Computer Vision

Assignment-3

Suhgulur

Akmehra

Qiuwshou

Part-1

Baseline-SVM

The Baseline was implemented for both Color image and Grey image. The code is presently runs for color image.

To run it for Grey image – change the value of color in **SVM.h** file, line 100 to false.

The command for execution was

./a3 train baseline

./a3 test baseline

Training

For each image extract features (40 X 40). Write these features into a file as required by Cornell multiclass SVM API. Call the train command.

Training

The test image features are also extracted and written into a file as required by Cornell multiclass SVM API. Call the classify command.

Other things tried

The image was reduced to 40 X 40 size. Tried converting image to HIS format and extract the Intensity for grey scale image which was giving higher accuracy. However we have kept it normal grey scale conversion of image in the code. The other dimensions 30X30 and 50X50 were having almost same accuracy. However drastically reducing it to 20 X 20 was having less accuracy and any image above 60 X 60 was taking too long to train.

Analysis of Accuracy

For colored image there was accuracy of 19%

For grey image there was accuracy around 7.6%. But on using Intensity channel of HSI format of the image the accuracy was around 14%.

Part-2

1.Eigen

The code is written **Eigen.h.**

Training

The following steps are followed to extract Eigen vectors from the training data.

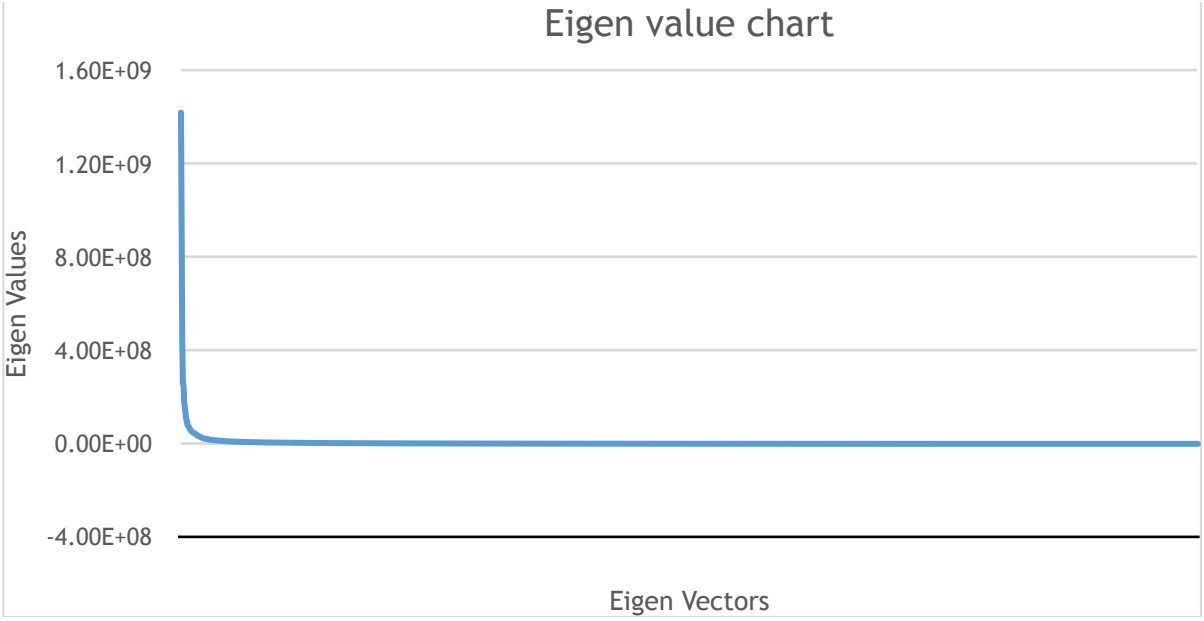
For each image,

1) Convert the image into grey scale and resize it.
2) Put the above unrolled image to a column of a matrix, let us call this matrix B.
3) Each image will have size * size features. Calculate the mean of the features.
4) Subtract mean matrix by each image vector in this matrix. Let us call is A.
5) Co-variance matrix $C = AA^T$
6) Eigen vectors are computed by CImg library API called symmetric_mean which return Eigen vectors and Eigen values.
7) We have printed Eigen Vectors by unrolling back to images in the folder Eigen/ EigenfaceXX.png . 0<=XX<K
8) Only the top K Eigen faces are printed and considered for further evaluation. K = 20 in our case. This is based on how muck Eigen values decreases.
9) Using these top-K Eigen vectors we project the input data of (size X size) in to K dimensions.
10) Send these data to Cornell SVM classifier API. Save the model file generated.

Testing

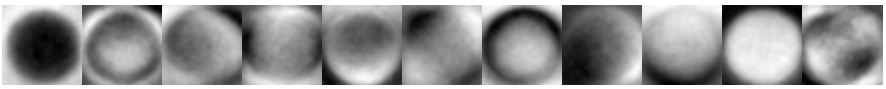
1) Load the saved Eigen vectors.
2) Read the test image and save it as a vector
3) Multiply the vectors and test image to project the test image into k features.
4) Give the data to SVM classifier which uses the saved model to classify the image.

Eigen values drop in the following way.



After seeing this we kept a threshold of k = 20. That is we consider top 20 features.

Eigen Images (Eigen faces). Only first 10 is shown here.



Analysis of Accuracy

The accuracy we are getting is 12%

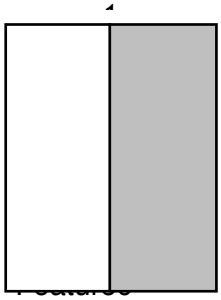
The accuracy in PCA is same as that of baseline SVM (Grey). Basically the job done by PCA is reduction of noise by only selecting features which is important (Feature Selection). This decreases the training time of the data. As it is decreasing the noise we expect slight improvement in accuracy which is also seen here.

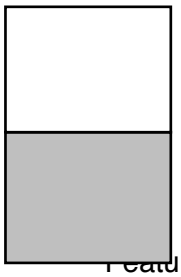
2.Haar Features:

The code is featured in **Haar.h**

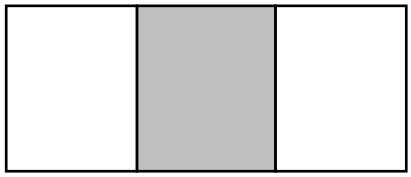
By far Haar was the most interesting and most time complex among the others in this assignment implemented. After reading Viola-Jones paper, we decided on taking the four features given in the paper.

The following are the four features, with ratios written on top of it.

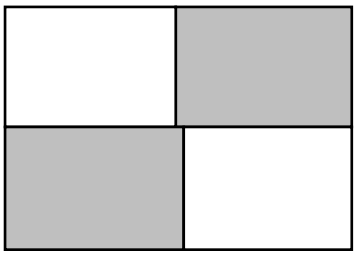




Feature1



Feature2



Feature3

Viola-Jones needs rectangles and for a rectangle given two opposite ends the extraction of extract area of rectangle specified by of it in a constant time. Firstly, 1000 rectangles positions are captured and stored in a place. During testing these rectangles are loaded in **load_modal** function

The type of rectangle is chosen randomly and for each rectangle the initial points are chosen randomly. The scale of the image is also chosen randomly which ranges from 1 to 8. Based on the randomly chosen initial points, feature and scale other points are calculated and stored in a file (**Haas/Features.txt**). The encoding of this file is explained in the **Haar.h** file.

Training Steps

1) For every train image create Integral Image (Summed matrix) and also the summed squared matrix.

2) Using this Integral images, for every 1000 features which was saved , calculate the value such that

   Val = Sum(Area of Black Regions) – Sum(Area of white regions)

   Note: To calculate area, variance normalization is done to overcome the problems like intensity differences in the training images. This is done using summed area as well as summed squared area. Acknowledgement for this technique: Viola-Jones paper.

3) These values are the feature values for that image.

Put all these features of each image in a format required for Cornell SVM multiclass classifier and train it.

Testing

For the test image, extract features in a similar way as above and write in a format required for Cornell SVM multiclass classifier and call classify API.

Execution steps:

./a3 train haar

./a3 test haar

Other techniques Tried:

1) Currently the number of rectangles chosen is 1000. The paper suggested to keep 1600. I tries for 4000 features which by itself was taking long time to train, however the accuracy was almost the same. Didn't proceed further than 4000.
2) Tried Normalizing the 1000 features of each image in terms of scales of [0, 10], [0, 1] or [0,255]. Not sure on how the SVM Classifier API handles the input, but on these cases the SVM train didn't complete for longer time.

Magic Parameters

Viola-Jones contained lot of magic parameters. There was too less time to tune these parameters

1) Number of rectangles.
2) The dimensions of the rectangles.
3) The scale of the rectangles.
4) Normalizing the data and its range.
5) Variance normalization technique parameters.
6) Using Intensity scale for instead of grey images.

Analysis of Accuracy

The accuracy is 13 %. I had expected better accuracy results but couldn't tune parameters due to shortage of time. This should basically behave better than above two methods.

3.Bag of words:

The code is featured in **Bags.h**

Using implementation of bag of words, we use k-mean algorithm to cluster the sift descriptors of each image into k visual words or features. Then each image is represented as a histogram over k features or words.

Training Steps

1) For every train image, resize the image and convert it into sift descriptors. .
2) Initialize 450 features. And each feature is a 128-d sift descriptor randomly picked from the train image. And for each class we randomly pick 18 features, so the total is 450.
3) For each sift descriptor in the train image, we compute the euclidean distances between it and each features, and choose a closest feature as the group of that descriptor. Then each image will be represented as a histogram over k features.
4) Then for each features, we count the members of each features and update the features by computing the centroid of all the members.
5) We repeat 3) and 4) for 10 iterations.
6) After that we save the information in a 25*128 image called "SVM/centroids.png". And it will be read during the test process.

Put all these features of each image in a format required for Cornell SVM multiclass classifier and train it.

Testing

1) For every test image, resize the image and convert it into sift descriptors. .

For the test image, extract features in a similar way as the train image and write in a format required for Cornell SVM multiclass classifier and call classify API.

Execution steps:

./a3 train sift

./a3 test sift

Magic Parameters

1) Number of features.
2) Size of images.
3) Initial features.

Analysis of Accuracy

Applying 25 features, resizing of the images to 40*40, 1000 iterations, we get 4% accuracy. The reason might be the size of image is too small, so we drop out to much information of the features in the image.

Applying 450 features, resizing of the images to 140*140, 10 iterations, we get average 30% accuracy. So the bigger the image, the more features and information we can get. Obviously the trade-off is running time. Also the number of iterations is not very important. One reason might be that the images of different classes are really distinctive, so their sift features are easy to distinguished. Since we are picking the same number of features from each class, the algorithm will converge very quickly.

Part-3

CNN

As given in the assignment downloaded the Overfeat package. Completed the basic installation steps as mentioned.

From the tutorial **overfeat** is used to extract features the API takes an image and dumps the features into a text file. The minimum dimension of the image should be 231 X 231, because the image dimension should be greater than the kernel in CNN. Color image was used as overfeat can handle color pixels.

For each image the features are dumped into a file. There is a specific format of how the file looks like. Parsing this file the features are extracted. This is done to all the training images and saved in the format similar to Part1 SVM and train API is called.

For testing image similar steps are followed to extract features from overfeat and written into file as expected by Cornell multiclass SVM API and classify API is called.

Execution Steps

./a3 train cnn

./a3 test cnn

Techniques tried:

The whole process of extracting features from overfeat package was very time consuming. It involved around an hour to complete the extraction of features.

1) Played around by extracting features at different layers of Nets.
2) Played around with keeping the resolution of the image ranging from 231 X 231 to 500 X 500.

Analysis of the accuracy:

Accuracy is 72% at layer 18. As expected neural nets is the best of all the methods so far. If check at –f (all layers) the accuracy was 70%.