracy: 0.9684 - val_loss: 0.0482 - val_accuracy: 0.9909
Epoch 4/20
95/95 [==============================] - 78s 825ms/step - loss: 0.0697 - accuracy: 0.9769 - val_loss: 0.0404 - val_accuracy: 0.9909
Epoch 5/20
95/95 [==============================] - 80s 843ms/step - loss: 0.0602 - accuracy: 0.9796 - val_loss: 0.0359 - val_accuracy: 0.9909
Epoch 6/20
95/95 [==============================] - 83s 877ms/step - loss: 0.0518 - accuracy: 0.9825 - val_loss: 0.0328 - val_accuracy: 0.9909
Epoch 7/20
95/95 [==============================] - 81s 851ms/step - loss: 0.0480 - accuracy: 0.9806 - val_loss: 0.0307 - val_accuracy: 0.9922
Epoch 8/20
95/95 [==============================] - 83s 872ms/step - loss: 0.0453 - accuracy: 0.9865 - val_loss: 0.0287 - val_accuracy: 0.9909
Epoch 9/20
95/95 [==============================] - 82s 864ms/step - loss: 0.0351 - accuracy: 0.9904 - val_loss: 0.0282 - val_accuracy: 0.9922
Epoch 10/20
95/95 [==============================] - 79s 834ms/step - loss: 0.0360 - accuracy: 0.9881 - val_loss: 0.0277 - val_accuracy: 0.9922
Epoch 11/20
95/95 [==============================] - 79s 834ms/step - loss: 0.0352 - accuracy: 0.9895 - val_loss: 0.0269 - val_accuracy: 0.9922
Epoch 12/20
95/95 [==============================] - 79s 827ms/step - loss: 0.0343 - accuracy: 0.9862 - val_loss: 0.0277 - val_accuracy: 0.9922
Epoch 13/20
95/95 [==============================] - 78s 826ms/step - loss: 0.0324 - accuracy: 0.9891 - val_loss: 0.0251 - val_accuracy: 0.9922
Epoch 14/20
95/95 [==============================] - 78s 825ms/step - loss: 0.0267 - accuracy: 0.9911 - val_loss: 0.0242 - val_accuracy: 0.9948
Epoch 15/20
95/95 [==============================] - 79s 835ms/step - loss: 0.0271 - accuracy: 0.9908 - val_loss: 0.0285 - val_accuracy: 0.9922
Epoch 16/20
95/95 [==============================] - 78s 825ms/step - loss: 0.0275 - accuracy: 0.9914 - val_loss: 0.0286 - val_accuracy: 0.9909
Epoch 17/20
95/95 [==============================] - 79s 828ms/step - loss: 0.0262 - accuracy: 0.9914 - val_loss: 0.0262 - val_accuracy: 0.9935
Epoch 18/20
95/95 [==============================] - 79s 829ms/step - loss: 0.0276 - accuracy: 0.9931 - val_loss: 0.0320 - val_accuracy: 0.9909
Epoch 19/20
95/95 [==============================] - 79s 834ms/step - loss: 0.0252 - accuracy: 0.9921 - val_loss: 0.0264 - val_accuracy: 0.9922
Epoch 20/20
95/95 [==============================] - 79s 831ms/step - loss: 0.0254 - accuracy: 0.9908 - val_loss: 0.0249 - val_accuracy: 0.9922
[INFO] evaluating network...
              precision    recall  f1-score   support

   with_mask       0.99      0.99      0.99       383
without_mask       0.99      0.99      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 [12]:
```
# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save("mask_detector.model", save_format="h5")

# 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")
plt.savefig("plot.png")
```

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



In [ ]:
```
```