Amrita Vishwa Vidyapeetham
Centre for Excellence in Computational Engineering and Networking
Amrita School of Engineering, Coimbatore

Image Classification

Prepared By:
M. Prasanna Teja [CB.EN.U4AIE21035]
Reddy Hema Radhika [CB.EN.U4AIE21050]
P. Sai Ravula [CB.EN.U4AIE21041]

Supervised by:
Ganga Gowri B
Asst. Professor

An End Semester Project submitted to the CEN department as a part of course evaluations of **Python for Machine Learning** for B. Tech in **Computer Science Engineering** – **Artificial Intelligence.**

TABLE OF CONTENTS

1.1 – Abstract

In this section of our paper, we will be implementing Image Classification for a dataset. Our implementation features the following classification techniques

- K – Nearest Neighbours
- Convolution Neural Networks

1.2 – Introduction

Image Classification is a task in computer vision and machine learning which involves assigning a label or class to an image based on its features. It is a fundamental problem in the field of computer vision and is used in many applications.

The goal of image classification is to train a machine learning model on a dataset of labelled images so that the model can make predictions on new unseen images. This process of classification involves extracting features such as texture, colour, shape and use them to train and classify the data. The accuracy of the model which is trained depends on the quality and the amount of data given to the data. Also, the hyper parameters used for the model also greatly affects the accuracy.

1.3 – Problem & Solution statements

Given an image dataset of cats and dogs, the task is to train a machine learning model that can accurately distinguish between the two classes of images. After training the model, it must be able to make predictions on unseen data of dogs and cats.

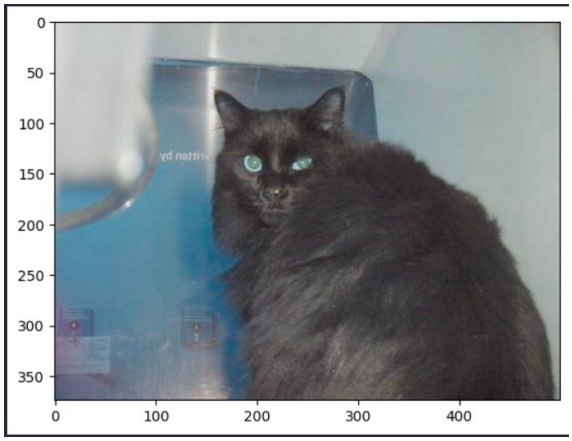
The problem is a binary classification task where the goal is to predict one of the two classes (dog or cat). So, we will be using two machine learning algorithms namely KNN and CNN to make our classification possible.

1.4 – Dataset & Data pre-processing

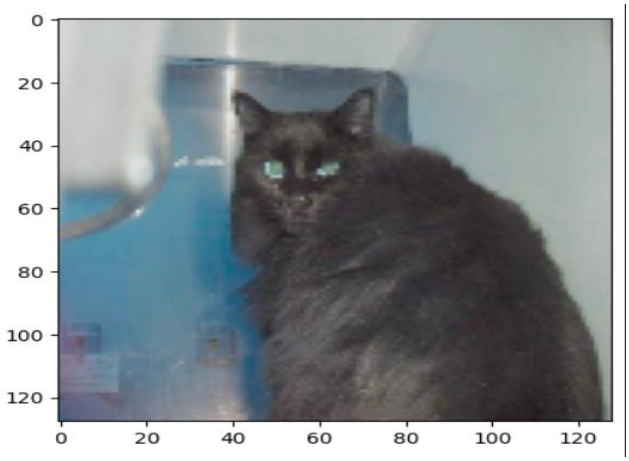
The dataset which we have used to train our model is taken from the Kaggle website which contains two image folders named training and test set which contains the images of cats and dogs (2500 images each).

Our dataset contains images of different resolutions. So, we will be converting the resolution of each and every image in our training set to the size of 128 x 128 so that our model

will get different images of same size and hence the training will be done in a sophisticated way.



Before changing resolution



After changing resolution

Also, we know that every image is an array of pixels which holds the intensity value of the picture. Generally, the value that each pixel holds is in the range of 0 to 255. Since, we will be applying operations on the picture which in turn is the operations resembles as the operations on the pixels itself. So, it is better to scale these values so that it will be easy for the algorithm to do computations on it.

```
array([[[[218, 218, 218],
         [215, 215, 215],
         [225, 225, 225],
         ...,
         [205, 213, 212],
         [212, 223, 221],
         [214, 225, 223]],
```

Before pixel scaling

```
array([[[[0.85490196, 0.85490196, 0.85490196],
         [0.84313725, 0.84313725, 0.84313725],
         [0.88235294, 0.88235294, 0.88235294],
         ...,
         [0.80392157, 0.83529412, 0.83137255],
         [0.83137255, 0.8745098 , 0.86666667],
         [0.83921569, 0.88235294, 0.8745098 ]],
```

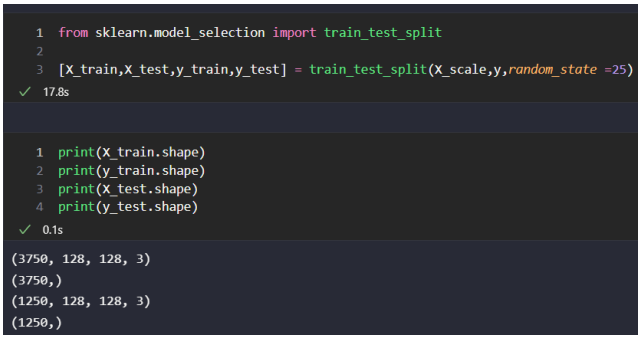
After pixel scaling

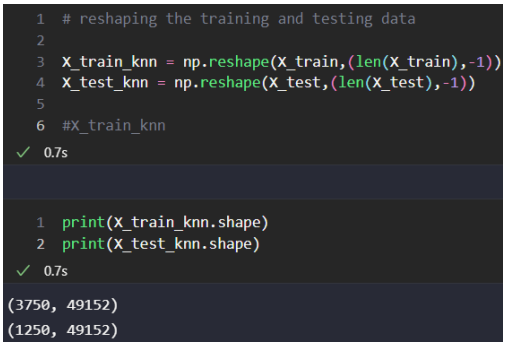
1.5 – Method – 1

The first method we are using for this classification is KNN. It works by finding the K nearest data points to a test data point and predicting the class based on the majority class of the nearest neighbours. The distance metric used for finding the nearest neighbours can be Euclidean, Manhattan**.** We used Euclidean method for our project. For finding that we will append all the distances in the list and we will sort it out to make it easy for finding the k-nearest neighbours of the test data point.

```python
from collections import Counter
import numpy as np

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):

        # Compute distances between x and all data points inn training data
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]

        # Sort by distance and return indices of the first k neighbors
        k_idx = np.argsort(distances)[: self.k]

        # Extract the labels of the k nearest neighbor training samples
        k_neighbor_labels = [self.y_train[i] for i in k_idx]

        # return the most common class label
        most_common = Counter(k_neighbor_labels).most_common(1)
        return most_common[0][0]
```

Now we will split our data into training and testing and give it to the model. Before giving it to model, we need to reshape the dimensions of our training and testing data as the dimensions won't satisfy the operations inside the working code.





Method – 2

The second method we are using for the classification is CNN. It is one of the mostly used in image classification technique in computer vision. It uses convolutional layers to extract features from the input image, pooling layers to reduce the spatial dimensions and complexity, and fully connected layers for prediction. The convolutional layers use filters to detect patterns in the image.

In our model, our input image is of 128 x 128 dimensions. Our filter is of size 3 x 3 which moves 3 blocks of each picture and extracts the features as it goes. So, after filtering the size will be changed to 126x126.

```
model = Sequential() #stacked layers in a sequence

# HIDDEN LAYERS
model.add(Conv2D(32, (3,3), activation = 'relu', input_shape = (128,128,3)))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(64, (3,3), activation= 'relu'))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(128, (3,3), activation= 'relu'))
model.add(MaxPooling2D((2,2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))

# OUTPUT LAYER
model.add(Dense(2, activation='sigmoid'))
```

The formula for it is **(input size-filter size+1).** So, 128-3+1 will be 126. After max pooling which is of size 2x2, it will be get halved as it takes only best feature out of the 4 elements i.e from 2x2. So, it forms 63x63 matrix. The process continues like this.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896

max_pooling2d (MaxPooling2D  (None, 63, 63, 32)       0
)
```

After convolution, we will flatten the output into a single layer. Next, we will take 2 fully connected layers. The dense layer helps to map the low-level features learned by the convolutional layers to high-level features that can be used for classification. The second dense layer contains the 2 classes and we used sigmoid so it gives us the probability of occurring

each class and by using that we will predict which class our image is whether it is dog or cat we will predict by using this.

If the output is not what we want we will do back processing. Backpropagation is an algorithm used to train neural networks, especially deep neural networks, to minimize the loss function. It works by calculating the gradient of the loss with respect to the network's parameters and updating the parameters in the opposite direction of the gradient to minimize the loss.

```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])  # for Back propagation
```

Data Augmentation:

Data augmentation is a technique used to artificially increase the size of a dataset by transforming the existing data in various ways. In data augmentation, various techniques are applied to the existing data, such as rotation, flipping, scaling, and adding noise, to generate new, augmented data. These augmented samples can then be used together with the original data to train the model.

```python
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory("D://SEM - 3//PML//p1//CNN//training_set",target_size = (128,128),batch_size = 16,class_mode = 'binary')
test_set = test_datagen.flow_from_directory("D://SEM - 3//PML//p1//CNN//test_set",target_size = (128,128),batch_size = 1,class_mode = 'binary')
```

1.6 – Results and Inferences

The KNN model which we have trained gave us approximately 55% accuracy. This low accuracy is due to the dataset and the pre-processing techniques that have been used and can be increased by performing more pre-processing on the data taken.

```python
k=5
clf = KNN(k=k)
clf.fit(X_train_knn, y_train)
predictions = clf.predict(X_test_knn)

print(accuracy_score(predictions,y_test)*100)
print(classification_report(predictions,y_test))
```

✓ 17m 32.6s

```
54.559999999999995
              precision    recall  f1-score   support

           0       0.69      0.54      0.60       809
           1       0.40      0.56      0.47       441

    accuracy                           0.55      1250
   macro avg       0.55      0.55      0.54      1250
weighted avg       0.59      0.55      0.56      1250
```

Classification report for KNN

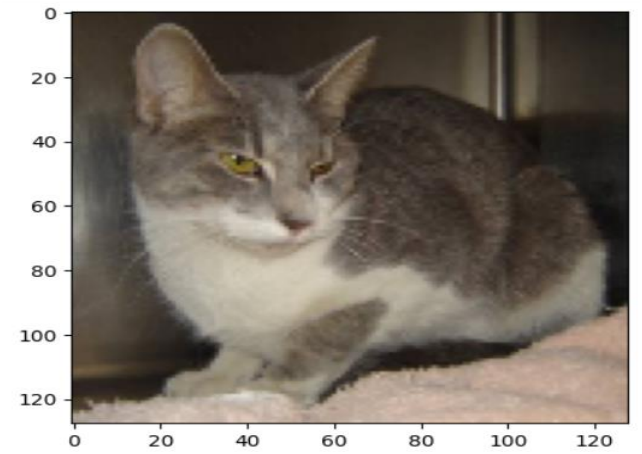
After fitting the data and running the model, at the end of final epoch we got 83% accuracy for our model as shown. But when we evaluated the model for test dataset using a command, we got 78% accuracy and 51% loss percentage. This reduction in accuracy is again due to the dataset and can be improved using pre – processing.

```
Epoch 15/15
313/313 [==============================] - 190s 607ms/step - loss: 0.3791 - accuracy: 0.8368

<keras.callbacks.History at 0x28175611570>


    1  model.evaluate(test_set)

2000/2000 [==============================] - 79s 39ms/step - loss: 0.5135 - accuracy: 0.7880

[0.5134977102279663, 0.7879999876022339]
```

Now, we will test our CNN model on a new image which it hasn't seen before.

```
1  t_path = "D://SEM - 3//PML//p1//CNN//test_set//cats//cat.4352.jpg"
2
3  img = image.load_img(t_path, target_size=(128,128,3))
4  img_array = image.img_to_array(img)
5  img_batch = np.expand_dims(img,axis=0)
6
7  plt.imshow(img)
8  plt.show
9  print(img.size)
```

Test on New data set




```
1/1 [========
IT IS A CAT
```

1.7 – Applications

Image classification has a wide range of real-world applications were detecting an image accurately is very important. Some of them include object recognition which is widely used by many automated robots and is also used in self-driving cars were identifying the people on the road, traffic signs etc is done. Another significant application of image classification is Face recognition which is used in modern mobile phones and many security systems. Yet another application is in medical image analysis where it is used to diagnose and classify medical conditions in images such as X-rays, MRI scans etc.

These are just a few applications involving image classification.

1.8 – Conclusion

As you can see, we have implemented two machine learning models which classifies the images into their respective classes where the latter model is doing the classification more accurately. The accuracy of the model depends on dataset taken also. There are many algorithms which we can use to classify the images. Our implementation is just a basic model which works well on a small dataset. But, in real these algorithms may not work that well for a large dataset. We can use much complex CNNs for the complex datasets.