CS5590 – Python and Deep Learning

Spring 2019

Department of Computer Science Electrical Engineering

University of Missouri Kansas City

Team Number #06

Lakshmana Kumar Mettu-16

Tarun Teja Kasturi-12

Pavan Kumar Chongala-04

Acknowledgement Statement

This work has been completed under the guidance of Dr. Yugi Lee and the TAs (Rohith Kumar Nagulapati, Bhargavi Nadendla) in CS5590 Python and Deep Learning, University of Missouri - Kansas City), Spring 2019.

Project Deployment

Performance and Evaluation of Model on IDC Cancer Data

By

Lakshmana Kumar Mettu - 16

Tarun Teja Kasturi - 11

Pavan Kumar Chongala - 04

TABLE OF CONTENTS

Project objective:

- This undertaking manages grouping Uninfected and infected cancer cell images using Machine Learning and Deep learning approaches.
- We compared and evaluated which model is best fit for predicting parasitized and un-parasitized images.

Motivation:

- Breast Cancer is the most common form of cancer and invasive ductal carcinoma (IDC) is the most common form of Breast Cancer. Accurately identifying and categorizing breast cancer subtypes is an important task. So, we have chosen this topic to evaluate accuracy using ML & DL approaches.

Significance:

We are intending to do build a Machine Learning and deep learning algorithm that reads the training data to understand the discrimination of IDC negative and positive cancer outcomes. The trained data is cross validated to obtain the accuracy of the built model. New data or test data is given as input to the model. The model classifies whether the result cancer is IDC positive or negative. This could be help doctor decide by taking his evaluation into consideration. This improves the chance of detection and reduces the unnecessary biopsies.

Dataset Details:

Primarily Dataset has around 300000 samples including both infected and un-infected cells.

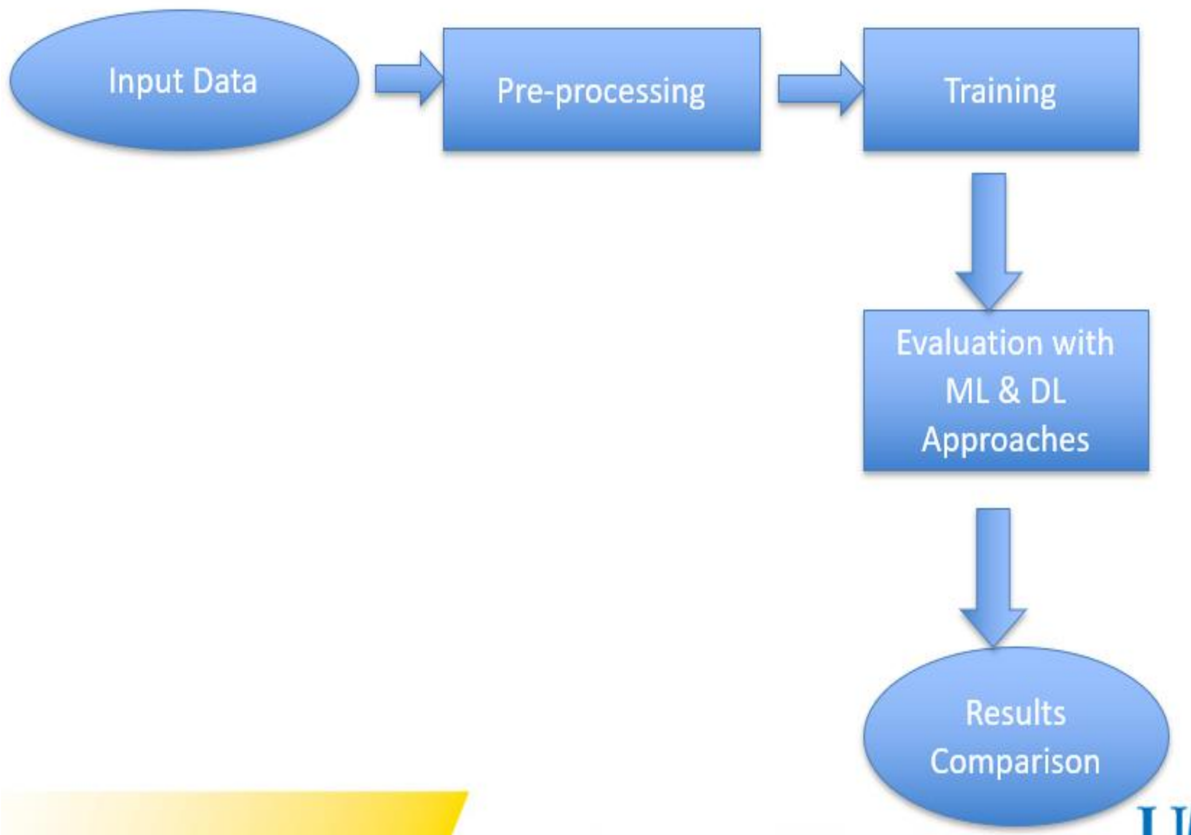
- It was obtained from https://www.ncbi.nlm.nih.gov/
- All Images are in size of 50*50.
- Sample Images are shown below.

- Above Images all are both infected and un-infected images of IDC cancer.

Dependencies

- Python-PyCharm
- TensorFlow
- Keras
- Jupyter-Notebook

Architecture:



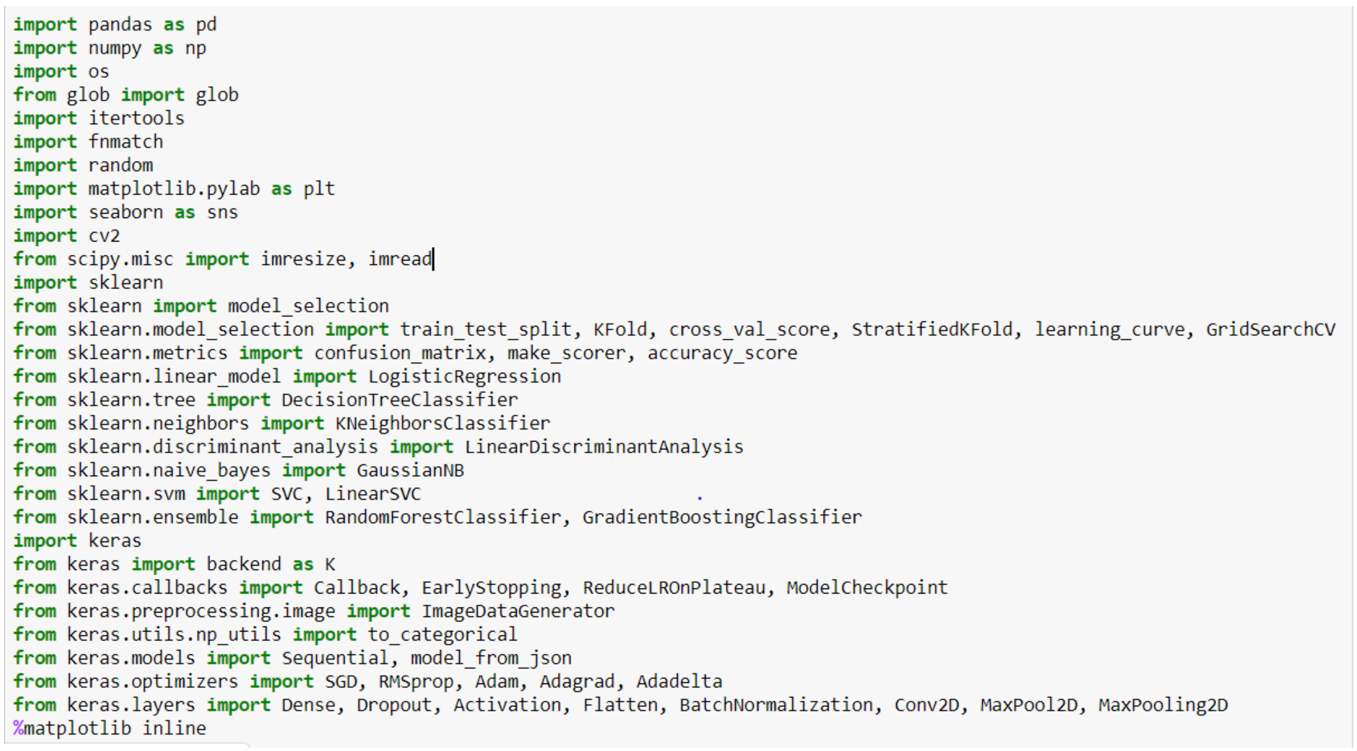
Workflow:

- Data Pre-processing Including Normalization, Sampling (Random over and Under Sampling).
- Model Training
- ML & DL Approaches

As like above we divided the increments in our project. We successfully completed the increments and each increment details are given below. The following code snippets help us to describe about project milestones along with brief description.

1.Modules Importing:

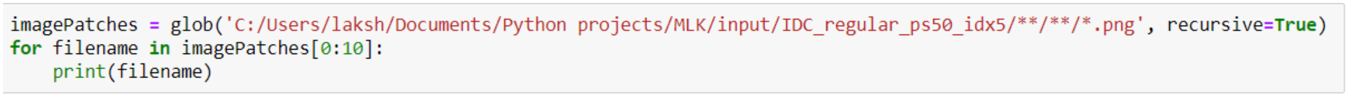
Firstly, we imported the required modules to build up this project. Major libraries are computer vision, sklearn, NumPy, keras, matplotlib.



```
import pandas as pd
import numpy as np
import os
from glob import glob
import itertools
import fnmatch
import random
import matplotlib.pylab as plt
import seaborn as sns
import cv2
from scipy.misc import imresize, imread
import sklearn
from sklearn import model_selection
from sklearn.model_selection import train_test_split, KFold, cross_val_score, StratifiedKFold, learning_curve, GridSearchCV
from sklearn.metrics import confusion_matrix, make_scorer, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import keras
from keras import backend as K
from keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
from keras.models import Sequential, model_from_json
from keras.optimizers import SGD, RMSprop, Adam, Adagrad, Adadelta
from keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization, Conv2D, MaxPool2D, MaxPooling2D
%matplotlib inline
```

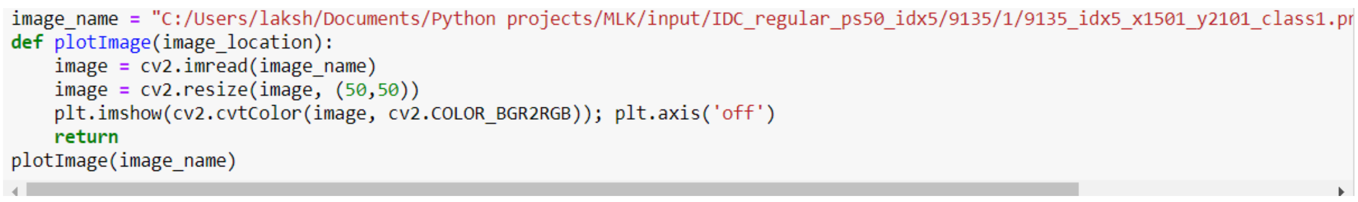
2.Dataset Importing:

Here we have imported the dataset using glob.



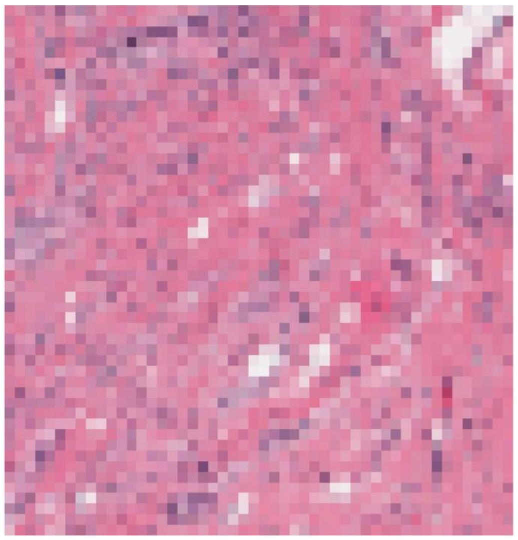
```
imagePatches = glob('C:/Users/laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5/**/**/*.png', recursive=True)
for filename in imagePatches[0:10]:
    print(filename)
```

3.Below snippet showing that reading input image using cv2 package.



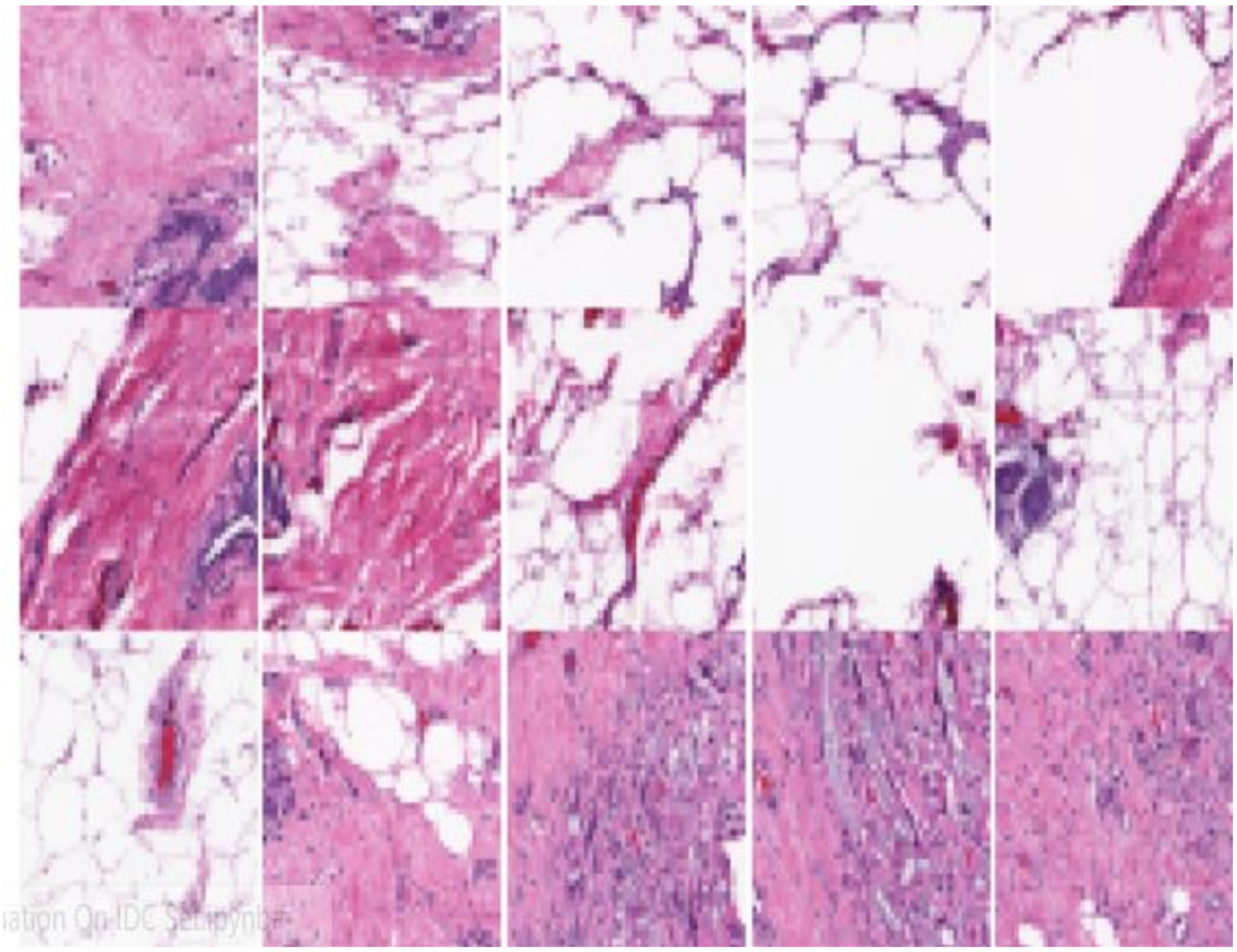
```
image_name = "C:/Users/laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5/9135/1/9135_idx5_x1501_y2101_class1.pr
def plotImage(image_location):
    image = cv2.imread(image_name)
    image = cv2.resize(image, (50,50))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)); plt.axis('off')
    return
plotImage(image_name)
```

The output for above snippet is shown below, which is size of 50*50**.**



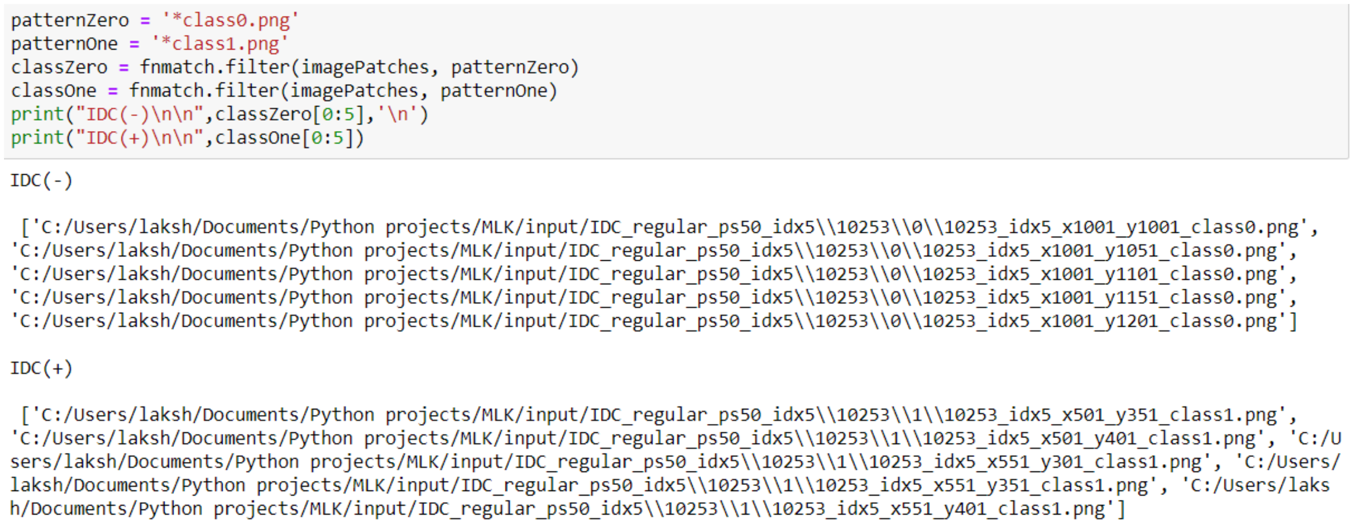
```
# Plot Multiple Images
bunchOfImages = imagePatches
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in bunchOfImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (50, 50))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```

4.We are plotting bunch of images using array and computer vision package. corresponding output are shown below.



5.Now we are differentiating the images as infected and un-infected and assigned to two folders namely class zero and class one.

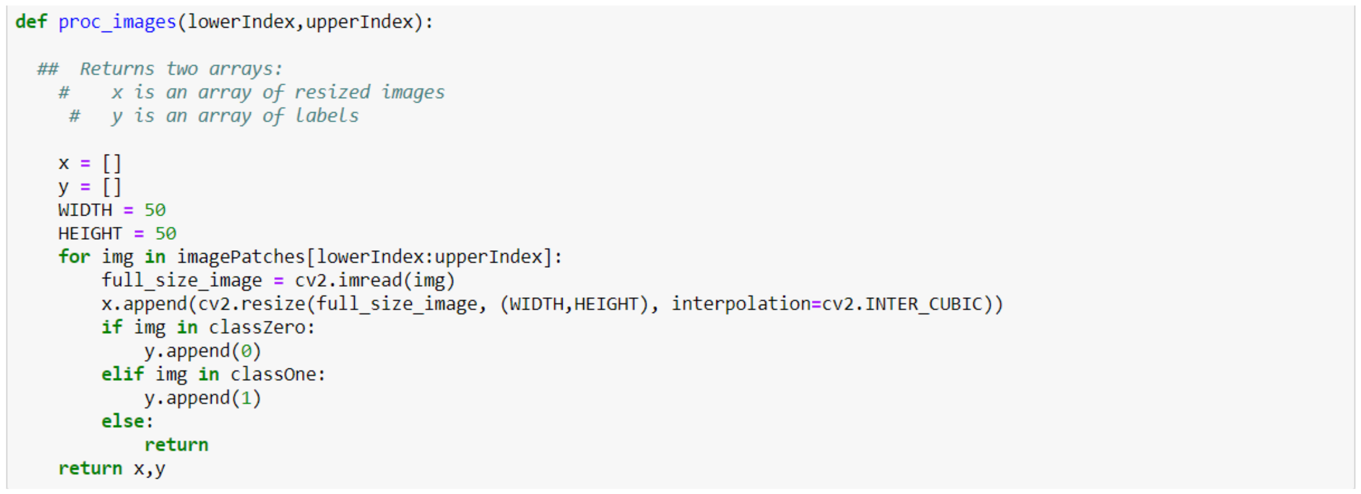
- Class zero corresponds to IDC(-) images.
- Class one corresponds to IDC(+) images.
- Corresponding code snippet and output are shown below.



```python
patternZero = '*class0.png'
patternOne = '*class1.png'
classZero = fnmatch.filter(imagePatches, patternZero)
classOne = fnmatch.filter(imagePatches, patternOne)
print("IDC(-)\n\n",classZero[0:5],'\n')
print("IDC(+)\n\n",classOne[0:5])
```

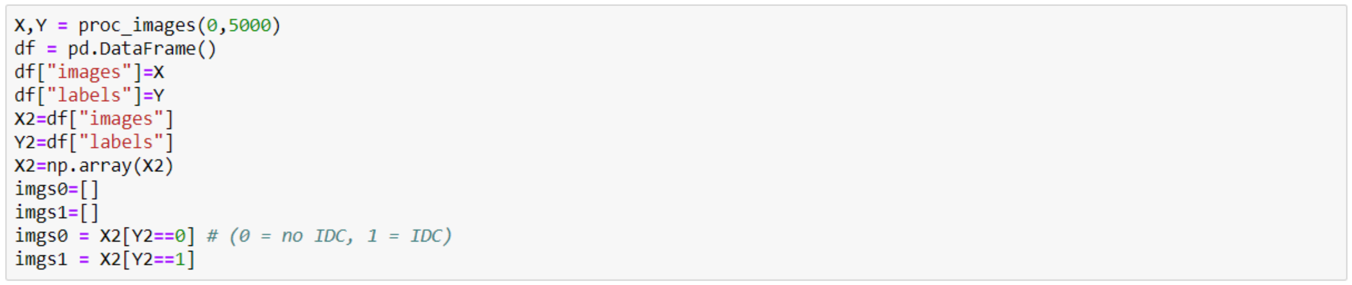
```
IDC(-)

 ['C:/Users/laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5\\10253\\0\\10253_idx5_x1001_y1001_class0.png',
'C:/Users/laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5\\10253\\0\\10253_idx5_x1001_y1051_class0.png',
'C:/Users/laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5\\10253\\0\\10253_idx5_x1001_y1101_class0.png',
'C:/Users/laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5\\10253\\0\\10253_idx5_x1001_y1151_class0.png',
'C:/Users/laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5\\10253\\0\\10253_idx5_x1001_y1201_class0.png']

IDC(+)

 ['C:/Users/laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5\\10253\\1\\10253_idx5_x501_y351_class1.png',
'C:/Users/laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5\\10253\\1\\10253_idx5_x501_y401_class1.png', 'C:/U
sers/laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5\\10253\\1\\10253_idx5_x551_y301_class1.png', 'C:/Users/
laksh/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5\\10253\\1\\10253_idx5_x551_y351_class1.png', 'C:/Users/laks
h/Documents/Python projects/MLK/input/IDC_regular_ps50_idx5\\10253\\1\\10253_idx5_x551_y401_class1.png']
```

6.Similary we are separating all images as two sets using two arrays shown in snippet.

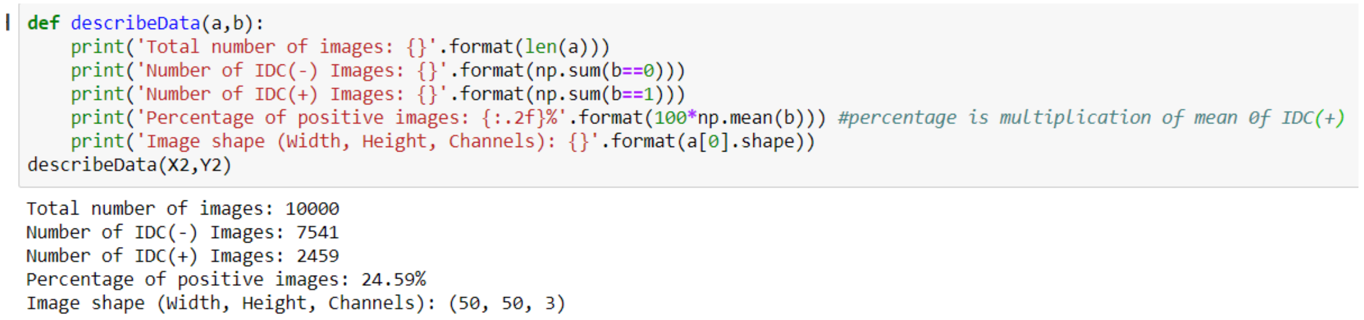


```python
def proc_images(lowerIndex,upperIndex):

  ##  Returns two arrays:
    #    x is an array of resized images
     #   y is an array of labels

    x = []
    y = []
    WIDTH = 50
    HEIGHT = 50
    for img in imagePatches[lowerIndex:upperIndex]:
        full_size_image = cv2.imread(img)
        x.append(cv2.resize(full_size_image, (WIDTH,HEIGHT), interpolation=cv2.INTER_CUBIC))
        if img in classZero:
            y.append(0)
        elif img in classOne:
            y.append(1)
        else:
            return
    return x,y
```

7. Here we are taking 5000 samples to train the image data. Before training we are saving 5000 samples as two sets like positive and negative IDC. Printing Positive and negative type of IDC images also shown below.



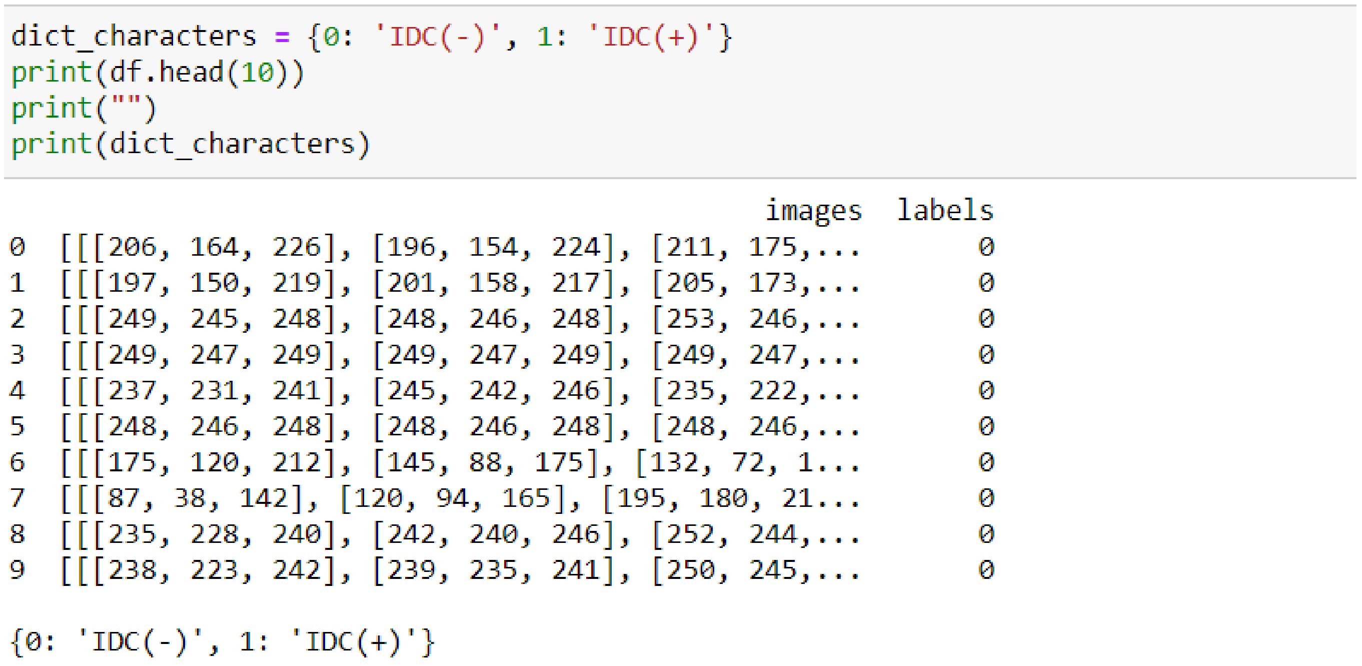
```
X,Y = proc_images(0,5000)
df = pd.DataFrame()
df["images"]=X
df["labels"]=Y
X2=df["images"]
Y2=df["labels"]
X2=np.array(X2)
imgs0=[]
imgs1=[]
imgs0 = X2[Y2==0] # (0 = no IDC, 1 = IDC)
imgs1 = X2[Y2==1]
```



```
def describeData(a,b):
    print('Total number of images: {}'.format(len(a)))
    print('Number of IDC(-) Images: {}'.format(np.sum(b==0)))
    print('Number of IDC(+) Images: {}'.format(np.sum(b==1)))
    print('Percentage of positive images: {:.2f}%'.format(100*np.mean(b))) #percentage is multiplication of mean 0f IDC(+)
    print('Image shape (Width, Height, Channels): {}'.format(a[0].shape))
describeData(X2,Y2)
```

```
Total number of images: 10000
Number of IDC(-) Images: 7541
Number of IDC(+) Images: 2459
Percentage of positive images: 24.59%
Image shape (Width, Height, Channels): (50, 50, 3)
```

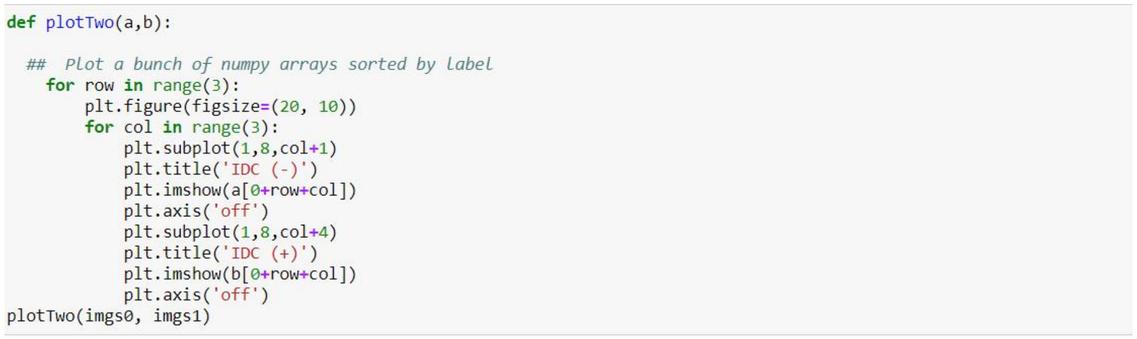
8. Using Dictnory function we are just showing the pixel intensity values. It has three channels Red ,Green,Blue.



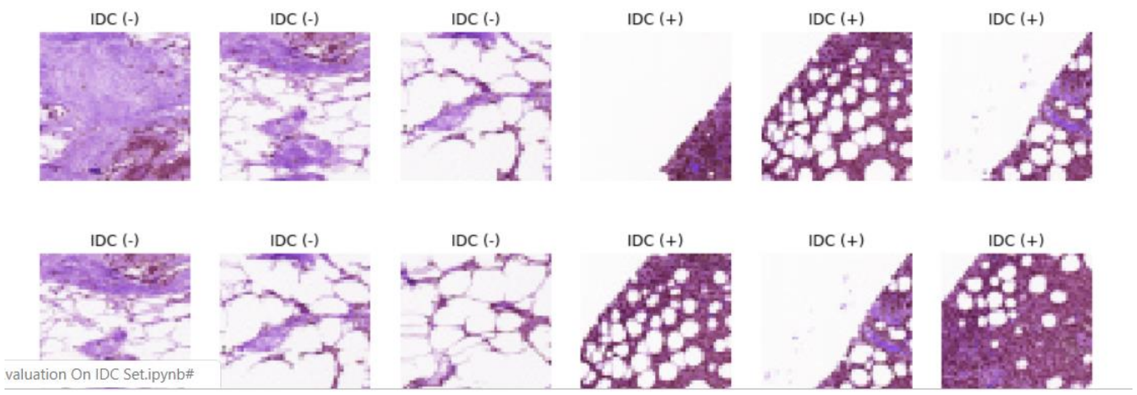
```
dict_characters = {0: 'IDC(-)', 1: 'IDC(+)'}
print(df.head(10))
print("")
print(dict_characters)
```

```
                                              images  labels
0  [[[206, 164, 226], [196, 154, 224], [211, 175,...       0
1  [[[197, 150, 219], [201, 158, 217], [205, 173,...       0
2  [[[249, 245, 248], [248, 246, 248], [253, 246,...       0
3  [[[249, 247, 249], [249, 247, 249], [249, 247,...       0
4  [[[237, 231, 241], [245, 242, 246], [235, 222,...       0
5  [[[248, 246, 248], [248, 246, 248], [248, 246,...       0
6  [[[175, 120, 212], [145, 88, 175], [132, 72, 1...       0
7  [[[87, 38, 142], [120, 94, 165], [195, 180, 21...       0
8  [[[235, 228, 240], [242, 240, 246], [252, 244,...       0
9  [[[238, 223, 242], [239, 235, 241], [250, 245,...       0

{0: 'IDC(-)', 1: 'IDC(+)'}
```

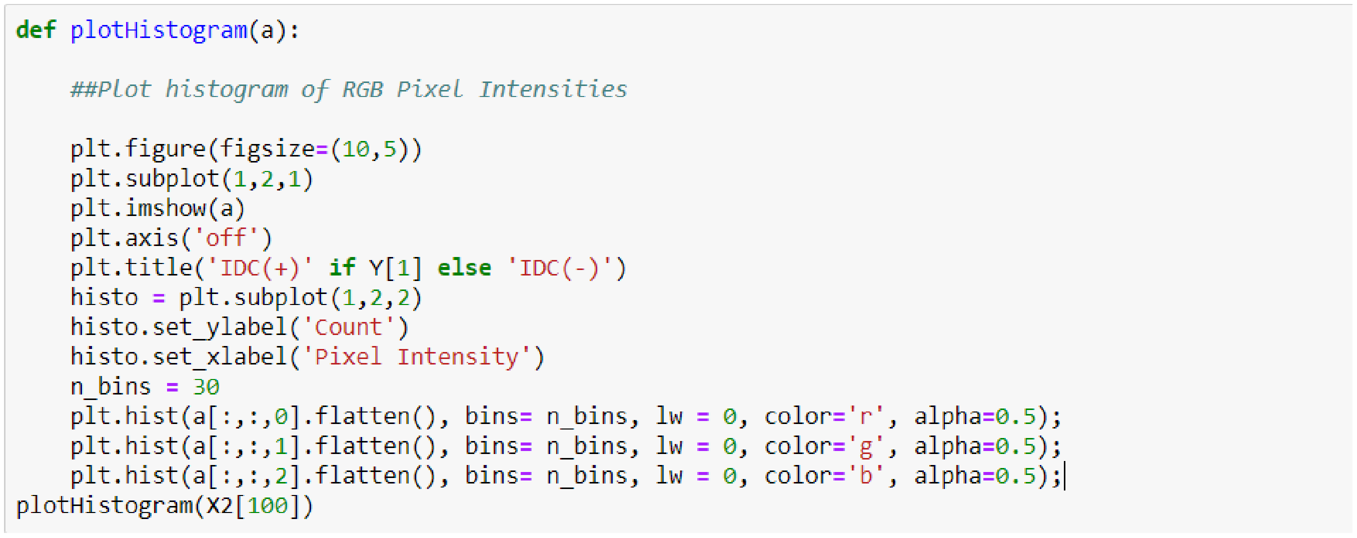
10.Here we are plotting bunch of numpy arrays sorted by label.



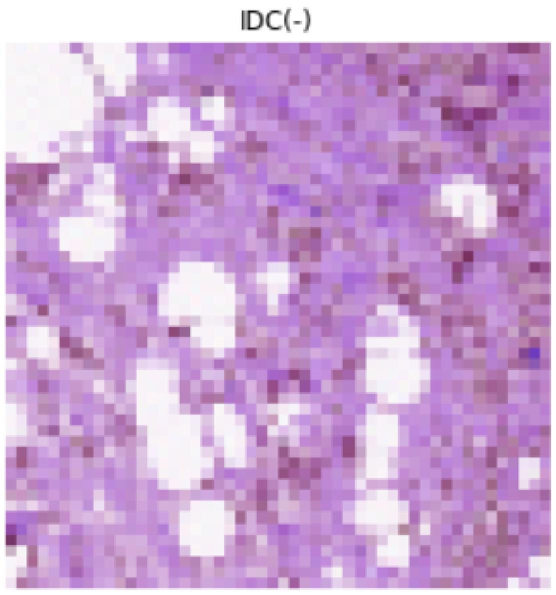
```
def plotTwo(a,b):

  ##  Plot a bunch of numpy arrays sorted by label
    for row in range(3):
        plt.figure(figsize=(20, 10))
        for col in range(3):
            plt.subplot(1,8,col+1)
            plt.title('IDC (-)')
            plt.imshow(a[0+row+col])
            plt.axis('off')
            plt.subplot(1,8,col+4)
            plt.title('IDC (+)')
            plt.imshow(b[0+row+col])
            plt.axis('off')
plotTwo(imgs0, imgs1)
```

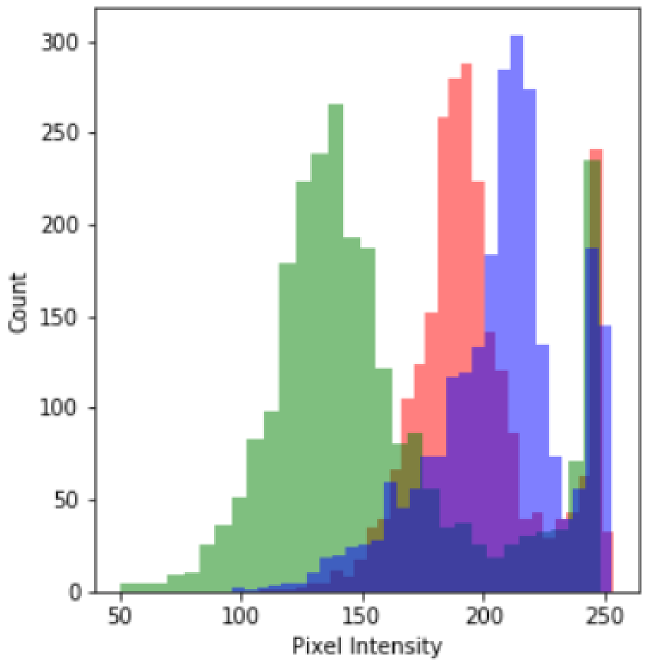


11.Plot Histogram of RGB pixel Intensities ,It helps to identify which part of the image has significance part.



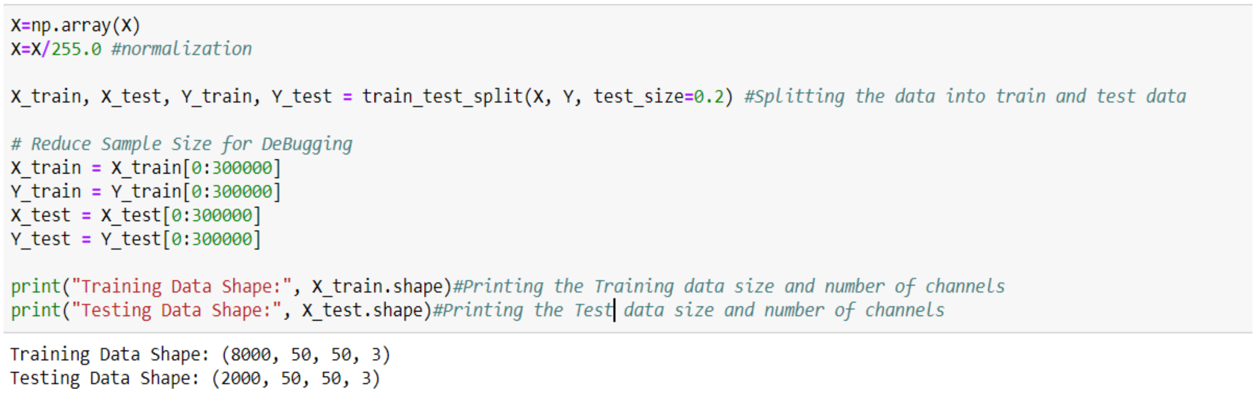
```
def plotHistogram(a):

    ##Plot histogram of RGB Pixel Intensities

    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    plt.imshow(a)
    plt.axis('off')
    plt.title('IDC(+)' if Y[1] else 'IDC(-)')
    histo = plt.subplot(1,2,2)
    histo.set_ylabel('Count')
    histo.set_xlabel('Pixel Intensity')
    n_bins = 30
    plt.hist(a[:,:,0].flatten(), bins= n_bins, lw = 0, color='r', alpha=0.5);
    plt.hist(a[:,:,1].flatten(), bins= n_bins, lw = 0, color='g', alpha=0.5);
    plt.hist(a[:,:,2].flatten(), bins= n_bins, lw = 0, color='b', alpha=0.5);
plotHistogram(X2[100])
```





12.Following Snippet showing Data normalization and Splitting the data into test and train samples. Also Printed the train data shape and test data shape.

Data Normalization: Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges.



```
X=np.array(X)
X=X/255.0 #normalization

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2) #Splitting the data into train and test data

# Reduce Sample Size for DeBugging
X_train = X_train[0:300000]
Y_train = Y_train[0:300000]
X_test = X_test[0:300000]
Y_test = Y_test[0:300000]

print("Training Data Shape:", X_train.shape)#Printing the Training data size and number of channels
print("Testing Data Shape:", X_test.shape)#Printing the Test data size and number of channels
```

```
Training Data Shape: (8000, 50, 50, 3)
Testing Data Shape: (2000, 50, 50, 3)
```

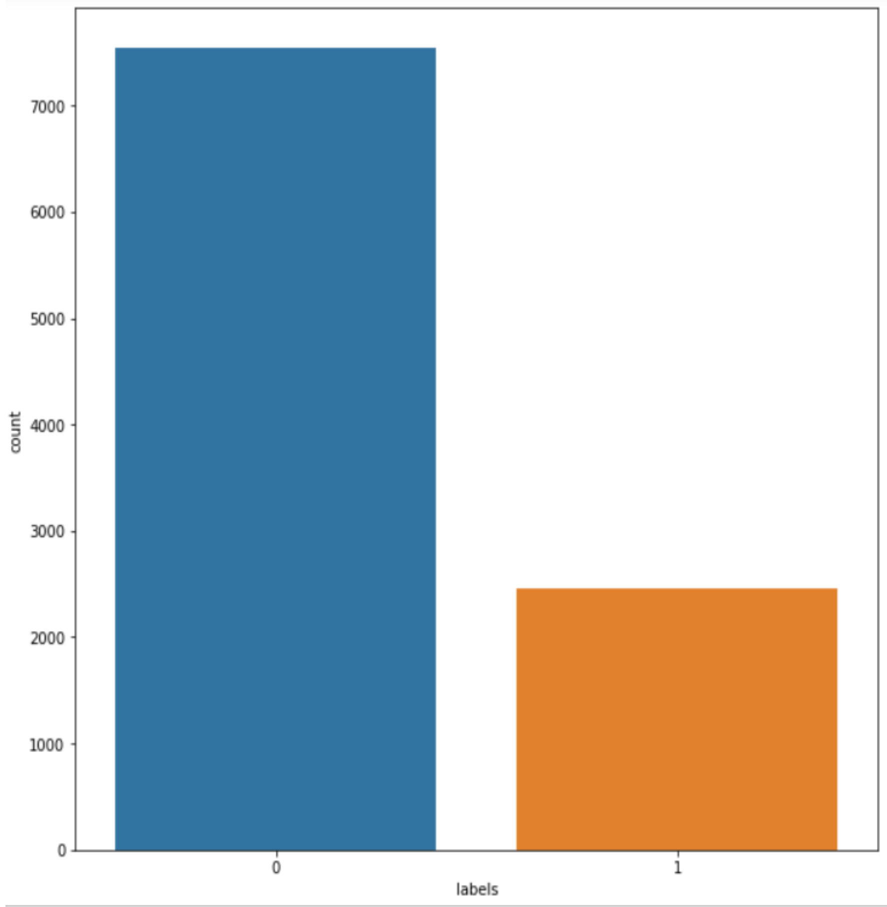
13.Here encoding to machine language as 0's and 1's and number of classes taken as two.

```
# Encode labels to hot vectors
Y_trainHot = to_categorical(Y_train, num_classes = 2)
Y_testHot = to_categorical(Y_test, num_classes = 2)
```

14.Here we are plotting the bar graph to show number of IDC positive and negative images present in the given samples.

```
lab = df['labels']
dist = lab.value_counts()
sns.countplot(lab)
print(dict_characters)
```

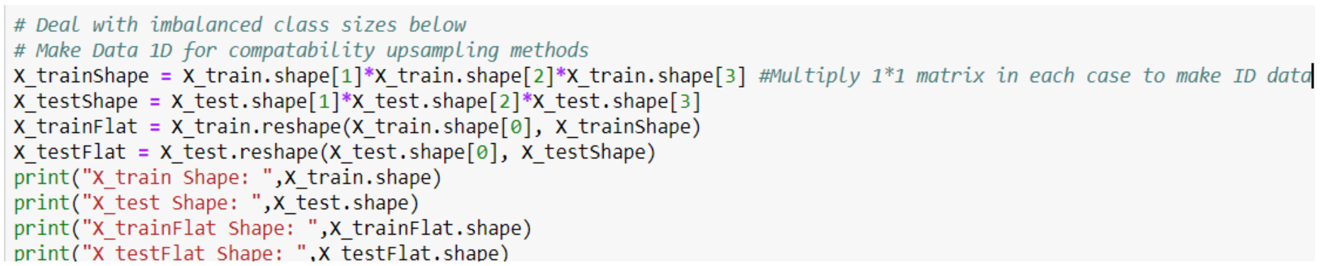
```
{0: 'IDC(-)', 1: 'IDC(+)'}
```



From the graph we observed that number negative images more than positive images .It leads to wrong predictions on test data further lead to over fitting of data. To avoid this we used random sampling to make imbalanced data to balanced data ,So that It will equal number of samples for both IDC positive and negative by dropping negative side or adding positive samples to IDC(+) side.

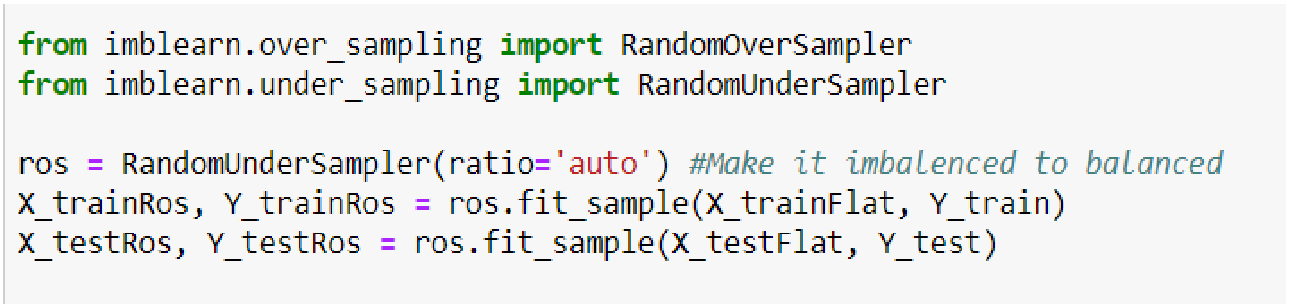
The corresponding code snippet and output are shown below.

15.Making ID data to reduce complexity.

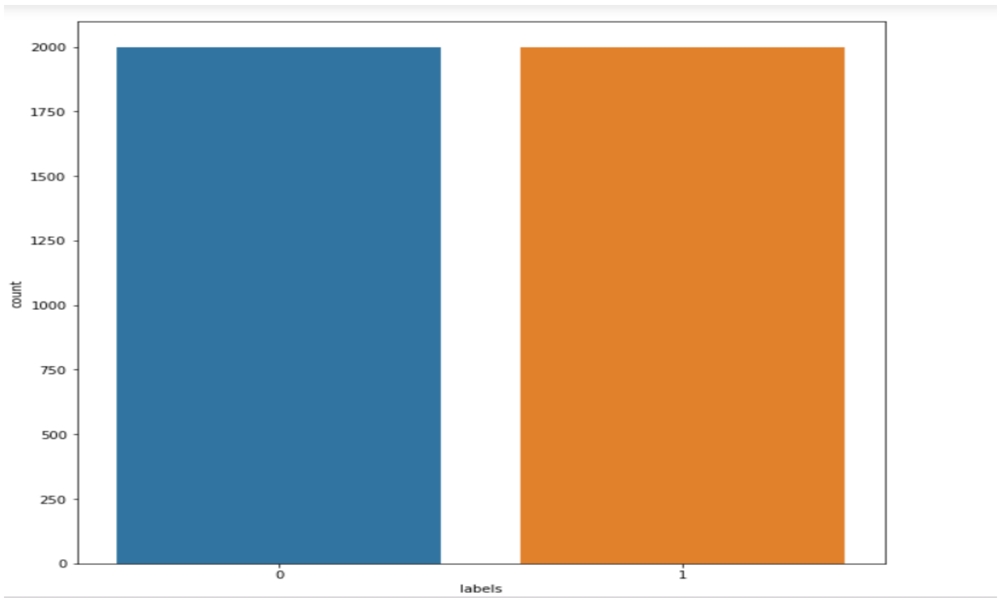


```
# Deal with imbalanced class sizes below
# Make Data 1D for compatability upsampling methods
X_trainShape = X_train.shape[1]*X_train.shape[2]*X_train.shape[3] #Multiply 1*1 matrix in each case to make ID data
X_testShape = X_test.shape[1]*X_test.shape[2]*X_test.shape[3]
X_trainFlat = X_train.reshape(X_train.shape[0], X_trainShape)
X_testFlat = X_test.reshape(X_test.shape[0], X_testShape)
print("X_train Shape: ",X_train.shape)
print("X_test Shape: ",X_test.shape)
print("X_trainFlat Shape: ",X_trainFlat.shape)
print("X_testFlat Shape: ",X_testFlat.shape)
```

15.Random Sampling to make imbalanced to balanced data.

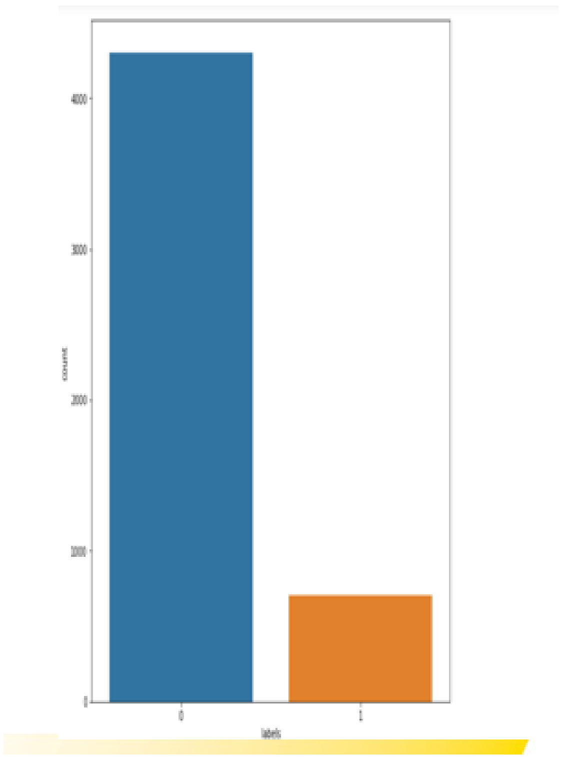


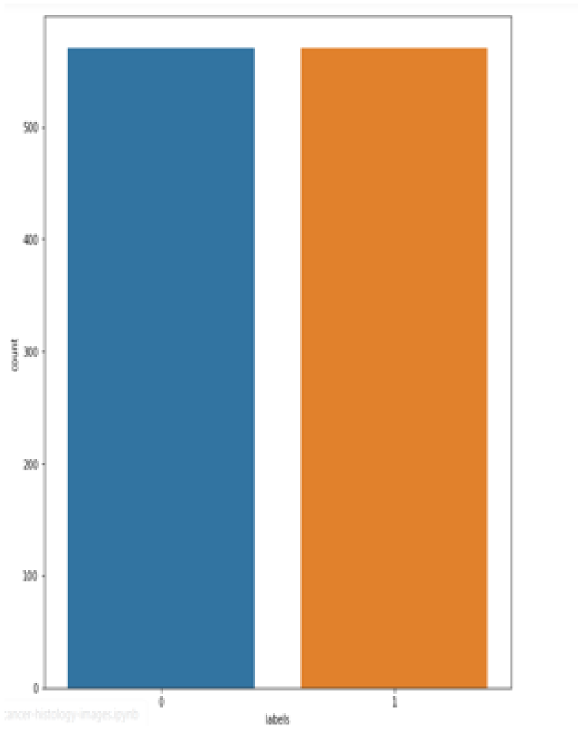
```
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

ros = RandomUnderSampler(ratio='auto') #Make it imbalenced to balanced
X_trainRos, Y_trainRos = ros.fit_sample(X_trainFlat, Y_train)
X_testRos, Y_testRos = ros.fit_sample(X_testFlat, Y_test)
```



Data Balancing technique for imbalanced data.

a.Before Sampling   b. After Sampling





16.The below code snippets showing the old class weights (before sampling) and new class weights (after sampling).

```
from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.unique(Y_train), Y_train)
print("Old Class Weights: ",class_weight)
from sklearn.utils import class_weight
class_weight2 = class_weight.compute_class_weight('balanced', np.unique(Y_trainRos), Y_trainRos)
print("New Class Weights: ",class_weight2)
```

```
Old Class Weights:  [0.66644452 2.002002  ]
New Class Weights:  [1. 1.]
```

17.Machine learning Models Implementation:

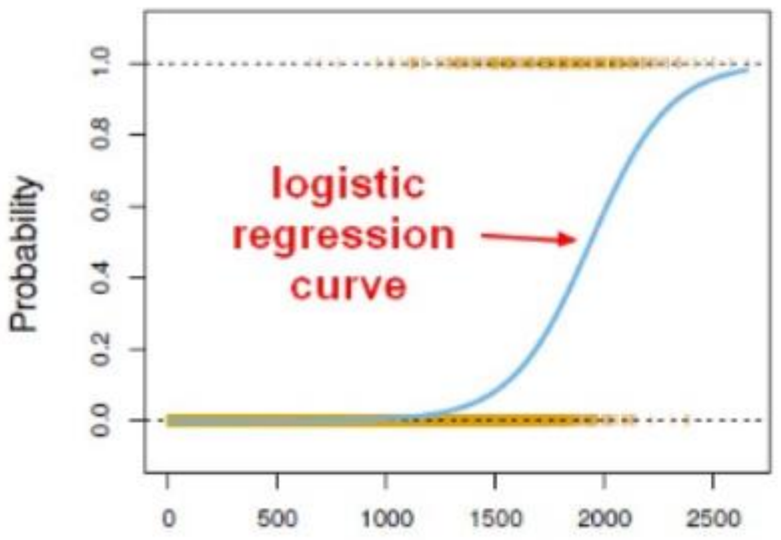
Model-1: Logistic Regression

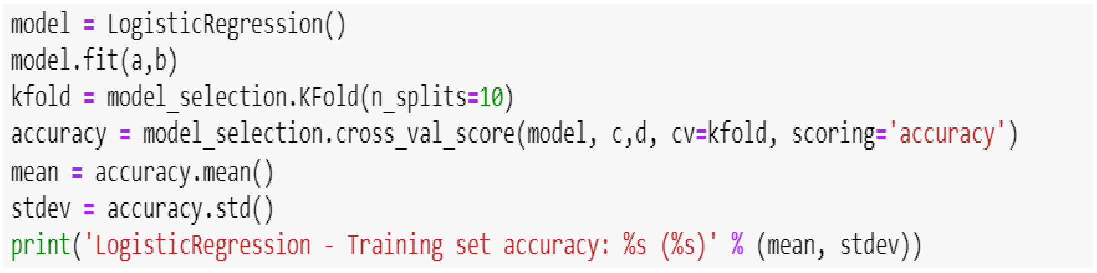
Logistic Regression using K-fold Cross Validation:

K-fold cross validation:

K-Fold CV is where a given data set is split into a ***K*** number of sections/folds where each fold is used as a testing set at some point. Here we have taken 10-Fold cross validation(K=10). Here, the data set is split into 10 folds. In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 10 folds have been used as the testing set**.**

Logistic Regression: It is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.





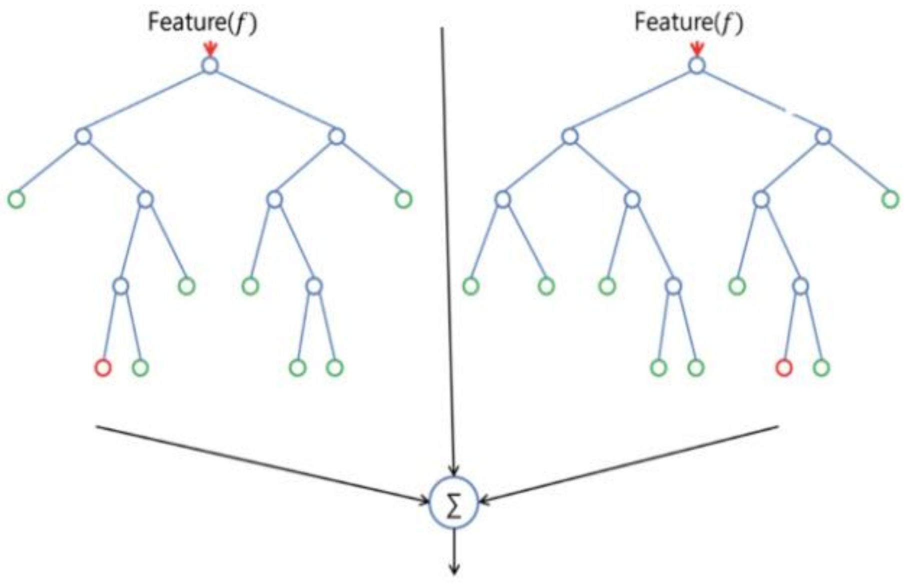
```
model = LogisticRegression()
model.fit(a,b)
kfold = model_selection.KFold(n_splits=10)
accuracy = model_selection.cross_val_score(model, c,d, cv=kfold, scoring='accuracy')
mean = accuracy.mean()
stdev = accuracy.std()
print('LogisticRegression - Training set accuracy: %s (%s)' % (mean, stdev))
```

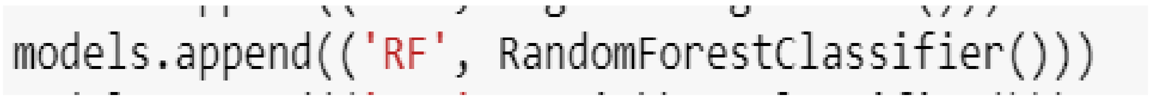
Here we have taken K=10.

Attained accuracy is 0.891

Model-2: Random Forest Classifier

- Random forest will build multiple decision trees
- These decision trees are them merged to get accurate prediction
- This technique can both used in classification and regression



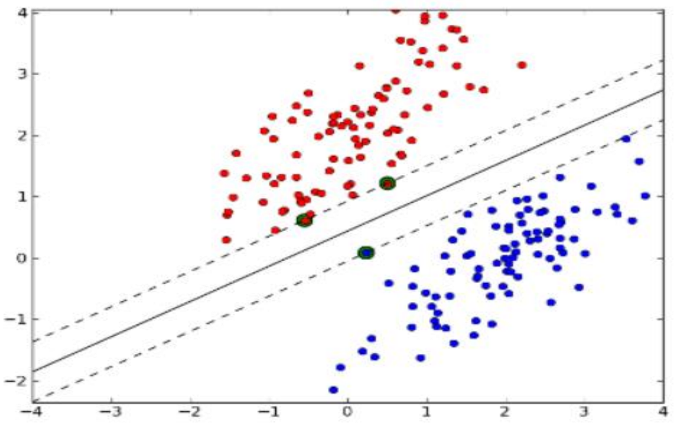


```
models.append(('RF', RandomForestClassifier()))
```

- Accuracy using Random Forest Classifier is 0.894

Model-3: SVM

- Support Vector machine is advanced way of clustering the data and divides the data into best fit linear hyperplane.
- The svm.svc splits the data in to clusters and uses different kernel tricks to test it
- It is a supervised learning technique
- Linear Kernel: K(X,Y)=XTY



```
models.append(('SVM', SVC()))
```

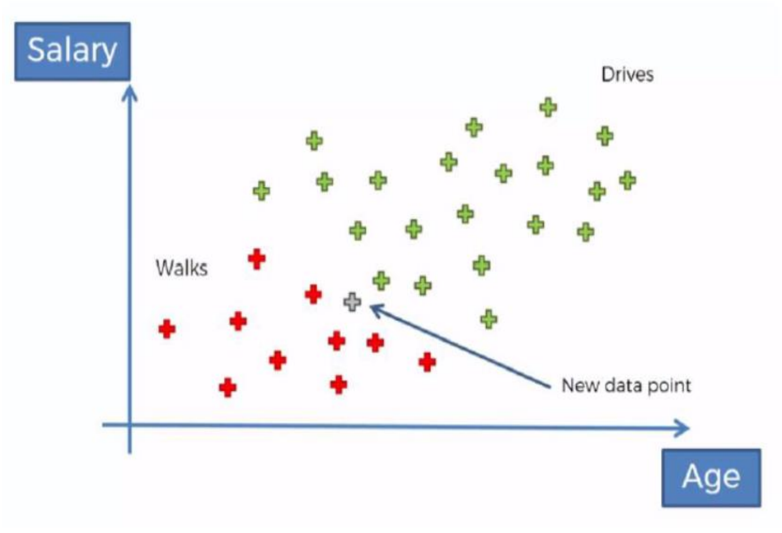
- Accuracy is 0.869

Model-4: Naive Bayes':

This technique is based on Bayes' Theorem

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Bayes' Theorem



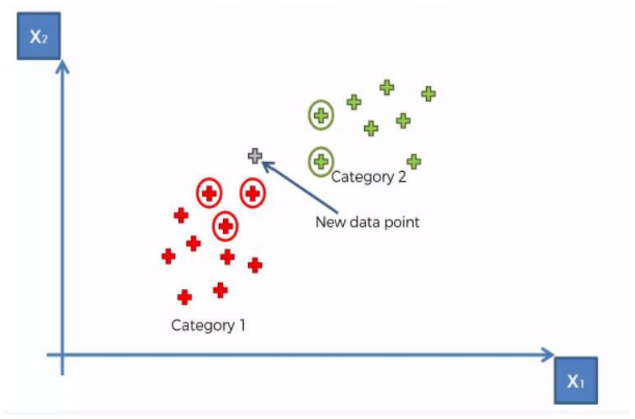
- According to this algorithm, given a class presence of one feature is independent of the other feature
- Using this assumption this classifier classifies a given data point.

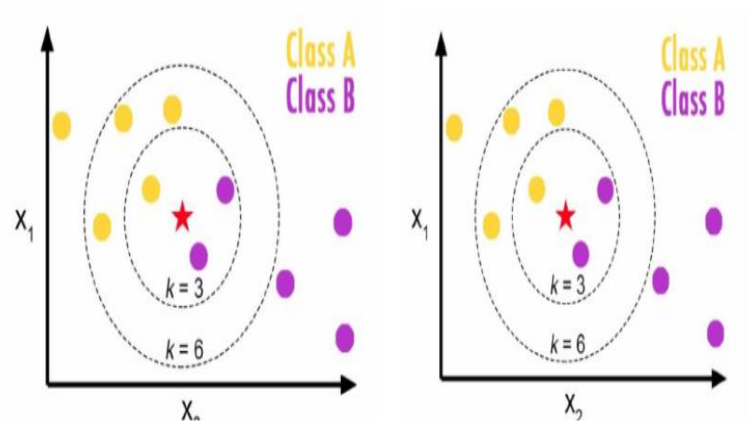
```
models.append(('GNB', GaussianNB()))
```

- Accuracy is 0.7510

Model-5: k-nearest neighbors classification algorithm

- Since it is classification algorithm, it decides which category the data point belongs to as shown below.
- The value "k" will determine how many data points around should be considered to make classification as shown in the below image.



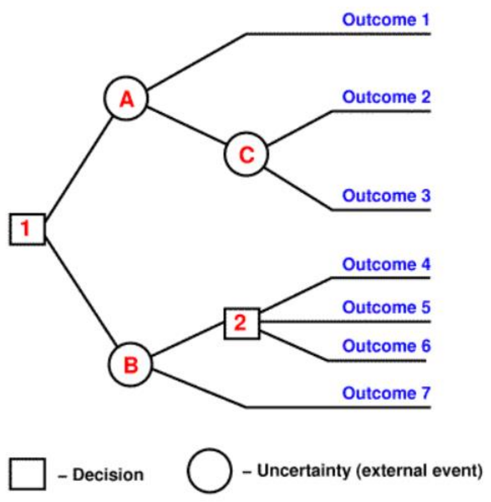


```
models.append(('KNN', KNeighborsClassifier()))
```

- Accuracy obtained as 0.874

Model-6: Decision Tree Classifier: It is the most ground-breaking and prevalent device for classification and prediction methodology. A Decision tree is a flowchart like tree structure, where each inner node means a test on a trait, each branch speaks to a result of the test, and each leaf node (terminal node) holds a class name

- It classifies and goes down in a hierarchical manner from root node to the leaf node
- Whole process is done to classify the given instance



```
models.append(('DTC', DecisionTreeClassifier()))
```

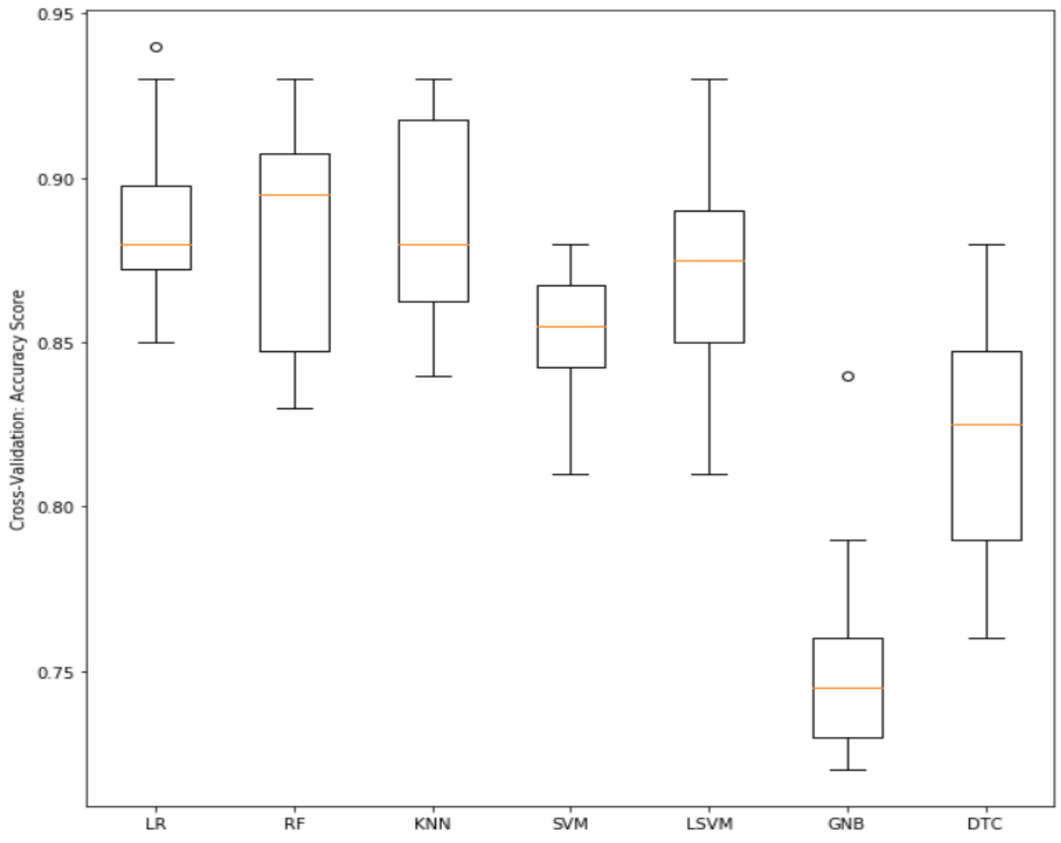
- Accuracy obtained as 0.871
- A minute disadvantage of this decision tree is that Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Also, small change in the data can cause a completely different tree being produced which is called as variance.

The code snippet for all machine learning models are given below:

```python
# Compare Performance of Classification Algorithms
def compareABunchOfDifferentModelsAccuracy(a,b,c,d):

    print('')
    print('Compare Multiple Classifiers:')
    print('')
    print('K-Fold Cross-Validation Accuracy:')
    print('')
    models = []
    models.append(('LR', LogisticRegression()))
    models.append(('RF', RandomForestClassifier()))
    models.append(('KNN', KNeighborsClassifier()))
    models.append(('SVM', SVC()))
    models.append(('LSVM', LinearSVC()))
    models.append(('GNB', GaussianNB()))
    models.append(('DTC', DecisionTreeClassifier()))
```

```python
    resultsAccuracy = []
    names = []
    for name, model in models:
        model.fit(a, b)
        kfold = model_selection.KFold(n_splits=10)
        accuracy_results = model_selection.cross_val_score(model, c, d, cv=kfold, scoring='accuracy')
        resultsAccuracy.append(accuracy_results)
        names.append(name)
        accuracyMessage = "%s: %f (%f)" % (name, accuracy_results.mean(), accuracy_results.std())
        print(accuracyMessage)
    # boxplot algorithm comparison
    fig = plt.figure()
    fig.suptitle('Algorithm Comparison: Accuracy')
    ax = fig.add_subplot(111)
    plt.boxplot(resultsAccuracy)
    ax.set_xticklabels(names)
    ax.set_ylabel('Cross-Validation: Accuracy Score')
    plt.show()
    return
compareABunchOfDifferentModelsAccuracy(X_trainFlat, Y_train, X_testFlat, Y_test)
```

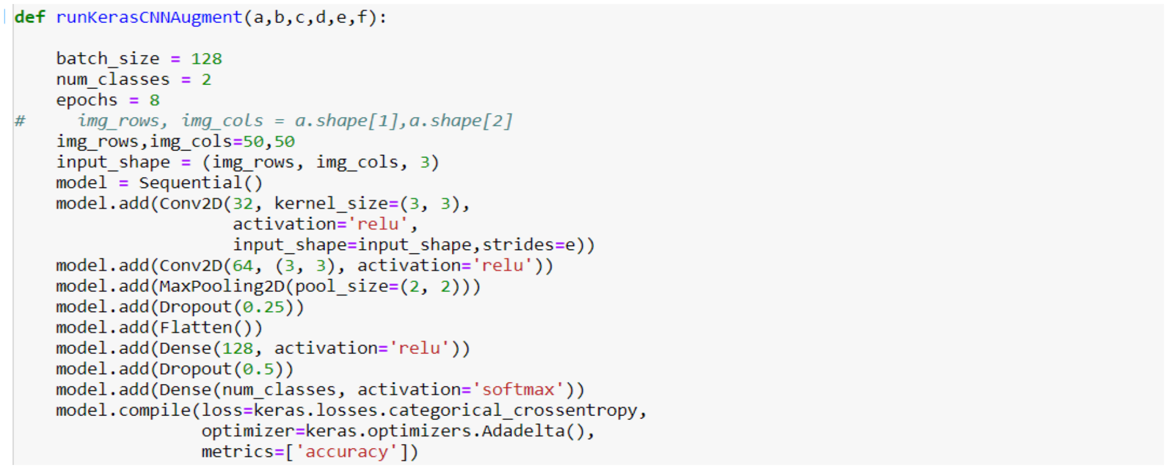
Accuracy Comparison:



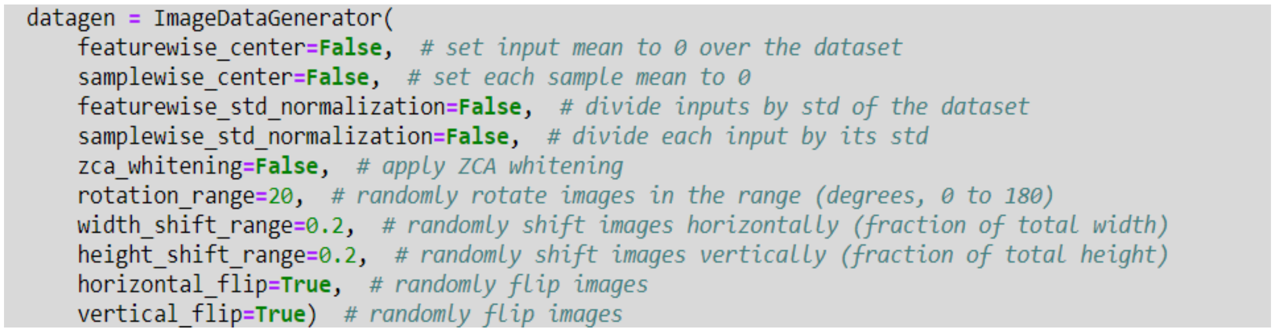
Deep Learning Approaches:

CNN:

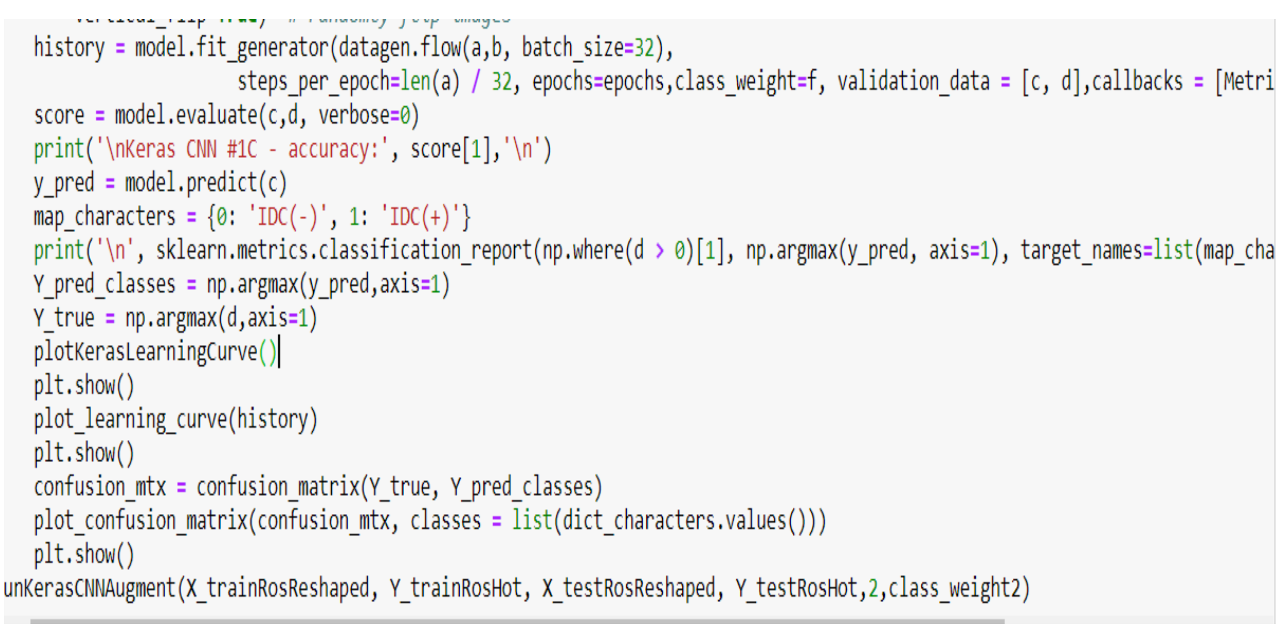
- We have trained the model using neural networks approach.
- We used CNN model to train the model and tuned the hyper parameters to attain best accuracy on train as well as test data.
- A **Convolutional Neural Network (Convent/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a Convent is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics.



```
def runKerasCNNAugment(a,b,c,d,e,f):

    batch_size = 128
    num_classes = 2
    epochs = 8
#     img_rows, img_cols = a.shape[1],a.shape[2]
    img_rows,img_cols=50,50
    input_shape = (img_rows, img_cols, 3)
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=input_shape,strides=e))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer=keras.optimizers.Adadelta(),
                  metrics=['accuracy'])
```



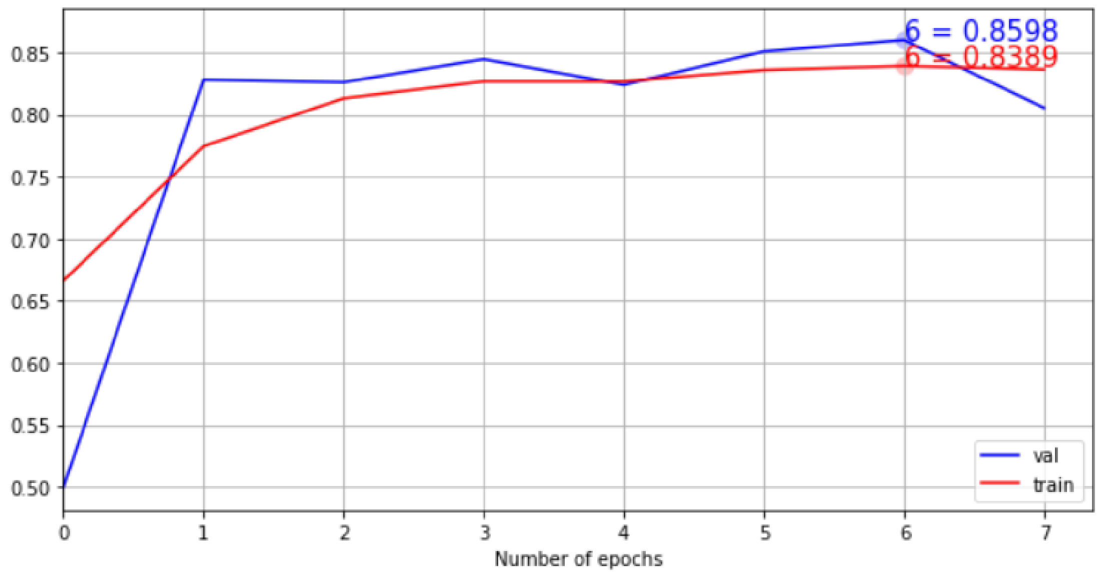
```
datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=20,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.2,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.2,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=True,  # randomly flip images
    vertical_flip=True)  # randomly flip images
```

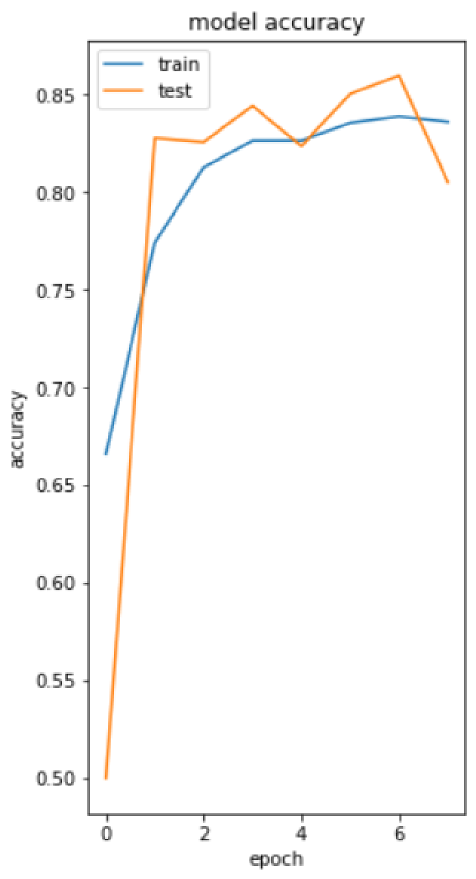


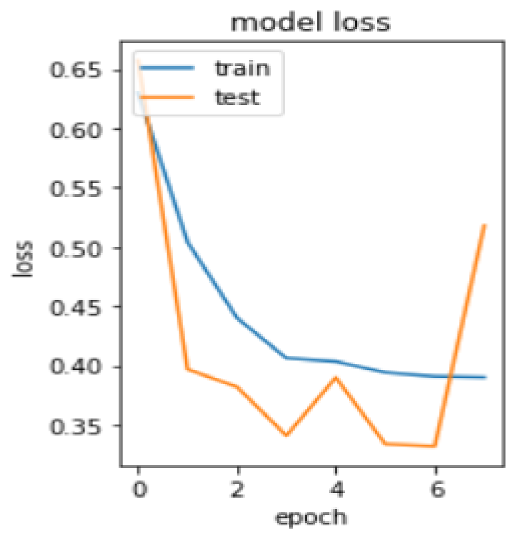
```
  history = model.fit_generator(datagen.flow(a,b, batch_size=32),
                      steps_per_epoch=len(a) / 32, epochs=epochs,class_weight=f, validation_data = [c, d],callbacks = [Metri
  score = model.evaluate(c,d, verbose=0)
  print('\nKeras CNN #1C - accuracy:', score[1],'\n')
  y_pred = model.predict(c)
  map_characters = {0: 'IDC(-)', 1: 'IDC(+)'}
  print('\n', sklearn.metrics.classification_report(np.where(d > 0)[1], np.argmax(y_pred, axis=1), target_names=list(map_cha
  Y_pred_classes = np.argmax(y_pred,axis=1)
  Y_true = np.argmax(d,axis=1)
  plotKerasLearningCurve()
  plt.show()
  plot_learning_curve(history)
  plt.show()
  confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
  plot_confusion_matrix(confusion_mtx, classes = list(dict_characters.values()))
  plt.show()
unKerasCNNAugment(X_trainRosReshaped, Y_trainRosHot, X_testRosReshaped, Y_testRosHot,2,class_weight2)
```

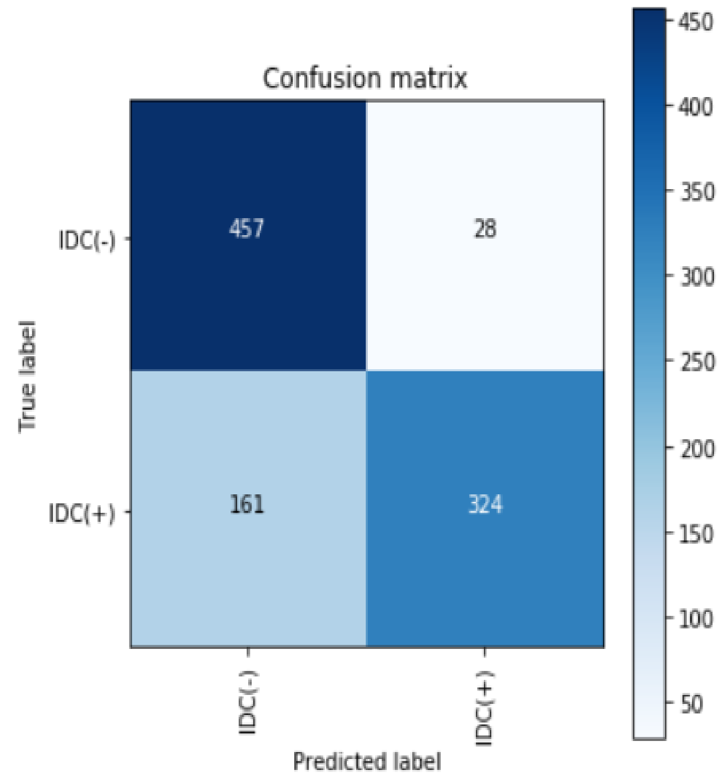
```
Epoch 1/8
124/123 [==============================] - 9s 72ms/step - loss: 0.6295 - acc: 0.6657 - val_loss: 0.6571 - val_acc: 0.5000
Epoch 2/8
124/123 [==============================] - 10s 77ms/step - loss: 0.5032 - acc: 0.7750 - val_loss: 0.3968 - val_acc: 0.8278
Epoch 3/8
124/123 [==============================] - 10s 78ms/step - loss: 0.4400 - acc: 0.8125 - val_loss: 0.3821 - val_acc: 0.8258
Epoch 4/8
124/123 [==============================] - 9s 76ms/step - loss: 0.4056 - acc: 0.8269 - val_loss: 0.3409 - val_acc: 0.8443
Epoch 5/8
124/123 [==============================] - 9s 76ms/step - loss: 0.4038 - acc: 0.8269 - val_loss: 0.3900 - val_acc: 0.8237
Epoch 6/8
124/123 [==============================] - 9s 76ms/step - loss: 0.3939 - acc: 0.8356 - val_loss: 0.3340 - val_acc: 0.8505
Epoch 7/8
124/123 [==============================] - 10s 81ms/step - loss: 0.3914 - acc: 0.8393 - val_loss: 0.3321 - val_acc: 0.8598
Epoch 8/8
124/123 [==============================] - 10s 77ms/step - loss: 0.3904 - acc: 0.8357 - val_loss: 0.5179 - val_acc: 0.8052

Keras CNN #1C - accuracy: 0.8051546389294654

              precision    recall  f1-score   support

      IDC(-)       0.74      0.94      0.83       485
      IDC(+)       0.92      0.67      0.77       485

   micro avg       0.81      0.81      0.81       970
   macro avg       0.83      0.81      0.80       970
weighted avg       0.83      0.81      0.80       970
```

The validation and training accuracy with respect to number of epochs are given below.




model accuracy
train
test
accuracy
epoch
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
2
4
6


model loss
train
test
loss
epoch
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
2
4
6



Comparison & Performance Evaluation of Models:

Model Name	Attained Accuracy
Logistic Regression	0.891
Support Vector Machine	0.869
K-Nearest Neighbors Classifier	0.874
Random Forest Classifier	0.894
Naive Bayes’	0.751
Decision Tree Classifier	0.871
CNN Approach	0.841

Conclusion:

- Hence, we can say that both Machine learning and Deep learning approaches are fit for this data to test whether the image has IDC or Not.
- And There is no large variation in accuracy when we changed the number of samples
- We have changed and observed the accuracy.
- For 5000 samples the training accuracy is 0.891
- For 10000 Samples we attained accuracy as 0.90

Future Scope:

- This structure is easy to deploy in doctor's office and would save additional cost and time required to put in for testing purposes in order to identify if the cell contains IDC or not.
- This project can also be deployed on a website and hosted virtually thereby providing ease of usage because of its easy to understand interface.

Members Contribution:

As concerning to learn everything, we were implemented each part of the code together. Each member in our team involved in every aspect of code development gain exposure on both Machine Learning and Deep Learning Models.

Presentation & Demo Links:

https://umkc.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=19f40616-50b8-4662-a256-aa4c01602847
https://umkc.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=30839b97-8487-4a54-8463-aa4c0175ca93

GitHub Link:

https://github.com/lakshmanamettu/Python-Deep-Learning-project

References:

- https://www.ncbi.nlm.nih.gov/search/all/?term=idc%20breast%20cancer
- https://towardsdatascience.com/an-intuitive-understanding-to-neural-style-transfer-e85fd80394be
- https://stackoverflow.com/questions/22741319/what-does-random-sample-method-in-python-do
- https://www.kaggle.com/simjeg/lymphoma-subtype-classification-fl-vs-cll