# CT Scan Image Classification

SCREENSHOTS

ASHWIN SAVANE | FULL STACK DATA SCIENCE (LU) | 4/11/2023

## CT Scan Image Classification

**Dataset information :-**

1. This dataset contains 1252 CT scans that are positive for SARS-CoV-2 infection (COVID-19) and 1230 CT scans for patients non-infected by SARS-CoV-2, 2482 CT scans in total.
2. These data have been collected from real patients in hospitals from Sao Paulo, Brazil.
3. The aim of this dataset is to encourage the research and development of artificial intelligent methods which are able to identify if a person is infected by SARS-CoV-2 through the analysis of his/her CT scans.

**Importing Libraries**

```
[1] !pip install --upgrade pip
    !pip install tensorflow
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.10/dist-packages (23.1.2)
Collecting pip
  Downloading pip-23.3.1-py3-none-any.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 13.7 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.1.2
    Uninstalling pip-23.1.2:
      Successfully uninstalled pip-23.1.2
Successfully installed pip-23.3.1
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.14.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
impor Import "tensorflow.keras.preprocessing.image" could not be resolved (reportMissingImports)
from
impor Loading...
from View Problem (Alt+F8)  Checking for quick fixes...
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import warnings
warnings.filterwarnings("ignore")
```

**1: Data Preprocessing**

**1:1 Loading the images**

```
[3] # Unzip the data.zip file
    import zipfile
    with zipfile.ZipFile('/content/Covid.zip', 'r') as zip_ref:
        zip_ref.extractall('data')
    with zipfile.ZipFile('/content/Non-Covid.zip','r')  as zip_ref:
        zip_ref.extractall('data')

    # Directory containing the extracted data
    data_dir = 'data'
```

✓ 0s completed at 19:41

**1:2 Resizing images and Normalizing image data**

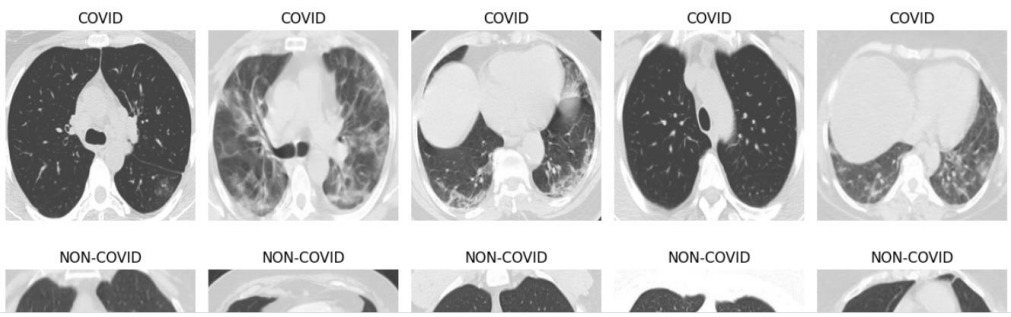
```
# Fixed size of (224, 224) for resizing the images
image_size = (224, 224)

# Converting images to RGB mode and resizing them
covid_images = [img.convert('RGB').resize(image_size) for img in covid_images]
non_covid_images = [img.convert('RGB').resize(image_size) for img in non_covid_images]

# Converting the images to arrays and normalizing the pixel values
covid_data = np.array([img_to_array(img)/255.0 for img in covid_images])
non_covid_data = np.array([img_to_array(img)/255.0 for img in non_covid_images])

# Note:- Here covid_data and non_covid_data now contains the preprocessed and normalized image data.
```

**2: Data Augmentation**

```
# Defining all data augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=20,        # Random rotation (±20 degrees)
    width_shift_range=0.2,    # Random horizontal shift (±20% of image width)
    height_shift_range=0.2,   # Random vertical shift (±20% of image height)
    shear_range=0.2,          # Random shear
    zoom_range=0.2,           # Random zoom
    horizontal_flip=True      # Random horizontal flip
)

# Fitting the data augmentation generator on the data
datagen.fit(covid_data)
datagen.fit(non_covid_data)
```

```
# Display images and count for both Covid and Non Covid class
plt.figure(figsize=(12, 6))
```

```
# Display images and count for both Covid and Non Covid class
plt.figure(figsize=(12, 6))
for i in range(5):
    plt.subplot(2, 5, i+1)
    plt.imshow(covid_images[i])
    plt.title("COVID")
    plt.axis('off')

for i in range(5):
    plt.subplot(2, 5, i+6)
    plt.imshow(non_covid_images[i])
    plt.title("NON-COVID")
    plt.axis('off')

plt.tight_layout()
plt.show()
```




**3: Data Generator & Train Test Split**

```
# Combining COVID and non-COVID data and create labels (0 for non-COVID, 1 for COVID)
X = np.concatenate([non_covid_data, covid_data])
y = np.concatenate([np.zeros(len(non_covid_data)), np.ones(len(covid_data))])

# Spliting data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Creating data generators for training and validation
train_generator = datagen.flow(X_train, y_train, batch_size=32)
val_generator = datagen.flow(X_val, y_val, batch_size=32)
```

+ Code + Text

```
# Shapes of the training and validation sets
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

```
((1984, 224, 224, 3), (497, 224, 224, 3), (1984,), (497,))
```

## 4. Model Building

### 4.1 ResNet50 Model Architecture

```
# Loading the pre-trained ResNet50 model without the top (classification) layers
base_model = ResNet50(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Adding new layers on top of the ResNet50 base model
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(256, activation='relu')(x)
output_layer = Dense(1, activation='sigmoid')(x)

# Creating the final model with our above specified layers
model = Model(inputs=base_model.input, outputs=output_layer)

# Freezing the base ResNet layers to prevent them from being updated during training
for layer in base_model.layers:
    layer.trainable = False

# Compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 0s 0us/step
```

```
# Getting Model summary
model.summary()
```

```
Model: "model"
```

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | [(None, 224, 224, 3)] | 0 | [] |
| conv1_pad (ZeroPadding2D) | (None, 230, 230, 3) | 0 | ['input_1[0][0]'] |
| conv1_conv (Conv2D) | (None, 112, 112, 64) | 9472 | ['conv1_pad[0][0]'] |
| conv1_bn (BatchNormalization) | (None, 112, 112, 64) | 256 | ['conv1_conv[0][0]'] |
| conv1_relu (Activation) | (None, 112, 112, 64) | 0 | ['conv1_bn[0][0]'] |
| pool1_pad (ZeroPadding2D) | (None, 114, 114, 64) | 0 | ['conv1_relu[0][0]'] |
| pool1_pool (MaxPooling2D) | (None, 56, 56, 64) | 0 | ['pool1_pad[0][0]'] |
| conv2_block1_1_conv (Conv2D) | (None, 56, 56, 64) | 4160 | ['pool1_pool[0][0]'] |
| conv2_block1_1_bn (BatchNormalization) | (None, 56, 56, 64) | 256 | ['conv2_block1_1_conv[0][0]'] |
| conv2_block1_1_relu (Activation) | (None, 56, 56, 64) | 0 | ['conv2_block1_1_bn[0][0]'] |
| conv2_block1_2_conv (Conv2D) | (None, 56, 56, 64) | 36928 | ['conv2_block1_1_relu[0][0]'] |
| conv2_block1_2_bn (BatchNormalization) | (None, 56, 56, 64) | 256 | ['conv2_block1_2_conv[0][0]'] |
| conv2_block1_2_relu (Activation) | (None, 56, 56, 64) | 0 | ['conv2_block1_2_bn[0][0]'] |

```
rmalization)

conv5_block3_1_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block3_1_bn[0][0]']
ation)

conv5_block3_2_conv (Conv2  (None, 7, 7, 512)           2359808   ['conv5_block3_1_relu[0][0]']
D)

conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block3_2_conv[0][0]']
rmalization)

conv5_block3_2_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block3_2_bn[0][0]']
ation)

conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)          1050624   ['conv5_block3_2_relu[0][0]']
D)

conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)          8192      ['conv5_block3_3_conv[0][0]']
rmalization)

conv5_block3_add (Add)      (None, 7, 7, 2048)          0         ['conv5_block2_out[0][0]',
                                                                   'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activati  (None, 7, 7, 2048)          0         ['conv5_block3_add[0][0]']
on)

global_average_pooling2d (  (None, 2048)                0         ['conv5_block3_out[0][0]']
GlobalAveragePooling2D)

dense (Dense)               (None, 256)                 524544    ['global_average_pooling2d[0][
                                                                   0]']

dense_1 (Dense)             (None, 1)                   257       ['dense[0][0]']

==================================================================================================
Total params: 24112513 (91.98 MB)
Trainable params: 524801 (2.00 MB)
Non-trainable params: 23587712 (89.98 MB)
__________________________________________________________________________________________________
```

**5: Model Training**

**5.1: Training the model**

```
# Define callbacks for early stopping and model checkpoint
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)

# Train the model using the data generators
history = model.fit(
    train_generator,
    steps_per_epoch=len(X_train) // 32,
    epochs=50,
    validation_data=val_generator,
    validation_steps=len(X_val) // 32,
    callbacks=[early_stopping, model_checkpoint]
)
```

```
Epoch 1/50
62/62 [==============================] - 49s 585ms/step - loss: 0.7089 - accuracy: 0.5307 - val_loss: 0.7147 - val_accuracy: 0.4896
Epoch 2/50
62/62 [==============================] - 35s 567ms/step - loss: 0.6939 - accuracy: 0.5746 - val_loss: 0.6841 - val_accuracy: 0.5604
Epoch 3/50
62/62 [==============================] - 35s 567ms/step - loss: 0.6910 - accuracy: 0.5559 - val_loss: 0.6697 - val_accuracy: 0.6250
Epoch 4/50
62/62 [==============================] - 34s 553ms/step - loss: 0.6684 - accuracy: 0.5852 - val_loss: 0.7174 - val_accuracy: 0.5042
Epoch 5/50
62/62 [==============================] - 36s 586ms/step - loss: 0.6850 - accuracy: 0.5660 - val_loss: 0.6557 - val_accuracy: 0.6104
Epoch 6/50
62/62 [==============================] - 31s 504ms/step - loss: 0.6729 - accuracy: 0.5766 - val_loss: 0.6797 - val_accuracy: 0.5750
Epoch 7/50
62/62 [==============================] - 34s 544ms/step - loss: 0.6772 - accuracy: 0.5751 - val_loss: 0.6875 - val_accuracy: 0.5521
Epoch 8/50
62/62 [==============================] - 35s 573ms/step - loss: 0.6652 - accuracy: 0.5973 - val_loss: 0.6514 - val_accuracy: 0.5958
Epoch 9/50
```

```
[13] # Loading the best saved model from model checkpoint
     best_model = tf.keras.models.load_model('best_model.h5')

     # Evaluating the model on the validation set
     val_loss, val_accuracy = best_model.evaluate(X_val, y_val, batch_size=32)

     print("Validation Loss:", val_loss*100)
     print("Validation Accuracy:", val_accuracy*100)


     # Evaluate the model on the training set
     train_loss, train_accuracy = best_model.evaluate(X_train, y_train, batch_size=32)

     print("train Loss:", train_loss*100)
     print("train Accuracy:", train_accuracy*100)
```

```
16/16 [==============================] - 3s 145ms/step - loss: 0.6077 - accuracy: 0.6761
Validation Loss: 60.77374219894409
Validation Accuracy: 67.60563254356384
62/62 [==============================] - 6s 95ms/step - loss: 0.6066 - accuracy: 0.6714
train Loss: 60.66303849220276
train Accuracy: 67.13709831237793
```

**6: Model Evaluation and Prediction**

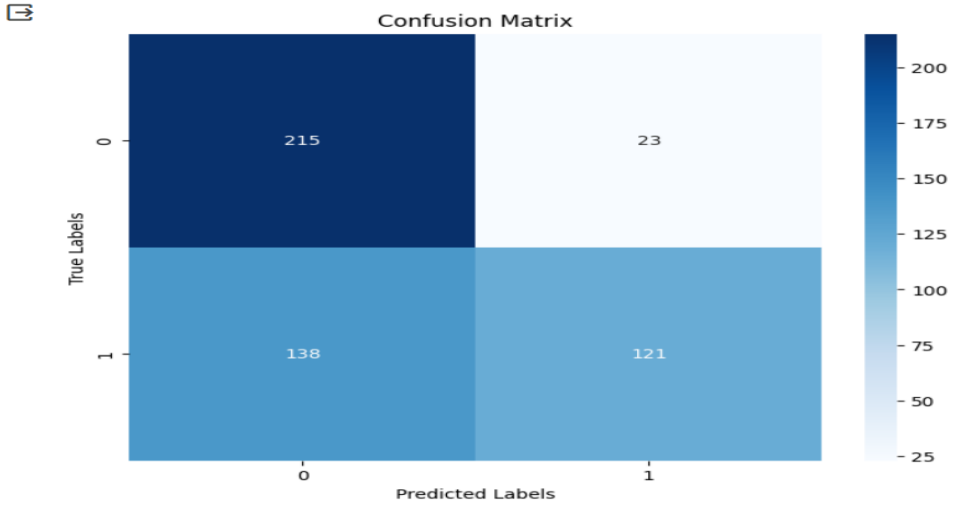
**6.1 Model Evaluation:**

```
# Load the best saved model
best_model = tf.keras.models.load_model('best_model.h5')

# Evaluate the model on the tes
y_pred = best_model.predict(X_val)
y_pred_binary = (y_pred > 0.5).astype(int)

# Calculate performance metrics
accuracy = accuracy_score(y_val, y_pred_binary)
precision = precision_score(y_val, y_pred_binary)
recall = recall_score(y_val, y_pred_binary)
f1 = f1_score(y_val, y_pred_binary)
conf_matrix = confusion_matrix(y_val, y_pred_binary)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

Loading...

```
16/16 [==============================] - 2s 93ms/step
Accuracy: 0.676056338028169
Precision: 0.8402777777777778
Recall: 0.4671814671814672
F1 Score: 0.6004962779156328
Confusion Matrix:
 [[215  23]
 [138 121]]
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

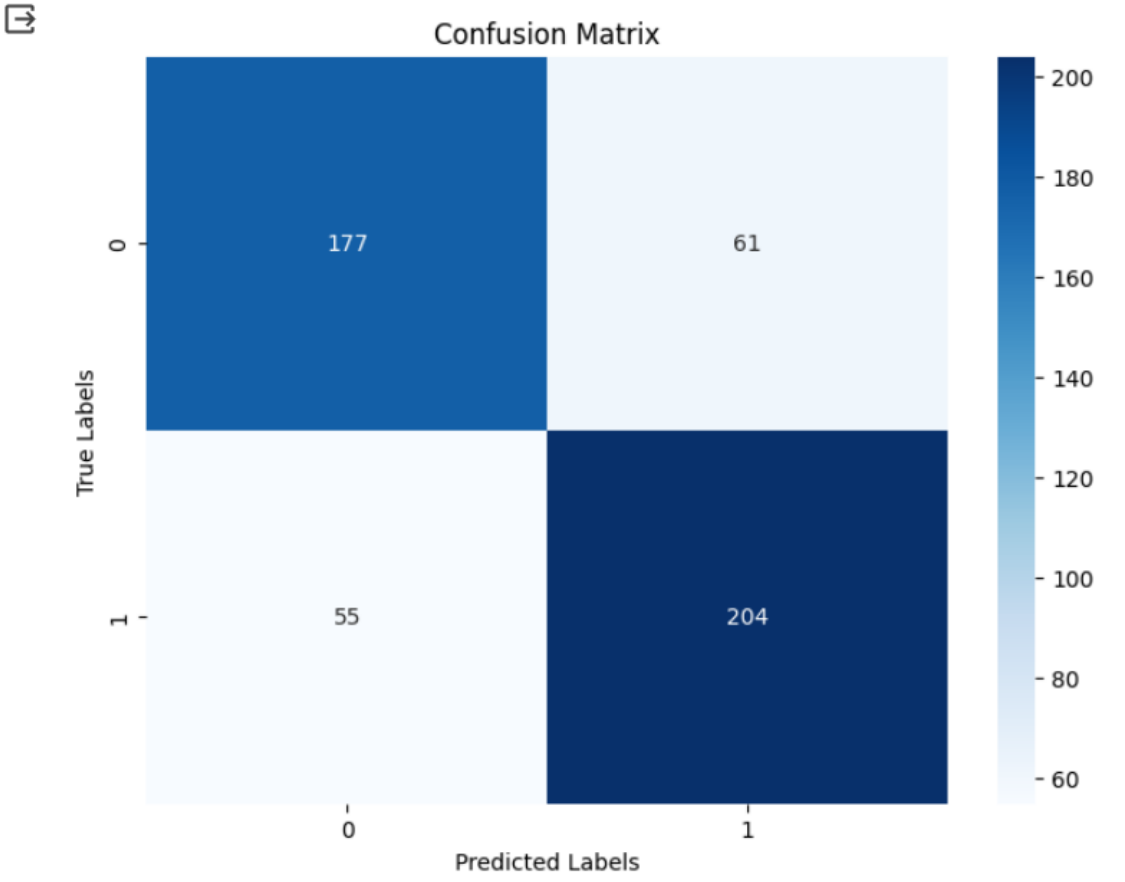


## 7. Fine Tuning our Best Model

```
[16] # Unfreezing last 20 layers for fine-tuning
     for layer in base_model.layers[-20:]:
         layer.trainable = True

     # Compiling the model again after unfreezing layers
     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

     # Train the model with fine-tuning
     fine_tune_epochs = 60
     history_fine = model.fit(
         datagen.flow(X_train, y_train, batch_size=32),
         steps_per_epoch=len(X_train) // 32,
         epochs=fine_tune_epochs,
         validation_data=datagen.flow(X_val, y_val, batch_size=32),
         validation_steps=len(X_val) // 32,
         callbacks=[early_stopping, model_checkpoint]
     )

     # Evaluate and predict as before
     best_fine_tuned_model = load_model('best_model.h5')

     y_pred_fine = best_fine_tuned_model.predict(X_val)
     y_pred_binary_fine = (y_pred_fine > 0.5).astype(int)

     accuracy_fine = accuracy_score(y_val, y_pred_binary_fine)
     precision_fine = precision_score(y_val, y_pred_binary_fine)
     recall_fine = recall_score(y_val, y_pred_binary_fine)
     f1_fine = f1_score(y_val, y_pred_binary_fine)
     conf_matrix_fine = confusion_matrix(y_val, y_pred_binary_fine)

     print("Accuracy (Fine-Tuned):", accuracy_fine*100)
     print("Precision (Fine-Tuned):", precision_fine*100)
     print("Recall (Fine-Tuned):", recall_fine)
```

```
Epoch 1/60
62/62 [==============================] - 46s 586ms/step - loss: 0.7921 - accuracy: 0.5323 - val_loss: 1.3981 - val_accuracy: 0.4854
Epoch 2/60
62/62 [==============================] - 35s 574ms/step - loss: 0.6631 - accuracy: 0.6028 - val_loss: 2.3008 - val_accuracy: 0.4750
Epoch 3/60
62/62 [==============================] - 30s 492ms/step - loss: 0.6453 - accuracy: 0.6300 - val_loss: 0.7276 - val_accuracy: 0.6250
Epoch 4/60
62/62 [==============================] - 35s 568ms/step - loss: 0.6047 - accuracy: 0.6532 - val_loss: 11.6483 - val_accuracy: 0.5167
Epoch 5/60
62/62 [==============================] - 30s 486ms/step - loss: 0.5954 - accuracy: 0.6739 - val_loss: 1.0878 - val_accuracy: 0.6375
Epoch 6/60
62/62 [==============================] - 34s 558ms/step - loss: 0.5818 - accuracy: 0.6900 - val_loss: 0.5830 - val_accuracy: 0.7250
Epoch 7/60
62/62 [==============================] - 35s 571ms/step - loss: 0.5604 - accuracy: 0.7077 - val_loss: 0.7939 - val_accuracy: 0.6833
Epoch 8/60
62/62 [==============================] - 35s 569ms/step - loss: 0.5489 - accuracy: 0.7092 - val_loss: 0.9997 - val_accuracy: 0.5500
Epoch 9/60
62/62 [==============================] - 30s 492ms/step - loss: 0.5568 - accuracy: 0.7072 - val_loss: 2.2351 - val_accuracy: 0.6042
Epoch 10/60
62/62 [==============================] - 35s 569ms/step - loss: 0.5332 - accuracy: 0.7324 - val_loss: 0.9328 - val_accuracy: 0.5833
Epoch 11/60
62/62 [==============================] - 36s 574ms/step - loss: 0.5270 - accuracy: 0.7334 - val_loss: 2.5325 - val_accuracy: 0.4812
16/16 [==============================] - 2s 94ms/step
Accuracy (Fine-Tuned): 76.65995975855131
Precision (Fine-Tuned): 76.9811320754717
Recall (Fine-Tuned): 0.7876447876447876
F1 Score (Fine-Tuned): 0.7786259541984734
Confusion Matrix (Fine-Tuned):
 [[177  61]
 [ 55 204]]
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_fine, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



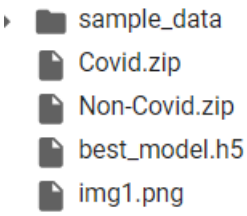

**Observations**

- After training our ResNet50 model with additional layers like Pooling layers and Dense layers we got an accuracy of 65.99%
- After fine tuning our Best Model, we received an accuracy of 77.86%.

```
def predict_ct_scan_class(test_image_path, model_path='best_model.h5'):
    # Loading the test image
    test_image = Image.open(test_image_path)

    # Performing preprocessing on image
    if test_image.mode == 'RGBA':
        test_image = test_image.convert('RGB')

    # Resize and normalize the test image
    image_size = (224, 224)
    test_image = test_image.resize(image_size)
    test_image_array = np.array(test_image) / 255.0
    test_image_array = np.expand_dims(test_image_array, axis=0)

    # Loading the trained model
    best_model = load_model(model_path)

    # Making the prediction
    prediction = best_model.predict(test_image_array)
    predicted_class = "COVID" if prediction[0][0] > 0.5 else "NON-COVID"

    return predicted_class
```

```
[21] test_image_path = '/content/img1.png'
     predicted_class = predict_ct_scan_class(test_image_path)

     print("Predicted class:", predicted_class)
```

```
1/1 [==============================] - 1s 1s/step
Predicted class: COVID
```




# THANK YOU!!!😍