COVID-19 DIAGNOSIS WITH DEEP LEARNING

Patrick Junghenn & Matteo Bucalossi
Machine Learning II – The George Washington University
May 3rd, 2021

Introduction

As the COVID-19 pandemic incredibly challenged the healthcare system throughout the world, we can look into technology solutions that could be streamline and support healthcare and medical operations, particularly as we continue to deal with an ongoing pandemic. Such challenges include a shortage of hospital beds, overrun facilities and hospitals, and an overall shortage of tests for the general public. One problem has also been caused by the delay in getting test results back, especially in the early stages of the pandemic and with polymerase chain reaction (PCR) tests, as several days may be required before doctors could provide a reliable diagnosis for COVID-19.

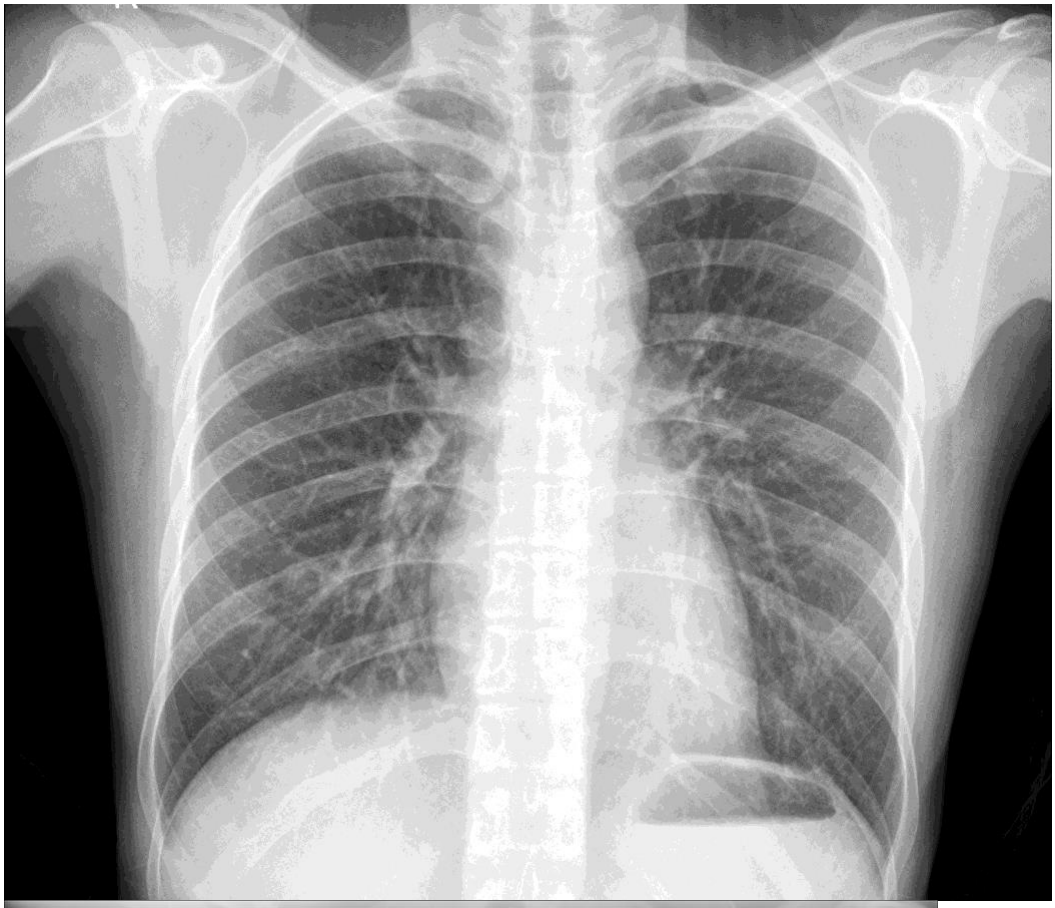
On the other hand, taking an X-ray of a patient can provide a tangible result in the matters of seconds. Providing with images of lungs that can be used to quickly diagnose patients with a suspected COVID-19 presence. Luckily, deep learning models such as convolutional neural networks (CNN) can be used to analyze and classify images, and thus improve prognostic predictions in triage phases.

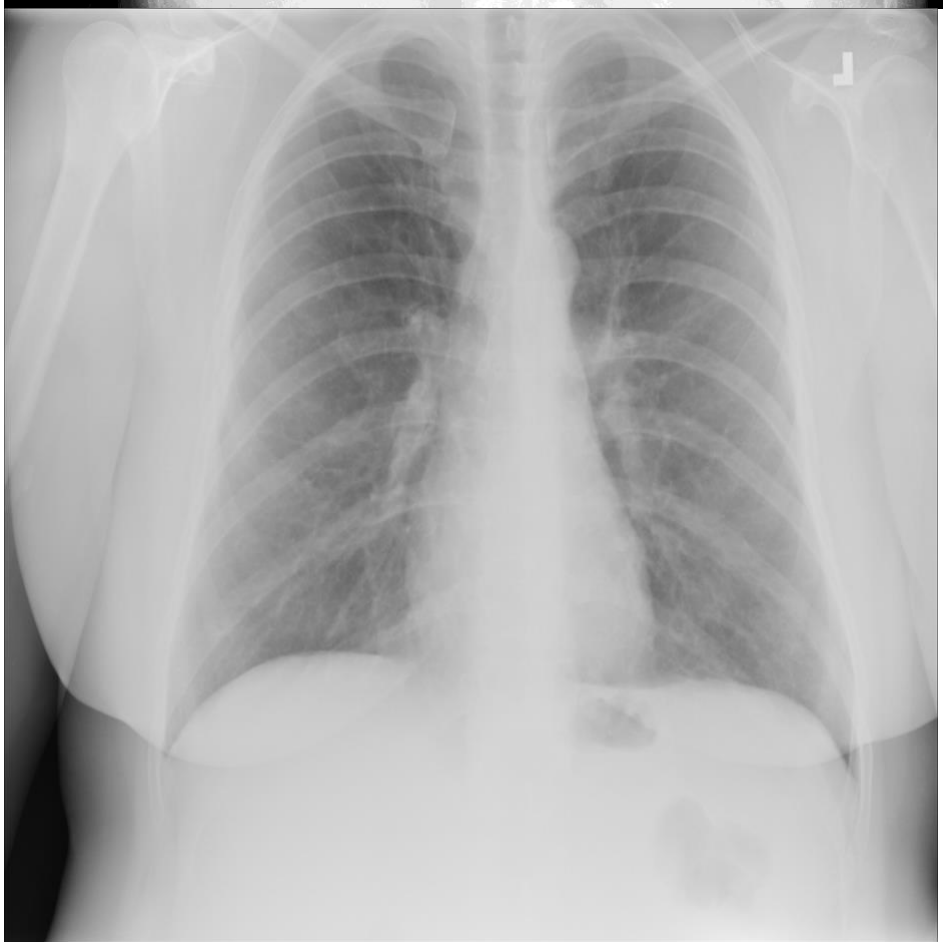
Since X-ray images can be quickly obtained from a radiology unit and easily fed into a computer for modeling and classification, this could turn out to be a very useful tool for overrun ER units and hospitals, speeding waiting and processing times significantly. Even considering a lower accuracy of 75% would bring incredible benefits to the diagnosis process: triage of suspected COVID patients would be much faster than now and hospitals would be able to prioritize severe patients first. This would also make the general public much safer, since COVID-19s rate of transmission depends on our ability to reliably identify infected patients with a lower rate of false negatives while also lowering false positives to ease the burden on healthcare facilities.

Description of the data

The dataset is available on Kaggle and it was prepared and cleaned by Joseph Paul Cohen at the University of Montreal. They built a public open dataset of chest X-ray and CT images of patients positive or suspected of COVID-19 and similar pneumonias, as collected from public sources as well as hospitals and physicians.

We used this dataset to train the model for this project using a set of 9,544 lung X-ray images, 5,500 of which were labeled as non-COVID cases and 4,044 as COVID ones. It is important to note that given the limited availability of X-ray images, the author used various augmentation techniques to expand the number of examples in the dataset. These included blurring, rotating, shifting, shearing, and brightness adjustments. We can see below an example of two X-ray images, respectively showing COVID-19 positive and negative cases.

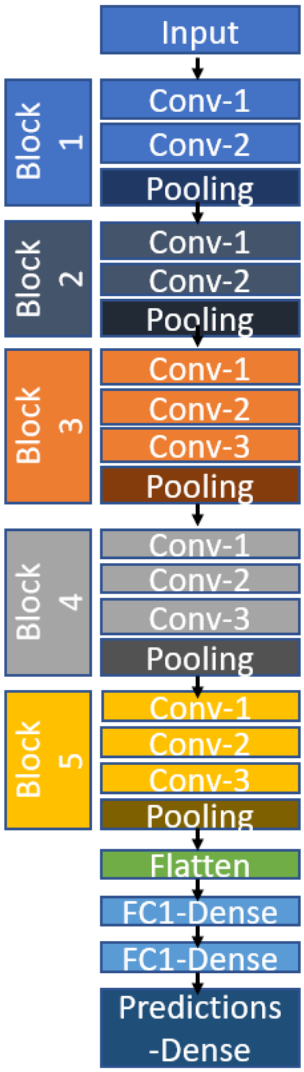




Description of the model

The problem at hand is one of image classification, and particularly of binary classification as we interested in predicting positive or negative diagnosis for the patients. Considering also the scarcity of training data We decided to use transfer learning to perform this classification task, hoping that such state-of-the-art technique would yield solid results.

We adopted the VGG16 model, a convolutional neural network with 16 layers. The model architecture can be seen below:



We can observe that the model uses convolution, pooling and fully connected layers Particularly, it starts with 2 convolution layers followed by pooling twice and then repeats 3 convolutions with pooling, and it ends with 3 dense layers including the prediction layer. We adopted the head of the model and eventually modified the final dense layers for our task.

VGG16 is implemented in TensorFlow's Keras framework, so we can easily call the pre-trained model from Keras' applications.

Experimental setup

As VGG16 only takes inputs of size 244x244, we resized our input vectors accordingly, and then split the dataset with 25% as test set. We then used the pre-trained VGG16 model and used transfer learning to get the weights learned on the Imagenet set. We finally added two dense layers to the base model with dropout, respectively with 64 and 1 neurons and ReLu and Sigmoid activation functions. This would provide the best output architecture for the base model as well as the binary image classification problem.

For training the model, we used binary cross entropy as metric for the loss, given the binary classification nature of our task, and we used accuracy to measure performance of the training. We decided to try a standard batch size of 128 and it turned out to work well. Because of the task at hand, Adam optimizer also seemed to be the best choice for our model.

We then implemented 2 callbacks for learning rate and early stopping. The former allowed to reduce the learning rate when a metric has stopped improving during training, with patience factor of 5 and a minimum value for the rate of 0.001. The model indeed benefited from this callback as soon as training stagnates there is no need for an higher learning rate. The second callback allowed the training to stop at the epoch when the performance stops improving on a hold-out validation set. We established an arbitrary number of epochs at 1000, and the training will be able to stop before then as we hope.

Results

We implemented and trained our model on GPU within Google Collab, which took less than one hour overall – one of the advantages of transfer learning. As we trained the model, the early stopping was able to stop the training process at the 88th epoch as showed below.

```
Epoch 82/100
11/11 [==============================] - 16s 1s/step - loss: 0.2625 - accuracy: 0.8996 - val_loss: 0.2740 - val_accuracy: 0.8958

Epoch 00082: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
Epoch 83/100
11/11 [==============================] - 17s 1s/step - loss: 0.2549 - accuracy: 0.9120 - val_loss: 0.2740 - val_accuracy: 0.8958
Epoch 84/100
11/11 [==============================] - 17s 1s/step - loss: 0.2588 - accuracy: 0.9124 - val_loss: 0.2740 - val_accuracy: 0.8958
Epoch 85/100
11/11 [==============================] - 16s 1s/step - loss: 0.2648 - accuracy: 0.8912 - val_loss: 0.2740 - val_accuracy: 0.8958
Epoch 86/100
11/11 [==============================] - 17s 1s/step - loss: 0.2552 - accuracy: 0.9180 - val_loss: 0.2740 - val_accuracy: 0.8958
Epoch 87/100
11/11 [==============================] - 16s 1s/step - loss: 0.2706 - accuracy: 0.9022 - val_loss: 0.2740 - val_accuracy: 0.8958

Epoch 00087: ReduceLROnPlateau reducing learning rate to 3.200000264769187e-07.
Epoch 88/100
11/11 [==============================] - 16s 1s/step - loss: 0.2719 - accuracy: 0.8917 - val_loss: 0.2740 - val_accuracy: 0.8958
```

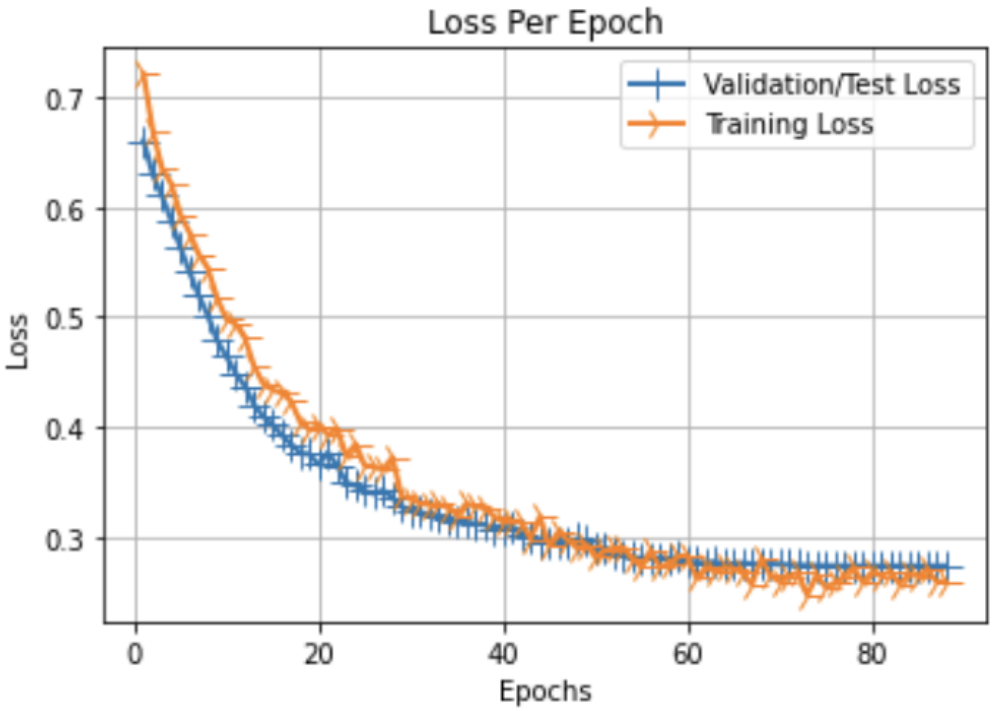
The model outputs confirmed that the learning rate was indeed reduced on the 59th epoch to 0.0002, on 70th epoch to 4, on the 77th to 8, on 82nd to 1.6, and finally on the 87th epoch to 3.2, this being the eventual value used to train the classifier.

After using the model to predict on the test set, we can see very strong results from the classification report by Keras.

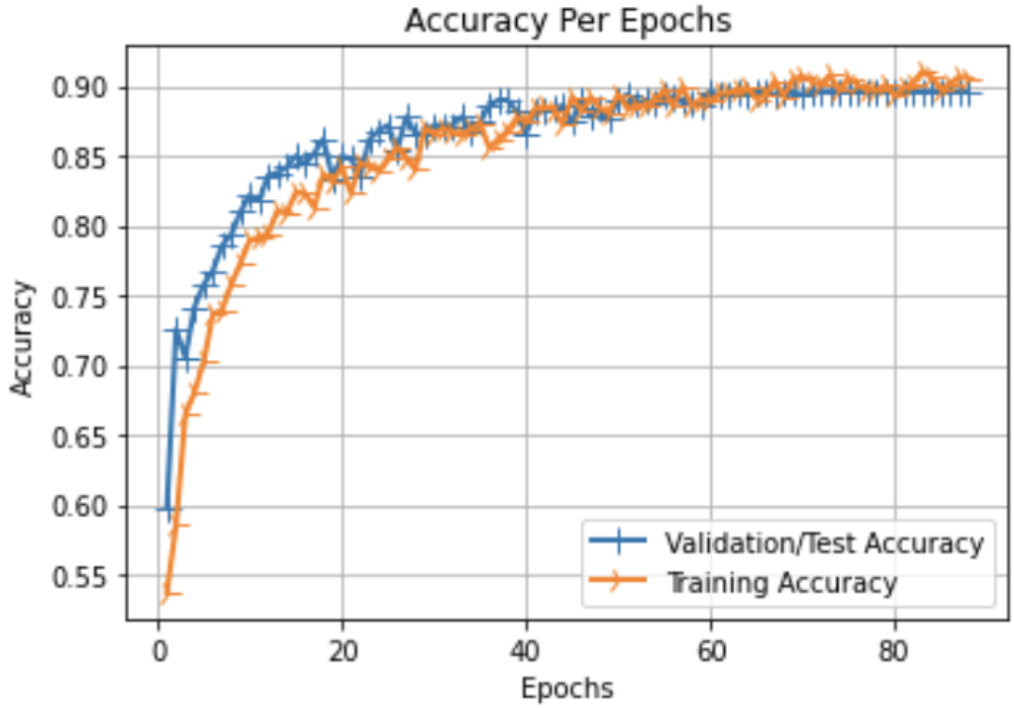
	precision	recall	f1-score
0	0.89	0.91	0.90
1	0.91	0.88	0.89
accuracy			0.90
macro avg	0.90	0.90	0.90
weighted avg	0.90	0.90	0.90

Using accuracy as a metric for evaluating the performance of our model, we can see a 90% accuracy score; at the same time, the weighted F1 score is also 90%, yielding a fairly solid result for our goal of streamlining and improving COVID-19 patients triage processes.

We can also visualize the loss per epoch as well as accuracy score per epoch.



We can clearly see that the test loss significantly decreases after 20 epochs and continues to do gradually so until 70+ epochs. This confirms the early stopping output at 80+ epochs, as we can see no further improvement in performance with this model would be likely to occur.



Equally, the accuracy improves greatly up until 20 epochs, to then only gradually increase until 70+ epochs, confirming what we have noticed from the loss function trend.

Finally, we can get a summary of the model, which conforms to the VGG16 architecture displayed above, with output dense layers with customized neuron size for our task.

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
average_pooling2d_5 (Average	(None, 1, 1, 512)	0
flatten_5 (Flatten)	(None, 512)	0
dense_10 (Dense)	(None, 64)	32832
dropout_5 (Dropout)	(None, 64)	0
dense_11 (Dense)	(None, 2)	130

Total params: 14,747,650
Trainable params: 32,962
Non-trainable params: 14,714,688

Conclusions

As stated in the goals of our project, even a lower accuracy result would have improved the clinical processes at stake: our output of 90% provides a strong classifier able to quickly diagnose likely COVID-19 positive cases. As we have assumed, the integration of such tool within the healthcare system could greatly benefit the response to this pandemic, easing the pressure on hospitals and ER units, and provide the public with quicker test results as well as better and timely assistance to the most severe patients.

A future project for implementing this tool on the front line would be to integrate this model within TensorFlow Lite, as the framework offers the capability to run MobileNets on mobile devices. Thus, we could export this tool to a mobile app and allow physicians and nurses to have access to this quick diagnosis tool directly on their devices, greatly improving the clinical workflow in times of crisis.

References

- Dataset: El-Shafai, Walid; Abd El-Samie, Fathi (2020), "Extensive COVID-19 X-Ray and CT Chest Images Dataset", Mendeley Data, V3, doi: 10.17632/8h65ywd2jr.3
- https://keras.io/api/callbacks/reduce_lr_on_plateau/
- https://keras.io/api/callbacks/early_stopping/
- https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a
- Joseph Paul Cohen and Paul Morrison and Lan Dao and Karsten Roth and Tim Q Duong and Marzyeh Ghassemi (2020) "COVID-19 Image Data Collection: Prospective Predictions Are the Future" arXiv:2006.11988
- Simonyan, K., Zisserman A. (2015) "Very Deep Convolutional Networks for Large-Scale Image Recognition" ICLR, arXiv:1409.1556

Appendix

```
# -*- coding: utf-8 -*-
from google.colab import drive

drive.mount('/content/drive')

from imutils import paths
import matplotlib.pyplot as plt
import tensorflow
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dropout, Flatten, Dense, Input,
AveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
```

```
import cv2
import os

imagePaths =
list(paths.list_images('/content/drive/Shareddrives/ML2_Final_Project/covid
'))

data = []
labels = []
for imagePath in imagePaths:
  print('.', end='')
  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)
data = np.array(data) / 255.0
labels = np.array(labels)

lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = tensorflow.keras.utils.to_categorical(labels)

# experimented with test size due to lack of data.
X_train, X_test, y_train, y_test = train_test_split(data, labels,
                                                    test_size = .25,
                                                    stratify=labels,
                                                    random_state=42)

# image augmentation for training
train_aug = ImageDataGenerator(rotation_range=15, fill_mode='nearest')

input_shape = (224, 224, 3)
# pretrained on imagenet
base_model = VGG16(weights='imagenet', include_top=False,
                   input_tensor=Input(shape=input_shape))

head_model = base_model.output
head_model = AveragePooling2D(pool_size=(4,4))(head_model)
head_model = Flatten()(head_model)
head_model = Dense(64, activation='relu')(head_model)
head_model = Dropout(0.5)(head_model)
head_model = Dense(1, activation='sigmoid')(head_model)

# head_model placed on top of vgg16
model = Model(inputs=base_model.input, outputs=head_model)
for layer in base_model.layers:
  layer.trainable = False

# callbacks
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss',
                              factor = 0.2,
                              patience = 5,
                              verbose = 1,
                              min_delta = 0.001)

earlystop = EarlyStopping(monitor = 'val_loss',
                          min_delta = 0,
                          patience = 5,
                          verbose = 1,
```

```
                           restore_best_weights = True)

callbacks = [reduce_lr, earlystop]

# compile model
n_epochs = 100
opt = Adam(lr=1e-3, decay= 1e-3/n_epochs)


model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])

# train full model
batch_size = 128
model_fitted = model.fit(train_aug.flow(X_train,
                                                    y_train,
                                                    batch_size= batch_size),
                                    steps_per_epoch=len(X_train) //
batch_size,
                                    validation_data=(X_test, y_test),
                                    validation_steps = len(X_test) //
batch_size,
                                    callbacks = callbacks,
                                    epochs=n_epochs)

# predictions
pred = model.predict(X_test, batch_size=batch_size)
pred = np.argmax(pred, axis=1)

# model evaluation
print(classification_report(y_test.argmax(axis=1), pred))

# Plotting loss of model with pretrained weights
fitted_dict = model_fitted.history
loss_values = fitted_dict['loss']
val_loss_values = fitted_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
line1 = plt.plot(epochs, val_loss_values, label='Validation/Test Loss')
line2 = plt.plot(epochs, loss_values, label='Training Loss')
plt.setp(line1, linewidth=2.0, marker = '+', markersize=12.0)
plt.setp(line2, linewidth=2.0, marker = '4', markersize=12.0)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Per Epoch')
plt.grid(True)
plt.legend()
plt.show()

# plotting accuracy
fitted_dict = model_fitted.history
acc_values = fitted_dict['accuracy']
val_acc_values = fitted_dict['val_accuracy']
epochs = range(1, len(loss_values) + 1)
line1 = plt.plot(epochs, val_acc_values, label='Validation/Test Accuracy')
line2 = plt.plot(epochs, acc_values, label='Training Accuracy')
plt.setp(line1, linewidth=2.0, marker = '+', markersize=10.0)
plt.setp(line2, linewidth=2.0, marker = '4', markersize=10.0)
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy Per Epochs')
plt.grid(True)
```

```
plt.legend()
plt.show()

print(model.summary()
```