

FLASH SALE — 20% OFF ALL my books and courses until Thursday at midnight EST!
10% of every purchase will be donated to The Child Mind Institute to help children/families suffering from
mental health issues during COVID-19.

01 20 55 52
DAYS HOURS MINUTES SECONDS
(<https://www.pyimagesearch.com/books-and-courses/>)

PYIMAGES EARCH

DEEP LEARNING ([HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/DEEP-LEARNING/](https://www.pyimagesearch.com/category/deep-learning/))

FACE APPLICATIONS ([HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/FACES/](https://www.pyimagesearch.com/category/faces/))

KERAS AND TENSORFLOW ([HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/KERAS-AND-TENSORFLOW/](https://www.pyimagesearch.com/category/keras-and-tensorflow/))

MEDICAL COMPUTER VISION ([HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/MEDICAL/](https://www.pyimagesearch.com/category/medical/))

OBJECT DETECTION ([HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/OBJECT-DETECTION/](https://www.pyimagesearch.com/category/object-detection/))

TUTORIALS ([HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/TUTORIALS/](https://www.pyimagesearch.com/category/tutorials/))

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

[Click here to download the source code to this post](#)

by **Adrian Rosebrock (<https://www.pyimagesearch.com/author/adrian/>)** on May 4, 2020
(<https://app.monstercampaigns.com/c/tortsem7qkvuxc4cyfi>)

In this tutorial, you will learn how to train a COVID-19 face mask detector with OpenCV, Keras/TensorFlow, and Deep Learning.

Last month, I authored a blog post on [detecting COVID-19 in X-ray images using deep learning \(https://www.pyimagesearch.com/2020/03/16/detecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning/\)](https://www.pyimagesearch.com/2020/03/16/detecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning/).

Readers really enjoyed learning from the timely, practical application of that tutorial, so today we are going to look at another COVID-related application of computer vision, **this one on detecting face masks with OpenCV and Keras/TensorFlow**.

I was inspired to author this tutorial after:

- 1 Receiving *numerous* requests from PyImageSearch readers asking that I write such a blog post
- 2 Seeing others implement their own solutions (my favorite being [Prajna Bhandary's \(https://www.linkedin.com/feed/update/urn%3Ali%3Aactivity%3A6655711815361761280/\)](https://www.linkedin.com/feed/update/urn%3Ali%3Aactivity%3A6655711815361761280/), which we are going to build from today)

If deployed correctly, the COVID-19 mask detector we're building here today could potentially be used to help ensure your safety and the safety of others (but I'll leave that to the medical professionals to decide on, implement, and distribute in the wild).

To learn how to create a COVID-19 face mask detector with OpenCV, Keras/TensorFlow, and Deep Learning, *just keep reading!*



Looking for the source code to this post?

[JUMP RIGHT TO THE DOWNLOADS SECTION](#) →

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

In this tutorial, we'll discuss our two-phase COVID-19 face mask detector, detailing how our computer vision/deep learning pipeline will be implemented.

From there, we'll review the dataset we'll be using to train our custom face mask detector.

I'll then show you how to implement a Python script to train a face mask detector on our dataset using Keras and TensorFlow.

We'll use this Python script to train a face mask detector and review the results.

Given the trained COVID-19 face mask detector, we'll proceed to implement two more additional Python scripts used to:

- 1 Detect COVID-19 face masks in *images*
- 2 Detect face masks in *real-time video streams*

We'll wrap up the post by looking at the results of applying our face mask detector.

I'll also provide some additional suggestions for further improvement.

Two-phase COVID-19 face mask detector

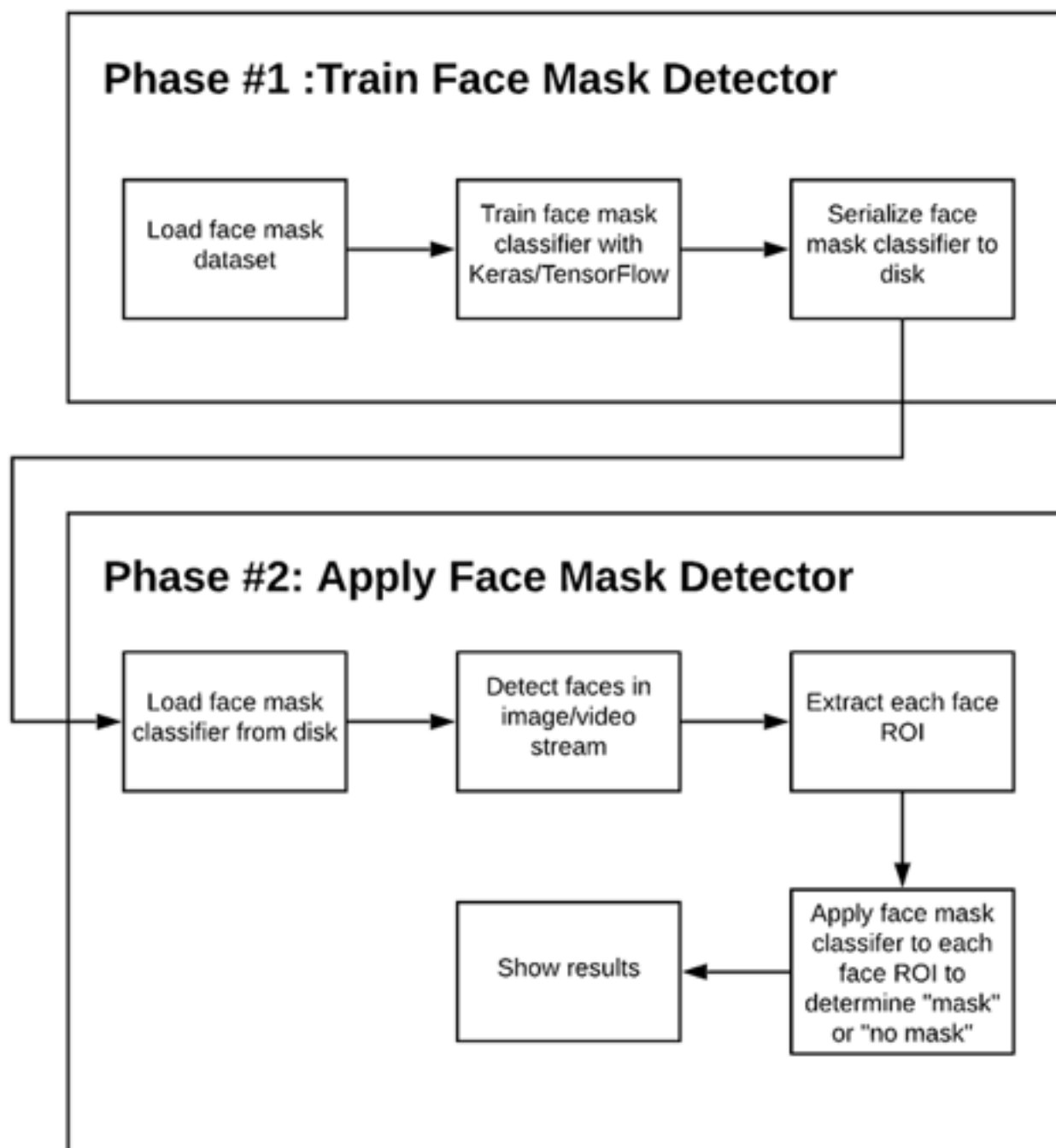


Figure 1: Phases and individual steps for building a COVID-19 face mask detector with computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras.

In order to train a custom face mask detector, we need to break our project into two distinct phases, each with its own respective sub-steps (as shown by **Figure 1** above):

- 1 **Training:** Here we'll focus on loading our face mask detection dataset from disk, training a model (using Keras/TensorFlow) on this dataset, and then serializing the face mask detector to disk
- 2 **Deployment:** Once the face mask detector is trained, we can then move on to loading the mask detector, performing face detection, and then classifying each face as `with_mask` or `without_mask`

We'll review each of these phases and associated subsets in detail in the remainder of this tutorial, but in the meantime, let's take a look at the dataset we'll be using to train our COVID-19 face mask detector.

Our COVID-19 face mask detection dataset

Figure 2: A face mask detection dataset consists of “with mask” and “without mask” images. We will use the dataset to build a COVID-19 face mask detector with computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras.

The dataset we'll be using here today was created by PyImageSearch reader [Prajna Bhandary](https://www.linkedin.com/feed/update/urn%3Ali%3Aactivity%3A6655711815361761280/).
(<https://www.linkedin.com/feed/update/urn%3Ali%3Aactivity%3A6655711815361761280/>)

This dataset consists of **1,376 images** belonging to two classes:

- `with_mask` : 690 images
- `without_mask` : 686 images

Our goal is to train a custom deep learning model to detect whether a person *is* or *is not* wearing a mask.

Note: For convenience, I have included the dataset created by Prajna in the “**Downloads**” section of this tutorial.

How was our face mask dataset created?

Prajna, like me, has been feeling down and depressed about the state of the world — thousands of people are dying each day, and for many of us, there is very little (if anything) we can do.

To help keep her spirits up, Prajna decided to distract herself by applying computer vision and deep learning to solve a real-world problem:

- Best case scenario — she could use her project to help others

- Worst case scenario — it gave her a much needed mental escape

Either way, it's win-win!

As programmers, developers, and computer vision/deep learning practitioners, we can *all* take a page from Prajna's book — let your skills become your distraction and your haven.

To create this dataset, Prajna had the ingenious solution of:

- 1 Taking *normal images of faces*
- 2 Then creating a *custom computer vision Python script* to add face masks to them, **thereby creating an *artificial* (but still real-world applicable) dataset**

This method is actually a lot easier than it sounds once you [apply facial landmarks to the problem \(https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/\)](https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/).

Facial landmarks allow us to automatically infer the location of facial structures, including:

- Eyes
- Eyebrows
- Nose
- Mouth
- Jawline

To use facial landmarks to build a dataset of faces wearing face masks, we need to first start with an image of a person *not* wearing a face mask:

Figure 3: To build a COVID-19/Coronavirus pandemic face mask dataset, we'll first start with a photograph of

someone not
wearing a
face.

From there, we apply face detection to compute the bounding box location of the face in the image:

Figure 4:

The next
step is to
apply
face
detection.
Here
we've
used a
deep
learning
method
to
perform
face
detection
with
OpenCV.

Once we know *where* in the image the face is, we can extract the face Region of Interest (ROI):





Figure 5: The next step is to extract the face ROI with OpenCV and NumPy slicing.

And from there, we apply facial landmarks, allowing us to localize the eyes, nose, mouth, etc.:



Figure 6: Then, we **detect facial landmarks** (<https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/>) using dlib so that we know where to place a mask on the face.

Next, we need an image of a mask (with a transparent background) such as the one below:






Figure 7: An example of a COVID-19/Coronavirus face mask/shield. This face mask will be overlaid on the original face ROI automatically since we know the face landmark locations.

This mask will be *automatically* applied to the face by using the facial landmarks (namely the points along the chin and nose) to compute *where* the mask will be placed.

The mask is then resized and rotated, placing it on the face:

Figure 8:

In this figure, the face mask is placed on the person's face in the original frame. It is difficult to tell at a glance that the COVID-19 mask has been applied with computer vision by way of OpenCV and dlib face landmarks.

We can then repeat this process for all of our input images, thereby creating our artificial face mask dataset:

Figure 9: An artificial set of COVID-19 face mask images is shown. This set will be part of our “with mask” / “without mask” dataset for COVID-19 face mask detection with computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras.

However, there is a caveat you should be aware of when using this method to artificially create a dataset!

If you use a set of images to create an artificial dataset of people wearing masks, **you cannot “re-use” the images *without masks* in your training set — you still need to gather non-face mask images that were *not* used in the artificial generation process!**

If you include the original images used to generate face mask samples as non-face mask samples, your model will become heavily biased and fail to generalize well. Avoid that at all costs by taking the time to gather new examples of faces without masks.

Covering how to use facial landmarks to apply a mask to a face is outside the scope of this tutorial, but if you want to learn more about it, I would suggest:

- 1 Referring to [Prajna’s GitHub repository](https://github.com/prajnasb/observations/tree/master/mask_classifier/Data_Generator) ([\(https://github.com/prajnasb/observations/tree/master/mask_classifier/Data_Generator\)](https://github.com/prajnasb/observations/tree/master/mask_classifier/Data_Generator))
- 2 Reading this tutorial on the PyImageSearch blog where I discuss [how to use facial landmarks to automatically apply sunglasses to a face](https://www.pyimagesearch.com/2018/11/05/creating-gifs-with-opencv/) ([\(https://www.pyimagesearch.com/2018/11/05/creating-gifs-with-opencv/\)](https://www.pyimagesearch.com/2018/11/05/creating-gifs-with-opencv/))

The same principle from my sunglasses post applies to building an artificial face mask dataset — use the facial landmarks to infer the facial structures, rotate and resize the mask, and then apply it to the image.

Project structure

Once you grab the files from the “**Downloads**” section of this article, you’ll be presented with the following directory structure:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
1. | $ tree --dirsfirst --filelimit 10
2. | .
3. | └─ dataset
4. |   └─ with_mask [690 entries]
5. |   └─ without_mask [686 entries]
```

```
6. | └─ examples
7. |   └─ example_01.png
8. |   └─ example_02.png
9. |   └─ example_03.png
10. | └─ face_detector
11. |   └─ deploy.prototxt
12. |   └─ res10_300x300_ssd_iter_140000.caffemodel
13. | └─ detect_mask_image.py
14. | └─ detect_mask_video.py
15. | └─ mask_detector.model
16. | └─ plot.png
17. | └─ train_mask_detector.py
18. |
19. | 5 directories, 10 files
```

The

`dataset/` directory contains the data described in the “*Our COVID-19 face mask detection dataset*” section.

Three image

`examples/` are provided so that you can test the static image face mask detector.

We’ll be reviewing three Python scripts in this tutorial:

- `train_mask_detector.py` : Accepts our input dataset and fine-tunes MobileNetV2 upon it to create our `mask_detector.model` . A training history `plot.png` containing accuracy/loss curves is also produced
- `detect_mask_image.py` : Performs face mask detection in static images
- `detect_mask_video.py` : Using your webcam, this script applies face mask detection to every frame in the stream

In the next two sections, we will train our face mask detector.

Implementing our COVID-19 face mask detector training script with Keras and TensorFlow

Now that we’ve reviewed our face mask dataset, let’s learn how we can use Keras and

TensorFlow to train a classifier to *automatically* detect whether a person is wearing a mask or not.

To accomplish this task, we'll be fine-tuning the **MobileNet V2 architecture** (<https://arxiv.org/abs/1801.04381>), a highly efficient architecture that can be applied to embedded devices with limited computational capacity (ex., Raspberry Pi, Google Coral, NVIDIA Jetson Nano, etc.).

Note: *If your interest is embedded computer vision, be sure to check out my **Raspberry Pi for Computer Vision book** (<https://www.pyimagesearch.com/raspberry-pi-for-computer-vision/>) which covers working with computationally limited devices for computer vision and deep learning.*

Deploying our face mask detector to embedded devices could reduce the cost of manufacturing such face mask detection systems, hence why we choose to use this architecture.

Let's get started!

Open up the

`train_mask_detector.py` file in your directory structure, and insert the following code:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
1. | # import the necessary packages
2. | from tensorflow.keras.preprocessing.image import ImageDataGenerator
3. | from tensorflow.keras.applications import MobileNetV2
4. | from tensorflow.keras.layers import AveragePooling2D
5. | from tensorflow.keras.layers import Dropout
6. | from tensorflow.keras.layers import Flatten
7. | from tensorflow.keras.layers import Dense
8. | from tensorflow.keras.layers import Input
9. | from tensorflow.keras.models import Model
10. | from tensorflow.keras.optimizers import Adam
11. | from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
12. | from tensorflow.keras.preprocessing.image import img_to_array
13. | from tensorflow.keras.preprocessing.image import load_img
14. | from tensorflow.keras.utils import to_categorical
15. | from sklearn.preprocessing import LabelBinarizer
16. | from sklearn.model_selection import train_test_split
17. | from sklearn.metrics import classification_report
18. | from imutils import paths
19. | import matplotlib.pyplot as plt
20. | import numpy as np
21. | import argparse
22. | import os
```

The imports for our training script may look intimidating to you either because there are so many or you are new to deep learning. If you are new, I would recommend reading both my **Keras tutorial** (<https://www.pyimagesearch.com/2018/09/10/keras-tutorial-how-to-get-started-with-keras-deep-learning-and-python/>) and **fine-tuning tutorial** (<https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/>) before moving forward.

Our set of

`tensorflow.keras` imports allow for:

- Data augmentation
- Loading the MobilNetV2 classifier (we will fine-tune this model with pre-trained **ImageNet** (<http://www.image-net.org/>) weights)
- Building a new fully-connected (FC) head
- Pre-processing
- Loading image data

We'll use scikit-learn (

`sklearn`) for binarizing class labels, segmenting our dataset, and printing a classification report.

My **imutils** (<https://github.com/jrosebr1/imutils/>)

`paths` implementation will help us to find and list images in our dataset. And we'll use `matplotlib` to plot our training curves.

To install the necessary software so that these imports are available to you, be sure to follow either one of my Tensorflow 2.0+ installation guides:

- **How to install TensorFlow 2.0 on Ubuntu** (<https://www.pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-ubuntu/>)

How to install TensorFlow 2.0 on macOS

- [How to install TensorFlow 2.0 on macOS](#)

(<https://www.pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-macos/>)

Let's go ahead and parse a few [command line arguments](#)

(<https://www.pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/>) that are required to launch our script from a terminal:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
24. | # construct the argument parser and parse the arguments
25. | ap = argparse.ArgumentParser()
26. | ap.add_argument("-d", "--dataset", required=True,
27. |     help="path to input dataset")
28. | ap.add_argument("-p", "--plot", type=str, default="plot.png",
29. |     help="path to output loss/accuracy plot")
30. | ap.add_argument("-m", "--model", type=str,
31. |     default="mask_detector.model",
32. |     help="path to output face mask detector model")
33. | args = vars(ap.parse_args())
```

Our command line arguments include:

- `--dataset` : The path to the input dataset of faces and and faces with masks
- `--plot` : The path to your output training history plot, which will be generated using `matplotlib`
- `--model` : The path to the resulting serialized face mask classification model

I like to define my deep learning hyperparameters in one place:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
35. | # initialize the initial learning rate, number of epochs to train for,
36. | # and batch size
37. | INIT_LR = 1e-4
38. | EPOCHS = 20
39. | BS = 32
```

Here, I've specified hyperparameter constants including my initial learning rate, number of training epochs, and batch size. Later, we will be applying a [learning rate decay schedule](#) (<https://www.pyimagesearch.com/2019/07/22/keras-learning-rate-schedules-and-decay/>), which is why we've named the learning rate variable

INIT_LR .

At this point, we're ready to load and pre-process our training data:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
41. | # grab the list of images in our dataset directory, then initialize
42. | # the list of data (i.e., images) and class images
43. | print("[INFO] loading images...")
44. | imagePath = list(paths.list_images(args["dataset"]))
45. | data = []
46. | labels = []
47. |
48. | # loop over the image paths
49. | for imagePath in imagePath:
50. |     # extract the class label from the filename
51. |     label = imagePath.split(os.path.sep)[-2]
52. |
53. |     # load the input image (224x224) and preprocess it
54. |     image = load_img(imagePath, target_size=(224, 224))
55. |     image = img_to_array(image)
56. |     image = preprocess_input(image)
57. |
58. |     # update the data and labels lists, respectively
59. |     data.append(image)
60. |     labels.append(label)
61. |
62. | # convert the data and labels to NumPy arrays
63. | data = np.array(data, dtype="float32")
64. | labels = np.array(labels)
```

In this block, we are:

- Grabbing all of the `imagePaths` in the dataset (**Line 44**)
- Initializing `data` and `labels` lists (**Lines 45 and 46**)
- Looping over the `imagePaths` and loading + pre-processing images (**Lines 49-60**). Pre-processing steps include resizing to 224×224 pixels, conversion to array format, and scaling the pixel intensities in the input image to the range $[-1, 1]$ (via the `preprocess_input` convenience function)
- Appending the pre-processed `image` and associated `label` to the `data` and `labels` lists, respectively (**Lines 59 and 60**)
- Ensuring our training data is in NumPy array format (**Lines 63 and 64**)

The above lines of code assume that your entire dataset is small enough to fit into memory. If your dataset is larger than the memory you have available, I suggest using HDF5, a strategy I cover in [**Deep Learning for Computer Vision with Python**](https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/) ([**\(https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/\)**](https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/)) (Practitioner Bundle Chapters 9 and 10).

Our data preparation work isn't done yet. Next, we'll encode our

`labels`, partition our dataset, and prepare for [**data augmentation**](https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/) ([**\(https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/\)**](https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/)):

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
66. | # perform one-hot encoding on the labels
67. | lb = LabelBinarizer()
68. | labels = lb.fit_transform(labels)
69. | labels = to_categorical(labels)
70. |
71. | # partition the data into training and testing splits using 75% of
72. | # the data for training and the remaining 25% for testing
73. | (trainX, testX, trainY, testY) = train_test_split(data, labels,
74. |     test_size=0.20, stratify=labels, random_state=42)
75. |
76. | # construct the training image generator for data augmentation
77. | aug = ImageDataGenerator(
78. |     rotation_range=20,
79. |     zoom_range=0.15,
80. |     width_shift_range=0.2,
81. |     height_shift_range=0.2,
82. |     shear_range=0.15,
83. |     horizontal_flip=True,
84. |     fill_mode="nearest")
```

Lines 67-69 one-hot encode our class labels, meaning that our data will be in the following format:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
1. | $ python train_mask_detector.py --dataset dataset
2. | [INFO] loading images...
3. | -> (trainX, testX, trainY, testY) = train_test_split(data, labels,
4. | (Pdb) labels[500:]
5. | array([[1., 0.],
6. |        [1., 0.],
7. |
8. |        [1., 0.],
9. |        ...,
10. |        [0., 1.],
11. |        [0., 1.],
12. |        [0., 1.]], dtype=float32)
(Pdb)
```

As you can see, each element of our

`labels` array consists of an array in which only one index is “hot” (i.e., 1).

Using scikit-learn’s convenience method, **Lines 73 and 74** segment our data into 80% training and the remaining 20% for testing.

During training, we’ll be applying on-the-fly mutations to our images in an effort to improve generalization. This is known as **data augmentation**

(<https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/>), where the random rotation, zoom, shear, shift, and flip parameters are established on **Lines 77-84**. We’ll use the

`aug` object at training time.

But first, we need to prepare MobileNetV2 for **fine-tuning**

(<https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/>):

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
86. | # load the MobileNetV2 network, ensuring the head FC layer sets are
87. | # left off
88. | baseModel = MobileNetV2(weights="imagenet", include_top=False,
89. |     input_tensor=Input(shape=(224, 224, 3)))
90. |
91. | # construct the head of the model that will be placed on top of the
92. | # the base model
93. | headModel = baseModel.output
94. | headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
95. | headModel = Flatten(name="flatten")(headModel)
96. | headModel = Dense(128, activation="relu")(headModel)
97. | headModel = Dropout(0.5)(headModel)
98. | headModel = Dense(2, activation="softmax")(headModel)
99. |
100. | # place the head FC model on top of the base model (this will become
101. | # the actual model we will train)
102. | model = Model(inputs=baseModel.input, outputs=headModel)
103. |
104. | # loop over all layers in the base model and freeze them so they will
105. | # *not* be updated during the first training process
106. | for layer in baseModel.layers:
107. |     layer.trainable = False
```

Fine-tuning setup is a three-step process:

- 1 Load MobileNet with pre-trained [ImageNet \(http://www.image-net.org/\)](http://www.image-net.org/) weights, leaving off head of network (**Lines 88 and 89**)
- 2 Construct a new FC head, and append it to the base in place of the old head (**Lines 93-102**)
- 3 Freeze the base layers of the network (**Lines 106 and 107**). The weights of these base layers will not be updated during the process of backpropagation, whereas the head layer weights *will* be tuned.

Fine-tuning is a strategy I nearly always recommend to establish a baseline model while saving considerable time. To learn more about the theory, purpose, and strategy, please refer to my [fine-tuning blog posts \(https://www.pyimagesearch.com/tag/fine-tuning/\)](https://www.pyimagesearch.com/tag/fine-tuning/) and [Deep Learning for Computer Vision with Python \(https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/\)](https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/) (Practitioner Bundle Chapter 5).

With our data prepared and model architecture in place for fine-tuning, we're now ready to compile and train our face mask detector network:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
109. | # compile our model
110. | print("[INFO] compiling model...")
111. | opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
112. | model.compile(loss="binary_crossentropy", optimizer=opt,
113. |               metrics=["accuracy"])
114. |
115. | # train the head of the network
116. | print("[INFO] training head...")
117. | H = model.fit(
118. |     aug.flow(trainX, trainY, batch_size=BS),
119. |     steps_per_epoch=len(trainX) // BS,
120. |     validation_data=(testX, testY),
121. |     validation_steps=len(testX) // BS,
122. |     epochs=EPOCHS)
```

Lines 111-113

`compile` our model with the `Adam` optimizer, a [learning rate decay schedule \(https://www.pyimagesearch.com/2019/07/22/keras-learning-rate-schedules-and-decay/\)](https://www.pyimagesearch.com/2019/07/22/keras-learning-rate-schedules-and-decay/), and binary cross-entropy. If you're building from this training script with > 2 classes, be sure to use categorical cross-entropy.

Face mask training is launched via **Lines 117-122**. Notice how our data augmentation object (`aug`) will be providing batches of mutated image data.

Once training is complete, we'll evaluate the resulting model on the test set:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
124. | # make predictions on the testing set
125. | print("[INFO] evaluating network...")
126. | predIdxs = model.predict(testX, batch_size=BS)
127. |
128. | # for each image in the testing set we need to find the index of the
129. | # label with corresponding largest predicted probability
130. | predIdxs = np.argmax(predIdxs, axis=1)
131. |
132. | # show a nicely formatted classification report
133. | print(classification_report(testY.argmax(axis=1), predIdxs,
134. |     target_names=lb.classes_))
135. |
136. | # serialize the model to disk
137. | print("[INFO] saving mask detector model...")
138. | model.save(args["model"], save_format="h5")
```

Here, **Lines 126-130** make predictions on the test set, grabbing the highest probability class label indices. Then, we

`print` a classification report in the terminal for inspection.

Line 138 serializes our face mask classification

`model` to disk.

Our last step is to plot our accuracy and loss curves:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
140. | # plot the training loss and accuracy
141. | N = EPOCHS
142. | plt.style.use("ggplot")
143. | plt.figure()
144. | plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
145. | plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
146. |
147. | plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
148. | plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
149. | plt.title("Training Loss and Accuracy")
150. | plt.xlabel("Epoch #")
151. | plt.ylabel("Loss/Accuracy")
152. | plt.legend(loc="lower left")
```

```
152. | plt.savefig(args["plot"])
```

Once our plot is ready, **Line 152** saves the figure to disk using the

```
--plot filepath.
```

Training the COVID-19 face mask detector with Keras/TensorFlow

We are now ready to train our face mask detector using Keras, TensorFlow, and Deep Learning.

Make sure you have used the **“Downloads”** section of this tutorial to download the source code and face mask dataset.

From there, open up a terminal, and execute the following command:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
1. | $ python train_mask_detector.py --dataset dataset
2. | [INFO] loading images...
3. | [INFO] compiling model...
4. | [INFO] training head...
5. | Train for 34 steps, validate on 276 samples
6. | Epoch 1/20
7. | 34/34 [=====] - 30s 885ms/step - loss: 0.6431 - accuracy:
   | 0.6676 - val_loss: 0.3696 - val_accuracy: 0.8242
8. | Epoch 2/20
9. | 34/34 [=====] - 29s 853ms/step - loss: 0.3507 - accuracy:
   | 0.8567 - val_loss: 0.1964 - val_accuracy: 0.9375
10. | Epoch 3/20
11. | 34/34 [=====] - 27s 800ms/step - loss: 0.2792 - accuracy:
   | 0.8820 - val_loss: 0.1383 - val_accuracy: 0.9531
12. | Epoch 4/20
13. | 34/34 [=====] - 28s 814ms/step - loss: 0.2196 - accuracy:
   | 0.9148 - val_loss: 0.1306 - val_accuracy: 0.9492
14. | Epoch 5/20
15. | 34/34 [=====] - 27s 792ms/step - loss: 0.2006 - accuracy:
   | 0.9213 - val_loss: 0.0863 - val_accuracy: 0.9688
16. | ...
17. | Epoch 16/20
18. | 34/34 [=====] - 27s 801ms/step - loss: 0.0767 - accuracy:
   | 0.9766 - val_loss: 0.0291 - val_accuracy: 0.9922
19. | Epoch 17/20
20. | 34/34 [=====] - 27s 795ms/step - loss: 0.1042 - accuracy:
   | 0.9616 - val_loss: 0.0243 - val_accuracy: 1.0000
21. | Epoch 18/20
22. | 34/34 [=====] - 27s 796ms/step - loss: 0.0804 - accuracy:
   | 0.9672 - val_loss: 0.0244 - val_accuracy: 0.9961
23. | Epoch 19/20
24. | 34/34 [=====] - 27s 793ms/step - loss: 0.0836 - accuracy:
```

```

24. | 0.9710 - val_loss: 0.0440 - val_accuracy: 0.9883
25. | Epoch 20/20
26. | 34/34 [=====] - 28s 838ms/step - loss: 0.0717 - accuracy:
    | 0.9710 - val_loss: 0.0270 - val_accuracy: 0.9922
27. | [INFO] evaluating network...
28. |           precision    recall  f1-score   support
29. |
30. |   with_mask           0.99         1.00         0.99         138
31. |  without_mask          1.00         0.99         0.99         138
32. |
33. |       accuracy
34. |   macro avg           0.99         0.99         0.99         276
35. |  weighted avg           0.99         0.99         0.99         276

```

Figure 10: COVID-19 face mask detector training accuracy/loss curves demonstrate high accuracy and little signs of overfitting on the data. We're now ready to apply our knowledge of computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras to perform face mask detection.

As you can see, we are obtaining **~99% accuracy** on our test set.

Looking at **Figure 10**, we can see there are little signs of overfitting, with the validation loss *lower* than the training loss (a phenomenon I discuss [in this blog post](https://www.pyimagesearch.com/2019/10/14/why-is-my-validation-loss-lower-than-my-training-loss/) (<https://www.pyimagesearch.com/2019/10/14/why-is-my-validation-loss-lower-than-my-training-loss/>)).

Given these results, we are hopeful that our model will generalize well to images *outside* our training and testing set.

Implementing our COVID-19 face mask detector for images with OpenCV

Now that our face mask detector is trained, let's learn how we can:

- 1 Load an input image from disk
- 2 Detect faces in the image
- 3 Apply our face mask detector to classify the face as either `with_mask` or `without_mask`

Open up the

`detect_mask_image.py` file in your directory structure, and let's get started:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
1. | # import the necessary packages
2. | from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
3. | from tensorflow.keras.preprocessing.image import img_to_array
4. | from tensorflow.keras.models import load_model
5. | import numpy as np
6. | import argparse
7. | import cv2
8. | import os
```

Our driver script requires three TensorFlow/Keras imports to (1) load our MaskNet model and (2) pre-process the input image.

OpenCV is required for display and image manipulations.

The next step is to parse **command line arguments**

(<https://www.pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/>):

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
10. | # construct the argument parser and parse the arguments
11. | ap = argparse.ArgumentParser()
12. | ap.add_argument("-i", "--image", required=True,
13. |     help="path to input image")
14. | ap.add_argument("-f", "--face", type=str,
15. |     default="face_detector",
16. |     help="path to face detector model directory")
17. | ap.add_argument("-m", "--model", type=str,
18. |     default="mask_detector.model",
19. |     help="path to trained face mask detector model")
20. | ap.add_argument("-c", "--confidence", type=float, default=0.5,
21. |     help="minimum probability to filter weak detections")
22. | args = vars(ap.parse_args())
```

Our four command line arguments include:

- `--image` : The path to the input image containing faces for inference
- `--face` : The path to the face detector model directory (we need to localize faces prior to classifying them)
- `--model` : The path to the face mask detector model that we trained earlier in this tutorial
- `--confidence` : An optional probability threshold can be set to override 50% to filter

weak face detections

Next, we'll load both our face detector and face mask classifier models:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
24. | # load our serialized face detector model from disk
25. | print("[INFO] loading face detector model...")
26. | prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
27. | weightsPath = os.path.sep.join([args["face"],
28. |     "res10_300x300_ssd_iter_140000.caffemodel"])
29. | net = cv2.dnn.readNet(prototxtPath, weightsPath)
30. |
31. | # load the face mask detector model from disk
32. | print("[INFO] loading face mask detector model...")
33. | model = load_model(args["model"])
```

With our deep learning models now in memory, our next step is to load and pre-process an input image:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
35. | # load the input image from disk, clone it, and grab the image spatial
36. | # dimensions
37. | image = cv2.imread(args["image"])
38. | orig = image.copy()
39. | (h, w) = image.shape[:2]
40. |
41. | # construct a blob from the image
42. | blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),
43. |     (104.0, 177.0, 123.0))
44. |
45. | # pass the blob through the network and obtain the face detections
46. | print("[INFO] computing face detections...")
47. | net.setInput(blob)
48. | detections = net.forward()
```

Upon loading our

`--image` from disk (**Line 37**), we make a copy and grab frame dimensions for future scaling and display purposes (**Lines 38 and 39**).

Pre-processing is handled by [OpenCV's blobFromImage function](https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/)

(<https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/>) (**Lines 42 and 43**). As shown in the parameters, we resize to 300×300 pixels and perform mean subtraction.

Lines 47 and 48 then perform face detection to localize *where* in the image all faces are.

Once we know where each face is predicted to be, we'll ensure they meet the

`--confidence` threshold before we extract the faceROIs:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
50. | # loop over the detections
51. | for i in range(0, detections.shape[2]):
52. |     # extract the confidence (i.e., probability) associated with
53. |     # the detection
54. |     confidence = detections[0, 0, i, 2]
55. |
56. |     # filter out weak detections by ensuring the confidence is
57. |     # greater than the minimum confidence
58. |     if confidence > args["confidence"]:
59. |         # compute the (x, y)-coordinates of the bounding box for
60. |         # the object
61. |         box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
62. |         (startX, startY, endX, endY) = box.astype("int")
63. |
64. |         # ensure the bounding boxes fall within the dimensions of
65. |         # the frame
66. |         (startX, startY) = (max(0, startX), max(0, startY))
67. |         (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
```

Here, we loop over our

`detections` and extract the `confidence` to measure against the `--confidence` threshold (**Lines 51-58**).

We then compute bounding

`box` value for a particular face and ensure that the box falls within the boundaries of the image (**Lines 61-67**).

Next, we'll run the face ROI through our MaskNet model:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
69. |     # extract the face ROI, convert it from BGR to RGB channel
70. |     # ordering, resize it to 224x224, and preprocess it
71. |     face = image[startY:endY, startX:endX]
72. |     face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
73. |     face = cv2.resize(face, (224, 224))
74. |     face = img_to_array(face)
75. |     face = preprocess_input(face)
76. |     face = np.expand_dims(face, axis=0)
```

```

77. |
78. |         # pass the face through the model to determine if the face
79. |         # has a mask or not
80. |         (mask, withoutMask) = model.predict(face)[0]

```

In this block, we:

- Extract the `face` ROI via NumPy slicing (**Line 71**)
- Pre-process the ROI the same way we did during training (**Lines 72-76**)
- Perform mask detection to predict `with_mask` or `without_mask` (**Line 80**)

From here, we will annotate and display the result!

```

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
82. |         # determine the class label and color we'll use to draw
83. |         # the bounding box and text
84. |         label = "Mask" if mask > withoutMask else "No Mask"
85. |         color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
86. |
87. |         # include the probability in the label
88. |         label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
89. |
90. |         # display the label and bounding box rectangle on the output
91. |         # frame
92. |         cv2.putText(image, label, (startX, startY - 10),
93. |                     cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
94. |         cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)
95. |
96. |     # show the output image
97. |     cv2.imshow("Output", image)
98. |     cv2.waitKey(0)

```

First, we determine the class

`label` based on probabilities returned by the mask detector model (**Line 84**) and assign an associated `color` for the annotation (**Line 85**). The `color` will be “green” for `with_mask` and “red” for `without_mask`.

We then draw the

`label` text (including class and probability), as well as a bounding box `rectangle` for the face, using OpenCV drawing functions (**Lines 92-94**).

Once all detections have been processed, **Lines 97 and 98** display the output

image .

COVID-19 face mask detection in images with OpenCV

Let's put our COVID-19 face mask detector to work!

Make sure you have used the “**Downloads**” section of this tutorial to download the source code, example images, and pre-trained face mask detector.

From there, open up a terminal, and execute the following command:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
```

```
1. | $ python detect_mask_image.py --image examples/example_01.png
2. | [INFO] loading face detector model...
3. | [INFO] loading face mask detector model...
4. | [INFO] computing face detections...
```

Figure 11: Is this man wearing a COVID-19/Coronavirus face mask in public? Yes, he is and our computer vision and deep learning method using Python, OpenCV, and TensorFlow/Keras has made it possible to detect the presence of the mask automatically. ([Image Source \(https://www.sunset.com/lifestyle/shopping/fashionable-flu-masks-germ-protection\)](https://www.sunset.com/lifestyle/shopping/fashionable-flu-masks-germ-protection))

As you can see, our face mask detector correctly labeled this image as

Mask .

Let's try another image, this one of a person *not* wearing a face mask:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
```

```
1. | $ python detect_mask_image.py --image examples/example_02.png
2. | [INFO] loading face detector model...
3. | [INFO] loading face mask detector model...
4. | [INFO] computing face detections...
```

Figure 12: Uh oh. I'm not wearing a COVID-19 face mask in this picture. Using Python, OpenCV,

and
TensorFlow/Keras,
our system has
correctly detected
“No Mask” for my
face.

Our face mask detector has correctly predicted

No Mask .

Let’s try one final image:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
1. | $ python detect_mask_image.py --image examples/example_03.png
2. | [INFO] loading face detector model...
3. | [INFO] loading face mask detector model...
4. | [INFO] computing face detections...
```

Figure 13: What is going on in this result? Why is the lady in the foreground not detected as wearing a COVID-19 face mask? Has our COVID-19 face mask detector built with computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras failed us?

([Image Source](https://www.medicaldevice-network.com/news/coronavirus-outbreak-mask-production/)
(<https://www.medicaldevice-network.com/news/coronavirus-outbreak-mask-production/>))

What happened here?

Why is it that we were able to *detect* the faces of the two gentlemen in the background and *correctly classify* mask/no mask for them, but we could not detect the woman in the foreground?

I discuss the reason for this issue in the “*Suggestions for further improvement*” section later in this tutorial, **but the gist is that we’re *too reliant* on our two-stage process.**

Keep in mind that in order to classify whether or not a person is wearing in mask, we first

need to perform face detection — **if a face is not found (which is what happened in this image), then the mask detector cannot be applied!**

The reason we cannot detect the face in the foreground is because:

- 1 It's too obscured by the mask
- 2 The dataset used to train the *face detector* did not contain example images of people wearing face masks

Therefore, if a large portion of the face is occluded, our face detector will likely fail to detect the face.

Again, I discuss this problem in more detail, including how to improve the accuracy of our mask detector, in the “*Suggestions for further improvement*” section of this tutorial.

Implementing our COVID-19 face mask detector in real-time video streams with OpenCV

At this point, we know we can apply face mask detection to static images — ***but what about real-time video streams?***

Is our COVID-19 face mask detector capable of running in real-time?

Let's find out.

Open up the

`detect_mask_video.py` file in your directory structure, and insert the following code:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
1. | # import the necessary packages
2. | from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
3. | from tensorflow.keras.preprocessing.image import img_to_array
4. | from tensorflow.keras.models import load_model
5. | from imutils.video import VideoStream
6. | import numpy as np
7. | import argparse
8. | import imutils
9. | import time
10. | import cv2
11. | import os
```

The algorithm for this script is the same, but it is pieced together in such a way to allow for

The algorithm for this script is the same, but it is pieced together in such a way to allow for processing every frame of your webcam stream.

Thus, the only difference when it comes to imports is that we need a

`VideoStream` class and `time`. Both of these will help us to work with the stream. We'll also take advantage of `imutils` for its aspect-aware resizing method.

Our face detection/mask prediction logic for this script is in the

`detect_and_predict_mask` function:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
13. | def detect_and_predict_mask(frame, faceNet, maskNet):
14. |     # grab the dimensions of the frame and then construct a blob
15. |     # from it
16. |     (h, w) = frame.shape[:2]
17. |     blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
18. |         (104.0, 177.0, 123.0))
19. |
20. |     # pass the blob through the network and obtain the face detections
21. |     faceNet.setInput(blob)
22. |     detections = faceNet.forward()
23. |
24. |     # initialize our list of faces, their corresponding locations,
25. |     # and the list of predictions from our face mask network
26. |     faces = []
27. |     locs = []
28. |     preds = []
```

By defining this convenience function here, our frame processing loop will be a little easier to read later.

This function detects faces and then applies our face mask classifier to each face ROI.

Such a function consolidates our code — it could even be moved to a separate Python file if you so choose.

Our

`detect_and_predict_mask` function accepts three parameters:

- `frame` : A frame from our stream
- `faceNet` : The model used to detect where in the image faces are
- `maskNet` : Our COVID-19 face mask classifier model

Inside, we construct a

`blob`, `detect faces`, and initialize lists, two of which the function is set to return. These lists include our `faces` (i.e., ROIs), `locs` (the face locations), and `preds` (the list of mask/no mask predictions).

From here, we'll loop over the face

`detections` :

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
30. |     # loop over the detections
31. |     for i in range(0, detections.shape[2]):
32. |         # extract the confidence (i.e., probability) associated with
33. |         # the detection
34. |         confidence = detections[0, 0, i, 2]
35. |
36. |         # filter out weak detections by ensuring the confidence is
37. |         # greater than the minimum confidence
38. |         if confidence > args["confidence"]:
39. |             # compute the (x, y)-coordinates of the bounding box for
40. |             # the object
41. |             box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
42. |             (startX, startY, endX, endY) = box.astype("int")
43. |
44. |             # ensure the bounding boxes fall within the dimensions of
45. |             # the frame
46. |             (startX, startY) = (max(0, startX), max(0, startY))
47. |             (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
```

Inside the loop, we filter out weak detections (**Lines 34-38**) and extract bounding boxes while ensuring bounding box coordinates do not fall outside the bounds of the image (**Lines 41-47**).

Next, we'll add face ROIs to two of our corresponding lists:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
49. |         # extract the face ROI, convert it from BGR to RGB channel
50. |         # ordering, resize it to 224x224, and preprocess it
51. |         face = frame[startY:endY, startX:endX]
52. |         face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
53. |         face = cv2.resize(face, (224, 224))
54. |         face = img_to_array(face)
55. |         face = preprocess_input(face)
56. |         face = np.expand_dims(face, axis=0)
57. |
```

```

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

```

After extracting face ROIs and pre-processing (**Lines 51-56**), we append the the

`face` ROIs and bounding boxes to their respective lists.

We're now ready to run our

`faces` through our mask predictor:

```

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
63. |         # only make a predictions if at least one face was detected
64. |         if len(faces) > 0:
65. |             # for faster inference we'll make batch predictions on *all*
66. |             # faces at the same time rather than one-by-one predictions
67. |             # in the above `for` loop
68. |             preds = maskNet.predict(faces)
69. |
70. |         # return a 2-tuple of the face locations and their corresponding
71. |         # locations
72. |         return (locs, preds)

```

The logic here is built for speed. First we ensure at least one face was detected (**Line 64**) — if not, we'll

```

return empty_preds .

```

Secondly, we are performing inference on our *entire batch* of

`faces` in the frame so that our pipeline is faster (**Line 68**). It wouldn't make sense to write another loop to make predictions on each face individually due to the overhead (especially if you are using a GPU that requires a lot of overhead communication on your system bus). **It is more efficient to perform predictions in batch.**

Line 72 returns our face bounding box locations and corresponding mask/not mask predictions to the caller.

Next, we'll define our **command line arguments**

#!/usr/bin/env python3
coding: utf-8
Copyright (c) 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 2680, 2681, 2682, 2683, 2684, 2685, 2686, 2687, 2688, 2689, 2690, 2691, 2692, 2693, 2694, 2695, 2696, 2697, 2698, 2699, 2700, 2701, 2702, 2703, 2704, 2705, 2706, 2707, 2708, 2709, 2710, 2711, 2712, 2713, 2714, 2715, 2716, 2717, 2718, 2719, 2720, 2721, 2722, 2723, 2724, 2725, 2726, 2727, 2728, 2729, 2730, 2731, 2732, 2733, 2734, 2735, 2736, 2737, 2738, 2739, 2740, 2741, 2742, 2743, 2744, 2745, 2746, 2747, 2748, 2749, 2750, 2751, 2752, 2753, 2754, 2755, 2756, 2757, 2758, 2759, 2760, 2761, 2762, 2763, 2764, 2765, 2766, 2767, 2768, 2769, 2770, 2771, 2772, 2773, 2774, 2775, 2776, 2777, 2778, 2779, 2780, 2781, 2782, 2783, 2784, 2785, 2786, 2787, 2788, 2789, 2790, 2791, 2792, 2793, 2794, 2795, 2796, 2797, 2798, 2799, 2800, 2801, 2802, 2803, 2804, 2805, 2806, 2807, 2808, 2809, 2810, 2811, 2812, 2813, 2814, 2815, 2816, 2817, 2818, 2819, 2820, 2821, 2822, 2823, 2824, 2825, 2826, 2827, 2828, 2829, 2830, 2831, 2832, 2833, 2834, 2835, 2836, 2837, 2838, 2839, 2840, 2841, 2842, 2843, 2844, 2845, 2846, 2847, 2848, 2849, 2850, 2851, 2852, 2853, 2854, 2855, 2856, 2857, 2858, 2859, 2860, 2861, 2862, 2863, 2864, 2865, 2866, 2867, 2868, 2869, 2870, 2871, 2872, 2873, 2874, 2875, 2876, 2877, 2878, 2879, 2880, 2881, 2882, 2883, 2884, 2885, 2886, 2887, 2888, 2889, 2890, 2891, 2892, 2893, 2894, 2895, 2896, 2897, 2898, 2899, 2900, 2901, 2902, 2903, 2904, 2905, 2906, 2907, 2908, 2909, 2910, 2911, 2912, 2913, 2914, 2915, 2916, 2917, 2918, 2919, 2920, 2921, 2922, 2923, 2924, 2925, 2926, 2927, 2928, 2929, 2930, 2931, 2932, 2933, 2934, 2935, 2936, 2937, 2938, 2939, 2940, 2941, 2942, 2943, 2944, 2945, 2946, 2947, 2948, 2949, 2950, 2951, 2952, 2953, 2954, 2955, 2956, 2957, 2958, 2959, 2960, 2961, 2962, 2963, 2964, 2965, 2966, 2967, 2968, 2969, 2970, 2971, 2972, 2973, 2974, 2975, 2976, 2977, 2978, 2979, 2980, 2981, 2982, 2983, 2984, 2985, 2986, 2987, 2988, 2989, 2990, 2991, 2992, 2993, 2994, 2995, 2996, 2997, 2998, 2999, 3000, 3001, 3002, 3003, 3004, 3005, 3006, 3007, 3008, 3009, 3010, 3011, 3012, 3013, 3014, 3015, 3016, 3017, 3018, 3019, 3020, 3021, 3022, 3023, 3024, 3025, 3026, 3027, 3028, 3029, 3030, 3031, 3032, 3033, 3034, 3035, 3036, 3037, 3038, 3039, 3040, 3041, 3042, 3043, 3044, 3045, 3046, 3047, 3048, 3049, 3050, 3051, 3052, 3053, 3054, 3055, 3056, 3057, 3058, 3059, 3060, 3061, 3062, 3063, 3064, 3065, 3066, 3067, 3068, 3069, 3070, 3071, 3072, 3073, 3074, 3075, 3076, 3077, 3078, 3079, 3080, 3081, 3082, 3083, 3084, 3085, 3086, 3087, 3088, 3089, 3090, 3091, 3092, 3093, 3094, 3095, 3096, 3097, 3098, 3099, 3100, 3101, 3102, 3103, 3104, 3105, 3106, 3107, 3108, 3109, 3110, 3111, 3112, 3113, 3114, 3115, 3116, 3117, 3118, 3119, 3120, 3121, 3122, 3123, 3124, 3125, 3126, 3127, 3128, 3129, 3130, 3131, 3132, 3133, 3134, 3135, 3136, 3137, 3138, 3139, 3140, 3141, 3142, 3143, 3144, 3145, 3146, 3147, 3148, 3149, 3150, 3151, 3152, 3153, 3154, 3155, 3156, 3157, 3158, 3159, 3160, 3161, 3162, 3163, 3164, 3165, 3166, 3167, 3168, 3169, 3170, 3171, 3172, 3173, 3174, 3175, 3176, 3177, 3178, 3179, 3180, 3181, 3182, 3183, 3184, 3185, 3186, 3187, 3188, 3189, 3190, 3191, 3192, 3193, 3194, 3195, 3196, 3197, 3198, 3199, 3200, 3201, 3202, 3203, 3204, 3205, 3206, 3207, 3208, 3209, 3210, 3211, 3212, 3213, 3214, 3215, 3216, 3217, 3218, 3219, 3220, 3221, 3222, 3223, 3224, 3225, 3226, 3227, 3228, 3229, 3230, 3231, 3232, 3233, 3234, 3235, 3236, 3237, 3238, 3239, 3240, 3241, 3242, 3243, 3244, 3245, 3246, 3247, 3248, 3249, 3250, 3251, 3252, 3253, 3254, 3255, 3256, 3257, 3258, 3259, 3260, 3261, 3262, 3263, 3264, 3265, 3266, 3267, 3268, 3269, 3270, 3271, 3272, 3273, 3274, 3275, 3276, 3277, 3278, 3279, 3280, 3281, 3282, 3283, 3284, 3285, 3286, 3287, 3288, 3289, 3290, 3291, 3292, 3293, 3294, 3295, 3296, 3297, 3298, 3299, 3300, 3301, 3302, 3303, 3304, 3305, 3306, 3307, 3308, 3309, 3310, 3311, 3312, 3313, 3314, 3315, 3316, 3317, 3318, 3319, 3320, 3321, 3322, 3323, 3324, 3325, 3326, 3327, 3328, 3329, 3330, 3331, 3332, 3333, 3334, 3335, 3336, 3337, 3338, 3339, 3340, 3341, 3342, 3343, 3344, 3345, 3346, 3347, 3348, 3349, 3350, 3351, 3352, 3353, 3354, 3355, 3356, 3357, 3358, 3359, 3360, 3361, 3362, 3363, 3364, 3365, 3366, 3367, 3368, 3369, 3370, 3371, 3372, 3373, 3374, 3375, 3376, 3377, 3378, 3379, 3380, 3381, 3382, 3383, 3384, 3385, 3386, 3387, 3388, 3389, 3390, 3391, 3392, 3393, 3394, 3395, 3396, 3397, 3398, 3399, 3400, 3401, 3402, 3403, 3404, 3405, 3406, 3407, 3408, 3409, 3410, 3411, 3412, 3413, 3414, 3415, 3416, 3417, 3418, 3419, 3420, 3421, 3422, 3423, 3424, 3425, 3426, 3427, 3428, 3429, 3430, 3431, 3432, 3433, 3434, 3435, 3436, 3437, 3438, 3439, 3440, 3441, 3442, 3443, 3444, 3445, 3446, 3447, 3448, 3449, 3450, 3451, 3452, 3453, 3454, 3455, 3456, 3457, 3458, 3459, 3460, 3461, 3462, 3463, 3464, 3465, 3466, 3467, 3468, 3469, 3470, 3471, 3472, 3473, 3474, 3475, 3476, 3477, 3478, 3479, 3480, 3481, 3482, 3483, 3484, 3485, 3486, 3487, 3488, 3489, 3490, 3491, 3492, 3493, 3494, 3495, 3496, 3497, 3498, 3499, 3500, 3501, 3502, 3503, 3504, 3505, 3506, 3507, 3508, 3509, 3510, 3511, 3512, 3513, 3514, 3515, 3516, 3517, 3518, 3519, 3520, 3521, 3522, 3523, 3524, 3525, 3526, 3527, 3528, 3529, 3530, 3531, 3532, 3533, 3534, 3535, 3536, 3537, 3538, 3539, 3540, 3541, 3542, 3543, 3544, 3545, 3546, 3547, 3548, 3549, 3550, 3551, 3552, 3553, 3554, 3555, 3556, 3557, 3558, 3559, 3560, 3561, 3562, 3563, 3564, 3565, 3566, 3567, 3568, 3569, 3570, 3571, 3572, 3573, 3574, 3575, 3576, 3577, 3578, 3579, 3580, 3581, 3582, 3583, 3584, 3585, 3586, 3587, 3588, 3589, 3590, 3591, 3592, 3593, 3594, 3595, 3596, 3597, 3598, 3599, 3600, 3601, 3602, 3603, 3604, 3605, 3606, 3607, 3608, 3609, 3610, 3611, 3612, 3613, 3614, 3615, 3616, 3617, 3618, 3619, 3620, 3621, 3622, 3623, 3624, 3625, 3626, 3627, 3628, 3629, 3630, 3631, 3632, 3633, 3634, 3635, 3636, 3637, 3638, 3639, 3640, 3641, 3642, 3643, 3644, 3645, 3646, 3647, 3648, 3649, 3650, 3651, 3652, 3653, 3654, 3655, 3656, 3657, 3658, 3659, 3660, 3661, 3662, 3663, 3664, 3665, 3666, 3667, 3668, 3669, 3670, 3671, 3672, 3673, 3674, 3675, 3676, 3677, 3678, 3679, 3680, 3681, 3682, 3683, 3684, 3685, 3686, 3687, 3688, 3689, 3690, 3691, 3692, 3693, 3694, 3695, 3696, 3697, 3698, 3699, 3700, 3701, 3702, 3703, 3704, 3705, 3706, 3707, 3708, 3709, 3710, 3711, 3712, 3713, 3714, 3715, 3716, 3717, 3718, 3719, 3720, 3721, 3722, 3723, 3724, 3725, 3726, 3727, 3728, 3729, 3730, 3731, 3732, 3733, 3734, 3735, 3736, 3737, 3738, 3739, 3740, 3741, 3742, 3743, 3744, 3745, 3746, 3747, 3748, 3749, 3750, 3751, 3752, 3753, 3754, 3755, 3756, 3757, 3758, 3759, 3760, 3761, 3762, 3763, 3764, 3765, 3766, 3767, 3768, 3769, 3770, 3771, 3772, 3773, 3774, 3775, 3776, 3777, 3778, 3779, 3780, 3781, 3782, 3783, 3784, 3785, 3786, 3787, 3788, 3789, 3790, 3791, 3792, 3793, 3794, 3795, 3796, 3797, 3798, 3799, 3800, 3801, 3802, 3803, 3804, 3805, 3806, 3807, 3808, 3809, 3810, 3811, 3812, 3813, 3814, 3815, 3816, 3817, 3818, 3819, 3820, 3821, 3822, 3823, 3824, 3825, 3826, 3827, 3828, 3829, 3830, 3831, 3832, 3833, 3834, 3835, 3836, 3837, 3838, 3839, 3840, 3841, 3842, 3843, 3844, 3845, 3846, 3847, 3848, 3849, 3850, 3851, 3852, 3853, 3854, 3855, 3856, 3857, 3858, 3859, 3860, 3861, 3862, 3863, 3864, 3865, 3866, 3867, 3868, 3869, 3870, 3871, 3872, 3873, 3874, 3875, 3876, 3877, 3878, 3879, 3880, 3881, 3882, 3883, 3884, 3885, 3886, 3887, 3888, 3889, 3890, 3891, 3892, 3893, 3894, 3895, 3896, 3897, 3898, 3899, 3900, 3901, 3902, 3903, 3904, 3905, 3906, 3907, 3908, 3909, 3910, 3911, 3912, 3913, 3914, 3915, 3916, 3917, 3918, 3919, 3920, 3921, 3922, 3923, 3924, 3925, 3926, 39

arguments/):

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
74. | # construct the argument parser and parse the arguments
75. | ap = argparse.ArgumentParser()
76. | ap.add_argument("-f", "--face", type=str,
77. |     default="face_detector",
78. |     help="path to face detector model directory")
79. | ap.add_argument("-m", "--model", type=str,
80. |     default="mask_detector.model",
81. |     help="path to trained face mask detector model")
82. | ap.add_argument("-c", "--confidence", type=float, default=0.5,
83. |     help="minimum probability to filter weak detections")
84. | args = vars(ap.parse_args())
```

Our command line arguments include:

- `--face` : The path to the face detector directory
- `--model` : The path to our trained face mask classifier
- `--confidence` : The minimum probability threshold to filter weak face detections

With our imports, convenience function, and command line

`args` ready to go, we just have a few initializations to handle before we loop over frames:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
86. | # load our serialized face detector model from disk
87. | print("[INFO] loading face detector model...")
88. | prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
89. | weightsPath = os.path.sep.join([args["face"],
90. |     "res10_300x300_ssd_iter_140000.caffemodel"])
91. | faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
92. |
93. | # load the face mask detector model from disk
94. | print("[INFO] loading face mask detector model...")
95. | maskNet = load_model(args["model"])
96. |
97. | # initialize the video stream and allow the camera sensor to warm up
98. | print("[INFO] starting video stream...")
99. | vs = VideoStream(src=0).start()
100. | time.sleep(2.0)
```

Here we have initialized our:

- Face detector

- COVID-19 face mask detector
- Webcam video stream

Let's proceed to loop over frames in the stream:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
102. | # loop over the frames from the video stream
103. | while True:
104. |     # grab the frame from the threaded video stream and resize it
105. |     # to have a maximum width of 400 pixels
106. |     frame = vs.read()
107. |     frame = imutils.resize(frame, width=400)
108. |
109. |     # detect faces in the frame and determine if they are wearing a
110. |     # face mask or not
111. |     (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
```

We begin looping over frames on **Line 103**. Inside, we grab a

`frame` from the stream and `resize` it (**Lines 106 and 107**).

From there, we put our convenience utility to use; **Line 111** detects and predicts whether people are wearing their masks or not.

Let's post-process (i.e., annotate) the COVID-19 face mask detection results:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
113. | # loop over the detected face locations and their corresponding
114. | # locations
115. | for (box, pred) in zip(locs, preds):
116. |     # unpack the bounding box and predictions
117. |     (startX, startY, endX, endY) = box
118. |     (mask, withoutMask) = pred
119. |
120. |     # determine the class label and color we'll use to draw
121. |     # the bounding box and text
122. |     label = "Mask" if mask > withoutMask else "No Mask"
123. |     color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
124. |
125. |     # include the probability in the label
126. |     label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
127. |
128. |
129. |     # display the label and bounding box rectangle on the output
130. |     # frame
131. |     cv2.putText(frame, label, (startX, startY - 10),
132. |                 cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
133. |     cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
```

Inside our loop over the prediction results (beginning on **Line 115**), we:

- Unpack a face bounding box and mask/not mask prediction (**Lines 117 and 118**)
- Determine the `label` and `color` (**Lines 122-126**)
- Annotate the `label` and face bounding box (**Lines 130-132**)

Finally, we display the results and perform cleanup:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
134. |     # show the output frame
135. |     cv2.imshow("Frame", frame)
136. |     key = cv2.waitKey(1) & 0xFF
137. |
138. |     # if the `q` key was pressed, break from the loop
139. |     if key == ord("q"):
140. |         break
141. |
142. |     # do a bit of cleanup
143. |     cv2.destroyAllWindows()
144. |     vs.stop()
```

After the

`frame` is displayed, we capture `key` presses. If the user presses `q` (quit), we `break` out of the loop and perform housekeeping.

Great job implementing your real-time face mask detector with Python, OpenCV, and deep learning with TensorFlow/Keras!

Detecting COVID-19 face masks with OpenCV in real-time

To see our real-time COVID-19 face mask detector in action, make sure you use the **“Downloads”** section of this tutorial to download the source code and pre-trained face mask detector model.

You can then launch the mask detector in real-time video streams using the following command:

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
1. | $ python detect_mask_video.py
2. | [INFO] loading face detector model...
3. | [INFO] loading face mask detector model...
4. | [INFO] starting video stream...
```

Here, you can see that our face mask detector is capable of running in real-time (and is *correct* in its predictions as well).

Suggestions for improvement

As you can see from the results sections above, our face mask detector is working quite well despite:

- 1 Having limited training data
- 2 The `with_mask` class being artificially generated (see the “*How was our face mask dataset created?*” section above).

To improve our face mask detection model further, **you should gather *actual images* (rather than artificially generated images) of people wearing masks.**

While our artificial dataset worked well in this case, there’s no substitute for the real thing.

Secondly, you should also gather images of faces that may “confuse” our classifier into thinking the person is wearing a mask when in fact they are not — potential examples include shirts wrapped around faces, bandana over the mouth, etc.

All of these are examples of something that *could* be confused as a face mask by our face mask detector.

Finally, you should consider training a dedicated two-class *object detector* rather than a simple image classifier.

Our current method of detecting whether a person is wearing a mask or not is a two-step process:

- 1 **Step #1:** Perform face detection
- 2 **Step #2:** Apply our face mask detector to each face

The *problem* with this approach is that a face mask, by definition, obscures part of the face. If enough of the face is obscured, the face cannot be detected, *and therefore, the face mask detector will not be applied.*

To circumvent that issue, you should train a two-class object detector that consists of a

`with_mask` class and `without_mask` class.

Combining an object detector with a dedicated

`with_mask` class will allow improvement of the model in two respects.

First, the object detector will be able to naturally detect people wearing masks that otherwise would have been impossible for the face detector to detect due to too much of the face being obscured.

Secondly, this approach reduces our computer vision pipeline to a single step — rather than applying face detection and *then* our face mask detector model, all we need to do is apply the object detector to give us bounding boxes for people both

with_mask and without_mask in a single forward pass of the network.

Not only is such a method more computationally efficient, it's also more “elegant” and end-to-end.

What's next?

[\(https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/\)](https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)

Figure 14: If you want to learn to train your own deep learning models on your own datasets, pick up a copy of *[Deep Learning for Computer Vision with Python](https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)* ([\(https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/\)](https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)), and begin studying. My team and I will be there every step of the way, ensuring you can execute example code and get your questions answered.

Inside today's tutorial, we covered training a face mask detector. If you're inspired to create your own deep learning projects, I would recommend reading my book, *[Deep Learning for Computer Vision with Python](https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)* ([\(https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/\)](https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)).

I crafted my book so that it perfectly balances theory with implementation, ensuring you properly master:

- **Deep learning fundamentals and theory without unnecessary mathematical fluff.** I present the basic equations and back them up with code walkthroughs that you can implement and easily understand. You don't need a degree in advanced mathematics to understand this book
- **How to implement your own custom neural network architectures.** Not only will you learn how to implement state-of-the-art architectures, including ResNet, SqueezeNet, etc., but you'll *also* learn how to create your own custom CNNs
- **How to train CNNs on your own datasets.** Most deep learning tutorials don't teach you how to work with your own custom datasets. Mine do. You'll be training CNNs on your own datasets in no time
- **Object detection (Faster R-CNNs, Single Shot Detectors, and RetinaNet) and instance segmentation (Mask R-CNN).** Use these chapters to create your own custom object detectors and segmentation networks

You'll also find answers and proven code recipes to:

- **Create and prepare your own custom image datasets** for image classification, object detection, and segmentation
- **Hands-on tutorials (with lots of code)** that not only show you the *algorithms* behind deep learning for computer vision but their *implementations* as well
- Put my **tips, suggestions, and best practices** into action, ensuring you maximize the accuracy of your models

My readers enjoy my **no-nonsense teaching style** that is guaranteed to help you master deep learning for image understanding and visual recognition.

If you're ready to dive in, **just click here** (<https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/>). And if you aren't convinced yet, just grab my **free** PDF of sample chapters and the entire table of contents by filling the form in the lower right of **this page** (<https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/>).

Grab my **free** sample chapters!
(<https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/>)

Summary

In this tutorial, you learned how to create a COVID-19 face mask detector using OpenCV, Keras/TensorFlow, and Deep Learning.

To create our face mask detector, we trained a two-class model of people *wearing masks* and people *not wearing masks*.

We fine-tuned MobileNetV2 on our *mask/no mask* dataset and obtained a classifier that is **~99% accurate**.

We then took this face mask classifier and applied it to both *images* and *real-time video streams* by:

- 2 Extracting each individual face
- 3 Applying our face mask classifier

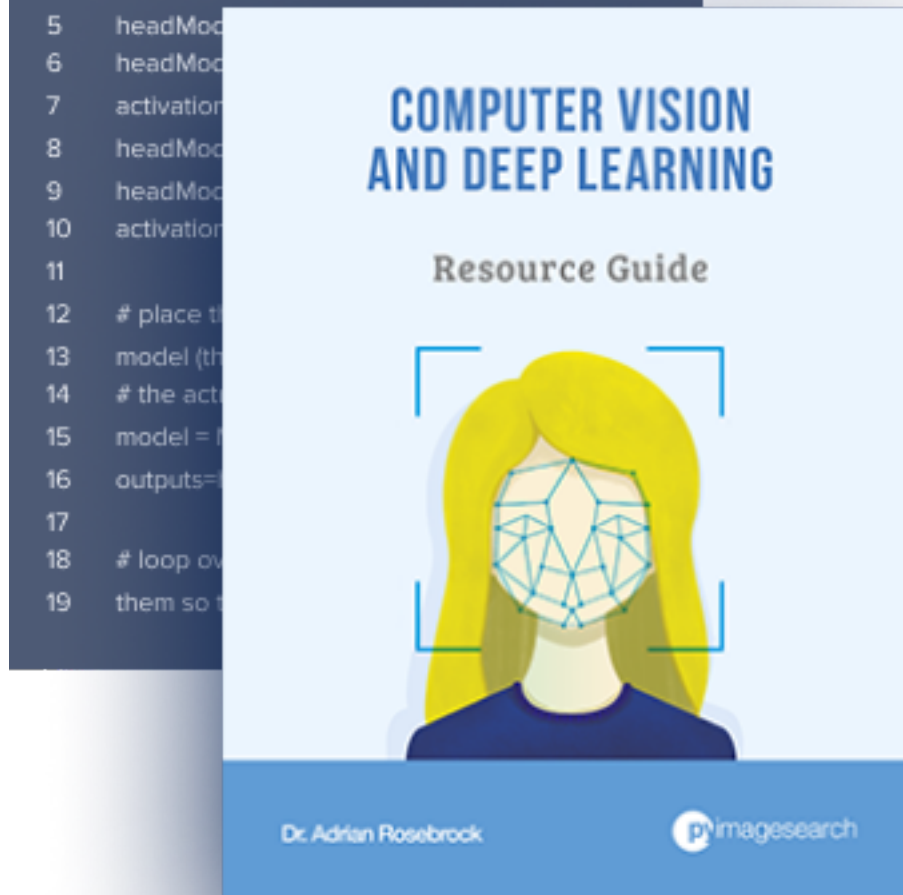
Our face mask detector is accurate, and since we used the MobileNetV2 architecture, it's also *computationally efficient*, making it easier to deploy the model to embedded systems (Raspberry Pi, Google Coral, Jetson, Nano, etc.).

I hope you enjoyed this tutorial!

To download the source code to this post (including the pre-trained COVID-19 face mask detector model), *just enter your email address in the form below!*



```
1 # construct the head of the model that will be
2 placed on top of the
3 # the base model
4 headModel = baseModel.output
```

Download the Source Code and FREE 17-page Resource Guide

Enter your email address below to get a .zip of the code and a **FREE 17-page Resource Guide on Computer Vision, OpenCV, and Deep Learning**. Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL!





About the Author

Hi there, I'm Adrian Rosebrock, PhD. All too often I see developers, students, and researchers wasting their time, studying the wrong things, and generally struggling to get started with Computer Vision, Deep Learning, and OpenCV. I created this website to show you what I believe is the best possible way to get your start.

Previous Article:

Fine-tuning ResNet with Keras, TensorFlow, and Deep Learning

(<https://www.pyimagesearch.com/2020/04/27/fine-tuning-resnet-with-keras-tensorflow-and-deep-learning/>)

Next Article:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

(<https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/>)

Before you leave a comment...

Hey, Adrian here, author of the PyImageSearch blog. I'd love to hear from you; however, I am super busy with some special projects right now and do not have the time to moderate and personally respond to blog post comments.

I'd be happy to help you with your question or project, but **I have to politely ask you to purchase one of my books or courses first.**

(<https://www.pyimagesearch.com/books-and-courses/>)

Why bother becoming a PyImageSearch customer?

- You'll receive a **great education** through my premium content
- You'll receive **priority support**
- You'll receive a **guaranteed response** from myself and my team
- You'll be able to **confidently and successfully** apply Computer Vision, Deep Learning, and OpenCV to your projects

Click here to see my full catalog of books and courses.

(<https://www.pyimagesearch.com/books-and-courses/>) Take a look and I hope to see you on the other side!

Similar articles

Adding a web interface to our image search engine with Flask

December 8, 2014

(<https://www.pyimagesearch.com/2014/12/08/adding-web-interface-image-search-engine-flask/>)



DEEP LEARNING

DL4CV

Configuring Ubuntu for deep learning with Python

September 25, 2017

(<https://www.pyimagesearch.com/2017/09/25/configuring-ubuntu-for-deep-learning-with-python/>)



MACHINE LEARNING

TUTORIALS

(Faster) Non-Maximum Suppression in Python

February 16, 2015

(<https://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/>)



You can learn Computer Vision, Deep Learning, and OpenCV.

Get your FREE 17 page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF. Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL.

Topics

[Deep Learning \(https://www.pyimagesearch.com/category/deep-learning-2/\)](https://www.pyimagesearch.com/category/deep-learning-2/)

[Dlib Library \(https://www.pyimagesearch.com/category/dlib/\)](https://www.pyimagesearch.com/category/dlib/)

[Embedded/IoT and Computer Vision \(https://www.pyimagesearch.com/category/embedded/\)](https://www.pyimagesearch.com/category/embedded/)

[Face Applications \(https://www.pyimagesearch.com/category/faces/\)](https://www.pyimagesearch.com/category/faces/)

[Image Processing \(https://www.pyimagesearch.com/category/image-processing/\)](https://www.pyimagesearch.com/category/image-processing/)

[Interviews \(https://www.pyimagesearch.com/category/interviews/\)](https://www.pyimagesearch.com/category/interviews/)

[Keras \(https://www.pyimagesearch.com/category/keras/\)](https://www.pyimagesearch.com/category/keras/)

[Machine Learning and Computer Vision \(https://www.pyimagesearch.com/category/machine-learning-2/\)](https://www.pyimagesearch.com/category/machine-learning-2/)

[Medical Computer Vision \(https://www.pyimagesearch.com/category/medical/\)](https://www.pyimagesearch.com/category/medical/)

[Optical Character Recognition \(OCR\) \(https://www.pyimagesearch.com/category/optical-character-recognition-ocr/\)](https://www.pyimagesearch.com/category/optical-character-recognition-ocr/)

[Object Detection \(https://www.pyimagesearch.com/category/object-detection/\)](https://www.pyimagesearch.com/category/object-detection/)

[Object Tracking \(https://www.pyimagesearch.com/category/object-tracking/\)](https://www.pyimagesearch.com/category/object-tracking/)

[OpenCV Tutorials \(https://www.pyimagesearch.com/category/opencv/\)](https://www.pyimagesearch.com/category/opencv/)

[Raspberry Pi \(https://www.pyimagesearch.com/category/raspberry-pi/\)](https://www.pyimagesearch.com/category/raspberry-pi/)

Books & Courses

[FREE CV, DL, and OpenCV Crash Course \(https://www.pyimagesearch.com/free-opencv-computer-vision-deep-learning-crash-course/\)](https://www.pyimagesearch.com/free-opencv-computer-vision-deep-learning-crash-course/)

[Practical Python and OpenCV \(https://www.pyimagesearch.com/practical-python-opencv/\)](https://www.pyimagesearch.com/practical-python-opencv/)

[Deep Learning for Computer Vision with Python \(https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/\)](https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)

[PyImageSearch Gurus Course \(https://www.pyimagesearch.com/pyimagesearch-gurus/\)](https://www.pyimagesearch.com/pyimagesearch-gurus/)

[Raspberry Pi for Computer Vision \(https://www.pyimagesearch.com/raspberry-pi-for-computer-vision/\)](https://www.pyimagesearch.com/raspberry-pi-for-computer-vision/)

PyImageSearch

[Get Started \(https://www.pyimagesearch.com/start-here/\)](https://www.pyimagesearch.com/start-here/)

[OpenCV Install Guides \(https://www.pyimagesearch.com/opencv-tutorials-resources-guides/\)](https://www.pyimagesearch.com/opencv-tutorials-resources-guides/)

[About \(https://www.pyimagesearch.com/about/\)](https://www.pyimagesearch.com/about/)

[FAQ \(https://www.pyimagesearch.com/faqs/\)](https://www.pyimagesearch.com/faqs/)

[Blog \(https://www.pyimagesearch.com/topics/\)](https://www.pyimagesearch.com/topics/)

[Contact \(https://www.pyimagesearch.com/contact/\)](https://www.pyimagesearch.com/contact/)

[Privacy Policy \(https://www.pyimagesearch.com/privacy-policy/\)](https://www.pyimagesearch.com/privacy-policy/)



Just a suggestion...

If you're interested in Deep Learning, why not take a look at ***Deep Learning for Computer Vision with Python?***

This book is a deep dive into the world of computer vision and deep learning. Inside you'll find **super practical walkthroughs, hands-on tutorials** (with lots of code), and **a no-nonsense teaching style** that is guaranteed to help you master deep learning and computer vision.

[Tell me more!](#)



Just a suggestion...

[Click to learn more](#)