An Efficient Deep Learning Model to Detect The Presence of COVID-19 in Patients From Their X-Ray Images

SAYAK GIRI, Department of Statistics, St. Xavier's College (Autonomous), Kolkata, West Bengal, India, sayakgl998@gmail.com

RISHAV GIRI, Department of Computer Science, Heritage Institute of Technology (Autonomous), Kolkata, West Bengal, India, rishavgiri.work@gmail.com

CCS Concepts: Deep Learning → Convolutional Neural Networks

Additional keywords and phrases: Covid-19, X-ray imaging, VGGNet model, Binary classification

Abstract:

Coronavirus disease (COVID-19) is an infectious disease caused by a newly discovered coronavirus.

Most people infected with the COVID-19 virus will experience mild to moderate respiratory illness and recover without requiring special treatment. Older people, and those with underlying medical problems like cardiovascular disease, diabetes, chronic respiratory disease, and cancer are more likely to develop serious illness.

The best way to prevent and slow down transmission is be well informed about the COVID-19 virus, the disease it causes and how it spreads. Protect yourself and others from infection by washing your hands or using an alcohol based rub frequently and not touching your face.

The COVID-19 virus spreads primarily through droplets of saliva or discharge from the nose when an infected person coughs or sneezes, so it's important that we also practice respiratory etiquette (for example, by coughing into a flexed elbow).

At this time, there are no specific vaccines or treatments for COVID-19. However, there are many ongoing clinical trials evaluating potential treatments.

Artificial intelligence applied to the medical domain can have very real consequences.

COVID-19 tests are currently hard to come by — there are simply not enough of them and they cannot be manufactured fast enough, which is causing panic.

Given that there are limited COVID-19 testing kits, we need to rely on other diagnosis measures.

In this project, we are going to explore X-ray images as doctors frequently use X-rays and CT scans to diagnose pneumonia, lung inflammation, abscesses, and/or enlarged lymph nodes and try to detect positive COVID-19 cases from the given X-ray dataset.

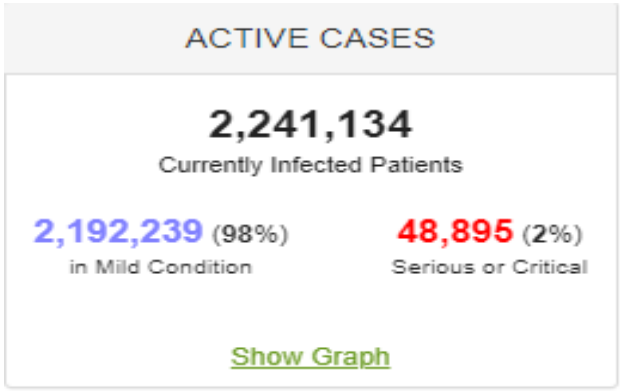
Since COVID-19 attacks the epithelial cells that line our respiratory tract, we can use X-rays to analyze the health of a patient's lungs.

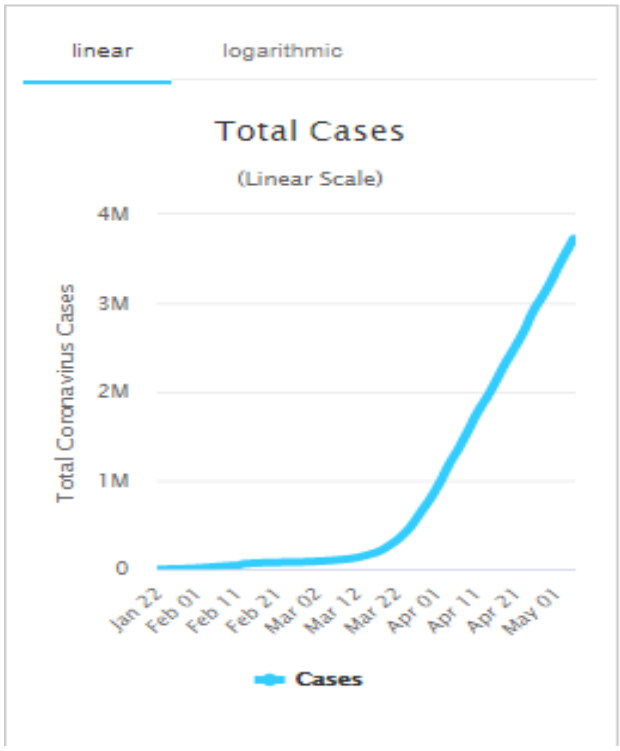
And given that nearly all hospitals have X-ray imaging machines, it could be possible to use X-rays to test for COVID-19 without the dedicated test kits.

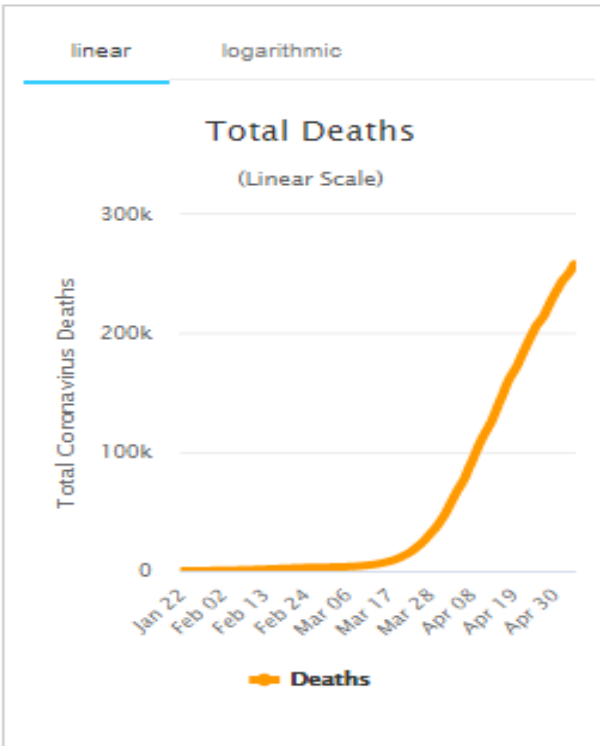
A drawback is that X-ray analysis requires a radiology expert and takes significant time — which is precious when people are sick around the world. Therefore developing an automated analysis system is required to save medical professionals valuable time.

Statistics:

The coronavirus Covid-19 is affecting 212 countries and territories around the world and 2 international conveyances. As of 6th May, 2020, the statistics are as shown below:







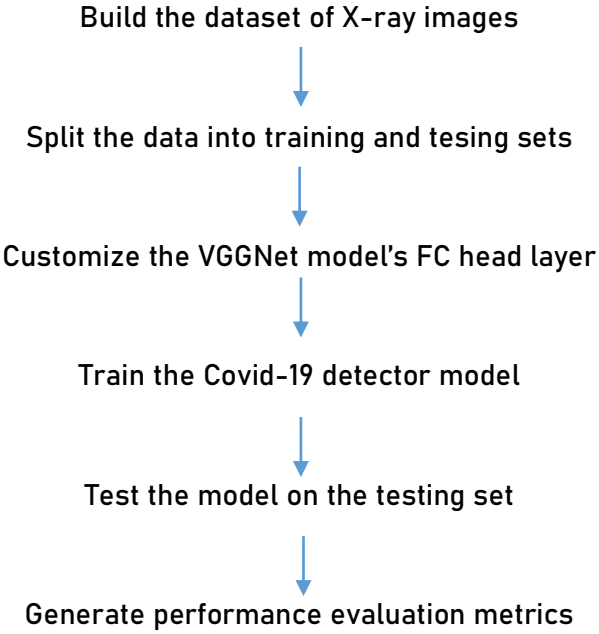
Symptom table for Covid-19 patients:

GENDER	All patients N = 121	Early* (0-2 days) N = 36	Intermediate (3-5 days) N=33	Late (6-12 days) N=25
Men	61 (50)	16 (44)	16 (48)	11 (44)
Women	60 (50)	20 (56)	17 (52)	14 (56)
AGE (YEARS)				
Mean	45	45	47	50
Range	18 - 80	19 – 79	22 – 75	22 – 80
Standard Deviation	15.6	15.1	14.8	16.3
EXPOSURE HISTORY				
Recent Travel to Wuhan	92 (76)	33 (92)	28 (85)	21 (84)
Exposure to Infected Patient	7 (6)	2 (6)	2 (6)	2 (8)
Unknown Exposure	22 (18)	1 (3)	3 (9)	2 (8)
SYMPTOMS				
Fever	74 (61)	22 (61)	26 (79)	23 (92)
Cough	58 (48)	19 (53)	21 (64)	15 (60)
Sputum Production	20 (17)	7 (19)	5 (15)	6 (24)

Algorithm/Methodology and Flowchart:

I. Pre-process the given X ray images and convert them to RGB channel ordering to make them ready for our CNN.
II. One-hot encode the image labels and create the training/testing split at 80%/20%.
III. Initialize the data augmentation generator object.
IV. Initialize the VGGNet model and set it up for fine tuning.
V. Instantiate the VGG16 network with weights pre-trained on ImageNet leaving off the FC layer head. Construct a new fully-connected layer head comprising POOL, FC, SOFTMAX layers and append it on top of VGG16. Freeze the CONV layer such that only the FC layer head layer will be trained.
VI. Compile the network using the Learning Rate Decay and Adam optimizer. Use binary cross entropy loss rather than categorical cross entropy because it is a class 2 problem.
VII. To start the CNN training process, make a call to Keras' fit_generator method, while passing in our chest X-ray data via the data augmentation object.
VIII. For evaluation, first make predictions on the testing set and grab the prediction indices. Generate and print out a classification report using scikit-learn's helper utility.
IX. Generate a confusion matrix and use it to derive the accuracy, sensitivity, and specificity.

X. Plot the training accuracy/loss history for inspection, outputting the plot to an image file.
XI. Plot our training accuracy/loss history for inspection, outputting the plot to an image file.
XII. Finally serialize the keras COVID-19 classifier model to disk.



Formulae:

1. A feed-forward neural network can be thought of as the composition of number of functions
$$f(x)=f_L(\dots f_2(f_1(x;w_1);w_2)\dots),w_L).f(x)=f_L(\dots f_2(f_1(x;w_1);w_2)\dots),w_L).$$

   Each function f_lf_l takes as input a datum x_lx_l and a parameter vector w_lw_l and produces as output a datum $x_{l+1}x_{l+1}$. While the type and sequence of functions is usually handcrafted, the parameters $w=(w_1,\dots,w_L)w=(w_1,\dots,w_L)$ are *learned from data* in order to solve a target problem, for example classifying images or sounds.

2. The simplest non-linearity is obtained by following a linear filter by a *non-linear activation function*, applied identically to each component (i.e. point-wise) of a feature map. The simplest such function is the *Rectified Linear Unit (ReLU)*
$$y_{ijk}=\max\{0,x_{ijk}\}.$$

3. Another important CNN building block is channel-wise normalisation. This operator normalises the vector of feature channels at each spatial location in the input map xx. The form of the normalisation operator is actually rather curious:
$$y_{ijk'}=x_{ijk}(\kappa+\alpha\sum k\in G(k')x^2_{ijk})\beta y_{ijk'}=x_{ijk}(\kappa+\alpha\sum k\in G(k')x_{ijk}^2)\beta$$

where $G(k)=[k-\lfloor\rho^2\rfloor,k+\lceil\rho^2\rceil]\cap 1,2,\ldots,KG(k)=[k-\lfloor\rho^2\rfloor,k+\lceil\rho^2\rceil]\cap 1,2,\ldots,K$ is a group of $\rho\rho$ consecutive feature channels in the input map.

4. The parameters of a CNN $w=(w_1,\ldots w_L)w=(w_1,\ldots w_L)$ should be learned in such a manner that the overall CNN function $z=f(x;w)z=f(x;w)$ achieves a desired goal. In some cases, the goal is to model the distribution of the data, which leads to a *generative objective*. Here, however, we will use ff as a *regressor* and obtain it by minimising a *discriminative objective*. In simple terms, we are given:

- examples of the desired input-output relations $(x1,z1),\ldots,(xn,zn)(x1,z1),\ldots,(xn,zn)$ where xixi are input data and zizi corresponding output values;
- and a loss $\ell(z,z\hat{})\ell(z,z\hat{})$ that expresses the penalty for predicting $z\hat{}z\hat{}$ instead of zz.

We use those to write the empirical loss of the CNN ff by averaging over the examples:

$$L(w)=1n\sum i=1n\ell(z_i,f(x_i;w))$$

5. Training CNNs is normally done using a gradient-based optimization method. The CNN ff is the composition of LL layers f_1f_1 each with parameters w_1w_1, which in the simplest case of a chain looks like:

$$x_0\longrightarrow f_1\uparrow w_1\longrightarrow x_1\longrightarrow f_2\uparrow w_2\longrightarrow x_2\longrightarrow\cdots\longrightarrow x_{L-1}\longrightarrow f_L\uparrow w_L\longrightarrow x_Lx_0\longrightarrow f_1\uparrow w_1\longrightarrow x_1\longrightarrow f_2\uparrow w-$$
$$\longrightarrow x_2\longrightarrow\cdots\longrightarrow x_{L-1}\longrightarrow f_L\uparrow w_L\longrightarrow x_L$$

During learning, the last layer of the network is the *loss function* that should be minimized. Hence, the output $x_L=x_Lx_L=x_L$ of the network is a **scalar** quantity (a single number).

6. The gradient is easily computed using using the **chain rule**. If *all* network variables and parameters are scalar, this is given by:

$$\partial f\partial w_1(x_0;w_1,\ldots,w_L)=\partial fL\partial x_{L-1}(x_{L-1};w_L)\times\cdots\times\partial f1+1\partial x_1(x_i;w_{1+1})\times\partial f1\partial w1(x_{1-1};w_1)$$

7. The CNN computes as per our definition and optimization of the following objective function:

$$E(w,b)=\lambda 2\|w\|2+1|P|\sum(u,v)\in P\max\{0,1-f(x;w,b)(u,v)\}+1|N|\sum(u,v)\in N\max\{0,f(x;w,b)(u,v)\}.$$

We train the CNN by minimising the objective function with respect to ww and bb. We do so by using an algorithm called *gradient descent with momentum*. Given the current solution $(w_t,b_t)(w_t,b_t)$, this is updated to $(w_{t+1},b_{t+1})(w_{t+1},b_{t+1})$ by following the direction of fastest descent of the objective E(wt,bt)E(wt,bt) as given by the negative gradient $-\nabla E-\nabla E$. However, gradient updates are smoothed by considering a *momentum* term $(\bar{w}t,\bar{\mu}t)(\bar{w}t,\bar{\mu}t)$, yielding the update equations

$$\bar{w}_{t+1}\leftarrow\mu\bar{w}t+\eta\partial E\partial wt,w_{t+1}\leftarrow w_t-\bar{w}t.\bar{w}_{t+1}\leftarrow\mu\bar{w}_{t+\eta}\partial E\partial w_t,w_{t+1}\leftarrow w_t-\bar{w}_t.$$

and similarly for the bias term. Here $\mu\mu$ is the *momentum rate* and $\eta\eta$ the *learning rate*.

Tools:

- Environment: Python 3.0 in Spyder IDE.
- Packages and libraries: keras, tensorflow, numpy, sklearn, matplotlib, ggplot, imutils, cv2
- OS: Windows 10

Results:

After 25 epochs, the following result was obtained.

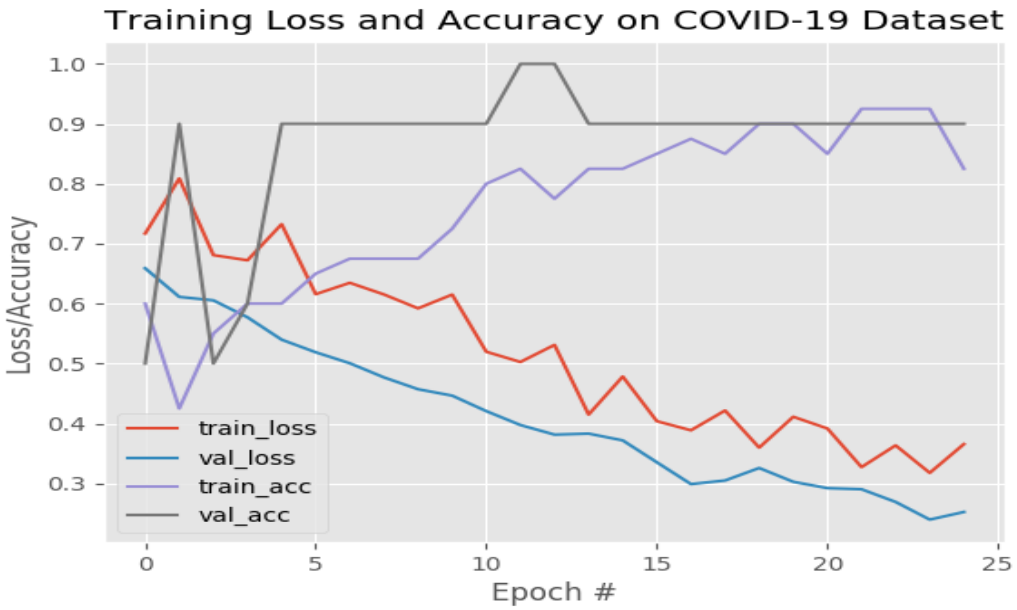
```
Epoch 25/25
5/5 [==============================] - 1s 101ms/step - loss: 0.3655 - accuracy: 0.8250 -
val_loss: 0.2522 - val_accuracy: 0.9000
[INFO] evaluating network...
              precision    recall  f1-score   support

       covid       0.83      1.00      0.91         5
      normal       1.00      0.80      0.89         5

    accuracy                           0.90        10
   macro avg       0.92      0.90      0.90        10
weighted avg       0.92      0.90      0.90        10

[[5 0]
 [1 4]]
acc: 0.9000
sensitivity: 1.0000
specificity: 0.8000
```

As we can see from the above results, COVID-19 detector is obtaining approximately 90% accuracy on our testing dataset based solely on X-ray images — no other data, including geographical location, population density, etc. was used to train this model.

We also obtain 100% sensitivity and 80% specificity implying that:

- Of patients that do have COVID-19 (i.e., true positives), we could accurately identify them as "COVID-19 positive" 100% of the time using our model.
- Of patients that do not have COVID-19 (i.e., true negatives), we could accurately identify them as "COVID-19 negative" only 80% of the time using our model.

As our training history plot shows, our network is not overfitting, despite having very limited training data.



Limitations and Future Improvements:

- Lack of enough reliable data to train our CNN.
- Hospitals are already overwhelmed with the number of COVID-19 cases, and given patients rights and confidentiality, it becomes even harder to assemble quality medical image datasets in a timely fashion.
- In the next 12-18 months we'll have more high quality COVID-19 image datasets; but for the time being, we can only make do with what we have.
- The possibility remains that our model is learning patterns that are not relevant to COVID-19, and instead are just variations between the two data splits (i.e., positive versus negative COVID-19 diagnosis).
- Our future (and better) COVID-19 detectors will be multi-modal.
- Currently we are using only image data (i.e., X-rays) — better automatic COVID-19 detectors should leverage multiple data sources not limited to just images, including patient vitals, population density, geographical location, etc. Image data by itself is typically not sufficient for these types of applications.

References:

[1] Data Modelling & Analysing Coronavirus (COVID19) Spread - https://in.springboard.com/blog/data-modelling-covid/

[2] WHO – Health topics/ Coronavirus

[3] Kaggle – Coronavirus datasets - https://www.kaggle.com/tags/covid19

[4] Github - https://github.com/ieee8023/covid-chestxray-dataset/tree/master/images

[5] Cornell University - Prediction of COVID-19 Disease Progression in India : Under the Effect of National Lockdown - https://arxiv.org/abs/2004.03147?fbclid=IwAR0C415fGKdmbDIJxs0Ln0ibhtNJ20sdg2nfQvPXHnBV9c-brZrYnwe2co0

[6] Ministry of Health and Family Welfare - https://www.mohfw.gov.in/

[7] Multi-layer CNN Features Fusion and Classifier Optimization for Face Recognition - https://dl.acm.org/doi/10.1145/3297156.3297208

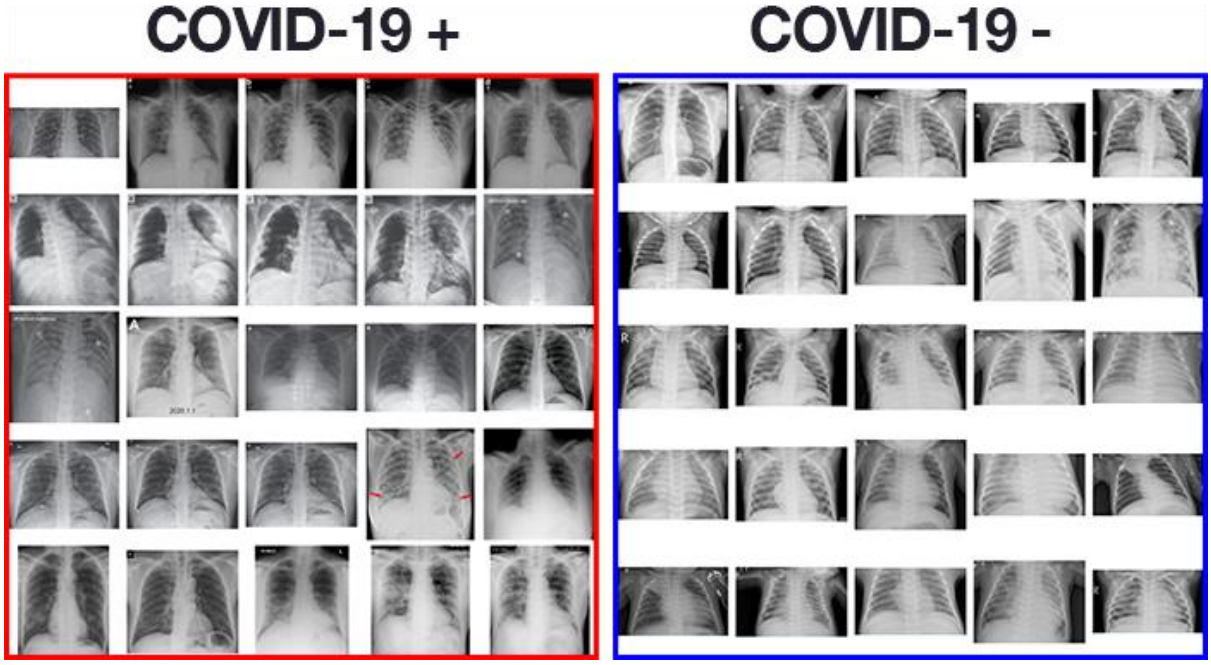
[8] VGG Convolutional Neural Networks Practical (By Andrea Vedaldi and Andrew Zisserman)-

http://www.robots.ox.ac.uk/~vgg/practicals/cnn/

[9] Transmission line corona and X-rays (J.M. Silva_, H.H. Fleischmann , C.H. Shih)- https://ieeexplore.ieee.org/document/1308382

Appendix:

Exploring the dataset:



CoronaVirus (COVID-19) chest X-ray image data. On the left we have positive (i.e., infected) X-ray images, whereas on the right we have negative samples. In total, we have (25+25 =) 50 X-ray images in our dataset.

Training code:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
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.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from imutils import paths
import matplotlib.pyplot as plt
```

```
import numpy as np
import argparse
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
	help="path to input dataset")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
	help="path to output loss/accuracy plot")
ap.add_argument("-m", "--model", type=str, default="covid19.model",
	help="path to output loss/accuracy plot")
args = vars(ap.parse_args())

# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-3
EPOCHS = 25
BS = 8

# 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...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []

# loop over the image paths
for imagePath in imagePaths:
	# extract the class label from the filename
	label = imagePath.split(os.path.sep)[-2]

	# load the image, swap color channels, and resize it to be a fixed
	# 224x224 pixels while ignoring aspect ratio
	image = cv2.imread(imagePath)
	image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
	image = cv2.resize(image, (224, 224))
```

```
                # update the data and labels lists, respectively
                data.append(image)
                labels.append(label)

# convert the data and labels to NumPy arrays while scaling the pixel
# intensities to the range [0, 255]
data = np.array(data) / 255.0
labels = np.array(labels)

# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
                test_size=0.20, stratify=labels, random_state=13)

# initialize the training data augmentation object
trainAug = ImageDataGenerator(
                rotation_range=15,
                fill_mode="nearest")

# load the VGG16 network, ensuring the head FC layer sets are left
# off
baseModel = VGG16(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=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, 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 = 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_generator(
		trainAug.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_))

# compute the confusion matrix and and use it to derive the raw
```

```
# accuracy, sensitivity, and specificity
cm = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])

# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))

# 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 on COVID-19 Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])

# serialize the model to disk
print("[INFO] saving COVID-19 detector model...")
model.save(args["model"], save_format="h5")
```