**Contents**

Introduction 3

Data Collection 4

Training the Model 5 – 9

Predictions 10

Result 11

**Introduction**

The 2019 novel coronavirus (COVID-19) presents several unique features. While the diagnosis is confirmed using polymerase chain reaction (PCR), infected patients with pneumonia may present on chest X-ray and computed tomography (CT) images with a pattern that is only moderately characteristic for the human eye. In late January, a Chinese team published a paper detailing the clinical and paraclinical features of COVID-19. They reported that patients present abnormalities in chest CT images with most having bilateral involvement. Bilateral multiple lobular and subsegmental areas of consolidation constitute the typical findings in chest CT images of intensive care unit (ICU) patients on admission. In comparison, non-ICU patients show bilateral ground-glass opacity and subsegmental areas of consolidation in their chest CT images. In these patients, later chest CT images display bilateral ground-glass opacity with resolved consolidation.

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.

**Data Collection**

The COVID-19 X-ray images we’ll be using was curated by Dr. Joseph Cohen, a postdoctoral fellow at the University of Montreal. Dr. Cohen started collecting X-ray images of COVID-19 cases and publishing them in this GitHub repo : <https://github.com/ieee8023/covid-chestxray-dataset>.

In order to create covid-19 x-ray image dataset, we:

* Parsed metadata.csv file found in the above repo.
* Selected all rows positive for covid-19 (ignoring MERS, SARS, ARDS etc.) with PA views as our normal x-ray image dataset has the same view.

The script for this is found in build\_covid\_dataset.py file.

To collect the images of healthy patients, we used Kaggle’s Chest X-Ray images (Pneumonia) dataset : <https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia>

The script for building the normal dataset is found in sample\_kaggle\_dataset.py file.

In total, we collected 50 images, 25 of both sample types.

**Training the Model**

After collecting the dataset, we started building a Convolutional Neural Network to automatically diagnose Covid-19 using Keras and TensorFlow.

We start by importing the necessary packages

* 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

Next we will parse command line arguments and initialize hyperparameters

* 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())

Our three command line arguments include:

* --dataset

: The path to our input dataset of chest X-ray images.

* --plot

: An optional path to an output training history plot. By default the plot is named

plot.png

 unless otherwise specified via the command line.

* --model

: The optional path to our output COVID-19 model; by default it will be named

covid19.model

.

Then we initiliaze learning rate, number of epochs to train for, and batch size:

* INIT\_LR = 1e-3
* EPOCHS = 25
* BS = 8

Now in our dataset directory, we have two folders covid and normal both containing images for same. We grab all the images in the dataset directory and for each image, we extract the label, preprocess it by converting it to RGB and resizing it to 224x224 pixels so that it is ready for our CNN. Finally, we append these images and labels to data and label lists respectively.

* print("[INFO] loading images...")
* imagePaths = list(paths.list\_images(args["dataset"]))
* data = []
* labels = []
* for imagePath in imagePaths:
* # extract the class label from the filename
* label = imagePath.split(os.path.sep)[-2]
* image = cv2.imread(imagePath)
* image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)
* image = cv2.resize(image, (224, 224))
* data.append(image)
* labels.append(label)

We then scale pixel intensities to the range [0,1] and convert both our data and labels to NumPy array format.

* data = np.array(data) / 255.0
* labels = np.array(labels)

Next we’ll one-hot encode (convert to [0, 1] form) our labels and create our training/testing splits :

* lb = LabelBinarizer()
* labels = lb.fit\_transform(labels)
* labels = to\_categorical(labels)
* (trainX, testX, trainY, testY) = train\_test\_split(data, labels,

test\_size=0.20, stratify=labels, random\_state=42)

In order to ensure that our model generalizes, we perform data augmentation by setting the random image rotation setting to 15 degrees clockwise or counterclockwise.

* # initialize the training data augmentation object
* trainAug = ImageDataGenerator(rotation\_range=15, fill\_mode="nearest")

Then we instantiate the VGG16 network with weights pre-trained on ImageNet, leaving off the FC layer head, and construct a new fully-connected layer head and append it on top of VGG16. We then freeze the CONV weights of VGG16 such that only the FC layer head will be trained.

* baseModel = VGG16(weights="imagenet", include\_top=False,
* input\_tensor=Input(shape=(224, 224, 3)))
* 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)
* model = Model(inputs=baseModel.input, outputs=headModel)
* for layer in baseModel.layers:
* layer.trainable = False

Finally, we compile the network with learning rate decay and the Adam" optimizer. We use “binary\_crossentropy” as this is a 2-class problem. Then we make a call to [Kera’s HYPERLINK "https://www.pyimagesearch.com/2018/12/24/how-to-use-keras-fit-and-fit\_generator-a-hands-on-tutorial/"fit\_generator **HYPERLINK "https://www.pyimagesearch.com/2018/12/24/how-to-use-keras-fit-and-fit\_generator-a-hands-on-tutorial/",**](https://www.pyimagesearch.com/2018/12/24/how-to-use-keras-fit-and-fit_generator-a-hands-on-tutorial/) while passing in our chest X-ray data via our data augmentation object

* print("[INFO] compiling model...")
* opt = Adam(lr=INIT\_LR, decay=INIT\_LR / EPOCHS)
* model.compile(loss="binary\_crossentropy", optimizer=opt,

metrics=["accuracy"])

* 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,

pochs=EPOCHS)

Next, we’ll evaluate our model and generate a classification report using scikit-learn’s helper utility.

* 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\_))

We then plot our training accuracy/loss history for inspection, outputting the plot to an image file.

* 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"])

Then finally we save the trained model to disk.

* print("[INFO] saving COVID-19 detector model...")
* model.save(args["model"], save\_format="h5")

**Predictions**

The script predict.py takes an image as an -i argument and a trained model as an -m argument through command line. It then loads that model and returns the probability values for covid-19 and normal classes for the image and model in context.

Moreover, our training\_model.py script plots the following graph while evaluating our model.



**Result**

We see our model obtains 90-92% accuracy on our sample dataset based solely on x-ray images and no other data.

Using some statistical evaluation, we also observe that our model obtained **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.

Moreover, as our history plot shows, our network is not overfitting, despite having very limited data.