Configuration Manual:Lung cancer detection using machine learning techniques and image processing

MSc Research Project
Data Analytics

Sumit Jadhav
Student ID: 18129633

School of Computing
National College of Ireland

Supervisor: Dr. Mohammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sumit Jadhav
Student ID:	18129633
Programme:	Data Analytics
Year:	2018
Module:	MSc Research Project
Supervisor:	Dr. Mohammad Iqbal
Submission Due Date:	20/12/2018
Project Title:	Configuration Manual:Lung cancer detection using machine learning techniques and image processing
Word Count:	788
Page Count:	12

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:	
Date:	12th December 2019

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:Lung cancer detection using machine learning techniques and image processing

Sumit Jadhav
18129633

1 Introduction

This configuration manual will help to replicate the research "Lung Cancer Detection Using Classification Algorithms" from scratch. This configuration manual gives detailed information of the required prerequisite to set up and successfully build, run and test this research using suggested framework.

This manual is divided into following sections: Section 2 gives the details of the environment setup. Section 3 discusses about the libraries required for implementing this project. Section 4 gives all the details regarding the dataset. Section 5 explains how the models are implemented and contains the information regarding code repository.

2 Hardware Specification

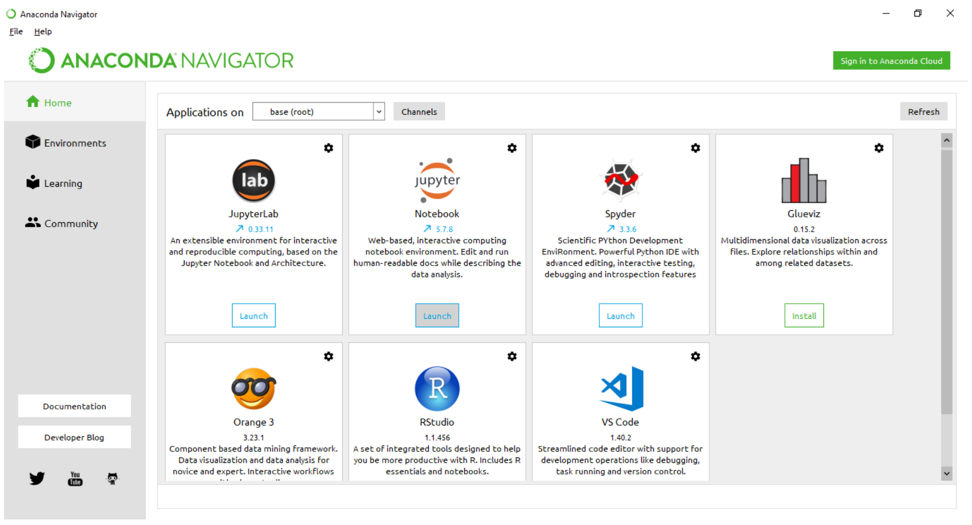
- Operating System: Windows 10 Home Single Language (10.0, Build 18362)
- Processor: Intel(R) Core (TM) i5-3317U CPU @ 1.70GHz (4 CPUs)
- Installed RAM: 8.00 GB
- System Type: 64-bit OS, x64-based processor

2.1 Software Specification

- Anaconda Navigator for Windows (Version 1.9.7)
- Jupyter Notebook (Version 6.0.2)
- 2.3 Programming Requisites
- Python (Version 3.7.5)

2.2 Python Environment Setup

The project was completely implemented using python language, so Anaconda framework was selected. In Anaconda framework there are several preinstalled environments like Jupyter lab, Jupyter Notebook, Spyder, Glueviz, Orange 3, R Studio and VS Code. Figure 1 explains the launch of Jupyter Notebook environment. Jupyter provides an interface to write the code, build the models and testing of it.



JupyterLab
0.33.11
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
5.7.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Spyder
3.3.6
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Glueviz
0.15.2
Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3
3.23.1
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

VS Code
1.40.2
Streamlined code editor with support for development operations like debugging, task running and version control.

Figure 1: Classification models Evaluation Score

3 Libraries required

All the libraries which were required to build this research project are mentioned below in the table 1 along with the commands which can be used to import them. Also, some of the libraries are required to download before using them.

Table 1: Library and Command

Library	Command
Pandas	import pandas as pd
numpy	import numpy as np
matplotlib	import matplotlib.pyplot as plt

sklearn	import sklearn
	from sklearn import metrics
	from sklearn.ensemble import RandomForestClassifier
	from sklearn import svm, metrics, datasets
	from sklearn.ensemble import AdaBoostClassifier

skimage	from skimage import io
	from skimage.io import imread
	from skimage.transform import resize

os	import os
cv2	import cv2
tqdm	from tqdm import tqdm

tensorflow	from tensorflow.keras.callbacks import TensorBoard
	import tensorflow as tf
	from tensorflow.keras.models import Sequential
	from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, BatchNormalization

time	import time
Path	from pathlib import Path
Xgboost	from xgboost import XGBClassifier

4 Dataset

The dataset for this research can be accessed from ? https://www.kaggle.com/kmader/lungnodemalignancy The dataset has This dataset is completely available on the public domain named Kaggle and was created by Kevin Mader. This dataset has a total of 6,691 images in which 4,165 images are labeled as benign and 2,562 images are labeled as malignant. This dataset was extracted from The Cancer Imaging Archive (TCIA): https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI and was converted into a multipage Tagged Image File Format (TIFF) format. This image can be viewed with a specific software specified by the author which are ImageJ or KNIME. These images were split and converted into jpg as TIFF does not supports compression and for neural network full resolution images can cause memory operation errors.

4.1 Data Pre-processing

As the dataset was present in a multipage tiff it was split into an individual image and converted to a Joint Photographic Experts Group (JPEG) format. This was done using TIFF Splitter tool. As all the images should be in same dimensions the reshaping of the images was done to the dimension of 64x64 by using Python Jupyter and further antialiasing filter was applied. Figure 2 explains the steps taken in data preprocessing.

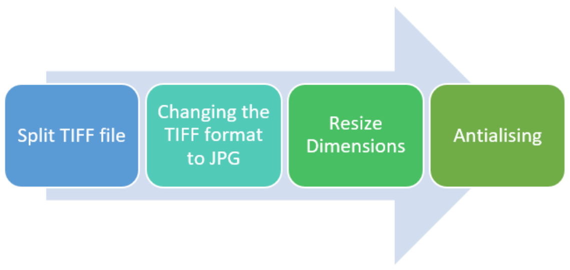
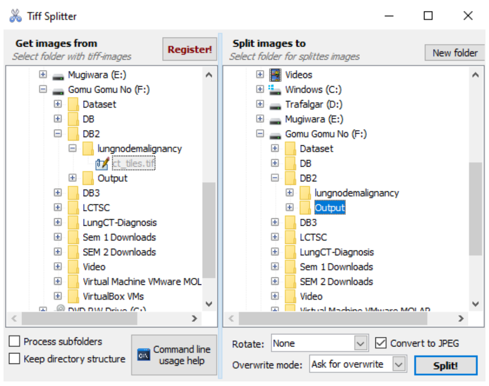


Figure 2: Image Pre-processing and Transformation

For splitting the TIFF file, a tiff splitter application was used. This tool can be downloaded from https://tiff-splitter.windows10compatible.com/ . After downloading this tool is ready to install and after installation following interface can be seen. In this interface selection of the input TIFF file and destination folder is required. For the conversion of the image in checkbox is required to be clicked on.The interface of the tool is explained in figure 3.

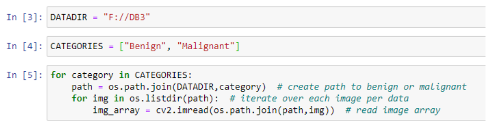
4.2 Loading the Dataset

After this further preprocessing of image in done where these images and stored in an array and then resize and grayscale conversion operation is performed with the help of the following code mentioned in figure 4:



Rotate: None
[✓] Convert to JPEG
Overwrite mode: Ask for overwrite

Figure 3: Image Pre-processing and Transformation



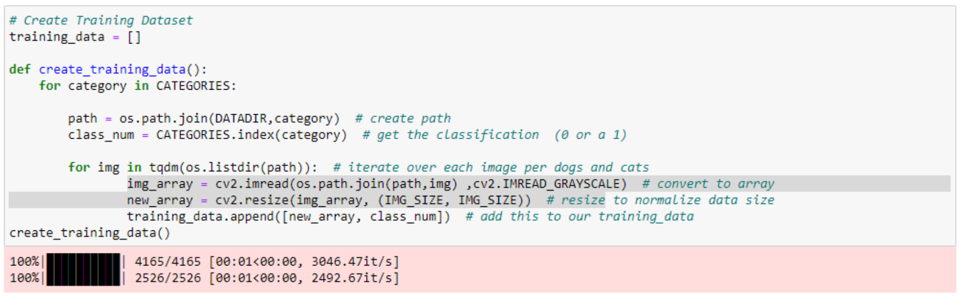
```
In [3]: DATADIR = "F://DB3"

In [4]: CATEGORIES = ["Benign", "Malignant"]

In [5]: for category in CATEGORIES:
            path = os.path.join(DATADIR,category)  # create path to benign or malignant
            for img in os.listdir(path):  # iterate over each image per data
                img_array = cv2.imread(os.path.join(path,img))  # read image array
```

Figure 4: Python code for loading the dataset

5 classification Model Implementation

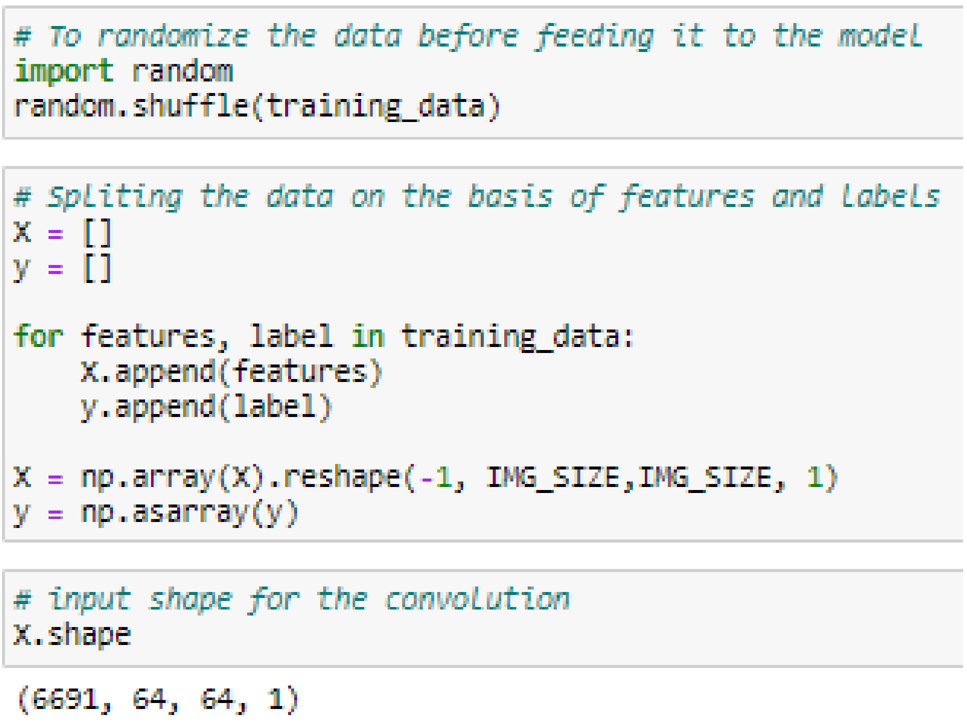
5.1 Convolution Neural Network (CNN)

Implementation process for CNN is done completely from the scratch and the model can be explained further with the help of comments present in the code.



```
# Create Training Dataset
training_data = []

def create_training_data():
    for category in CATEGORIES:

        path = os.path.join(DATADIR,category)  # create path
        class_num = CATEGORIES.index(category)  # get the classification  (0 or a 1)

        for img in tqdm(os.listdir(path)):  # iterate over each image per dogs and cats
                img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to normalize data size
                training_data.append([new_array, class_num])  # add this to our training_data
create_training_data()
```

```
100%|██████████| 4165/4165 [00:01<00:00, 3046.47it/s]
100%|██████████| 2526/2526 [00:01<00:00, 2492.67it/s]
```

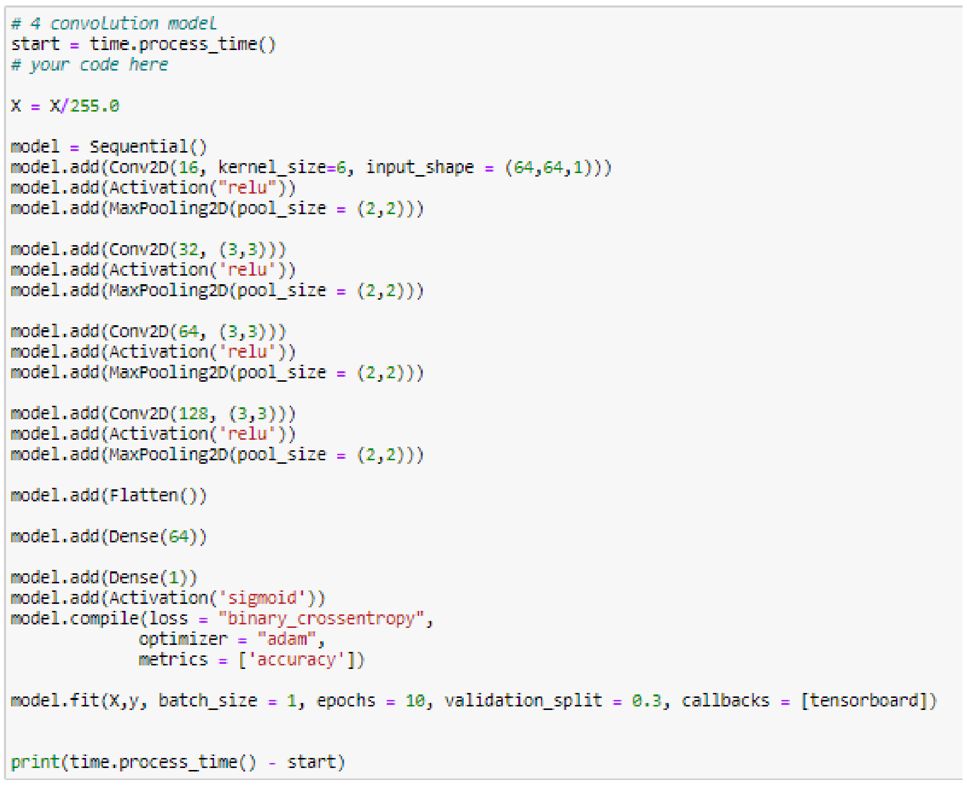


```python
# To randomize the data before feeding it to the model
import random
random.shuffle(training_data)
```

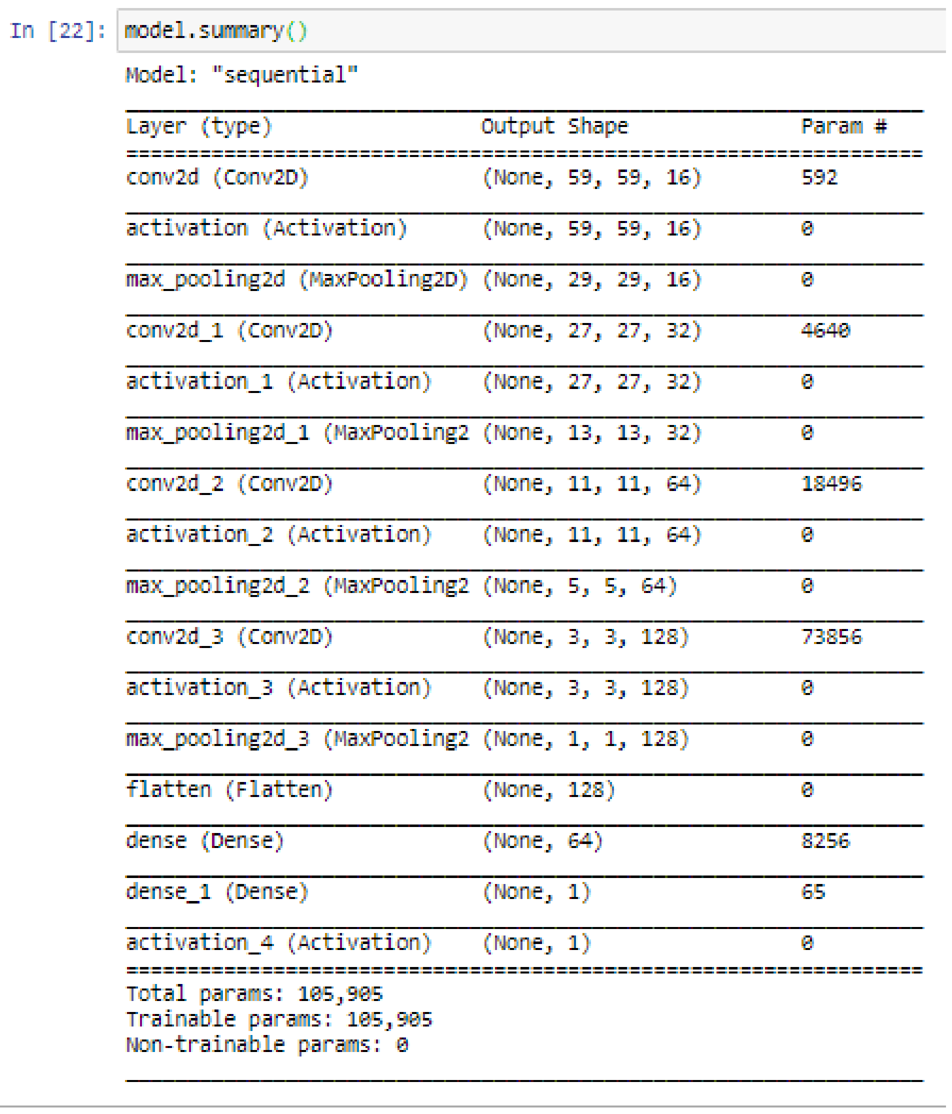
```python
# Spliting the data on the basis of features and labels
X = []
y = []

for features, label in training_data:
    X.append(features)
    y.append(label)

X = np.array(X).reshape(-1, IMG_SIZE,IMG_SIZE, 1)
y = np.asarray(y)
```

```python
# input shape for the convolution
X.shape
```

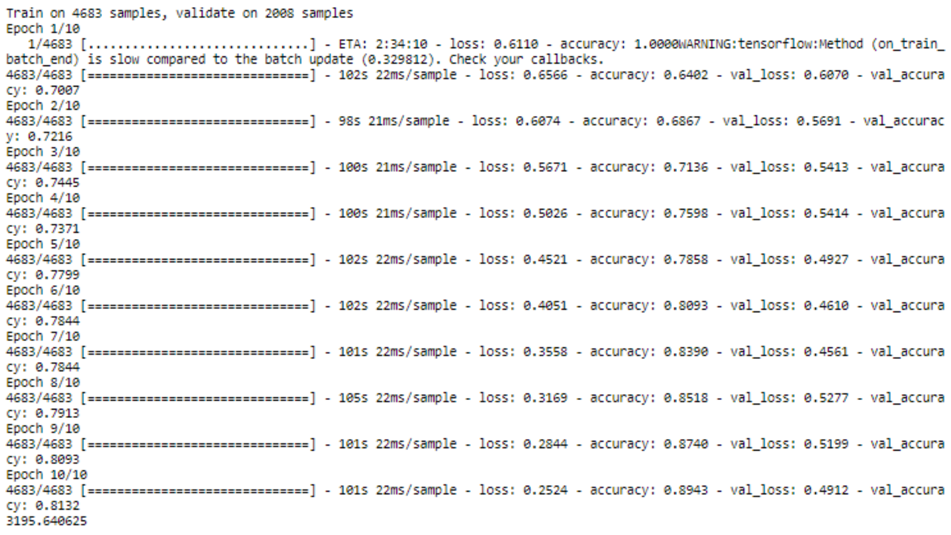
```
(6691, 64, 64, 1)
```



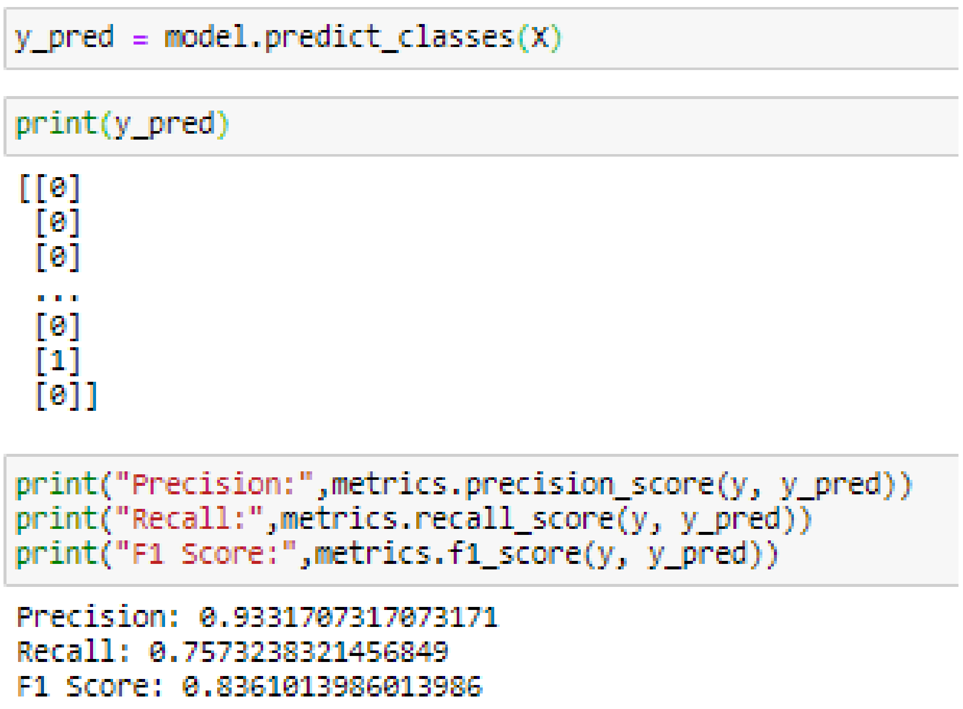
```python
# 4 convolution model
start = time.process_time()
# your code here

X = X/255.0

model = Sequential()
model.add(Conv2D(16, kernel_size=6, input_shape = (64,64,1)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(32, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(128, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Flatten())

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss = "binary_crossentropy",
             optimizer = "adam",
             metrics = ['accuracy'])

model.fit(X,y, batch_size = 1, epochs = 10, validation_split = 0.3, callbacks = [tensorboard])


print(time.process_time() - start)
```



```
In [22]: model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 59, 59, 16)        592
_________________________________________________________________
activation (Activation)      (None, 59, 59, 16)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 29, 29, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 27, 27, 32)        4640
_________________________________________________________________
activation_1 (Activation)    (None, 27, 27, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 11, 11, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
activation_3 (Activation)    (None, 3, 3, 128)         0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 64)                8256
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0
=================================================================
Total params: 105,905
Trainable params: 105,905
Non-trainable params: 0
_________________________________________________________________
```



```
Train on 4683 samples, validate on 2008 samples
Epoch 1/10
   1/4683 [..............................] - ETA: 2:34:10 - loss: 0.6110 - accuracy: 1.0000WARNING:tensorflow:Method (on_train_
batch_end) is slow compared to the batch update (0.329812). Check your callbacks.
4683/4683 [==============================] - 102s 22ms/sample - loss: 0.6566 - accuracy: 0.6402 - val_loss: 0.6070 - val_accura
cy: 0.7007
Epoch 2/10
4683/4683 [==============================] - 98s 21ms/sample - loss: 0.6074 - accuracy: 0.6867 - val_loss: 0.5691 - val_accurac
y: 0.7216
Epoch 3/10
4683/4683 [==============================] - 100s 21ms/sample - loss: 0.5671 - accuracy: 0.7136 - val_loss: 0.5413 - val_accura
cy: 0.7445
Epoch 4/10
4683/4683 [==============================] - 100s 21ms/sample - loss: 0.5026 - accuracy: 0.7598 - val_loss: 0.5414 - val_accura
cy: 0.7371
Epoch 5/10
4683/4683 [==============================] - 102s 22ms/sample - loss: 0.4521 - accuracy: 0.7858 - val_loss: 0.4927 - val_accura
cy: 0.7799
Epoch 6/10
4683/4683 [==============================] - 102s 22ms/sample - loss: 0.4051 - accuracy: 0.8093 - val_loss: 0.4610 - val_accura
cy: 0.7844
Epoch 7/10
4683/4683 [==============================] - 101s 22ms/sample - loss: 0.3558 - accuracy: 0.8390 - val_loss: 0.4561 - val_accura
cy: 0.7844
Epoch 8/10
4683/4683 [==============================] - 105s 22ms/sample - loss: 0.3169 - accuracy: 0.8518 - val_loss: 0.5277 - val_accura
cy: 0.7913
Epoch 9/10
4683/4683 [==============================] - 101s 22ms/sample - loss: 0.2844 - accuracy: 0.8740 - val_loss: 0.5199 - val_accura
cy: 0.8093
Epoch 10/10
4683/4683 [==============================] - 101s 22ms/sample - loss: 0.2524 - accuracy: 0.8943 - val_loss: 0.4912 - val_accura
cy: 0.8132
3195.640625
```



```
y_pred = model.predict_classes(X)
```

```
print(y_pred)
```

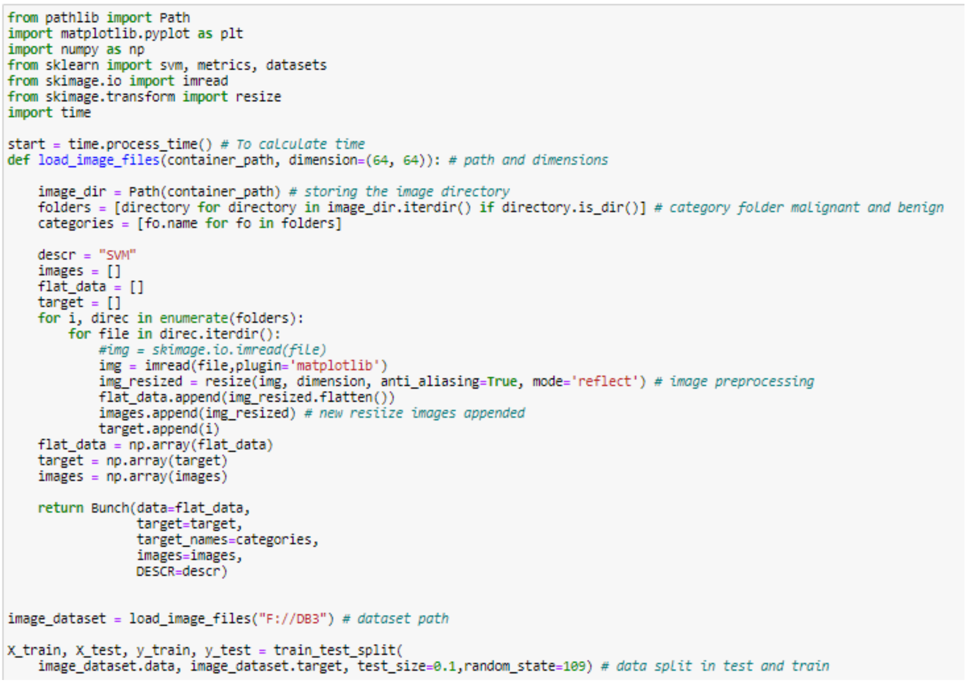
```
[[0]
 [0]
 [0]
 ...
 [0]
 [1]
 [0]]
```

```
print("Precision:",metrics.precision_score(y, y_pred))
print("Recall:",metrics.recall_score(y, y_pred))
print("F1 Score:",metrics.f1_score(y, y_pred))
```

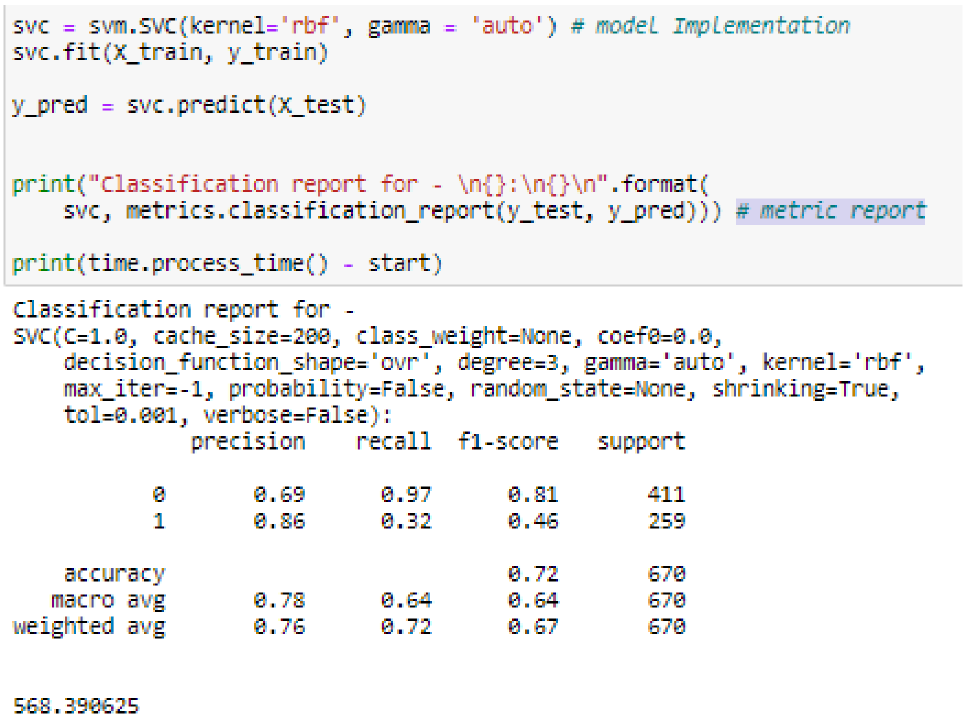
```
Precision: 0.9331707317073171
Recall: 0.7573238321456849
F1 Score: 0.8361013986013986
```

5.2 Support Vector Machine (SVM)

For applying SVM following code was used for implementation:



```python
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm, metrics, datasets
from skimage.io import imread
from skimage.transform import resize
import time

start = time.process_time() # To calculate time
def load_image_files(container_path, dimension=(64, 64)): # path and dimensions

    image_dir = Path(container_path) # storing the image directory
    folders = [directory for directory in image_dir.iterdir() if directory.is_dir()] # category folder malignant and benign
    categories = [fo.name for fo in folders]

    descr = "SVM"
    images = []
    flat_data = []
    target = []
    for i, direc in enumerate(folders):
        for file in direc.iterdir():
            #img = skimage.io.imread(file)
            img = imread(file,plugin='matplotlib')
            img_resized = resize(img, dimension, anti_aliasing=True, mode='reflect') # image preprocessing
            flat_data.append(img_resized.flatten())
            images.append(img_resized) # new resiize images appended
            target.append(i)
    flat_data = np.array(flat_data)
    target = np.array(target)
    images = np.array(images)

    return Bunch(data=flat_data,
                 target=target,
                 target_names=categories,
                 images=images,
                 DESCR=descr)


image_dataset = load_image_files("F://DB3") # dataset path

X_train, X_test, y_train, y_test = train_test_split(
    image_dataset.data, image_dataset.target, test_size=0.1,random_state=109) # data split in test and train
```

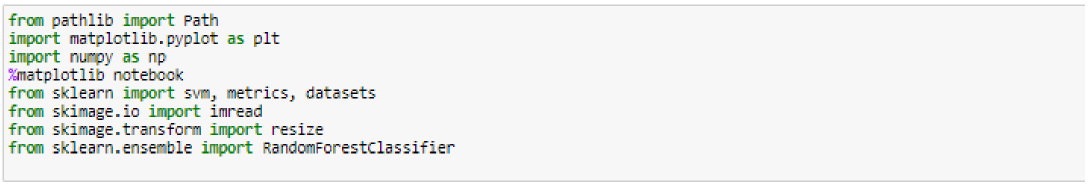


```python
svc = svm.SVC(kernel='rbf', gamma = 'auto') # model Implementation
svc.fit(X_train, y_train)

y_pred = svc.predict(X_test)


print("Classification report for - \n{}:\n{}\n".format(
    svc, metrics.classification_report(y_test, y_pred))) # metric report

print(time.process_time() - start)
```

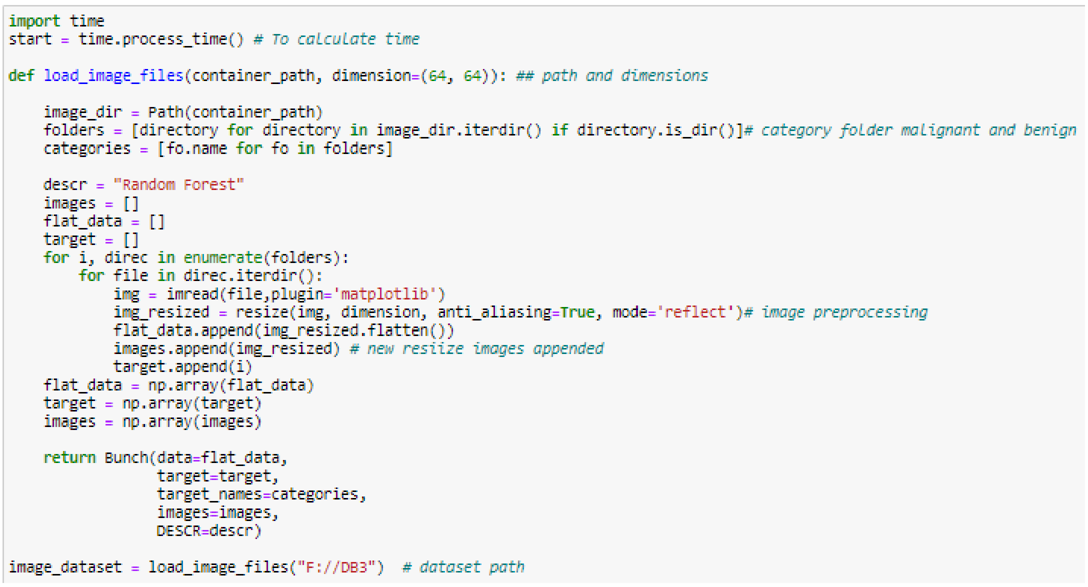
```
Classification report for -
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False):
              precision    recall  f1-score   support

           0       0.69      0.97      0.81       411
           1       0.86      0.32      0.46       259

    accuracy                           0.72       670
   macro avg       0.78      0.64      0.64       670
weighted avg       0.76      0.72      0.67       670


568.390625
```

5.3 Random Forest (RF)

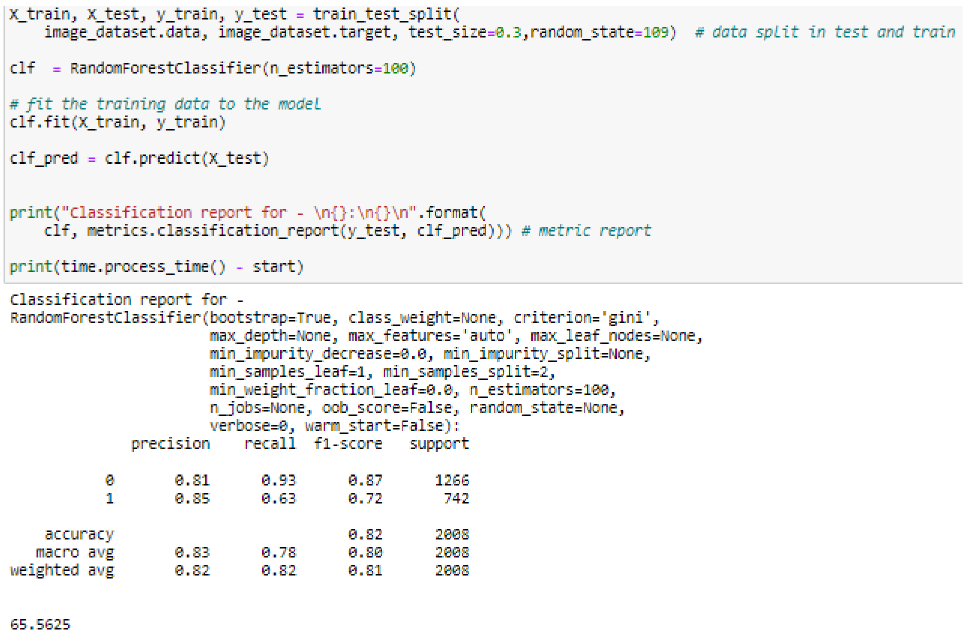
For applying RF following code was used for implementation:



```python
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
%matplotlib notebook
from sklearn import svm, metrics, datasets
from skimage.io import imread
from skimage.transform import resize
from sklearn.ensemble import RandomForestClassifier
```



```python
import time
start = time.process_time() # To calculate time

def load_image_files(container_path, dimension=(64, 64)): ## path and dimensions

    image_dir = Path(container_path)
    folders = [directory for directory in image_dir.iterdir() if directory.is_dir()]# category folder malignant and benign
    categories = [fo.name for fo in folders]

    descr = "Random Forest"
    images = []
    flat_data = []
    target = []
    for i, direc in enumerate(folders):
        for file in direc.iterdir():
            img = imread(file,plugin='matplotlib')
            img_resized = resize(img, dimension, anti_aliasing=True, mode='reflect')# image preprocessing
            flat_data.append(img_resized.flatten())
            images.append(img_resized) # new resiize images appended
            target.append(i)
    flat_data = np.array(flat_data)
    target = np.array(target)
    images = np.array(images)

    return Bunch(data=flat_data,
                 target=target,
                 target_names=categories,
                 images=images,
                 DESCR=descr)

image_dataset = load_image_files("F://DB3")  # dataset path
```

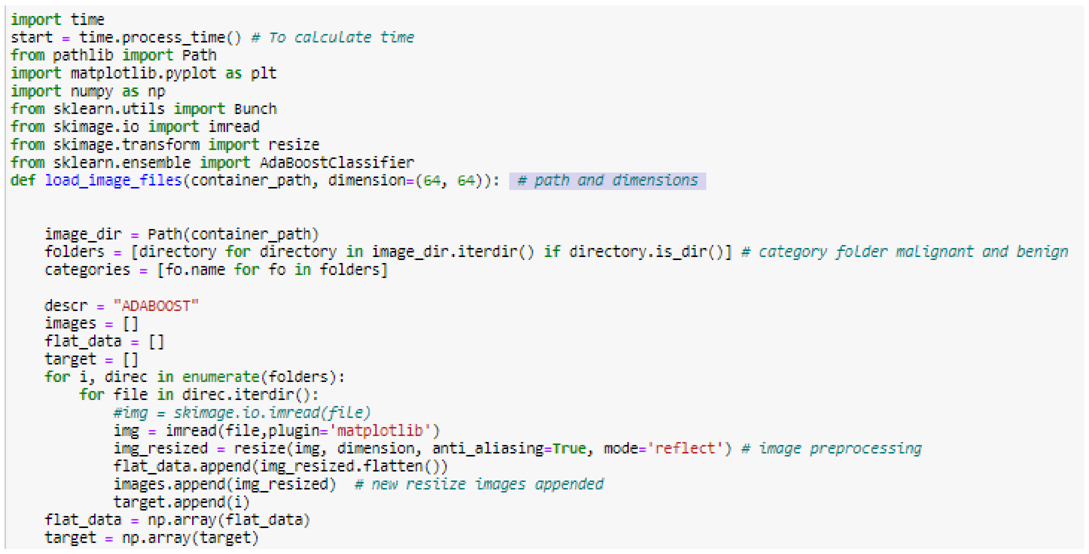


```
X_train, X_test, y_train, y_test = train_test_split(
    image_dataset.data, image_dataset.target, test_size=0.3,random_state=109)  # data split in test and train

clf  = RandomForestClassifier(n_estimators=100)

# fit the training data to the model
clf.fit(X_train, y_train)

clf_pred = clf.predict(X_test)


print("Classification report for - \n{}:\n{}\n".format(
    clf, metrics.classification_report(y_test, clf_pred))) # metric report

print(time.process_time() - start)
```

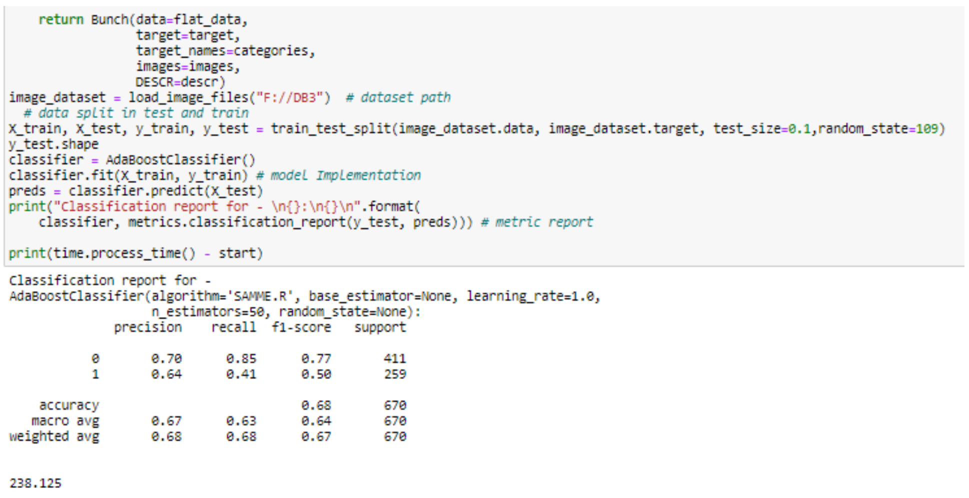
```
Classification report for -
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False):
              precision    recall  f1-score   support

           0       0.81      0.93      0.87      1266
           1       0.85      0.63      0.72       742

    accuracy                           0.82      2008
   macro avg       0.83      0.78      0.80      2008
weighted avg       0.82      0.82      0.81      2008


65.5625
```

5.4 Adaptive Boost (ADABost)

For applying ADABost following code was used for implementation:



```
import time
start = time.process_time() # To calculate time
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
from sklearn.utils import Bunch
from skimage.io import imread
from skimage.transform import resize
from sklearn.ensemble import AdaBoostClassifier
def load_image_files(container_path, dimension=(64, 64)):  # path and dimensions


    image_dir = Path(container_path)
    folders = [directory for directory in image_dir.iterdir() if directory.is_dir()] # category folder malignant and benign
    categories = [fo.name for fo in folders]

    descr = "ADABOOST"
    images = []
    flat_data = []
    target = []
    for i, direc in enumerate(folders):
        for file in direc.iterdir():
            #img = skimage.io.imread(file)
            img = imread(file,plugin='matplotlib')
            img_resized = resize(img, dimension, anti_aliasing=True, mode='reflect') # image preprocessing
            flat_data.append(img_resized.flatten())
            images.append(img_resized)  # new resize images appended
            target.append(i)
    flat_data = np.array(flat_data)
    target = np.array(target)
```

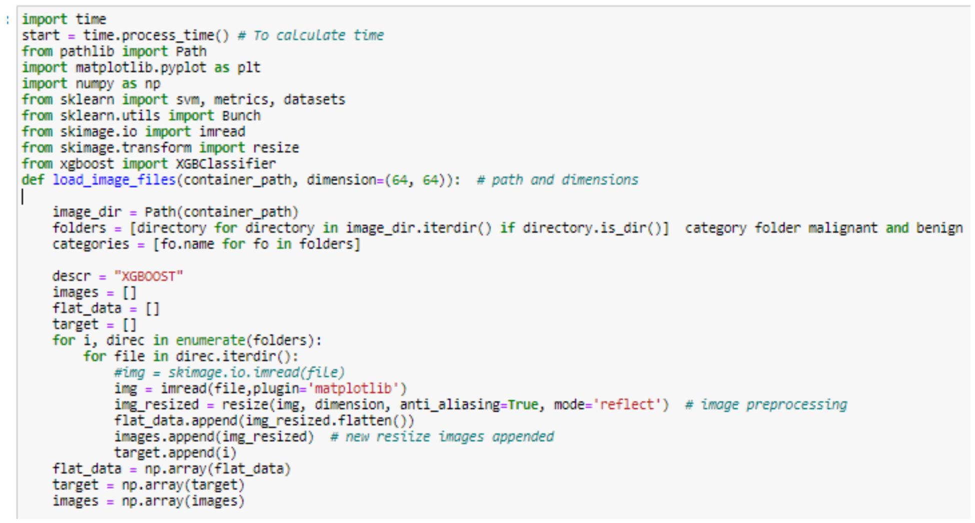


```
    return Bunch(data=flat_data,
                 target=target,
                 target_names=categories,
                 images=images,
                 DESCR=descr)
image_dataset = load_image_files("F://DB3")  # dataset path
  # data split in test and train
X_train, X_test, y_train, y_test = train_test_split(image_dataset.data, image_dataset.target, test_size=0.1,random_state=109)
y_test.shape
classifier = AdaBoostClassifier()
classifier.fit(X_train, y_train) # model Implementation
preds = classifier.predict(X_test)
print("Classification report for - \n{}:\n{}\n".format(
    classifier, metrics.classification_report(y_test, preds))) # metric report

print(time.process_time() - start)
```

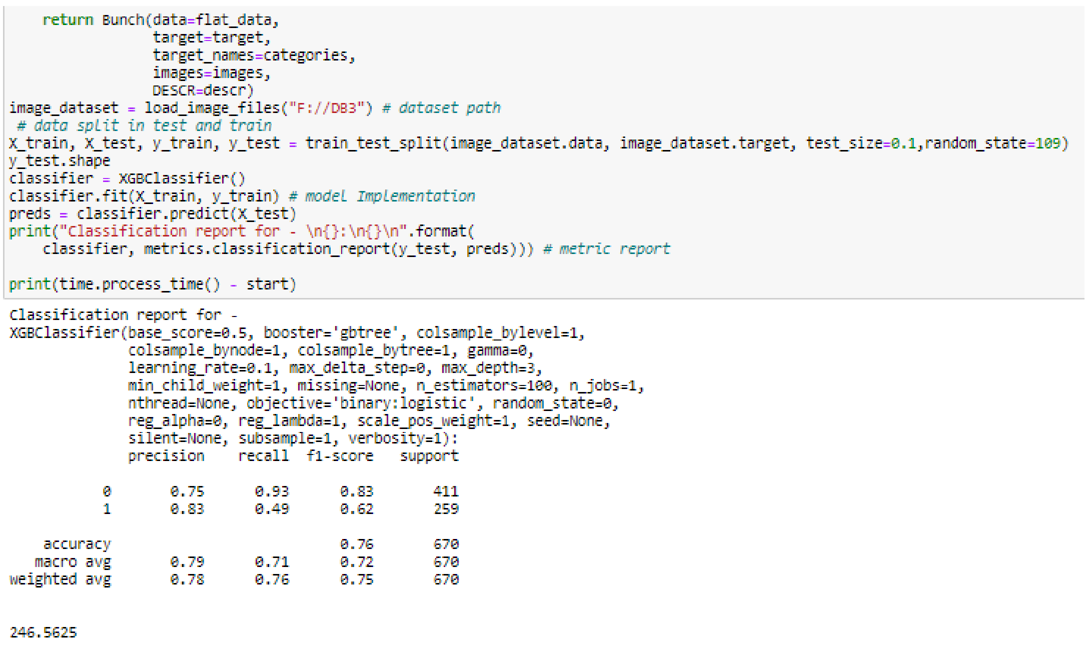
```
Classification report for -
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=None):
              precision    recall  f1-score   support

           0       0.70      0.85      0.77       411
           1       0.64      0.41      0.50       259

    accuracy                           0.68       670
   macro avg       0.67      0.63      0.64       670
weighted avg       0.68      0.68      0.67       670


238.125
```

5.5 Extreme Gradient Boost (XGBoost)

For applying XGBoost following code was used for implementation:



```
import time
start = time.process_time() # To calculate time
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm, metrics, datasets
from sklearn.utils import Bunch
from skimage.io import imread
from skimage.transform import resize
from xgboost import XGBClassifier
def load_image_files(container_path, dimension=(64, 64)):  # path and dimensions

    image_dir = Path(container_path)
    folders = [directory for directory in image_dir.iterdir() if directory.is_dir()]  category folder malignant and benign
    categories = [fo.name for fo in folders]

    descr = "XGBOOST"
    images = []
    flat_data = []
    target = []
    for i, direc in enumerate(folders):
        for file in direc.iterdir():
            #img = skimage.io.imread(file)
            img = imread(file,plugin='matplotlib')
            img_resized = resize(img, dimension, anti_aliasing=True, mode='reflect')  # image preprocessing
            flat_data.append(img_resized.flatten())
            images.append(img_resized)  # new resiize images appended
            target.append(i)
    flat_data = np.array(flat_data)
    target = np.array(target)
    images = np.array(images)
```



```
    return Bunch(data=flat_data,
                 target=target,
                 target_names=categories,
                 images=images,
                 DESCR=descr)
image_dataset = load_image_files("F://DB3") # dataset path
 # data split in test and train
X_train, X_test, y_train, y_test = train_test_split(image_dataset.data, image_dataset.target, test_size=0.1,random_state=109)
y_test.shape
classifier = XGBClassifier()
classifier.fit(X_train, y_train) # model Implementation
preds = classifier.predict(X_test)
print("Classification report for - \n{}:\n{}\n".format(
    classifier, metrics.classification_report(y_test, preds))) # metric report

print(time.process_time() - start)
```

```
Classification report for -
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1):
              precision    recall  f1-score   support

           0       0.75      0.93      0.83       411
           1       0.83      0.49      0.62       259

    accuracy                           0.76       670
   macro avg       0.79      0.71      0.72       670
weighted avg       0.78      0.76      0.75       670


246.5625
```

6 References

- Dataset Source : https://www.kaggle.com/kmader/lungnodemalignancy
- Code Reference for CNN : https://pythonprogramming.net/loading-custom-data-deep-learning-python-tensorflow-keras/
- Code Reference for Other CLassification Models: https://github.com/Abhishek-Arora/Image-Classification-Using-SVM/blob/master/src/imageClassifier.py
- Code Reference for Tensorflow Learning: https://www.tensorflow.org/api_docs/python/tf
- Code Reference for Learning Keras : https://www.tensorflow.org/api_docs/python/tf/keras/