```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#%matplotlib inline
import seaborn as sns
import cv2
import os
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.models import Model, Input
from keras.layers import Dense, Conv2D, BatchNormalization, GlobalAveragePooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.resnet import ResNet50
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, Input, Flatten

# Supress info, warnings and error messages
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

At this point, the image data is loaded and its paths are entered into a Pandas DataFrame, along with their tag (coronavirus or normal image) and an ID representing each tag.

```
disease_types = ['COVID', 'non-COVID']

train_dir = data_dir = '/content/drive/MyDrive/CT scan'

train_data = []

for index, sp in enumerate(disease_types):
    for file in os.listdir(os.path.join(train_dir, sp)):
        train_data.append([sp + "/" + file, index, sp])

train = pd.DataFrame(train_data, columns = ['File', 'ID','Disease Type'])
train
```

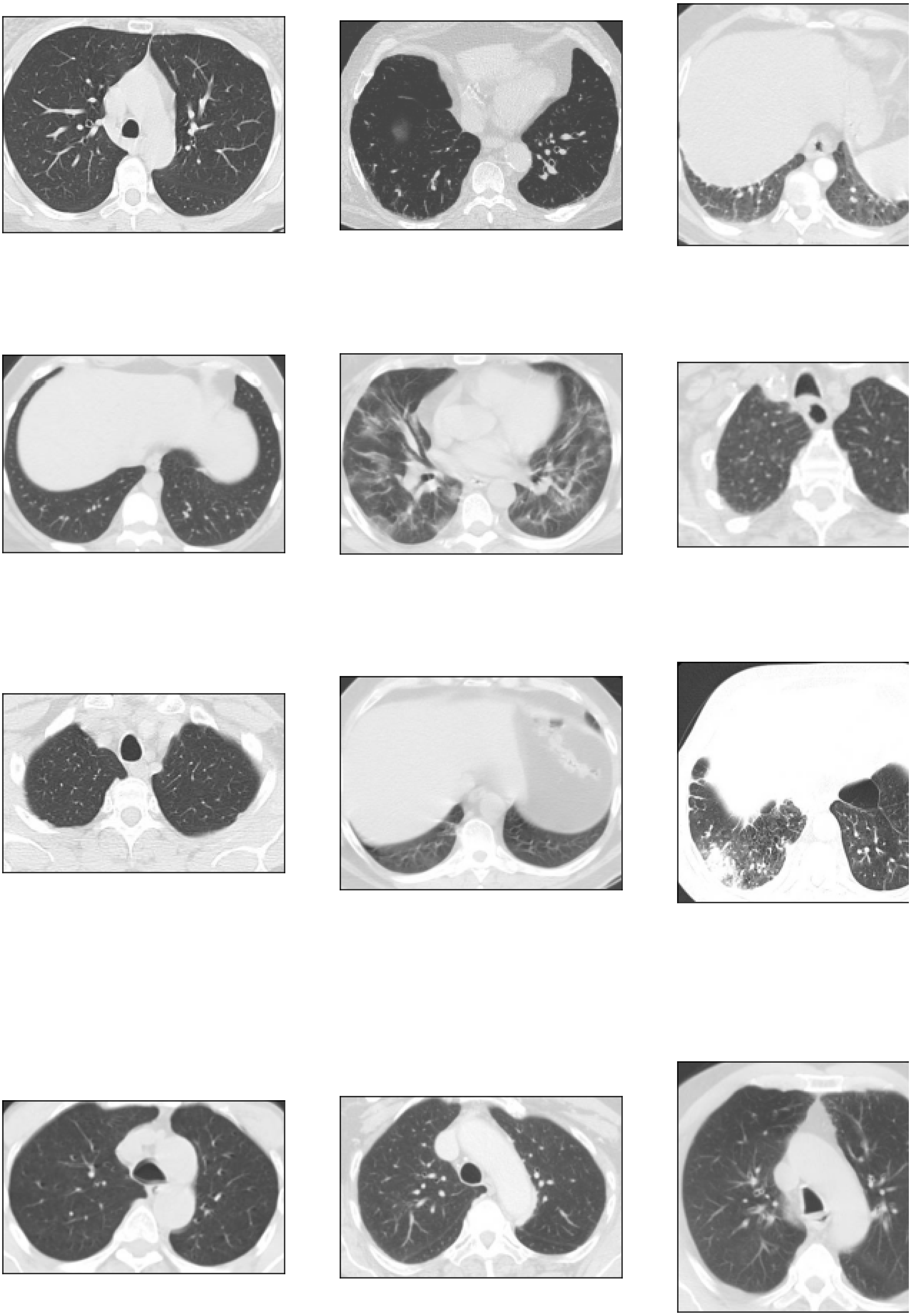
	File	ID	Disease Type
0	COVID/Covid (709).png	0	COVID
1	COVID/Covid (831).png	0	COVID
2	COVID/Covid (863).png	0	COVID
3	COVID/Covid (792).png	0	COVID
4	COVID/Covid (781).png	0	COVID
...	...	...	...
2476	non-COVID/Non-Covid (791).png	1	non-COVID
2477	non-COVID/Non-Covid (83).png	1	non-COVID

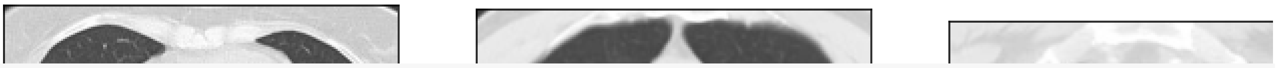
Then, the data are randomly shuffled to separate the training and test set, according to which the network will be trained and tested, respectively. The percentage of the training set corresponds to 80% of the data, while that of the test set, to the remaining 20% of the total data. In the pre-processing stage, the images are cropped to dimensions 224x224, categorized according to the class to which they belong and subjected to accidental alteration of some features, such as shift, inversion, focus, etc.

```
Seed = 40

train = train.sample(frac = 1, replace=False, random_state = Seed)

# Reset indices (row numbers)
train = train.reset_index(drop = True)

sns.countplot(x = "ID", data = train).set_title("Frequency Histogram (0: COVID, 1:Non-COVI
train
```

	File	ID	Disease Type
0	COVID/Covid (863).png	0	COVID
1	COVID/Covid (165).png	0	COVID
2	COVID/Covid (424).png	0	COVID
3	non-COVID/Non-Covid (812).png	1	non-COVID
4	COVID/Covid (1199).png	0	COVID
...	...	...	...
2476	non-COVID/Non-Covid (261).png	1	non-COVID
2477	non-COVID/Non-Covid (829).png	1	non-COVID
2478	non-COVID/Non-Covid (1051).png	1	non-COVID

```
def plot_defects(defect_types, rows, cols):
    fig, ax = plt.subplots(rows, cols, figsize=(12, 12))
    defect_files = train['File'][train['Disease Type'] == defect_types].values

    n = 0
    fig.suptitle(defect_types, fontsize = 22, color = "white")
    for i in range(rows):
        for j in range(cols):
            image_path = os.path.join(data_dir, defect_files[n])
            ax[i, j].set_xticks([])
            ax[i, j].set_yticks([])
            ax[i, j].imshow(cv2.imread(image_path))
            n += 1


plot_defects('COVID', 3, 3)
plot_defects('non-COVID', 3, 3)
```





```
IMAGE_SIZE = 224

# OpenCV Function to load colored image
def read_image(filepath):
    return cv2.imread(os.path.join(data_dir, filepath))

# OpenCV Function to resize an image
def resize_image(image, image_size):
    return cv2.resize(image.copy(), image_size, interpolation = cv2.INTER_AREA)
```

```
X_train = np.zeros((train.shape[0], IMAGE_SIZE, IMAGE_SIZE, 3))

for i, file in enumerate(train['File'].values):
    image = read_image(file)
    if image is not None:
        X_train[i] = resize_image(image, (IMAGE_SIZE, IMAGE_SIZE))

X_Train = X_train / 255.0   # Pixel normalization
print('Train Shape:', X_Train.shape)

Y_train = to_categorical(train['ID'].values, num_classes = 2)

print(Y_train)
```

```
Train Shape: (2481, 224, 224, 3)
[[1. 0.]
 [1. 0.]
 [1. 0.]
 ...
 [0. 1.]
 [0. 1.]
 [0. 1.]]
```

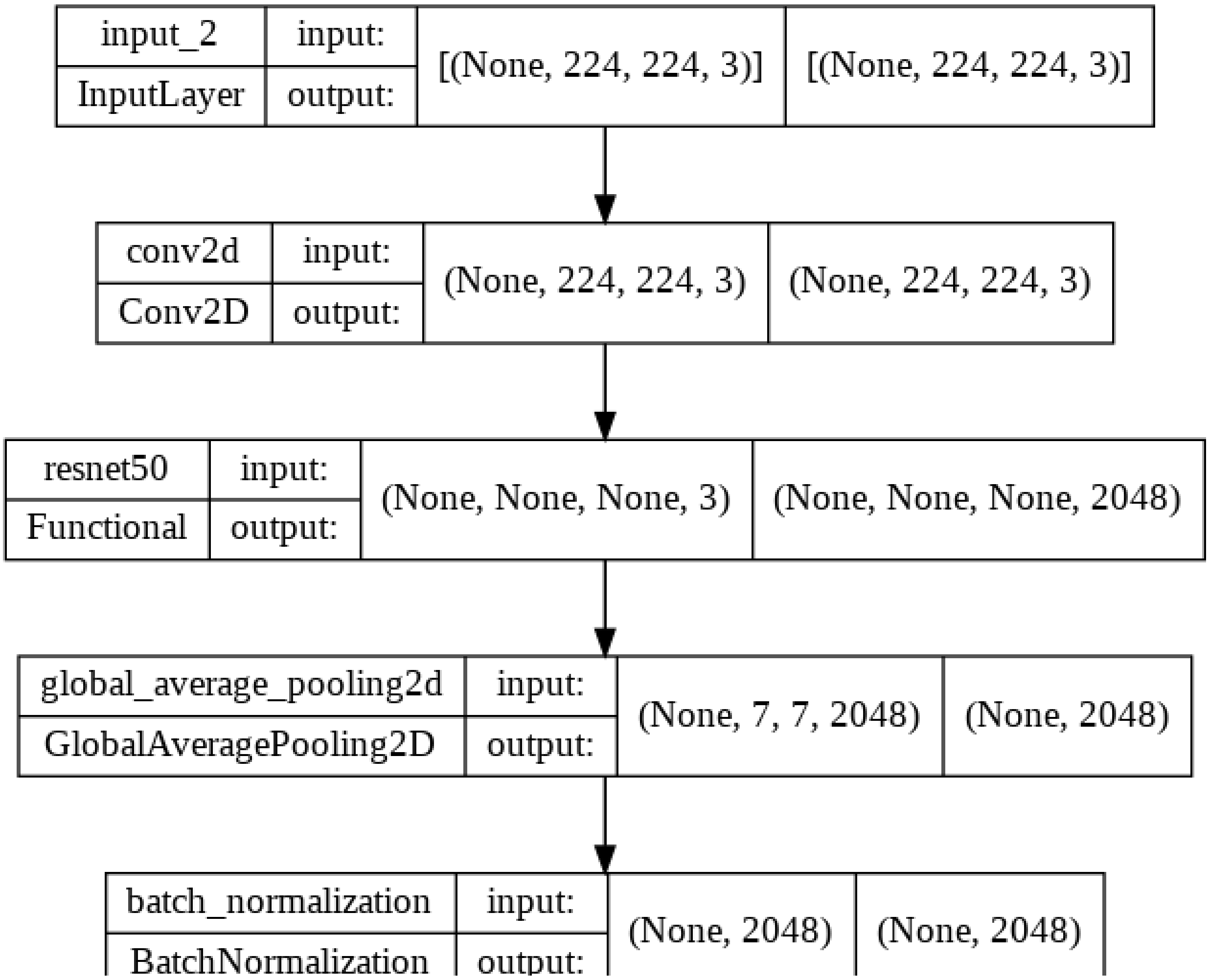
```
# Dataframe split to train and validation set (80% train and 20% validation)
X_train, X_val, Y_train, Y_val = train_test_split(X_Train,
                                                  Y_train,
                                                  test_size = 0.2, # Percent 20% of the da
                                                  random_state = Seed)

print(f'X_train:', X_train.shape)
print(f'X_val:', X_val.shape)
print(f'Y_train:', Y_train.shape)
print(f'Y_val:', Y_val.shape)
```

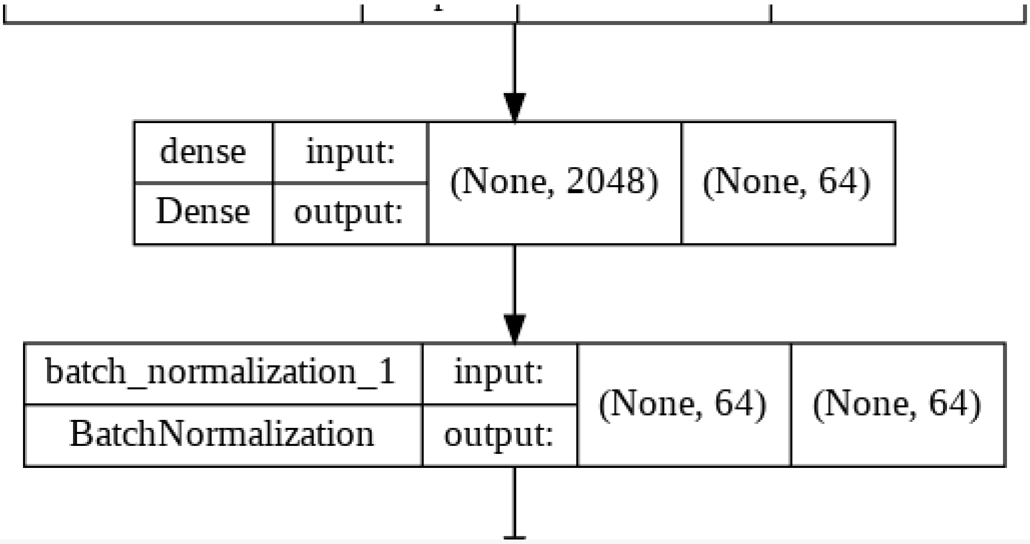
```
X_train: (1984, 224, 224, 3)
X_val: (497, 224, 224, 3)
Y_train: (1984, 2)
Y_val: (497, 2)
```

```
# Architectural function for Resnet50
def build_resnet50(IMAGE_SIZE, channels):

    resnet50 = ResNet50(weights = 'imagenet', include_top = False)

    input = Input(shape = (IMAGE_SIZE, IMAGE_SIZE, channels))
    x = Conv2D(3, (3, 3), padding = 'same')(input)
    x = resnet50(x)
    x = GlobalAveragePooling2D()(x)
    x = BatchNormalization()(x)
    x = Dense(64, activation = 'relu')(x)
    x = BatchNormalization()(x)

    output = Dense(2, activation = 'softmax')(x)

    # model
    model = Model(input, output)

    optimizer = Adam(learning_rate = 0.003, beta_1 = 0.9, beta_2 = 0.999, epsilon = 0.1, d
    model.compile(loss = 'categorical_crossentropy',  # minimize the negative multinomial
                  optimizer = optimizer,
                  metrics = ['accuracy'])
    model.summary()

    return model
```

```
channels = 3

model = build_resnet50(IMAGE_SIZE, channels)
annealer = ReduceLROnPlateau(monitor = 'val_accuracy',  # Reduce learning rate when Valida
                             factor = 0.70,  # Rate by which the learning rate will decrea
                             patience = 5,   # number of epochs without improvement, after
                             verbose = 1,    # Display messages
                             min_lr = 1e-4   # lower limit on the learning rate.
                            )
checkpoint = ModelCheckpoint('model.h5', verbose = 1, save_best_only = True)  # Save neura

# Generates batches of image data with data augmentation
datagen = ImageDataGenerator(rotation_range = 360, # Degree range for random rotations
                        width_shift_range = 0.2,   # Range for random horizontal shifts
                        height_shift_range = 0.2,  # Range for random vertical shifts
                        zoom_range = 0.2,          # Range for random zoom
                        horizontal_flip = True,    # Randomly flip inputs horizontally
                        vertical_flip = True)      # Randomly flip inputs vertically

datagen.fit(X_train)

plot_model(model, to_file = 'convnet.png', show_shapes = True, show_layer_names = True)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/r
94773248/94765736 [==============================] - 1s 0us/step
94781440/94765736 [==============================] - 1s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 conv2d (Conv2D)             (None, 224, 224, 3)       84

 resnet50 (Functional)       (None, None, None, 2048)  23587712

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 batch_normalization (BatchN  (None, 2048)             8192
 ormalization)

 dense (Dense)               (None, 64)                131136

 batch_normalization_1 (Batc  (None, 64)               256
 hNormalization)

 dense_1 (Dense)             (None, 2)                 130

=================================================================
Total params: 23,727,510
Trainable params: 23,670,166
Non-trainable params: 57,344
_________________________________________________________________
```






```
BATCH_SIZE = 32
EPOCHS = 50

# Fit of the model that includes the augmented images in terms of their characteristics
hist = model.fit(datagen.flow(X_train, Y_train, batch_size = BATCH_SIZE),
              steps_per_epoch = X_train.shape[0] // BATCH_SIZE,
              epochs = EPOCHS,
              verbose = 1,
              callbacks = [annealer, checkpoint],
              validation_data = (X_val, Y_val))
```

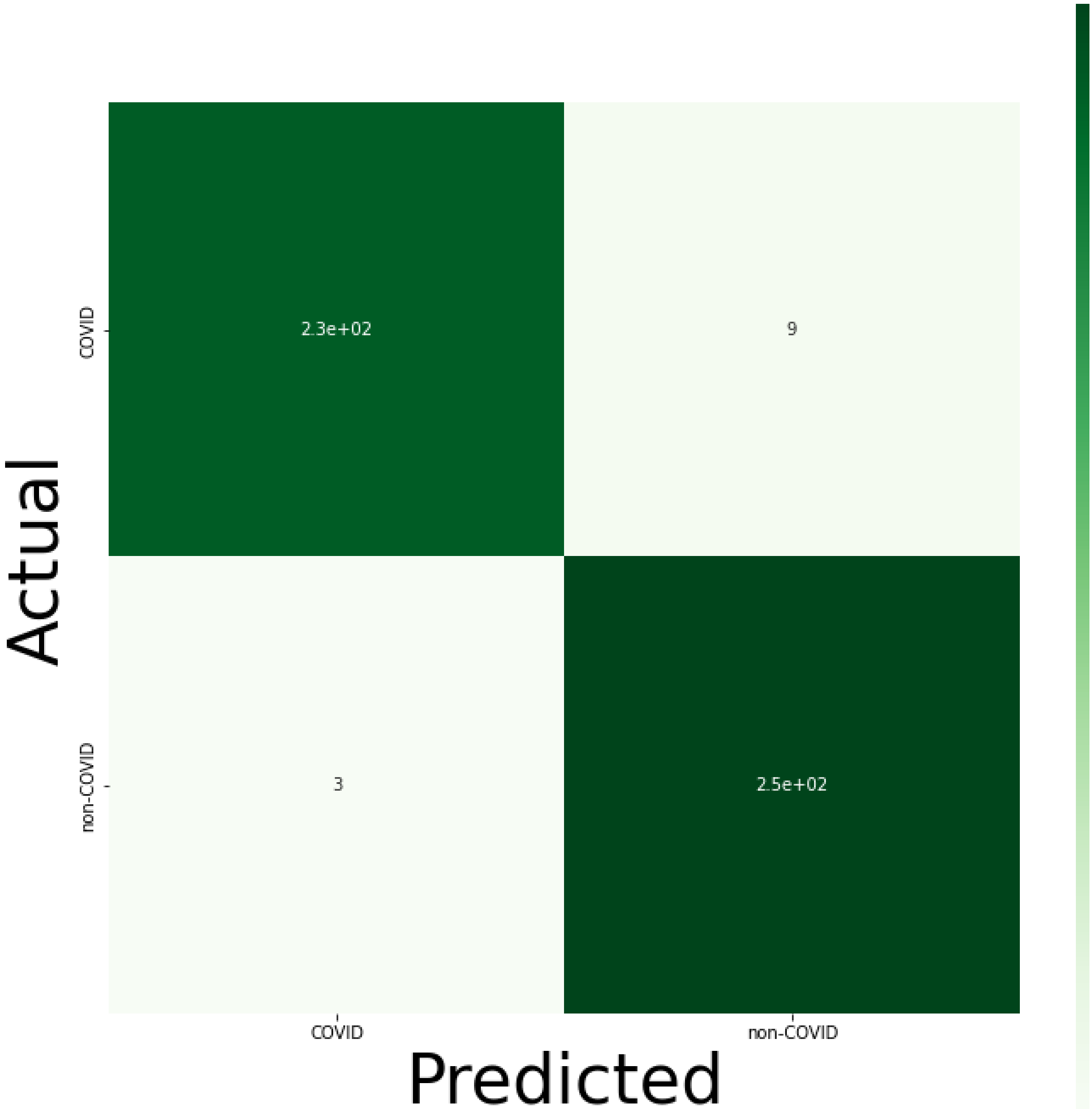
```
Epoch 1/50
62/62 [==============================] - ETA: 0s - loss: 0.5884 - accuracy: 0.736
Epoch 1: val_loss improved from inf to 1.97931, saving model to model.h5
62/62 [==============================] - 50s 517ms/step - loss: 0.5884 - accuracy
Epoch 2/50
62/62 [==============================] - ETA: 0s - loss: 0.4038 - accuracy: 0.832
Epoch 2: val_loss did not improve from 1.97931
62/62 [==============================] - 27s 439ms/step - loss: 0.4038 - accuracy
Epoch 3/50
62/62 [==============================] - ETA: 0s - loss: 0.3291 - accuracy: 0.875
Epoch 3: val_loss did not improve from 1.97931
62/62 [==============================] - 27s 439ms/step - loss: 0.3291 - accuracy
Epoch 4/50
62/62 [==============================] - ETA: 0s - loss: 0.2664 - accuracy: 0.894
Epoch 4: val_loss did not improve from 1.97931
62/62 [==============================] - 28s 445ms/step - loss: 0.2664 - accuracy
Epoch 5/50
62/62 [==============================] - ETA: 0s - loss: 0.2630 - accuracy: 0.894
Epoch 5: val_loss did not improve from 1.97931
62/62 [==============================] - 28s 445ms/step - loss: 0.2630 - accuracy
Epoch 6/50
62/62 [==============================] - ETA: 0s - loss: 0.2097 - accuracy: 0.911
Epoch 6: val_loss did not improve from 1.97931
62/62 [==============================] - 28s 441ms/step - loss: 0.2097 - accuracy
Epoch 7/50
62/62 [==============================] - ETA: 0s - loss: 0.1971 - accuracy: 0.919
Epoch 7: val_loss did not improve from 1.97931
62/62 [==============================] - 28s 445ms/step - loss: 0.1971 - accuracy
Epoch 8/50
62/62 [==============================] - ETA: 0s - loss: 0.1693 - accuracy: 0.930
Epoch 8: val_loss improved from 1.97931 to 1.65606, saving model to model.h5
62/62 [==============================] - 29s 465ms/step - loss: 0.1693 - accuracy
```

```
Epoch 9/50
62/62 [==============================] - ETA: 0s - loss: 0.1531 - accuracy: 0.938
Epoch 9: val_loss improved from 1.65606 to 0.96230, saving model to model.h5
62/62 [==============================] - 30s 480ms/step - loss: 0.1531 - accuracy
Epoch 10/50
62/62 [==============================] - ETA: 0s - loss: 0.1475 - accuracy: 0.940
Epoch 10: val_loss did not improve from 0.96230
62/62 [==============================] - 28s 443ms/step - loss: 0.1475 - accuracy
Epoch 11/50
62/62 [==============================] - ETA: 0s - loss: 0.1344 - accuracy: 0.950
Epoch 11: val_loss did not improve from 0.96230
62/62 [==============================] - 28s 445ms/step - loss: 0.1344 - accuracy
Epoch 12/50
62/62 [==============================] - ETA: 0s - loss: 0.1258 - accuracy: 0.949
Epoch 12: val_loss did not improve from 0.96230
62/62 [==============================] - 28s 446ms/step - loss: 0.1258 - accuracy
Epoch 13/50
62/62 [==============================] - ETA: 0s - loss: 0.0939 - accuracy: 0.964
Epoch 13: val_loss improved from 0.96230 to 0.69623, saving model to model.h5
62/62 [==============================] - 29s 469ms/step - loss: 0.0939 - accuracy
Epoch 14/50
62/62 [==============================] - ETA: 0s - loss: 0.1032 - accuracy: 0.956
Epoch 14: val_loss improved from 0.69623 to 0.54984, saving model to model.h5
62/62 [==============================] - 29s 466ms/step - loss: 0.1032 - accuracy
Epoch 15/50
```

```
Y_pred = model.predict(X_val)

Y_pred = np.argmax(Y_pred, axis = 1)
Y_true = np.argmax(Y_val, axis = 1)

cm = confusion_matrix(Y_true, Y_pred)
plt.figure(figsize = (12, 12))
ax = sns.heatmap(cm, cmap = plt.cm.Greens, annot = True, square = True, xticklabels = dise
ax.set_ylabel('Actual', fontsize = 40)
ax.set_xlabel('Predicted', fontsize = 40)



TP = cm[1][1]
print(f"True Positive: {TP}")



FN = cm[1][0]
print(f"False Negative: {FN}")

TN = cm[0][0]
print(f"True Negative: {TN}")

FP = cm[0][1]
print(f"False Positive: {FP}")

# Sensitivity, recall, or true positive rate
print(f"True Positive Rate: {TP / (TP + FN)}")

# Specificity or true negative rate
print(f"True Negative Rate: {TN / (TN + FP)}\n")
```

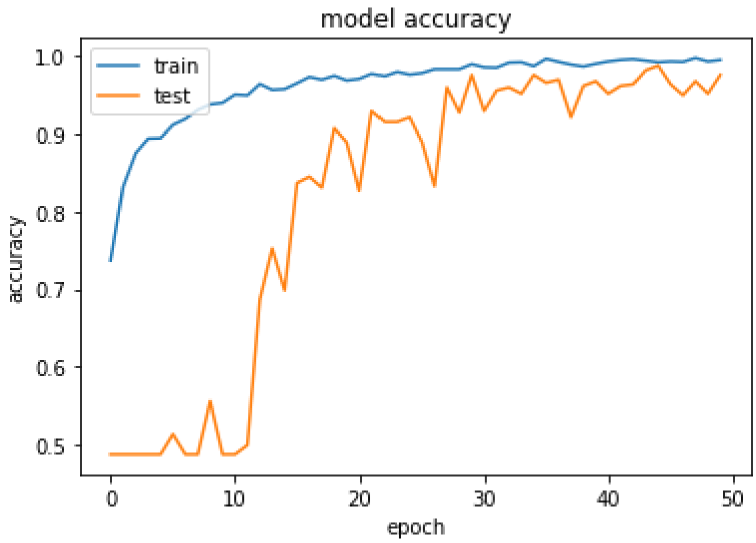
```
final_loss, final_accuracy = model.evaluate(X_val, Y_val)
print(f"\nFinal Loss: {final_loss}, Final Accuracy: {final_accuracy}")
```

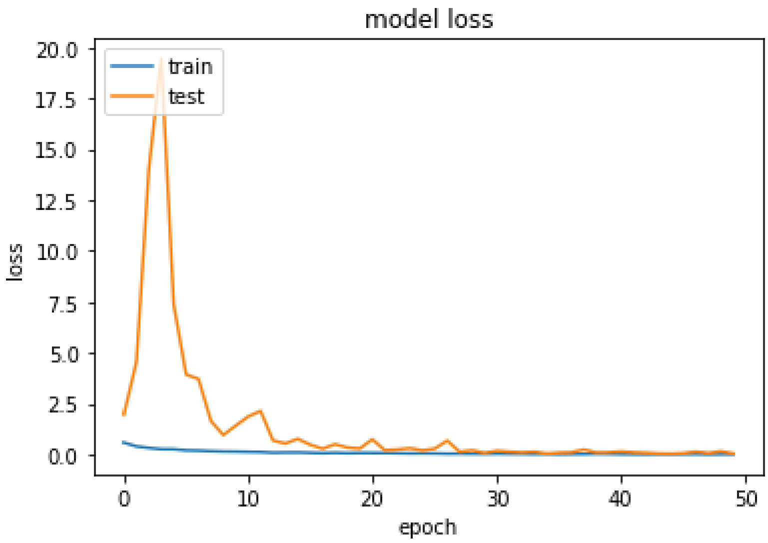
```
True Positive: 252
False Negative: 3
True Negative: 233
False Positive: 9
True Positive Rate: 0.9882352941176471
True Negative Rate: 0.9628099173553719

16/16 [==============================] - 2s 98ms/step - loss: 0.0540 - accuracy: 0.9

Final Loss: 0.05403857305645943, Final Accuracy: 0.9758551120758057
```




```
# Accuracy plot
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
```

```
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()

# Loss plot
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()
```

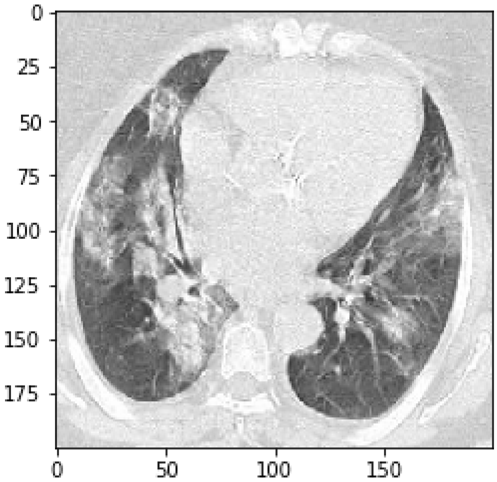





```
from keras.preprocessing import image

img = image.load_img('/content/drive/MyDrive/CT scan/COVID/Covid (1007).png', grayscale =
show_img = image.load_img('/content/drive/MyDrive/CT scan/COVID/Covid (1007).png', graysca
disease_class = ['Covid-19','Non Covid-19']
x = image.img_to_array(img)
x = np.expand_dims(x, axis = 0)
x /= 255

custom = model.predict(x)
```

```
print(custom[0])

plt.imshow(show_img)
plt.show()

a = custom[0]
ind = np.argmax(a)

print('Prediction:',disease_class[ind])
```

```
[1.000000e+00 4.365647e-11]
```




```
Prediction: Covid-19
```