Covid_prediction

April 4, 2022

0.1 Covid prediction project

1.1 Introduction The corona-virus outbreak came to light on December, 2019 when China informed the World Health Organisation of a cluster of cases of pneumonia of an unknown cause in Wuhan City in Hubei Province. Subsequently the disease spread to more Provinces in China, and to the rest of the world. The WHO has now declared it a pandemic. The virus has been named SARS-CoV-2 and the disease is now called COVID-19. The virus that causes COVID-19 is mainly transmitted through droplets generated when an infected person coughs, sneezes, or exhales. These droplets are too heavy to hang in the air, and quickly fall on floors or surfaces. You can be infected by breathing in the virus if you are within close proximity of someone who has COVID-19, or by touching a contaminated surface and then your eyes, nose or mouth. The COVID-19 pandemic has stretched the healthcare systems in every country in the world to its limit as they had to deal with a large number of deaths. Early detection of the COVID-19 in a faster, easier, and cheaper way can help in saving lives and reduce the burden on healthcare professionals. Artificial intelligence can play a big role in identifying COVID-19 by applying image processing techniques to X-ray images. As COVID-19 attacks the epithelial cells that line our respiratory tract, we can use medical X-rays to analyze the health of a patient's lung. Hence, the target of this project is to build an end-to-end project which processes the X-ray of the patients (Convolutional neural network algorithm) and predicts whether the person is COVID-19 positive or normal along with its probability. The use case which can be derived from this project is of prioritization of the patients on the basis of the X-ray classification being predicted (either normal or COVID-19 positive).

1.2 Dataset For the purpose of this project, the Large COVID-19 CT scan slice dataset from kaggle was used.

It consists of 7,593 COVID-19 images from 466 patients, 6,893 normal images from 604 patients, and 2,618 CAP images from 60 patients and it derives by curating data from 7 public datasets. Here, the CAP-CT images were removed, and only the former two were considered.

Note: Since a convolutional neural network will be created for solving this problem, and considering that CNN models require access to a GPU, Google Colab will be used. Colab notebooks are like those provided by jupyter, but they have GPU (NVIDIA Tesla K80) as an extra addition and tensorflow packages preinstalled.

1.3 Deep Learning models Three models were created to find the most suitable for the specific problem.

- A simple convolutional neural network.

- The same architecture was used for the second model, but data augmentation techniques were introduced, to find out if it will help to better accuracy and predictions.

- For the third model, transfer learning techniques were considered, by using a pretrained VGG-16 model (with data augmentation).

Data augmentation takes the approach of generating more training data from existing training samples, by augmenting the samples via a number of random transformations that yield believablelooking images. The goal is that at training time, the model will not see the exact same picture twice. This helps to expose the model to more aspects of the data and generalize better. All images were resized to 128x128 for faster training

[2]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import keras
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import os
import glob
import seaborn as sns
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from tensorflow.keras.optimizers import Adam ,RMSprop
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score
%matplotlib inline
```

1.3.1 Dataset split The dataset is not properly splited, as there are subfolders for COVID and NonCOVID predictions. However, to proceed to the formulation of the CNN, the dataset has to be preprocessed to train, test and validation splits. So we have to transform the structure of the dataset, so it will consist of train, test, and validation subfolders. Without this directory structure, we cannot use the flow_from_directory technique for the generators. There are several ways to achieve that, i.e. create a function to create folders with splits, but the split-folders Python module was considered, as it does all that automatically. An 80% train set was selected, with 10% for both validation and test splits.

[3]:
```
#  !pip install split-folders
#  !pip install visualkeras
# # For visualizing the created model
```

[4]:
```
#Setting the base directory
# DIR = '/content/gdrive/MyDrive/'
```

[5]:
```
import splitfolders # or import split_folders
# Split with a ratio.
# To only split into training and validation set, set a tuple to `ratio`,`(.8, .
↪2)`.
splitfolders.ratio("C:\\Users\\lefte\\Desktop\\curated_data", output="C:
↪\\Users\\lefte\\Desktop\\curated_data\\output", seed=1337, ratio=(.8,.1, .
↪1), group_prefix=None) # default values
```

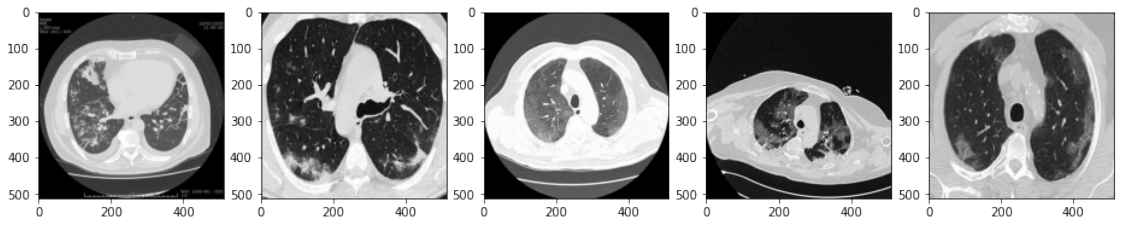
```
Copying files: 14486 files [00:17, 838.98 files/s]
```

[6]:
```
# Add covid ct images to train and test sets
source_covid = '.\\curated_data\\2COVID\\'
train_covid ='.\\curated_data\\output\\train\\2COVID'
test_covid = '.\\curated_data\\output\\test\\2COVID'
val_covid = '.\\curated_data\\output\\val\\2COVID'

# Add normal ct images to train and test sets
source_normal = '.\\curated_data\\1NonCOVID\\'
train_normal ='.\\curated_data\\output\\train\\1NonCOVID'
test_normal = '.\\curated_data\\output\\test\\1NonCOVID'
val_normal = '.\\curated_data\\output\\val\\1NonCOVID'
```

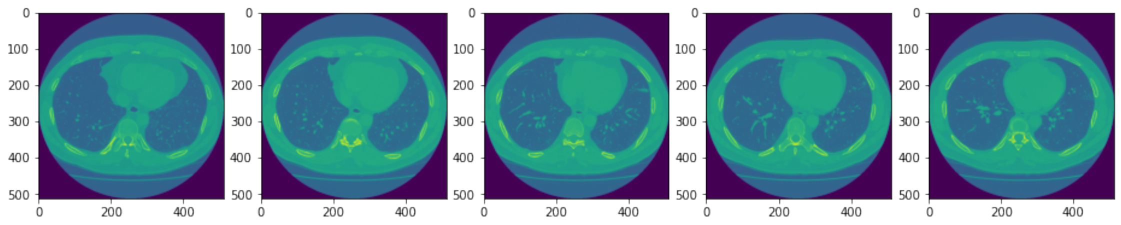
[7]:
```
# Overview of train and test datasets
print("Train covid images :",len(os.listdir(train_covid)))
print("Train normal images :",len(os.listdir(train_normal)))
print("Total training images :",len(os.listdir(train_covid))+ len(os.
↪listdir(train_normal)))
print()
print("Test covid images :",len(os.listdir(test_covid)))
print("Test normal images :",len(os.listdir(test_normal)))
print("Total testing images :",len(os.listdir(test_covid))+ len(os.
↪listdir(test_covid)))
print()
print("Validation covid images :",len(os.listdir(val_covid)))
print("Validation normal images :",len(os.listdir(val_normal)))
print("Total Validation images :",len(os.listdir(val_covid))+ len(os.
↪listdir(val_covid)))
```

```
Train covid images : 6074
Train normal images : 5514
Total training images : 11588

Test covid images : 760
Test normal images : 690
Total testing images : 1520

Validation covid images : 759
Validation normal images : 689
Total Validation images : 1518
```

[8]:
```
# Plotting the 5 first covid ct images
f,axes = plt.subplots(1,5, figsize=(16,4))
images = os.listdir(train_covid)[:5] # 5 images
for i, img in enumerate(images):
    img = plt.imread(os.path.join(train_covid,img))
    axes[i].imshow(img)
```



[9]:
```
# Plotting the first 5 normal ct images
f,axes = plt.subplots(1,5, figsize=(16,4))
images = os.listdir(train_normal)[:5]
for i, img in enumerate(images):
    img = plt.imread(os.path.join(train_normal,img))
    axes[i].imshow(img)
```



[10]:
```
TRAIN_PATH = '.\\curated_data\\output\\train\\'
TEST_PATH = '.\\curated_data\\output\\test\\'
VAL_PATH = '.\\curated_data\\output\\val\\'
BATCH_SIZE = 16
```

[11]:
```
#Image augmentation
trn_datagen = ImageDataGenerator(rescale = 1./255)
tst_datagen = ImageDataGenerator(rescale = 1./255)
#Training data generator
trn_generator = trn_datagen.flow_from_directory(TRAIN_PATH,
# target_size=(IMG_HEIGHT,IMG_WIDTH),
```

1.4 First model

[14]:
```
#Image augmentation
trn_datagen = ImageDataGenerator(rescale = 1./255
                                   )

tst_datagen = ImageDataGenerator(rescale = 1./255)

#Training data generator
trn_generator = trn_datagen.flow_from_directory(TRAIN_PATH,
                                                  # target_size=(IMG_HEIGHT,␣
 ↪IMG_WIDTH),
                                                  batch_size=BATCH_SIZE,
                                                  color_mode="grayscale",
                                                  shuffle=True,
                                                  target_size=(128, 128),
                                                  class_mode='binary')

#Validation data generator
valid_generator = tst_datagen.flow_from_directory(VAL_PATH,
                                                        # ␣
 ↪target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                         batch_size=BATCH_SIZE,
                                                         class_mode='binary',
                                                         shuffle=True,
                                                         target_size=(128, 128),
                                                         color_mode="grayscale")

tst_generator = tst_datagen.flow_from_directory(TEST_PATH,
                                                        # ␣
 ↪target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                         batch_size=BATCH_SIZE,
                                                         class_mode='binary',
                                                         shuffle=False,
                                                         target_size=(128, 128),
                                                         color_mode="grayscale")
```

```
Found 11588 images belonging to 2 classes.
Found 1448 images belonging to 2 classes.
Found 1450 images belonging to 2 classes.
```

[15]:
```
lmodel = Sequential()

lmodel.add(Conv2D(128,(3, 3), activation='relu', padding="valid",␣
 ↪input_shape=(128, 128, 1)))
lmodel.add(MaxPooling2D(pool_size=(2, 2)))

lmodel.add(Conv2D(64,(3, 3), activation='relu', padding="valid"))
lmodel.add(MaxPooling2D(pool_size=(2, 2)))

lmodel.add(Flatten()) # Flattening involves transforming the entire pooled␣
 ↪feature map
# matrix into a single column which is then fed to the neural network for␣
 ↪processing.

lmodel.add(Dense(32, activation='relu'))
lmodel.add(Dropout(0.5))

lmodel.add(Dense(1, activation='sigmoid'))
```

[16]:
```
lmodel.compile(loss='binary_crossentropy', optimizer=keras.optimizers.
 ↪RMSprop(lr=2e-5), metrics=['accuracy'])

history = lmodel.fit_generator(trn_generator,
                              verbose=1,
                              validation_data = valid_generator,
                              validation_steps = 50,
                              steps_per_epoch = 100,
                              epochs=100)
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/training.py:1844: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/100
100/100 [==============================] - 1200s 12s/step - loss: 0.6612 -
accuracy: 0.6647 - val_loss: 0.5639 - val_accuracy: 0.8000
Epoch 2/100
100/100 [==============================] - 883s 9s/step - loss: 0.5260 -
accuracy: 0.8235 - val_loss: 0.5193 - val_accuracy: 0.7912
Epoch 3/100
100/100 [==============================] - 754s 8s/step - loss: 0.5006 -
accuracy: 0.8151 - val_loss: 0.4833 - val_accuracy: 0.8050
Epoch 4/100
100/100 [==============================] - 583s 6s/step - loss: 0.4979 -
accuracy: 0.8056 - val_loss: 0.4545 - val_accuracy: 0.8150
Epoch 5/100
100/100 [==============================] - 525s 5s/step - loss: 0.4835 -
accuracy: 0.8068 - val_loss: 0.4555 - val_accuracy: 0.8000
Epoch 6/100
100/100 [==============================] - 431s 4s/step - loss: 0.4601 -
accuracy: 0.8016 - val_loss: 0.4364 - val_accuracy: 0.8112
Epoch 7/100
100/100 [==============================] - 380s 4s/step - loss: 0.4822 -
```

```
accuracy: 0.7816 - val_loss: 0.4251 - val_accuracy: 0.8138
Epoch 8/100
100/100 [==============================] - 344s 3s/step - loss: 0.4104 -
accuracy: 0.8469 - val_loss: 0.4186 - val_accuracy: 0.8087
Epoch 9/100
100/100 [==============================] - 269s 3s/step - loss: 0.4497 -
accuracy: 0.8072 - val_loss: 0.4170 - val_accuracy: 0.8200
Epoch 10/100
100/100 [==============================] - 228s 2s/step - loss: 0.4318 -
accuracy: 0.8118 - val_loss: 0.3943 - val_accuracy: 0.8288
Epoch 11/100
100/100 [==============================] - 216s 2s/step - loss: 0.3957 -
accuracy: 0.8375 - val_loss: 0.3822 - val_accuracy: 0.8288
Epoch 12/100
100/100 [==============================] - 201s 2s/step - loss: 0.4208 -
accuracy: 0.8283 - val_loss: 0.3827 - val_accuracy: 0.8300
Epoch 13/100
100/100 [==============================] - 157s 2s/step - loss: 0.4047 -
accuracy: 0.8240 - val_loss: 0.3878 - val_accuracy: 0.8388
Epoch 14/100
100/100 [==============================] - 140s 1s/step - loss: 0.4114 -
accuracy: 0.8225 - val_loss: 0.3714 - val_accuracy: 0.8525
Epoch 15/100
100/100 [==============================] - 124s 1s/step - loss: 0.3989 -
accuracy: 0.8276 - val_loss: 0.3608 - val_accuracy: 0.8438
Epoch 16/100
100/100 [==============================] - 114s 1s/step - loss: 0.3967 -
accuracy: 0.8432 - val_loss: 0.3719 - val_accuracy: 0.8587
Epoch 17/100
100/100 [==============================] - 88s 885ms/step - loss: 0.3696 -
accuracy: 0.8624 - val_loss: 0.3827 - val_accuracy: 0.8238
Epoch 18/100
100/100 [==============================] - 78s 763ms/step - loss: 0.3691 -
accuracy: 0.8440 - val_loss: 0.3632 - val_accuracy: 0.8587
Epoch 19/100
100/100 [==============================] - 70s 704ms/step - loss: 0.3576 -
accuracy: 0.8697 - val_loss: 0.3524 - val_accuracy: 0.8637
Epoch 20/100
100/100 [==============================] - 66s 665ms/step - loss: 0.3547 -
accuracy: 0.8655 - val_loss: 0.3476 - val_accuracy: 0.8625
Epoch 21/100
100/100 [==============================] - 61s 617ms/step - loss: 0.3524 -
accuracy: 0.8567 - val_loss: 0.3500 - val_accuracy: 0.8438
Epoch 22/100
100/100 [==============================] - 45s 453ms/step - loss: 0.3598 -
accuracy: 0.8452 - val_loss: 0.3564 - val_accuracy: 0.8650
Epoch 23/100
100/100 [==============================] - 48s 483ms/step - loss: 0.3443 -
```

```
accuracy: 0.8714 - val_loss: 0.3308 - val_accuracy: 0.8612
Epoch 24/100
100/100 [==============================] - 41s 409ms/step - loss: 0.3414 -
accuracy: 0.8807 - val_loss: 0.3397 - val_accuracy: 0.8700
Epoch 25/100
100/100 [==============================] - 37s 377ms/step - loss: 0.3690 -
accuracy: 0.8376 - val_loss: 0.3553 - val_accuracy: 0.8725
Epoch 26/100
100/100 [==============================] - 32s 320ms/step - loss: 0.3307 -
accuracy: 0.8728 - val_loss: 0.3411 - val_accuracy: 0.8662
Epoch 27/100
100/100 [==============================] - 34s 338ms/step - loss: 0.3494 -
accuracy: 0.8674 - val_loss: 0.3523 - val_accuracy: 0.8687
Epoch 28/100
100/100 [==============================] - 35s 348ms/step - loss: 0.3870 -
accuracy: 0.8443 - val_loss: 0.3270 - val_accuracy: 0.8800
Epoch 29/100
100/100 [==============================] - 34s 340ms/step - loss: 0.3546 -
accuracy: 0.8580 - val_loss: 0.3367 - val_accuracy: 0.8700
Epoch 30/100
100/100 [==============================] - 22s 223ms/step - loss: 0.3517 -
accuracy: 0.8623 - val_loss: 0.3102 - val_accuracy: 0.8913
Epoch 31/100
100/100 [==============================] - 26s 264ms/step - loss: 0.3321 -
accuracy: 0.8776 - val_loss: 0.3089 - val_accuracy: 0.8875
Epoch 32/100
100/100 [==============================] - 19s 191ms/step - loss: 0.3221 -
accuracy: 0.8830 - val_loss: 0.3331 - val_accuracy: 0.8700
Epoch 33/100
100/100 [==============================] - 20s 206ms/step - loss: 0.3163 -
accuracy: 0.8733 - val_loss: 0.3314 - val_accuracy: 0.8775
Epoch 34/100
100/100 [==============================] - 23s 230ms/step - loss: 0.3201 -
accuracy: 0.8757 - val_loss: 0.3126 - val_accuracy: 0.8925
Epoch 35/100
100/100 [==============================] - 17s 162ms/step - loss: 0.3510 -
accuracy: 0.8647 - val_loss: 0.3247 - val_accuracy: 0.8750
Epoch 36/100
100/100 [==============================] - 18s 177ms/step - loss: 0.3098 -
accuracy: 0.8883 - val_loss: 0.2972 - val_accuracy: 0.8925
Epoch 37/100
100/100 [==============================] - 21s 212ms/step - loss: 0.3410 -
accuracy: 0.8702 - val_loss: 0.3099 - val_accuracy: 0.8788
Epoch 38/100
100/100 [==============================] - 19s 188ms/step - loss: 0.3152 -
accuracy: 0.8707 - val_loss: 0.2935 - val_accuracy: 0.8900
Epoch 39/100
100/100 [==============================] - 17s 171ms/step - loss: 0.3179 -
```

```
accuracy: 0.8655 - val_loss: 0.3267 - val_accuracy: 0.8662
Epoch 40/100
100/100 [==============================] - 17s 165ms/step - loss: 0.3068 -
accuracy: 0.8796 - val_loss: 0.2925 - val_accuracy: 0.8950
Epoch 41/100
100/100 [==============================] - 15s 146ms/step - loss: 0.3357 -
accuracy: 0.8806 - val_loss: 0.3294 - val_accuracy: 0.8700
Epoch 42/100
100/100 [==============================] - 17s 169ms/step - loss: 0.3306 -
accuracy: 0.8662 - val_loss: 0.3157 - val_accuracy: 0.8825
Epoch 43/100
100/100 [==============================] - 18s 184ms/step - loss: 0.3389 -
accuracy: 0.8634 - val_loss: 0.2898 - val_accuracy: 0.8913
Epoch 44/100
100/100 [==============================] - 16s 163ms/step - loss: 0.3136 -
accuracy: 0.8734 - val_loss: 0.3017 - val_accuracy: 0.8813
Epoch 45/100
100/100 [==============================] - 15s 155ms/step - loss: 0.3351 -
accuracy: 0.8698 - val_loss: 0.3004 - val_accuracy: 0.8863
Epoch 46/100
100/100 [==============================] - 15s 141ms/step - loss: 0.2979 -
accuracy: 0.8876 - val_loss: 0.2771 - val_accuracy: 0.9038
Epoch 47/100
100/100 [==============================] - 15s 145ms/step - loss: 0.3252 -
accuracy: 0.8749 - val_loss: 0.3007 - val_accuracy: 0.8863
Epoch 48/100
100/100 [==============================] - 15s 147ms/step - loss: 0.3320 -
accuracy: 0.8770 - val_loss: 0.3125 - val_accuracy: 0.8737
Epoch 49/100
100/100 [==============================] - 16s 156ms/step - loss: 0.3194 -
accuracy: 0.8809 - val_loss: 0.2837 - val_accuracy: 0.9000
Epoch 50/100
100/100 [==============================] - 15s 149ms/step - loss: 0.2996 -
accuracy: 0.8846 - val_loss: 0.3135 - val_accuracy: 0.8750
Epoch 51/100
100/100 [==============================] - 14s 144ms/step - loss: 0.3066 -
accuracy: 0.8864 - val_loss: 0.2943 - val_accuracy: 0.8925
Epoch 52/100
100/100 [==============================] - 14s 142ms/step - loss: 0.3281 -
accuracy: 0.8757 - val_loss: 0.2851 - val_accuracy: 0.8938
Epoch 53/100
100/100 [==============================] - 15s 148ms/step - loss: 0.2862 -
accuracy: 0.8822 - val_loss: 0.2915 - val_accuracy: 0.8900
Epoch 54/100
100/100 [==============================] - 14s 141ms/step - loss: 0.2782 -
accuracy: 0.8904 - val_loss: 0.2929 - val_accuracy: 0.8900
Epoch 55/100
100/100 [==============================] - 14s 142ms/step - loss: 0.3156 -
```

```
accuracy: 0.8795 - val_loss: 0.2626 - val_accuracy: 0.8913
Epoch 56/100
100/100 [==============================] - 14s 144ms/step - loss: 0.3334 -
accuracy: 0.8669 - val_loss: 0.2793 - val_accuracy: 0.8938
Epoch 57/100
100/100 [==============================] - 15s 151ms/step - loss: 0.3327 -
accuracy: 0.8678 - val_loss: 0.2975 - val_accuracy: 0.8800
Epoch 58/100
100/100 [==============================] - 15s 154ms/step - loss: 0.2700 -
accuracy: 0.8984 - val_loss: 0.2681 - val_accuracy: 0.9000
Epoch 59/100
100/100 [==============================] - 15s 154ms/step - loss: 0.3024 -
accuracy: 0.8875 - val_loss: 0.2812 - val_accuracy: 0.8938
Epoch 60/100
100/100 [==============================] - 14s 144ms/step - loss: 0.3051 -
accuracy: 0.8862 - val_loss: 0.2809 - val_accuracy: 0.8963
Epoch 61/100
100/100 [==============================] - 14s 142ms/step - loss: 0.2689 -
accuracy: 0.9093 - val_loss: 0.2908 - val_accuracy: 0.8863
Epoch 62/100
100/100 [==============================] - 14s 142ms/step - loss: 0.2721 -
accuracy: 0.8991 - val_loss: 0.2947 - val_accuracy: 0.8800
Epoch 63/100
100/100 [==============================] - 14s 143ms/step - loss: 0.3031 -
accuracy: 0.8856 - val_loss: 0.2553 - val_accuracy: 0.9050
Epoch 64/100
100/100 [==============================] - 14s 141ms/step - loss: 0.3037 -
accuracy: 0.8842 - val_loss: 0.2784 - val_accuracy: 0.9013
Epoch 65/100
100/100 [==============================] - 14s 142ms/step - loss: 0.2876 -
accuracy: 0.8943 - val_loss: 0.2867 - val_accuracy: 0.8838
Epoch 66/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2707 -
accuracy: 0.9003 - val_loss: 0.2862 - val_accuracy: 0.8900
Epoch 67/100
100/100 [==============================] - 14s 144ms/step - loss: 0.2630 -
accuracy: 0.9102 - val_loss: 0.2817 - val_accuracy: 0.8938
Epoch 68/100
100/100 [==============================] - 14s 141ms/step - loss: 0.2657 -
accuracy: 0.8999 - val_loss: 0.2789 - val_accuracy: 0.8875
Epoch 69/100
100/100 [==============================] - 14s 144ms/step - loss: 0.2739 -
accuracy: 0.9036 - val_loss: 0.2801 - val_accuracy: 0.8925
Epoch 70/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2652 -
accuracy: 0.8869 - val_loss: 0.2650 - val_accuracy: 0.9000
Epoch 71/100
100/100 [==============================] - 14s 141ms/step - loss: 0.2878 -
```

```
accuracy: 0.8970 - val_loss: 0.2661 - val_accuracy: 0.8975
Epoch 72/100
100/100 [==============================] - 14s 142ms/step - loss: 0.2908 -
accuracy: 0.8851 - val_loss: 0.2656 - val_accuracy: 0.8988
Epoch 73/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2796 -
accuracy: 0.9007 - val_loss: 0.2708 - val_accuracy: 0.9013
Epoch 74/100
100/100 [==============================] - 15s 146ms/step - loss: 0.3245 -
accuracy: 0.8778 - val_loss: 0.2670 - val_accuracy: 0.9025
Epoch 75/100
100/100 [==============================] - 14s 144ms/step - loss: 0.2751 -
accuracy: 0.8998 - val_loss: 0.2699 - val_accuracy: 0.8963
Epoch 76/100
100/100 [==============================] - 14s 144ms/step - loss: 0.2741 -
accuracy: 0.8961 - val_loss: 0.2747 - val_accuracy: 0.8913
Epoch 77/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2789 -
accuracy: 0.9059 - val_loss: 0.2467 - val_accuracy: 0.9100
Epoch 78/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2762 -
accuracy: 0.8931 - val_loss: 0.2358 - val_accuracy: 0.9075
Epoch 79/100
100/100 [==============================] - 14s 144ms/step - loss: 0.3090 -
accuracy: 0.8982 - val_loss: 0.2474 - val_accuracy: 0.9100
Epoch 80/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2984 -
accuracy: 0.8944 - val_loss: 0.2446 - val_accuracy: 0.9100
Epoch 81/100
100/100 [==============================] - 14s 144ms/step - loss: 0.2906 -
accuracy: 0.8854 - val_loss: 0.2764 - val_accuracy: 0.9000
Epoch 82/100
100/100 [==============================] - 14s 144ms/step - loss: 0.2629 -
accuracy: 0.8923 - val_loss: 0.2614 - val_accuracy: 0.8975
Epoch 83/100
100/100 [==============================] - 14s 144ms/step - loss: 0.3108 -
accuracy: 0.8761 - val_loss: 0.2795 - val_accuracy: 0.8888
Epoch 84/100
100/100 [==============================] - 14s 144ms/step - loss: 0.2774 -
accuracy: 0.8950 - val_loss: 0.2619 - val_accuracy: 0.9000
Epoch 85/100
100/100 [==============================] - 14s 144ms/step - loss: 0.2498 -
accuracy: 0.9060 - val_loss: 0.2566 - val_accuracy: 0.9038
Epoch 86/100
100/100 [==============================] - 14s 144ms/step - loss: 0.2681 -
accuracy: 0.9124 - val_loss: 0.2659 - val_accuracy: 0.9062
Epoch 87/100
100/100 [==============================] - 14s 141ms/step - loss: 0.2895 -
```

```
accuracy: 0.8873 - val_loss: 0.2340 - val_accuracy: 0.9212
Epoch 88/100
100/100 [==============================] - 15s 147ms/step - loss: 0.2598 -
accuracy: 0.9047 - val_loss: 0.2278 - val_accuracy: 0.9125
Epoch 89/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2462 -
accuracy: 0.9130 - val_loss: 0.2480 - val_accuracy: 0.9087
Epoch 90/100
100/100 [==============================] - 14s 142ms/step - loss: 0.2374 -
accuracy: 0.9193 - val_loss: 0.2330 - val_accuracy: 0.9112
Epoch 91/100
100/100 [==============================] - 14s 145ms/step - loss: 0.2545 -
accuracy: 0.9029 - val_loss: 0.2667 - val_accuracy: 0.8975
Epoch 92/100
100/100 [==============================] - 14s 145ms/step - loss: 0.2656 -
accuracy: 0.8940 - val_loss: 0.2435 - val_accuracy: 0.9038
Epoch 93/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2426 -
accuracy: 0.9045 - val_loss: 0.2208 - val_accuracy: 0.9162
Epoch 94/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2173 -
accuracy: 0.9215 - val_loss: 0.2433 - val_accuracy: 0.9112
Epoch 95/100
100/100 [==============================] - 15s 149ms/step - loss: 0.2396 -
accuracy: 0.9075 - val_loss: 0.2728 - val_accuracy: 0.8925
Epoch 96/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2458 -
accuracy: 0.9069 - val_loss: 0.2441 - val_accuracy: 0.9100
Epoch 97/100
100/100 [==============================] - 14s 143ms/step - loss: 0.2841 -
accuracy: 0.8859 - val_loss: 0.2553 - val_accuracy: 0.9000
Epoch 98/100
100/100 [==============================] - 14s 140ms/step - loss: 0.2541 -
accuracy: 0.9071 - val_loss: 0.2529 - val_accuracy: 0.8950
Epoch 99/100
100/100 [==============================] - 14s 142ms/step - loss: 0.2613 -
accuracy: 0.8964 - val_loss: 0.2639 - val_accuracy: 0.8963
Epoch 100/100
100/100 [==============================] - 15s 146ms/step - loss: 0.2510 -
accuracy: 0.9153 - val_loss: 0.2551 - val_accuracy: 0.9000
```

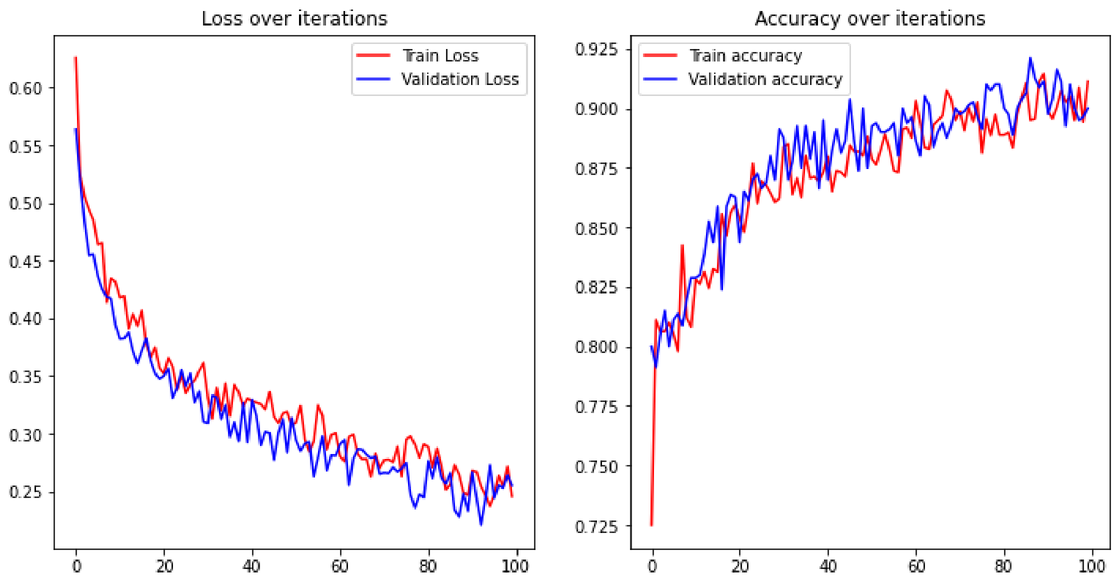
[17]:
```
history.history.keys()
```

[17]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[18]:
```
n = len(history.history["loss"])

fig = plt.figure(figsize=(12,6))
```

```
ax = fig.add_subplot(1,2,1)
ax.plot(range(n), (history.history["loss"]),'r', label="Train Loss")
ax.plot(range(n), (history.history["val_loss"]),'b', label="Validation Loss")
ax.legend()
ax.set_title('Loss over iterations')

ax = fig.add_subplot(1,2,2)
ax.plot(range(n), (history.history["accuracy"]),'r', label="Train accuracy")
ax.plot(range(n), (history.history["val_accuracy"]),'b', label="Validation␣
 ↪accuracy")
ax.legend()

ax.set_title('Accuracy over iterations')
```

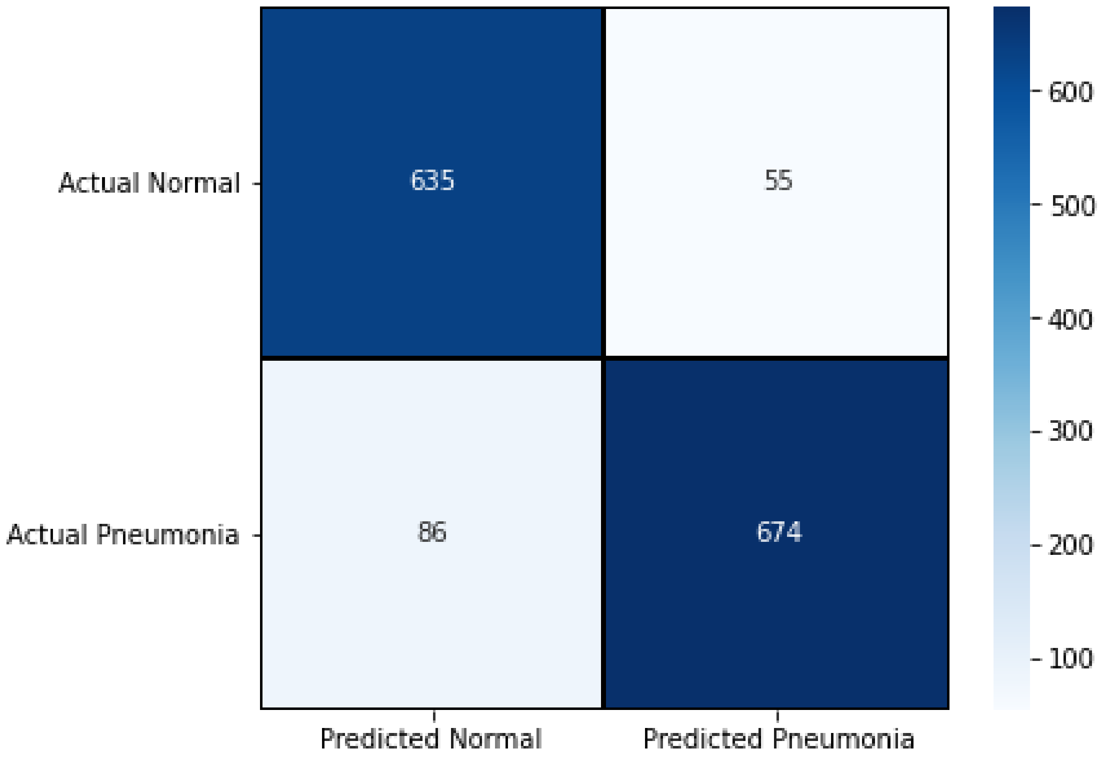
[18]:
```
Text(0.5, 1.0, 'Accuracy over iterations')
```



[19]:
```
predics = lmodel.predict_classes(tst_generator)
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
[20]: cm_l = confusion_matrix(tst_generator.classes, predics)
      cm_pdl = pd.DataFrame(cm_l , index = ['0','1'] , columns = ['0','1'])
```

```
[21]: plt.figure(figsize = (6,5))
      sns.heatmap(cm_pdl,cmap= "Blues", linecolor = 'black' ,
                  linewidth = 1 , annot = True, fmt='',
                  xticklabels = ['Predicted Normal', 'Predicted Pneumonia'],
                  yticklabels = ['Actual Normal', 'Actual Pneumonia'])
      plt.yticks(rotation=0)
      plt.show()
```



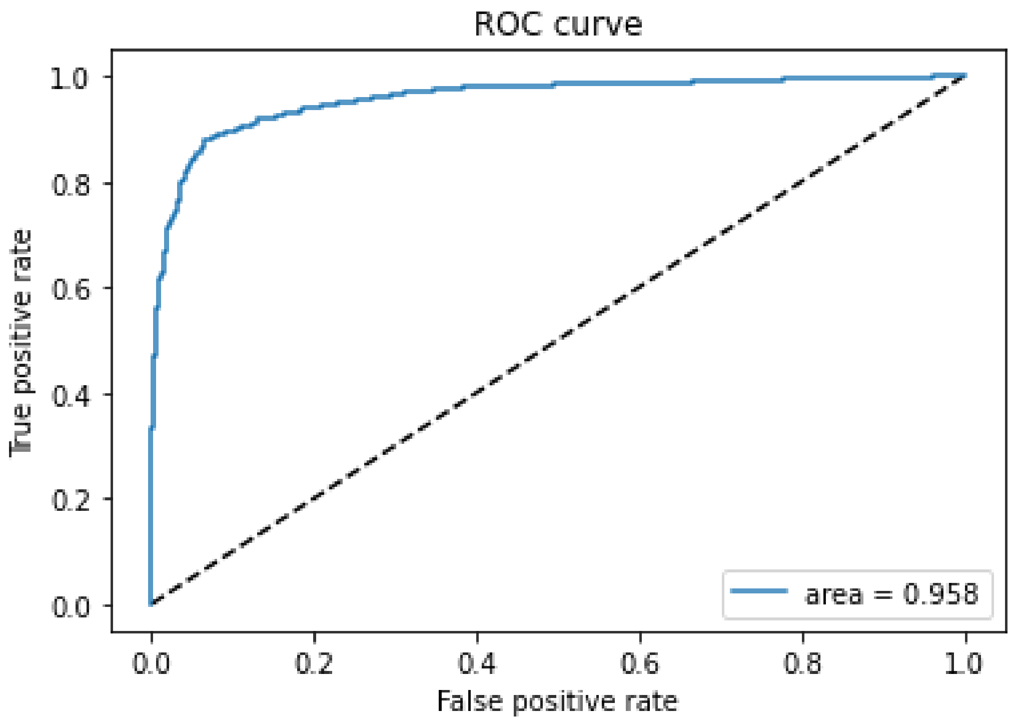
```
[23]: # make a prediction
      y_pred_kerasl = lmodel.predict_generator(tst_generator, valid_generator.samples␣
       ↪// valid_generator.batch_size+1) #(test_gen, steps=len(df_val), verbose=1)
      fpr_kerasl, tpr_kerasl, thresholds_kerasl = roc_curve(tst_generator.classes,␣
       ↪y_pred_kerasl)
      auc_kerasl = auc(fpr_kerasl, tpr_kerasl)
      auc_kerasl
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/training.py:1905: UserWarning:
`Model.predict_generator` is deprecated and will be removed in a future version.
Please use `Model.predict`, which supports generators.
```

```
  warnings.warn('`Model.predict_generator` is deprecated and '
```

[23]:
```
0.9575381388253241
```

[25]:
```
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_kerasl, tpr_kerasl, label='area = {:.3f}'.format(auc_kerasl))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```



1.5 Second model

[ ]:
```
#Image augmentation
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   horizontal_flip=True, # randomly flipping␣
 ↪half the images horizontally
                                   rotation_range=5, # in value of degrees, a␣
 ↪range within which to randomly rotate pictures
                                   width_shift_range=0.05,
                                   height_shift_range=0.05,
                                    shear_range=0.05, # randomly applying shear␣
 ↪transformations
                                    zoom_range=0.05, # randomly zooming images
                                    )

test_datagen = ImageDataGenerator(rescale = 1./255) # Data augmentation␣
 ↪techniques ONLY for the train_datagen

#Training data generator
train_generator = train_datagen.flow_from_directory(TRAIN_PATH,
                                                   # target_size=(IMG_HEIGHT,␣
 ↪IMG_WIDTH),
                                                   batch_size=BATCH_SIZE,
                                                   color_mode="grayscale",
                                                   shuffle=True,
                                                   target_size=(128, 128),
                                                   class_mode='binary')

#Validation data generator
validation_generator = test_datagen.flow_from_directory(VAL_PATH,
                                                         # ␣
 ↪target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                         batch_size=BATCH_SIZE,
                                                         class_mode='binary',
                                                         shuffle=True,
                                                         target_size=(128, 128),
                                                         color_mode="grayscale")

test_generator = test_datagen.flow_from_directory(TEST_PATH,
                                                         # ␣
 ↪target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                         batch_size=BATCH_SIZE,
                                                         class_mode='binary',
                                                         shuffle=False,
                                                         target_size=(128, 128),
                                                         color_mode="grayscale")
```

```
Found 11588 images belonging to 2 classes.
Found 1448 images belonging to 2 classes.
Found 1450 images belonging to 2 classes.
```

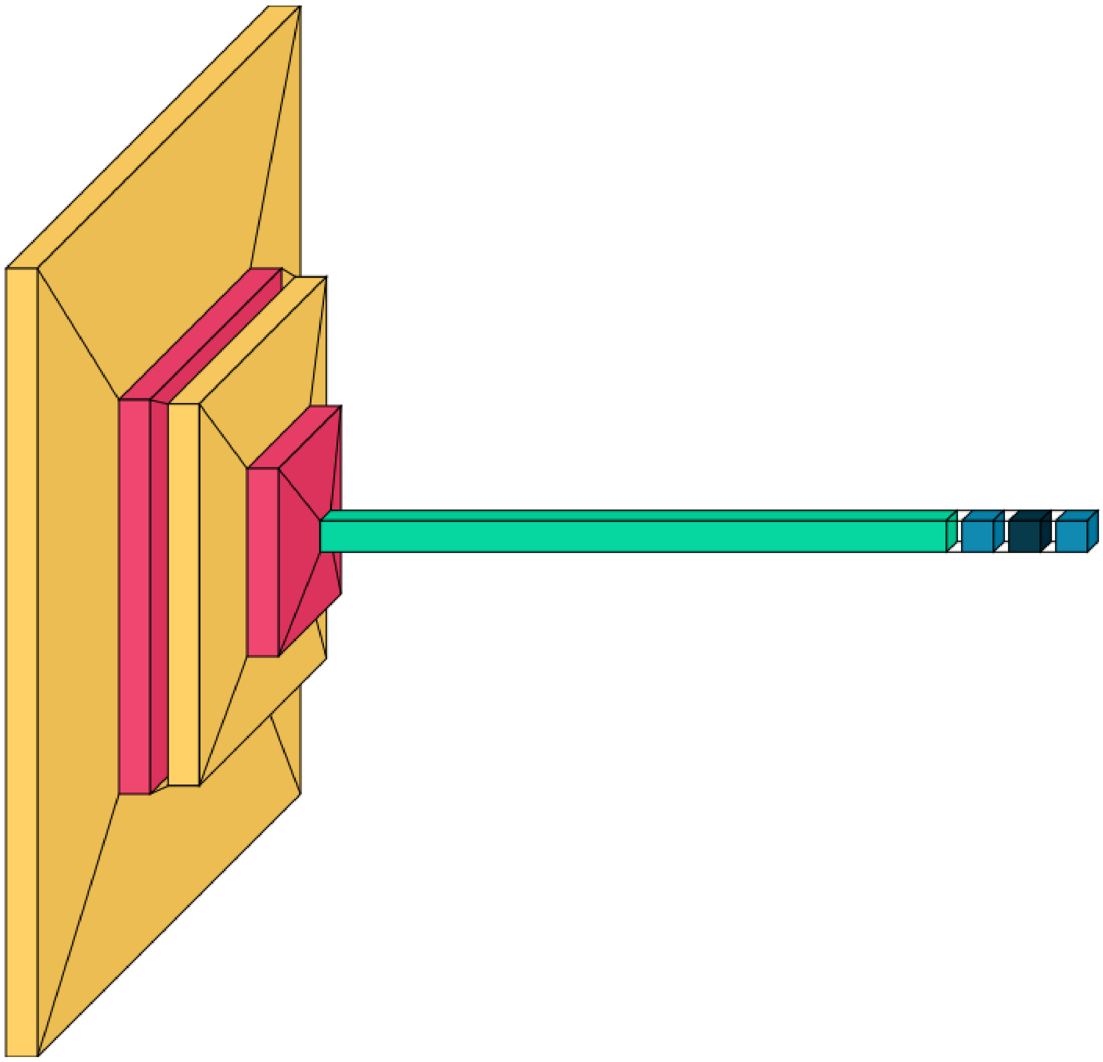
[ ]:
```
print(train_generator.class_indices)
print(test_generator.class_indices)
```

```
{'1NonCOVID': 0, '2COVID': 1}
{'1NonCOVID': 0, '2COVID': 1}
```

```
[ ]: model = Sequential()

     model.add(Conv2D(128,(3, 3), activation='relu', padding="valid",␣
      ↪input_shape=(128, 128, 1)))
     model.add(MaxPooling2D(pool_size=(2, 2)))

     model.add(Conv2D(64,(3, 3), activation='relu', padding="valid"))
     model.add(MaxPooling2D(pool_size=(2, 2)))

     model.add(Flatten()) # Flattening involves transforming the entire pooled␣
      ↪feature map
     # matrix into a single column which is then fed to the neural network for␣
      ↪processing.

     model.add(Dense(32, activation='relu'))
     model.add(Dropout(0.5))

     model.add(Dense(1, activation='sigmoid'))
```

```
[ ]: import visualkeras

     visualkeras.layered_view(model).show() # display using your system viewer
     visualkeras.layered_view(model, to_file='output.png') # write to disk
     visualkeras.layered_view(model, to_file='output.png').show() # write and show

     visualkeras.layered_view(model)
```

```
[ ]:
```



```
[ ]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 128)     1280
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 128)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 64)        73792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 57600)             0
```

```
_________________________________________________________________
dense (Dense)                (None, 32)                1843232
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,918,337
Trainable params: 1,918,337
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
model.compile(loss='binary_crossentropy', optimizer=keras.optimizers.
 ↪RMSprop(lr=2e-5), metrics=['accuracy'])

history = model.fit_generator(train_generator,
                              verbose=1,
                              validation_data = validation_generator,
                              validation_steps = 50,
                              steps_per_epoch = 100,
                              epochs=100)
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/training.py:1844: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/100
100/100 [==============================] - 276s 3s/step - loss: 0.6890 -
accuracy: 0.5373 - val_loss: 0.6529 - val_accuracy: 0.7763
Epoch 2/100
100/100 [==============================] - 201s 2s/step - loss: 0.6489 -
accuracy: 0.7233 - val_loss: 0.5981 - val_accuracy: 0.7812
Epoch 3/100
100/100 [==============================] - 159s 2s/step - loss: 0.6118 -
accuracy: 0.7323 - val_loss: 0.5467 - val_accuracy: 0.8062
Epoch 4/100
100/100 [==============================] - 136s 1s/step - loss: 0.5766 -
accuracy: 0.7674 - val_loss: 0.5316 - val_accuracy: 0.7862
Epoch 5/100
100/100 [==============================] - 108s 1s/step - loss: 0.5642 -
accuracy: 0.7522 - val_loss: 0.5036 - val_accuracy: 0.8012
Epoch 6/100
100/100 [==============================] - 94s 937ms/step - loss: 0.5409 -
accuracy: 0.7743 - val_loss: 0.5128 - val_accuracy: 0.7875
Epoch 7/100
100/100 [==============================] - 83s 836ms/step - loss: 0.5604 -
```

```
accuracy: 0.7540 - val_loss: 0.4742 - val_accuracy: 0.8150
Epoch 8/100
100/100 [==============================] - 79s 797ms/step - loss: 0.5173 -
accuracy: 0.7848 - val_loss: 0.4879 - val_accuracy: 0.8025
Epoch 9/100
100/100 [==============================] - 67s 672ms/step - loss: 0.5145 -
accuracy: 0.7825 - val_loss: 0.4795 - val_accuracy: 0.8075
Epoch 10/100
100/100 [==============================] - 57s 570ms/step - loss: 0.5076 -
accuracy: 0.7799 - val_loss: 0.4622 - val_accuracy: 0.8163
Epoch 11/100
100/100 [==============================] - 52s 526ms/step - loss: 0.5008 -
accuracy: 0.7891 - val_loss: 0.4537 - val_accuracy: 0.8138
Epoch 12/100
100/100 [==============================] - 46s 468ms/step - loss: 0.5316 -
accuracy: 0.7625 - val_loss: 0.4486 - val_accuracy: 0.8150
Epoch 13/100
100/100 [==============================] - 42s 422ms/step - loss: 0.5222 -
accuracy: 0.7813 - val_loss: 0.4307 - val_accuracy: 0.8238
Epoch 14/100
100/100 [==============================] - 36s 362ms/step - loss: 0.4991 -
accuracy: 0.7777 - val_loss: 0.4754 - val_accuracy: 0.7937
Epoch 15/100
100/100 [==============================] - 34s 339ms/step - loss: 0.5179 -
accuracy: 0.7815 - val_loss: 0.4561 - val_accuracy: 0.8037
Epoch 16/100
100/100 [==============================] - 36s 367ms/step - loss: 0.4965 -
accuracy: 0.7785 - val_loss: 0.4651 - val_accuracy: 0.8012
Epoch 17/100
100/100 [==============================] - 30s 296ms/step - loss: 0.5060 -
accuracy: 0.7738 - val_loss: 0.4454 - val_accuracy: 0.8075
Epoch 18/100
100/100 [==============================] - 27s 274ms/step - loss: 0.4761 -
accuracy: 0.7976 - val_loss: 0.4527 - val_accuracy: 0.8000
Epoch 19/100
100/100 [==============================] - 23s 229ms/step - loss: 0.4524 -
accuracy: 0.8081 - val_loss: 0.4229 - val_accuracy: 0.8138
Epoch 20/100
100/100 [==============================] - 24s 240ms/step - loss: 0.4728 -
accuracy: 0.7819 - val_loss: 0.4488 - val_accuracy: 0.7987
Epoch 21/100
100/100 [==============================] - 22s 221ms/step - loss: 0.4899 -
accuracy: 0.7970 - val_loss: 0.4530 - val_accuracy: 0.7925
Epoch 22/100
100/100 [==============================] - 22s 224ms/step - loss: 0.4601 -
accuracy: 0.8070 - val_loss: 0.4487 - val_accuracy: 0.7937
Epoch 23/100
100/100 [==============================] - 20s 202ms/step - loss: 0.4750 -
```

```
accuracy: 0.7960 - val_loss: 0.4142 - val_accuracy: 0.8238
Epoch 24/100
100/100 [==============================] - 19s 193ms/step - loss: 0.4740 -
accuracy: 0.7931 - val_loss: 0.4146 - val_accuracy: 0.8138
Epoch 25/100
100/100 [==============================] - 18s 181ms/step - loss: 0.4824 -
accuracy: 0.7896 - val_loss: 0.4282 - val_accuracy: 0.8062
Epoch 26/100
100/100 [==============================] - 16s 165ms/step - loss: 0.4937 -
accuracy: 0.7837 - val_loss: 0.4273 - val_accuracy: 0.7962
Epoch 27/100
100/100 [==============================] - 17s 170ms/step - loss: 0.4905 -
accuracy: 0.7797 - val_loss: 0.4257 - val_accuracy: 0.8000
Epoch 28/100
100/100 [==============================] - 15s 151ms/step - loss: 0.4559 -
accuracy: 0.8146 - val_loss: 0.4234 - val_accuracy: 0.8075
Epoch 29/100
100/100 [==============================] - 15s 153ms/step - loss: 0.4436 -
accuracy: 0.8071 - val_loss: 0.4022 - val_accuracy: 0.8150
Epoch 30/100
100/100 [==============================] - 15s 153ms/step - loss: 0.4491 -
accuracy: 0.8209 - val_loss: 0.3987 - val_accuracy: 0.8250
Epoch 31/100
100/100 [==============================] - 14s 141ms/step - loss: 0.4568 -
accuracy: 0.8036 - val_loss: 0.4159 - val_accuracy: 0.8050
Epoch 32/100
100/100 [==============================] - 15s 154ms/step - loss: 0.4514 -
accuracy: 0.8144 - val_loss: 0.3999 - val_accuracy: 0.8150
Epoch 33/100
100/100 [==============================] - 15s 146ms/step - loss: 0.4542 -
accuracy: 0.7934 - val_loss: 0.4345 - val_accuracy: 0.7912
Epoch 34/100
100/100 [==============================] - 14s 139ms/step - loss: 0.4420 -
accuracy: 0.8059 - val_loss: 0.4010 - val_accuracy: 0.8050
Epoch 35/100
100/100 [==============================] - 13s 135ms/step - loss: 0.4671 -
accuracy: 0.7939 - val_loss: 0.3921 - val_accuracy: 0.8112
Epoch 36/100
100/100 [==============================] - 13s 134ms/step - loss: 0.4680 -
accuracy: 0.8025 - val_loss: 0.4180 - val_accuracy: 0.7987
Epoch 37/100
100/100 [==============================] - 14s 143ms/step - loss: 0.4352 -
accuracy: 0.8174 - val_loss: 0.4195 - val_accuracy: 0.8025
Epoch 38/100
100/100 [==============================] - 13s 131ms/step - loss: 0.4571 -
accuracy: 0.8045 - val_loss: 0.4241 - val_accuracy: 0.7975
Epoch 39/100
100/100 [==============================] - 14s 135ms/step - loss: 0.4260 -
```

```
accuracy: 0.8114 - val_loss: 0.4070 - val_accuracy: 0.8075
Epoch 40/100
100/100 [==============================] - 13s 130ms/step - loss: 0.4808 -
accuracy: 0.7846 - val_loss: 0.4241 - val_accuracy: 0.8000
Epoch 41/100
100/100 [==============================] - 14s 139ms/step - loss: 0.4409 -
accuracy: 0.8069 - val_loss: 0.3889 - val_accuracy: 0.8163
Epoch 42/100
100/100 [==============================] - 14s 137ms/step - loss: 0.4367 -
accuracy: 0.8058 - val_loss: 0.3809 - val_accuracy: 0.8188
Epoch 43/100
100/100 [==============================] - 13s 132ms/step - loss: 0.4669 -
accuracy: 0.8078 - val_loss: 0.4025 - val_accuracy: 0.8125
Epoch 44/100
100/100 [==============================] - 13s 132ms/step - loss: 0.4669 -
accuracy: 0.8054 - val_loss: 0.4023 - val_accuracy: 0.8112
Epoch 45/100
100/100 [==============================] - 13s 132ms/step - loss: 0.4760 -
accuracy: 0.7955 - val_loss: 0.4066 - val_accuracy: 0.8037
Epoch 46/100
100/100 [==============================] - 13s 130ms/step - loss: 0.4268 -
accuracy: 0.8358 - val_loss: 0.3911 - val_accuracy: 0.8163
Epoch 47/100
100/100 [==============================] - 13s 129ms/step - loss: 0.4409 -
accuracy: 0.8079 - val_loss: 0.3937 - val_accuracy: 0.8112
Epoch 48/100
100/100 [==============================] - 13s 126ms/step - loss: 0.4100 -
accuracy: 0.8236 - val_loss: 0.3953 - val_accuracy: 0.8037
Epoch 49/100
100/100 [==============================] - 13s 129ms/step - loss: 0.4335 -
accuracy: 0.8177 - val_loss: 0.3855 - val_accuracy: 0.8112
Epoch 50/100
100/100 [==============================] - 13s 129ms/step - loss: 0.4276 -
accuracy: 0.8094 - val_loss: 0.3983 - val_accuracy: 0.8100
Epoch 51/100
100/100 [==============================] - 13s 127ms/step - loss: 0.4242 -
accuracy: 0.8261 - val_loss: 0.3707 - val_accuracy: 0.8350
Epoch 52/100
100/100 [==============================] - 13s 131ms/step - loss: 0.4437 -
accuracy: 0.8093 - val_loss: 0.3512 - val_accuracy: 0.8363
Epoch 53/100
100/100 [==============================] - 13s 129ms/step - loss: 0.4344 -
accuracy: 0.8170 - val_loss: 0.3816 - val_accuracy: 0.8200
Epoch 54/100
100/100 [==============================] - 13s 129ms/step - loss: 0.4515 -
accuracy: 0.8189 - val_loss: 0.3865 - val_accuracy: 0.8100
Epoch 55/100
100/100 [==============================] - 13s 128ms/step - loss: 0.4319 -
```

```
accuracy: 0.8108 - val_loss: 0.3788 - val_accuracy: 0.8275
Epoch 56/100
100/100 [==============================] - 13s 128ms/step - loss: 0.3850 -
accuracy: 0.8506 - val_loss: 0.3854 - val_accuracy: 0.8225
Epoch 57/100
100/100 [==============================] - 13s 131ms/step - loss: 0.4170 -
accuracy: 0.8200 - val_loss: 0.3973 - val_accuracy: 0.8050
Epoch 58/100
100/100 [==============================] - 13s 132ms/step - loss: 0.4137 -
accuracy: 0.8353 - val_loss: 0.3899 - val_accuracy: 0.8188
Epoch 59/100
100/100 [==============================] - 13s 135ms/step - loss: 0.4338 -
accuracy: 0.8084 - val_loss: 0.3936 - val_accuracy: 0.8062
Epoch 60/100
100/100 [==============================] - 13s 132ms/step - loss: 0.4276 -
accuracy: 0.8111 - val_loss: 0.3703 - val_accuracy: 0.8313
Epoch 61/100
100/100 [==============================] - 13s 130ms/step - loss: 0.4217 -
accuracy: 0.8160 - val_loss: 0.3875 - val_accuracy: 0.8125
Epoch 62/100
100/100 [==============================] - 13s 130ms/step - loss: 0.4092 -
accuracy: 0.8212 - val_loss: 0.3977 - val_accuracy: 0.8075
Epoch 63/100
100/100 [==============================] - 13s 129ms/step - loss: 0.4262 -
accuracy: 0.8089 - val_loss: 0.3542 - val_accuracy: 0.8300
Epoch 64/100
100/100 [==============================] - 13s 128ms/step - loss: 0.4154 -
accuracy: 0.8123 - val_loss: 0.3729 - val_accuracy: 0.8225
Epoch 65/100
100/100 [==============================] - 13s 128ms/step - loss: 0.4004 -
accuracy: 0.8287 - val_loss: 0.3783 - val_accuracy: 0.8175
Epoch 66/100
100/100 [==============================] - 13s 127ms/step - loss: 0.4145 -
accuracy: 0.8154 - val_loss: 0.3712 - val_accuracy: 0.8225
Epoch 67/100
100/100 [==============================] - 13s 129ms/step - loss: 0.3948 -
accuracy: 0.8319 - val_loss: 0.3903 - val_accuracy: 0.8112
Epoch 68/100
100/100 [==============================] - 13s 127ms/step - loss: 0.4205 -
accuracy: 0.8192 - val_loss: 0.3902 - val_accuracy: 0.8100
Epoch 69/100
100/100 [==============================] - 13s 130ms/step - loss: 0.4309 -
accuracy: 0.8077 - val_loss: 0.3890 - val_accuracy: 0.8250
Epoch 70/100
100/100 [==============================] - 13s 127ms/step - loss: 0.3987 -
accuracy: 0.8368 - val_loss: 0.3908 - val_accuracy: 0.8188
Epoch 71/100
100/100 [==============================] - 13s 128ms/step - loss: 0.4412 -
```

```
accuracy: 0.8175 - val_loss: 0.3894 - val_accuracy: 0.8125
Epoch 72/100
100/100 [==============================] - 13s 127ms/step - loss: 0.3787 -
accuracy: 0.8486 - val_loss: 0.3717 - val_accuracy: 0.8325
Epoch 73/100
100/100 [==============================] - 13s 133ms/step - loss: 0.3967 -
accuracy: 0.8439 - val_loss: 0.3632 - val_accuracy: 0.8325
Epoch 74/100
100/100 [==============================] - 13s 126ms/step - loss: 0.4163 -
accuracy: 0.8212 - val_loss: 0.3661 - val_accuracy: 0.8413
Epoch 75/100
100/100 [==============================] - 13s 129ms/step - loss: 0.3889 -
accuracy: 0.8399 - val_loss: 0.3919 - val_accuracy: 0.8188
Epoch 76/100
100/100 [==============================] - 13s 129ms/step - loss: 0.4087 -
accuracy: 0.8234 - val_loss: 0.3775 - val_accuracy: 0.8263
Epoch 77/100
100/100 [==============================] - 13s 127ms/step - loss: 0.3976 -
accuracy: 0.8269 - val_loss: 0.3535 - val_accuracy: 0.8487
Epoch 78/100
100/100 [==============================] - 13s 129ms/step - loss: 0.4031 -
accuracy: 0.8400 - val_loss: 0.3668 - val_accuracy: 0.8375
Epoch 79/100
100/100 [==============================] - 13s 128ms/step - loss: 0.3892 -
accuracy: 0.8558 - val_loss: 0.3938 - val_accuracy: 0.8313
Epoch 80/100
100/100 [==============================] - 13s 129ms/step - loss: 0.4766 -
accuracy: 0.8052 - val_loss: 0.3904 - val_accuracy: 0.8363
Epoch 81/100
100/100 [==============================] - 13s 127ms/step - loss: 0.4290 -
accuracy: 0.8233 - val_loss: 0.3960 - val_accuracy: 0.8188
Epoch 82/100
100/100 [==============================] - 13s 131ms/step - loss: 0.4048 -
accuracy: 0.8300 - val_loss: 0.3793 - val_accuracy: 0.8250
Epoch 83/100
100/100 [==============================] - 13s 133ms/step - loss: 0.4410 -
accuracy: 0.8075 - val_loss: 0.3738 - val_accuracy: 0.8338
Epoch 84/100
100/100 [==============================] - 13s 128ms/step - loss: 0.4035 -
accuracy: 0.8274 - val_loss: 0.3644 - val_accuracy: 0.8338
Epoch 85/100
100/100 [==============================] - 13s 131ms/step - loss: 0.4347 -
accuracy: 0.8254 - val_loss: 0.3560 - val_accuracy: 0.8338
Epoch 86/100
100/100 [==============================] - 13s 131ms/step - loss: 0.4049 -
accuracy: 0.8267 - val_loss: 0.3906 - val_accuracy: 0.8325
Epoch 87/100
100/100 [==============================] - 13s 132ms/step - loss: 0.4244 -
```

```
accuracy: 0.8292 - val_loss: 0.3561 - val_accuracy: 0.8413
Epoch 88/100
100/100 [==============================] - 13s 133ms/step - loss: 0.4313 -
accuracy: 0.8203 - val_loss: 0.3724 - val_accuracy: 0.8350
Epoch 89/100
100/100 [==============================] - 13s 130ms/step - loss: 0.3803 -
accuracy: 0.8537 - val_loss: 0.3748 - val_accuracy: 0.8288
Epoch 90/100
100/100 [==============================] - 13s 133ms/step - loss: 0.4294 -
accuracy: 0.8141 - val_loss: 0.4398 - val_accuracy: 0.8062
Epoch 91/100
100/100 [==============================] - 13s 130ms/step - loss: 0.4085 -
accuracy: 0.8478 - val_loss: 0.3884 - val_accuracy: 0.8288
Epoch 92/100
100/100 [==============================] - 13s 133ms/step - loss: 0.4093 -
accuracy: 0.8285 - val_loss: 0.4248 - val_accuracy: 0.8100
Epoch 93/100
100/100 [==============================] - 13s 131ms/step - loss: 0.3858 -
accuracy: 0.8391 - val_loss: 0.3741 - val_accuracy: 0.8300
Epoch 94/100
100/100 [==============================] - 13s 131ms/step - loss: 0.3946 -
accuracy: 0.8394 - val_loss: 0.3700 - val_accuracy: 0.8375
Epoch 95/100
100/100 [==============================] - 13s 133ms/step - loss: 0.4185 -
accuracy: 0.8393 - val_loss: 0.3699 - val_accuracy: 0.8425
Epoch 96/100
100/100 [==============================] - 13s 132ms/step - loss: 0.4016 -
accuracy: 0.8348 - val_loss: 0.3756 - val_accuracy: 0.8288
Epoch 97/100
100/100 [==============================] - 13s 134ms/step - loss: 0.4313 -
accuracy: 0.8232 - val_loss: 0.4330 - val_accuracy: 0.8200
Epoch 98/100
100/100 [==============================] - 13s 132ms/step - loss: 0.3873 -
accuracy: 0.8443 - val_loss: 0.3690 - val_accuracy: 0.8388
Epoch 99/100
100/100 [==============================] - 13s 131ms/step - loss: 0.4405 -
accuracy: 0.8141 - val_loss: 0.3628 - val_accuracy: 0.8438
Epoch 100/100
100/100 [==============================] - 13s 133ms/step - loss: 0.3970 -
accuracy: 0.8472 - val_loss: 0.3665 - val_accuracy: 0.8400
```

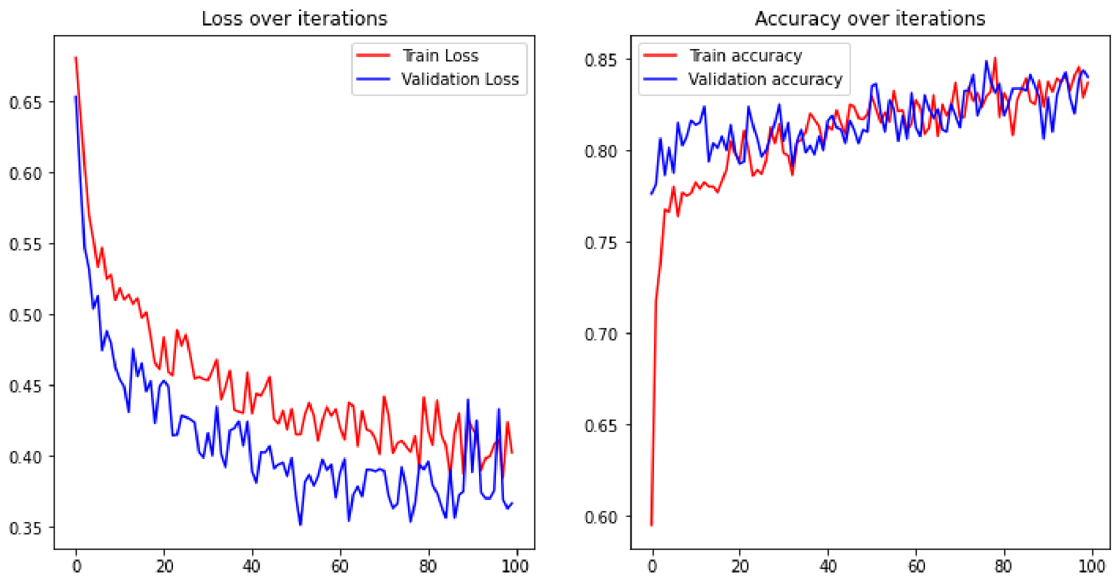
[ ]:
```
history.history.keys()
```

[ ]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[ ]:
```
n = len(history.history["loss"])

fig = plt.figure(figsize=(12,6))
```

```
ax = fig.add_subplot(1,2,1)
ax.plot(range(n), (history.history["loss"]),'r', label="Train Loss")
ax.plot(range(n), (history.history["val_loss"]),'b', label="Validation Loss")
ax.legend()
ax.set_title('Loss over iterations')

ax = fig.add_subplot(1,2,2)
ax.plot(range(n), (history.history["accuracy"]),'r', label="Train accuracy")
ax.plot(range(n), (history.history["val_accuracy"]),'b', label="Validation␣
 ↪accuracy")
ax.legend()

ax.set_title('Accuracy over iterations')
```

[ ]:
```
Text(0.5, 1.0, 'Accuracy over iterations')
```



[ ]:
```
model.save('/content/gdrive/MyDrive/covid19/output/covid.h5')
```

[ ]:
```
model = keras.models.load_model('/content/gdrive/MyDrive/covid19/output/covid.
 ↪h5')
```

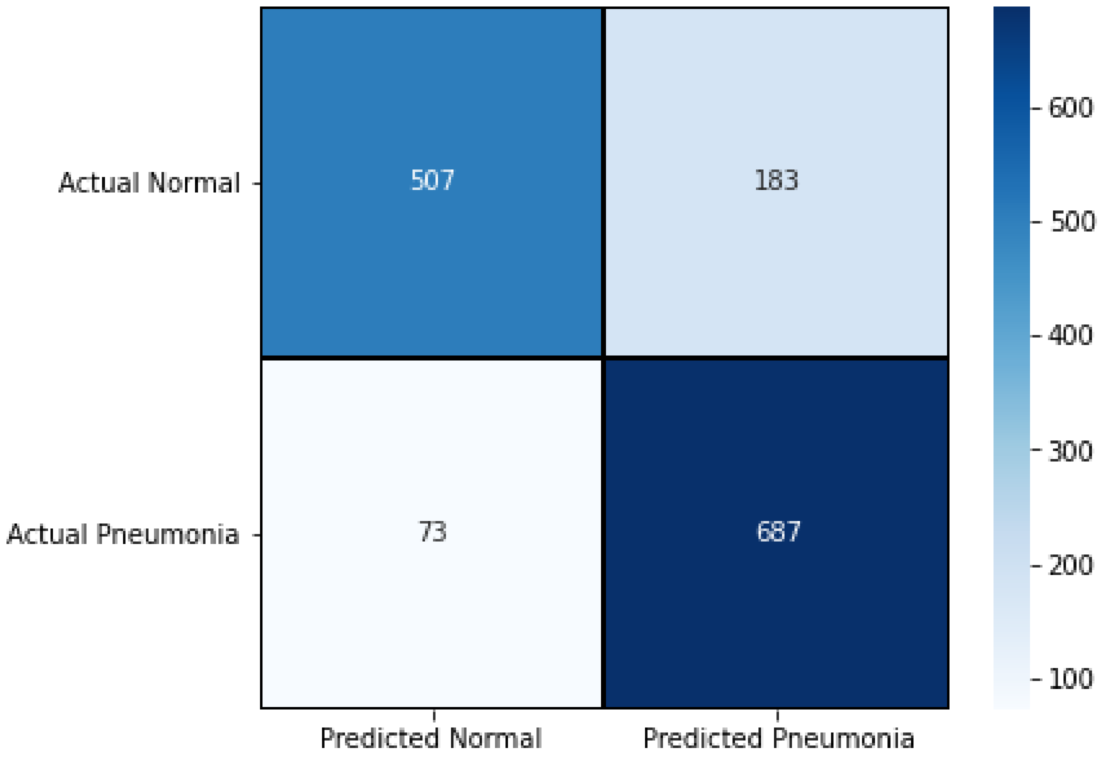
[ ]:
```
predictions = model.predict_classes(test_generator)
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
```

```
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

[ ]:
```
cm = confusion_matrix(test_generator.classes, predictions)
cm_pd = pd.DataFrame(cm , index = ['0','1'] , columns = ['0','1'])
```

[ ]:
```
plt.figure(figsize = (6,5))
sns.heatmap(cm_pd,cmap= "Blues", linecolor = 'black' ,
            linewidth = 1 , annot = True, fmt='',
            xticklabels = ['Predicted Normal', 'Predicted Pneumonia'],
            yticklabels = ['Actual Normal', 'Actual Pneumonia'])
plt.yticks(rotation=0)
plt.show()
```



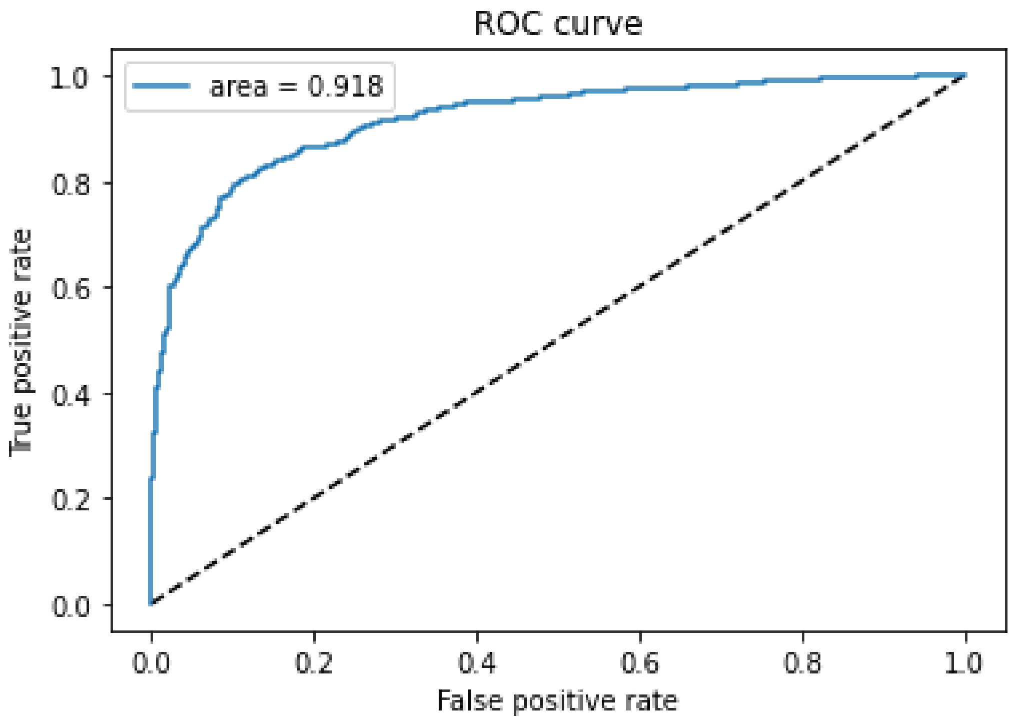
[ ]:
```
# make a prediction
y_pred_keras = model.predict_generator(test_generator, validation_generator.
 ↪samples // validation_generator.batch_size+1) #(test_gen, steps=len(df_val),␣
 ↪verbose=1)
fpr_keras, tpr_keras, thresholds_keras = roc_curve(test_generator.classes,␣
 ↪y_pred_keras)
```

```
auc_keras = auc(fpr_keras, tpr_keras)
auc_keras
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/training.py:1905: UserWarning:
`Model.predict_generator` is deprecated and will be removed in a future version.
Please use `Model.predict`, which supports generators.
  warnings.warn('`Model.predict_generator` is deprecated and '
```

```
[ ]: 0.917509534706331
```

```
[ ]: plt.figure(1)
     plt.plot([0, 1], [0, 1], 'k--')
     plt.plot(fpr_keras, tpr_keras, label='area = {:.3f}'.format(auc_keras))
     plt.xlabel('False positive rate')
     plt.ylabel('True positive rate')
     plt.title('ROC curve')
     plt.legend(loc='best')
     plt.show()
```



1.6 Third model

[ ]:
```
conv_base = VGG16(weights='imagenet',
                  include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
```

[ ]:
```
from tensorflow.keras.applications.vgg16 import VGG16

base = VGG16(weights = 'imagenet', include_top = False, input_shape = (128,␣
 ↪128, 3))

for layer in base.layers:
    layer.trainable = False

vgg_model = Sequential()
vgg_model.add(base)

vgg_model.add(Dense(256, activation='relu'))
vgg_model.add(Dropout(0.5))
vgg_model.add(Flatten())
vgg_model.add(Dense(128, activation='relu'))
vgg_model.add(Dropout(0.5))
vgg_model.add(Dense(1, activation='sigmoid'))

vgg_model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 4, 4, 512)         14714688
_________________________________________________________________
dense_13 (Dense)             (None, 4, 4, 256)         131328
_________________________________________________________________
dropout_9 (Dropout)          (None, 4, 4, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_14 (Dense)             (None, 128)               524416
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 129
=================================================================
Total params: 15,370,561
Trainable params: 655,873
```

```
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
[ ]: #Image augmentation
     train_datagen = ImageDataGenerator(rescale = 1./255,
                                        horizontal_flip=True, # randomly flipping␣
      ↪half the images horizontally
                                        rotation_range=5, # in value of degrees, a␣
      ↪range within which to randomly rotate pictures
                                        width_shift_range=0.05,
                                        height_shift_range=0.05,
                                        shear_range=0.05, # randomly applying shear␣
      ↪transformations
                                        zoom_range=0.05, # randomly zooming images
                                        )

     test_datagen_vgg16 = ImageDataGenerator(rescale = 1./255) # Data augmentation␣
      ↪techniques ONLY for the train_datagen

     #Training data generator
     train_generator_vgg16 = train_datagen_vgg16.flow_from_directory(TRAIN_PATH,
                                                           # target_size=(IMG_HEIGHT,␣
      ↪IMG_WIDTH),
                                                           batch_size=BATCH_SIZE,
                                                           shuffle=True,
                                                           target_size=(128, 128),
                                                           class_mode='binary')

     #Validation data generator
     validation_generator_vgg16 = test_datagen_vgg16.flow_from_directory(VAL_PATH,
                                                                 # ␣
      ↪target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                                 batch_size=BATCH_SIZE,
                                                                 class_mode='binary',
                                                                 shuffle=True,
                                                                 target_size=(128, 128)
                                                                 )

     test_generator_vgg16 = test_datagen_vgg16.flow_from_directory(TEST_PATH,
                                                                 # ␣
      ↪target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                                 batch_size=BATCH_SIZE,
                                                                 class_mode='binary',
                                                                 shuffle=False,
                                                                 target_size=(128, 128)
                                                                 )
```

```
Found 11588 images belonging to 2 classes.
Found 1448 images belonging to 2 classes.
Found 1450 images belonging to 2 classes.
```

[ ]:
```
vgg_model.compile(loss='binary_crossentropy', optimizer=keras.optimizers.
 ↪RMSprop(lr=2e-5), metrics=['accuracy'])

history = vgg_model.fit_generator(train_generator_vgg16,
                              verbose=1,
                              validation_data = validation_generator_vgg16,
                              validation_steps = 50,
                              steps_per_epoch = 100,
                              epochs=100)
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/training.py:1844: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/100
100/100 [==============================] - 558s 5s/step - loss: 0.8247 -
accuracy: 0.5100 - val_loss: 0.5557 - val_accuracy: 0.8037

Exception ignored in: <function IteratorResourceDeleter.__del__ at
0x7fd48af98f80>
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-
packages/tensorflow/python/data/ops/iterator_ops.py", line 535, in __del__
    handle=self._handle, deleter=self._deleter)
  File "/usr/local/lib/python3.7/dist-
packages/tensorflow/python/ops/gen_dataset_ops.py", line 1264, in
delete_iterator
    _ctx, "DeleteIterator", name, handle, deleter)
KeyboardInterrupt:

Epoch 2/100
100/100 [==============================] - 383s 4s/step - loss: 0.6894 -
accuracy: 0.5923 - val_loss: 0.5230 - val_accuracy: 0.7713
Epoch 3/100
100/100 [==============================] - 317s 3s/step - loss: 0.6024 -
accuracy: 0.7024 - val_loss: 0.4860 - val_accuracy: 0.7788
Epoch 4/100
100/100 [==============================] - 248s 2s/step - loss: 0.5423 -
accuracy: 0.7443 - val_loss: 0.4875 - val_accuracy: 0.7750
Epoch 5/100
100/100 [==============================] - 202s 2s/step - loss: 0.5710 -
accuracy: 0.7272 - val_loss: 0.4718 - val_accuracy: 0.7788
Epoch 6/100
100/100 [==============================] - 178s 2s/step - loss: 0.4887 -
```

```
accuracy: 0.7863 - val_loss: 0.4601 - val_accuracy: 0.7862
Epoch 7/100
100/100 [==============================] - 158s 2s/step - loss: 0.5245 -
accuracy: 0.7806 - val_loss: 0.4302 - val_accuracy: 0.7962
Epoch 8/100
100/100 [==============================] - 140s 1s/step - loss: 0.4813 -
accuracy: 0.7988 - val_loss: 0.4353 - val_accuracy: 0.7987
Epoch 9/100
100/100 [==============================] - 116s 1s/step - loss: 0.5063 -
accuracy: 0.7710 - val_loss: 0.4487 - val_accuracy: 0.7850
Epoch 10/100
100/100 [==============================] - 101s 1s/step - loss: 0.4569 -
accuracy: 0.8042 - val_loss: 0.4550 - val_accuracy: 0.7825
Epoch 11/100
100/100 [==============================] - 96s 957ms/step - loss: 0.4819 -
accuracy: 0.7986 - val_loss: 0.4532 - val_accuracy: 0.7950
Epoch 12/100
100/100 [==============================] - 85s 851ms/step - loss: 0.4295 -
accuracy: 0.8071 - val_loss: 0.4085 - val_accuracy: 0.8200
Epoch 13/100
100/100 [==============================] - 72s 720ms/step - loss: 0.4710 -
accuracy: 0.7875 - val_loss: 0.4241 - val_accuracy: 0.8087
Epoch 14/100
100/100 [==============================] - 62s 622ms/step - loss: 0.4240 -
accuracy: 0.8234 - val_loss: 0.4054 - val_accuracy: 0.8087
Epoch 15/100
100/100 [==============================] - 57s 569ms/step - loss: 0.4663 -
accuracy: 0.8004 - val_loss: 0.4318 - val_accuracy: 0.8350
Epoch 16/100
100/100 [==============================] - 55s 552ms/step - loss: 0.4157 -
accuracy: 0.8194 - val_loss: 0.3830 - val_accuracy: 0.8425
Epoch 17/100
100/100 [==============================] - 57s 566ms/step - loss: 0.4577 -
accuracy: 0.8012 - val_loss: 0.3874 - val_accuracy: 0.8375
Epoch 18/100
100/100 [==============================] - 43s 426ms/step - loss: 0.3979 -
accuracy: 0.8488 - val_loss: 0.3968 - val_accuracy: 0.8350
Epoch 19/100
100/100 [==============================] - 44s 439ms/step - loss: 0.4094 -
accuracy: 0.8154 - val_loss: 0.4018 - val_accuracy: 0.8475
Epoch 20/100
100/100 [==============================] - 41s 409ms/step - loss: 0.3908 -
accuracy: 0.8406 - val_loss: 0.4270 - val_accuracy: 0.8400
Epoch 21/100
100/100 [==============================] - 36s 358ms/step - loss: 0.4099 -
accuracy: 0.8204 - val_loss: 0.3826 - val_accuracy: 0.8400
Epoch 22/100
100/100 [==============================] - 31s 308ms/step - loss: 0.4016 -
```

```
accuracy: 0.8187 - val_loss: 0.4353 - val_accuracy: 0.8325
Epoch 23/100
100/100 [==============================] - 29s 293ms/step - loss: 0.4175 -
accuracy: 0.8130 - val_loss: 0.3590 - val_accuracy: 0.8500
Epoch 24/100
100/100 [==============================] - 30s 302ms/step - loss: 0.3886 -
accuracy: 0.8400 - val_loss: 0.4095 - val_accuracy: 0.8500
Epoch 25/100
100/100 [==============================] - 25s 250ms/step - loss: 0.3593 -
accuracy: 0.8418 - val_loss: 0.3827 - val_accuracy: 0.8562
Epoch 26/100
100/100 [==============================] - 25s 252ms/step - loss: 0.3857 -
accuracy: 0.8223 - val_loss: 0.3384 - val_accuracy: 0.8600
Epoch 27/100
100/100 [==============================] - 23s 231ms/step - loss: 0.3760 -
accuracy: 0.8414 - val_loss: 0.3953 - val_accuracy: 0.8537
Epoch 28/100
100/100 [==============================] - 23s 228ms/step - loss: 0.3884 -
accuracy: 0.8200 - val_loss: 0.3442 - val_accuracy: 0.8725
Epoch 29/100
100/100 [==============================] - 23s 229ms/step - loss: 0.3824 -
accuracy: 0.8410 - val_loss: 0.4027 - val_accuracy: 0.8487
Epoch 30/100
100/100 [==============================] - 23s 221ms/step - loss: 0.3999 -
accuracy: 0.8222 - val_loss: 0.3644 - val_accuracy: 0.8700
Epoch 31/100
100/100 [==============================] - 20s 201ms/step - loss: 0.3623 -
accuracy: 0.8472 - val_loss: 0.3621 - val_accuracy: 0.8612
Epoch 32/100
100/100 [==============================] - 21s 206ms/step - loss: 0.3847 -
accuracy: 0.8339 - val_loss: 0.3556 - val_accuracy: 0.8662
Epoch 33/100
100/100 [==============================] - 18s 184ms/step - loss: 0.3781 -
accuracy: 0.8449 - val_loss: 0.3587 - val_accuracy: 0.8612
Epoch 34/100
100/100 [==============================] - 20s 196ms/step - loss: 0.3450 -
accuracy: 0.8575 - val_loss: 0.3911 - val_accuracy: 0.8662
Epoch 35/100
100/100 [==============================] - 18s 184ms/step - loss: 0.3510 -
accuracy: 0.8568 - val_loss: 0.3748 - val_accuracy: 0.8612
Epoch 36/100
100/100 [==============================] - 19s 190ms/step - loss: 0.4073 -
accuracy: 0.8231 - val_loss: 0.3867 - val_accuracy: 0.8600
Epoch 37/100
100/100 [==============================] - 18s 178ms/step - loss: 0.3732 -
accuracy: 0.8464 - val_loss: 0.3370 - val_accuracy: 0.8700
Epoch 38/100
100/100 [==============================] - 18s 182ms/step - loss: 0.3380 -
```

```
accuracy: 0.8501 - val_loss: 0.3366 - val_accuracy: 0.8712
Epoch 39/100
100/100 [==============================] - 18s 177ms/step - loss: 0.3768 -
accuracy: 0.8509 - val_loss: 0.3599 - val_accuracy: 0.8637
Epoch 40/100
100/100 [==============================] - 18s 180ms/step - loss: 0.3323 -
accuracy: 0.8648 - val_loss: 0.3101 - val_accuracy: 0.8813
Epoch 41/100
100/100 [==============================] - 18s 180ms/step - loss: 0.3492 -
accuracy: 0.8550 - val_loss: 0.3570 - val_accuracy: 0.8737
Epoch 42/100
100/100 [==============================] - 18s 179ms/step - loss: 0.3708 -
accuracy: 0.8409 - val_loss: 0.3460 - val_accuracy: 0.8637
Epoch 43/100
100/100 [==============================] - 18s 177ms/step - loss: 0.2933 -
accuracy: 0.8881 - val_loss: 0.3288 - val_accuracy: 0.8788
Epoch 44/100
100/100 [==============================] - 17s 173ms/step - loss: 0.3954 -
accuracy: 0.8334 - val_loss: 0.3237 - val_accuracy: 0.8775
Epoch 45/100
100/100 [==============================] - 18s 180ms/step - loss: 0.3656 -
accuracy: 0.8606 - val_loss: 0.3590 - val_accuracy: 0.8687
Epoch 46/100
100/100 [==============================] - 17s 173ms/step - loss: 0.3213 -
accuracy: 0.8779 - val_loss: 0.3520 - val_accuracy: 0.8712
Epoch 47/100
100/100 [==============================] - 18s 178ms/step - loss: 0.3594 -
accuracy: 0.8506 - val_loss: 0.3400 - val_accuracy: 0.8700
Epoch 48/100
100/100 [==============================] - 17s 171ms/step - loss: 0.3460 -
accuracy: 0.8568 - val_loss: 0.3309 - val_accuracy: 0.8725
Epoch 49/100
100/100 [==============================] - 18s 175ms/step - loss: 0.2957 -
accuracy: 0.8807 - val_loss: 0.3538 - val_accuracy: 0.8687
Epoch 50/100
100/100 [==============================] - 17s 170ms/step - loss: 0.3219 -
accuracy: 0.8694 - val_loss: 0.3621 - val_accuracy: 0.8625
Epoch 51/100
100/100 [==============================] - 17s 171ms/step - loss: 0.3348 -
accuracy: 0.8618 - val_loss: 0.3289 - val_accuracy: 0.8763
Epoch 52/100
100/100 [==============================] - 17s 174ms/step - loss: 0.3252 -
accuracy: 0.8703 - val_loss: 0.3570 - val_accuracy: 0.8587
Epoch 53/100
100/100 [==============================] - 17s 172ms/step - loss: 0.3257 -
accuracy: 0.8698 - val_loss: 0.3538 - val_accuracy: 0.8587
Epoch 54/100
100/100 [==============================] - 17s 170ms/step - loss: 0.3367 -
```

```
accuracy: 0.8700 - val_loss: 0.2926 - val_accuracy: 0.8950
Epoch 55/100
100/100 [==============================] - 17s 170ms/step - loss: 0.3422 -
accuracy: 0.8496 - val_loss: 0.3081 - val_accuracy: 0.8813
Epoch 56/100
100/100 [==============================] - 17s 174ms/step - loss: 0.2942 -
accuracy: 0.8776 - val_loss: 0.3152 - val_accuracy: 0.8788
Epoch 57/100
100/100 [==============================] - 17s 175ms/step - loss: 0.3459 -
accuracy: 0.8438 - val_loss: 0.3280 - val_accuracy: 0.8737
Epoch 58/100
100/100 [==============================] - 17s 170ms/step - loss: 0.2984 -
accuracy: 0.8797 - val_loss: 0.3212 - val_accuracy: 0.8763
Epoch 59/100
100/100 [==============================] - 17s 171ms/step - loss: 0.3468 -
accuracy: 0.8530 - val_loss: 0.3155 - val_accuracy: 0.8687
Epoch 60/100
100/100 [==============================] - 17s 170ms/step - loss: 0.2994 -
accuracy: 0.8849 - val_loss: 0.3333 - val_accuracy: 0.8712
Epoch 61/100
100/100 [==============================] - 17s 170ms/step - loss: 0.3160 -
accuracy: 0.8701 - val_loss: 0.3419 - val_accuracy: 0.8775
Epoch 62/100
100/100 [==============================] - 17s 171ms/step - loss: 0.3190 -
accuracy: 0.8763 - val_loss: 0.2947 - val_accuracy: 0.8712
Epoch 63/100
100/100 [==============================] - 17s 172ms/step - loss: 0.3298 -
accuracy: 0.8578 - val_loss: 0.2934 - val_accuracy: 0.8825
Epoch 64/100
100/100 [==============================] - 17s 174ms/step - loss: 0.2888 -
accuracy: 0.8757 - val_loss: 0.2937 - val_accuracy: 0.8825
Epoch 65/100
100/100 [==============================] - 17s 174ms/step - loss: 0.3235 -
accuracy: 0.8688 - val_loss: 0.3314 - val_accuracy: 0.8662
Epoch 66/100
100/100 [==============================] - 17s 175ms/step - loss: 0.3382 -
accuracy: 0.8563 - val_loss: 0.3147 - val_accuracy: 0.8700
Epoch 67/100
100/100 [==============================] - 17s 175ms/step - loss: 0.3098 -
accuracy: 0.8765 - val_loss: 0.3406 - val_accuracy: 0.8650
Epoch 68/100
100/100 [==============================] - 17s 173ms/step - loss: 0.3331 -
accuracy: 0.8806 - val_loss: 0.3327 - val_accuracy: 0.8700
Epoch 69/100
100/100 [==============================] - 18s 176ms/step - loss: 0.2865 -
accuracy: 0.8941 - val_loss: 0.2867 - val_accuracy: 0.8800
Epoch 70/100
100/100 [==============================] - 17s 174ms/step - loss: 0.3338 -
```

```
accuracy: 0.8658 - val_loss: 0.3069 - val_accuracy: 0.8838
Epoch 71/100
100/100 [==============================] - 17s 175ms/step - loss: 0.3247 -
accuracy: 0.8631 - val_loss: 0.3132 - val_accuracy: 0.8737
Epoch 72/100
100/100 [==============================] - 18s 177ms/step - loss: 0.3245 -
accuracy: 0.8717 - val_loss: 0.3085 - val_accuracy: 0.8788
Epoch 73/100
100/100 [==============================] - 17s 175ms/step - loss: 0.3163 -
accuracy: 0.8640 - val_loss: 0.3112 - val_accuracy: 0.8750
Epoch 74/100
100/100 [==============================] - 18s 180ms/step - loss: 0.2931 -
accuracy: 0.8817 - val_loss: 0.2989 - val_accuracy: 0.8838
Epoch 75/100
100/100 [==============================] - 18s 182ms/step - loss: 0.3320 -
accuracy: 0.8621 - val_loss: 0.3046 - val_accuracy: 0.8813
Epoch 76/100
100/100 [==============================] - 17s 174ms/step - loss: 0.3119 -
accuracy: 0.8748 - val_loss: 0.3034 - val_accuracy: 0.8838
Epoch 77/100
100/100 [==============================] - 18s 178ms/step - loss: 0.3025 -
accuracy: 0.8772 - val_loss: 0.3107 - val_accuracy: 0.8712
Epoch 78/100
100/100 [==============================] - 17s 174ms/step - loss: 0.2704 -
accuracy: 0.8919 - val_loss: 0.3252 - val_accuracy: 0.8675
Epoch 79/100
100/100 [==============================] - 18s 178ms/step - loss: 0.2833 -
accuracy: 0.8822 - val_loss: 0.3126 - val_accuracy: 0.8888
Epoch 80/100
100/100 [==============================] - 18s 179ms/step - loss: 0.3159 -
accuracy: 0.8655 - val_loss: 0.2938 - val_accuracy: 0.8800
Epoch 81/100
100/100 [==============================] - 18s 178ms/step - loss: 0.3082 -
accuracy: 0.8734 - val_loss: 0.3166 - val_accuracy: 0.8813
Epoch 82/100
100/100 [==============================] - 18s 176ms/step - loss: 0.2965 -
accuracy: 0.8902 - val_loss: 0.3123 - val_accuracy: 0.8838
Epoch 83/100
100/100 [==============================] - 18s 177ms/step - loss: 0.3031 -
accuracy: 0.8734 - val_loss: 0.3165 - val_accuracy: 0.8813
Epoch 84/100
100/100 [==============================] - 18s 179ms/step - loss: 0.2601 -
accuracy: 0.8871 - val_loss: 0.3450 - val_accuracy: 0.8625
Epoch 85/100
100/100 [==============================] - 18s 177ms/step - loss: 0.2769 -
accuracy: 0.8952 - val_loss: 0.2934 - val_accuracy: 0.8838
Epoch 86/100
100/100 [==============================] - 18s 180ms/step - loss: 0.2848 -
```

```
accuracy: 0.8708 - val_loss: 0.2915 - val_accuracy: 0.8888
Epoch 87/100
100/100 [==============================] - 18s 179ms/step - loss: 0.2985 -
accuracy: 0.8806 - val_loss: 0.2557 - val_accuracy: 0.9025
Epoch 88/100
100/100 [==============================] - 18s 179ms/step - loss: 0.3246 -
accuracy: 0.8807 - val_loss: 0.2989 - val_accuracy: 0.8850
Epoch 89/100
100/100 [==============================] - 18s 179ms/step - loss: 0.2485 -
accuracy: 0.8928 - val_loss: 0.3201 - val_accuracy: 0.8725
Epoch 90/100
100/100 [==============================] - 18s 178ms/step - loss: 0.3134 -
accuracy: 0.8709 - val_loss: 0.3004 - val_accuracy: 0.8813
Epoch 91/100
100/100 [==============================] - 18s 179ms/step - loss: 0.3246 -
accuracy: 0.8597 - val_loss: 0.2872 - val_accuracy: 0.8838
Epoch 92/100
100/100 [==============================] - 18s 178ms/step - loss: 0.2751 -
accuracy: 0.8980 - val_loss: 0.3215 - val_accuracy: 0.8662
Epoch 93/100
100/100 [==============================] - 18s 177ms/step - loss: 0.3127 -
accuracy: 0.8677 - val_loss: 0.3091 - val_accuracy: 0.8725
Epoch 94/100
100/100 [==============================] - 18s 180ms/step - loss: 0.3033 -
accuracy: 0.8729 - val_loss: 0.3232 - val_accuracy: 0.8775
Epoch 95/100
100/100 [==============================] - 18s 176ms/step - loss: 0.3321 -
accuracy: 0.8638 - val_loss: 0.2590 - val_accuracy: 0.8913
Epoch 96/100
100/100 [==============================] - 18s 178ms/step - loss: 0.2836 -
accuracy: 0.8839 - val_loss: 0.3018 - val_accuracy: 0.8863
Epoch 97/100
100/100 [==============================] - 18s 177ms/step - loss: 0.2899 -
accuracy: 0.8805 - val_loss: 0.2738 - val_accuracy: 0.9013
Epoch 98/100
100/100 [==============================] - 18s 179ms/step - loss: 0.2824 -
accuracy: 0.8836 - val_loss: 0.3004 - val_accuracy: 0.8750
Epoch 99/100
100/100 [==============================] - 18s 180ms/step - loss: 0.2752 -
accuracy: 0.8758 - val_loss: 0.2789 - val_accuracy: 0.8913
Epoch 100/100
100/100 [==============================] - 18s 179ms/step - loss: 0.2877 -
accuracy: 0.8829 - val_loss: 0.3029 - val_accuracy: 0.8750
```

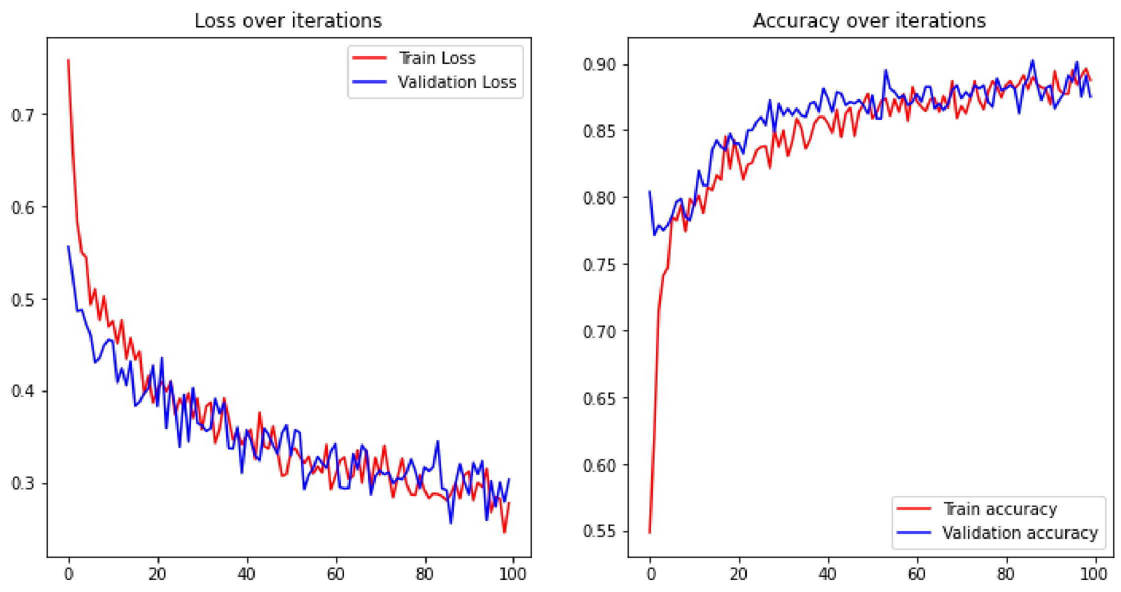
[ ]:
```
vgg_model.save('/content/gdrive/MyDrive/covid19/output/vgg_covid.h5')
```

[ ]:
```
history.history.keys()
```

```
[ ]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
[ ]: n = len(history.history["loss"])

     fig = plt.figure(figsize=(12,6))

     ax = fig.add_subplot(1,2,1)
     ax.plot(range(n), (history.history["loss"]),'r', label="Train Loss")
     ax.plot(range(n), (history.history["val_loss"]),'b', label="Validation Loss")
     ax.legend()
     ax.set_title('Loss over iterations')

     ax = fig.add_subplot(1,2,2)
     ax.plot(range(n), (history.history["accuracy"]),'r', label="Train accuracy")
     ax.plot(range(n), (history.history["val_accuracy"]),'b', label="Validation␣
      ↪accuracy")
     ax.legend()

     ax.set_title('Accuracy over iterations')
```

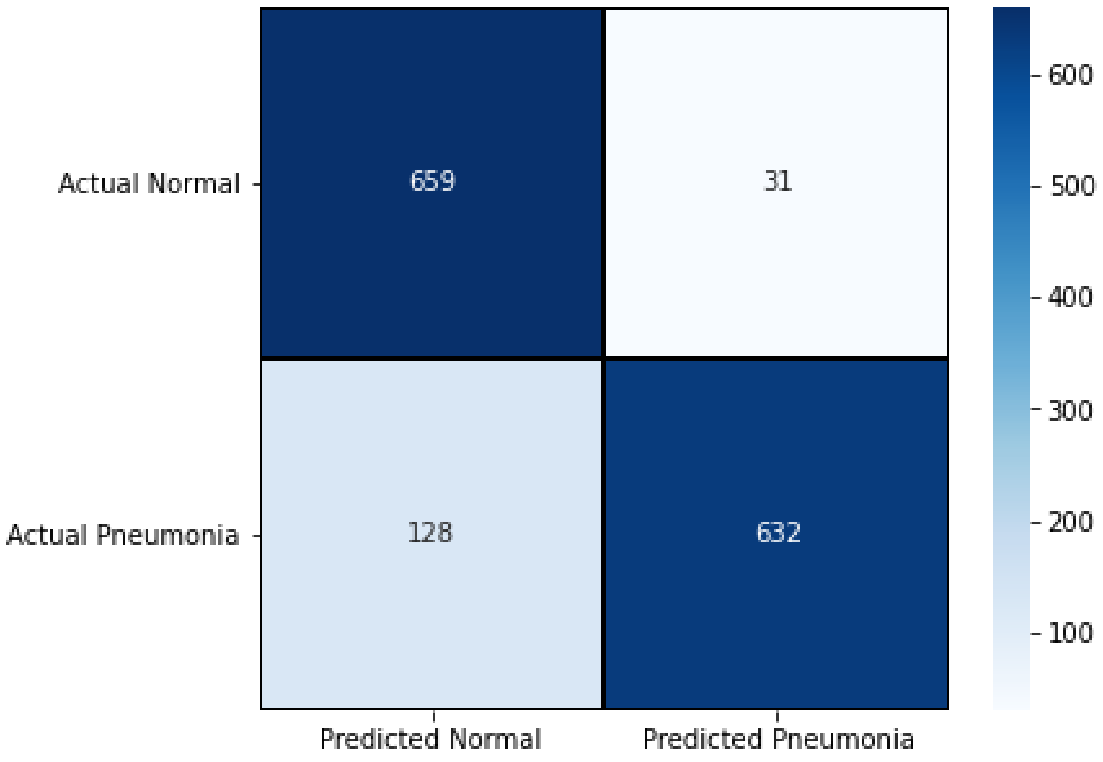
```
[ ]: Text(0.5, 1.0, 'Accuracy over iterations')
```



```
[ ]: vgg16_predictions = vgg_model.predict_classes(test_generator_vgg16)

     cm_16 = confusion_matrix(test_generator_vgg16.classes, vgg16_predictions)
     cm_pd16 = pd.DataFrame(cm_16 , index = ['0','1'] , columns = ['0','1'])
```

```
plt.figure(figsize = (6,5))
sns.heatmap(cm_pd16,cmap= "Blues", linecolor = 'black' ,
            linewidth = 1 , annot = True, fmt='',
            xticklabels = ['Predicted Normal', 'Predicted Pneumonia'],
            yticklabels = ['Actual Normal', 'Actual Pneumonia'])
plt.yticks(rotation=0)
plt.show()
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```



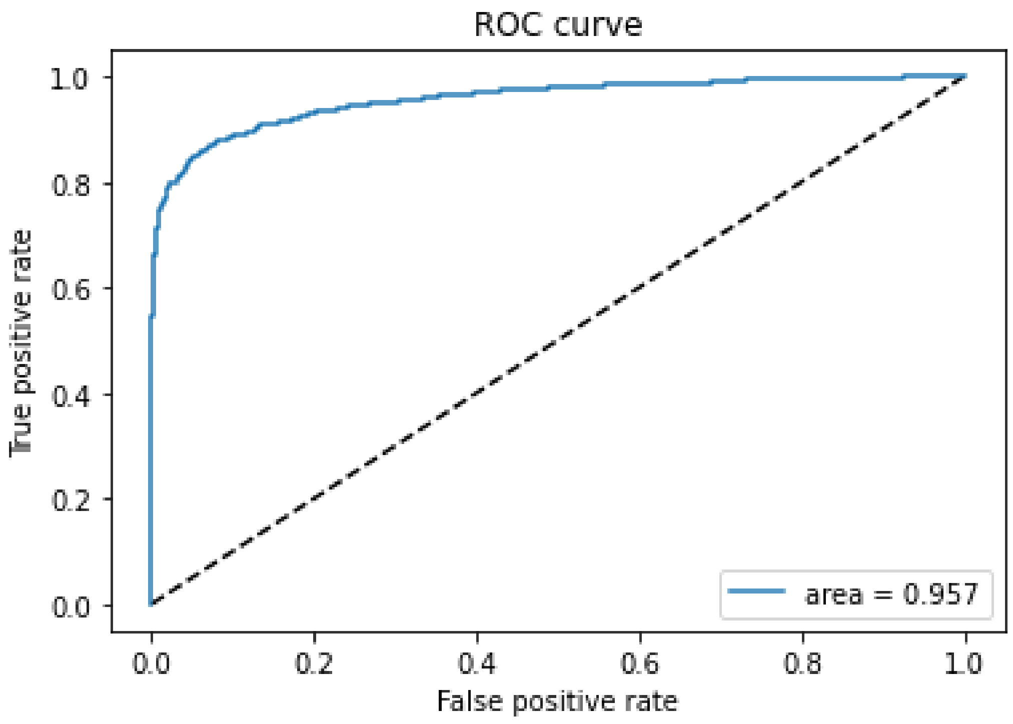
[ ]:
```
# make a prediction
y_pred_keras_vgg = vgg_model.predict_generator(test_generator_vgg16,␣
↪validation_generator_vgg16.samples // validation_generator_vgg16.
↪batch_size+1) #(test_gen, steps=len(df_val), verbose=1)
```

```
fpr_keras_vgg, tpr_keras_vgg, thresholds_keras_vgg =␣
 ↪roc_curve(test_generator_vgg16.classes, y_pred_keras_vgg)
auc_keras_vgg = auc(fpr_keras_vgg, tpr_keras_vgg)
auc_keras_vgg
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/training.py:1905: UserWarning:
`Model.predict_generator` is deprecated and will be removed in a future version.
Please use `Model.predict`, which supports generators.
  warnings.warn('`Model.predict_generator` is deprecated and '
```

```
[ ]: 0.9574656750572083
```

```
[ ]: plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras_vgg, tpr_keras_vgg, label='area = {:.3f}'.
 ↪format(auc_keras_vgg))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```



1.7 Conclusion

For this project, a dataset with more than 10.000 CT scans was utilized to create three convolutional neural network algorithms, for the accurate prediction of whether an individual has developed COVID-19 or not.

Accuracies and losses after 100 epochs:

	Accuracy	Val accuracy	Loss	Val loss
1st Model	0.9	0.91	0.25	0.255
2nd Model	0.85	0.84	0.4	0.37
3rd Model	0.88	0.875	0.28	0.30

For the AUC scores:

	AUC score
1st Model	0.96
2nd Model	0.9175
2nd Model	0.96

False predictions:

	False Predictions
1st Model	141
2nd Model	256
3rd Model	159

The higher the AUC, the better the performance of the model at distinguishing between the classes.

Recommendation: Even though the first model was quite simple, it seems that it gives the best results as both accuracies and true predictions are higher than the other two models.

1.7.1 Next steps

However, the predictions heatmap shows that the 1st model misclassifies lots of images. It was tested on a test set which consists of 1450 images and missclasified the 10% of them. There are several things to do in order to fix that, such as:

- Use a different CNN model (add more hidden layers, tweak the hyperparameters etc.). The disadvantege would be that the training process would eventually be far slower.
- Using k-fold cross validation alongside with hyperparameters tuning techniques (i.e. keras-tuner; finds the optimal number of layers for the specific problem). Again the training process

would require a lot of time.

- Using a different pretrained CNN architecture.
- Choose a different shape for the images. Here the images were compressed to 128x128.