```
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
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/Colab Notebooks'

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 (215).png	0	COVID
1	COVID/Covid (124).png	0	COVID
2	COVID/Covid (26).png	0	COVID
3	COVID/Covid (187).png	0	COVID
4	COVID/Covid (203).png	0	COVID
...	...	...	...
2476	non-COVID/Non-Covid (1132).png	1	non-COVID
2477	non-COVID/Non-Covid (1061).png	1	non-COVID
2478	non-COVID/Non-Covid (1174).png	1	non-COVID
2479	non-COVID/Non-Covid (1022).png	1	non-COVID
2480	non-COVID/Non-Covid (1028).png	1	non-COVID

2481 rows × 3 columns

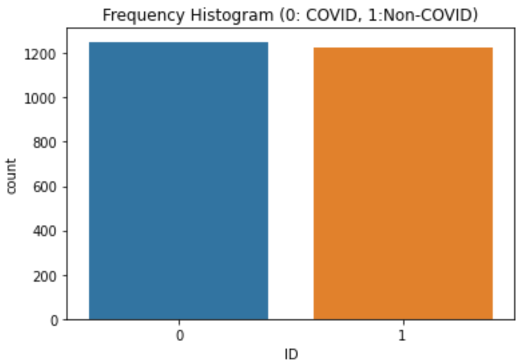
Then, the data are randomly shuffled to separate the training and test set, according to which the network will be trained and tested, respectively. The percentage of the training set corresponds to 80% of the data, while that of the test set, to the remaining 20% of the total data. In the pre-processing 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
```

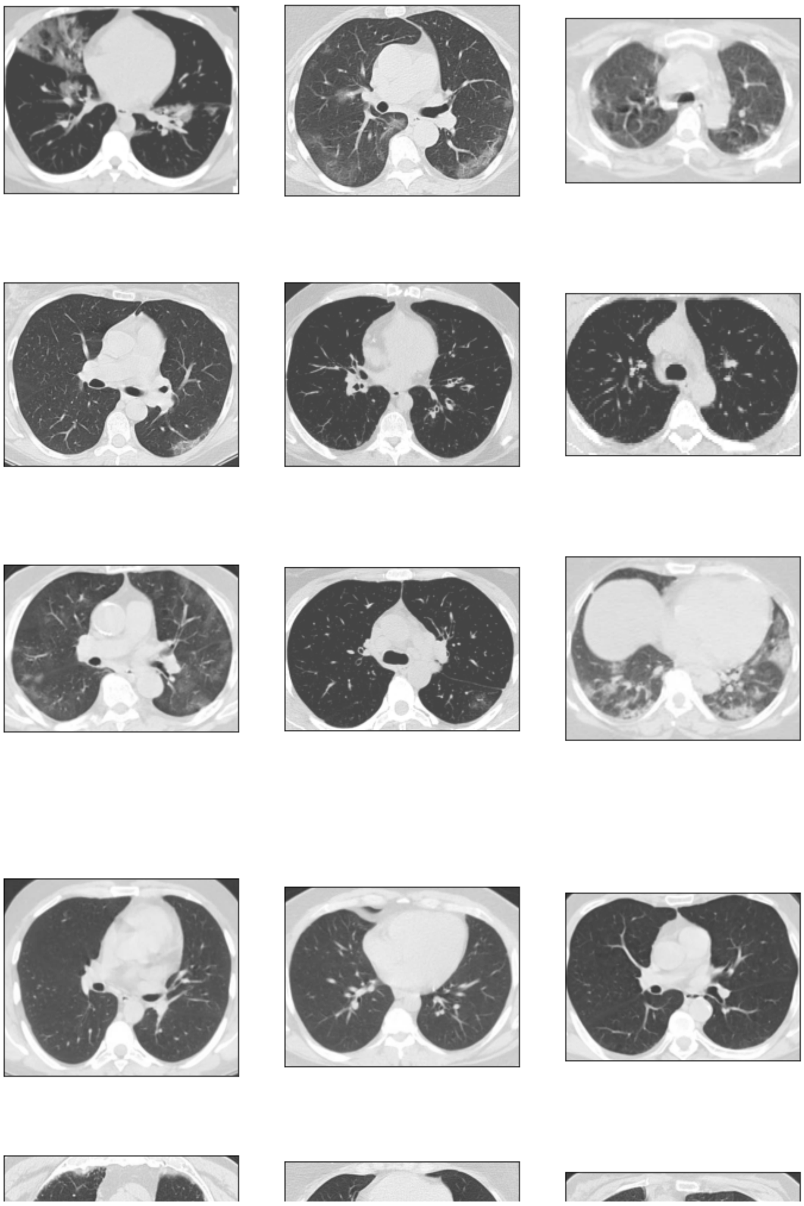
```
train = train.sample(frac = 1, replace=False, random_state = Seed)

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

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

	File	ID	Disease Type
0	COVID/Covid (26).png	0	COVID
1	COVID/Covid (716).png	0	COVID
2	COVID/Covid (579).png	0	COVID
3	non-COVID/Non-Covid (266).png	1	non-COVID
4	COVID/Covid (852).png	0	COVID
...	...	...	...
2476	non-COVID/Non-Covid (543).png	1	non-COVID
2477	non-COVID/Non-Covid (164).png	1	non-COVID
2478	non-COVID/Non-Covid (990).png	1	non-COVID
2479	non-COVID/Non-Covid (723).png	1	non-COVID
2480	non-COVID/Non-Covid (100).png	1	non-COVID

2481 rows × 3 columns




```
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)
```

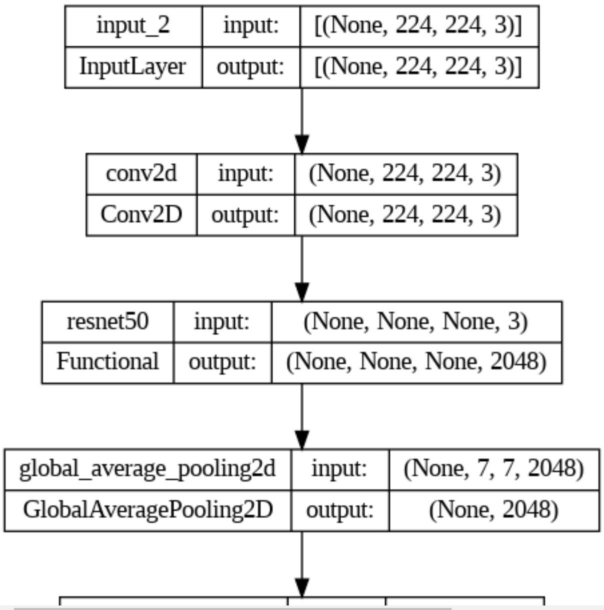
```
Train Shape: (2481, 224, 224, 3)
[[1. 0.]
 [1. 0.]
 [1. 0.]
 ...
 [0. 1.]
 [0. 1.]
 [0. 1.]]
```

```
# 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 data is using as test set
                                                  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, decay = 0.0)
    model.compile(loss = 'categorical_crossentropy',  # minimize the negative multinomial log-likelihood also known as the cross-entropy.
                  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 Validation accuracy remains constant
                             factor = 0.70,  # Rate by which the learning rate will decrease
                             patience = 5,   # number of epochs without improvement, after which the learning rate will decrease
                             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 neural network weights

# 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-applicatio
94765736/94765736 [==============================] - 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
_________________________________________________________________
```

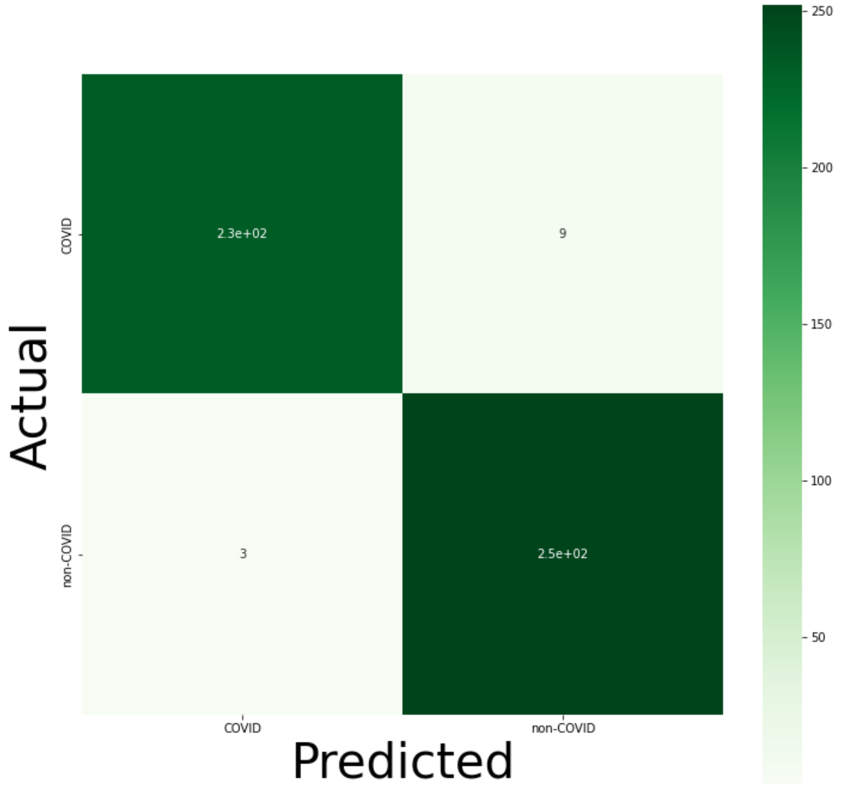



```
BATCH_SIZE = 32
EPOCHS = 50

# 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))
```

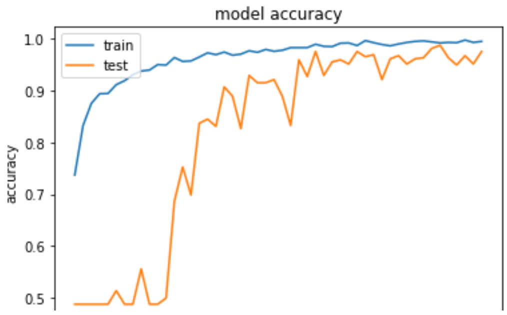
```
Epoch 38: val_loss did not improve from 0.05099
62/62 [==============================] - 29s 463ms/step - loss: 0.0270 - accuracy: 0.9894 - val_loss: 0.2484 - val_accuracy: 0.92
Epoch 39/50
62/62 [==============================] - ETA: 0s - loss: 0.0389 - accuracy: 0.9869
Epoch 39: val_loss did not improve from 0.05099
62/62 [==============================] - 28s 446ms/step - loss: 0.0389 - accuracy: 0.9869 - val_loss: 0.1025 - val_accuracy: 0.96
Epoch 40/50
62/62 [==============================] - ETA: 0s - loss: 0.0329 - accuracy: 0.9904
Epoch 40: ReduceLROnPlateau reducing learning rate to 0.0010289999307133257.

Epoch 40: val_loss did not improve from 0.05099
62/62 [==============================] - 28s 444ms/step - loss: 0.0329 - accuracy: 0.9904 - val_loss: 0.1111 - val_accuracy: 0.96
Epoch 41/50
62/62 [==============================] - ETA: 0s - loss: 0.0198 - accuracy: 0.9934
Epoch 41: val_loss did not improve from 0.05099
62/62 [==============================] - 30s 479ms/step - loss: 0.0198 - accuracy: 0.9934 - val_loss: 0.1560 - val_accuracy: 0.95
Epoch 42/50
62/62 [==============================] - ETA: 0s - loss: 0.0165 - accuracy: 0.9955
Epoch 42: val_loss did not improve from 0.05099
62/62 [==============================] - 28s 444ms/step - loss: 0.0165 - accuracy: 0.9955 - val_loss: 0.1116 - val_accuracy: 0.96
Epoch 43/50
62/62 [==============================] - ETA: 0s - loss: 0.0145 - accuracy: 0.9965
Epoch 43: val_loss did not improve from 0.05099
62/62 [==============================] - 28s 446ms/step - loss: 0.0145 - accuracy: 0.9965 - val_loss: 0.0868 - val_accuracy: 0.96
Epoch 44/50
62/62 [==============================] - ETA: 0s - loss: 0.0218 - accuracy: 0.9945
Epoch 44: val_loss did not improve from 0.05099
62/62 [==============================] - 28s 446ms/step - loss: 0.0218 - accuracy: 0.9945 - val_loss: 0.0536 - val_accuracy: 0.98
Epoch 45/50
62/62 [==============================] - ETA: 0s - loss: 0.0207 - accuracy: 0.9924
Epoch 45: val_loss improved from 0.05099 to 0.04413, saving model to model.h5
62/62 [==============================] - 29s 467ms/step - loss: 0.0207 - accuracy: 0.9924 - val_loss: 0.0441 - val_accuracy: 0.98
Epoch 46/50
62/62 [==============================] - ETA: 0s - loss: 0.0198 - accuracy: 0.9934
Epoch 46: val_loss did not improve from 0.04413
62/62 [==============================] - 28s 445ms/step - loss: 0.0198 - accuracy: 0.9934 - val_loss: 0.0705 - val_accuracy: 0.96
Epoch 47/50
62/62 [==============================] - ETA: 0s - loss: 0.0217 - accuracy: 0.9929
Epoch 47: val_loss did not improve from 0.04413
62/62 [==============================] - 28s 445ms/step - loss: 0.0217 - accuracy: 0.9929 - val_loss: 0.1450 - val_accuracy: 0.94
Epoch 48/50
62/62 [==============================] - ETA: 0s - loss: 0.0120 - accuracy: 0.9980
Epoch 48: val_loss did not improve from 0.04413
62/62 [==============================] - 28s 444ms/step - loss: 0.0120 - accuracy: 0.9980 - val_loss: 0.0763 - val_accuracy: 0.96
Epoch 49/50
62/62 [==============================] - ETA: 0s - loss: 0.0170 - accuracy: 0.9934
Epoch 49: val_loss did not improve from 0.04413
62/62 [==============================] - 30s 482ms/step - loss: 0.0170 - accuracy: 0.9934 - val_loss: 0.1637 - val_accuracy: 0.95
Epoch 50/50
62/62 [==============================] - ETA: 0s - loss: 0.0174 - accuracy: 0.9955
Epoch 50: ReduceLROnPlateau reducing learning rate to 0.0007202999433502554.
```

```
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 = disease_types, yticklabels = disease_types)
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

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

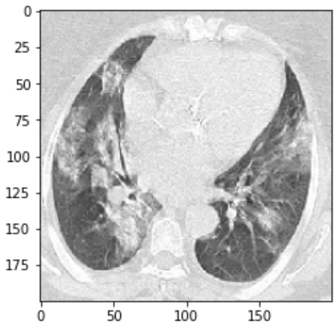



```
# Accuracy plot
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()

# 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 = False, target_size = (224, 224))
show_img = image.load_img('/content/drive/MyDrive/CT scan/COVID/Covid (1007).png', grayscale = False, target_size = (200, 200))
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
```