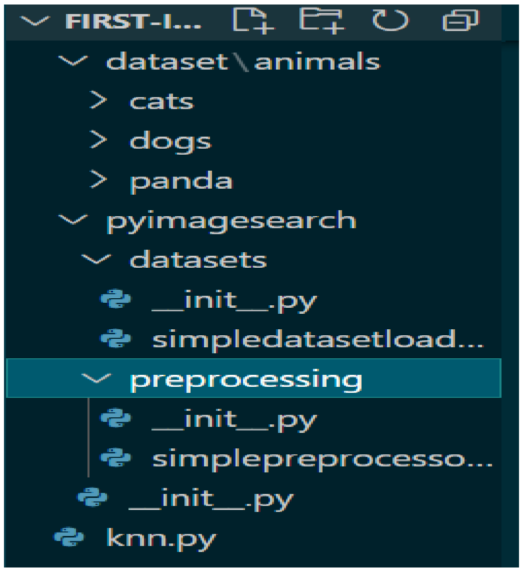
First Image Classifier with opencv

We will be working with the below file structure.And we will go over the python files explain how they function,The dataset contains folders for cats,dogs,and panda. For the classification ,Knn algorithm we will used.



```
FIRST-I...
  dataset\animals
    cats
    dogs
    panda
  pyimagesearch
    datasets
      __init__.py
      simpledatasetload...
    preprocessing
      __init__.py
      simplepreprocesso...
    __init__.py
  knn.py
```

We will look at the preprocessing file first be preprocess the each image as we are loading it

Preprocessering:There are several preprocessing methods one can do on the images but here we are just going to be resizing the image ignoring aspect ratio.

1) Script:Import open cv

```
# import the necessary packages
import cv2
```

2) Script:Defining the class to take width,height and inter with default value cv2.INTER_AREA

```
class SimplePreprocessor:
    def __init__(self, width, height, inter=cv2.INTER_AREA):
        # store the target image width, height, and interpolation
        # method used when resizing
        self.width = width
        self.height = height
        self.inter = inter
```

3) Script:Define a method preprocess method for the class and resize the image passed into it

```
    def preprocess(self, image):
        # resize the image to a fixed size, ignoring the aspect
        # ratio
        return cv2.resize(image, (self.width, self.height),
            interpolation=self.inter)
```

DataLoader:Defining the dataloader class

1) Script:Import the needed packages and define the constructor taking preprocessors one will like to use and if no preprocessor is passed,set the preprocessed to be used to an empty array

```
# import the necessary packages
import numpy as np
import cv2
import os

class SimpleDatasetLoader:
    def __init__(self, preprocessors=None):
        # store the image preprocessor
        self.preprocessors = preprocessors

        # if the preprocessors are None, initialize them as an
        # empty list
        if self.preprocessors is None:
            self.preprocessors = []
```

2) Script:We define a load method taking in imagePaths as parameter and verbose default -1 which help to print to console the number of images loaded.Data,label holds the images and corresponding labels respectively.Looping through the imagePaths we load the image and get the class name from the paths

```

    def load(self, imagePaths, verbose=-1):
        # initialize the list of features and labels
        data = []
        labels = []

        # loop over the input images
        for (i, imagePath) in enumerate(imagePaths):
            # load the image and extract the class label
assuming
            # that our path has the following format:
            # /path/to/dataset/{class}/{image}.jpg
            image = cv2.imread(imagePath)
            label = imagePath.split(os.path.sep)[-2]
```

3) Script: checking if any preprocessors exist.If true apply each preprocessor on the the image and save back to image variable .Repeat the same for other preprecessors by looping.we append the preprocessed image into data and and also append it corresponding label

```
  # check to see if our preprocessors are not None
            if self.preprocessors is not None:
                # loop over the preprocessors and apply each
to
                # the image
                for p in self.preprocessors:
                    image = p.preprocess(image)

            # treat our processed image as a "feature vector"
            # by updating the data list followed by the labels
            data.append(image)
            labels.append(label)
```

4) Script:if verbose is specified we get the number of image loaded printed in console.In either case of verbose we return the data array and the labels array back to the caller.

```
            # show an update every `verbose` images
            if verbose > 0 and i > 0 and (i + 1) % verbose ==
0:
                print("[INFO] processed {}/{}".format(i + 1,
                    len(imagePaths)))

        # return a tuple of the data and labels
        return (np.array(data), np.array(labels))
```

The KNN Classifier.

1) Script: load necessary packages

```
# import the necessary packages
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from pyimagesearch.preprocessing import
SimplePreprocessor
from pyimagesearch.datasets import SimpleDatasetLoader
from imutils import paths
import argparse
```

2) Script:get program parameter indicating path to dataset,the nearest which is option and jobs

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
    help="path to input dataset")
ap.add_argument("-k", "--neighbors", type=int, default=1,
    help="# of nearest neighbors for classification")
ap.add_argument("-j", "--jobs", type=int, default=-1,
    help="# of jobs for k-NN distance (-1 uses all
available cores)")
args = vars(ap.parse_args())
```

3) Script:Get the filepath of each image,create an object from our simple preprocessor indicating to use a width and height of 32 respectively. Create an object from our loader using the preprocessor object and load the images using the imagePaths and telling it to print to console the number of images loaded.the result is a tuple of data images and corresponding label.We reshape it from 3000*32*32*3 to 3000* (32*32*3) because KNN algorithm need just single list of pixel intensities

```
# grab the list of images that we'll be describing
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))

# initialize the image preprocessor, load the dataset
from disk,
# and reshape the data matrix
sp = SimplePreprocessor(32, 32)
sdl = SimpleDatasetLoader(preprocessors=[sp])
(data, labels) = sdl.load(imagePaths, verbose=500)
data = data.reshape((data.shape[0], 3072))
```

```
# show some information on memory consumption of the
images
print("[INFO] features matrix: {:.1f}MB".format(
    data.nbytes / (1024 * 1024.0)))
```

4) Script:We encode our label and split our dataset into train and test

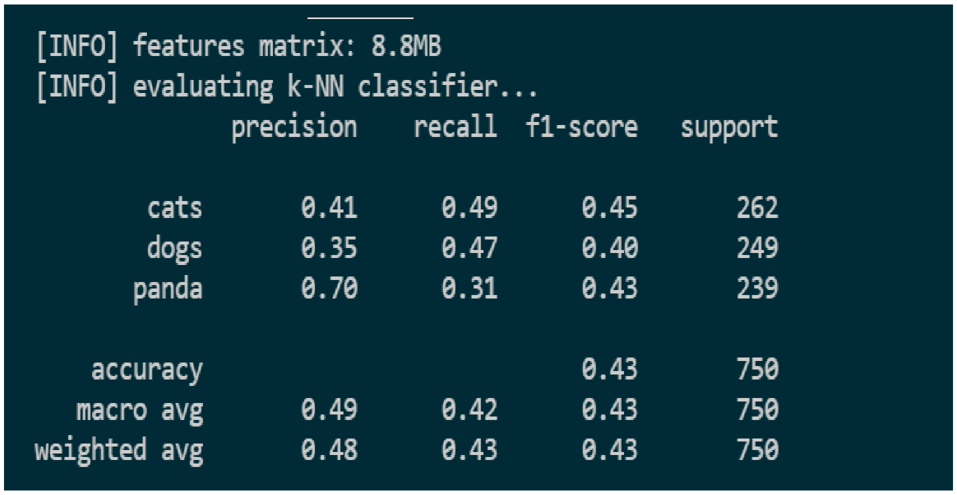
```
# encode the labels as integers
le = LabelEncoder()
labels = le.fit_transform(labels)

# partition the data into training and testing splits
using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data,
labels,
    test_size=0.25, random_state=42)
```

5) Script:we define our knn classifier ,train it using model.fit and the train data.We also report it evaluation on the test data

```
# train and evaluate a k-NN classifier on the raw pixel
intensities
print("[INFO] evaluating k-NN classifier...")
model =
KNeighborsClassifier(n_neighbors=args["neighbors"],
    n_jobs=args["jobs"])
model.fit(trainX, trainY)
print(classification_report(testY, model.predict(testX),
    target_names=le.classes_))
```

Result:The classifier didn't now perform well on both cats and dogs

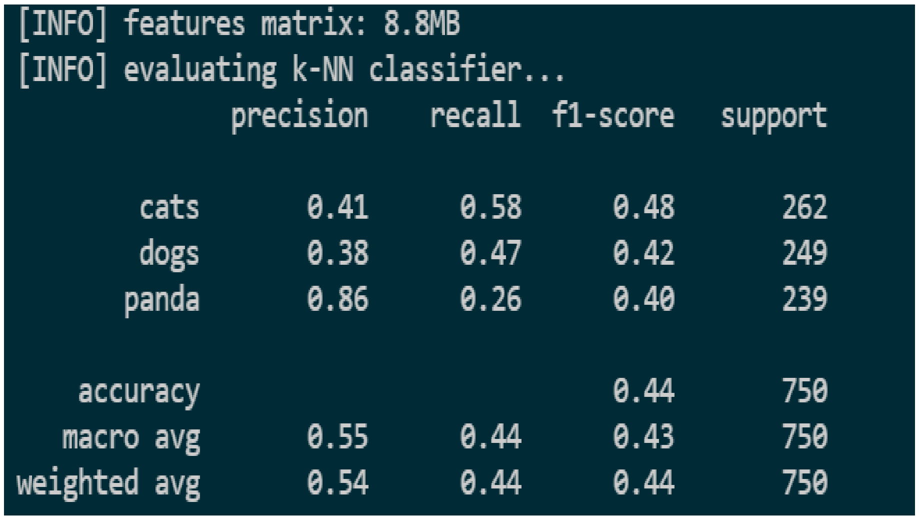


```
[INFO] features matrix: 8.8MB
[INFO] evaluating k-NN classifier...
              precision    recall  f1-score   support

        cats       0.41      0.49      0.45       262
        dogs       0.35      0.47      0.40       249
       panda       0.70      0.31      0.43       239

    accuracy                           0.43       750
   macro avg       0.49      0.42      0.43       750
weighted avg       0.48      0.43      0.43       750
```

Increasing k to 3 gives the below result.Overall performance is 54%,learning on panda increase and dogs improved a bit while noting happened for cat.



```
[INFO] features matrix: 8.8MB
[INFO] evaluating k-NN classifier...
              precision    recall  f1-score   support

        cats       0.41      0.58      0.48       262
        dogs       0.38      0.47      0.42       249
       panda       0.86      0.26      0.40       239

    accuracy                           0.44       750
   macro avg       0.55      0.44      0.43       750
weighted avg       0.54      0.44      0.44       750
```