Assignment 3

Pradeep Ravilla

Maria Soledad Elli

Venkatesh Raizaday

Part 1: Baseline

In part 1 we used the svm_multiclass library as suggested. The baseline approach resizes the image to a specific size and unrolls it into a single row vector. This is done for each image and written to a file which is given as an input to the svm.

The svm generates a model in the training phase and the model is used during the testing phase to give out the results.

Performance:-

The running time of the program increase as we increase the subsample size of the image but the performance gets better.

Img Type: Sample Size	10 x 10	40 x 40
Color	15%	18%
Greyscale	8%	10%

Note: We were having problem with using the binaries hence used the source code. The downside is that it needs to be compiled every time the machine is changed. So, if you get permission denied during runtime, just compile the files in the folder svm_multiclass.

Part 2: Traditional Features

Eigenfood:-

We applied Principal Component Analysis (PCA) on the grayscale feature vectors. To do that we followed the below steps:

1. Collect all the feature vectors of all the images in Training set into one Data Matrix(Data).

1. Calculate the Empirical Mean along each dimension 6400 in our case, as the image size is 80x80. (u)

2. Calculate the Deviations from the mean. We subtracted the empirical mean vector from each row of the data matrix. B = Data - h.u^T, where h is an nx1 column vector of all 1s. h[i] = 1 where i = 1,....n

3. Calculate the eigendecomposition using SVD function in CImg on B. which will return U, S, V. The transpose of the V matrix will give the Eigen Vectors. The (S^T)*S will give a Diagonal Matrix of Square Root of Eigen Values. We squared the values of S are calculated to get the actual Eigen Values.

4. We took the approach of doing trial and Error for finding the correct value of dimension k, instead of finding the amount of variance remaining after projecting the original data into k-dimension. We chose k value less than the number of images in the training sample.

5. After deciding on the value of 'k' we select the top eigen vectors, and project the data vectors into the k-dimension and get feature vector for putting it in the SVM training file.

6. While classifying the same above procedure is followed and the SVM classifies the image based on the projected feature vector values.

Haar-like Features:-

In this approach we used 5 of viola jones features used in the face detection experiment. The steps are as follows:-

Train:

- Generate random feature vectors of the viola jones features with respect to size and position.
- Write these features to a file as they will be used again during testing phase.
- Load the train image and convert it to grayscale.
- Get the integral image for the train image.
- Get the difference of sum for each randomly generated vector and use these as features for the svm.
- Run the svm and generate the model.

Test:

- Load the haar like features generated during training phase.
- Load the test image, convert to greyscale and calculate the integral image for the same.
- Get the difference of sum for each randomly generated vector and give these as features to the svm.
- Give the file to svm for classification
- SVM generates the predictions and the error rate is written to the console.

Performance:-

Haar Features/SVM error	1.0	100.0
2000	14%	22%
10000		

Note: To change the parameters you need manually change the command in HaarLikeFeat.h train and test functions.

Bags-of-words:-

In order to classify the images with the Bag of word approach, first, for each training image we extracted SIFT interest points and their descriptors. After computing the descriptors for all the images, we clustered all of them using the Opencv implementation of kmeans with k equals to 500. This will cluster

together all the SIFT points that are similiar, without taking into account from which image they come from and will give us a label for each sample we have. These labels will then be used to build the final descriptor for each image. Each of these descriptors will be a histogram counting how many of the SIFT points belong to a particular label. This is, each image will be represented by a vector of size k, where each index will have a count of how many of the image's interest points belong to the cluster i, where i = 1, ..., k.

All this information is then transformed as a SVM input and after running the SVM command, the svm model is created.

For the testing part, the same procedure is done for each image but in here, the labels assigned to each image are not valid until the execution of the svm program for classification.

Performance:-

For the svm algorithm, we are choosing a linear kernel with the C parameter set in 100. The execution time is very high because of the amount of information extracted from the images. The training part it's taking approximate 60 min including: SIFT extraction, Kmeans and SVM. The testing part is taking less than 10 minutes to run, but the accuracy is very low, around 6%.

Part 3: Deep Features

For the CNN approach, the descriptors should be extracted from a middle layer of the pre-trained Overfeat CNN. So, for each image, the CNN will return a feature vector of dimensions n*w*h and each feature is extracted after w*h values. Each of these features will be added to the SVM input file.

Note: in order to run the CNN, you need to download the weight files using the download_weights.py script provided in the overfeat folder.

Problems Faced:-

- The process is very slow hence each iteration of testing takes a lot of time. Because of this, is very difficult to try different parameter configurations.
- Sometimes the SVM takes forever to converge.