National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Amal Sunny
Student ID: x22169806

School of Computing
National College of Ireland

Supervisor: Harshani Nagahamulla

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name	Amal Sunny
Student ID	X22169806
Programme	MSc in Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Harshani Nagahamulla
Submission Due Date:	14/12/2023
Project Title:	Deep Learning-Based Lung Cancer Detection and Classification from Medical Images
Word Count:	919
Page Count:	24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature	Amal Sunny
Date	13/12/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Amal Sunny
Student ID: x22169806

1 Introduction

This Configuration Manual lists together all prerequisites needed to duplicate the studies and their effects on a specific setting. A glimpse of the source for Data Importing and classifying data as per categories and after those images are augmented and prediction algorithms are built for detection of class.
The report is organized as follows, with details relating to environment configuration provided in Section 2. Information about data gathering is detailed in Section 3. Data Classification is done in Section 4. Image Segmentation is included in Section 5. In section 6, the Image Augmentation is described. Details well about models that were created and tested are provided in Section 7. How the results are calculated and shown is described in Section 8.

2 System Requirements

The specific needs for hardware as well as software to put the research into use are detailed in this section.

2.1 Hardware Requirements

The necessary hardware specs are shown in Figure 1 below.

Installed RAM	8.00 GB
Device ID	3950356C-7150-4109-85F9-F565A7A70EFF
Product ID	00331-10000-00001-AA229
System type	64-bit operating system, x64-based processor
Edition	Windows 10 Pro
Version	22H2

Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda 3 (Version 4.8.0)- [Can be downloaded from https://www.anaconda.com/download]
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.7.6)

2.3 Code Execution

The code can be run in Jupyter Notebook. The Jupyter Notebook comes with Anaconda 3, run the Jupyter Notebook from startup. This will open Jupyter Notebook in web browser. The web browser will show the folder structure of the system, move to the folder where the code file is located. Open the code file from the folder and to run the code, go to the Kernel menu and run all cells.

3 Data Collection

The dataset is taken from Kaggle public repository from the link https://www.kaggle.com/datasets/adityamahimkar/iqothnccd-lung-cancer-dataset. The Iraq-Oncology Teaching Hospital/National Center for Cancer Diseases (IQ-OTH/NCCD) lung cancer dataset includes CT scans of patients diagnosed with lung cancer in different stages, as well as healthy subjects.

4 Data Classification

Figure 2 includes a list of every Python library necessary to complete the project.

```
import numpy as np
import pandas as pd
import glob
import shutil
import matplotlib.pyplot as plt
import os
import cv2
from sklearn import preprocessing
import random
from sklearn.utils import class_weight
import tensorflow
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization, Flatten, Conv2D,MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv1D, Layer,Attention, GlobalAveragePooling2D
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19
from keras.applications.inception_v3 import InceptionV3
from keras.applications.efficientnet_v2 import EfficientNetV2B0
from keras.applications.densenet import DenseNet121
```

Figure 2: Necessary Python libraries

The Figure 3, illustrate the code to set data path and set image size and categories and the check count of images in each category.

```
bengin = './archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Bengin cases/'
malignant = './archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Malignant cases/'
normal = './archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Normal cases/'
```

```
categories = ['Bengin', 'Malignant', 'Normal']
print(categories)
```

```
['Bengin', 'Malignant', 'Normal']
```

```
#initialization and importing for data analysi
count_bengin=len(os.listdir(bengin))
count_malignant=len(os.listdir(malignant))
count_normal=len(os.listdir(normal))
count_bengin, count_malignant, count_normal
```

```
(120, 561, 416)
```

Figure 3: Data Path

The Figure 4, illustrate bar plot of the value counts.

```
#plotting graph for cancer cell types count
fig = plt.figure(figsize = (10, 5))
values=[count_bengin, count_malignant, count_normal]
#creating the bar plot
plt.bar(categories, values, color ='blue', width = 0.4)

plt.xlabel("Classes")
plt.ylabel("Number of Images")
plt.show()
```

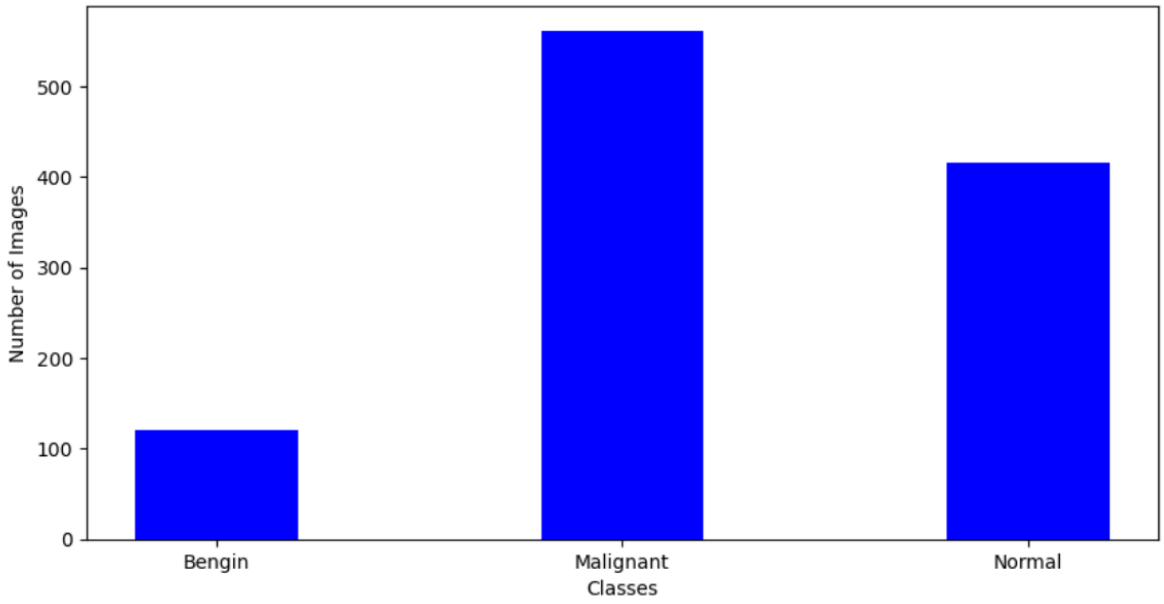

Figure 4: Data Classes

The Figure 5, illustrate the generate list of images for each class.

```
bengin = glob.glob(bengin+ "*.jpg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(bengin))
print ('\n'.join(bengin[:5]))
```

```
Total of 120  images.
First 5 filenames:
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Bengin cases\Bengin case (1).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Bengin cases\Bengin case (10).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Bengin cases\Bengin case (100).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Bengin cases\Bengin case (101).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Bengin cases\Bengin case (102).jpg
```

```
malignant = glob.glob(malignant + "*.jpg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(malignant))
print ('\n'.join(malignant[:5]))
```

```
Total of 561  images.
First 5 filenames:
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Malignant cases\Malignant case (1).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Malignant cases\Malignant case (10).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Malignant cases\Malignant case (100).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Malignant cases\Malignant case (101).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Malignant cases\Malignant case (102).jpg
```

```
normal = glob.glob(normal + "*.jpg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(normal))
print ('\n'.join(normal[:5]))
```

```
Total of 416  images.
First 5 filenames:
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Normal cases\Normal case (1).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Normal cases\Normal case (10).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Normal cases\Normal case (100).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Normal cases\Normal case (101).jpg
./archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/Normal cases\Normal case (102).jpg
```

Figure 5: Images in each class

5 Image Segmentation

The Figure 6, illustrate the read the image and show image shape and plot image.

```
def plot_image(img, cmap='gray'):
    fig = plt.figure(figsize=(8,8))
    axes = fig.add_subplot(111)
    axes.imshow(img, cmap=cmap)
```

```
img1 = cv2.imread(bengin[1])
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
plot_image(img1)
width, height, dimension = img1.shape
print(f'Width RGB = {width}')
print(f'Height RGB = {height}')
print(f'Dimension RGB = {dimension}')
```

```
Width RGB = 512
Height RGB = 512
Dimension RGB = 3
```

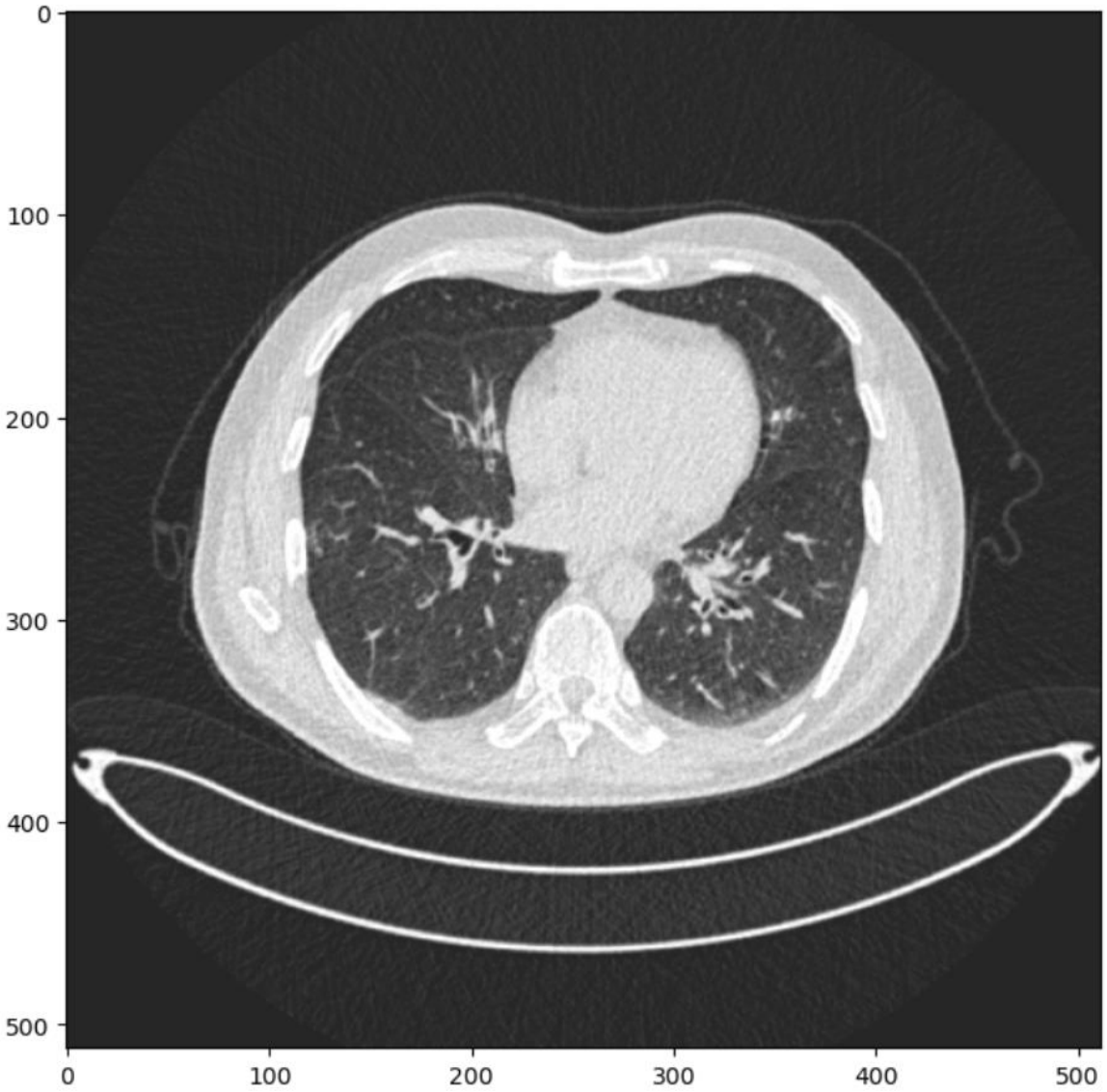

Figure 6: Read Image

Figures 7 show the code used read the image in gray scale.

```
img1_gray = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
plot_image(img1_gray)
width, height = img1_gray.shape
print(f'Width Grayscale = {width}')
print(f'Height Grayscale = {height}')
print(f'Image Shape Grayscale {img1_gray.shape}')
```

```
Width Grayscale = 512
Height Grayscale = 512
Image Shape Grayscale (512, 512)
```

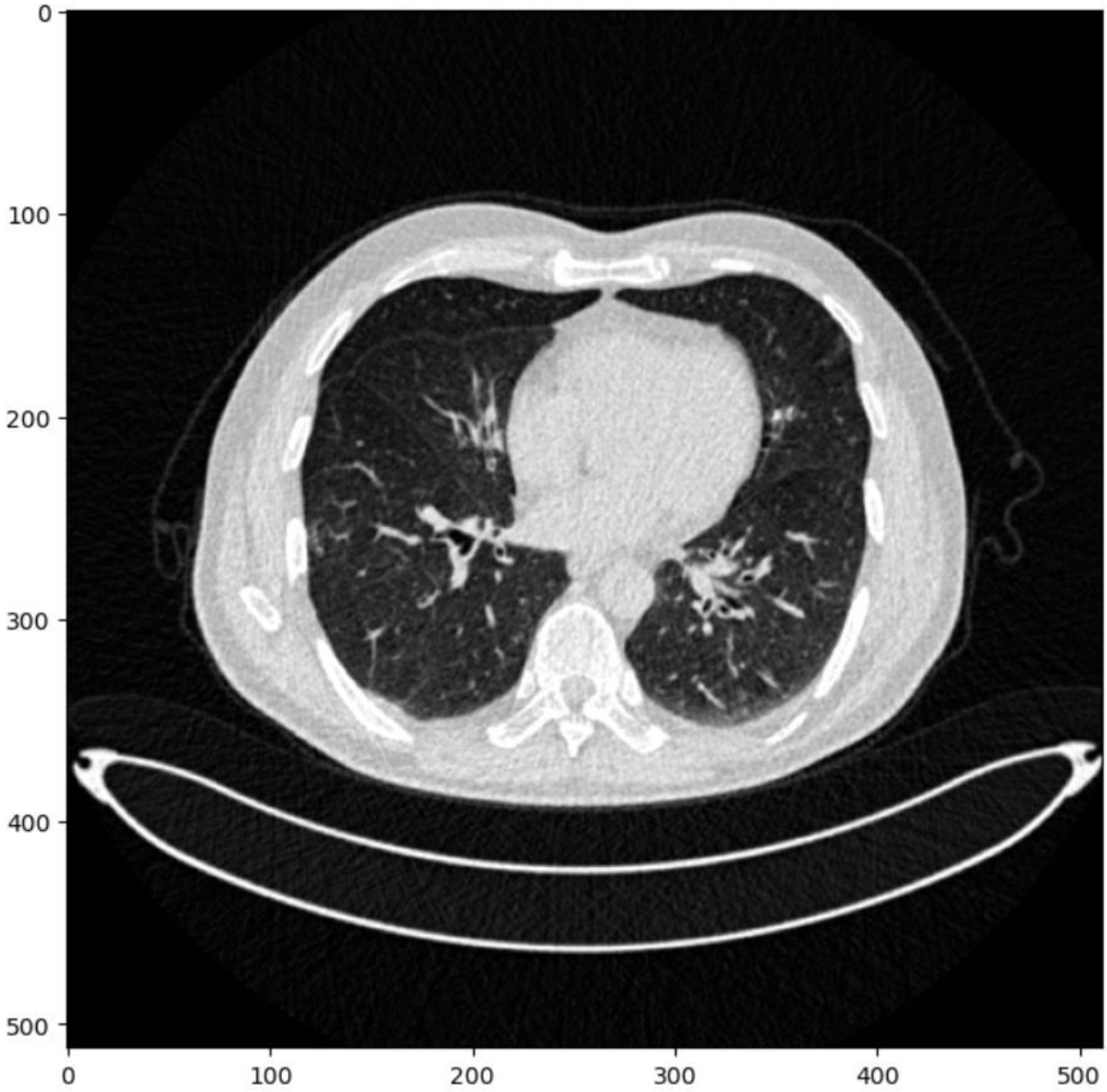

Figure 7: Read Grayscale Image

The Figure 8, illustrate the code to generate adaptive thresholds of the images.

```
img1_gray = cv2.adaptiveThreshold(img1_gray,5,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,11,3)
plot_image(img1_gray)
```

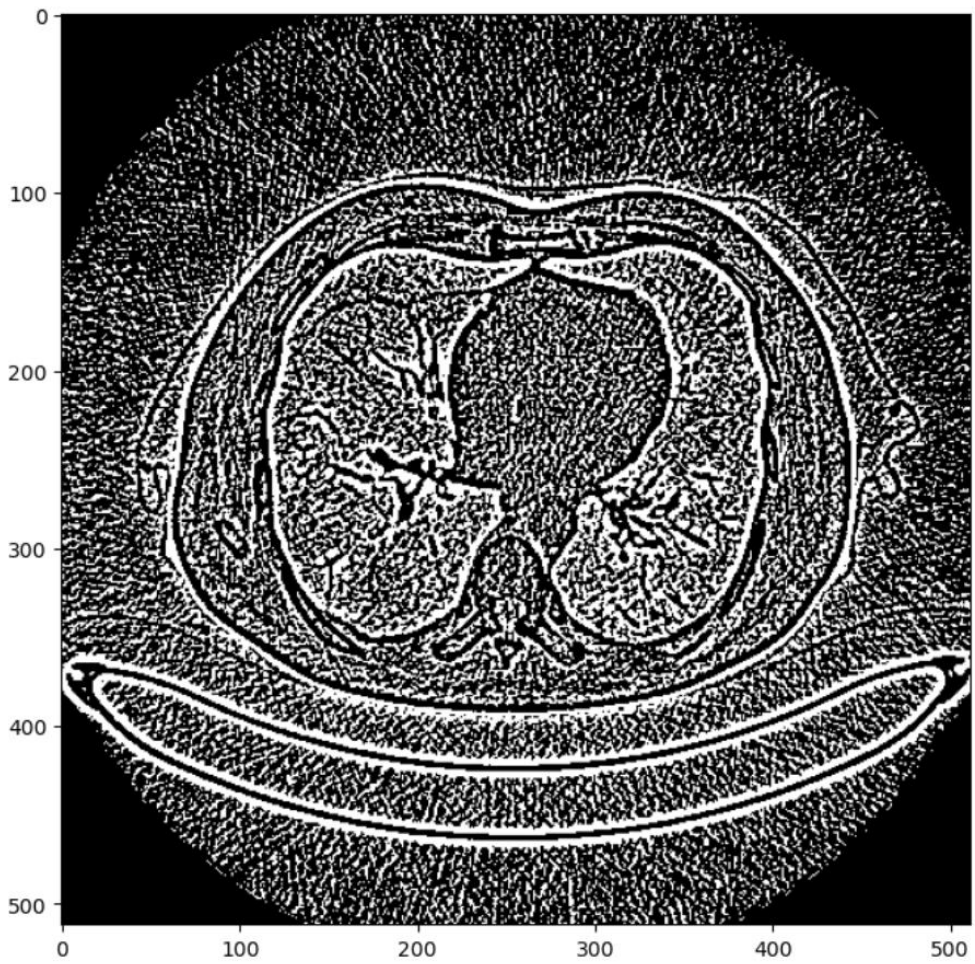

Figure 8: Adaptive threshold

The Figure 9, illustrate image contours.

```
contours = cv2.findContours(img1_gray, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = contours[0] if len(contours) == 2 else contours[1]
contours = sorted(contours, key=cv2.contourArea, reverse=True)
for c in contours:
    x,y,w,h = cv2.boundingRect(c)
    img1_ROI = img1[y:y+h, x:x+w]
    break
plot_image(img1_ROI)
width, height, dimension = img1_ROI.shape
print(f'Width = {width}')
print(f'Height = {height}')
print(f'Dimension = {dimension}')
```

```
Width = 414
Height = 512
Dimension = 3
```

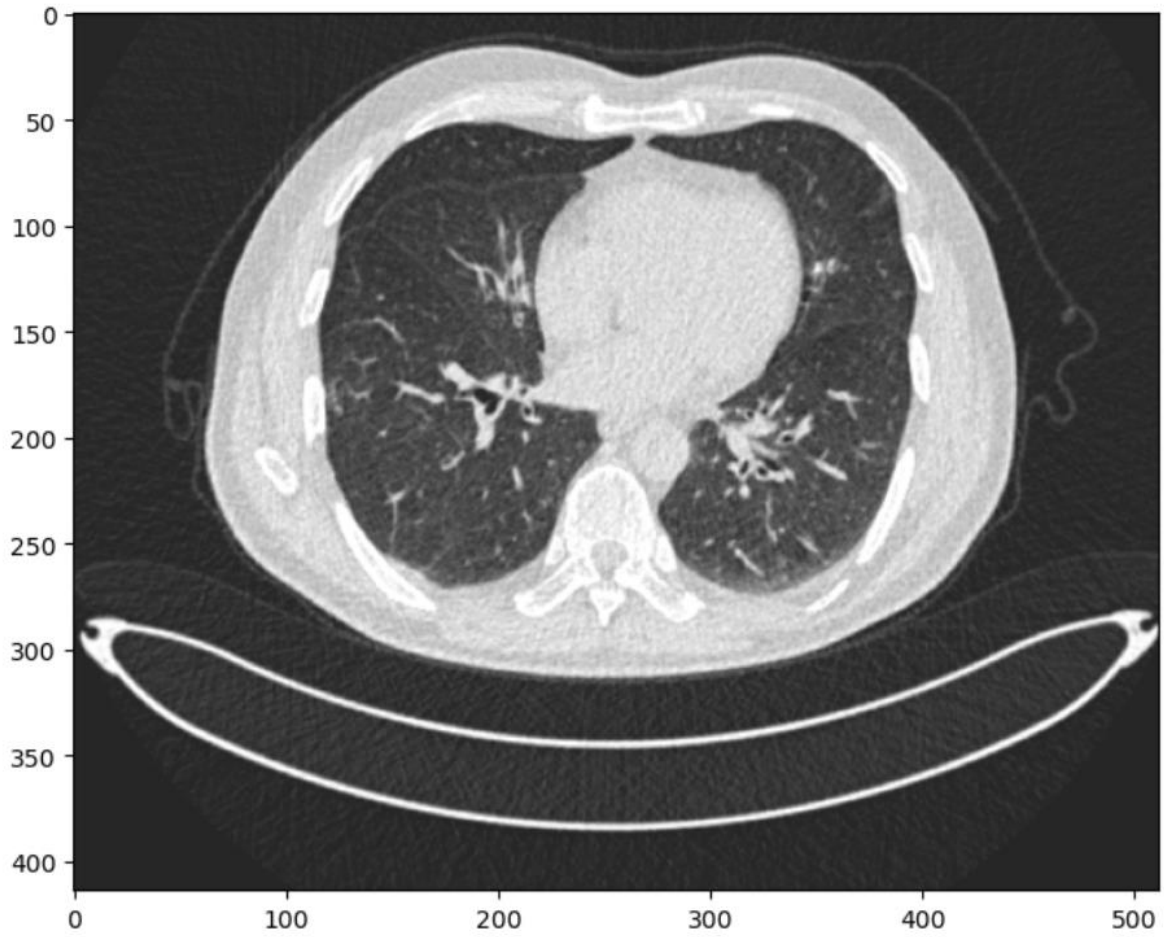

Figure 9: Image contouring

6 Image Augmentation

The Figure 10, illustrate the code to use Image Data Generator to generate augmented images for the deep learning models.

```
path = './archive/The IQ-OTHNCCD lung cancer dataset/The IQ-OTHNCCD lung cancer dataset/'
img_size = 328
```

```
gen = ImageDataGenerator(rescale=1./255)
```

```
train = gen.flow_from_directory(directory=path, target_size=(img_size,img_size), class_mode='sparse')
```

```
Found 1097 images belonging to 3 classes.
```

```
# This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 4, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()

augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

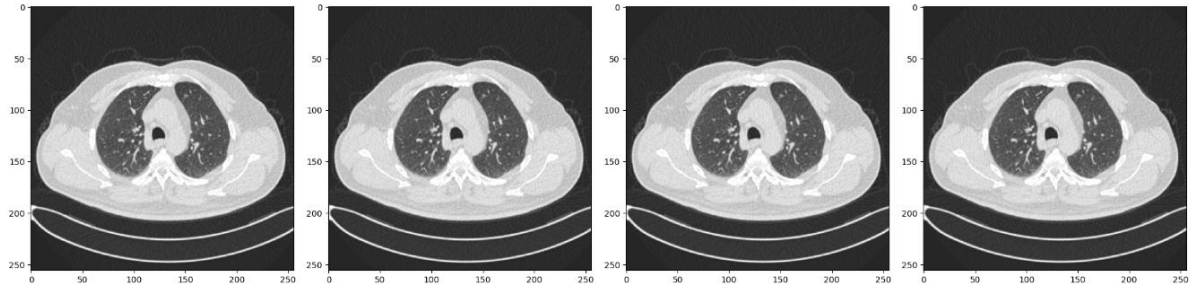

Figure 10: Image Data Generator

Figures 11 show the code to create data with different height and width shift to check for appropriate size.

```
gen = ImageDataGenerator(rescale=1./255,  width_shift_range=0.2, height_shift_range=0.3)
train = gen.flow_from_directory(directory=path, target_size=(img_size,img_size), class_mode='sparse')
```

```
Found 1097 images belonging to 3 classes.
```

```
augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

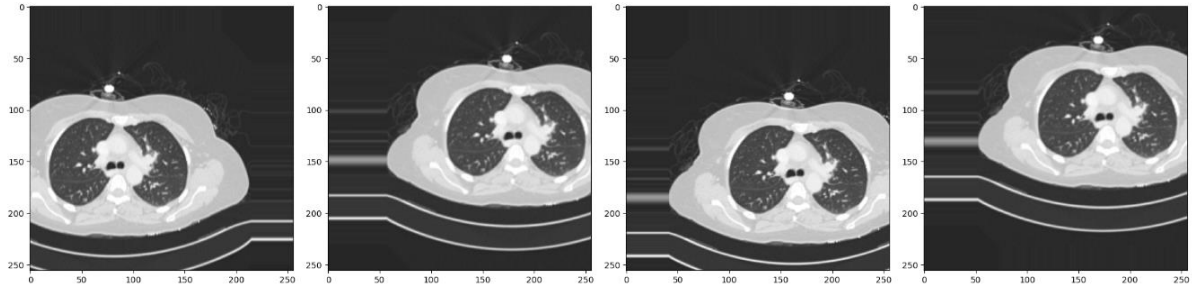

Figure 11: Image Data Generator

Figures 12 show the code to create data with rescaling the images.

```
gen = ImageDataGenerator(rescale=1./255)
train = gen.flow_from_directory(directory=path, target_size=(img_size,img_size))
```

```
Found 1097 images belonging to 3 classes.
```

```
augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

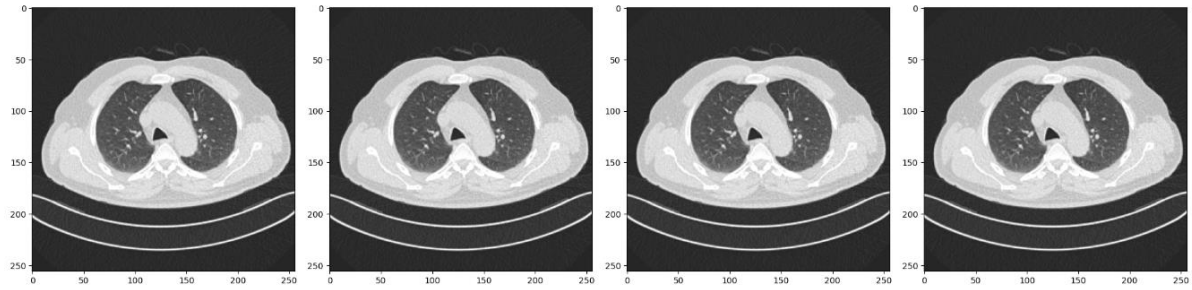

Figure 12: Image Data Generator

The Figure 13-14, illustrate the code to data and doing validation split to generate train and test data.

```
batch_size = 32
datagen= ImageDataGenerator(rescale=1./255, validation_split=0.2)
```

```
train = datagen.flow_from_directory(path, target_size=(img_size, img_size), batch_size=batch_size, subset='training', shuffle=Tru
        classes=['Bengin cases', 'Malignant cases', 'Normal cases'])
augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

```
Found 878 images belonging to 3 classes.
```

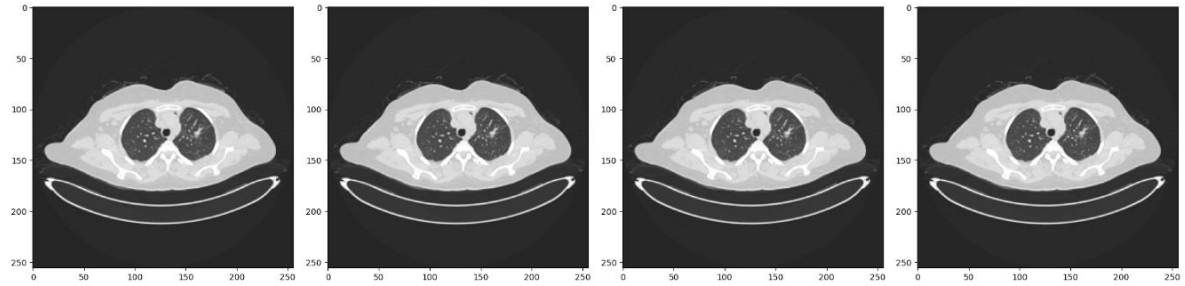

Figure 13: Image Data Generator

```
test = datagen.flow_from_directory(path, target_size=(img_size, img_size), batch_size=batch_size, subset='validation',
        shuffle=True, classes=['Bengin cases', 'Malignant cases', 'Normal cases'])
augmented_images = [test[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

```
Found 219 images belonging to 3 classes.
```

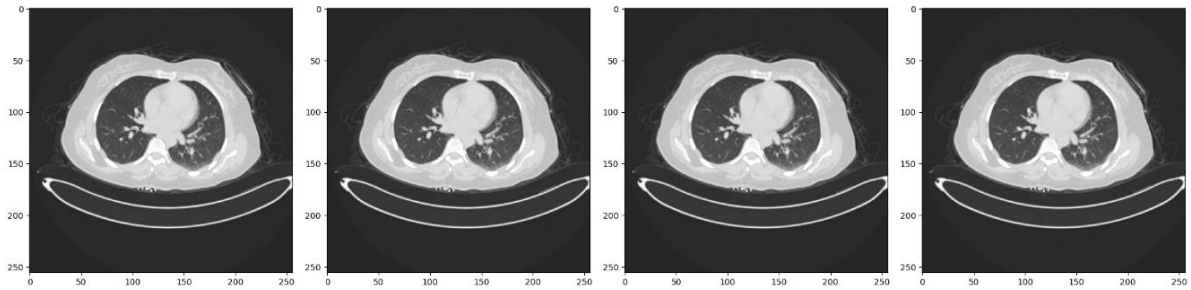

Figure 14: Image Data Generator

7 Machine Learning Models

7.1 CNN

```
model = Sequential()
model.add(Conv2D(128, (3, 3), activation='relu', input_shape=(img_size, img_size, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.4))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.4))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(3, activation='sigmoid'))
model.compile(optimizer='adam', loss='categorical_crossentropy',  metrics=['accuracy'])
```

```
callback = EarlyStopping( monitor='val_accuracy', patience=1, verbose=1, mode='max')
history = model.fit(train, epochs=10, validation_data=test, shuffle = True, callbacks=[callback])
```

```
Epoch 1/10
28/28 [==============================] - 290s 10s/step - loss: 1.6148 - accuracy: 0.5456 - val_loss: 1.0259 - val_accuracy: 0.5
753
Epoch 2/10
28/28 [==============================] - 256s 9s/step - loss: 0.6741 - accuracy: 0.7198 - val_loss: 0.9326 - val_accuracy: 0.53
42
Epoch 2: early stopping
```

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Categorical Crossentropy')
ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
ax1.grid()
ax1.legend()
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
ax2.grid()
ax2.legend()
```

```
<matplotlib.legend.Legend at 0x2669be31f90>
```

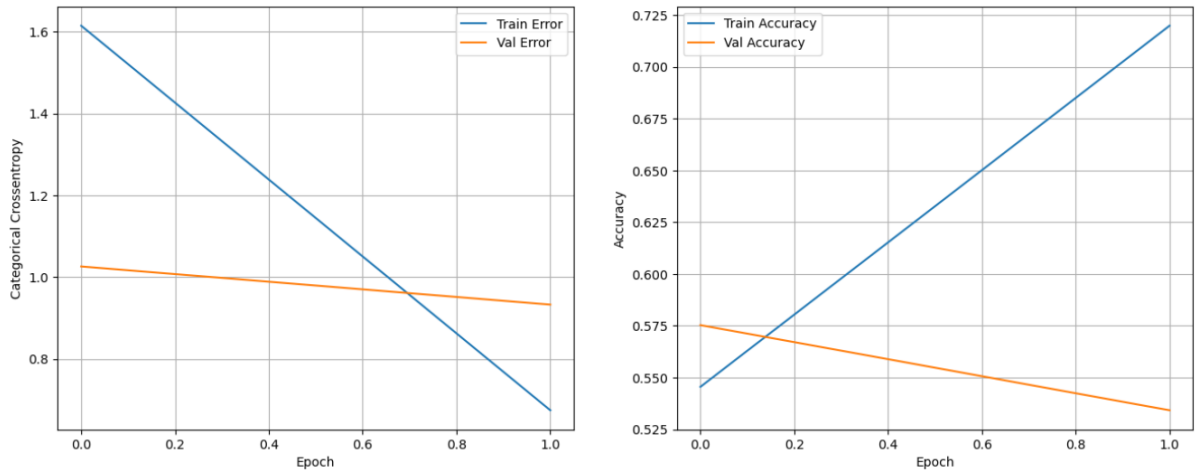

Figure 15: Implementation of CNN

7.2 InceptionNet

```
# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(img_size, img_size, 3))
```

```
# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(3, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
    layer.trainable = False
```

```
model.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
 input_2 (InputLayer)        [(None, 256, 256, 3)]     0         []

 conv2d_100 (Conv2D)         (None, 127, 127, 32)      864       ['input_2[0][0]']

 batch_normalization_94 (Ba  (None, 127, 127, 32)      96        ['conv2d_100[0][0]']
 tchNormalization)

 activation_94 (Activation)  (None, 127, 127, 32)      0         ['batch_normalization_94[0][0]
                                                                 ']

 conv2d_101 (Conv2D)         (None, 125, 125, 32)      9216      ['activation_94[0][0]']

 batch_normalization_95 (Ba  (None, 125, 125, 32)      96        ['conv2d_101[0][0]']
 tchNormalization)

 activation_95 (Activation)  (None, 125, 125, 32)      0         ['batch_normalization_95[0][0]
                                                                 ']
```

```
callback = EarlyStopping( monitor='val_accuracy', patience=1, verbose=1, mode='max')
history = model.fit(train, epochs=10, validation_data=test, shuffle = True, callbacks=[callback])
```

```
Epoch 1/10
28/28 [==============================] - 102s 4s/step - loss: 0.2196 - accuracy: 0.9100 - val_loss: 0.7578 - val_accuracy: 0.73
97
Epoch 2/10
28/28 [==============================] - 108s 4s/step - loss: 0.1621 - accuracy: 0.9419 - val_loss: 0.6911 - val_accuracy: 0.75
34
Epoch 3/10
28/28 [==============================] - 115s 4s/step - loss: 0.1348 - accuracy: 0.9510 - val_loss: 0.7418 - val_accuracy: 0.73
97
Epoch 3: early stopping
```

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Categorical Crossentropy')
ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
ax1.grid()
ax1.legend()
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
ax2.grid()
ax2.legend()
```

```
<matplotlib.legend.Legend at 0x266b6aab190>
```

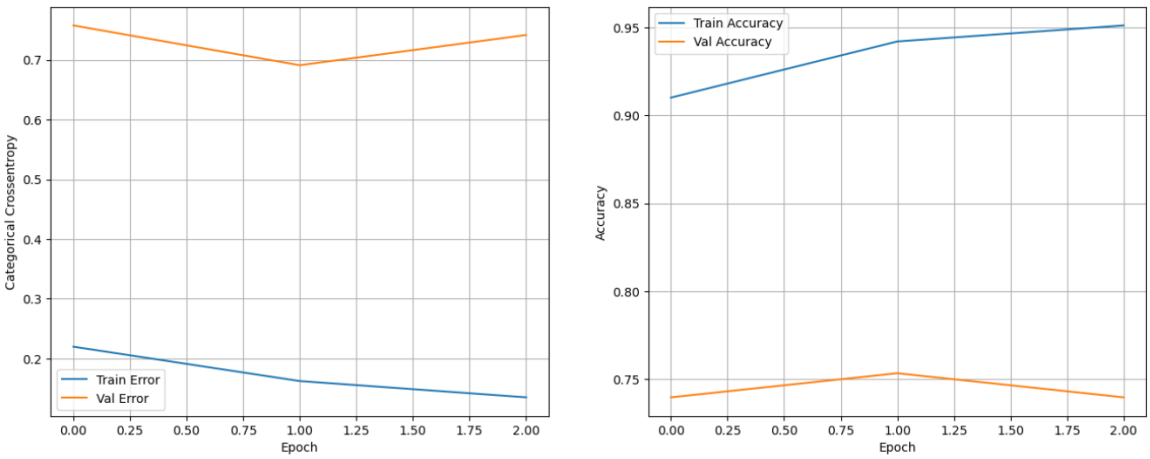

Figure 16: Implementation of InceptionNet

7.3 DenseNet121

```
model = DenseNet121(include_top=False, weights="imagenet", input_shape=(img_size, img_size, 3))
model.summary()
```

```
Model: "densenet121"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 256, 256, 3)]     0

 zero_padding2d_2 (ZeroPadd  (None, 262, 262, 3)       0
 ing2D)

 conv1/conv (Conv2D)         (None, 128, 128, 64)      9408

 conv1/bn (BatchNormalizati  (None, 128, 128, 64)      256
 on)

 conv1/relu (Activation)     (None, 128, 128, 64)      0

 zero_padding2d_3 (ZeroPadd  (None, 130, 130, 64)      0
 ing2D)
```

```
denseNet = model.output
denseNet = Flatten()(denseNet)
denseNet = Dense(256, activation='relu')(denseNet)
denseNet = Dropout(0.02)(denseNet)
output_layer = Dense(3, activation='tanh')(denseNet)

model = Model(inputs=model.input, outputs=output_layer)
```

```
model.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 256, 256, 3)]     0

 zero_padding2d_2 (ZeroPadd  (None, 262, 262, 3)       0
 ing2D)

 conv1/conv (Conv2D)         (None, 128, 128, 64)      9408

 conv1/bn (BatchNormalizati  (None, 128, 128, 64)      256
 on)

 conv1/relu (Activation)     (None, 128, 128, 64)      0

 zero_padding2d_3 (ZeroPadd  (None, 130, 130, 64)      0
 ing2D)
```

```
callback = EarlyStopping( monitor='val_accuracy', patience=1, verbose=1, mode='max')
history = model.fit(train, epochs=10, validation_data=test, shuffle = True, callbacks=[callback])
```

```
Epoch 1/10
28/28 [==============================] - 1580s 55s/step - loss: 3.8364 - accuracy: 0.6606 - val_loss: 1.7664 - val_accuracy: 0.
5114
Epoch 2/10
28/28 [==============================] - 1488s 53s/step - loss: 3.4880 - accuracy: 0.5968 - val_loss: 1.7664 - val_accuracy: 0.
5114
Epoch 2: early stopping
```

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Categorical Crossentropy')
ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
ax1.grid()
ax1.legend()
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
ax2.grid()
ax2.legend()
```

```
<matplotlib.legend.Legend at 0x266957f2950>
```

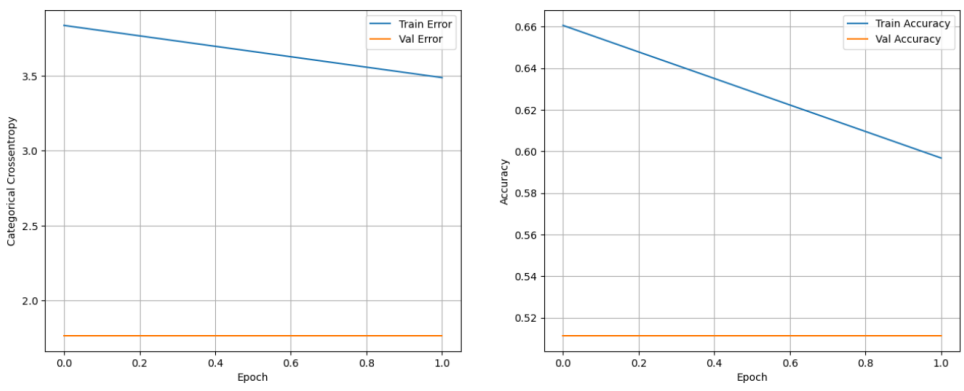

Figure 17: Implementation of DenseNet121

7.5 EfficientNet

```
base_model = EfficientNetV2B0(weights='imagenet', include_top=False, input_shape=(img_size, img_size, 3))
base_model.summary
```

```
<bound method Model.summary of <keras.src.engine.functional.Functional object at 0x000002669B9FB390>>
```

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.35)(x)
x = Dense(32, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(8, activation='sigmoid')(x)
predictions = Dense(3, activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

```
model.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "model_7"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_5 (InputLayer)        [(None, 256, 256, 3)]     0         []

 rescaling (Rescaling)       (None, 256, 256, 3)       0         ['input_5[0][0]']

 normalization (Normalizati  (None, 256, 256, 3)       0         ['rescaling[0][0]']
 on)

 stem_conv (Conv2D)          (None, 128, 128, 32)      864       ['normalization[0][0]']

 stem_bn (BatchNormalizatio  (None, 128, 128, 32)      128       ['stem_conv[0][0]']
 n)

 stem_activation (Activatio  (None, 128, 128, 32)      0         ['stem_bn[0][0]']
 n)
```

```
callback = EarlyStopping( monitor='val_accuracy', patience=1, verbose=1, mode='max')
history = model.fit(train, epochs=10, validation_data=test, callbacks=[callback])
```

```
Epoch 1/10
28/28 [==============================] - 238s 7s/step - loss: 2.9201 - accuracy: 0.3041 - val_loss: 1.0463 - val_accuracy: 0.10
96
Epoch 2/10
28/28 [==============================] - 209s 7s/step - loss: 1.1005 - accuracy: 0.2358 - val_loss: 1.0230 - val_accuracy: 0.10
96
Epoch 2: early stopping
```

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Categorical Crossentropy')
ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
ax1.grid()
ax1.legend()
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
ax2.grid()
ax2.legend()
```

```
<matplotlib.legend.Legend at 0x266ce4c1a50>
```

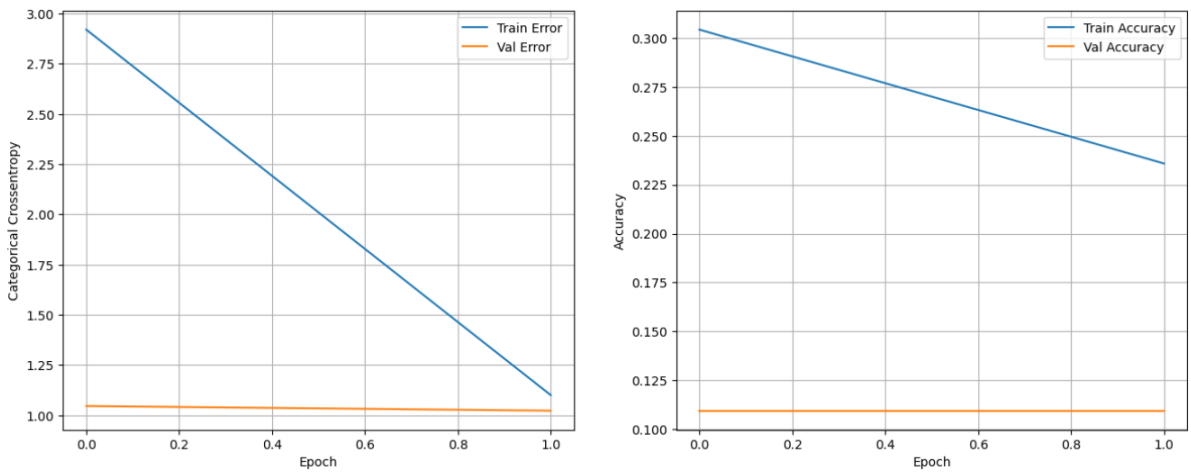

Figure 18: Implementation of EfficientNet

7.4 VGG16

```
model = VGG16(include_top=False,  weights='imagenet',  input_shape=(img_size, img_size, 3))
model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
vgg16 = model.output
vgg16 = Flatten()(vgg16)
vgg16 = Dense(512, activation='relu')(vgg16)
output_layer = Dense(3, activation='relu')(vgg16)

model = Model(inputs=model.input, outputs=output_layer)
```

Figure 19: Implementation of VGG16

```
model.compile(optimizer = 'sgd', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "model_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 flatten_5 (Flatten)         (None, 32768)             0

 dense_26 (Dense)            (None, 512)               16777728

 dense_27 (Dense)            (None, 3)                 1539

=================================================================
Total params: 31493955 (120.14 MB)
Trainable params: 31493955 (120.14 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
callback = EarlyStopping( monitor='val_accuracy', patience=1, verbose=1, mode='max')
history = model.fit(train, epochs=10, validation_data=test, callbacks=[callback])
```

```
Epoch 1/10
28/28 [==============================] - 1488s 53s/step - loss: 6.3370 - accuracy: 0.5023 - val_loss: 6.4074 - val_accuracy: 0.
5114
Epoch 2/10
28/28 [==============================] - 1494s 53s/step - loss: 6.4082 - accuracy: 0.5114 - val_loss: 6.4017 - val_accuracy: 0.
5114
Epoch 2: early stopping
```

Figure 20: Implementation of VGG16

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Categorical Crossentropy')
ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
ax1.grid()
ax1.legend()
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
ax2.grid()
ax2.legend()
```

```
<matplotlib.legend.Legend at 0x266b96cf050>
```

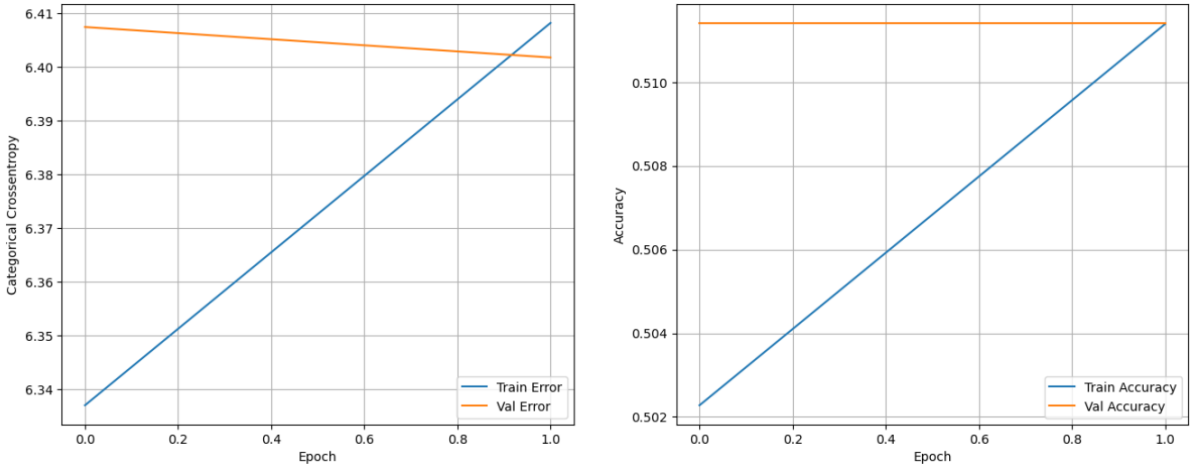

Figure 21: Implementation of VGG16

7.5 VGG19

```
model = VGG19(include_top=False,  weights='imagenet',  input_shape=(img_size, img_size, 3))
model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv4 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv4 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv4 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 20024384 (76.39 MB)
Trainable params: 20024384 (76.39 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
vgg19 = model.output
vgg19 = Dense(512, activation='relu')(vgg19)
vgg19 = Flatten()(vgg19)
vgg19 = Dense(256, activation='relu')(vgg19)
vgg19 = Dropout(0.02)(vgg19)
output_layer = Dense(3, activation='sigmoid')(vgg19)

model = Model(inputs=model.input, outputs=output_layer)
```

Figure 22: Implementation of VGG19

```
model.compile(optimizer = 'sgd', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "model_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv4 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv4 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv4 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 flatten_6 (Flatten)         (None, 32768)             0

 dense_28 (Dense)            (None, 256)               8388864

 dropout_14 (Dropout)        (None, 256)               0

 dense_29 (Dense)            (None, 3)                 771

 dense_30 (Dense)            (None, 512)               2048

 flatten_7 (Flatten)         (None, 512)               0

 dense_31 (Dense)            (None, 256)               131328

 dropout_15 (Dropout)        (None, 256)               0

 dense_32 (Dense)            (None, 3)                 771

=================================================================
Total params: 28548166 (108.90 MB)
Trainable params: 28548166 (108.90 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 23: Implementation of VGG19

```
callback = EarlyStopping( monitor='val_accuracy', patience=1, verbose=1, mode='max')
history = model.fit(train, epochs=10, validation_data=test, callbacks=[callback])
```

```
Epoch 1/10
28/28 [==============================] - 2150s 77s/step - loss: 1.0430 - accuracy: 0.5068 - val_loss: 1.0000 - val_accuracy: 0.
5114
Epoch 2/10
28/28 [==============================] - 1878s 67s/step - loss: 0.9849 - accuracy: 0.5114 - val_loss: 0.9717 - val_accuracy: 0.
5114
Epoch 2: early stopping
```

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Categorical Crossentropy')
ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
ax1.grid()
ax1.legend()
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
ax2.grid()
ax2.legend()
```

```
<matplotlib.legend.Legend at 0x266a1132e90>
```

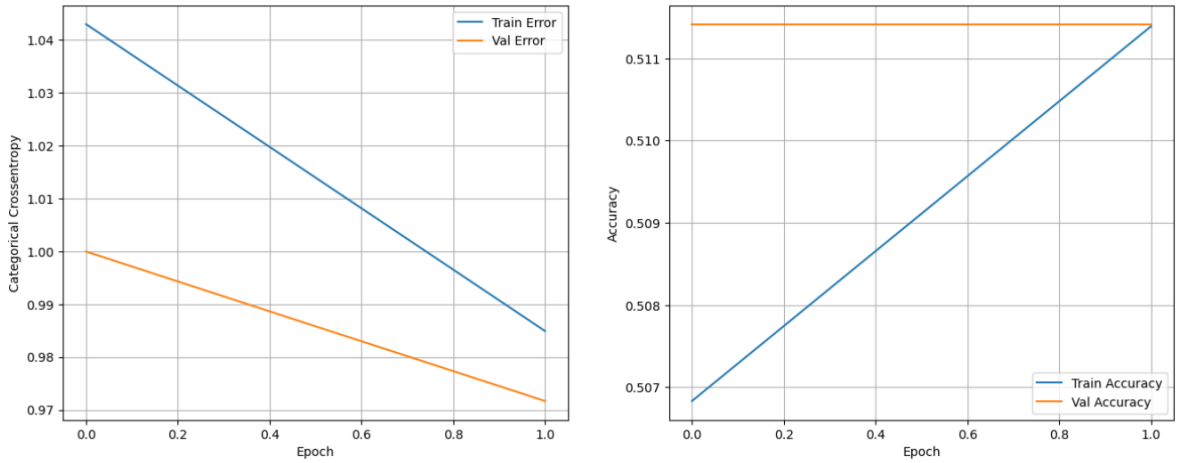

Figure 24: Implementation of VGG19

8 Model result

This section explains the performance of the models.

8.1 Model Scores

```
modelScores.columns = ['Models', 'Accuracy']
```

```
modelScores
```

	Models	Accuracy
0	CNN	53.424656
0	InceptionNet	73.972601
0	Dense Net	51.141554
0	EfficientNet	10.958904
0	EfficientNet	10.958904
0	VGG16	51.141554
0	VGG19	51.141554

Figure 25: Model Performance

```
plt.figure(figsize=(8,5))
plt.bar(modelScores['Models'], modelScores['Accuracy'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3, 4, 5],
 [Text(0, 0, 'CNN'),
  Text(1, 0, 'InceptionNet'),
  Text(2, 0, 'Dense Net'),
  Text(3, 0, 'EfficientNet'),
  Text(4, 0, 'VGG16'),
  Text(5, 0, 'VGG19')])
```

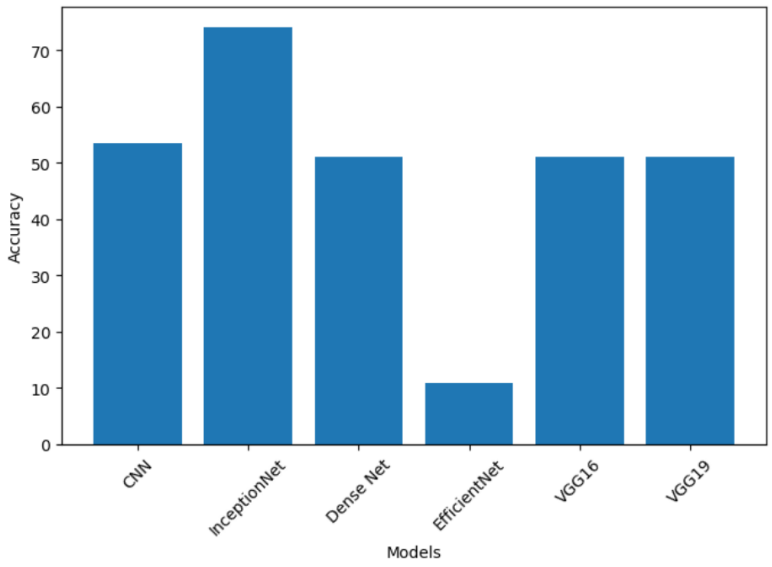

Figure 26: Model Performance

References

https://www.kaggle.com/datasets/adityamahimkar/iqothnccd-lung-cancer-dataset

OpenCV: OpenCV modules

Keras Applications