HOG_with_svm_classifier

May 13, 2019

Libraries Import: glob > For Files name reading from directory.
pandas > For Datamanipulation and analysis.
cv2 > called opencv, Image processing and Computer vision tasks.
numpy > Mathametics and Multi-dimensional arrays and matrices handling.
matplotlib > plotting library showing graphs and images
skimage > Image processing and computer vision library.
sklearn > DataScience, Machine Learning techniques library.

[1]:
```
import glob
import pandas as pd  # as means that we use pandas library short form  as pd
import cv2
import numpy as np
from matplotlib import pyplot as plt # matplotlib is big library, we are just␣
 ↪calling pyplot function
                                     # for showing images
from skimage.feature import hog #We are calling only hog

from sklearn.decomposition import PCA # Calling the PCA funtion from sklearn
from sklearn.svm import SVC # # Calling the SVM function from sklearn
from sklearn.externals import joblib # Calling the joblib function from␣
 ↪sklearn, use for model saving
                                      # and loading.
```

Libraries version: Below are the current version of libraries the program is tested on

[2]:
```
# to get the version information we need to call for full library
import matplotlib
import skimage
import sklearn
```

[3]:
```
print ('Pandas Version: ',pd.__version__)
print ('OpenCv Version: ',cv2.__version__)
print ('Numpy Version: ',np.__version__)
print ('Matplotlib Version: ',matplotlib.__version__)
print ('skimage Version: ',skimage.__version__)
print ('sklearn Version: ',sklearn.__version__)
```

```
Pandas Version:  0.24.2
OpenCv Version:  3.3.1
```

```
Numpy Version:  1.16.3
Matplotlib Version:  3.0.3
skimage Version:  0.15.0
sklearn Version:  0.20.3
```

0.0.1 Training Data Annotation Reading:

The Annotations or taging of images are saved in csv file. It contains all the images names, bounding box coordinates respective to class ID. What we are going to do in this step is to read all csv files in each folder of training data. And then merge them and make one. Reason for doing this is to make easiness when training SVM.

Commands Explaination:

1. glob.glob(*.file_extension) >Read all the file depending on the file extension given after .* it will return files path
2. pd.read_csv() >Read the csv file using pandas. As this csv file seprated by ';' thats why we use sep=; to get same sequence column values.
3. main_Training.append() >Append the csv files in pandas as stacking file. Main training is main file and new doc appending on it.

[4]:
```
Training_Images_Directory='dataset/Training' #training dataset directory.
```

[5]:
```
csv_files_training=glob.glob(Training_Images_Directory+'/**/*.
 ↪csv',recursive=True) # recursive true means it will check
                                                                #␣
 ↪folder inside the folder as well.
main_Training=pd.read_csv(csv_files_training[0],sep=';') # reading first csv␣
 ↪and assining it main csv.
for i in range(1,len(csv_files_training)): # for loop iteration from 1 to␣
 ↪number of files, by this way can get the all the files read by csv_files␣
 ↪using glob.
    new_doc=pd.read_csv(csv_files_training[i],sep=';') # reading the new csv␣
 ↪file as new doc
    main_Training=main_Training.append(new_doc, ignore_index=True) # appending␣
 ↪the csv files making a big csv that consists of all the csv files.
main_Training
```

[5]:

	Filename	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId
0	01153_00000.ppm	141	142	12	12	128	130	0
1	01153_00001.ppm	120	123	10	10	109	113	0
2	01153_00002.ppm	105	107	9	9	96	98	0
3	01160_00000.ppm	94	105	8	9	86	95	0
4	01160_00001.ppm	128	139	11	12	117	127	0
5	01160_00002.ppm	110	118	9	10	101	108	0
6	01797_00000.ppm	85	95	7	8	77	87	0
7	01797_00001.ppm	180	193	15	16	164	177	0
8	01797_00002.ppm	121	133	10	11	110	121	0

9	01798_00000.ppm	81	88	7	7	74	81	0
10	01798_00001.ppm	240	249	20	21	220	227	0
11	01798_00002.ppm	136	146	11	12	124	134	0
12	01799_00000.ppm	253	263	21	22	232	241	0
13	01799_00001.ppm	139	151	12	13	127	137	0
14	01799_00002.ppm	99	110	8	9	91	100	0
15	00025_00000.ppm	57	61	5	5	52	56	1
16	00025_00001.ppm	82	88	7	7	74	81	1
17	00025_00002.ppm	107	118	9	10	98	108	1
18	00029_00000.ppm	57	61	5	5	51	55	1
19	00029_00001.ppm	73	79	6	7	67	72	1
20	00029_00002.ppm	128	75	11	6	116	69	1
21	00079_00000.ppm	100	112	8	9	91	103	1
22	00079_00001.ppm	132	151	11	13	120	137	1
23	00079_00002.ppm	264	167	22	14	242	153	1
24	00136_00000.ppm	80	84	7	7	73	76	1
25	00136_00001.ppm	178	176	15	15	163	160	1
26	00136_00002.ppm	51	54	5	5	46	48	1
27	00143_00000.ppm	49	50	5	5	44	44	1
28	00143_00001.ppm	63	67	5	6	57	61	1
29	00143_00002.ppm	144	160	12	13	131	146	1
...	...	...	...	...	...	...	...	...
4561	01910_00000.ppm	70	39	6	5	63	34	61
4562	01910_00001.ppm	37	37	5	5	31	31	61
4563	01910_00002.ppm	123	122	10	10	112	111	61
4564	01911_00000.ppm	68	71	6	6	61	65	61
4565	01911_00001.ppm	57	58	5	5	52	52	61
4566	01911_00002.ppm	47	47	5	5	42	42	61
4567	01916_00000.ppm	62	57	5	5	57	52	61
4568	01916_00001.ppm	47	48	5	5	42	43	61
4569	01916_00002.ppm	108	55	9	5	99	50	61
4570	01930_00000.ppm	128	130	11	11	117	118	61
4571	01930_00001.ppm	83	84	7	7	76	77	61
4572	01930_00002.ppm	60	58	5	5	55	53	61
4573	01933_00000.ppm	165	86	14	7	150	79	61
4574	01933_00001.ppm	86	91	7	8	79	83	61
4575	01933_00002.ppm	60	58	5	5	55	52	61
4576	01935_00000.ppm	81	43	7	5	73	38	61
4577	01935_00001.ppm	81	56	7	5	74	50	61
4578	01935_00002.ppm	82	36	7	5	74	31	61
4579	01955_00000.ppm	132	134	11	11	120	122	61
4580	01955_00001.ppm	82	80	7	7	74	73	61
4581	01955_00002.ppm	59	61	5	5	54	56	61
4582	01956_00000.ppm	139	141	12	12	126	128	61
4583	01956_00001.ppm	87	86	7	7	80	78	61
4584	01956_00002.ppm	62	61	5	5	56	55	61
4585	01957_00000.ppm	121	119	10	10	111	108	61

```
4586  01957_00001.ppm     75      74       6       6      69      67       61
4587  01957_00002.ppm     55      53       5       5      49      48       61
4588  01959_00000.ppm    111      67       9       6     102      61       61
4589  01959_00001.ppm     39      39       5       5      34      34       61
4590  01959_00002.ppm    110      56       9       5     101      50       61

[4591 rows x 8 columns]
```

0.0.2 Visulization of Classes Images:

For Visulization of seperate classes images, we first need to select one images from the training data folder. For this we choose first image read by main_training dataset. and then make a new pandas Dataframe that consists of one images of each class, we assigned it One_Example. You can look on it then observe the classID. After that I use matplotlib to display the images one by one in a stack manner.

Commands Explaination:

1. if main_Training.values[i,-1] not in sss > sss is the list, which saves values of classid. We are iterating the i so that we can read each row of upper diffine main_training dataset of csv's. main_Training.values[i,-1] shows the last column of that row that is class id. so what are we doing if the classid is not in sss then only run it otherwise not.
2. .append() > Append use for list to append some new values.
3. pd.DataFrame > Making a new pandas_data frame from list or array. as we are storing each class one row in oneexample list then we make dataframe that shown below.
4. .head() > Pandas dataframe only head lines show hat means only start line showing.
5. plt.figure() > Initlizing the new figure using plt. and figure size is [x,y] respective.
6. plt.subplots_adjust() > plt adjusting the subplot parameters in this case we are adjusting the word space between two subplots.
7. Pandas['column_name'] > This is used for reading the specific column in pandas.
8. cv2.imread(img_path) > Reading the image using opencv.
9. img[y1:y2,x1:x2] > Used for taking the specific part of image or croping it. we use it in 8 line of below ploting box
10. cv2.cvtColor() > Changing the color type like(hsv,bgr,rgb). In default opencv read bgr color for visulizing the right color we need to convert it RGB channel colors.
11. plt.subplot(rows,column,place_numbers) > plt subplot give easy way to plot multiple images to one place.
12. plt.imshow(image) > displaying the image.
13. cv2.imwrite(filename.png,image) > Saving the image.

[6]:
```
sss=[] # making a new list.
oneexample=[]
for i in range(len(main_Training.values)): # iterating 0-len(main_Training.
 ↪values) that is training dataset lenght.
    if main_Training.values[i,-1] not in sss:
        oneexample.append(main_Training.values[i,:]) #appending the␣
 ↪main_Training dataset row number i.
        sss.append(main_Training.values[i,-1]) # appending the class id of row␣
 ↪number i

# Making a new pandas dataframe from oneexample list. and giving it columns␣
 ↪names.
One_Example=pd.DataFrame(oneexample,columns=['Filename', 'Width', 'Height',␣
 ↪'Roi.X1', 'Roi.Y1', 'Roi.X2', 'Roi.Y2', 'ClassId'])
One_Example.head()
```

[6]:

	Filename	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId
0	01153_00000.ppm	141	142	12	12	128	130	0
1	00025_00000.ppm	57	61	5	5	52	56	1
2	00270_00000.ppm	59	62	5	5	53	57	2
3	00207_00000.ppm	147	160	12	13	135	146	3
4	00145_00000.ppm	46	52	5	5	41	46	4

[7]:
```
plt.figure(figsize=[14,25]) # set image size
plt.subplots_adjust(wspace = 0.2)# set distance between the subplots

i = 0 # i is the index use for subploting the means at what number subplot will␣
 ↪be plot.
for i in range(len(One_Example)): # range depends on number of classes
    # we are getting the image path from csv_files name thats is dataset/
 ↪Training\\00000\\GT-00000.csv
    # then spliting it at GT and we have dataset/Training\\00000\\ after that␣
 ↪adding the name of one
    # example that we are dealing like dataset/Testing\\00001\\01983_00002.ppm
    img_path=csv_files_training[One_Example['ClassId'][i]].
 ↪split('GT')[0]+One_Example['Filename'][i]
    img = cv2.imread(img_path) # opencv use for reading image.
    # croping the image based on the coordinates given us by csv files
    crop_image=img[One_Example['Roi.Y1'][i]:One_Example['Roi.
 ↪X2'][i],One_Example['Roi.X1'][i]:One_Example['Roi.Y2'][i]]
    #converting the color RGB so that we can actually view it.
    crop_image=cv2.cvtColor(crop_image,cv2.COLOR_BGR2RGB)
    plt.subplot(13,5,i+1)
    i+=1
    imgplot = plt.imshow(crop_image)
plt.show() # plot showing at the end.
```


STOP
70
km

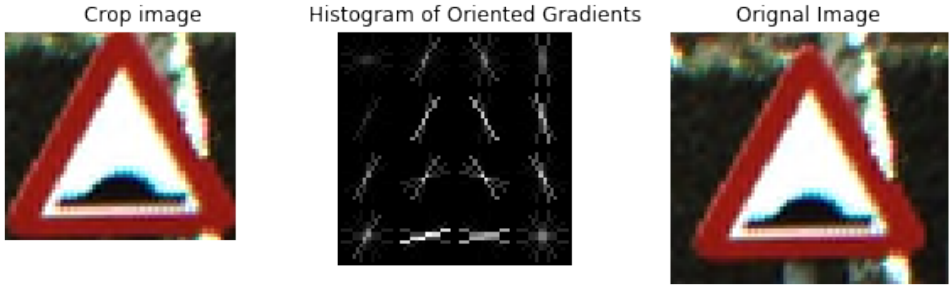
0.0.3 Saving the classes Images:

We need to save each class images so that we can use it as testing time drawing it adjasent to detected image.class image name is same classID.png. All images will be saved in 'classes_images' folder. if 'classes_images' directory doesn't exist then you need to make it.

[8]:
```
for i in range(len(One_Example)): # range depends on number of classes
    # we are getting the image path from csv_files name thats is dataset/
↪Training\\00000\\GT-00000.csv
    # then spliting it at GT and we have dataset/Training\\00000\\ after that␣
↪adding the name of one
    # example that we are dealing like dataset/Testing\\00001\\01983_00002.ppm
    img_path=csv_files_training[One_Example['ClassId'][i]].
↪split('GT')[0]+One_Example['Filename'][i]
    img = cv2.imread(img_path) # opencv use for reading image.
    # croping the image based on the coordinates given us by csv files
    crop_image=img[One_Example['Roi.Y1'][i]:One_Example['Roi.
↪X2'][i],One_Example['Roi.X1'][i]:One_Example['Roi.Y2'][i]]
    cv2.imwrite('classes_images/'+str(One_Example['ClassId'][i])+'.
↪png',crop_image) # use for saving the image.
```

0.0.4 Histogram of oriented gradients (HOG) Features from Images

HOG decomposes an image into small squared cells, computes an histogram of oriented gradients in each cell, normalizes the result using a block-wise pattern, and return a descriptor for each cell. We Stack the cells into a squared image region used it as an image window descriptor for object detection, by SVM Classifier.



HoG Example

We are using HoG from skimage library built in function. skimage Compute a Histogram of Oriented Gradients (HOG) by

1. computing the gradient image in x and y
2. computing gradient histograms

3. normalising across blocks

4. flattening into a feature vector

At first computes order image gradients. These capture contour, silhouette and some texture information, while providing further resistance to illumination variations. The locally dominant color channel is used, which provides color invariance to a large extent. Variant methods may also include second order image derivatives, which act as primitive bar detectors - a useful feature for capturing, e.g. bar like structures in bicycles and limbs in humans.

The third stage aims to produce an encoding that is sensitive to local image content while remaining resistant to small changes in pose or appearance. The adopted method pools gradient orientation information locally in the same way as the SIFT 2 feature. The image window is divided into small spatial regions, called "cells". For each cell we accumulate a local 1-D histogram of gradient or edge orientations over all the pixels in the cell. This combined cell-level 1-D histogram forms the basic "orientation histogram" representation. Each orientation histogram divides the gradient angle range into a fixed number of predetermined bins. The gradient magnitudes of the pixels in the cell are used to vote into the orientation histogram.

The fourth stage computes normalisation, which takes local groups of cells and contrast normalises their overall responses before passing to next stage. Normalisation introduces better invariance to illumination, shadowing, and edge contrast. It is performed by accumulating a measure of local histogram "energy" over local groups of cells that we call "blocks". The result is used to normalise each cell in the block. Typically each individual cell is shared between several blocks, but its normalisations are block dependent and thus different. The cell thus appears several times in the final output vector with different normalisations. This may seem redundant but it improves the performance. We refer to the normalised block descriptors as Histogram of Oriented Gradient (HOG) descriptors.

The final step collects the HOG descriptors from all blocks of a dense overlapping grid of blocks covering the detection window into a combined feature vector for use in the window classifier.[0]

Command Explaination:

1. cv2.resize(input_image, (new width,new hight)) > For resizing the image to new dimension, in this case we choose

2. hog(crop_image, orientations=8) > Apply hog features on crop image. in 8 different orientations gradients. in our case it returns the 2592 size array.

[9]:
```
def images_to_hog(main,csv_files): # function defining that can be call for␣
↪both test and training
    Features=[]
    Labels=[]
    for i in range(0,len(main)): #len(main)
        # we are getting the image path from csv_files name thats is dataset/
↪Training\\00000\\GT-00000.csv
        # then spliting it at GT and we have dataset/Training\\00000\\ after␣
↪that adding the name of one
        # example that we are dealing like dataset/Testing\\00001\\01983_00002.
↪ppm
```

```
        img_path=csv_files[main['ClassId'][i]].
↪split('GT')[0]+main['Filename'][i]
        img = cv2.imread(img_path) # opencv use for reading image.
        # croping the image based on the coordinates given us by csv files
        crop_image=img[main['Roi.Y1'][i]:main['Roi.X2'][i],main['Roi.X1'][i]:
↪main['Roi.Y2'][i]]
        crop_image=cv2.resize(crop_image, (64, 64)) #Resize the image to 64*64.
        # Apply Hog from skimage library it takes image as crop image.Number of␣
↪orientation bins that gradient
        # need to calculate.
        descriptor = hog(crop_image, orientations=8)
        Features.append(descriptor)#hog features saving
        Labels.append(main['ClassId'][i])#class id saving

    Features=np.array(Features)# converting to numpy array.
    Labels=np.array(Labels)
    return Features,Labels
```

[10]:
```
Features_Training,Labels_Training=images_to_hog(main_Training,csv_files_training)␣
↪# giving values to images_to_hog function
print ('Training HOG output Features shape : ',Features_Training.shape)
print ('Training HOG output Labels shape: ',Labels_Training.shape)
```

```
Training HOG output Features shape :  (4591, 2592)
Training HOG output Labels shape:  (4591,)
```

0.0.5 Test Data Annotation Reading:

Reading all the test csv files and merging them as same as we done in training part.

[11]:
```
Test_Images_Directory='dataset/Testing'
```

[12]:
```
csv_files_Testing=glob.glob(Test_Images_Directory+'/**/*.csv',recursive=True) #␣
↪recursive true means it will check
                                                                              #␣
↪folder inside the folder as well.
main_Testing=pd.read_csv(csv_files_Testing[0],sep=';') # reading first csv and␣
↪assining it main csv.
# for loop iteration from 1 to number of files, by this way can get the all the␣
↪files read by csv_files using glob.
for i in range(1,len(csv_files_Testing)):
    new_doc=pd.read_csv(csv_files_Testing[i],sep=';')# reading the new csv file␣
↪as new doc
    # appending the csv files making a big csv that consists of all the csv␣
↪files.
    main_Testing=main_Testing.append(new_doc, ignore_index=True)
main_Testing
```

[12]:

	Filename	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId
0	00017_00000.ppm	107	108	9	9	98	99	0
1	00017_00001.ppm	94	98	8	8	86	89	0
2	00017_00002.ppm	100	106	8	9	91	96	0
3	00021_00000.ppm	57	61	5	5	51	55	0
4	00021_00001.ppm	59	62	5	5	53	56	0
5	00021_00002.ppm	60	67	5	6	55	60	0
6	01983_00000.ppm	105	110	9	9	95	100	1
7	01983_00001.ppm	216	213	18	18	197	195	1
8	01983_00002.ppm	69	76	6	6	63	69	1
9	01987_00000.ppm	58	63	5	5	53	57	1
10	01987_00001.ppm	85	97	7	8	78	89	1
11	01987_00002.ppm	157	176	13	15	143	161	1
12	01988_00000.ppm	191	216	16	18	175	197	1
13	01988_00001.ppm	92	103	8	9	83	94	1
14	01988_00002.ppm	62	68	5	6	56	62	1
15	01991_00000.ppm	57	60	5	5	51	55	1
16	01991_00001.ppm	86	94	7	8	79	85	1
17	01991_00002.ppm	193	189	16	16	177	173	1
18	02220_00000.ppm	151	156	13	13	138	143	1
19	02220_00001.ppm	62	69	5	6	56	63	1
20	02220_00002.ppm	44	46	5	5	39	40	1
21	00398_00000.ppm	212	219	18	18	193	201	1
22	00398_00001.ppm	119	129	10	11	108	118	1
23	00398_00002.ppm	152	163	13	14	139	149	1
24	00414_00000.ppm	92	96	8	8	84	88	1
25	00414_00001.ppm	74	80	6	7	68	72	1
26	00414_00002.ppm	63	70	5	6	57	63	1
27	00424_00000.ppm	124	131	10	11	114	120	1
28	00424_00001.ppm	99	107	8	9	90	97	1
29	00424_00002.ppm	82	87	7	7	75	79	1
...	...	...	...	...	...	...	...	...
2504	00596_00000.ppm	131	107	11	9	119	98	61
2505	00596_00001.ppm	128	91	11	8	116	83	61
2506	00596_00002.ppm	117	64	10	5	106	58	61
2507	00602_00000.ppm	56	61	5	5	51	56	61
2508	00602_00001.ppm	50	53	5	5	44	47	61
2509	00602_00002.ppm	45	46	5	5	39	40	61
2510	00606_00000.ppm	111	54	9	5	102	48	61
2511	00606_00001.ppm	58	57	5	5	53	52	61
2512	00606_00002.ppm	48	48	5	5	43	42	61
2513	00611_00000.ppm	52	36	5	5	46	30	61
2514	00611_00001.ppm	83	46	7	5	75	41	61
2515	00611_00002.ppm	39	43	5	5	34	38	61
2516	00612_00000.ppm	42	40	5	5	37	35	61
2517	00612_00001.ppm	62	47	5	5	56	41	61
2518	00612_00002.ppm	64	38	5	5	58	32	61

```
2519  00615_00000.ppm   127     94     11      8    115     85      61
2520  00615_00001.ppm   121     80     10      7    110     72      61
2521  00615_00002.ppm    79     82      7      7     72     74      61
2522  00619_00000.ppm    39     38      5      5     33     33      61
2523  00619_00001.ppm    72     51      6      5     65     45      61
2524  00619_00002.ppm    72     43      6      5     66     37      61
2525  00701_00000.ppm   144    146     12     12    131    133      61
2526  00701_00001.ppm   229    236     19     20    209    216      61
2527  00701_00002.ppm   175    179     15     15    160    164      61
2528  00713_00000.ppm    94     86      8      7     86     78      61
2529  00713_00001.ppm   107     96      9      8     97     88      61
2530  00713_00002.ppm   179     85     15      7    164     78      61
2531  00718_00000.ppm    67     70      6      6     61     63      61
2532  00718_00001.ppm    57     58      5      5     51     53      61
2533  00718_00002.ppm    45     45      5      5     39     39      61

[2534 rows x 8 columns]
```

[13]:
```
Features_Testing,Labels_Testing=images_to_hog(main_Testing,csv_files_Testing) #␣
 ↪giving values to images_to_hog function
print ('Testing HOG output Features shape : ',Features_Testing.shape)
print ('Testing HOG output Labels shape: ',Labels_Testing.shape)
```

```
Testing HOG output Features shape :  (2534, 2592)
Testing HOG output Labels shape:  (2534,)
```

0.0.6 PCA on HOG Features

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process.

In our case Our Features output from HOG length is 2592, thats take time when we calculate the SVM and model try to overfit. so thats why we apply pca to reduce 2592 components to 100. [1]

Command Explaination:

1. PCA(n_components=n) > Calling the function from sklearn, with number of components are fixed to 100.
2. pca.fit_transform(Dataset) > It has two parts, at first it Fit the model on dataset, remembar the fitting weights and parameters then apply the dimensionality reduction to the same dataset as transform.
3. pca.transform(new_Dataset) > The same trained PCA can be applied to new dataset using transform command on that dataset.

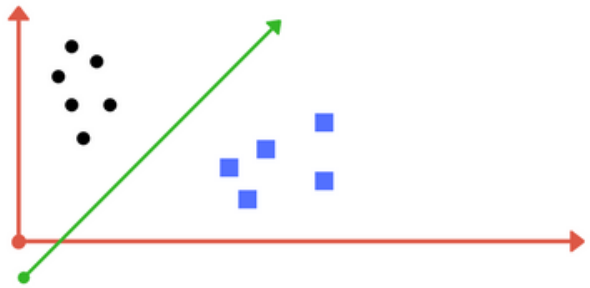
[14]:
```
# Applying PCA
pca = PCA(n_components = 100)
X_train = pca.fit_transform(Features_Training)
X_test = pca.transform(Features_Testing)

print ('New Train Dataset shape after PCA: ',X_train.shape)
print ('New Test Dataset shape after PCA: ',X_test.shape)
```

```
New Train Dataset shape after PCA:  (4591, 100)
New Test Dataset shape after PCA:  (2534, 100)
```

0.0.7 SVM Classifier

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

One Example is shown in below pictures where you can see the data to divide into two dimentional space.



SVM 2 dimentional

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role. The kernel functions are used to map the original dataset (linear/nonlinear) into a higher dimensional space with view to making it linear dataset. The linear, polynomial and RBF or Gaussian kernel are simply different in case of making the hyperplane decision boundary between the classes. Usually linear and polynomial kernels are less time consuming and provides less accuracy than the rbf or Gaussian kernels.

Intuitively, the gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

Gaussian RBF(Radial Basis Function) is another popular Kernel method used in SVM models for more. RBF kernel is a function whose value depends on the distance from the origin or from some point. Gaussian Kernel is of the following format;

$$K(X_1, X_2) = exponent(-\gamma ||X_1 - X_2||^2)$$

Guassian Kernal Format

||X1—X2|| = Euclidean distance between X1 & X2 Using the distance in the original space we calculate the dot product (similarity) of X1 & X2.

Note: similarity is the angular distance between two points.

Parameters:

1. C: Inverse of the strength of regularization. Behavior: As the value of 'c' increases the model gets overfits.

   As the value of 'c' decreases the model underfits.We are using default C=1.0

2. : Gamma (used only for RBF kernel)

   Behavior: As the value of ' ' increases the model gets overfits.

   As the value of ' ' decreases the model underfits. gamma='scale' uses 1 / (n_features * X.var()) as value of gamma/

[15]:
```
# Fitting classifier to the Training set
classifier=SVC(kernel='rbf',gamma='scale') # Calling the function SVC to␣
 ↪implement SVM
classifier.fit(X_train,Labels_Training) # Training the Classifier on Train date

print ('SVM Mean Accuracy of Training dataset: ',classifier.
 ↪score(X_train,Labels_Training))
print ('SVM Mean Accuracy of Test dataset: ',classifier.
 ↪score(X_test,Labels_Testing))
```

```
SVM Mean Accuracy of Training dataset:  0.9934654759311696
SVM Mean Accuracy of Test dataset:  0.9629044988161011
```

[16]:
```
# Saving the model. this model will be used when we test on on new data rather␣
 ↪then train each time.
joblib.dump(pca, 'pca.pkl') # joblib.dump Persist an arbitrary Python object␣
 ↪into one file
joblib.dump(classifier, 'svm.pkl')
```

[16]:
```
['svm.pkl']
```

0.0.8 Model Testing on Image:

Lets test the model on some image and implement all the code for one image, the code will first load the image apply HOG then apply SVM and predict the class and show the ground truth image.

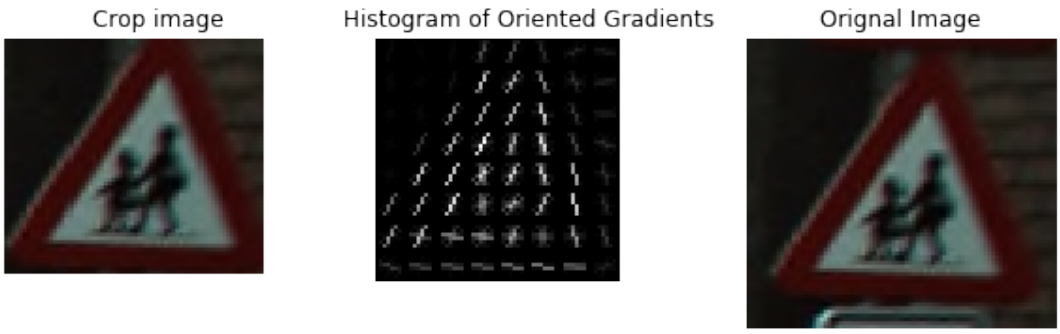
[18]:
```
# Loading the mode into same name
pca = joblib.load('pca.pkl')
classifier = joblib.load('svm.pkl')
```

[19]:
```
image_number=100 # select the image number from test dataset and see its result.
```

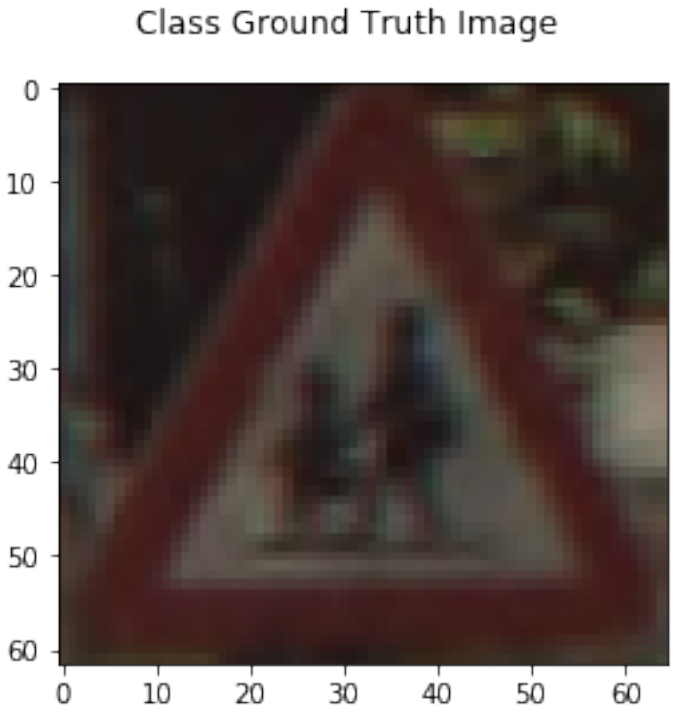
[20]:
```
from skimage.exposure import exposure #for displaying th hog image.
img_path=csv_files_Testing[main_Testing['ClassId'][image_number]].
 ↪split('GT')[0]+main_Testing['Filename'][image_number]
print ('Reading Image from Path: ',img_path)
img = cv2.imread(img_path)
crop_image=img[main_Testing['Roi.Y1'][image_number]:main_Testing['Roi.
 ↪X2'][image_number],main_Testing['Roi.X1'][image_number]:main_Testing['Roi.
 ↪Y2'][image_number]]

crop_image0=cv2.resize(crop_image, (64, 64))
descriptor,imagehog = hog(crop_image0, orientations=8, visualize=True)
descriptor_pca=pca.transform(descriptor.reshape(1,-1))

# Initilize the 3 axis so that we can plot side by side
fig, (ax1, ax2,ax3) = plt.subplots(1, 3, figsize=(10, 10), sharex=True,␣
 ↪sharey=True)

#ploting crop image
ax1.axis('off')
ax1.imshow(cv2.cvtColor(crop_image,cv2.COLOR_BGR2RGB), cmap=plt.cm.gray)
ax1.set_title('Crop image')

# Rescale histogram for better display,Return image after stretching or␣
 ↪shrinking its intensity levels
hog_image_rescaled = exposure.rescale_intensity(imagehog, in_range=(0, 10))
#ploting Hog image
ax2.axis('off')
ax2.imshow(imagehog, cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
#ploting Orignal image
ax3.axis('off')
ax3.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB), cmap=plt.cm.gray)
ax3.set_title('Orignal Image')
plt.show()
# class predition of image using SVM
Predicted_Class=classifier.predict(descriptor_pca)[0]
print ('Predicted Class: ',Predicted_Class)

ground_truth_image=cv2.imread('classes_images/'+str(Predicted_Class)+'.png')

fig = plt.figure()
plt.imshow(cv2.cvtColor(ground_truth_image,cv2.COLOR_BGR2RGB), cmap=plt.cm.gray)
```

```
fig.suptitle('Class Ground Truth Image')
plt.show()
```

```
Reading Image from Path:  dataset/Testing\00007\02337_00000.ppm
```




```
Predicted Class:  7
```



[ ]: