Amit Gupta
Nov 30 · 6 min read · Listen

Image Classification Techniques and Comparison

What is Image Classification? Let us first understand the concept before getting started.

Image classification refers to **the task of extracting information classes from a multiband raster image.** In simple words it means that a machine learning model which is capable to take unknown image as an input and classify it into the one of the classes for which it is trained for.

For example below is the example for a image classification :

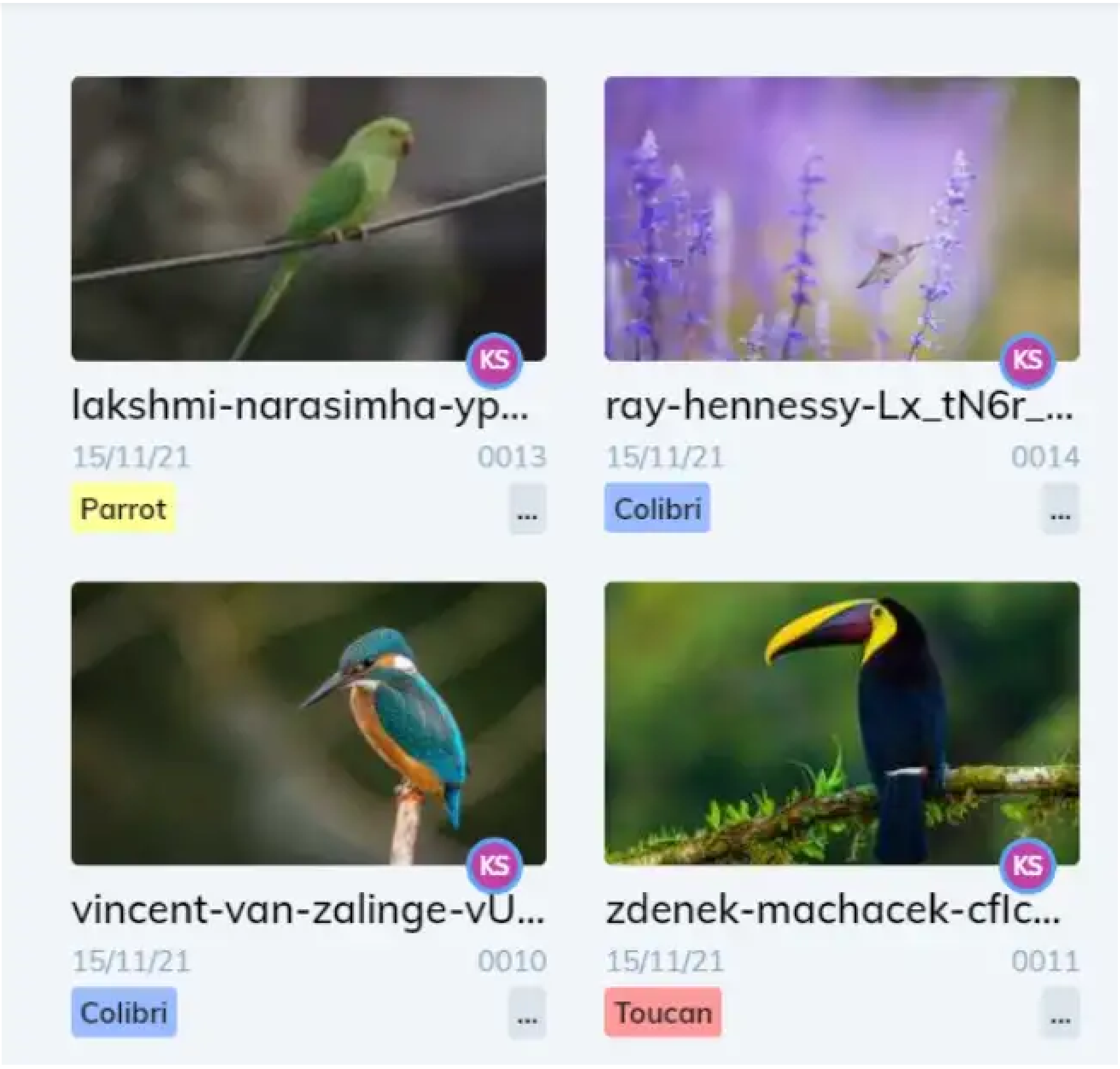


Image Classification using V7 [1]

Today, we will understand this concept by training few models using different algorithms and even one neural network model to get a clear idea.
Outline of this project :

- Data collection
- Preprocessing and Splitting data
- Training

Let us now begin the journey!

We will be using dataset provided on Kaggle. You can find it using this link. Download it and unzip it in a folder. Create a new Colab/kaggle notebook to proceed with the further steps.

Now, let us look into the structure of the data and its size. We have several folders with images in it. In total we have 6 classes namely : *[fox, hyena, cheetah, lion, wolf, tiger]*. As we see there are images of different size, hence we will preprocess it to make it even.

2. Preprocessing and Splitting Data:

Before resizing the data, first we have image path with us. But just having path is not enough we need actual image to work on it. So let us first collect image path from all the folder and then load it into our project and then carry on the further steps.

```
def make_dataframes(sdir):
    bad_images=[]
    classes=[ 'cheetah', 'fox', 'hyena','lion','tiger', 'wolf']
    filepaths=[]
    labels=[]
    all_data = {}
    classlist=sorted(os.listdir(sdir) )
    for klass in classlist:
        classpath=os.path.join(sdir, klass)
        subdir=os.listdir(classpath)[0]
        subpath=os.path.join(classpath,  subdir)
        if os.path.isdir(subpath):
            flist=sorted(os.listdir(subpath))
            for i in range (len(classes)):
                if classes[i] in classpath:
                    klass=classes[i]
            desc=f'{klass:10s}-{subdir:17s}'
            count = 0
            for f in tqdm(flist, ncols=130,desc=desc, unit='files', colour='blu
                if count <=100:
                  count+=1
                  fpath=os.path.join(subpath,f)
                  try:
                      img=cv2.imread(fpath)
                      shape=img.shapepy
                      all_data[fpath] = klass
```

This function is made by Gerry. We will tweak it for our use. After running the code on our data we will have all our images in *all_data* variable.

Then we will split the data into ***training, testing and validation*** sets.

```
all_data = make_dataframes(sdir)
l = list(all_data.items())
random.shuffle(l)
all_data = dict(l)
filepaths = []
labels = []

for entry in all_data:
  filepaths.append(entry)
  labels.append(all_data[entry])

training = filepaths[:int(len(filepaths)*0.8)]
validation = filepaths[int(len(filepaths)*0.8):int(len(filepaths)*0.9)]
testing = filepaths[int(len(filepaths)*0.9):]

train_label = labels[:int(len(labels)*0.8)]
valid_label = labels[int(len(labels)*0.8):int(len(labels)*0.9)]
test_label = labels[int(len(labels)*0.9):]
```

We are splitting train:test:validation into the ratio of 8:1:1

Now we will resize the images we have into 224,224,3 dimension.

```
x_train = []
for img in training:
    img_arr=cv2.imread(img)
    img_arr=cv2.resize(img_arr,(224,224))
    x_train.append(img_arr)

x_test=[]
for img in testing:
    img_arr=cv2.imread(img)
    img_arr=cv2.resize(img_arr,(224,224))
```

```
        img_arr=cv2.resize(img_arr,(224,224))
        x_val.append(img_arr)
```

After getting the resized image we will then normalize the data

```
train_x=np.array(x_train)
test_x=np.array(x_test)
val_x=np.array(x_val)

train_x=train_x/255.0
test_x=test_x/255.0
val_x=val_x/255.0
```

Since we are going to work on machine learning algorithms and not CNN we will convert the images into 2D array to reduce the computation.

```
nsamples, nx, ny, nrgb = train_x.shape
reshaped_train = train_x.reshape((nsamples,nx*ny*nrgb))

nsamples_test, nx_test, ny_test, nrgb_test = test_x.shape
reshaped_test = test_x.reshape((nsamples_test,nx_test*ny_test*nrgb_test))
```

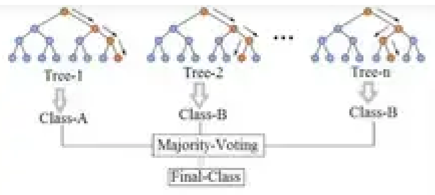
3. Training Models:

We are about to train model using 4 algorithms and 1 CNN.

The algorithms are:

a. Random Forest Classifier

A random forest is a meta estimator that employs averaging to increase predicted accuracy and reduce overfitting after fitting numerous decision tree classifiers to distinct dataset subsamples.



[3]

We will use sklearn library to make use of Random Forest Classifier algorithm.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_s
model=RandomForestClassifier(verbose = 1,n_estimators=150,max_depth=5)
model.fit(reshaped_train,train_label)
```

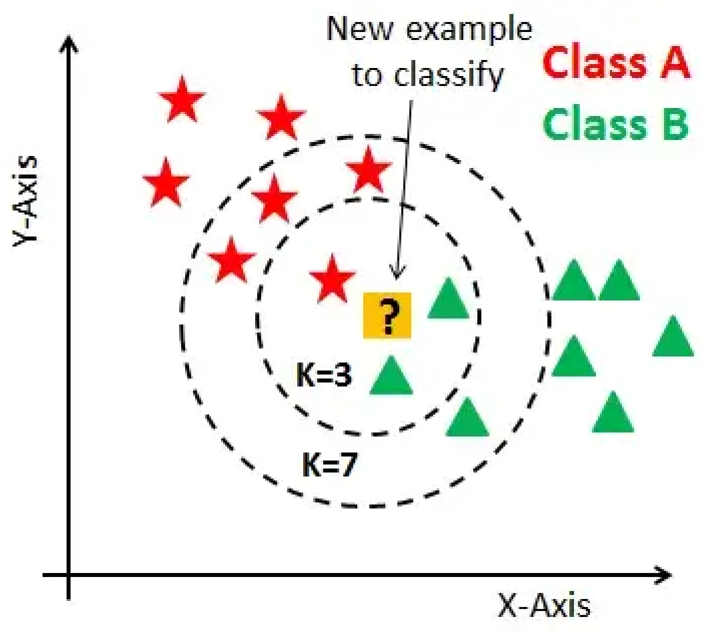
With the above code we are initializing a model with *n_estimators = 150 and max_depth = 5* after trail and error.

After running the code we can get the trained model with the accuracy of **85%.** We will visualize the results in the next step. So let us now continue with the another algorithm.

```
         fox       0.92      0.82      0.87        28
       hyena       0.83      0.68      0.75        37
        lion       0.71      0.90      0.79        30
       tiger       0.86      0.93      0.89        27
        wolf       1.00      0.96      0.98        28

    accuracy                           0.85       182
   macro avg       0.86      0.86      0.86       182
weighted avg       0.86      0.85      0.85       182

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:    0.0s finished
array([[28,  1,  0,  3,  0,  0],
       [ 3, 23,  2,  0,  0,  0],
       [ 0,  1, 25,  8,  3,  0],
       [ 2,  0,  1, 27,  0,  0],
       [ 0,  0,  2,  0, 25,  0],
       [ 0,  0,  0,  0,  1, 27]], dtype=int64)
```

b. KNN

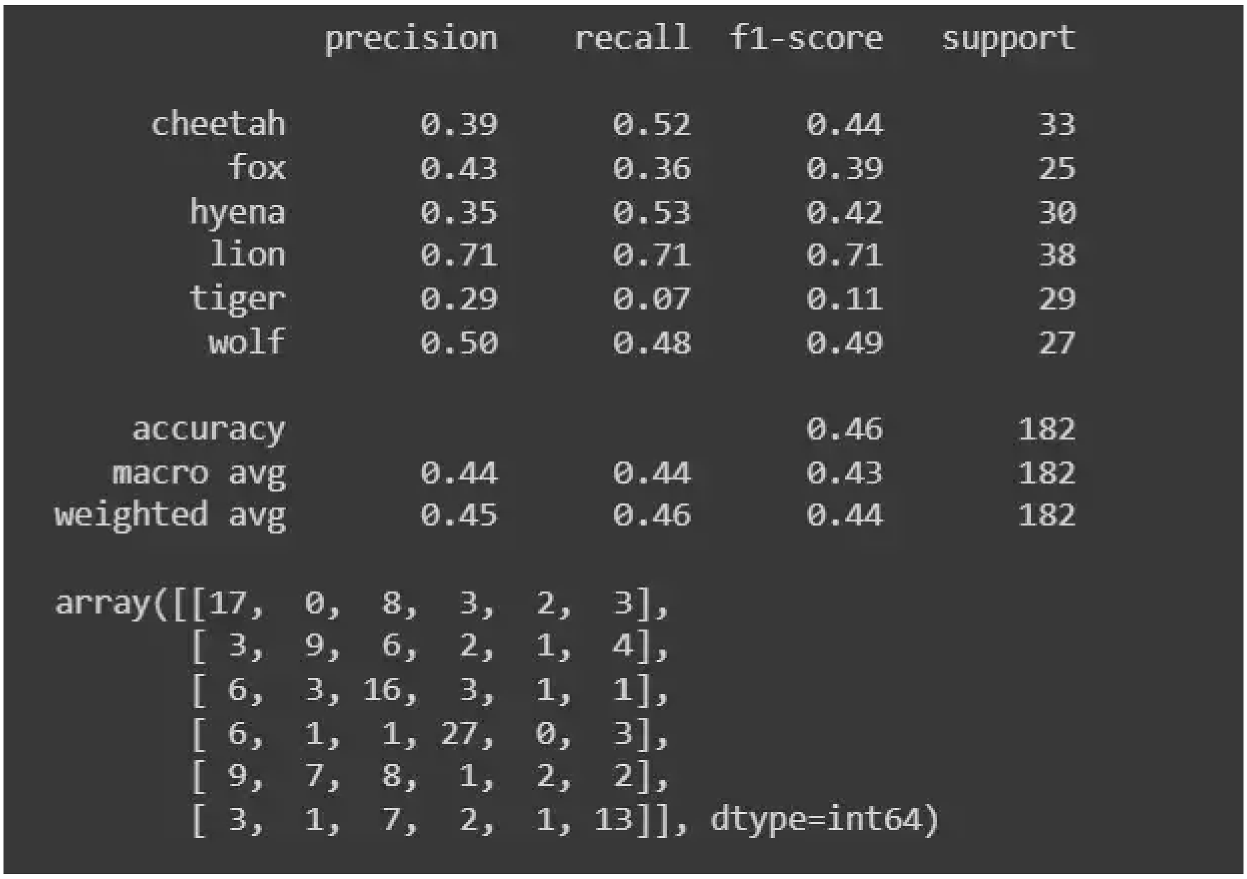
The result of k-NN classification is a class membership. The class that an object is assigned to based on the majority vote of its k closest neighbors is determined by the item's neighbors (k is a positive integer, typically small). The object is simply put into the class of its one nearest neighbor if k = 1.



KNN

```
knn.fit(reshaped_train,train_label)
```

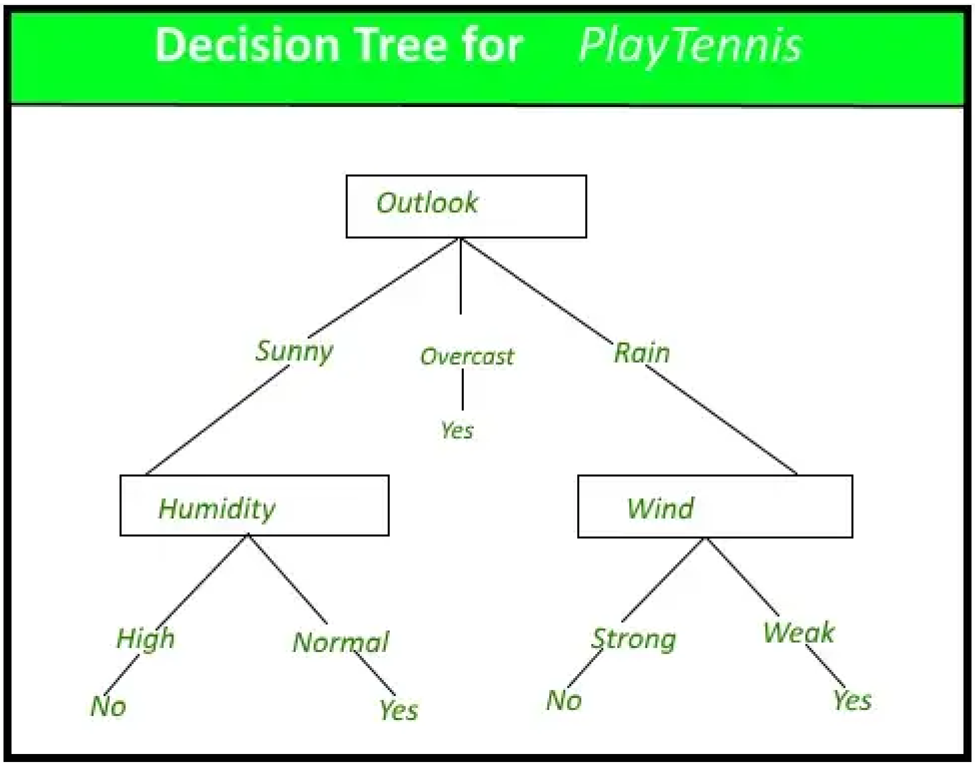
We will create a model with *n_neighbors=8*. After training the model we will test it on our validation set and we get an accuracy of **46%** which is way less than our previous model.



```
              precision    recall  f1-score   support

     cheetah       0.39      0.52      0.44        33
         fox       0.43      0.36      0.39        25
       hyena       0.35      0.53      0.42        30
        lion       0.71      0.71      0.71        38
       tiger       0.29      0.07      0.11        29
        wolf       0.50      0.48      0.49        27

    accuracy                           0.46       182
   macro avg       0.44      0.44      0.43       182
weighted avg       0.45      0.46      0.44       182

array([[17,  0,  8,  3,  2,  3],
       [ 3,  9,  6,  2,  1,  4],
       [ 6,  3, 16,  3,  1,  1],
       [ 6,  1,  1, 27,  0,  3],
       [ 9,  7,  8,  1,  2,  2],
       [ 3,  1,  7,  2,  1, 13]], dtype=int64)
```

c. Decision Tree Classifier

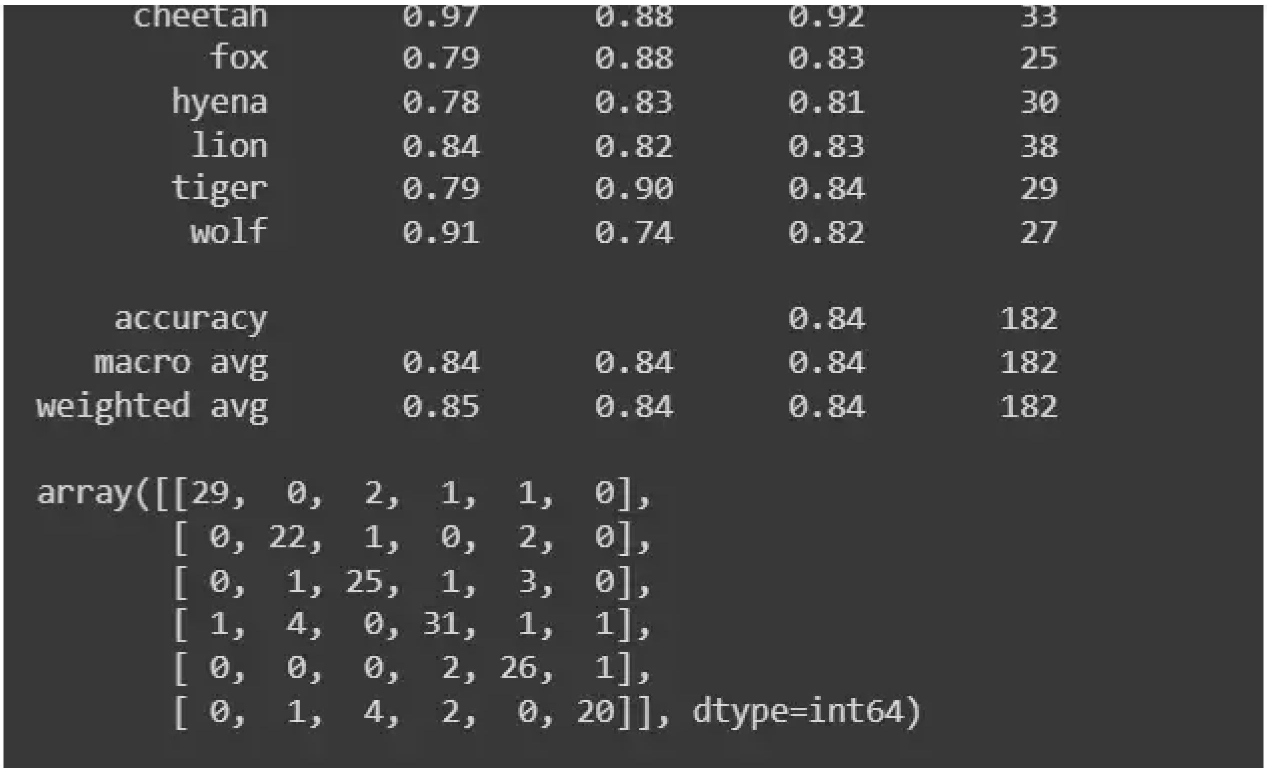
A decision support tool known as a decision tree employs a tree-like model to represent options and their potential outcomes, including utility, resource costs, and chance event outcomes. One technique to show an algorithm that solely uses conditional control statements is to use this method.



Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(reshaped_train,train_label)
```

With the above trained model we get an accuracy of **84%.**



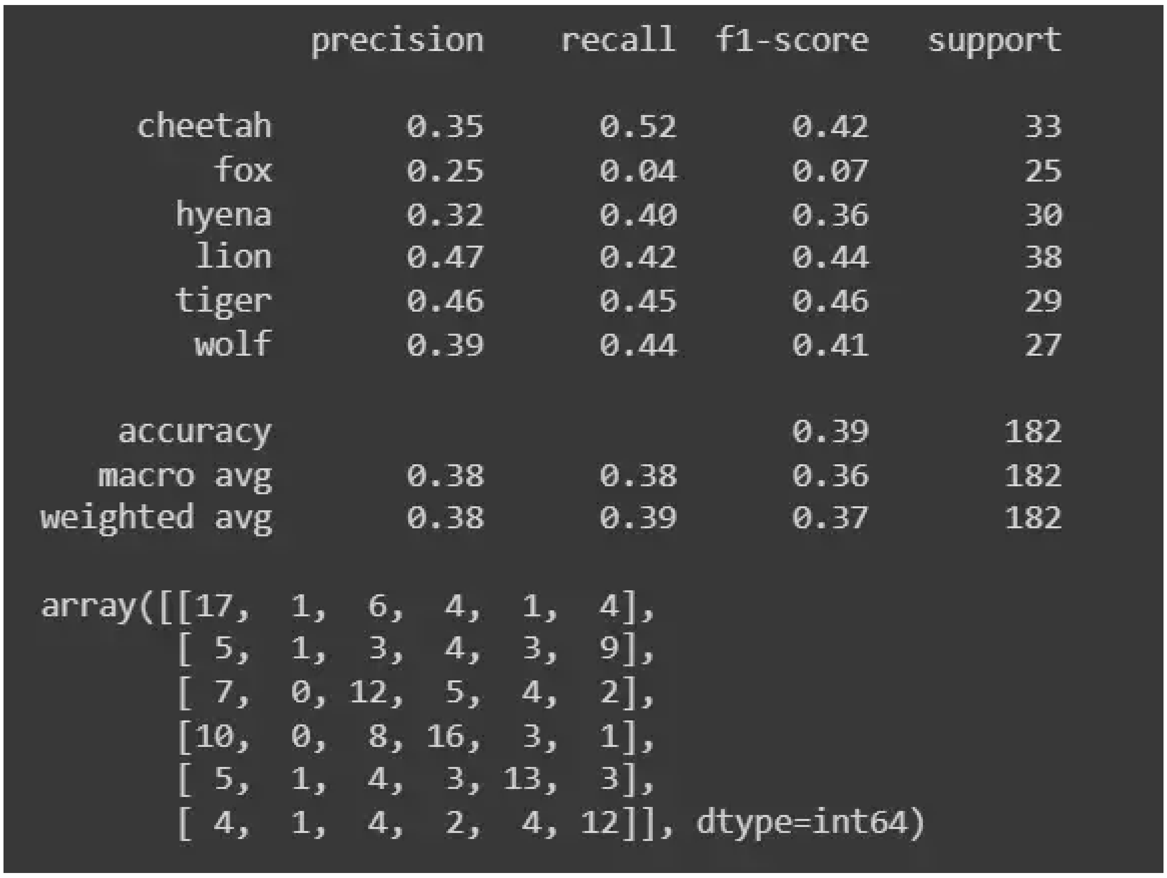
```
     cheetah       0.97      0.88      0.92        33
         fox       0.79      0.88      0.83        25
       hyena       0.78      0.83      0.81        30
        lion       0.84      0.82      0.83        38
       tiger       0.79      0.90      0.84        29
        wolf       0.91      0.74      0.82        27

    accuracy                           0.84       182
   macro avg       0.84      0.84      0.84       182
weighted avg       0.85      0.84      0.84       182

array([[29,  0,  2,  1,  1,  0],
       [ 0, 22,  1,  0,  2,  0],
       [ 0,  1, 25,  1,  3,  0],
       [ 1,  4,  0, 31,  1,  1],
       [ 0,  0,  0,  2, 26,  1],
       [ 0,  1,  4,  2,  0, 20]], dtype=int64)
```

d. Naive Bayes' Classifier:

The family of straightforward "probabilistic classifiers" known as "naive Bayes classifiers" in statistics is based on the application of Bayes' theorem with strong independence assumptions between the features. Despite being among the simplest Bayesian network models, they can reach great levels of accuracy when used in conjunction with kernel density estimation.

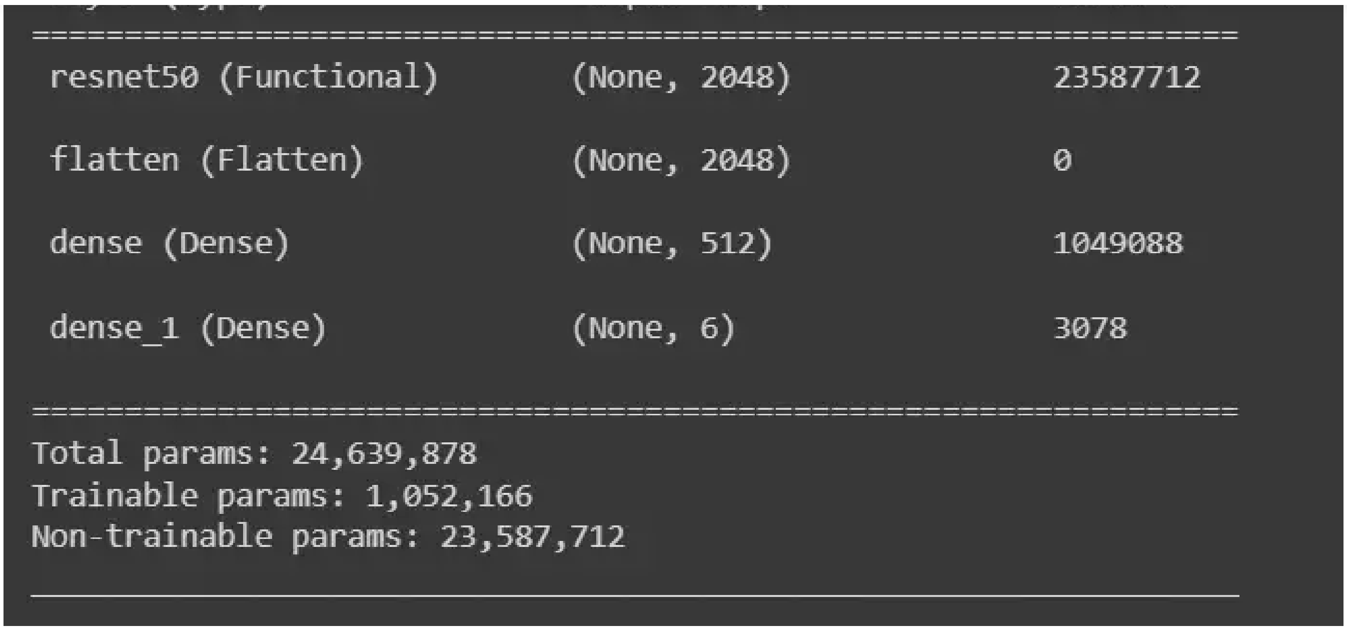
$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

```
from sklearn.naive_bayes import GaussianNB
```


```
              precision    recall  f1-score   support

     cheetah       0.35      0.52      0.42        33
         fox       0.25      0.04      0.07        25
       hyena       0.32      0.40      0.36        30
        lion       0.47      0.42      0.44        38
       tiger       0.46      0.45      0.46        29
        wolf       0.39      0.44      0.41        27

    accuracy                           0.39       182
   macro avg       0.38      0.38      0.36       182
weighted avg       0.38      0.39      0.37       182

array([[17,  1,  6,  4,  1,  4],
       [ 5,  1,  3,  4,  3,  9],
       [ 7,  0, 12,  5,  4,  2],
       [10,  0,  8, 16,  3,  1],
       [ 5,  1,  4,  3, 13,  3],
       [ 4,  1,  4,  2,  4, 12]], dtype=int64)
```

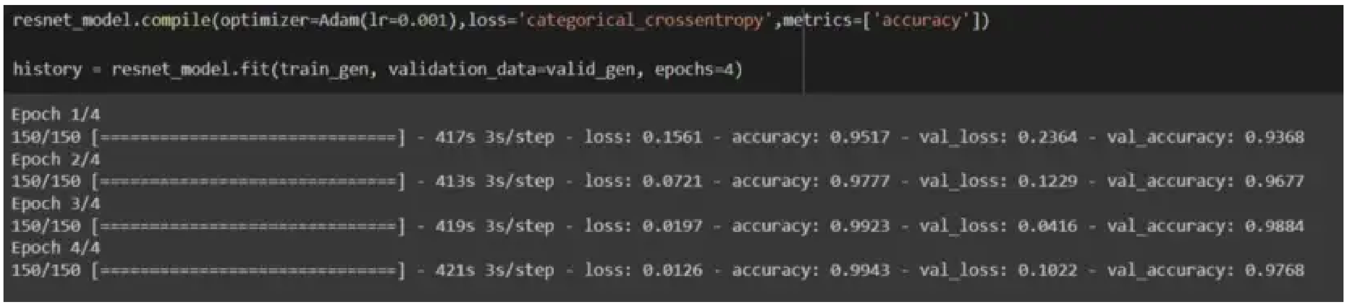
e. CNN model (Using ResNet50)

Here we will be performing transfer learning. The base model is ResNet50 which we will add some more additional layers on above of it to make it more efficient for our dataset.



```
=================================================================
 resnet50 (Functional)       (None, 2048)             23587712

 flatten (Flatten)           (None, 2048)             0

 dense (Dense)               (None, 512)              1049088

 dense_1 (Dense)             (None, 6)                3078

=================================================================
Total params: 24,639,878
Trainable params: 1,052,166
Non-trainable params: 23,587,712
_________________________________________________________________
```

Above is the summary of the model that we are going to train.

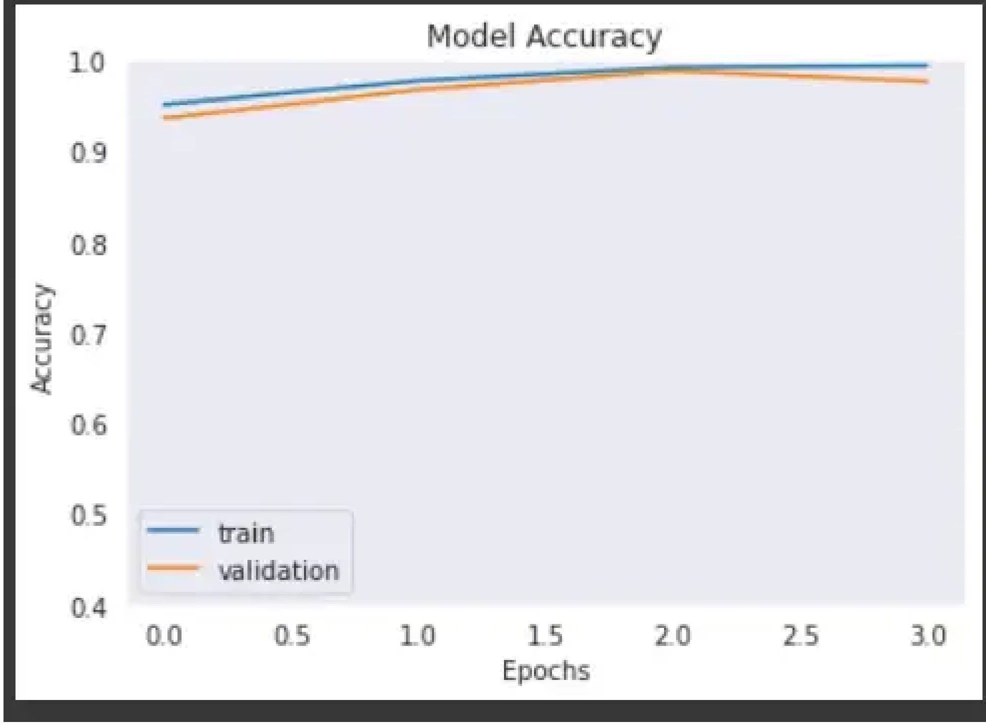


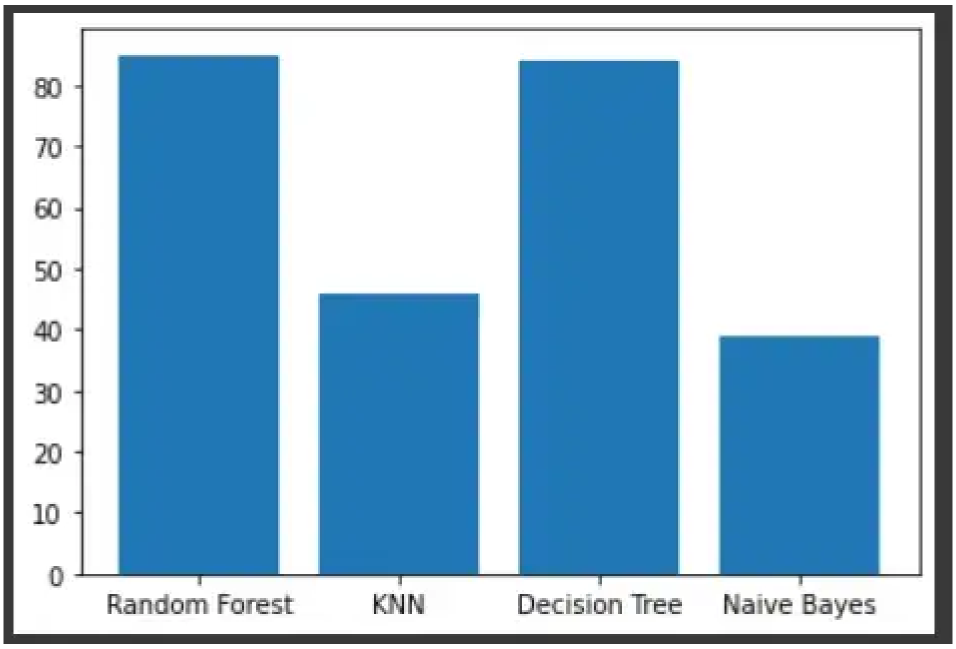
```
resnet_model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])

history = resnet_model.fit(train_gen, validation_data=valid_gen, epochs=4)
```

```
Epoch 1/4
150/150 [==============================] - 417s 3s/step - loss: 0.1561 - accuracy: 0.9517 - val_loss: 0.2364 - val_accuracy: 0.9368
Epoch 2/4
150/150 [==============================] - 413s 3s/step - loss: 0.0721 - accuracy: 0.9777 - val_loss: 0.1229 - val_accuracy: 0.9677
Epoch 3/4
150/150 [==============================] - 419s 3s/step - loss: 0.0197 - accuracy: 0.9923 - val_loss: 0.0416 - val_accuracy: 0.9884
Epoch 4/4
150/150 [==============================] - 421s 3s/step - loss: 0.0126 - accuracy: 0.9943 - val_loss: 0.1022 - val_accuracy: 0.9768
```

As we can see we trained the CNN for 4 epoch which resulted into the accuracy of **99.43%.** This is the power of CNN!

```
plt.plot(history.history['val_accuracy'])
plt.axis(ymin=0.4,ymax=1)
plt.grid()
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'validation'])
plt.show()
```





Youtube Video Link :

Contributions:

- Did trial and error on all the algorithms to tune the hyperparameters.
- Create a metric graph for the CNN
- Based on the confusion matrix generated, tweaked the data to get an efficient model.
- Wrote the code for the algorithms used by myself. Except for CNN took reference from [3].
- Understood the concepts of the algorithms in-depth and performed it to get clear idea of them.

Challenges:

- Training with images requires computation power for CNN.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#:~:text=A%20random%20forest%20classifier.,accuracy%20and%20control%20over%2Dfitting.

[3] https://www.kaggle.com/code/gpiosenka/animals-f1-score-99-9