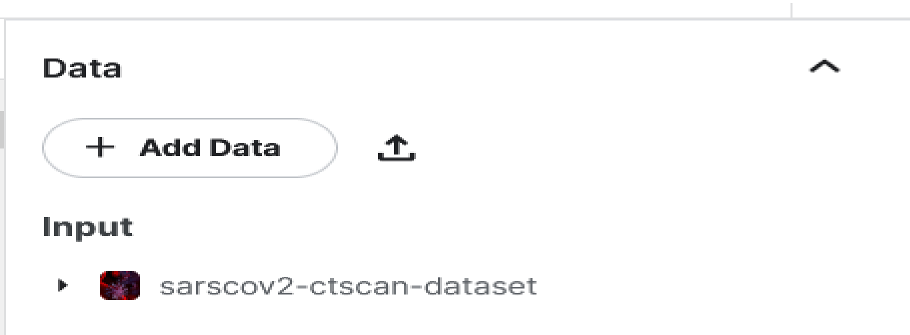
CT Scan Image Classification

Code and Output:

Step 1: Import all the libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import cv2
import os
from tqdm import tqdm
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,Sequential, Input, load_model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization, AveragePooling2D, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.applications.resnet50 import ResNet50
```

Step 2: Add data in the kaggle and convert it into dataframe



[3]:
```
disease_types=['COVID', 'non-COVID']
data_dir = '../input/sarscov2-ctscan-dataset'
train_dir = os.path.join(data_dir)
```

[4]:
```
train_data = []
for defects_id, sp in enumerate(disease_types):
    for file in os.listdir(os.path.join(train_dir, sp)):
        train_data.append(['{}/{}'.format(sp, file), defects_id, sp])

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

[4]:

	File	DiseaseID	Disease Type
0	COVID/Covid (230).png	0	COVID
1	COVID/Covid (1195).png	0	COVID
2	COVID/Covid (182).png	0	COVID
3	COVID/Covid (817).png	0	COVID
4	COVID/Covid (631).png	0	COVID

Step 3: Random or reset the index of the images

[5]:

```
SEED = 42
train = train.sample(frac=1, random_state=SEED)
train.index = np.arange(len(train)) # Reset indices
train.head()
```

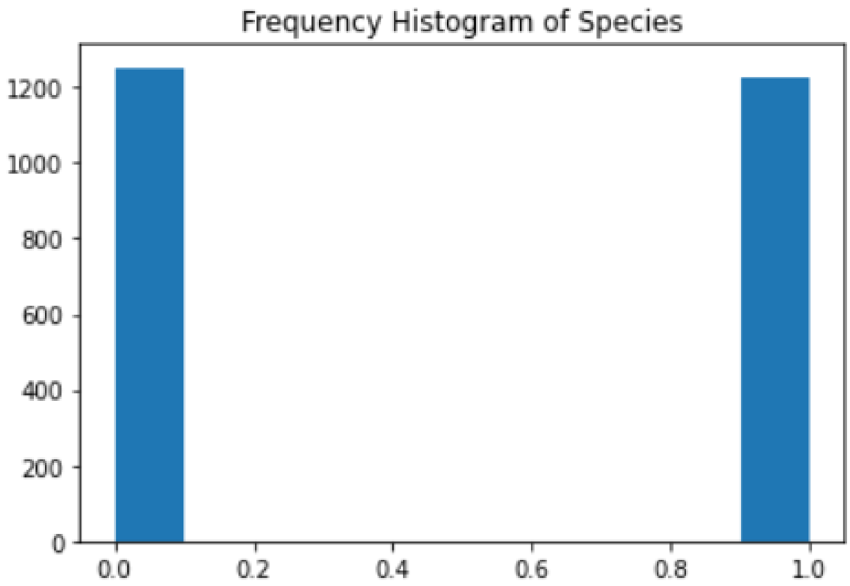
[5]:

	File	DiseaseID	Disease Type
0	COVID/Covid (54).png	0	COVID
1	COVID/Covid (1035).png	0	COVID
2	non-COVID/Non-Covid (21).png	1	non-COVID
3	non-COVID/Non-Covid (248).png	1	non-COVID
4	COVID/Covid (409).png	0	COVID

Step 4: Create a histogram

[6]:

```
plt.hist(train['DiseaseID'])
plt.title('Frequency Histogram of Species')
plt.figure(figsize=(12, 12))
plt.show()
```

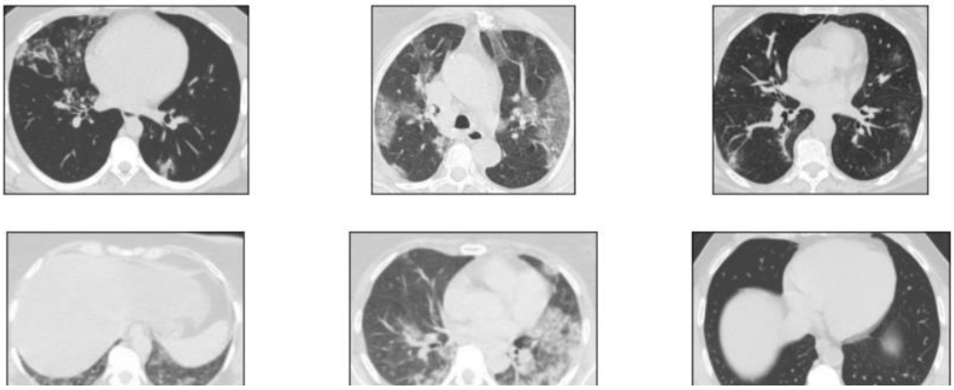



```
<Figure size 864x864 with 0 Axes>
```

Step 5: Plot Covid and Non Covid Images

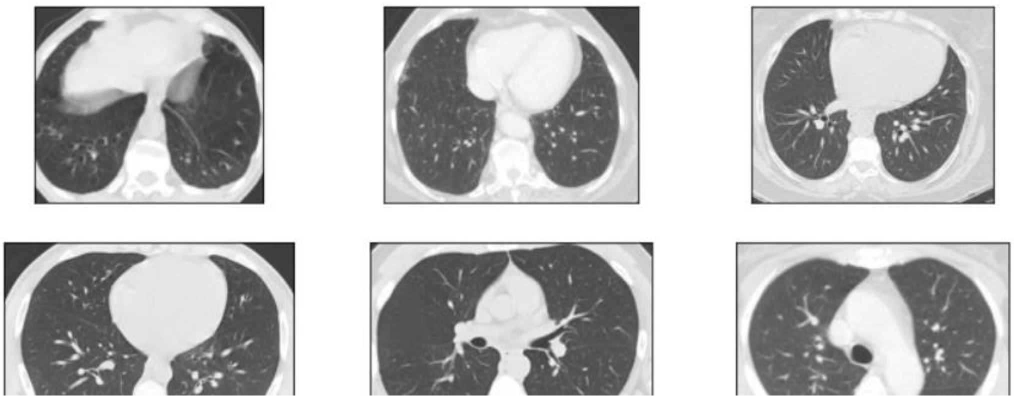
[7]:

```
def plot_defects(defect_types, rows, cols):
    fig, ax = plt.subplots(rows, cols, figsize=(12, 8))
    defect_files = train['File'][train['Disease Type'] == defect_types].values
    n = 0
    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
# Displays first n images of class from training set
plot_defects('COVID', 3, 3)
```



[8]:

```
def plot_defects(defect_types, rows, cols):
    fig, ax = plt.subplots(rows, cols, figsize=(12, 8))
    defect_files = train['File'][train['Disease Type'] == defect_types].values
    n = 0
    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
# Displays first n images of class from training set
plot_defects('non-COVID', 3, 3)
```



Step 6: Resize the Image, normalize the image. Also Split the data into train and validation sets

[9]:
```
IMAGE_SIZE = 64
def read_image(filepath):
    return cv2.imread(os.path.join(data_dir, filepath)) # Loading a color image is the default flag
# Resize image to target size
def resize_image(image, image_size):
    return cv2.resize(image.copy(), image_size, interpolation=cv2.INTER_AREA)
```

[10]:
```
X_train = np.zeros((train.shape[0], IMAGE_SIZE, IMAGE_SIZE, 3))
for i, file in tqdm(enumerate(train['File'].values)):
    image = read_image(file)
    if image is not None:
        X_train[i] = resize_image(image, (IMAGE_SIZE, IMAGE_SIZE))
# Normalize the data
X_Train = X_train / 255.
print('Train Shape: {}'.format(X_Train.shape))
```

```
2481it [00:36, 68.81it/s]
Train Shape: (2481, 64, 64, 3)
```

[11]:
```
Y_train = train['DiseaseID'].values
Y_train = to_categorical(Y_train, num_classes=2)
print(Y_train.shape)
```

```
(2481, 2)
```

[12]:
```
BATCH_SIZE = 64

# Split the train and validation sets
X_train, X_val, Y_train, Y_val = train_test_split(X_Train, Y_train, test_size=0.2, random_state=SEED)
```

[13]:
```
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, 64, 64, 3)
X_val: (497, 64, 64, 3)
Y_train: (1984, 2)
Y_val: (497, 2)
```

Step 7: Create a resnet model(ResNet50) also use data augmentation, checkpoint, early stopping and ReduceLROnPlateau and fit the model. You will save the model as .h5 file for the later use of the prediction

[14]:
```
EPOCHS = 50
SIZE=64
N_ch=3
```

+ Code   + Markdown

[15]:
```
def build_resnet50():
    resnet50 = ResNet50(weights='imagenet', include_top=False)

    input = Input(shape=(SIZE, SIZE, N_ch))
    x = Conv2D(3, (3, 3), padding='same')(input)

    x = resnet50(x)

    x = GlobalAveragePooling2D()(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

    # multi output
    output = Dense(2,activation = 'softmax', name='root')(x)
```

```
    # model
    model = Model(input,output)

    optimizer = Adam(lr=0.003, beta_1=0.9, beta_2=0.999, epsilon=0.1, decay=0.0)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    model.summary()

    return model
```

[16]:
```
model = build_resnet50()
#es = EarlyStopping(monitor= "loss", patience= 3, verbose=1)
annealer = ReduceLROnPlateau(monitor='val_accuracy', factor=0.70, patience=3, verbose=1, min_lr=1e-4)
checkpoint = ModelCheckpoint('model.h5', verbose=1, save_best_only=True)
# 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)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-a
94773248/94765736 [==============================] - 1s 0us/step
94781440/94765736 [==============================] - 1s 0us/step
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 64, 64, 3)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 64, 64, 3)         84
_________________________________________________________________
resnet50 (Functional)        (None, None, None, 2048)  23587712
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
batch_normalization (BatchNo (None, 2048)              8192
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 256)               524544
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
root (Dense)                 (None, 2)                 514
=================================================================
Total params: 24,122,070
Trainable params: 24,064,342
Non-trainable params: 57,728
_________________________________________________________________
/opt/conda/lib/python3.7/site-packages/keras/optimizer_v2/optimizer_v2
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

[17]:

```python
# Fits the model on batches with real-time data augmentation
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))
```

```
2022-09-27 09:52:54.300088: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are
Epoch 1/50
31/31 [==============================] - 68s 2s/step - loss: 1.1983 - accuracy: 0.5207 - val_loss: 0.9307 - val_accuracy: 0.5513

Epoch 00001: val_loss improved from inf to 0.93066, saving model to model.h5
/opt/conda/lib/python3.7/site-packages/keras/utils/generic_utils.py:497: CustomMaskWarning: Custom mask layers require a config and m
layer must be passed to the custom_objects argument.
  category=CustomMaskWarning)
Epoch 2/50
31/31 [==============================] - 58s 2s/step - loss: 0.9818 - accuracy: 0.5862 - val_loss: 1.4836 - val_accuracy: 0.5513

Epoch 00002: val_loss did not improve from 0.93066
Epoch 3/50
31/31 [==============================] - 58s 2s/step - loss: 0.9142 - accuracy: 0.6265 - val_loss: 0.8767 - val_accuracy: 0.4487

Epoch 00003: val_loss improved from 0.93066 to 0.87672, saving model to model.h5
Epoch 4/50
31/31 [==============================] - 58s 2s/step - loss: 0.7981 - accuracy: 0.6431 - val_loss: 18.2905 - val_accuracy: 0.5513

Epoch 00004: ReduceLROnPlateau reducing learning rate to 0.002100000018253922.

Epoch 00004: val_loss did not improve from 0.87672
Epoch 5/50
```

```
Epoch 00048: val_loss did not improve from 0.33933
Epoch 49/50
31/31 [==============================] - 57s 2s/step - loss: 0.4010 - accuracy: 0.8296 - val_loss: 0.3452 - val_accuracy: 0.8431

Epoch 00049: val_loss did not improve from 0.33933
Epoch 50/50
31/31 [==============================] - 59s 2s/step - loss: 0.3773 - accuracy: 0.8448 - val_loss: 0.3346 - val_accuracy: 0.8551

Epoch 00050: ReduceLROnPlateau reducing learning rate to 0.00012106080830562859.

Epoch 00050: val_loss improved from 0.33933 to 0.33456, saving model to model.h5
```

+ Code + Markdown

[18]:
```
model = load_model('./model.h5')
final_loss, final_accuracy = model.evaluate(X_val, Y_val)
print('Final Loss: {}, Final Accuracy: {}'.format(final_loss, final_accuracy))
```

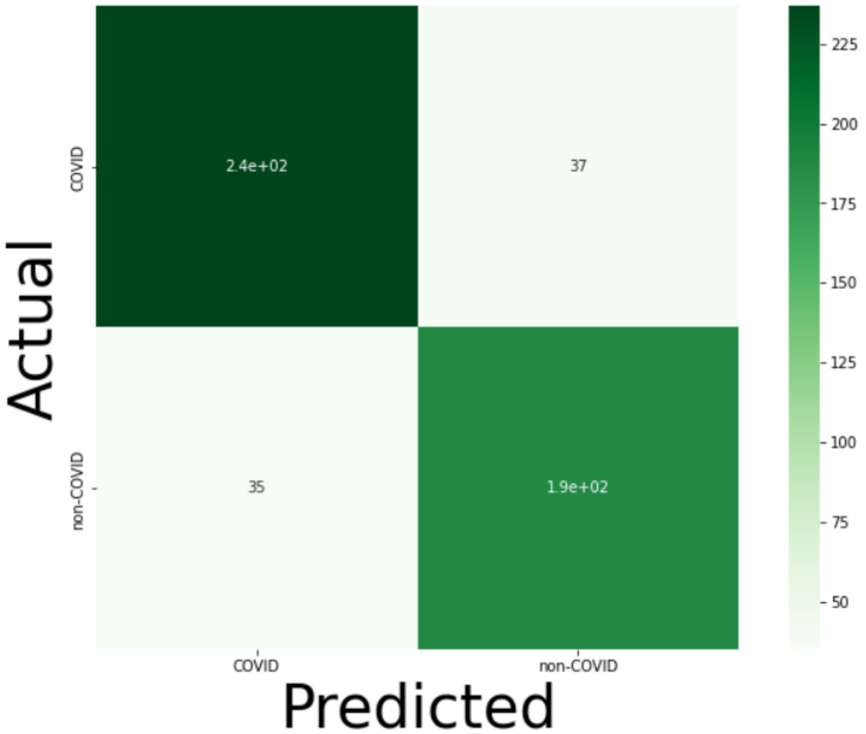
```
16/16 [==============================] - 6s 252ms/step - loss: 0.3346 - accuracy: 0.8551
Final Loss: 0.33455565571784973, Final Accuracy: 0.8551307916641235
```

Final accuracy is 85% which is good

Step 8: Create a confusion matrix and get the actual and predicted values of data

[19]:
```
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, 8))
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)
```

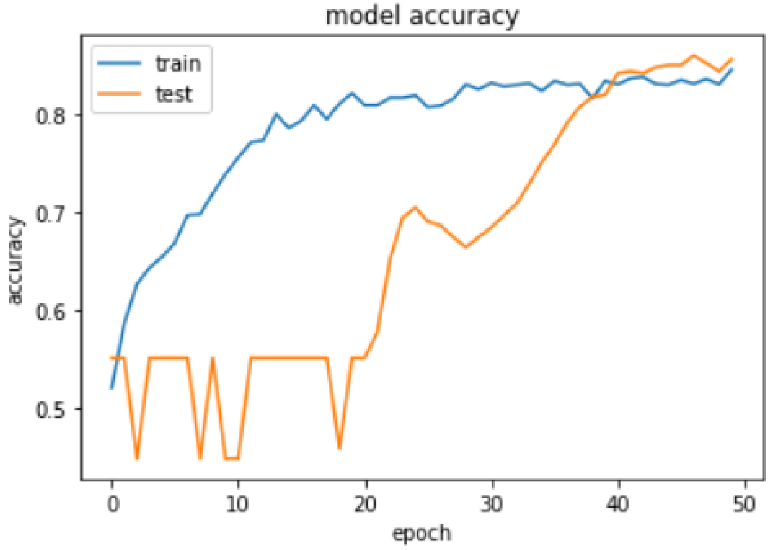
[19]: Text(0.5, 51.0, 'Predicted')

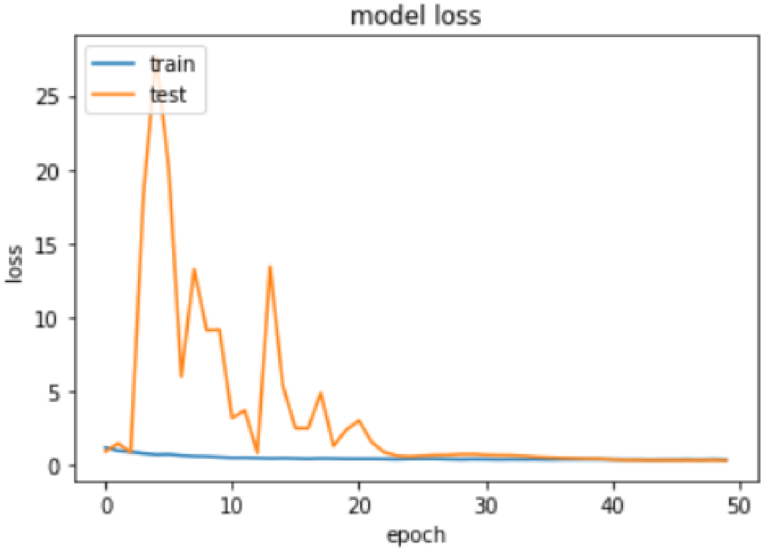


Step 9: Plot accuracy and loss

[20]:

```
# 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()
```





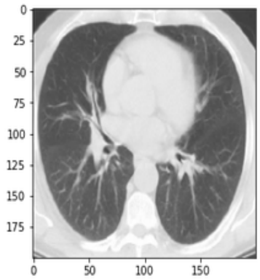
As you can see the accuracy, loss of the model training and testing is similar so it's not overfitting or underfitting

Step 10: Predict weather the image is covid or non covid image

[21]:

```
from skimage import io
from keras.preprocessing import image
img = image.load_img('../input/sarscov2-ctscan-dataset/non-COVID/Non-Covid (100).png', grayscale=False, target_size=(64, 64))
show_img=image.load_img('../input/sarscov2-ctscan-dataset/non-COVID/Non-Covid (100).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])
```

```
[0.43176723 0.5682328 ]
```

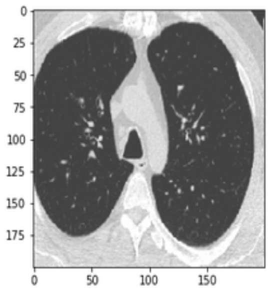



```
Prediction: Non Covid-19
```

[22]:

```
from skimage import io
from keras.preprocessing import image
img = image.load_img('../input/sarscov2-ctscan-dataset/COVID/Covid (100).png', grayscale=False, target_size=(64, 64))
show_img=image.load_img('../input/sarscov2-ctscan-dataset/COVID/Covid (100).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])
```

```
[0.8984409  0.10155904]
```




```
Prediction: Covid-19
```