Team members: Lijiang Guo, Shivani Gupta, Snehil Vishwakarma.

Part 1

We choose to resize images to 40×40. Them we used `SVM multiclass`[1] to train a a multicalss SVM classifier. For greyscale image, the classification rate on testing set is 8%. For color image, the classification rate on testing set is 15.2%. In both cases, the training for SVM is very fast.

Part 2

Question 1 Eigenfood

We first resized all images to 40×40 greyscale. The eigenvalues are shown in Figure 1. When we choose first 30 eigenvectors, the classification accuracy is 4.8%; when we choose to use first 200 eigenvectors, the classification accuracy dropped a little to 3.6%. It is kind of counter-intuitive, because when using more features the accuracy dropped. It is probably because there more noises in the tailing eigenvectors, which actually are not useful in classification task. We plot the first 9 eigenfoods in Figure 2.

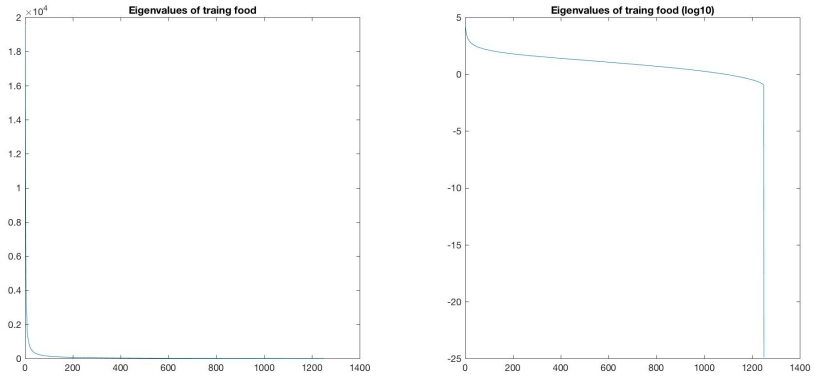

Figure 1: Part 2 Q 1: Eigenvalues.

Question 2 Harr-like features

First, to make images easy to work with a standard set templates, we resize all images to the same size of 300×300. We designed our templates as such. For a rectangle with height

[1] https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html

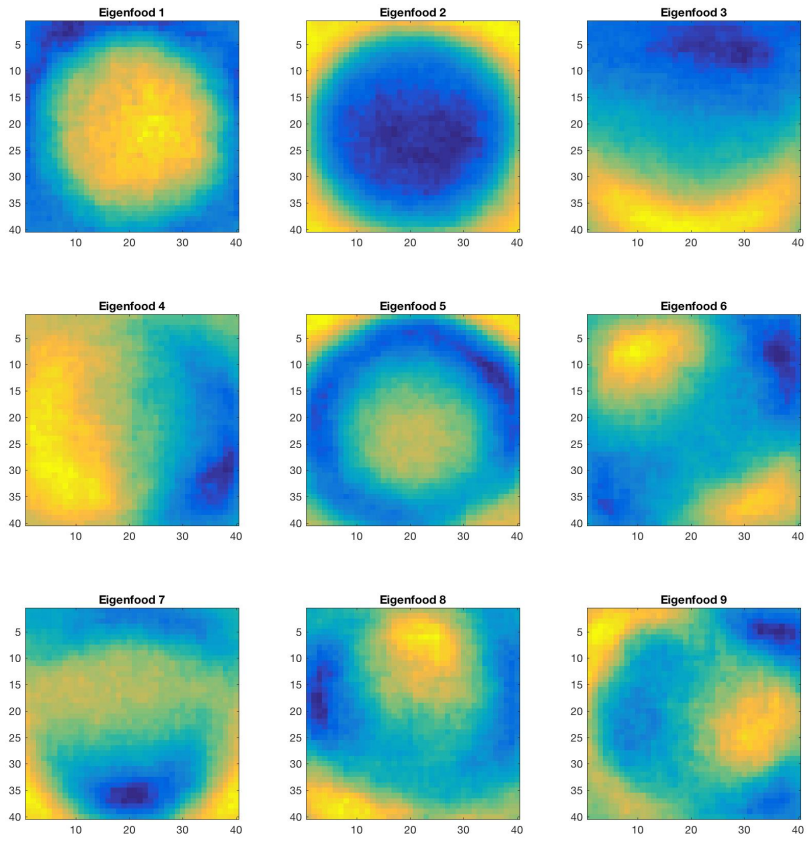

Figure 2: Part 2 Q 1: First 9 eigenfoods.

h and width w, we divide it into 4 quadrants, and set each quadrant's value to -1 or 1. We design 6 patterns templates Figure 3. Then we set each pattern to 5 different sizes, this gives us a total of 30 Harr-like feature. Then for each image we randomly assign a location to each 1 of the 30 features to get a feature. This gives each image a 30 dimensional feature vector. We then put the training set through `SVM multic`lass— to train a SVM classifier. On the test set, we get 6% accuracy.

To further improve this accuracy, we could (1) increase the number of templates, or (2) increase the sample points for each template. Due to time limit, we didn't experiment these options.

Question 3

In this question, we were asked to use SIFT features to create a bag-of-words descriptor for training images, and then use these to classify testing images. There are 1250 training

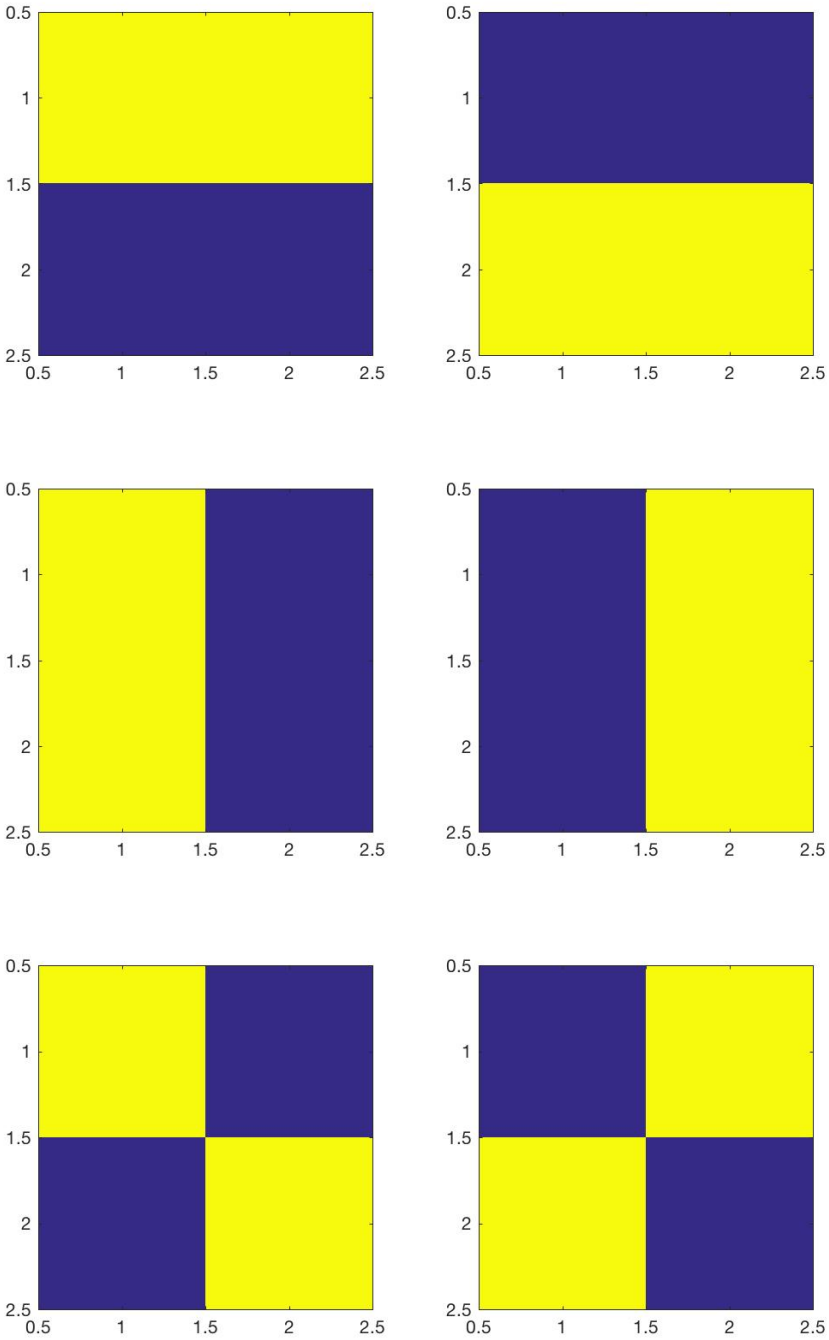

Figure 3: Part 2 Q 2: Templates for Harr-like features.

images, and the total number of SIFT features extracted from these images is about 1 million, which my computer cannot perform K-means clustering in a reasonable time. Therefore, we decide to resize training images in order to reduce the total number of SIFT features. When we resize all images to 200×200 and choose 100 clusters, the classification rate on test set is about 29.4%. When we resize all images to 300×300 and choose 100 clusters, the classification rate on test set is about 32.8%. We used `Matlab`'s `kmeans` function to cluster SIFT features.

Comparison

Method (number of features)	Test set Accuracy
Greyscale (40×40)	8%
Color ($40 \times 40 \times 3$)	15.2%
Eigenfood (30)	4.8%
Eigenfood (200)	3.6%
Harr (30)	6%
BOW ($200 \times 200, 100$)	29.4%
BOW ($300 \times 300, 100$)	32.8%

The Eigenfood and Harr methods do not outperform the naive SVM method. The Bag-of-word method performs significantly better than all others. We think the *Harr* method had the potential to catch up with bag-of-words method, but we need to carefully design the feature templates.

Part 3

Implementation Details

We have used overfeat binary for feature extraction.

Training:

1. If `extract_features_overfeat = 1` in Deep.h file then overfeat library is used to extract features again, else SVM uses already extracted features. By default it is set to 0 as it takes too long to extract the features.
2. Each Image is resized to 231x231 (minimum size of Image required by overfeat) and passed to overfeat binary
3. The extracted features are then written to `train_features` file in the format expected by `svm_multiclass` library.
4. `svm_multiclass_learn` is used to train the `model_file`.
    - Kernel : RBF

- Gamma : 0.0625
- Cost : 3.2

Testing:

1. Resize the image to 231x231 and extract the features.
2. Use `svm_multiclass_classify` to predict the class of the image

Comparison

Algo = Deep takes much longer than any other Algo to train as well as test, but gives the best accuracy. Accuracy 71%.

Problems

- `model_file, train_features` size too big too to be pushed.
- `train_features` takes about 75mins
- test takes about 40mins