The University of Texas at El Paso

CS 4363 Final Semester Project

Vehicle Prediction using Standford Cars Dataset

Author:

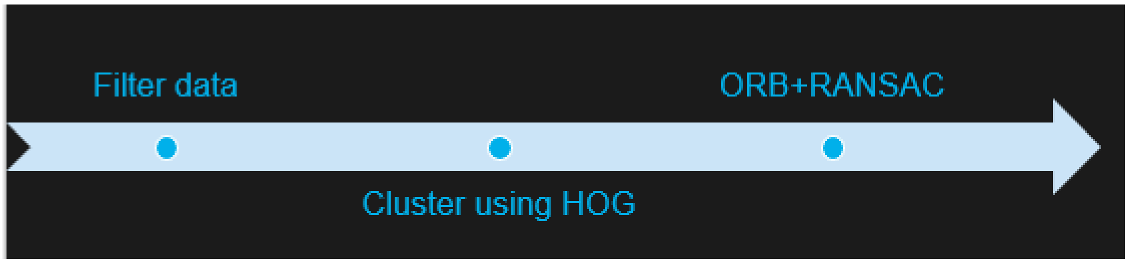
R Noah Padilla

1. Introduction

Vehicle prediction software applications using common technologies today is not as prevalent as it could be considering how far other software applications like facial recognition or weather prediction is with the use of computer vision and machine learning algorithms. In light of this situation, I proposed and completed a machine learning and computer vision program that was able to identify vehicles without the use of any type of CNN. The dataset that was used to complete this project was the Stanford Cars Dataset which contains 16,185 images of 196 classes of cars, however, I only used 8144 images due to labels and measuring performance.

2. Approaching the problem

The goal is to predict the vehicle **make** and **model** of a given input image, hence a classification algorithm at hand. Although there are many ways to approach this problem, I wanted to use knowledge from the material taught in this class to solve this problem so that's what I did –first, cluster the dataset then classify the image of interest looking inside the cluster centroid its closest too. But what **features** from these images would we be using for clustering and classifying? First, knowing that different vehicles have different shapes (different slopes at different parts of the vehicle) I decided to use Histogram of Oriented Gradients to represent each image rather than pixel intensities, mean image, or some other image representation. Next, to classify the image or vehicle of interest is by looking at the cluster centroid closest to the vehicle of interest then use the ORB feature extractor and the RANSAC feature matching algorithm to get the best possible match by comparing each image within that cluster. To track the progress of my approach I had to use the given train data (8144 images) as the full dataset rather than the 16,185. Essentially, doing this it would allow me to measure the performance of my algorithm since the other 8041 images were unlabeled. Below is a representation of the process.



3. Algorithm

In short, the algorithm that was used was –

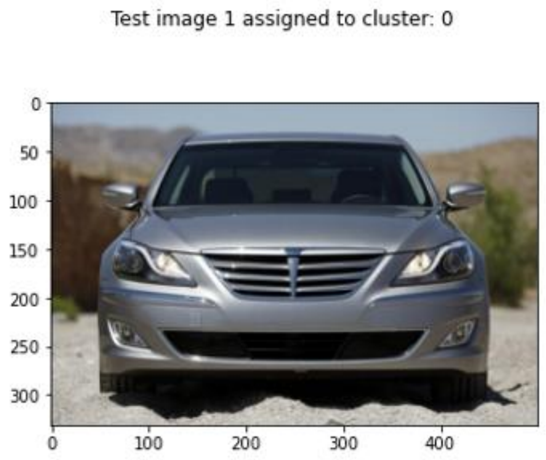
1. Filter the data – Remove any images that were not 3 layers deep (not R, G, B)
2. Convert the Train and Test samples into HOG representations
3. Cluster the Training samples using k-means clustering algorithm and HOG image representations with 2 clusters (cars and trucks)
4. Predict which cluster each of the Test samples would be in using HOG representations
5. For each test sample in its predicted cluster – find its match using ORB and RANSAC of the original image features (R, G, B image **NOT HOG**)

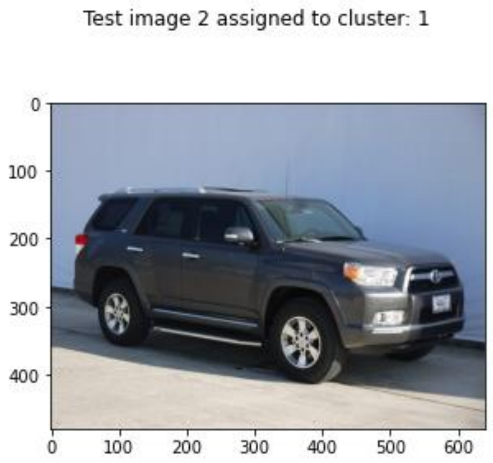
4. Experiments

1. **The first experiment** was to cluster the data by cars and trucks – to do this I found that modifying the number of bins when converting images to their respective HOG representations would help yield a **car cluster** and a **truck cluster**. The vehicles that were no cars or trucks went into the cluster they most looked similar with (for example a hummer was mostly associated with the truck cluster and a van was mostly associated with the car cluster). Furthermore, 12-69 bins were shown to be insufficient to cluster cars and trucks into different clusters. After setting the number of bins to 128, sufficient clustering was shown to demonstrate a car and truck dataset using 2000 images. This showed that the number of samples that needed to be trained when clustering had to be AT LEAST above 2000 samples. Below is a run that demonstrated clustering classification (0=car,1=truck) -
2. **The second experiment** was to start classifying the dataset using 5389 total samples with 99.9965% (5371 images) train data and .35% (18 images) test data split. This split allowed me to train as much as possible while allowing a decent amount to be tested.
3. **The third experiment** was to start classifying the dataset using 8144 total samples with 99.78% (8126 images) train data and .22% (18 images) test data split. This split allowed me to train as much as possible while allowing a decent amount to be tested.

5. Results

1. For experiment 1, sufficient results were shown to separate cars and trucks into separate clusters. Below are 2 examples of that –

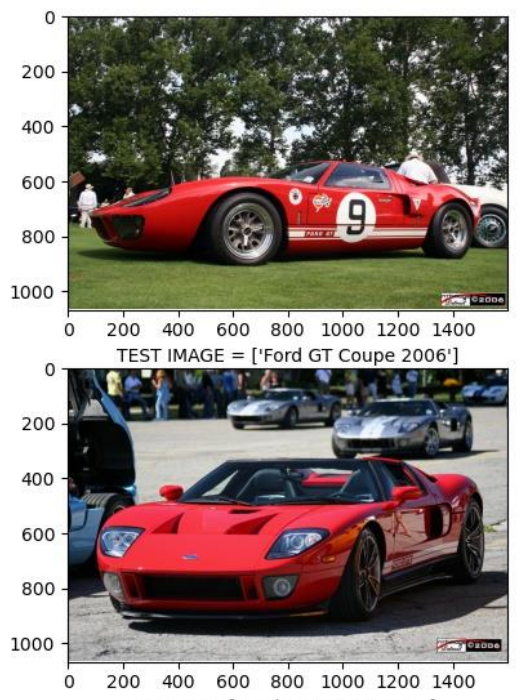




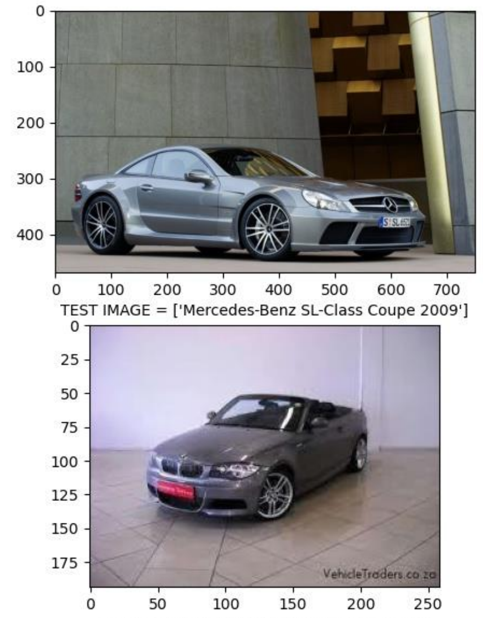
2. For experiment 2, I attained an accuracy of 22%. The total time taken to convert the data to HOG representation took 1338 seconds or 22 minutes. Unfortunately, due to the inefficiency of this program and the deadline of this project I was not able to print matches of this run or the total time taken to find matches of all test samples using ORB and RANSAC.

3. For experiment 3, I attained an accuracy of 47%. The total time taken to convert the images to HOG representations took 2237 seconds or 37 minutes. Unfortunately, due to the deadline of this project I was not able to print the total time taken to find matches of all test samples using ORB and RANSAC. Some matches of this run are shown below –

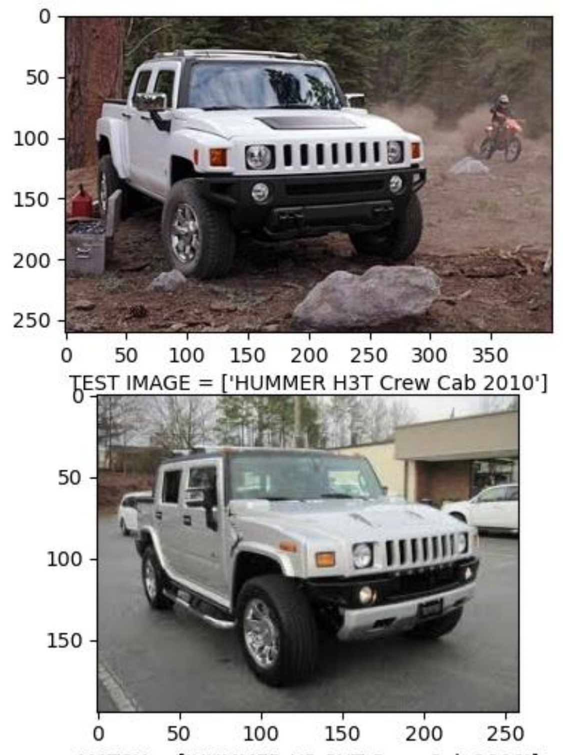
Test image 16


TEST IMAGE = ['Ford GT Coupe 2006']
MATCH = ['Ford GT Coupe 2006']

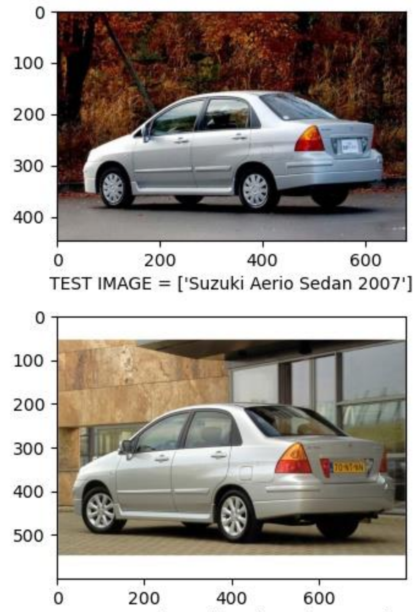
Test image 15


TEST IMAGE = ['Mercedes-Benz SL-Class Coupe 2009']
MATCH = ['BMW 1 Series Convertible 2012']

Test image 8


TEST IMAGE = ['HUMMER H3T Crew Cab 2010']
MATCH = ['HUMMER H2 SUT Crew Cab 2009']

Test image 11

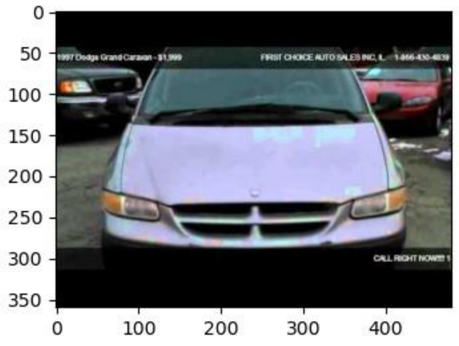

TEST IMAGE = ['Suzuki Aerio Sedan 2007']
MATCH = ['Suzuki Aerio Sedan 2007']

Test image 0

TEST IMAGE = ['Ford Fiesta Sedan 2012']

MATCH = ['Ford Fiesta Sedan 2012']

Test image 9


TEST IMAGE = ['Dodge Caravan Minivan 1997']

MATCH = ['Dodge Caravan Minivan 1997']

Test image 3

TEST IMAGE = ['Bugatti Veyron 16.4 Coupe 2009']

MATCH = ['Audi TT RS Coupe 2012']

Test image 5

TEST IMAGE = ['GMC Savana Van 2012']

MATCH = ['Chevrolet Express Cargo Van 2007']

6. Conclusion

In conclusion, my algorithm was extremely inefficient due to the number of image conversions to a HOG representation and the number of comparisons that the ORB+RANSAC portion of the algorithm had to make. The time is taken and accuracy is prime reasons why a CNN should be considered for image classification rather than my approach. Although my algorithm was inefficient, it was fun and I learned that optimization of code or even a beefier PC could've helped compute this much faster. I also learned that saving your model is very important when running models with thousands of samples of data.

7. Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 13 23:28:28 2021

@Author: R Noah Padilla

@Co-Authors: Olac Fuentes (ORB AND RANSAC PORTION AS WELL AS ONE HOT AND ALL OF UTILS FUNCTIONS)

"""

'''
```

This program is a computer vision/ machine learning algorithm designed to predict a car.

The way the algorithm works -

1. Use a clustering algorithm (HOG's) to seperate the vehicle type (cars/vans vs trucks)
2. Use the ORB + RANSAC algorithm to find the exact match
3. Calculate the accuracy of the algorithm

TODOS:

    [x] - Load and normalize the data from the train and test set (start small)

    [x] - Convert the train and test images into HOG's with an optimal number of bins

    [x] - Cluster the training set to seperate cars/vans and trucks //use 5% of training data

    [x] - Test the cluster with the test data //use 20 images

    [x] - Use ORB and RANSAC to find the make and model

'''

```
import numpy as np

from matplotlib import transforms

import matplotlib.image as mpimg
import matplotlib.pyplot as plt

import os

import time


import scipy.io

from scipy.io import loadmat


from sklearn.metrics import accuracy_score

from skimage import transform as tf

from sklearn.cluster import KMeans

from sklearn.model_selection import train_test_split

from sklearn.neural_network import MLPClassifier



from utils import *


def _onehot(integer_labels):

    """Return matrix whose rows are onehot encodings of integers."""

    n_rows = len(integer_labels)

    n_cols = integer_labels.max() + 1

    onehot = np.zeros((n_rows, n_cols), dtype='uint8')

    onehot[np.arange(n_rows), integer_labels] = 1

    return onehot
```

```
if __name__ == "__main__":

    plt.close("all")

    image_dir_train = '.\\cars_train\\cars_train\\'
    image_dir_test = '.\\cars_test\\cars_test\\'
    image_files_train = os.listdir(image_dir_train)
    image_files_test= os.listdir(image_dir_test)

    cars_metadata=loadmat('.\\devkit\\cars_meta.mat')
    cars_train_annos=loadmat('.\\devkit\\cars_train_annos.mat')
    cars_test_annos=loadmat('.\\devkit\\cars_test_annos.mat')

    cars_meta=[]
    for x in cars_metadata['class_names'][0]:
        cars_meta.append(x)

    cars_classes=pd.DataFrame(cars_meta,columns=['cars_classes_exist_in_data'])

    #Get the train labels - taken from https://www.kaggle.com/vipulgote4/stanford-cars-data-with-
labels-for-tf-operations/notebook
    fname=[[x.flatten()[0] for x in i]  for i in cars_train_annos['annotations'][0]]
    column_list=['bbox_x1','bbox_y1','bbox_x2','bbox_y2','class','fname']
    train_df=pd.DataFrame(fname,columns=column_list)

    train_df['class']=train_df['class'] - 1 ### all values start from zero because above class_classes
df index for classes started from zero hence.



    #Convert labels to np array and debugg

    cars_classes = np.asarray(cars_classes)

    labels = np.array(train_df['class']) # values range from 0 to 195



    '''

    #USED TO DEBUGG TO GET THE TITLE OF CAR USING LABELS ARRAY

    for i,image_file in enumerate(image_files_train[:20]):


        image = mpimg.imread(image_dir_train + image_file)


        fig = plt.figure()

        ax = fig.add_subplot(1, 2, 1)

        imgplot = plt.imshow(image)

        ax.set_title('Train car is a: ' + str(cars_classes[labels[i]]))

    '''


    #print(labels.shape)
    labels = _onehot(labels)
    #print(labels.shape)


    '''
    #USED TO DEBUGG TO GET THE TITLE OF CAR USING LABELS ARRAY
    for i,image_file in enumerate(image_files_train[:20]):


        image = mpimg.imread(image_dir_train + image_file)


        fig = plt.figure()
        ax = fig.add_subplot(1, 2, 1)
        imgplot = plt.imshow(image)
        ax.set_title('Car is a: ' + str(cars_classes[np.argmax(labels[i])]))
    '''
    #---------------------------- LOAD TRAIN DATA AND TRAIN LABELS -------------------------
-
    num_samples = 8144 #8144/2 = 4072
    test_size = .0022 #percentage of total that will be test set


    print("\nTrain data is first", str( int(num_samples - (test_size * num_samples)) ) ,"of",
num_samples,"samples")
    print("\nTest data is last", str( int(test_size * num_samples) ) ,"of", num_samples,"samples")
    #TODO - MAKE SURE LATER ON YOU REMOVE THE LABELED DATA THAT
CORRESPONDED TO THE IMAGES THAT DIDNT HAVE 3 LAYERS
    X = []
    y = []
    start = time.time()
    print("\n> Loading", str(num_samples) , "data samples...")
    for i,image_file in enumerate(image_files_train[:num_samples]):

        image = mpimg.imread(image_dir_train + image_file)

        if len(image.shape) == 3 and (image.shape[0] > 100 and image.shape[1] > 100):
            X.append(np.array(image))
            y.append(np.array(labels[i]))

        if i > 0 and (i+1) % 200 == 0:
            print("Pre-Processed", i+1 , "/", str(num_samples) , "images"  )

    elapsed_time = time.time()-start
    print('Elapsed time preprocessing: {0:.6f} secs'.format(elapsed_time))

    #splt the data
    X_train, X_test, y_train, y_test = X[:len(X) - int(len(X) * test_size)], X[len(X) - int(len(X) *
test_size):] , y[:len(X) - int(len(X) * test_size)] , y[len(X) - int(len(X) * test_size):]
```

```
    '''
    #USED TO DEBUGG TO MAKE SURE TRAIN AND TEST SAMPLES ARE LABELED
CORRECTLY AFTER SPLITTING
    for i,img in enumerate(X_train[:5]):

        fig = plt.figure()
        ax = fig.add_subplot(1, 2, 1)
        imgplot = plt.imshow(img)
        ax.set_title('Train car is a: ' + str(cars_classes[np.argmax(y_train[i])]))

    for i,img in enumerate(X_test[:5]):

        fig = plt.figure()
        ax = fig.add_subplot(1, 2, 1)
        imgplot = plt.imshow(img)
        ax.set_title('Test car is a: ' + str(cars_classes[np.argmax(y_test[i])]))
    '''

    #---------------------------- CONVERT TRAIN AND TEST IMAGES TO HOG ------------------
--------
    start = time.time()
    print("\n> Train set | Image representations -> HOG representations...")
```

```
HOGS_train = []

bins=128

for i,x in enumerate(X_train):



    hog = histogram_of_gradients(x,bins=bins)

    HOGS_train.append(hog)#the training set are now represented using HOG



    if i > 0 and (i+1) % 100 == 0:

        print("Converted", i+1 , "/" , len(X_train) ,  "images")



print("\n> Test set | Image representations -> HOG representations...")

HOGS_test = []

bins=128

for i,x in enumerate(X_test):



    hog = histogram_of_gradients(x,bins=bins)

    HOGS_test.append(hog)#the training set are now represented using HOG



    if i > 0 and (i+1) % 100 == 0:

        print("Converted", i+1 , "/" , len(X_test),  "images")



elapsed_time = time.time()-start

print('\nElapsed time converting both train and test sets to HOG: {0:.6f}
secs'.format(elapsed_time))


    '''
    #------------------------------------------------ MLP ------------------------------------------------
    model = MLPClassifier( activation='tanh' , hidden_layer_sizes=(150,300) , alpha=1e-8,
batch_size=100, learning_rate='constant' , early_stopping=True, verbose=True)


    #------------- Fit the training data - never comment out fit and both predicts
    start = time.time()
    model.fit(np.array(HOGS_train), np.array(y_train))
    elapsed_time = time.time()-start
    print('-Elapsed time training HOG train set: {0:.6f} secs'.format(elapsed_time))



    #Test on practice test set - get a 99% accuracy
    start = time.time()
    pred = model.predict(np.array(HOGS_test))
    elapsed_time = time.time()-start
    print('Elapsed time testing HOG test set: {0:.6f} secs'.format(elapsed_time))
    print('** Accuracy on test set: {0:.6f}'.format(accuracy_score(y_test,pred)))


    for i,p in enumerate(pred[:5]):
        fig, ax = plt.subplots(2,figsize=(8,8))
        fig.suptitle('Test image ' + str(i) + ' assigned is a: ' + str(cars_classes[np.argmax(y_test[i])]))

        ax[0].imshow(X_test[i])

        ax[0].set_xlabel("ACTUAL= " + str(cars_classes[np.argmax(y_test[i])]))

        #ax[1].imshow(p[np.argmax(y_test[i])]])

        ax[1].set_xlabel("PREDICTED" + str(cars_classes[np.argmax(p[i])]))


    '''

    #-------------------------- PLACE EACH IMAGE (represented as HOG) INTO A CLUSTER ----
--------------- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    clusters = 2 #cluster for cars/vans vs trucks

    print("\n** Creating",clusters,"clusters now...**")

    #Create and fit data to 2 clusters (cars/vans, trucks)

    km = KMeans(n_clusters=clusters).fit(HOGS_train)

    print("** Finished grouping data into",clusters,"clusters...**")


    km_l = km.labels_

    #print('\nKMeans assignments/labels')

    #print(km_l)


    km_cc = km.cluster_centers_

    #print('\nFinal centroids:')

    #print(km_cc)
    km_ssd = km.inertia_ # sum of squared distance from each data point to its respective centroid
    #print('\nSSD: ')
    #print(km_ssd)

    #---------------------------------- PREDICT WHAT CLUSTERS THE TEST SAMPLES ARE
IN -------------------
    km_pred = km.predict(HOGS_test)

    pred = np.zeros((196,), dtype=int) #will stack other arrays but the first zero is just a
placeholder - first array doesnt count, it is needed for stacking but will remove when getting
accuracies

    orb = cv2.ORB_create()

    for i,x_test_sample in enumerate(X_test[:]):

        keypoints1, descriptors1 = orb.detectAndCompute(np.asarray(x_test_sample,
np.uint8),mask=None)

        best_p0 = None #stores the best point matches on image 1
        best_p1 = None #stores the best point matches on image 2
        max_feature_matches = 0 #will be used to count the matched features from two images
        matched_vehicle = None # stores the array containing the matched vehicle
    vehicle_name = ''

    for j,x_train_sample_label in enumerate(km_l):

      #if we look at the cluster which x_test_sample defines itself then we can use ORB
+RANSAC to look within cluster to find similar vehicle
      if km_pred[i] == x_train_sample_label:
        #print("x_test_sample: ", x_test_sample.shape, "| X_train[i] shape:", X_train[j].shape)
        #print("km_pred:" , km_pred[i], "| x_train_sample_label: ", x_train_sample_label)

        #--------------------------- USE ORB AND RANSAC TO FIND IDENTICAL
VEHICLE
        keypoints2, descriptors2 = orb.detectAndCompute(np.asarray(X_train[j],
np.uint8),mask=None)

        # Create BFMatcher object
        matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
        matches = matcher.match(descriptors1,descriptors2)

        #find the subest of these matches that can be explained using the ransac | for every
image find the best matching image

        # Extract data from orb objects and matcher
```

```
dist = np.array([m.distance for m in matches])

ind1 = np.array([m.queryIdx for m in matches])

ind2 = np.array([m.trainIdx for m in matches])


#TWEAKED BY NOAH - NEEDED TO CHANGE IF GOING THROUGH A LOOP

k1 = np.array([p.pt for p in keypoints1])

k2 = np.array([p.pt for p in keypoints2])

k1 = k1[ind1]

k2 = k2[ind2]

# keypoints1[i] and keypoints2[i] are a match


#--------------------BEST MATCH POINTS FROM BOTH IMAGES

p0, p1 = select_matches_ransac( k1, k2 )

if (p0.shape[0] > max_feature_matches):

    best_p0 = p0

    best_p1 = p1

    max_feature_matches = p0.shape[0]

    matched_vehicle = X_train[j]


    vehicle_name, vehicle_label = str(cars_classes[np.argmax(y_train[j])]), y_train[j]


if (i+1)%500==0:

    print("Matching test image ", i , "Looked through", i+1 , "images")
```

```
                #add to pred here


        if (i+1)%1==0:
            print("Matched", i+1 , "images")


        #display_correspondences(x_test_sample,matched_vehicle, best_p0, best_p1)
        #print the first 20 matches
        pred = np.vstack((pred,vehicle_label))
        if(i<20):
            fig, ax = plt.subplots(2,figsize=(8,8))
            fig.suptitle('Test image ' + str(i))
            ax[0].imshow(x_test_sample)
            ax[0].set_xlabel("TEST IMAGE = " + str(cars_classes[np.argmax(y_test[i])]))
            ax[1].imshow(matched_vehicle)
            ax[1].set_xlabel("MATCH = " + vehicle_name)


    print('\nAccuracy on test set: {0:.6f}'.format(accuracy_score(y_test,pred[1:]))) #evaluating
ytrue, ypred | first value of pred is a zero array - last value of test
    '''
    #acc = accuracy_score(y_test, y_test)
    #print('\nAccuracy on test set: {0:.6f}'.format(acc)) #evaluating ytrue, ypred
    '''




'''
    num_samples = 5389 #8144/2 = 4072

    test_size = .0035 #percentage of total that will be test set


    Train data is first 5370 of 5389 samples


Test data is last 18 of 5389 samples


> Loading 5389 data samples...

Pre-Processed 200 / 5389 images

Pre-Processed 400 / 5389 images

Pre-Processed 600 / 5389 images

Pre-Processed 800 / 5389 images

Pre-Processed 1000 / 5389 images

Pre-Processed 1200 / 5389 images

Pre-Processed 1400 / 5389 images

Pre-Processed 1600 / 5389 images

Pre-Processed 1800 / 5389 images

Pre-Processed 2000 / 5389 images
```

```
Pre-Processed 2200 / 5389 images

Pre-Processed 2400 / 5389 images

Pre-Processed 2600 / 5389 images

Pre-Processed 2800 / 5389 images

Pre-Processed 3000 / 5389 images

Pre-Processed 3200 / 5389 images

Pre-Processed 3400 / 5389 images

Pre-Processed 3600 / 5389 images

Pre-Processed 3800 / 5389 images

Pre-Processed 4000 / 5389 images

Pre-Processed 4200 / 5389 images

Pre-Processed 4400 / 5389 images

Pre-Processed 4600 / 5389 images

Pre-Processed 4800 / 5389 images

Pre-Processed 5000 / 5389 images

Pre-Processed 5200 / 5389 images

Elapsed time preprocessing: 218.505140 secs


> Train set | Image representations -> HOG representations...

Converted 100 / 5323 images

Converted 200 / 5323 images

Converted 300 / 5323 images

Converted 400 / 5323 images
Converted 500 / 5323 images

Converted 600 / 5323 images

Converted 700 / 5323 images

Converted 800 / 5323 images

Converted 900 / 5323 images

Converted 1000 / 5323 images

Converted 1100 / 5323 images

Converted 1200 / 5323 images

Converted 1300 / 5323 images

Converted 1400 / 5323 images

Converted 1500 / 5323 images

Converted 1600 / 5323 images

Converted 1700 / 5323 images

Converted 1800 / 5323 images

Converted 1900 / 5323 images

Converted 2000 / 5323 images

Converted 2100 / 5323 images

Converted 2200 / 5323 images

Converted 2300 / 5323 images

Converted 2400 / 5323 images

Converted 2500 / 5323 images

Converted 2600 / 5323 images

Converted 2700 / 5323 images
Converted 2800 / 5323 images

Converted 2900 / 5323 images

Converted 3000 / 5323 images

Converted 3100 / 5323 images

Converted 3200 / 5323 images

Converted 3300 / 5323 images

Converted 3400 / 5323 images

Converted 3500 / 5323 images

Converted 3600 / 5323 images

Converted 3700 / 5323 images

Converted 3800 / 5323 images

Converted 3900 / 5323 images

Converted 4000 / 5323 images

Converted 4100 / 5323 images

Converted 4200 / 5323 images

Converted 4300 / 5323 images

Converted 4400 / 5323 images

Converted 4500 / 5323 images

Converted 4600 / 5323 images

Converted 4700 / 5323 images

Converted 4800 / 5323 images

Converted 4900 / 5323 images

Converted 5000 / 5323 images
Converted 5100 / 5323 images

Converted 5200 / 5323 images

Converted 5300 / 5323 images


> Test set | Image representations -> HOG representations...


Elapsed time converting both train and test sets to HOG: 1338.171597 secs


** Creating 2 clusters now...**

** Finished grouping data into 2 clusters...**

Matched 1 images

Matched 2 images

Matched 3 images

Matched 4 images

Matched 5 images

Matched 6 images

Matched 7 images

Matched 8 images

Matched 9 images

Matched 10 images

Matched 11 images

Matched 12 images

Matched 13 images
Matched 14 images

Matched 15 images

Matched 16 images

Matched 17 images

Matched 18 images


Accuracy on test set: 0.222222
```

```
num_samples = 8144 #8144/2 = 4072

test_size = .0022 #percentage of total that will be test set
```

```
Train data is first 8126 of 8144 samples


Test data is last 17 of 8144 samples


> Loading 8144 data samples...

Pre-Processed 200 / 8144 images

Pre-Processed 400 / 8144 images

Pre-Processed 600 / 8144 images

Pre-Processed 800 / 8144 images
```

```
Pre-Processed 1000 / 8144 images

Pre-Processed 1200 / 8144 images

Pre-Processed 1400 / 8144 images

Pre-Processed 1600 / 8144 images

Pre-Processed 1800 / 8144 images

Pre-Processed 2000 / 8144 images

Pre-Processed 2200 / 8144 images

Pre-Processed 2400 / 8144 images

Pre-Processed 2600 / 8144 images

Pre-Processed 2800 / 8144 images

Pre-Processed 3000 / 8144 images

Pre-Processed 3200 / 8144 images

Pre-Processed 3400 / 8144 images

Pre-Processed 3600 / 8144 images

Pre-Processed 3800 / 8144 images

Pre-Processed 4000 / 8144 images

Pre-Processed 4200 / 8144 images

Pre-Processed 4400 / 8144 images

Pre-Processed 4600 / 8144 images

Pre-Processed 4800 / 8144 images

Pre-Processed 5000 / 8144 images

Pre-Processed 5200 / 8144 images

Pre-Processed 5400 / 8144 images
```

```
Pre-Processed 5600 / 8144 images

Pre-Processed 5800 / 8144 images

Pre-Processed 6000 / 8144 images

Pre-Processed 6200 / 8144 images

Pre-Processed 6400 / 8144 images

Pre-Processed 6600 / 8144 images

Pre-Processed 6800 / 8144 images

Pre-Processed 7000 / 8144 images

Pre-Processed 7200 / 8144 images

Pre-Processed 7400 / 8144 images

Pre-Processed 7600 / 8144 images

Pre-Processed 7800 / 8144 images

Pre-Processed 8000 / 8144 images

Elapsed time preprocessing: 368.246758 secs


> Train set | Image representations -> HOG representations...

Converted 100 / 8049 images

Converted 200 / 8049 images

Converted 300 / 8049 images

Converted 400 / 8049 images

Converted 500 / 8049 images

Converted 600 / 8049 images

Converted 700 / 8049 images
Converted 800 / 8049 images

Converted 900 / 8049 images

Converted 1000 / 8049 images

Converted 1100 / 8049 images

Converted 1200 / 8049 images

Converted 1300 / 8049 images

Converted 1400 / 8049 images

Converted 1500 / 8049 images

Converted 1600 / 8049 images

Converted 1700 / 8049 images

Converted 1800 / 8049 images

Converted 1900 / 8049 images

Converted 2000 / 8049 images

Converted 2100 / 8049 images

Converted 2200 / 8049 images

Converted 2300 / 8049 images

Converted 2400 / 8049 images

Converted 2500 / 8049 images

Converted 2600 / 8049 images

Converted 2700 / 8049 images

Converted 2800 / 8049 images

Converted 2900 / 8049 images

Converted 3000 / 8049 images
Converted 3100 / 8049 images

Converted 3200 / 8049 images

Converted 3300 / 8049 images

Converted 3400 / 8049 images

Converted 3500 / 8049 images

Converted 3600 / 8049 images

Converted 3700 / 8049 images

Converted 3800 / 8049 images

Converted 3900 / 8049 images

Converted 4000 / 8049 images

Converted 4100 / 8049 images

Converted 4200 / 8049 images

Converted 4300 / 8049 images

Converted 4400 / 8049 images

Converted 4500 / 8049 images

Converted 4600 / 8049 images

Converted 4700 / 8049 images

Converted 4800 / 8049 images

Converted 4900 / 8049 images

Converted 5000 / 8049 images

Converted 5100 / 8049 images

Converted 5200 / 8049 images

Converted 5300 / 8049 images
Converted 5400 / 8049 images

Converted 5500 / 8049 images

Converted 5600 / 8049 images

Converted 5700 / 8049 images

Converted 5800 / 8049 images

Converted 5900 / 8049 images

Converted 6000 / 8049 images

Converted 6100 / 8049 images

Converted 6200 / 8049 images

Converted 6300 / 8049 images

Converted 6400 / 8049 images

Converted 6500 / 8049 images

Converted 6600 / 8049 images

Converted 6700 / 8049 images

Converted 6800 / 8049 images

Converted 6900 / 8049 images

Converted 7000 / 8049 images

Converted 7100 / 8049 images

Converted 7200 / 8049 images

Converted 7300 / 8049 images

Converted 7400 / 8049 images

Converted 7500 / 8049 images

Converted 7600 / 8049 images
Converted 7700 / 8049 images

Converted 7800 / 8049 images

Converted 7900 / 8049 images

Converted 8000 / 8049 images


> Test set | Image representations -> HOG representations...


Elapsed time converting both train and test sets to HOG: 2237.243820 secs


** Creating 2 clusters now...**

** Finished grouping data into 2 clusters...**

Matched 1 images

Matched 2 images

Matched 3 images

Matched 4 images

Matched 5 images

Matched 6 images

Matched 7 images

Matched 8 images

Matched 9 images

Matched 10 images

Matched 11 images

Matched 12 images
Matched 13 images

Matched 14 images

Matched 15 images

Matched 16 images

Matched 17 images


Accuracy on test set: 0.470588

'''
```