# Assignment 3

**Authors:**

Achal Shah

Kushal Kokje

Subham Kumar

# Part 1

**1.**

## How to run code

./a3 train baseline

./a3 test baseline

## Code files:

svm.h

## Assumptions

We used SVM Multi Class library by Cornell University. By default, we are running SVM on color images.

We subsampled the image to 20x20 and 40x40 images.We trained SVM for grayscale and colored images and for 1600-d and 4800-d vectors. We also experimented with various svm kernels like RBF, Polynomial .We measured the accuracy of the generated model against test images given and results are summarized below:

| | **20x20 Image** | | **40x40 Image** | |
|---|---|---|---|---|
| | **GreyScale** | **Color** | **GreyScale** | **Color** |
| **# of features** | **400** | **1200** | **1600** | **4800** |
| | **Accuracy** | | | |
| **SVM Linear(C = 1)** | **10%** | **18%** | **12%** | **19%** |
| **SVM RBF(C=1, gamma = 0.1)** | **4%** | **4%** | **4%** | **4%** |
| **SVM Polynomial(C=1, degree = 2)** | **5%** | **4.4%** | **6%** | **5%** |

Below is the confusion matrix for grayscale images -

```
Confusion matrix:
                  ba br br ch ch cr fr ha ho ja ku la mu pa pi po pu sa sa sc sp su ta ti wa
              bagel  3. 0  0  1  1  2  0  1  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  1
              bread  1  0. 0  0  1  0  0  0  2  0  0  1  1  0  0  0  0  0  0  1  0  2  0  0  1
            brownie  1  0  1. 1  0  0  0  1  0  0  0  0  1  0  0  1  0  1  0  2  0  0  0  1  0
      chickennugget  0  0  0  0. 0  1  1  0  3  0  0  0  0  0  1  1  2  0  0  0  1  0  0  0  0
             churro  0  1  1  1  1. 0  0  0  0  0  0  0  0  0  0  0  0  3  0  0  1  0  0  1  1  0
          croissant  1  1  1  1  0  0. 0  2  0  0  0  0  0  0  0  0  0  1  1  1  0  0  1  0  0
        frenchfries  0  0  0  2  1  0  1. 0  1  1  0  0  0  1  0  0  1  0  1  0  0  0  1  0  0
          hamburger  1  0  0  1  0  1  1  6. 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
             hotdog  0  0  0  3  1  2  0  0  1. 0  0  0  0  0  0  0  0  0  1  0  1  1  0  0  0
           jambalaya  3  0  1  1  1  0  1  0  0  0. 0  0  0  0  0  1  2  0  0  0  0  0  0  0  0
      kungpaochicken  1  1  1  1  0  0  0  1  0  0  0. 0  0  0  1  0  0  1  1  0  1  0  0  0  1
             lasagna  1  0  0  0  0  1  0  2  2  0  0  1. 0  1  1  0  1  0  0  0  0  0  0  0  0
              muffin  0  0  1  1  1  0  0  3  0  0  0  0  0. 0  0  0  0  0  0  1  0  1  0  1  1
              paella  0  1  0  2  0  0  0  1  1  0  0  0  0  0. 0  1  2  0  2  0  0  0  0  0  0
               pizza  0  0  0  1  1  1  0  1  1  0  0  0  0  1  1. 0  2  0  0  0  0  0  0  1  0
             popcorn  0  1  0  1  0  0  0  0  1  0  0  1  1  0  2  1. 0  0  1  1  0  0  0  0  0
             pudding  0  0  0  0  0  0  0  1  1  1  0  0  0  0  1  0  1  3. 0  0  0  1  0  0  0  1
               salad  0  0  1  1  0  2  1  0  0  0  0  0  2  0  0  0  1  0. 0  0  1  0  0  1  0
              salmon  1  1  0  0  0  0  2  2  1  0  0  0  0  0  0  1  0  0  0. 1  0  0  0  1  0
               scone  0  0  0  0  1  1  0  1  2  0  0  2  0  0  0  0  1  0  0  1. 0  0  0  0  1
           spaghetti  0  0  0  0  1  1  2  0  0  0  0  0  0  1  0  0  1  0  0  1  2. 0  0  0  1
               sushi  1  1  1  0  0  0  0  1  0  0  0  0  1  0  0  1  1  1  0  1  0  1. 0  0  0
                taco  1  0  0  0  0  1  0  0  1  1  0  0  0  0  0  0  2  0  1  0  0  0  2. 0  1
            tiramisu  0  1  1  0  0  0  0  0  1  0  0  0  2  0  0  0  1  0  0  0  1  0  1  1. 1
              waffle  1  0  1  0  0  0  0  0  0  0  0  0  0  1  0  0  2  1  0  2  0  0  2  0  0.
Classifier accuracy: 26 of 250 =    10%  (versus random guessing accuracy of     4%)
```

Below is the confusion matrix for color images -

```
Confusion matrix:
                  ba br br ch ch cr fr ha ho ja ku la mu pa pi po pu sa sa sc sp su ta ti wa
              bagel  3. 0  0  1  2  0  0  1  0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  1
              bread  1  1. 0  0  0  0  0  0  2  0  0  1  0  0  1  1  0  0  0  0  0  2  0  0  1
            brownie  1  0  3. 0  0  0  0  2  0  1  0  0  0  0  0  0  0  0  1  1  0  0  0  1  0
      chickennugget  0  0  0  0. 0  1  2  0  2  0  0  0  1  0  0  0  3  0  0  0  1  0  0  0  0
             churro  0  0  0  1  1. 0  0  1  2  0  1  0  1  0  0  1  2  0  0  0  0  0  0  0  0
          croissant  0  1  1  2  2  0. 0  0  1  0  1  0  0  0  0  0  0  1  0  0  0  0  0  1  0
        frenchfries  0  0  0  1  0  0  2. 0  0  0  0  0  0  2  0  0  2  0  0  2  1  0  0  0  0
          hamburger  1  0  0  0  0  0  1  8. 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
             hotdog  1  1  0  3  0  2  0  0  1. 0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0
           jambalaya  2  0  1  3  0  0  1  0  1  0. 0  0  0  0  0  0  0  0  0  0  2  0  0  0  0
      kungpaochicken  0  0  1  1  0  0  0  1  1  0  0. 0  0  0  0  0  0  0  0  2  0  2  0  0  0  2
             lasagna  0  0  1  0  1  0  1  1  2  1  0  0. 0  1  0  0  1  0  0  0  0  0  0  0  1
              muffin  1  0  1  1  0  0  0  2  1  0  0  0  1. 0  0  0  0  0  1  0  0  0  0  1  1
              paella  0  0  0  0  0  1  0  1  1  0  0  0  0  2. 1  1  2  0  1  0  0  0  0  0  0
               pizza  2  0  0  1  1  0  0  0  0  0  1  1  0  0  3. 0  0  0  0  0  0  0  0  1  0
             popcorn  1  0  0  0  0  0  0  0  0  0  0  2  1  0  1  0. 0  1  0  2  0  1  1  0  0
             pudding  0  0  0  1  0  1  1  1  0  0  0  0  0  0  0  0  1  4. 0  0  0  0  0  0  0  1
               salad  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  3  5. 0  0  0  0  1  0  0
              salmon  1  2  0  0  0  0  1  0  1  0  0  0  0  0  0  0  1  0  0  0. 2  1  0  0  0  1
               scone  0  0  1  0  1  0  0  0  1  0  0  1  0  0  0  2  0  0  0  1. 1  0  0  1  1
           spaghetti  0  1  0  1  0  0  0  0  0  0  0  0  0  1  0  0  1  0  0  1  3. 0  1  0  1
               sushi  2  2  1  0  1  0  0  0  0  0  0  0  1  0  0  1  0  0  0  0  0  2. 0  0  0
                taco  0  0  2  1  0  1  1  0  0  0  1  0  0  0  1  0  0  0  0  0  1  0  1. 0  1
            tiramisu  1  1  1  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  1  3. 1
              waffle  1  0  0  0  0  0  1  1  0  0  1  0  0  1  0  0  0  0  0  0  1  0  2  0  2.
Classifier accuracy: 46 of 250 =    18%  (versus random guessing accuracy of     4%)
```

# Part 2

## 1. PCA

### Algorithm:

1. Take average of all images
2. Subtract average image from all training images
3. Find correlation matrix using cimg eigenfunction
4. Extract top k vectors which are representatives of each image

### SVM Pipeline:

1. Apply PCA on training images and store top k principal components
2. Write these top k principal components in SVM format
3. Use training principal components to extract features from test images
4. Apply PCA on test images having k features
5. Report confusion matrix and results

### Code files:

svm_pca.h

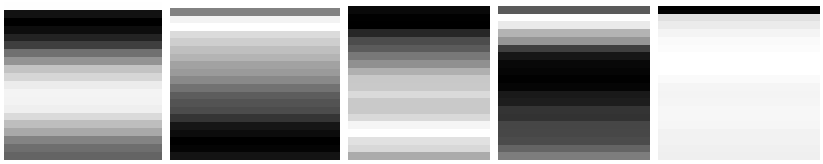
### Comment on eigenvalues:

Eigenvalues are initially decreasing slow but after 25 eigenvectors, values decreses significantly.

Initially, eigenvalues are representing maximum variance. But gradually, it gets decreased and at last, we have eigenvalues which are not much important.

| **Top 5 eigenvalues** | **Least 5 eigenvalues** |
|---|---|
| 1.36926e+10 | 255 |
| 2.4911e+08 | 172713 |
| 1.03651e+08 | 184342 |
| 6.30331e+07 | 186677 |
| 5.39532e+07 | 194695 |

**Top 5 Principal Components:**



## Testing:

1. 2 different size of images (40x40, 80x80)
2. 3 different number of principal components(50,100,200)
3. Different kind of kernels (Linear, RBF, Polynomial)

| | **20x20 Image** | | **40x40 Image** | |
|---|---|---|---|---|
| | **GreyScale** | **Color** | **GreyScale** | **Color** |
| **# of features** | 400 | 1200 | 1600 | 4800 |
| | **Accuracy** | | | |
| **SVM Linear(C = 1) PC= 100** | ~6% | ~8% | ~6% | ~7% |
| **SVM Linear(C = 1) PC= 200** | ~9% | ~7% | ~6% | ~8% |
| **SVM Linear(C = 1) PC= 50** | ~8% | ~9% | ~9% | ~10% |
| **SVM RBF(C=1, gamma = 0.1) PC= 100** | ~4% | ~4% | ~4% | ~4% |
| **SVM Polynomial(C=1, degree = 2) PC= 100** | ~5% | ~4% | ~6% | ~5% |

## 2. Haar Like features

A Haar-like feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize the subsections of an image.

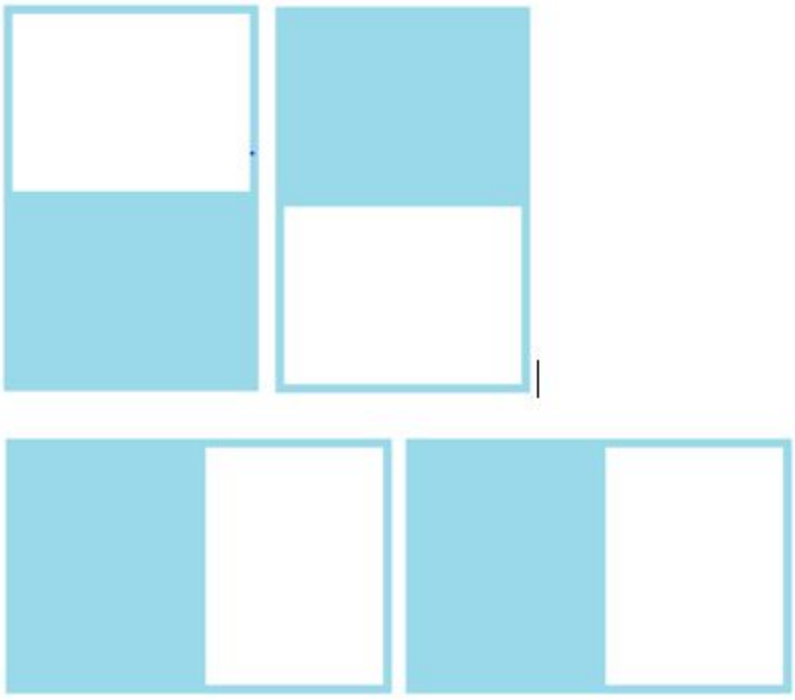
## How to run code:

./a3 train haar

./a3 test haar

## Features :

We have used below features,  light and dark portions of the rectangle



## Code Files :

The code is implemented in **svm_haar.h** header file.

## Approach:

1. Pick 1000 rectangle at random with 1/4 probability of picking above features.
2. Calculate the difference in the summation of the pixel intensities in the dark and the light regions of the rectangle
3. The image features retrieved in step 2 are further feed to SVM for classification. We used Cornell SVM multiclass classifier.

## Results:

| | **20x20 Image** | | **140x140 Image** | |
|---|---|---|---|---|
| | **GreyScale** | **Color** | **GreyScale** | **Color** |
| | **Accuracy** | | | |
| **SVM Linear(C = 1)<br># of rectangles: 500** | 5% | 6% | 26% | 28% |
| **SVM Linear(C = 1)<br># of rectangles: 1000** | 3% | 4% | 25% | 26% |
| **SVM RBF(C = 1)<br># of rectangles: 1000** | 4% | 4% | 4% | 4% |

## Accuracy:

The accuracy we got is 13 %. We believe that the accuracy can be further improved by tuning the parameters as described in Viola-Jones paper.

## 3. Bag of visual words

## Approach:

1. Get sift descriptors of random image
2. Initialize random centroids with any sift descriptors
3. Calculate distance between centroid and image descriptors
4. Assign clusters to each points
5. Update centroids by taking means of all descriptors in cluster
6. Create histogram of each image
7. Apply SVM to histogram of each image

## Code files:

svm_kmeans.h

## How to run the code

./a3 train eigen
./a3 test eigen

| | **20x20 Image** | | **140x140 Image** | |
|---|---|---|---|---|
| | **GreyScale** | **Color** | **GreyScale** | **Color** |
| | **Accuracy** | | | |
| **SVM Linear(C = 1)<br>k= 20<br>Kmeans iteration: 20<br># of features: 500** | 5% | 6% | 26% | 28% |
| **SVM Linear(C = 1)<br>k= 15<br>Kmeans iteration: 20<br># of features: 375** | 3% | 4% | 25% | 26% |
| **SVM Linear(C = 1)<br>k= 10<br>Kmeans iteration: 20<br># of features: 250** | 5% | 4% | 22% | 23% |

# Part 3

## Approach

- We used OverFeat package to extract features from the images.
- We also downloaded the pre trained weights .
- For each image, OverFeat use the weights and dumps the features into a files. Color image was used .
- This files was input to SVM Multiclass library and SVM then generates a model file. This model file is used by SVM to classify test images.

OverFeat allow us to extract features at different layers. We experimented with different layers and different resolutions of images.

## How to run the code

./a3 train deep
./a3 test deep

| | **150x150 Image** | **250x250 Image** | **350x350 Image** |
|---|---|---|---|
| | **Accuracy** | | |
| **SVM Linear(C = 1) With all layers** | 52% | 65% | 58% |
| **SVM Linear(C = 1) With 12th layer** | 40% | 58% | 55% |

## Reference:


1. https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html
2. **Robust Real-Time Face Detection, Viola Jones paper**
3. https://en.wikipedia.org/wiki/Haar-like_features
4. http://users.utcluj.ro/~tmarita/HCI/C7-8-extra/Face-detect/violaJones_CVPR2001.pdf
5. Discussion with Abhishek Mehra(Especially for haar like features)
6. http://cilvr.nyu.edu/doku.php?id=software:overfeat:start
7. https://en.wikipedia.org/wiki/Bag-of-words_model_in_computer_vision