PROJECT ON IMAGE CLASSIFICATION TO DETECT DIABETIC RETINOAPTHY TO PREVENT BLINDNESS

SUBJECT - DATA SCIENCE PROGRAMMING

PROFESSOR – DR. BALAJI PADMANABHAN

TEAM –
KHANINDER KOMMAGONI
VARUN YAVAGAL
PARITOSH KALLA

Table of Content

Introduction

Motivation –

Vision loss due to diabetic eye disease is on the rise and it is expected to reach epidemic proportions globally in the next few decades. In 2017, ~425 million people worldwide had diabetes, and this number is estimated to increase to 642 million by 2040. Diabetic retinopathy (DR) is the most common and insidious microvascular complication of diabetes, and can progress asymptomatically until a sudden loss of vision occurs. Almost all patients with type 1 diabetes mellitus and ~60% of patients with type 2 diabetes mellitus will develop retinopathy during the first 20 years from onset of diabetes. However, DR often remains undetected until it progresses to an advanced vision-threatening stage.

Problem Statement –

The current state of DR screening in the real world is based on assessment of color fundus photographs by a retina specialist or a trained grader, leaves a large proportion of patients undiagnosed and therefore receiving medical help too late, in part due to low adherence and access to retina screening visits. In-person expert examinations are impractical and unsustainable given the pandemic size of the diabetic population. Notwithstanding, early detection and prevention of DR progression are essential to mitigate the rising threat of DR.

The proposed Machine Learning models were intended to foresee future DR movement, characterized as 5-stages on the Early Treatment Diabetic Retinopathy Severity Scale. The goal is to scale efforts through technology; to gain the ability to automatically screen images for diabetic retinopathy and provide information on how severe the condition may be.

Related academic work –

Computer-aided diagnosis of diabetic retinopathy has been explored in the past to reduce the burden on ophthalmologist's and mitigate diagnostic inconsistencies between manual readers. Automated methods to detect diabetic retinopathy and reliably grade fundoscopic images of diabetic retinopathy patients have been active areas of research in computer vision. The first artificial neural networks explored the ability to classify patches of normal retina without blood vessels, normal retinas with blood vessels, pathologic retinas with exudates, and pathologic retinas with microaneurysms. The accuracy of being able to detect diabetic retinopathy compared to normal patches of retina was reported at 74%.

Past studies using various high bias, low variance digital image processing techniques have performed well at identifying one specific feature used in the detection of subtle disease such as the use of top-hat algorithm for microaneurysm detection. However, a variety of other features besides microaneurysms are efficacious for disease detection.

Additional methods of detecting microaneurysms and grading DR involving k-NN, support vector machines, and ensemble-based methods have yielded sensitivities and specificities within the 90% range using various feature extraction techniques and preprocessing algorithms.

Previous CNN studies for DR fundus images achieved sensitivities and specificities in the range of 90% for binary classification categories of normal or mild vs moderate or severe on much larger private datasets of 80,000 to 120,000 images. However, accuracy measures for the detection of four classes of DR, that is: no DR (R0), mild (R1), moderate (R2), and severe (R3) depend nontrivially on disease graded class collection ratios. While R0 and R3 stages are capable of achieving high sensitivity, the R1 and R2 computed recall rates are often low. Experiments from publicly available datasets suggest this is primarily attributable to the relative difficulty of detecting early stage DR. Furthermore, current accuracies for R1 and R2 stages are reported at 0% and 41%, respectively.

Related Industry work –

When people with diabetes visit their general practitioner, they're often referred to an ophthalmologist, who can check their eyes for signs of diabetic retinopathy. The disease damages the light-sensitive layer of tissue at the back of the eye known as the retina and is a leading cause of blindness, resulting in up to 24,000 cases each year in adults in the United States. But when diagnosed before symptoms appear, the disease can usually be managed, and the worst outcome avoided. Michael Abràmoff, a retinal specialist and computer scientist at the University of Iowa in Iowa City developed IDx-DR, an artificial intelligence (AI) system that can tell in minutes whether a person has a more-than-mild case of diabetic retinopathy. Such cases comprise only about 10% of people with diabetes, so under this AI system, ophthalmologists would have to examine many fewer people.

IDx-DR is the first device to be approved by the US Food and Drug Administration (FDA) to provide a screening decision without the need for a clinician. But it is not the only AI-based tool that is poised to transform the field of ophthalmology.

Google and its sister organization Verily, the life sciences division of Alphabet, are touting a new artificial intelligence tool that's enabling wider access to care and screening for diabetic retinopathy and diabetic macular edema for patients at a hospital in India.

Problem –

Imagine being able to detect blindness before it happened.

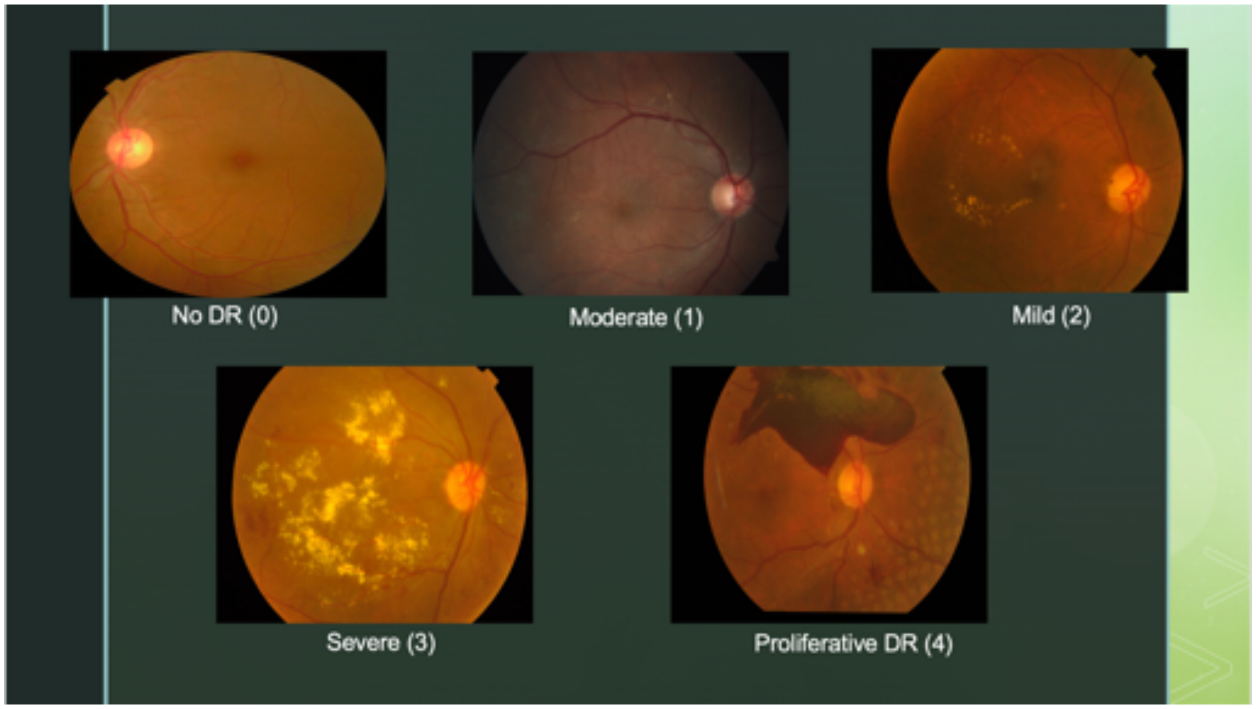
Millions of people suffer from diabetic retinopathy, the leading cause of blindness among working aged adults. Through this project we hope to detect and prevent this disease among people living in rural areas where medical screening is difficult to conduct.

Currently, technicians travel to these rural areas to capture images and then rely on highly trained doctors to review the images and provide diagnosis. Their goal is to scale their efforts through technology; to gain the ability to automatically screen images for disease and provide information on how severe the condition may be.

In this project, we have built a machine learning model to speed up disease detection. We have worked with thousands of images collected in rural areas to help identify diabetic retinopathy automatically. These models may be used to detect other sorts of diseases in the future, like glaucoma and macular degeneration.

Data –

The data for the project is sourced from Kaggle (https://www.kaggle.com/c/aptos2019-blindness-detection) where we have 3662 retina images divided into 5 classes labeled as (0,1,2,3,4).



Libraries –

1. Scikit Learn
2. Keras
3. Numpy
4. Pandas
5. TQDM

Methods/Techniques –

Decision Tree Algorithm –

A decision tree is a flowchart-like tree structure where an internal node represents feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

Gaussian Naïve Bayes Algorithm –

Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets. Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features.

Support Vector Machines Algorithm –

Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane (MMH) that best divides the dataset into classes.

Ensemble Algorithm –

In the world of Statistics and Machine Learning, Ensemble learning techniques attempt to make the performance of the predictive models better by improving their accuracy. Ensemble Learning is a process using which multiple machine learning models (such as classifiers) are strategically constructed to solve a particular problem.

Convolutional Neural Network (Deep Learning) –

A Convolutional Neural Network (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 CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN have the ability to learn these filters/characteristics.

Evaluation –

Preprocessing – We have used TQDM library to import the images into the python environment and saved them into an array, next we have converted the 3D array to 2D array so that it can be used into the classifier for predictions. We have split the images into 80-20 for training and testing purposes.

Version 1 - In this version we have filtered the images for two classes (0 and 4) and assessed the data using Decision Tree, SVC and Gaussian Naïve Bayes algorithm, the total images were 2100.

Version 2 – In this version to add complexity we have filtered the images for three classes (0,2,4) and applied the same machine learning algorithm as in version 1, this time the total images were 3099.

Version 3 – In this version to add to the complexity we have changed our approach and kept the classes as 0,2,4 and assessed the data using ensemble of Decision Tree, SVC and Gaussian Naïve Bayes and tested whether the ensemble has any effect on the performance of the model.

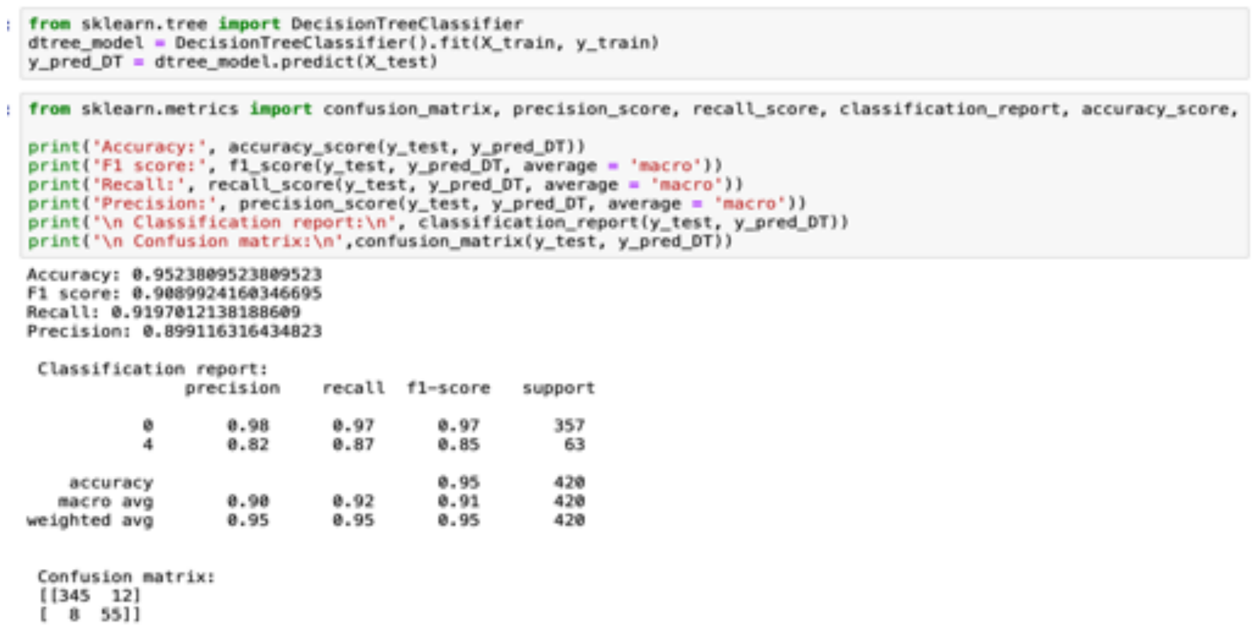
Version 4 – In this version we have added all the classes to the model to increase complexity and applied Decision Tree, SVC and Gaussian Naïve Bayes algorithm to the data, the total images are 3662.

Version 5 – To further improve the model performance we have applied Deep Learning (Convolutional Neural Network) to the entire dataset, we have used one dense layer and 'relu' as the activation function.

Results –

Version 1 –

Decision Tree Algorithm -

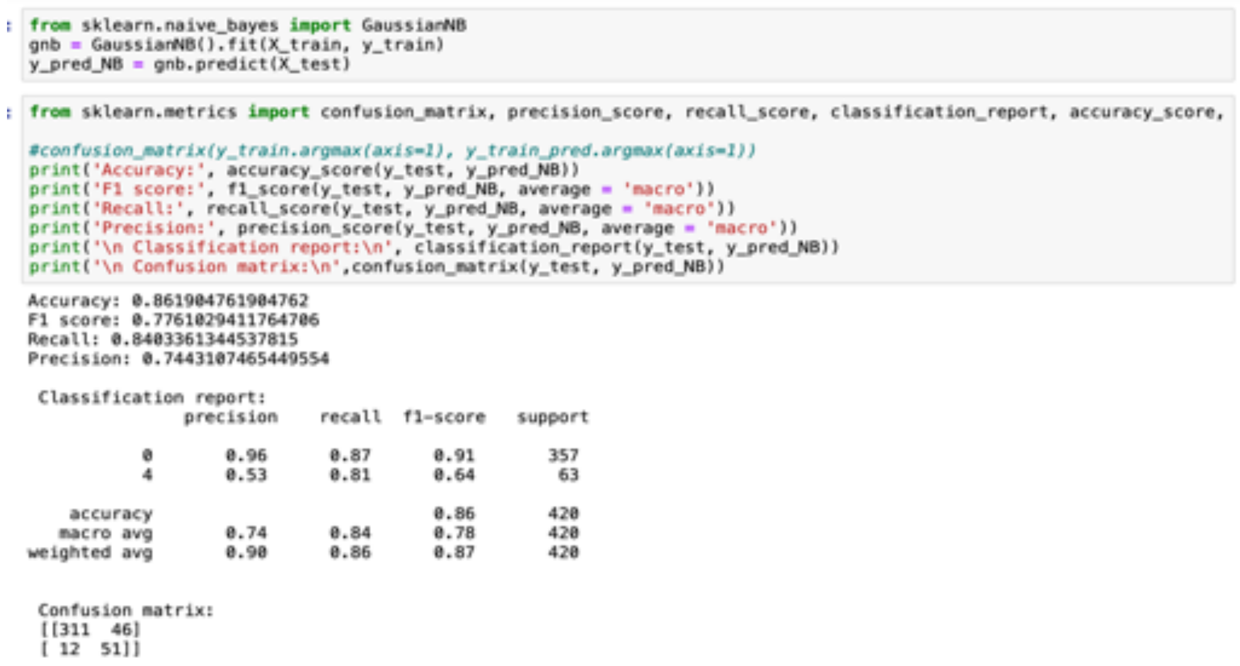


```
from sklearn.tree import DecisionTreeClassifier
dtree_model = DecisionTreeClassifier().fit(X_train, y_train)
y_pred_DT = dtree_model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, classification_report, accuracy_score,

print('Accuracy:', accuracy_score(y_test, y_pred_DT))
print('F1 score:', f1_score(y_test, y_pred_DT, average = 'macro'))
print('Recall:', recall_score(y_test, y_pred_DT, average = 'macro'))
print('Precision:', precision_score(y_test, y_pred_DT, average = 'macro'))
print('\n Classification report:\n', classification_report(y_test, y_pred_DT))
print('\n Confusion matrix:\n',confusion_matrix(y_test, y_pred_DT))
```

```
Accuracy: 0.9523809523809523
F1 score: 0.9089924160346695
Recall: 0.9197012138188609
Precision: 0.899116316434823

 Classification report:
               precision    recall  f1-score   support

           0       0.98      0.97      0.97       357
           4       0.82      0.87      0.85        63

    accuracy                           0.95       420
   macro avg       0.90      0.92      0.91       420
weighted avg       0.95      0.95      0.95       420


 Confusion matrix:
 [[345  12]
 [  8  55]]
```

Gaussian Naïve Bayes Algorithm –

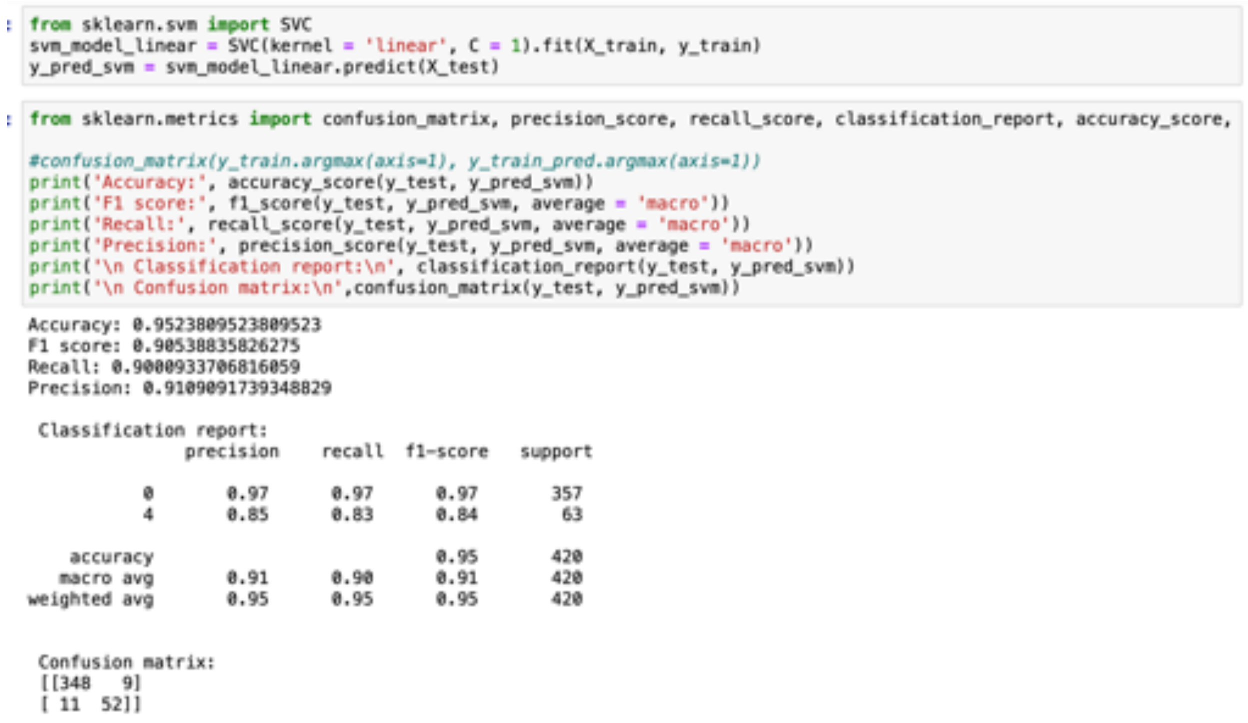


```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB().fit(X_train, y_train)
y_pred_NB = gnb.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, classification_report, accuracy_score,

#confusion_matrix(y_train.argmax(axis=1), y_train_pred.argmax(axis=1))
print('Accuracy:', accuracy_score(y_test, y_pred_NB))
print('F1 score:', f1_score(y_test, y_pred_NB, average = 'macro'))
print('Recall:', recall_score(y_test, y_pred_NB, average = 'macro'))
print('Precision:', precision_score(y_test, y_pred_NB, average = 'macro'))
print('\n Classification report:\n', classification_report(y_test, y_pred_NB))
print('\n Confusion matrix:\n',confusion_matrix(y_test, y_pred_NB))
```

```
Accuracy: 0.861904761904762
F1 score: 0.7761029411764706
Recall: 0.8403361344537815
Precision: 0.7443107465449554

 Classification report:
               precision    recall  f1-score   support

           0       0.96      0.87      0.91       357
           4       0.53      0.81      0.64        63

    accuracy                           0.86       420
   macro avg       0.74      0.84      0.78       420
weighted avg       0.90      0.86      0.87       420


 Confusion matrix:
 [[311  46]
 [ 12  51]]
```

Support Vector Machines Algorithm –



```
: from sklearn.svm import SVC
  svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
  y_pred_svm = svm_model_linear.predict(X_test)
```

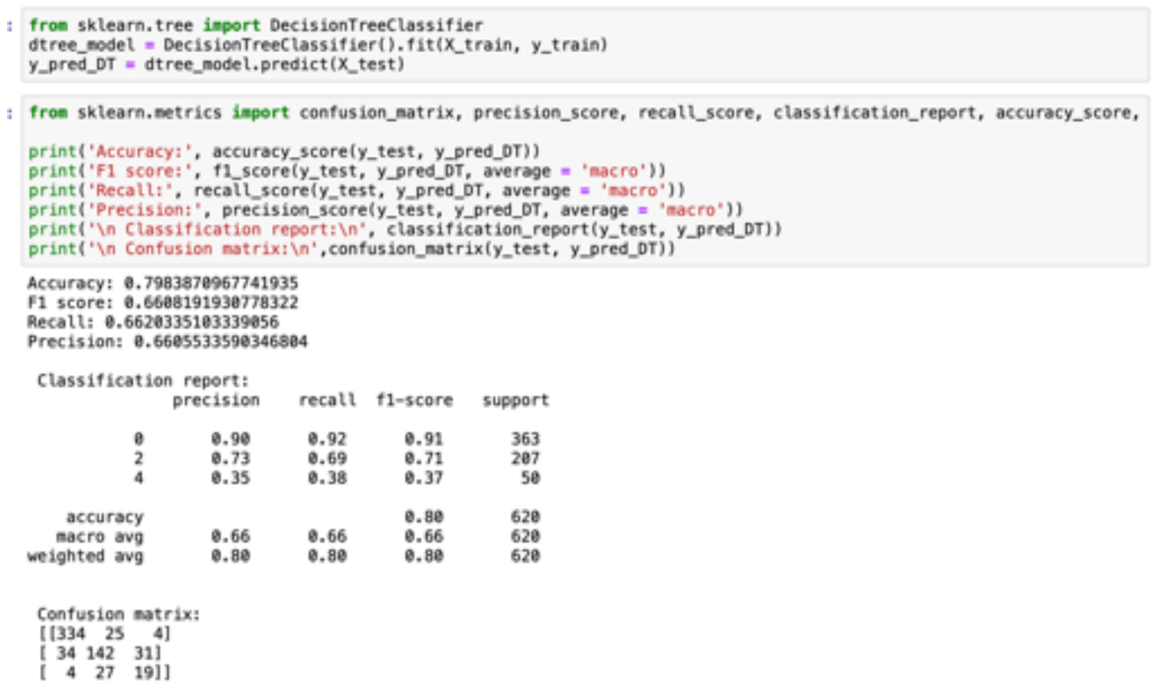
```
: from sklearn.metrics import confusion_matrix, precision_score, recall_score, classification_report, accuracy_score,

  #confusion_matrix(y_train.argmax(axis=1), y_train_pred.argmax(axis=1))
  print('Accuracy:', accuracy_score(y_test, y_pred_svm))
  print('F1 score:', f1_score(y_test, y_pred_svm, average = 'macro'))
  print('Recall:', recall_score(y_test, y_pred_svm, average = 'macro'))
  print('Precision:', precision_score(y_test, y_pred_svm, average = 'macro'))
  print('\n Classification report:\n', classification_report(y_test, y_pred_svm))
  print('\n Confusion matrix:\n',confusion_matrix(y_test, y_pred_svm))
```

```
Accuracy: 0.9523809523809523
F1 score: 0.90538835826275
Recall: 0.9000933706816059
Precision: 0.9109091739348829

 Classification report:
               precision    recall  f1-score   support

           0       0.97      0.97      0.97       357
           4       0.85      0.83      0.84        63

    accuracy                           0.95       420
   macro avg       0.91      0.90      0.91       420
weighted avg       0.95      0.95      0.95       420


 Confusion matrix:
 [[348   9]
 [ 11  52]]
```

Version 1 gave us very satisfactory results with both Decision Tree and SVC giving more 95% accuracy on the test set and Gaussian Naïve Bayes performance at above 85% but since there were only two classes of images having No DR and Proliferative DR so the difference was pretty clear. To test the performance of the models we added one more class of images in version 2.

Version 2 –

Decision Tree Algorithm –

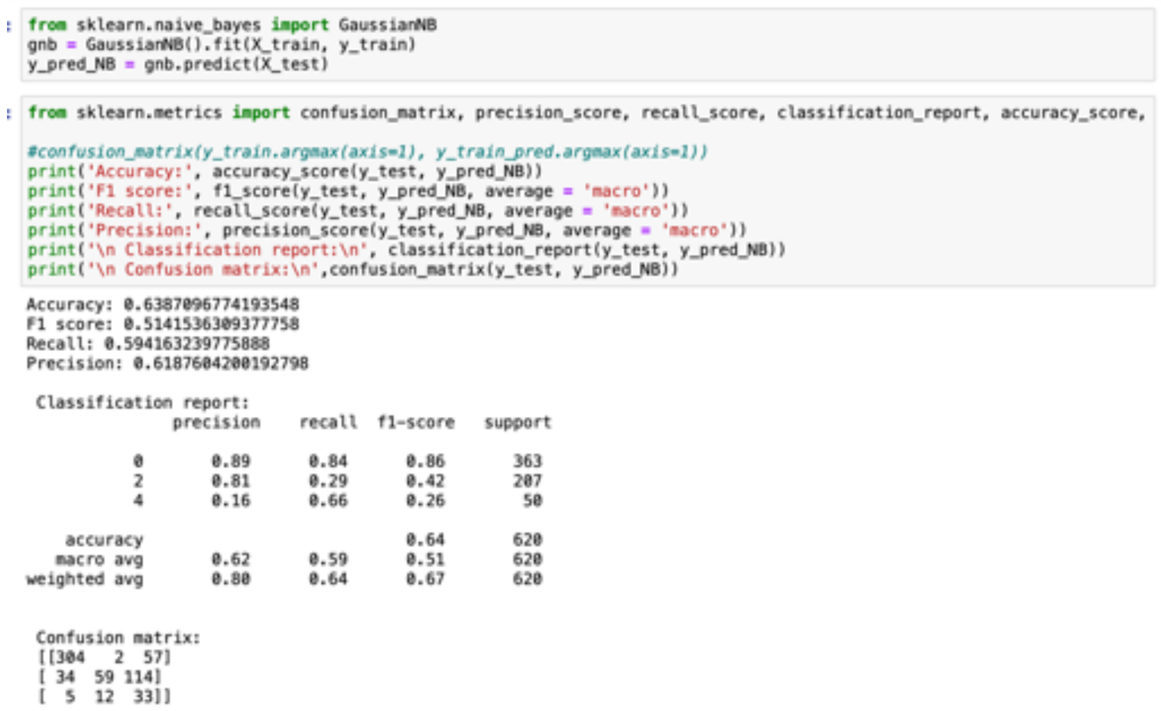


```
: from sklearn.tree import DecisionTreeClassifier
  dtree_model = DecisionTreeClassifier().fit(X_train, y_train)
  y_pred_DT = dtree_model.predict(X_test)
```

```
: from sklearn.metrics import confusion_matrix, precision_score, recall_score, classification_report, accuracy_score,

  print('Accuracy:', accuracy_score(y_test, y_pred_DT))
  print('F1 score:', f1_score(y_test, y_pred_DT, average = 'macro'))
  print('Recall:', recall_score(y_test, y_pred_DT, average = 'macro'))
  print('Precision:', precision_score(y_test, y_pred_DT, average = 'macro'))
  print('\n Classification report:\n', classification_report(y_test, y_pred_DT))
  print('\n Confusion matrix:\n',confusion_matrix(y_test, y_pred_DT))
```

```
Accuracy: 0.7983870967741935
F1 score: 0.6608191930778322
Recall: 0.6620335103339056
Precision: 0.6605533590346804

 Classification report:
               precision    recall  f1-score   support

           0       0.90      0.92      0.91       363
           2       0.73      0.69      0.71       207
           4       0.35      0.38      0.37        50

    accuracy                           0.80       620
   macro avg       0.66      0.66      0.66       620
weighted avg       0.80      0.80      0.80       620


 Confusion matrix:
 [[334  25   4]
 [ 34 142  31]
 [  4  27  19]]
```

Gaussian Naïve Bayes –

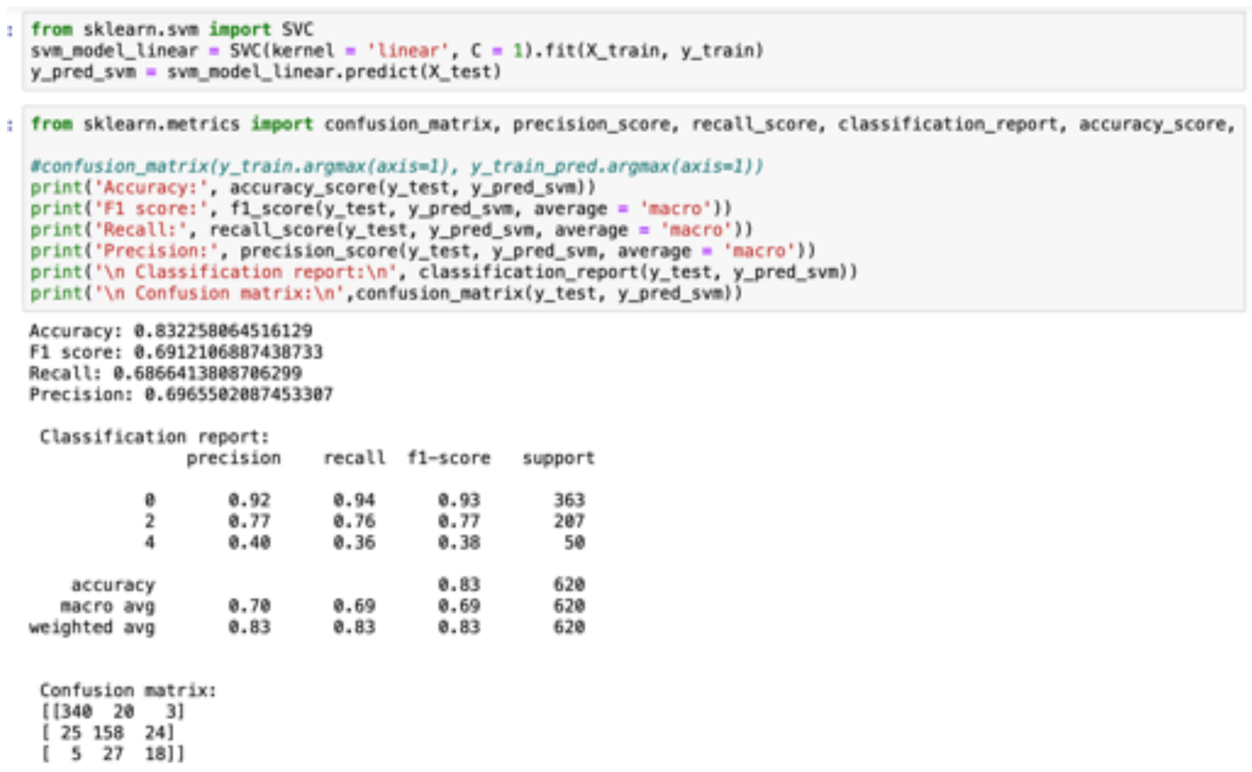


```
: from sklearn.naive_bayes import GaussianNB
  gnb = GaussianNB().fit(X_train, y_train)
  y_pred_NB = gnb.predict(X_test)
```

```
: from sklearn.metrics import confusion_matrix, precision_score, recall_score, classification_report, accuracy_score,

  #confusion_matrix(y_train.argmax(axis=1), y_train_pred.argmax(axis=1))
  print('Accuracy:', accuracy_score(y_test, y_pred_NB))
  print('F1 score:', f1_score(y_test, y_pred_NB, average = 'macro'))
  print('Recall:', recall_score(y_test, y_pred_NB, average = 'macro'))
  print('Precision:', precision_score(y_test, y_pred_NB, average = 'macro'))
  print('\n Classification report:\n', classification_report(y_test, y_pred_NB))
  print('\n Confusion matrix:\n',confusion_matrix(y_test, y_pred_NB))
```

```
Accuracy: 0.6387096774193548
F1 score: 0.5141536309377758
Recall: 0.594163239775888
Precision: 0.6187604200192798

 Classification report:
               precision    recall  f1-score   support

           0       0.89      0.84      0.86       363
           2       0.81      0.29      0.42       207
           4       0.16      0.66      0.26        50

    accuracy                           0.64       620
   macro avg       0.62      0.59      0.51       620
weighted avg       0.80      0.64      0.67       620


 Confusion matrix:
 [[304   2  57]
 [ 34  59 114]
 [  5  12  33]]
```

Support Vector Machines Algorithm –

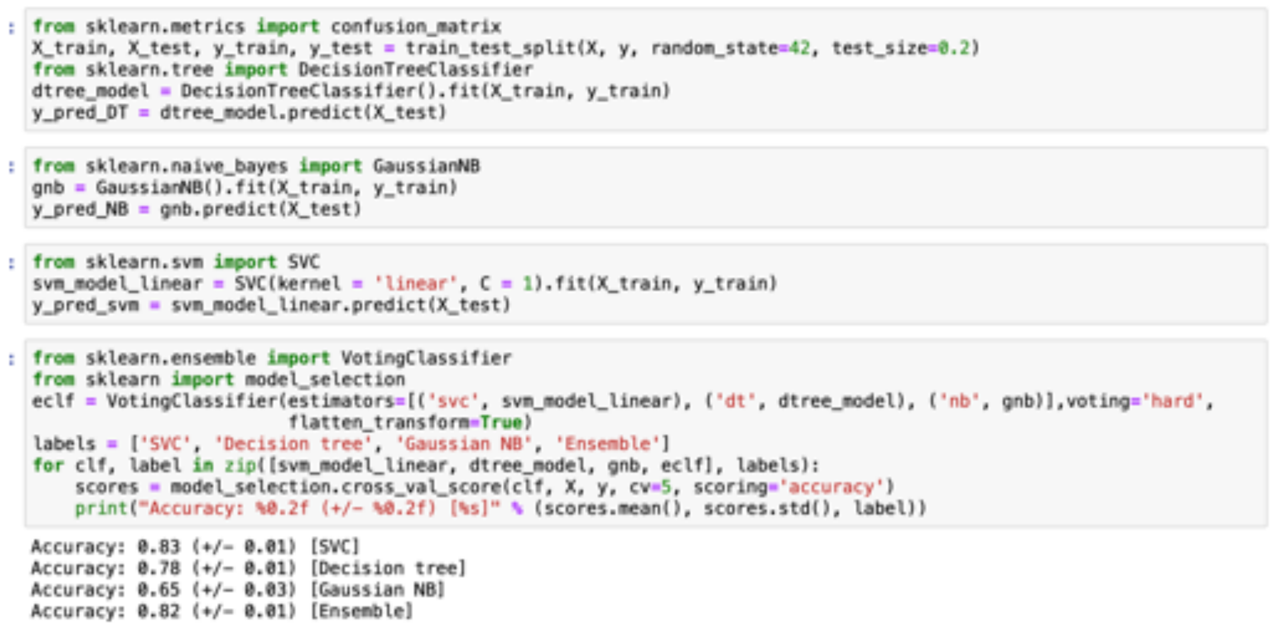


```
: from sklearn.svm import SVC
  svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
  y_pred_svm = svm_model_linear.predict(X_test)

: from sklearn.metrics import confusion_matrix, precision_score, recall_score, classification_report, accuracy_score,

  #confusion_matrix(y_train.argmax(axis=1), y_train_pred.argmax(axis=1))
  print('Accuracy:', accuracy_score(y_test, y_pred_svm))
  print('F1 score:', f1_score(y_test, y_pred_svm, average = 'macro'))
  print('Recall:', recall_score(y_test, y_pred_svm, average = 'macro'))
  print('Precision:', precision_score(y_test, y_pred_svm, average = 'macro'))
  print('\n Classification report:\n', classification_report(y_test, y_pred_svm))
  print('\n Confusion matrix:\n',confusion_matrix(y_test, y_pred_svm))
```

```
Accuracy: 0.832258064516129
F1 score: 0.6912106887438733
Recall: 0.6866413808706299
Precision: 0.6965502087453307

 Classification report:
               precision    recall  f1-score   support

           0       0.92      0.94      0.93       363
           2       0.77      0.76      0.77       207
           4       0.40      0.36      0.38        50

    accuracy                           0.83       620
   macro avg       0.70      0.69      0.69       620
weighted avg       0.83      0.83      0.83       620


 Confusion matrix:
 [[340  20   3]
 [ 25 158  24]
 [  5  27  18]]
```

Adding an additional class of images has definitely added complexity to all the models and has reduced the accuracy for Decision Tree, SVC to 79% and 83% resp. and 63% for Gaussian Naïve Bayes.

Version 3 –



```
: from sklearn.metrics import confusion_matrix
  X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2)
  from sklearn.tree import DecisionTreeClassifier
  dtree_model = DecisionTreeClassifier().fit(X_train, y_train)
  y_pred_DT = dtree_model.predict(X_test)
```

```
: from sklearn.naive_bayes import GaussianNB
  gnb = GaussianNB().fit(X_train, y_train)
  y_pred_NB = gnb.predict(X_test)
```

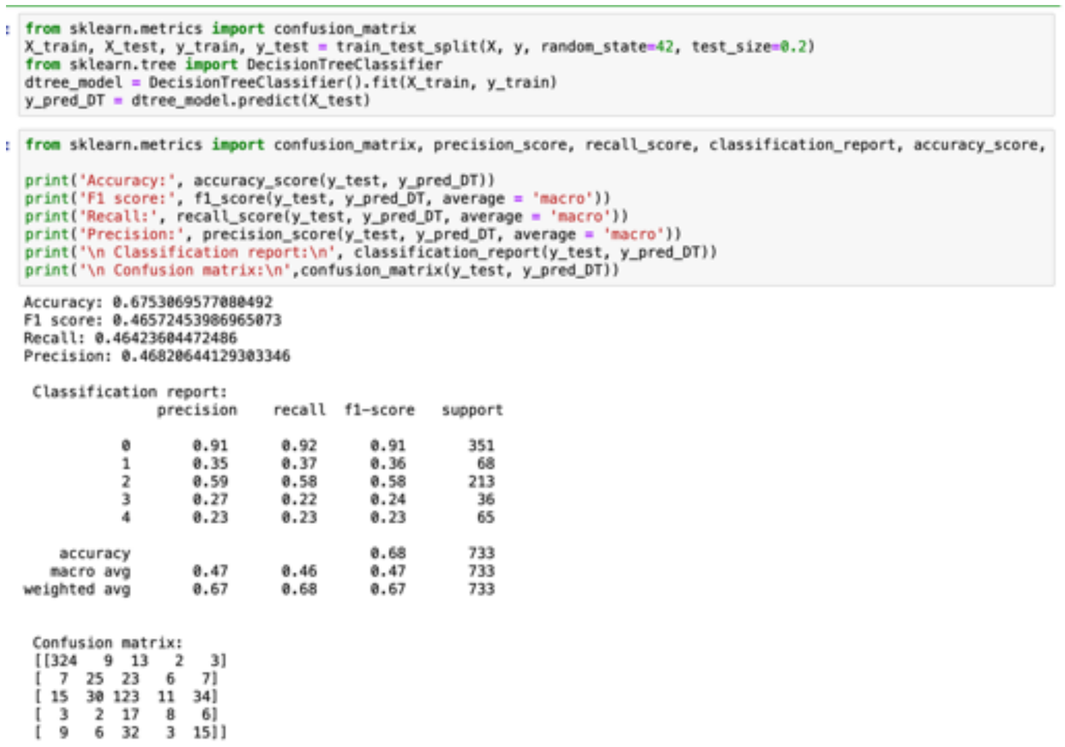
```
: from sklearn.svm import SVC
  svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
  y_pred_svm = svm_model_linear.predict(X_test)
```

```
: from sklearn.ensemble import VotingClassifier
  from sklearn import model_selection
  eclf = VotingClassifier(estimators=[('svc', svm_model_linear), ('dt', dtree_model), ('nb', gnb)],voting='hard',
                          flatten_transform=True)
  labels = ['SVC', 'Decision tree', 'Gaussian NB', 'Ensemble']
  for clf, label in zip([svm_model_linear, dtree_model, gnb, eclf], labels):
      scores = model_selection.cross_val_score(clf, X, y, cv=5, scoring='accuracy')
      print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

  Accuracy: 0.83 (+/- 0.01) [SVC]
  Accuracy: 0.78 (+/- 0.01) [Decision tree]
  Accuracy: 0.65 (+/- 0.03) [Gaussian NB]
  Accuracy: 0.82 (+/- 0.01) [Ensemble]
```

In this version we have applied Ensemble of Decision Tree, SVC and Gaussian Naïve Bayes to the three class of images, but it did not improve the performance of the model and the accuracy for ensemble was less than standalone SVC model.

Version 4 –

Decision Tree Algorithm –

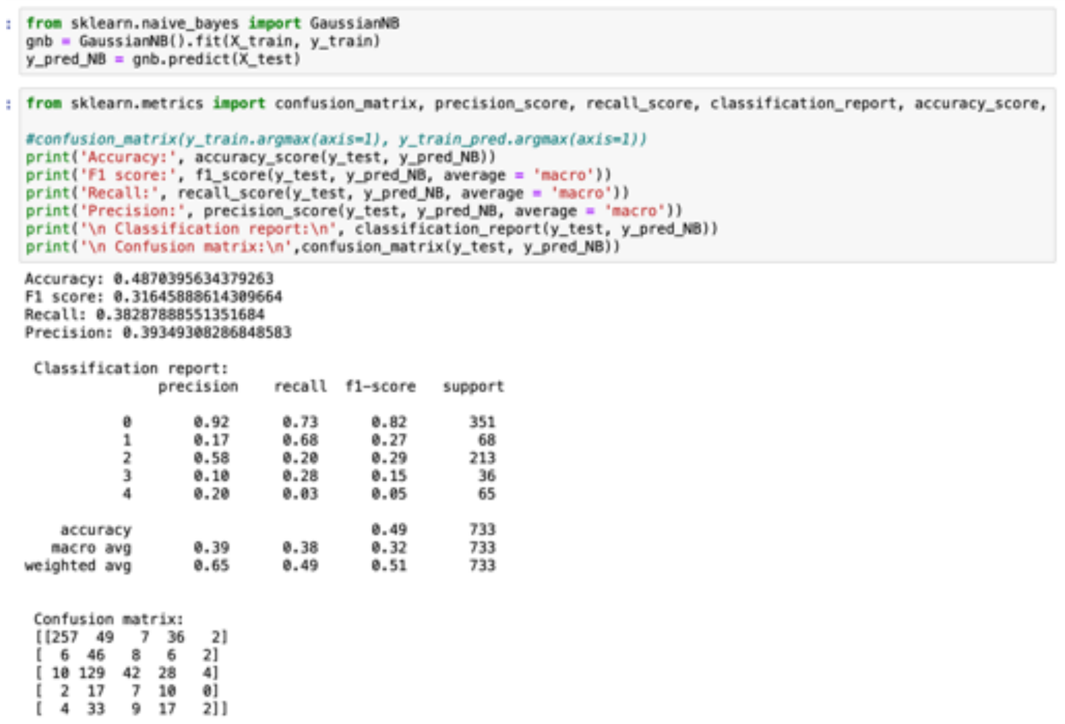


```
from sklearn.metrics import confusion_matrix
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2)
from sklearn.tree import DecisionTreeClassifier
dtree_model = DecisionTreeClassifier().fit(X_train, y_train)
y_pred_DT = dtree_model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, classification_report, accuracy_score,

print('Accuracy:', accuracy_score(y_test, y_pred_DT))
print('F1 score:', f1_score(y_test, y_pred_DT, average = 'macro'))
print('Recall:', recall_score(y_test, y_pred_DT, average = 'macro'))
print('Precision:', precision_score(y_test, y_pred_DT, average = 'macro'))
print('\n Classification report:\n', classification_report(y_test, y_pred_DT))
print('\n Confusion matrix:\n',confusion_matrix(y_test, y_pred_DT))
```

```
Accuracy: 0.6753069577080492
F1 score: 0.46572453986965073
Recall: 0.464236044472486
Precision: 0.46820644129303346

 Classification report:
               precision    recall  f1-score   support

           0       0.91      0.92      0.91       351
           1       0.35      0.37      0.36        68
           2       0.59      0.58      0.58       213
           3       0.27      0.22      0.24        36
           4       0.23      0.23      0.23        65

    accuracy                           0.68       733
   macro avg       0.47      0.46      0.47       733
weighted avg       0.67      0.68      0.67       733


 Confusion matrix:
 [[324   9  13   2   3]
 [  7  25  23   6   7]
 [ 15  30 123  11  34]
 [  3   2  17   8   6]
 [  9   6  32   3  15]]
```

Gaussian Naïve Bayes –

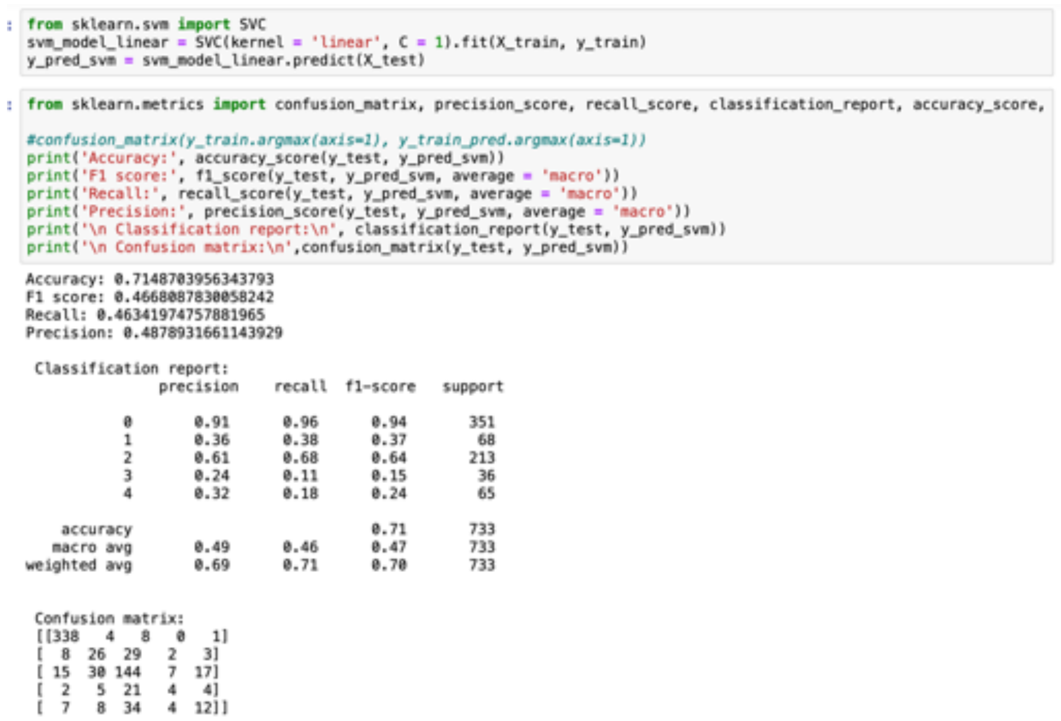


```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB().fit(X_train, y_train)
y_pred_NB = gnb.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, classification_report, accuracy_score,

#confusion_matrix(y_train.argmax(axis=1), y_train_pred.argmax(axis=1))
print('Accuracy:', accuracy_score(y_test, y_pred_NB))
print('F1 score:', f1_score(y_test, y_pred_NB, average = 'macro'))
print('Recall:', recall_score(y_test, y_pred_NB, average = 'macro'))
print('Precision:', precision_score(y_test, y_pred_NB, average = 'macro'))
print('\n Classification report:\n', classification_report(y_test, y_pred_NB))
print('\n Confusion matrix:\n',confusion_matrix(y_test, y_pred_NB))
```

```
Accuracy: 0.4870395634379263
F1 score: 0.31645888614309664
Recall: 0.38287888551351684
Precision: 0.39349308286848583

 Classification report:
               precision    recall  f1-score   support

           0       0.92      0.73      0.82       351
           1       0.17      0.68      0.27        68
           2       0.58      0.20      0.29       213
           3       0.10      0.28      0.15        36
           4       0.20      0.03      0.05        65

    accuracy                           0.49       733
   macro avg       0.39      0.38      0.32       733
weighted avg       0.65      0.49      0.51       733


 Confusion matrix:
 [[257  49   7  36   2]
 [  6  46   8   6   2]
 [ 10 129  42  28   4]
 [  2  17   7  10   0]
 [  4  33   9  17   2]]
```

Support Vector Machines Algorithm –



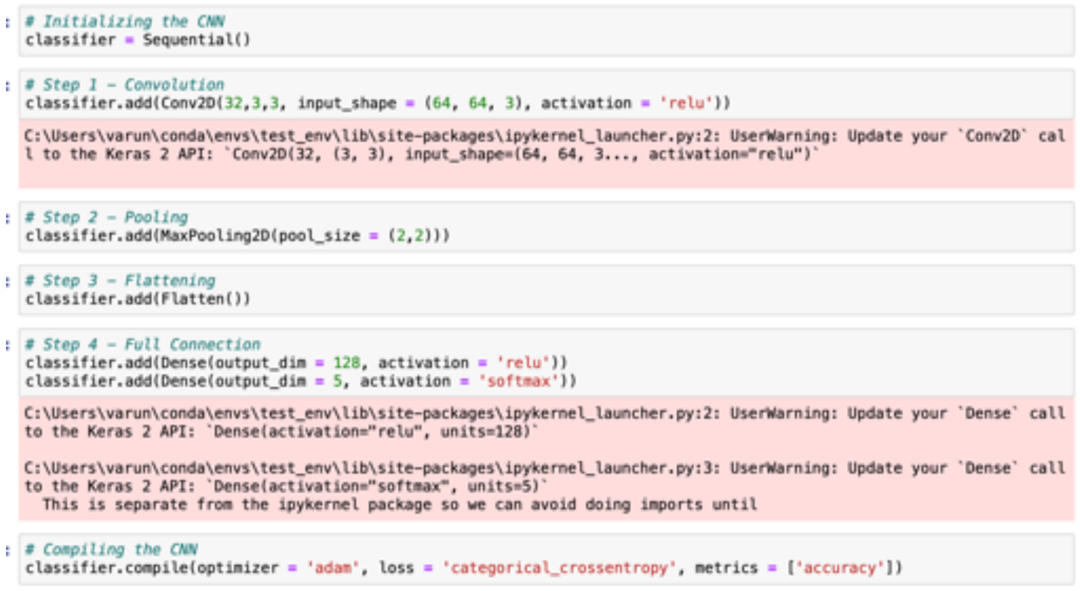
```
from sklearn.svm import SVC
svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
y_pred_svm = svm_model_linear.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, classification_report, accuracy_score,

#confusion_matrix(y_train.argmax(axis=1), y_train_pred.argmax(axis=1))
print('Accuracy:', accuracy_score(y_test, y_pred_svm))
print('F1 score:', f1_score(y_test, y_pred_svm, average = 'macro'))
print('Recall:', recall_score(y_test, y_pred_svm, average = 'macro'))
print('Precision:', precision_score(y_test, y_pred_svm, average = 'macro'))
print('\n Classification report:\n', classification_report(y_test, y_pred_svm))
print('\n Confusion matrix:\n',confusion_matrix(y_test, y_pred_svm))
```

```
Accuracy: 0.7148703956343793
F1 score: 0.4668087830058242
Recall: 0.46341974757881965
Precision: 0.4878931661143929

 Classification report:
               precision    recall  f1-score   support

           0       0.91      0.96      0.94       351
           1       0.36      0.38      0.37        68
           2       0.61      0.68      0.64       213
           3       0.24      0.11      0.15        36
           4       0.32      0.18      0.24        65

    accuracy                           0.71       733
   macro avg       0.49      0.46      0.47       733
weighted avg       0.69      0.71      0.70       733


 Confusion matrix:
 [[338   4   8   0   1]
 [  8  26  29   2   3]
 [ 15  30 144   7  17]
 [  2   5  21   4   4]
 [  7   8  34   4  12]]
```

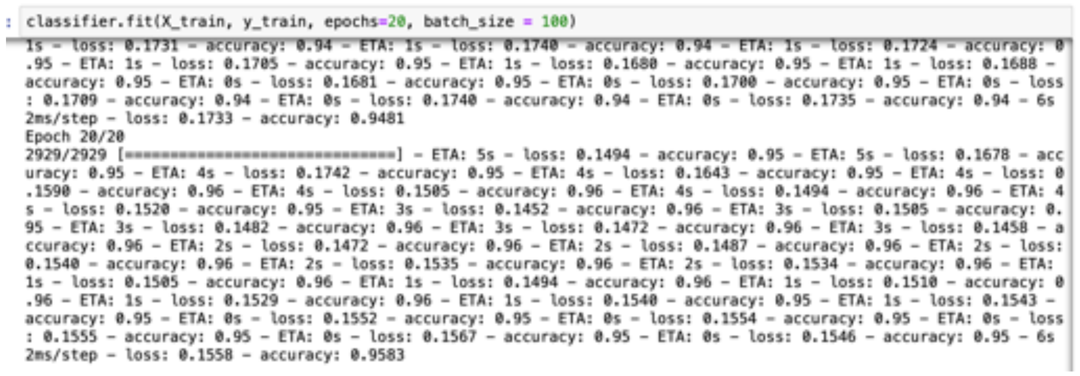
In this version we have further increased complexity by adding all the classes available and applied Decision Tree, SVC and Gaussian Naïve Bayes, the accuracy for all models have reduced even more with SVC at 71%, Decision Tree at 67% and Gaussian NB at 48%.

Version 5 (Deep Learning) –



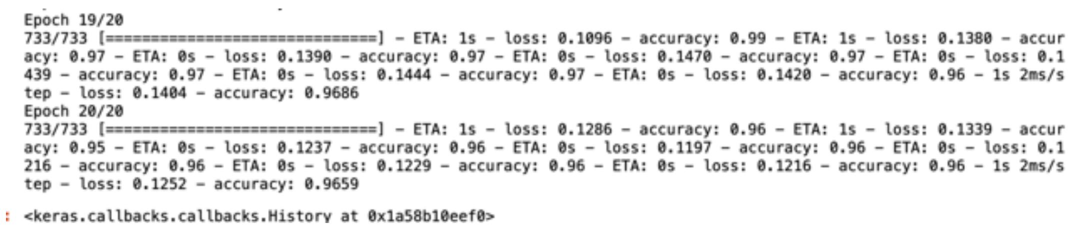
```
: # Initializing the CNN
  classifier = Sequential()

: # Step 1 - Convolution
  classifier.add(Conv2D(32,3,3, input_shape = (64, 64, 3), activation = 'relu'))

  C:\Users\varun\conda\envs\test_env\lib\site-packages\ipykernel_launcher.py:2: UserWarning: Update your `Conv2D` cal
  l to the Keras 2 API: `Conv2D(32, (3, 3), input_shape=(64, 64, 3..., activation="relu")`

: # Step 2 - Pooling
  classifier.add(MaxPooling2D(pool_size = (2,2)))

: # Step 3 - Flattening
  classifier.add(Flatten())

: # Step 4 - Full Connection
  classifier.add(Dense(output_dim = 128, activation = 'relu'))
  classifier.add(Dense(output_dim = 5, activation = 'softmax'))

  C:\Users\varun\conda\envs\test_env\lib\site-packages\ipykernel_launcher.py:2: UserWarning: Update your `Dense` call
  to the Keras 2 API: `Dense(activation="relu", units=128)`

  C:\Users\varun\conda\envs\test_env\lib\site-packages\ipykernel_launcher.py:3: UserWarning: Update your `Dense` call
  to the Keras 2 API: `Dense(activation="softmax", units=5)`
    This is separate from the ipykernel package so we can avoid doing imports until

: # Compiling the CNN
  classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

Train set results –



```
: classifier.fit(X_train, y_train, epochs=20, batch_size = 100)
  1s - loss: 0.1731 - accuracy: 0.94 - ETA: 1s - loss: 0.1740 - accuracy: 0.94 - ETA: 1s - loss: 0.1724 - accuracy: 0
  .95 - ETA: 1s - loss: 0.1705 - accuracy: 0.95 - ETA: 1s - loss: 0.1680 - accuracy: 0.95 - ETA: 1s - loss: 0.1688 -
  accuracy: 0.95 - ETA: 0s - loss: 0.1681 - accuracy: 0.95 - ETA: 0s - loss: 0.1700 - accuracy: 0.95 - ETA: 0s - loss
  : 0.1709 - accuracy: 0.94 - ETA: 0s - loss: 0.1740 - accuracy: 0.94 - ETA: 0s - loss: 0.1735 - accuracy: 0.94 - 6s
  2ms/step - loss: 0.1733 - accuracy: 0.9481
  Epoch 20/20
  2929/2929 [==============================] - ETA: 5s - loss: 0.1494 - accuracy: 0.95 - ETA: 5s - loss: 0.1678 - acc
  uracy: 0.95 - ETA: 4s - loss: 0.1742 - accuracy: 0.95 - ETA: 4s - loss: 0.1643 - accuracy: 0.95 - ETA: 4s - loss: 0
  .1590 - accuracy: 0.96 - ETA: 4s - loss: 0.1505 - accuracy: 0.96 - ETA: 4s - loss: 0.1494 - accuracy: 0.96 - ETA: 4
  s - loss: 0.1520 - accuracy: 0.95 - ETA: 3s - loss: 0.1452 - accuracy: 0.96 - ETA: 3s - loss: 0.1505 - accuracy: 0.
  95 - ETA: 3s - loss: 0.1482 - accuracy: 0.96 - ETA: 3s - loss: 0.1472 - accuracy: 0.96 - ETA: 3s - loss: 0.1458 - a
  ccuracy: 0.96 - ETA: 2s - loss: 0.1472 - accuracy: 0.96 - ETA: 2s - loss: 0.1487 - accuracy: 0.96 - ETA: 2s - loss:
  0.1540 - accuracy: 0.96 - ETA: 2s - loss: 0.1535 - accuracy: 0.96 - ETA: 2s - loss: 0.1534 - accuracy: 0.96 - ETA:
  1s - loss: 0.1505 - accuracy: 0.96 - ETA: 1s - loss: 0.1494 - accuracy: 0.96 - ETA: 1s - loss: 0.1510 - accuracy: 0
  .96 - ETA: 1s - loss: 0.1529 - accuracy: 0.96 - ETA: 1s - loss: 0.1540 - accuracy: 0.95 - ETA: 1s - loss: 0.1543 -
  accuracy: 0.95 - ETA: 0s - loss: 0.1552 - accuracy: 0.95 - ETA: 0s - loss: 0.1554 - accuracy: 0.95 - ETA: 0s - loss
  : 0.1555 - accuracy: 0.95 - ETA: 0s - loss: 0.1567 - accuracy: 0.95 - ETA: 0s - loss: 0.1546 - accuracy: 0.95 - 6s
  2ms/step - loss: 0.1558 - accuracy: 0.9583
```

Test set results –



```
  Epoch 19/20
  733/733 [==============================] - ETA: 1s - loss: 0.1096 - accuracy: 0.99 - ETA: 1s - loss: 0.1380 - accur
  acy: 0.97 - ETA: 0s - loss: 0.1390 - accuracy: 0.97 - ETA: 0s - loss: 0.1470 - accuracy: 0.97 - ETA: 0s - loss: 0.1
  439 - accuracy: 0.97 - ETA: 0s - loss: 0.1444 - accuracy: 0.97 - ETA: 0s - loss: 0.1420 - accuracy: 0.96 - 1s 2ms/s
  tep - loss: 0.1404 - accuracy: 0.9686
  Epoch 20/20
  733/733 [==============================] - ETA: 1s - loss: 0.1286 - accuracy: 0.96 - ETA: 1s - loss: 0.1339 - accur
  acy: 0.95 - ETA: 0s - loss: 0.1237 - accuracy: 0.96 - ETA: 0s - loss: 0.1197 - accuracy: 0.96 - ETA: 0s - loss: 0.1
  216 - accuracy: 0.96 - ETA: 0s - loss: 0.1229 - accuracy: 0.96 - ETA: 0s - loss: 0.1216 - accuracy: 0.96 - 1s 2ms/s
  tep - loss: 0.1252 - accuracy: 0.9659

: <keras.callbacks.callbacks.History at 0x1a58b10eef0>
```

In this version of applying Convolutional Neural Networks to all the classes of images, the results were fortunately very impressive as it gave 95% accuracy on train set and 96% on test set.

Future Work –

1. Going forward we can apply an Ensemble of Deep Learning models to all classes to further improve the accuracy and take it near to 100%.
2. Secondly, we also can apply an Ensemble of Machine Learning models to all the classes and check whether it has any effect on the performance.
3. Adding more noise in the form of retinal images after labeling them with the help of a trained ophthalmologist.

Implications –

Advances in computing and the availability of large data sets of retinal images have spurred the development of AI systems for detecting not only diabetic retinopathy, which is relatively easy to spot, but also other common eye diseases such as age-related macular degeneration (AMD) and glaucoma. These AI systems could improve the speed and accuracy of large-scale screening programmes, as well as improve access to eye examinations in underserved areas by enabling their provision at medical centers that could not otherwise offer eye care.

References -

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1705856/

https://www.nature.com/articles/s41746-019-0172-3

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5961805/

https://towardsdatascience.com/ensembling-convnets-using-keras-237d429157eb

https://www.kaggle.com/mgiraygokirmak/keras-cnn-multi-model-ensemble-with-voting

https://www.nature.com/articles/d41586-019-01111-y

https://www.healthcareitnews.com/news/google-verily-using-ai-screen-diabetic-retinopathy-india

https://www.kaggle.com/c/aptos2019-blindness-detection

www.datacamp.com