Final project - MLCV

Ajit Babu Vivekananth[1]
[1] *Masters Student*
Charles University,
Prague, Czech Republic.
[1]ajitbabu96@gmail.com

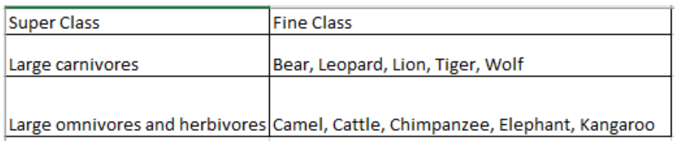
Abstract—

Task 1 -> Binary classification into two chosen superclasses.

Task 2 -> Multiclass classification into the fine classes of one superclass.

Dataset--- The CIFAR-100 dataset has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 super classes. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).

The superclasses Large carnivores & Large omnivores and herbivores have been picked for the project. And fine classes in Large carnivores have been picked for Multiclass classification task.



Super Class	Fine Class
Large carnivores	Bear, Leopard, Lion, Tiger, Wolf
Large omnivores and herbivores	Camel, Cattle, Chimpanzee, Elephant, Kangaroo

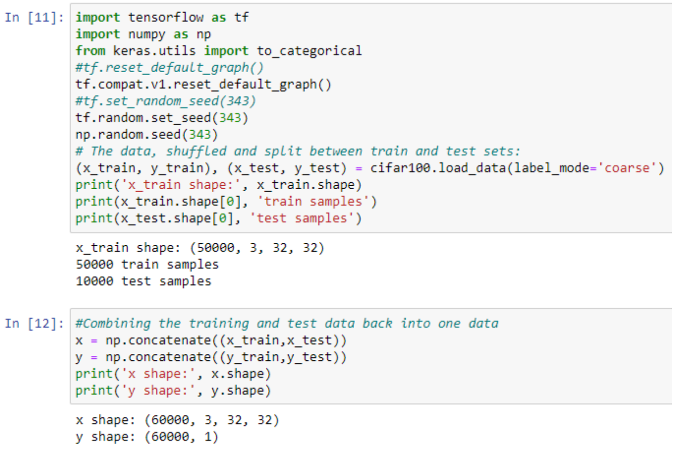
Environment & language used: Jupyter Notebook. Python language. Python Libraries: Keras, Sklearn, Numpy, Matplotlib, Pandas, Scipy, Tensorflow

Introduction

Cifar 100 dataset consists of train, test and meta data. I loaded them into notebook and combined the images(X), coarse labels(Y) (0-19) data into data frame and filtered this data frame on target variable (Large carnivores = 8 & Large omnivores and herbivores = 11). I used the meta data file in cifar 100 which consists of a serialized dictionary object to search for index of Large carnivores, Large omnivores and herbivores in the coarse label list and validated that 8 corresponds to Large carnivores and 11 corresponds to Large omnivores and herbivores through code and I have crosschecked it by plotting few images from the filtered dataset.

I. Data Initialization

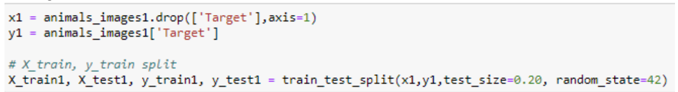
In order to split the dataset into two sets, that is one set for training and the other set for testing, I have splitted the dataset in an 80–20 ratio such that 80% data is trained. To reshape our dataset inputs (X_train1 and X_test1) to the shape that our model expects when I train the model, I used astype to cast a pandas object into float. And then I used 'one-hot-encode' on our target variable using get_dummies to convert categorical variable into dummy variables and to_categorical to convert a class vector (integers) to binary class matrix. This means that a column will be created for each output category and a binary variable is inputted for each category. Similarly for Multiclass classification, I have divided the training set and test set into 500 & 100 images per class respectively. And the training data has been combined and trained. Logistic Regression, Decision tree, Random forest and Convolutional Neural Network were used.



```
In [11]: import tensorflow as tf
         import numpy as np
         from keras.utils import to_categorical
         #tf.reset_default_graph()
         tf.compat.v1.reset_default_graph()
         #tf.set_random_seed(343)
         tf.random.set_seed(343)
         np.random.seed(343)
         # The data, shuffled and split between train and test sets:
         (x_train, y_train), (x_test, y_test) = cifar100.load_data(label_mode='coarse')
         print('x_train shape:', x_train.shape)
         print(x_train.shape[0], 'train samples')
         print(x_test.shape[0], 'test samples')

         x_train shape: (50000, 3, 32, 32)
         50000 train samples
         10000 test samples

In [12]: #Combining the training and test data back into one data
         x = np.concatenate((x_train,x_test))
         y = np.concatenate((y_train,y_test))
         print('x shape:', x.shape)
         print('y shape:', y.shape)

         x shape: (60000, 3, 32, 32)
         y shape: (60000, 1)
```

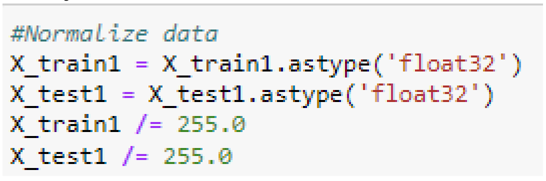
II. Binary Classification

Binary classification is task of classifying the elements of a set into two groups on the basis of a classification rule. For the task 1, I have used different classifications like Logistic Regression, Decision Tree Classifier, Random Forest Classifier & CNN. Cross-validation has been done with the keyword sklearn.model_selection.train_test_split() with x1, y1, test_size=0.20, random_state=42 as parameters, where x1&y1 are arrays.



```
x1 = animals_images1.drop(['Target'],axis=1)
y1 = animals_images1['Target']

# X_train, y_train split
X_train1, X_test1, y_train1, y_test1 = train_test_split(x1,y1,test_size=0.20, random_state=42)
```

The obtained data is then normalizeded into float32 and then divided by 255.



```
#Normalize data
X_train1 = X_train1.astype('float32')
X_test1 = X_test1.astype('float32')
X_train1 /= 255.0
X_test1 /= 255.0
```

The obtained parameters X_train1, y_train1, X_test1 & y_test1 are detailed below.

1). X_train - This includes all independent variables, these will be used to train the model, also as we have specified the test_size = 0.2, this means 80% of observations from the complete data will be used to train/fit the model and rest 20% will be used to test the model.

2). X_test - This is remaining 20% portion of the independent variables from the data which will not be used in the training phase and will be used to make predictions to test the accuracy of the model.

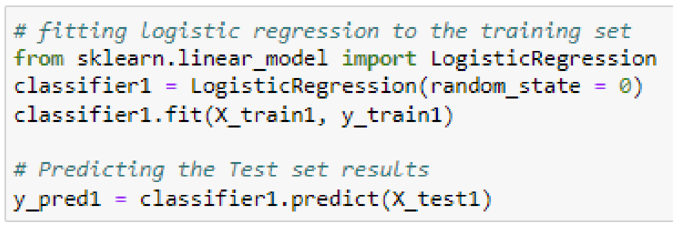
3). y_train - This is the dependent variable which needs to be predicted by this model, this includes category labels against your independent variables, we need to specify our dependent variable while training/fitting the model.

4). y_test - This data has category labels for the test data, these labels will be used to test the accuracy between actual and predicted categories.

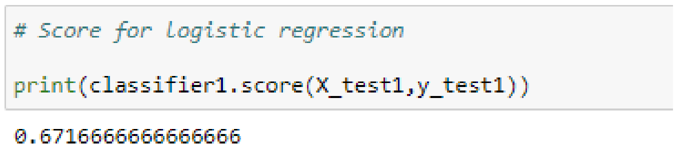
i. Logistic Regression

Logistic Regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes. I imported LogisticRegression from the library sklearn.linear_model to classify the training data. And with predict() function, I have predicted the test dataset and have stored in the variable y_pred. With classification_report() function, I have calculated the accuracy, macro-averaged precision and recall values.

Accuracy obtained: 67.1666%

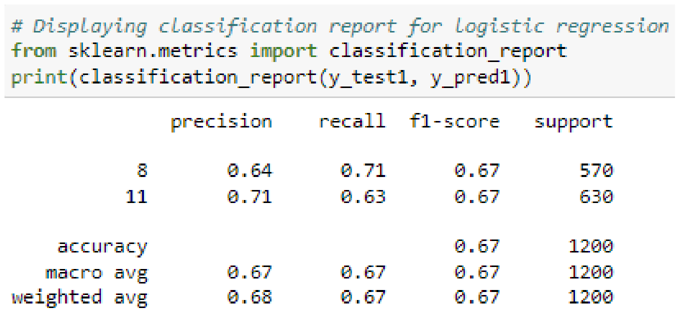


```
# fitting logistic regression to the training set
from sklearn.linear_model import LogisticRegression
classifier1 = LogisticRegression(random_state = 0)
classifier1.fit(X_train1, y_train1)

# Predicting the Test set results
y_pred1 = classifier1.predict(X_test1)
```



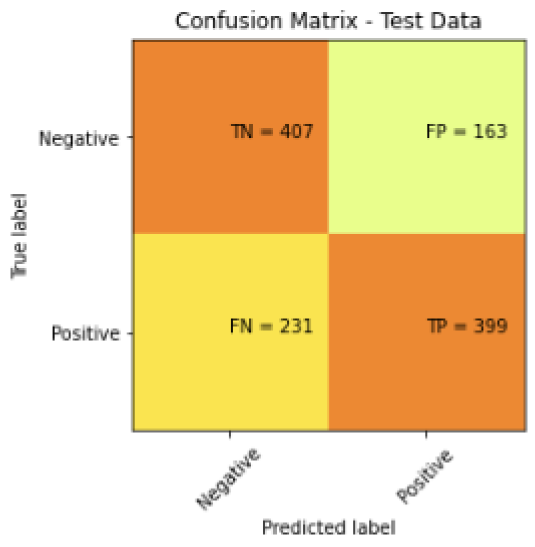
```
# Score for logistic regression

print(classifier1.score(X_test1,y_test1))
```

```
0.6716666666666666
```

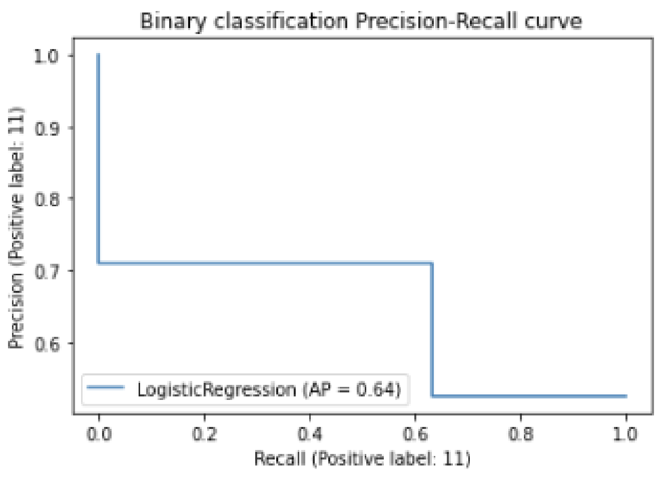


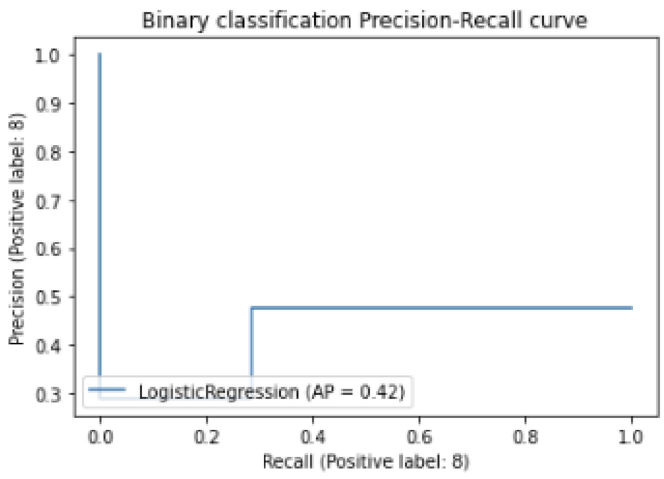
```
# Displaying classification report for logistic regression
from sklearn.metrics import classification_report
print(classification_report(y_test1, y_pred1))
```

```
              precision    recall  f1-score   support

           8       0.64      0.71      0.67       570
          11       0.71      0.63      0.67       630

    accuracy                           0.67      1200
   macro avg       0.67      0.67      0.67      1200
weighted avg       0.68      0.67      0.67      1200
```



Accuracy = (TN+TP)/(TN+FP+FN+TP)
= (407+399)/(407+163+231+399)=0.67166





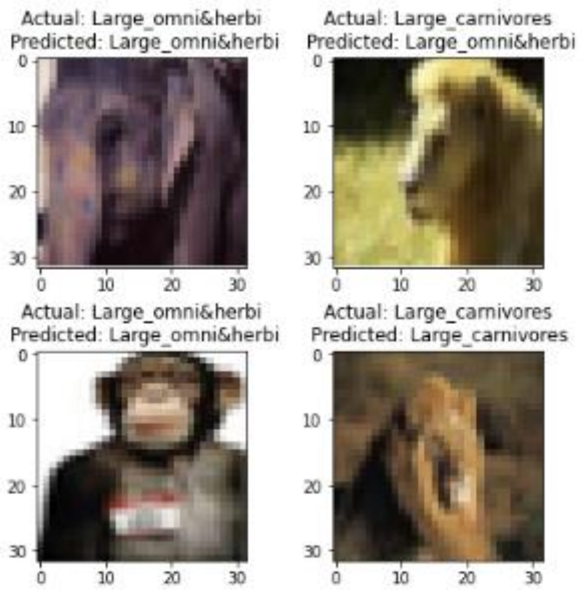

Actual: Large_omni&herbi
Predicted: Large_omni&herbi
Actual: Large_carnivores
Predicted: Large_omni&herbi
Actual: Large_omni&herbi
Predicted: Large_omni&herbi
Actual: Large_carnivores
Predicted: Large_carnivores

Fig: Output predicted from Logistic Regression

Fig: Output predicted from Naive Bayes classifier

ii. Decision Tree Classifier

Decision Tree Classifier is a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. I took Decision tress, because that I have picked Random Forest and the classification algorithm is architected with many decision trees and I wanted to test how a single decision tree performs. The construction of the model is similar to the Logistic Regression stated above, except that DecisionTreeClassifier from the library sklearn.linear_model is used for training data.

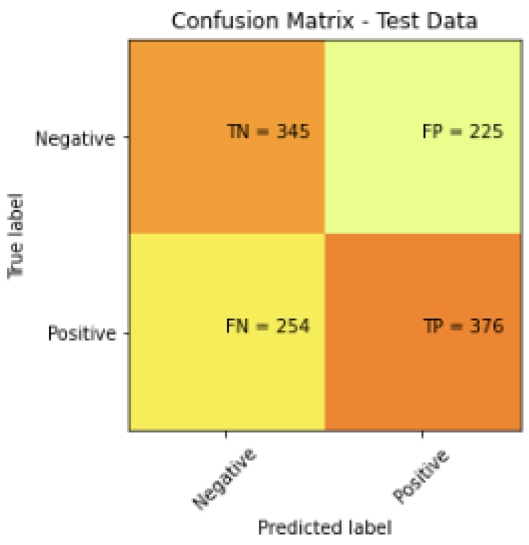
Accuracy obtained: 60.08%

```
# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train1, y_train1)

# Predicting the Test set results
y_pred1 = classifier.predict(X_test1)
```

```
# Score for decision tree model

print(classifier.score(X_test1,y_test1))
```

```
0.6008333333333333
```

Accuracy = (TN+TP)/(TN+FP+FN+TP)
= (345+376)/(345+376+225+254) = 0.6008

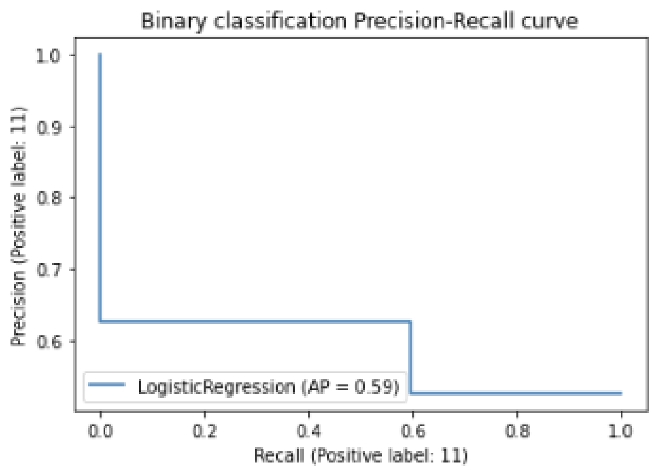


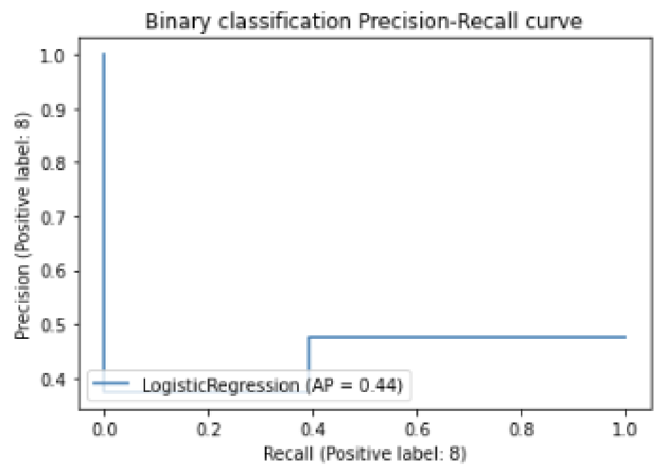
```
# Displaying classification report for Decision Tree model
from sklearn.metrics import classification_report
print(classification_report(y_test1, y_pred1))
```

```
              precision    recall  f1-score   support

           8       0.58      0.61      0.59       570
          11       0.63      0.60      0.61       630

    accuracy                           0.60      1200
   macro avg       0.60      0.60      0.60      1200
weighted avg       0.60      0.60      0.60      1200
```

```
from sklearn.metrics import precision_score, recall_score
print('Precision Large_Carnivores: %.3f' % precision_score(y_test1, y_pred1, pos_label=8))
print('Precision Large_omnivores_and_herbivores: %.3f' % precision_score(y_test1, y_pred1, pos_label=11))
print('Recall Large_Carnivores: %.3f' % recall_score(y_test1, y_pred1, pos_label=8))
print('Recall Large_omnivores_and_herbivores: %.3f' % recall_score(y_test1, y_pred1, pos_label=11))
```

```
Precision Large_Carnivores: 0.576
Precision Large_omnivores_and_herbivores: 0.626
Recall Large_Carnivores: 0.605
Recall Large_omnivores_and_herbivores: 0.597
```





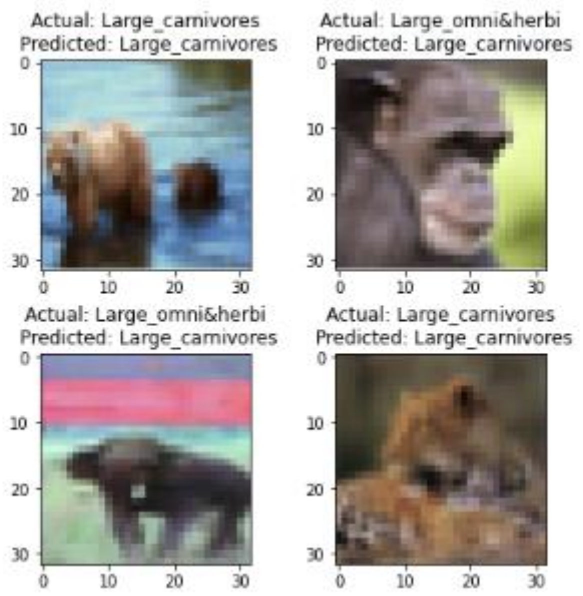
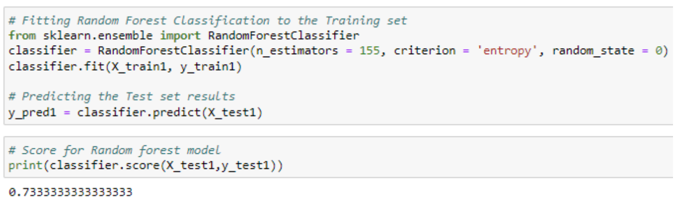

Actual: Large_carnivores
Predicted: Large_carnivores
Actual: Large_omni&herbi
Predicted: Large_carnivores
Actual: Large_omni&herbi
Predicted: Large_carnivores
Actual: Large_carnivores
Predicted: Large_carnivores

Fig: Output predicted from Decision Tree Classifier

iii. Random Forest Classifier

Random Forest is a robust machine learning algorithm that can be used for a variety of tasks including regression and classification. It is an ensemble method, meaning that a random forest model is made up of a large number of small decision trees, called estimators, which each produce their own predictions. The random forest model combines the predictions of the estimators to produce a more accurate prediction. The construction of the model is similar to the Logistic Regression stated above, except that RandomForestClassifier from the library sklearn.linear_model is used for training data.

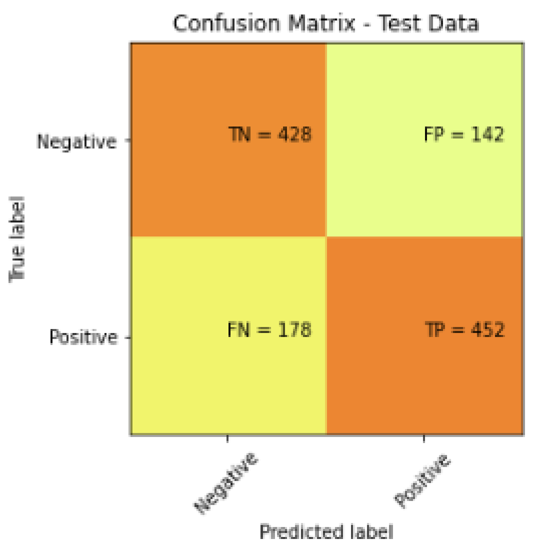
Accuracy obtained: 73.33%

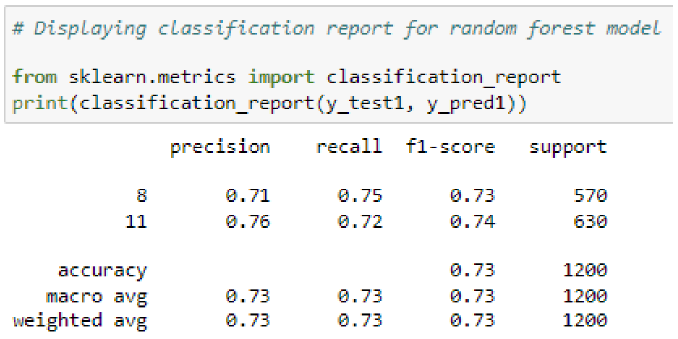


```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 155, criterion = 'entropy', random_state = 0)
classifier.fit(X_train1, y_train1)

# Predicting the Test set results
y_pred1 = classifier.predict(X_test1)
```

```
# Score for Random forest model
print(classifier.score(X_test1,y_test1))
```

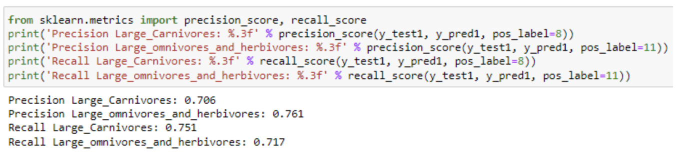
```
0.7333333333333333
```





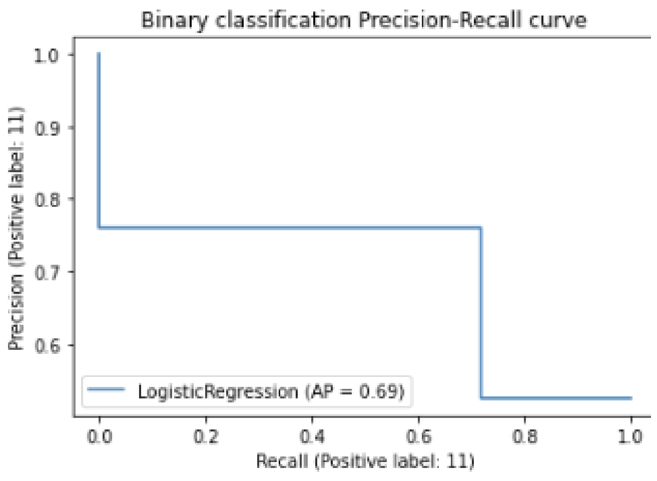
```
# Displaying classification report for random forest model

from sklearn.metrics import classification_report
print(classification_report(y_test1, y_pred1))
```

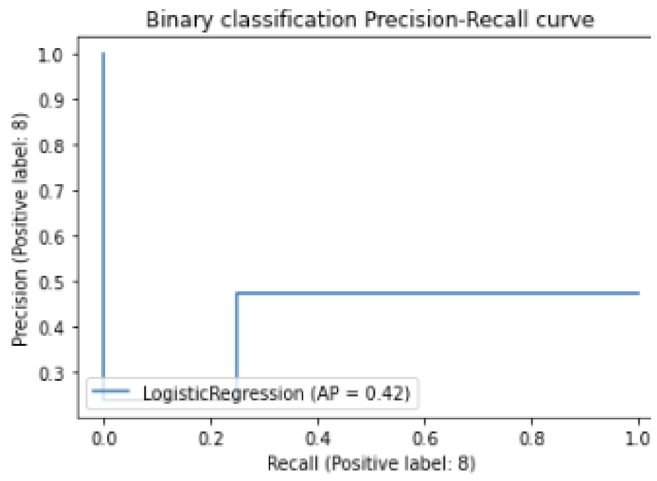
```
              precision    recall  f1-score   support

           8       0.71      0.75      0.73       570
          11       0.76      0.72      0.74       630

    accuracy                           0.73      1200
   macro avg       0.73      0.73      0.73      1200
weighted avg       0.73      0.73      0.73      1200
```



```
from sklearn.metrics import precision_score, recall_score
print('Precision Large_Carnivores: %.3f' % precision_score(y_test1, y_pred1, pos_label=8))
print('Precision Large_omnivores_and_herbivores: %.3f' % precision_score(y_test1, y_pred1, pos_label=11))
print('Recall Large_Carnivores: %.3f' % recall_score(y_test1, y_pred1, pos_label=8))
print('Recall Large_omnivores_and_herbivores: %.3f' % recall_score(y_test1, y_pred1, pos_label=11))
```

```
Precision Large_Carnivores: 0.706
Precision Large_omnivores_and_herbivores: 0.761
Recall Large_Carnivores: 0.751
Recall Large_omnivores_and_herbivores: 0.717
```





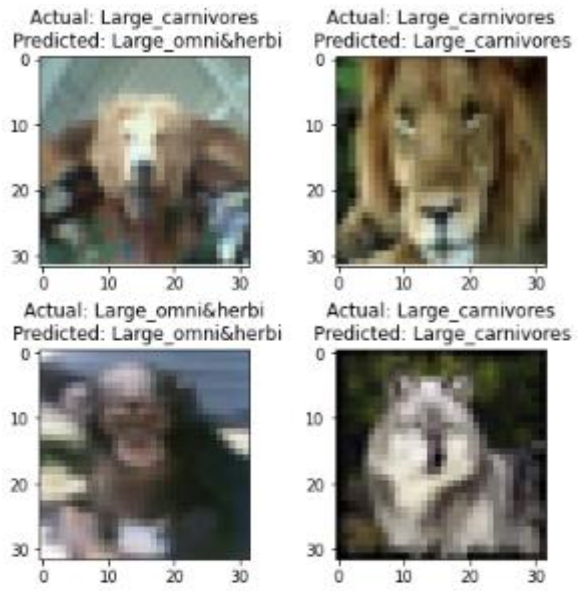
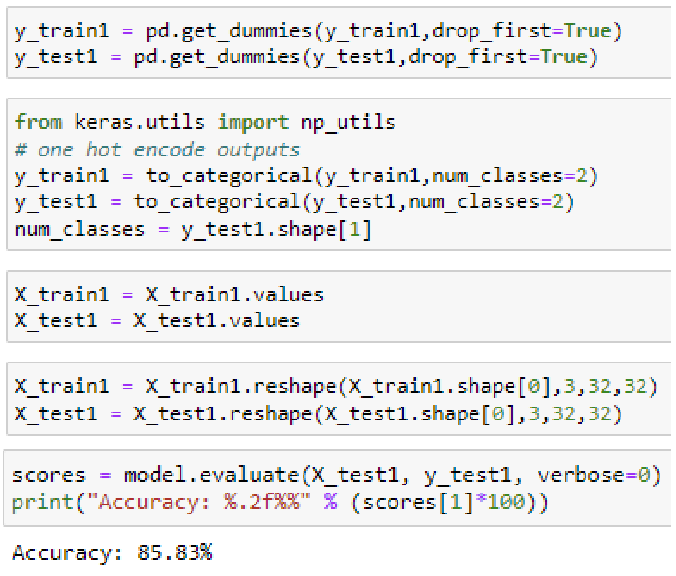

Actual: Large_carnivores
Predicted: Large_omni&herbi
Actual: Large_carnivores
Predicted: Large_carnivores
Actual: Large_omni&herbi
Predicted: Large_omni&herbi
Actual: Large_carnivores
Predicted: Large_carnivores

Fig: Output predicted from Random Forest Classifier

iv. CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Network is a supervised type of Deep learning, most preferable used in image recognition and computer vision. CNN has multiple layers that process and extract important features from the image. The CNN model runs with 150 epochs and batch size of 32 with steps per epoch as 150 having MaxPool and using dropout and image augmentation we tried to prevent overfitting of our model with a learning rate of 0.0005, which is neither too high nor too low. To train generator, I used ImageDataGenerator() to generate and fitting the model with fit library. Once all the epochs were executed, accuracy is calculated with evaluate().

Accuracy obtained: 85.83%



```
y_train1 = pd.get_dummies(y_train1,drop_first=True)
y_test1 = pd.get_dummies(y_test1,drop_first=True)
```

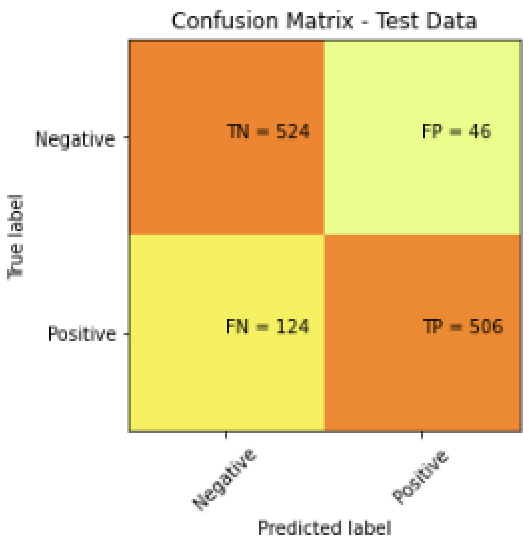
```
from keras.utils import np_utils
# one hot encode outputs
y_train1 = to_categorical(y_train1,num_classes=2)
y_test1 = to_categorical(y_test1,num_classes=2)
num_classes = y_test1.shape[1]
```

```
X_train1 = X_train1.values
X_test1 = X_test1.values
```

```
X_train1 = X_train1.reshape(X_train1.shape[0],3,32,32)
X_test1 = X_test1.reshape(X_test1.shape[0],3,32,32)
```

```
scores = model.evaluate(X_test1, y_test1, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 85.83%
```



Accuracy = (TN+TP)/(TN+FP+FN+TP)
= (524+506)/(524+506+124+46) = 0.8583

```
from sklearn.metrics import classification_report
cp = classification_report(np.argmax(y_test1,axis=1),y_pred1)
print(cp)
```

```
              precision    recall  f1-score   support

           0       0.81      0.92      0.86       570
           1       0.92      0.80      0.86       630

    accuracy                           0.86      1200
   macro avg       0.86      0.86      0.86      1200
weighted avg       0.87      0.86      0.86      1200
```

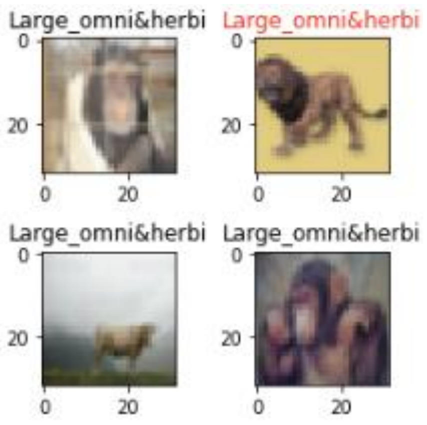


Fig: Output predicted from CNN

III.MULTICLASS CLASSIFICATION

Multiclass Classification, is a task which has more than two classes. In this project, the following classes were classified, Bear, Leopard, Lion, Tiger and Wolf from the Super class Large carnivores. These classes index is found with simple for loop, and then combined into animals_images. Then the training data are splitted, similar to binary classification, into 80–20 ratio. But I was unable to perform precision-recall plot, as it throws the error which multiclass data is not supported. I have used different classifications like Logistic Regression, Decision Tree Classifier & Random Forest Classifier, I also tried CNN but ended with error.

i.LOGISTIC REGRESSION

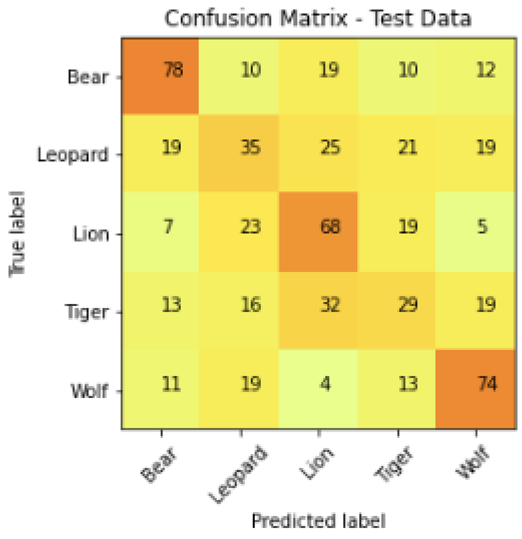
Logistic regression are best suited for binary classification, is limited to two-class classification problems. But some extensions like one-vs-rest can allow logistic regression to be used for multi-class classification problems. Also, we se that after training and predicting with Logistic regression on Multiclass the accuracy is too poor than it performed with binary classification.

Accuracy = 47.33%

```
# fitting logistic regression to the training set(Multi)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

```
print(classifier.score(X_test,y_test))
```

```
0.47333333333333333
```



Accuracy = (78+35+68+29+74)/(78+35+68+29+74+10+19+10+12+25+19 +21+19+5+19+13+4+32+23+16+19+11+13+7+19) = 0.4733

I couldn't obtain the precision-recall plot, as it throwed the error as Multiclass is not supported. But I did obtain the classification report displayed below.

```
# Displaying classification report for logistic regression
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           3       0.61      0.60      0.61       129
          42       0.34      0.29      0.32       119
          43       0.46      0.56      0.50       122
          88       0.32      0.27      0.29       109
          97       0.57      0.61      0.59       121

    accuracy                           0.47       600
   macro avg       0.46      0.47      0.46       600
weighted avg       0.46      0.47      0.47       600
```

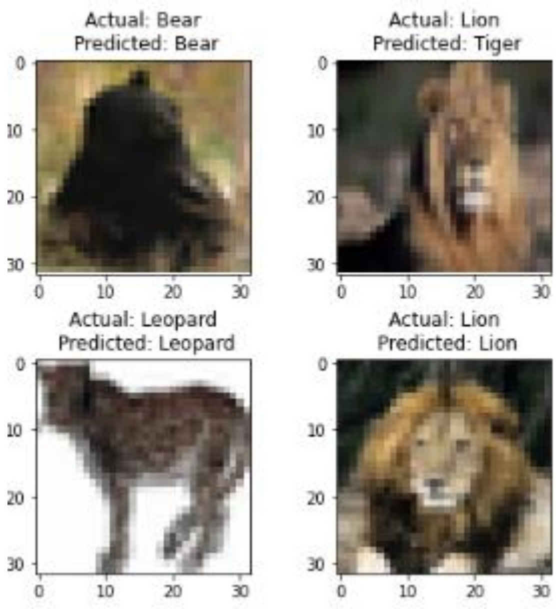
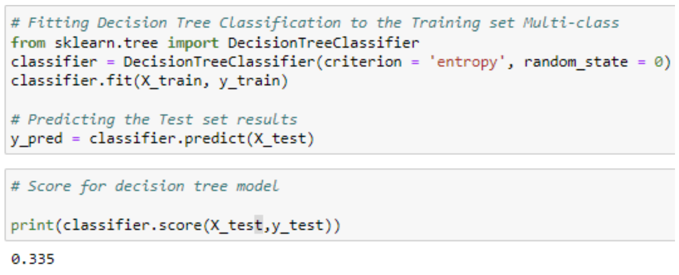


Fig: Output predicted from LR

ii.Decision Tree Classifier

Decision Tree Classifier uses decision trees to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). From the accuracy and other obtained results, we can see that decision tree has performed poorly with so many misclassifications.

Accuracy = 33.5%

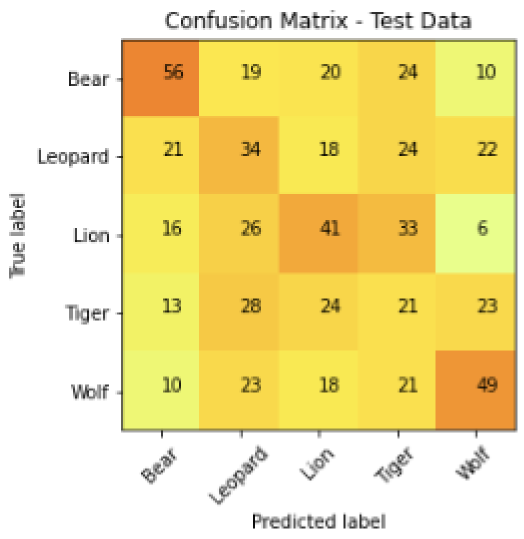


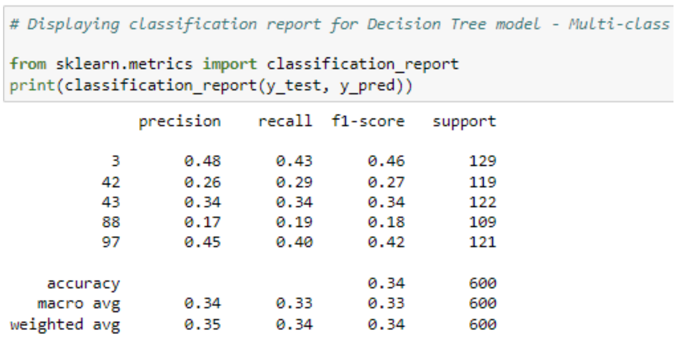
```
# Fitting Decision Tree Classification to the Training set Multi-class
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

```
# Score for decision tree model

print(classifier.score(X_test,y_test))
```

```
0.335
```

Accuracy = (56+34+41+21+49)/(19+20+24+10+18+24+22+6+33+23+21+18+23+10+13+28+24+21+24+16+26+21) = 0.335





```
# Displaying classification report for Decision Tree model - Multi-class

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           3       0.48      0.43      0.46       129
          42       0.26      0.29      0.27       119
          43       0.34      0.34      0.34       122
          88       0.17      0.19      0.18       109
          97       0.45      0.40      0.42       121

    accuracy                           0.34       600
   macro avg       0.34      0.33      0.33       600
weighted avg       0.35      0.34      0.34       600
```

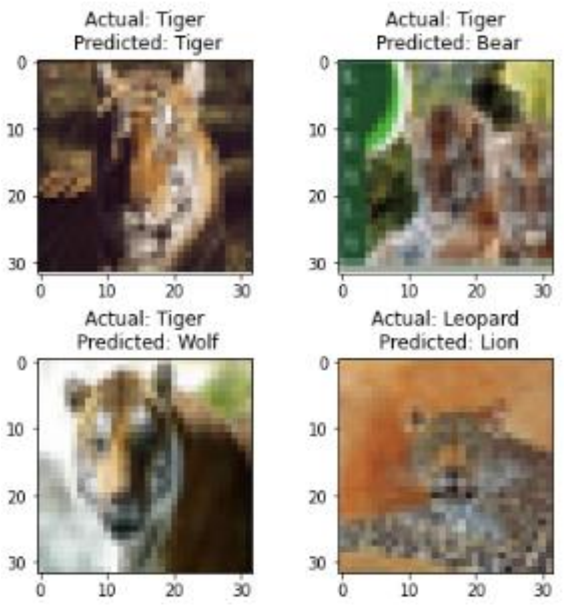
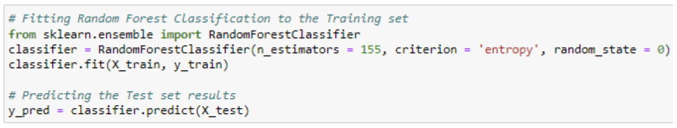


Fig: Output predicted from Decision Tree Classifier

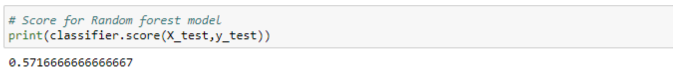
i. Random Forest Classifier

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees. Random forests generally outperformed decision trees and Logistic regression.

Accuracy = 57.167%

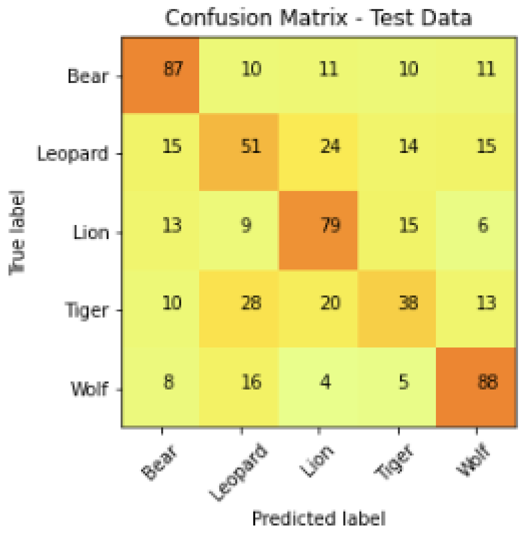


```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 155, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

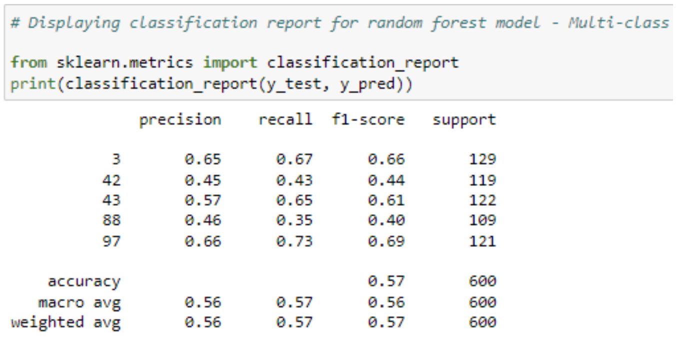


```
# Score for Random forest model
print(classifier.score(X_test,y_test))
```

```
0.5716666666666667
```



Accuracy = (87+51+79+38+88)/(10+11+10+11+24+14+15+15+6+13+8+16+4+5+10+28+20+13+9+15) = 0.57167



```
# Displaying classification report for random forest model - Multi-class

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           3       0.65      0.67      0.66       129
          42       0.45      0.43      0.44       119
          43       0.57      0.65      0.61       122
          88       0.46      0.35      0.40       109
          97       0.66      0.73      0.69       121

    accuracy                           0.57       600
   macro avg       0.56      0.57      0.56       600
weighted avg       0.56      0.57      0.57       600
```

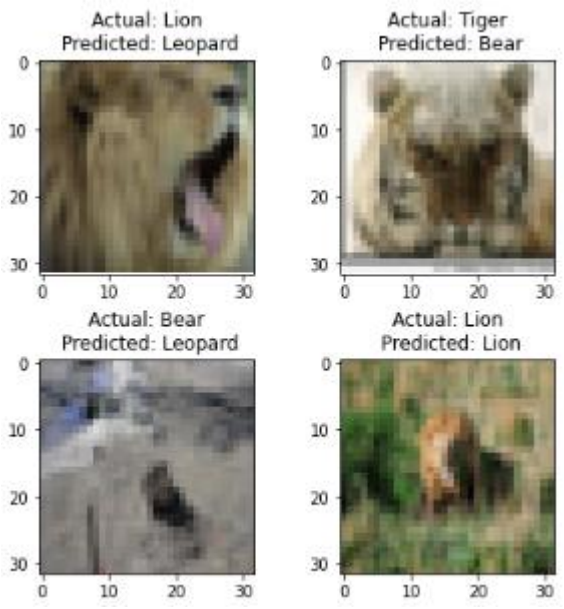
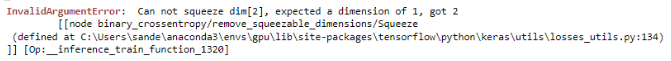


Fig: Output predicted from Random Forest Classifier

V. Convolutional Neural Network

Mostly similar to CNN used in binary classification has been tried to implement with multiclass classification. But ended up with the error displayed below. I believe CNN would have performed better with evaluated than other classifiers.



```
InvalidArgumentError:  Can not squeeze dim[2], expected a dimension of 1, got 2
	 [[node binary_crossentropy/remove_squeezable_dimensions/Squeeze
 (defined at C:\Users\sande\anaconda3\envs\gpu\lib\site-packages\tensorflow\python\keras\utils\losses_utils.py:134)
]] [Op:__inference_train_function_1320]
```

IV. Justification

In case of Binary classification, the accuracy obtained by CNN is far better than other algorithms. The goal of CNN is to reduce the images into a form which is easier to process without feature losses which are critical for getting a good prediction. This means that useful attributes from an already trained CNN can be extracted with its trained weights by feeding data on each level and tune the CNN a bit for the specific task. I picked Logistic Regression is to find out how well it performs as they are specifically best suited for two-class classification problems. I tried Random Forest because it adds additional randomness to the model, while growing the trees and it can handle large dataset efficiently. And I took Decision tress, because that I have picked Random Forest and the classification algorithm is architected with many decision trees and I wanted to test how a single decision tree performs. In case of Multiclass classification, Random Forest has performed better of the rest, as I got error with CNN couldn't obtain the output from it.

V. CONCLUSION

Task 1 has been completed with various binary classifiers like Logistic Regression, Decision Tree Classifier, Random Forest Classifier & CNN. Out of which CNN resulted with better accuracy of 85.83% while the least accuracy obtained was with Decision Tree classifier resulting 60.08%. In case of task 2, Random Forest have performed better than other classifiers, while Decision Tree classifier performed poorly with 33%. Decision Tree classifier is the least performer in both the tasks, while CNN and Random Forest showed good performance.