Kaggle “Dogs vs. Cats” Challenge — Complete Step by Step Guide — Part 1

In my opinion, one of the most exciting fields in Artificial Intelligence is Deep Learning. I find it interesting as to how easily we can extract knowledge from complex raw data structures such as images, audio etc. Moreover, I found Kaggle to be the best platform to practice and implement deep learning algorithms as Kaggle allows the user to find and publish data sets, explore and build models.

In this series of article “Keras Dogs vs. Cats Challenge”, I will try to give you a broad understanding of solving any Image Classification problem. we will address the ‘Dogs vs. Cats” problem with:

Introduction

In this article, we will be solving the famous Kaggle Challenge “Dogs vs. Cats” using Logistic Regression model from Scikit Learn. Scikit Learn is a machine learning library for a Python programming language which offers various important features for machine learning such as classification, regression, and clustering algorithms. For those who want to learn more about Scikit Learn, I find this great article from Deepanshu Gaur.

In this first article, we will discuss in depth about:

- Preparation and Pre-processing of Dataset
- Implementation of a Logistic Regression Model using Scikit Learn
- Training of Logistic Regression Model using Training Data
- Testing of the Trained Model on Test Data
- Key Insights and Conclusion

At the end of this article, you will have a working model for the Kaggle challenge “Dogs vs. Cats”, classifying images as cats vs dog. Moreover, you will have a better understanding of why CNN's are used for computer vision problems.

> *These set of articles are more focused on implementation details rather than the conceptual details of the algorithm. However, I will link the source for a better understanding of the*

algorithm.

Preparation and Pre-processing of Dataset

The dataset for this challenge can be found here. The training archive contains 25,000 images of dogs and cats and testing archive contains 12,500 images of dogs and cats. Unzip the folder in the root directory.

First, we import some libraries.

```
import os, cv2, itertools # cv2 -- OpenCV
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

Next, we define some global variables:

```
TRAIN_DIR = './train/'
TEST_DIR = './test1/'

ROWS = 64
COLS = 64
CHANNELS = 3
```

Now, Let's define two variable train_images and test_images which stores the path information of all the images from the root directory in the TRAIN_DIR and TEST_DIR respectively.

To be concrete, train_images, and test_images looks something like this:

train_images = [' ./train/dog.2677.jpg', './train/cat.7657.jpg', ...]

test_images = ['./test1/4391.jpg', './test1/1769.jpg', ...]

```
train_images = [TRAIN_DIR+i for i in os.listdir(TRAIN_DIR)]
test_images = [TEST_DIR+i for i in os.listdir(TEST_DIR)]
```

We define a function “read_image(file_path)” which reads the image at a given file path using OpenCV (cv2).

```
def read_image(file_path):
  img = cv2.imread(file_path, cv2.IMREAD_COLOR)
  return cv2.resize(img, (ROWS, COLS),interpolation=cv2.INTER_CUBIC)
```

Next, we define a function “prep_data(images)” which takes the list of all the image path (train_images, test_images) and prepare the data in the format needed by the Scikit Learn Logistic Regression method. We will discuss the format of the data later in the article.

```
def prep_data(images):
  m = len(images)
  n_x = ROWS*COLS*CHANNELS

  X = np.ndarray((n_x,m), dtype=np.uint8)
  y = np.zeros((1,m))
  print("X.shape is {}".format(X.shape))

  for i,image_file in enumerate(images) :
    image = read_image(image_file)
    X[:,i] = np.squeeze(image.reshape((n_x,1)))
    if 'dog' in image_file.lower() :
      y[0,i] = 1
    elif 'cat' in image_file.lower() :
      y[0,i] = 0
    else : # for test data
      y[0,i] = image_file.split('/')[-1].split('.')[0]

    if i%5000 == 0 :
      print("Proceed {} of {}".format(i, m))

  return X,y
```

Let’s, call our “prep_data()” function to prepare our training data into the format needed by the Scikit Learn Logistic Regression method.

```
X_train, y_train = prep_data(train_images)
X_test, test_idx = prep_data(test_images)
```

You should see an output like this.

```
X.shape is (12288, 25000)
Proceed 0 of 25000
Proceed 5000 of 25000
Proceed 10000 of 25000
Proceed 15000 of 25000
Proceed 20000 of 25000
X.shape is (12288, 12500)
Proceed 0 of 12500
Proceed 5000 of 12500
Proceed 10000 of 12500
```

Output

If yes, Congratulation! You have successfully prepared your dataset for the training of Logistic Regression model. If no, please comment below with your query, I will be happy to help.

Let's analyze our prepared train and test data.

```
print("Train shape: {}".format(X_train.shape))
print("Test shape: {}".format(X_test.shape))
```

```
Train shape: (12288, 25000)
Test shape: (12288, 12500)
```

Train and Test Data shape

Our X_train contains 25000 images and each image has been prepared as a 12288 (ROWS*COLS*CHANNELS = 64*64*3) featured vector. Similarly, our Y_train contains 12500 images and each image has been prepared as a 12288 featured vector.

Now, Let's define the classes for cats as 0 and dogs as 1.

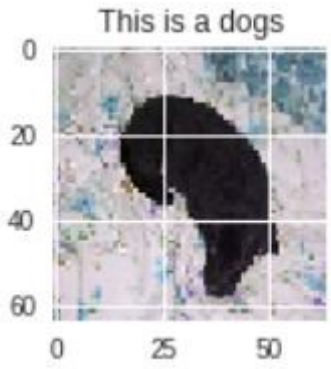
```
classes = {0: 'cats',
           1: 'dogs'}
```

We define a function "show_images(X, y, idx)" that takes the argument as X_train, y_train and the index of the image you want to see respectively and show you the image using matplotlib.pyplot library.

```
def show_images(X, y, idx) :
  image = X[idx]
  image = image.reshape((ROWS, COLS, CHANNELS))
  plt.figure(figsize=(4,2))
  plt.imshow(image)
  plt.title("This is a {}".format(classes[y[idx,0]]))
  plt.show()
```

Now, let's see a sample image. You can change the value of the third parameter (idx) here to see different pictures of our training data.

```
show_images(X_train.T, y_train.T, 2)
```



Output

Implementation of a Logistic Regression Model using Scikit Learn

The idea of Logistic Regression is to find a relationship between features and the probability of a particular outcome. For those who want to learn more about Logistic Regression, here is a great article by Apoorva Agrawal.

First, let's import the library for logistic regression from Scikit Learn.

```
# Logistic Regression
from sklearn.linear_model import LogisticRegressionCV
```

Implementation of a Logistic Regression model using Scikit Learn is super easy and simple. Just one line of code. YAY! We define the instance of Logistic Regression. We name the instance as clf (classifier). You can name it whatever you want.

```
clf = LogisticRegressionCV()
```

Scikit Learn contains two models for Logistic Regression, LogisticRegression() and LogisticRegressionCV(). The instance of the LogisticRegressionCV() divides the Train dataset into different Train/Validation Set combinations before training and helps test the model across different splits of data. The process is called K-Fold Cross Validation (that's where CV comes from). In case you want to know more about K-Fold Cross Validation, this is a great article by Krishni Hewa. Also, you can watch this video by Udacity.

Training of Logistic Regression Model using Training Data

First, taking the transpose of the X_train and y_train as this is the format needed by the Logistic Regression instance we defined earlier.

```
X_train_lr, y_train_lr = X_train.T, y_train.T.ravel()
```

Training the model using the fit method of the LogisticRegressionCV() instance. This may take time depending on the computation power of your system.

```
clf.fit(X_train_lr, y_train_lr)
```

You should see an output like this.

```
LogisticRegressionCV(Cs=10, class_weight=None, cv='warn', dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='warn', n_jobs=None, penalty='l2',
           random_state=None, refit=True, scoring=None, solver='lbfgs',
           tol=0.0001, verbose=0)
```

Output after Successful Training of Logistic Regression Model

If yes, Congratulation! you have successfully trained your Logistic Regression Model for classification an image as a cat or dog. If no, please comment below with your query, I will be happy to help.

If you do not have a GPU on your system, I will highly recommend you to train your model on Google Colab. The code in the GitHub link is also written on Google Colab.

> *Google Colab* — Google's free cloud service *for **AI developers**. With Colab, you can develop deep learning applications on the **GPU for free**. It gives you a free access to a **Nvidia Tesla K80 GPU** and **12 GB of RAM**. I myself trained my models on Google Colab and its easy and mucher fast to train on. Moreover, it's almost similar to a jupyter notebook. For more info, see this great article by fuat.*

Let's first see the accuracy of our trained model.

```
print("Model accuracy: {:.2f}%".format(clf.score(X_train_lr,
y_train_lr)*100))
```

Model Accuracy: 73%

It's a great start for our journey. As we are able to get an accuracy of 73% using Logistic Regression Model. In the next article, we will use state-of-the-art Convolutional Neural Network (CNN) for the same task and our accuracy will be much better than this. But for now, this is a great start. Congratulations!

Testing of the Trained Model on Test Data

Now as we have successfully trained our Logistic Regression Model for classifying dogs and cats, it's time to test it on unseen test data and see our classifier in action.

Let's define a "show_image_prediction(X, idx, model)" function that takes X_train.T, index of an image from the test set you want to do prediction on and trained model as an argument respectively and shows a plot (from matplotlib.pyplot library) indicating the image and our model prediction.

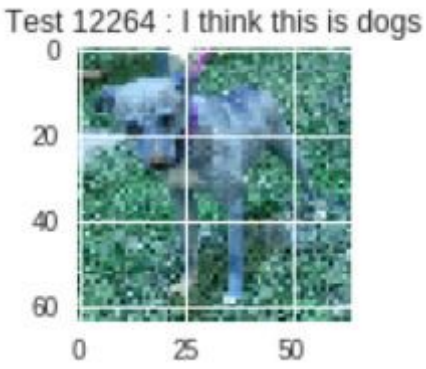
```
def show_image_prediction(X, idx, model) :
  image = X[idx].reshape(1,-1)
  image_class = classes[model.predict(image).item()]
  image = image.reshape((ROWS, COLS, CHANNELS))
  plt.figure(figsize = (4,2))
  plt.imshow(image)
  plt.title("Test {} : I think this is {}".format(idx, image_class))
  plt.show()
```

We now choose some 10 random images from the test set and see the predictions made by our algorithm.

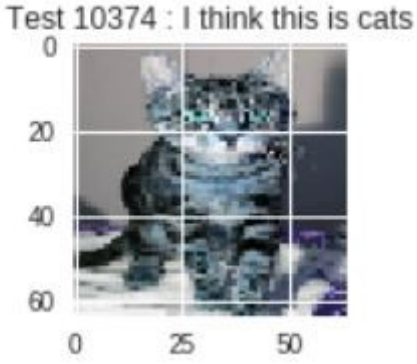
```
X_test_lr, test_idx = X_test.T, test_idx.T

for i in np.random.randint(0, len(X_test_lr), 10) :
  show_image_prediction(X_test_lr, i, clf)
```

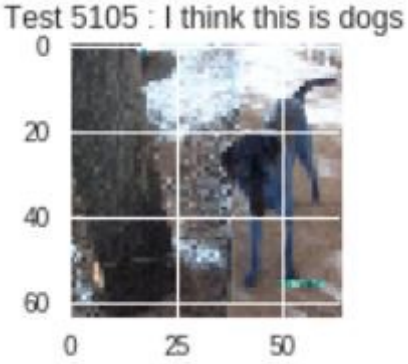
Results:



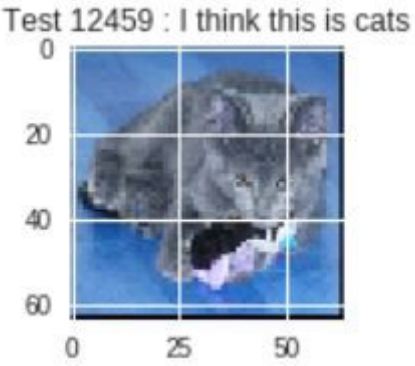
Prediction on Test Set Image



Prediction on Test Set Image



Prediction on Test Set Image



Prediction on Test Set Image

Yipeee! Our model is making quite good predictions. Congratulations on successfully developing a Logistic Regression Model for Image Classification.

Key Insights and Conclusion

- Even after pre-processing the image to dimensions (64, 64, 3) which is quite low resolution, the feature set for each image is of size 12288 which can be large for the dataset size of 25000 images and can be led to overfitting of the model.
- Due to the large resolution of images which converts to large feature vector for each image, models like logistic regression and simple neural networks often do not perform very good on computer vision applications. The main reason for this behavior is the huge amount of weights to be trained during backpropagation.
- Due to these reasons, a state-of-the-art algorithm for computer vision is Convolutional Neural Network(CNN) which we will discuss in the next article. We will see that CNN's give a much greater performance on computer vision problems.