Project Report

On

Real-world Image Classification Problem

Submitted to: Prof. Richard Souvenir
Dept. of Computer Science
CCI, UNCC

Submitted by: Gaurav Pant
800935199

Description

Implementation of problem of real world image classification where the goal is to classify the images into one of the 8 classes.,

- Bathroom
- Guestroom
- Pool
- Gym
- Restaurant
- Lobby
- Aerial View
- Business Center

We are provided with training examples with (somewhat noisy) ground truth labels and unlabeled testing images. The classification model should be able to identify the images for the new unseen test data accurately.

Approach

All image classification algorithms are based on the assumption that an image depicts one or more features and that each of these features belongs to one of several distinct and exclusive classes. The classes may be identified by supervised machine learning or by using the u*nsupervised classification* into sets of prototype classes. For our problem we will be using a supervised machine learning approach to classify the images. There are various image classification algorithms with their own set of pros and cons

Most of the image classification problem comprises of the below steps.

1. Feature Selection and Extraction
2. Building a Classifier
3. Prediction of the test Data using the classifier

So we have the above steps to build our image classification algorithm. At the beginning we took the approach of understanding the image classification libraries and how does they work. The majority of the image classification algorithm and models for python are available in the **sklearn** library. Then next phase is to decide on tuning the parameter supplied to the inbuilt methods for image classification. For eg. to understand the HOG(Histogram of Oriented Gradients) we have to first understand that what all arguments this function take as a part of the function definition and what it returns after the execution is over. Also I read some of the papers and journals to understand the advantage of one image classification algorithm over the other and which algorithm is suited for object classification

Initially we tried picking a subset of the images and performed a conversion of image to a normalized form of 256*256 shape. This will be a preprocessed image.

Naïve Approach:

Experimental Approach: First with a simple code of Image Histogram which is a very basic method of calculating a histogram and reshaping it to the 256, 256 size array. This approach doesn’t focus much on the key point of the image and therefore it won’t be the best approach for doing the feature extraction.

We switched to the other approach of generating HOG (Histogram of Oriented Gradient) for the feature extraction discussed later on the document.

We will be broadly discussing about the steps of the algorithm mentioned above.

1. Feature Selection and Extraction:

This step involves identifying the key feature for an image and normalizing it.We have taken the approach of HOG (Histogram of Oriented Gradient) for the feature extraction phase. HOG approach is used for Object Detection which will be the first step for the image classification. **HOG** is a feature descriptor technique used for object detection. It counts occurrences of gradient orientation in localized portions for an image. The main thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is taken. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing.

This comprises of below steps.

Generate hog.

1. Normalization of the Image
2. Computation of Gradient Histogram
3. Normalizing across blocks
4. Flattening into a feature Vector

Advantage of using HOG over other feature descriptor algorithms (SIFT etc.)

Most of the descriptors algorithm function the same and can frequently be used interchangeably. However, HOG also does contrast normalization over small overlapping cells and that gave a big boost in object class detection performance when used with SVMs. This was the primarily reason for selecting HOG over other feature descriptor algorithms.

HOG is computed for an entire image by dividing the image into smaller cells and summing up the gradients over every pixel within each cell in an image. HoGs are used to classify patches using classifiers such as SVM's.

Source: skimage, opencv2 library

Sample Results by plotting HOG:

Screen Shot of the HOG Generated from the train images:

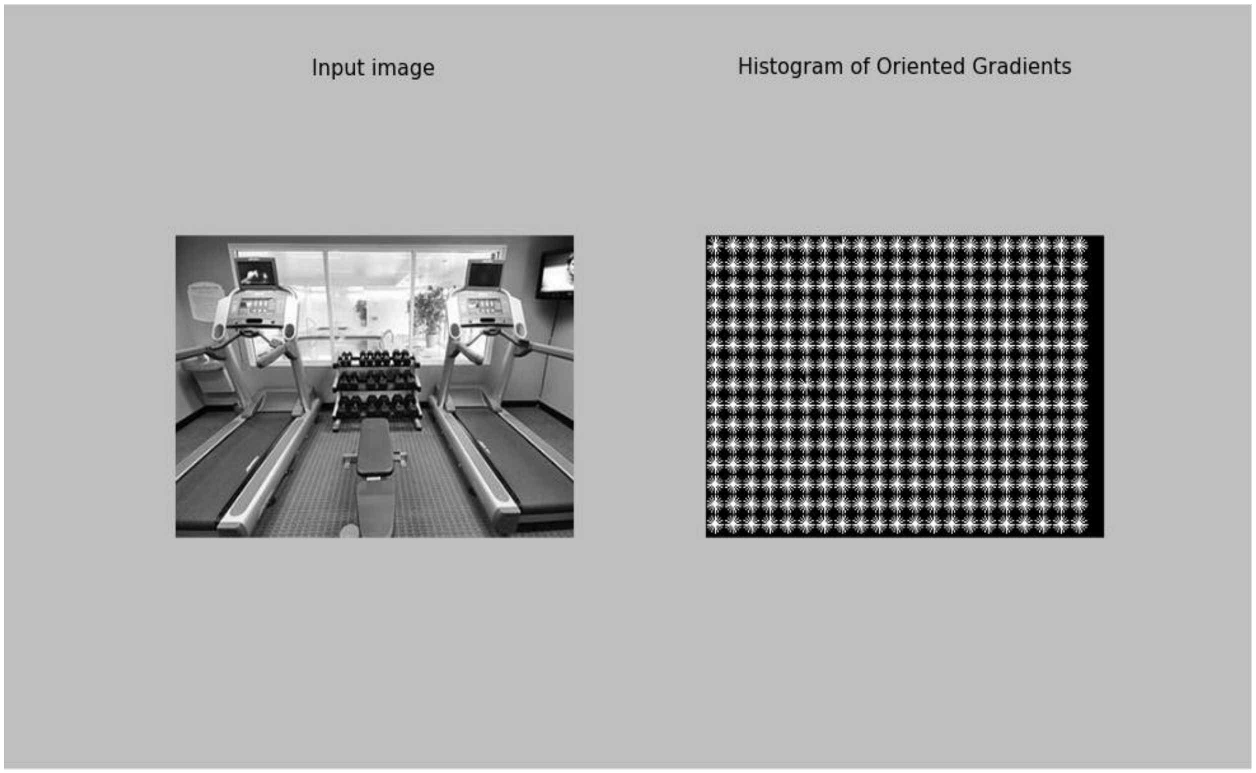

Figure 1

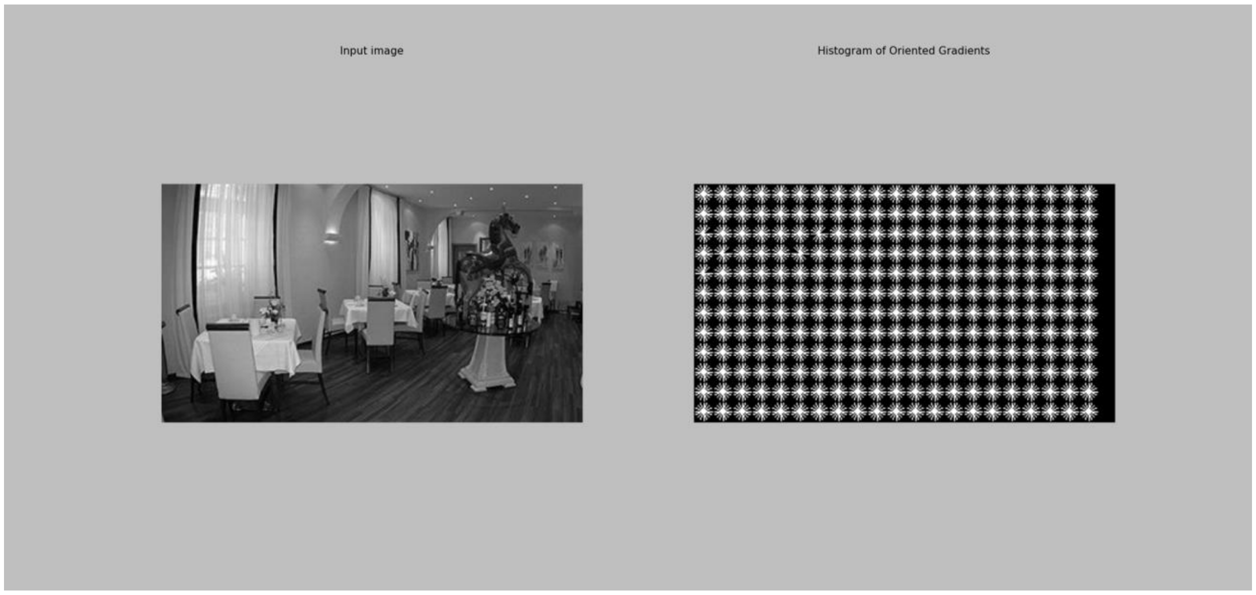

Figure 2

2. Building a Classifier

The next step in image classification using histogram of oriented Gradient descriptors is to feed the descriptors into some classification system based on supervised learning. The support vector machine (SVM) classifier which looks for an optimal hyperplane as a decision function. Once trained on images containing some particular object, the SVM classifier can make decisions regarding the presence of an object, such as a human, in additional test images.
Post the feature extraction is done then the next phase is to generate a classifier model with the set of trained images. Once the classifier is built then the prediction of the labels are done by using the generated classifier. There are various classifiers like **KNN, KMeanCentroid, SVM** are available in sklearn library.
After doing some testing and experimentation and based on the accuracy of the model, we finalized on the SVM model for the classification (results shown in the experiment section). We are using sklearn library package which contains svm classifier. Below is the function available from the library .

SVC(kernel='linear', C=C, probability=True)

Output from the python console for SVM with HOG

```
C:\Anaconda2\python.exe C:/Users/Gaurav/Desktop/machine_learning/projectwork/hog_final.py
----GenerateData done
----Prediction done
Predicted model accuracy with cross validation set: 0.78
----Prediction of Test data done
----Generation of CSV done
```

Output from the python console for KNN with HOG

```
C:\Anaconda2\python.exe C:/Users/Gaurav/Desktop/machine_learning/projectwork/hog_final.py
----GenerateData done
----Prediction done
Predicted model accuracy with cross validation set: 0.39
----Prediction of Test data done
----Generation of CSV done
```

The multiclass support is handled according to a one-vs-one scheme. We used sklearn **onevsrest** classifier and the C factor is set to 1.

svm.SVC(kernel='linear', C=C, probability=True)

Parameter:

C specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used. We did vary the value of C but the best is when we keep the value of C as 1.

Probabilty=true will allow us to do a soft classification on the labels.

3. Prediction of the test data using the classifier

Now once we have the classifier generated the next step involves the prediction of the labels for the test data. We also calculated the accuracy of the model using a cross validation set where the ~10% train images are used to test the classifier. All the test images are first converted using HOG and then passed through the SVM classifier generated above. We are using SVM's predict probability version for the prediction of the labels and the probability depicts the feature matching of an image with one of the 8 labels. Summing up the probability for each image across the 8 labels will be 1. All of the test images will be classified using the above classifier and then will be appended to a CSV list which will be used for uploading to Kaggle.

Algorithm Overview

This section describes the detailed step by step process of using the above approach in generating the code for the classification and the various method/functions descriptions used from the sklearn library.

- The first step is preprocessing of the data stored in the csv file and mapping the images to generate labels. As per the train.csv file we generated the labels Y from a number ranging from 1 to 8 to classify the images. 35k images are chosen for training the model and rest of the images are chosen for testing purpose for doing cross validation.
- Now once we have the image name and the corresponding label then we will generate a HOG for each image which will give us the actual X_Train data in a numpy and the concatenation is done for all the images.Below is how the ***hog*** function from the sklearn library works.

  HOG for the image return us a 1D (flattened) array. The hog method returns a 1D (flattened) array for each image after all the image are resized to a same format. Below is the sample format of the 1D numpy array for an image.

```
[ 0.46712927  0.04519766  0.03162402  0.01085104  0.00838655  0.01231392

  0.03215583  0.39234171  0.34961413  0.05080925  0.03446463  0.00457359

  0.03036342  0.04247109  0.1187972   0.36890669  0.39906714  0.19417252

  0.04990075  0.04498568  0.02798087  0.02503419  0.08005188  0.17880695

  0.31388812  0.04054082  0.00736121  0.00654988  0.07776435  0.056068340 ]
```

  This is a skimmed array (shown for brevity) and the normal descriptor array will be of the dimension (2048L,). We will gather the HOG descriptor for each image and will be storing in the numpy array. The function generate_hog in the code will return a numpy array of the all the images from the train images. Once the descriptors are calculated for each image by the hog method, these are concatenate and passed to build a SVM classifier to train a model.

- As discussed before the next phase after the preprocessing and generation of HOG is done for all the images then this will be supplied to the SVM classifier. The output from above will be treated as an input to the SVM classifier.

  Reason for selecting SVM model.

  Experimentation was done initially with the KNN classifier and the accuracy of the cross validation came roughly to be around 35. We were able to observe that the SVM classifier outperformed the KNN classifier.The goal of the SVM classifier is to produce a model based on the training data which will be able to predict the label for the training data accurately. A main advantage of SVM classification over KNN is that SVM performs well on datasets having many attributes, even when there are only a few cases that are available for the training process.

- Once the model is generated then we do the prediction of the test images by using **clf.fit** method and supply the test images descriptors to this. All the predicted test labels are appended to the CSV.

Below is the figure of how the above algorithm works.

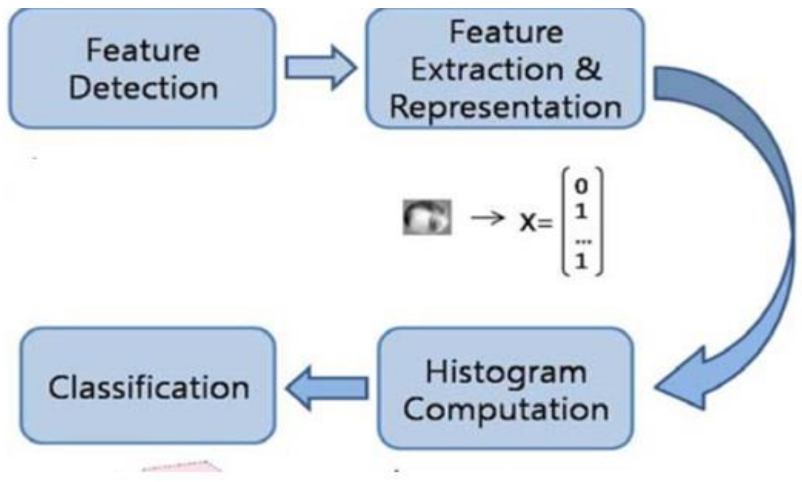

Fig. Workflow diagram for the Algorithm

Methods/Functions used in the code for above algorithm.

1. generateData: This generates the XTrain, YTrain from the train.csv file. It also generates the cross validation set for the classifier. XTrain here will be the image name and YTrain will be label

2. generate_hog: It returns the extracted feature vector for the test data. It uses HOG(Histogram of Oriented Gradients for doing feature extraction) .The images are first resized to a format of 256X256 since all the image size are not the same. The resized image will be given to the hog function in the code.

Next step is generating a classifier for the above images and fit the train data on this.

SVM classifier is used to fit the train data generated from the generate_hog method and once the model is fitted the prediction can be made from the generated model We will do the prediction of the cross validated data using ***SVM.predict*** function and then the score is calculated on that basis.

3. predicttestdata: This method takes the test images and generates hog and do feature extraction of all the test images, the classifier will predict the labels for each image and will be appended to a list to generate xtest and ytest (prediction generated from the classifier) list

4. generatePredictionCSV: This is the last step where the xtest and ytest generated above are written to a csv file and these are the labels for the train images.

Experiment

We did experimentation first on selecting the classification models. We started first with the KNN and then finalized on the SVM classifier based on the below experiment.

Model	Average Model Accuracy with the training Set
KNN	42%
SVM	70%

The result of the experiment led us to choose SVM over KNN classification. Tuning of the parameter in SVM classifier by varying the value of C which is default 1.0.
The experiment was performed by varying the size of testing and training data and average was taken from total of 5 experiments.

Result

We generated the classification model with a cross validation set of around 3000 images and training set of 35000 images. The average performance of the SVM classifier was approximately varying from 65%-75%. The test images are predicted using that classifier and the prediction for test images are generated in CSV file which is uploaded to the Kaggle.