B657: Computer Vision

Assignment 3: Image Warping, Matching and Stitching

Team Name: snaha-islammdl-knayem-a3

Part 1: Baseline

Command to run:

```
From Build: make
From Train: ./a3 train baseline fooddata2 COLOR
                 mode  algorithm  folder   color_mode
From Test: ./a3 test baseline fooddata2 COLOR
```

mode -> `train` or `test`
algorithm -> `baseline` or `eigen` or `haar` or `bow` or `deep`
folder -> in which folder train and test image folder exist
color_mode -> `GRAY` or `COLOR` (this is optional for `deep` algorithm)

Change the size:
To change the subsample image size, we use a variable named `size` in `SVM.h`.
Please remember to use `make` command after changing the size.
In Grayscale type subsample image, the feature vector is `SIZExSIZE` long, whereas for COLOR type, the feature vector is `SIZExSIZEx3` long.

```
[SVM.h] static const int size = 40;
```

Train Steps:

1. We have to resize all the images in appropriate `size` according to the **color_mode**.
2. Then store in appropriate format in `train.dat`. This `train.dat` is necessary for `svm_multiclass` package to train.
3. `svm_multiclass` package use this `train.dat` file and generate `model` file after many iterations.

Test Steps:

1. First step is same as train.
2. Then store in appropriate format in `test.dat`.
3. `test.dat` along with `model` file, `svm_multiclass` generate `prediction` file and gives the error rate details.

Statestices:

Image Type	**Image size**	**Test Error Rate**	**Total Training Time**
GRAY	20x20	87.60% (31 correct, 219 incorrect, 250 total)	520s
COLOR	20x20	82.40% (44 correct, 206 incorrect, 250 total)	158s
GRAY	40x40	89.60% (26 correct, 224 incorrect, 250 total)	445 sec
COLOR	40x40	82.40% (44 correct, 206 incorrect, 250 total)	317 sec
GRAY	60x60	88.40% (29 correct, 221 incorrect, 250 total)	595 sec
COLOR	60x60	79.20% (52 correct, 198 incorrect, 250 total)	634 sec
GRAY	80x80	88.00% (30 correct, 220 incorrect, 250 total)	992 sec
COLOR	80x80	81.60% (46 correct, 204 incorrect, 250 total)	1142 sec
GRAY	100x100	88.00% (30 correct, 220 incorrect, 250 total)	964 sec
COLOR	100x100	81.20% (47 correct, 203 incorrect, 250 total)	1363 sec
GRAY	150x150	88.00% (30 correct, 220 incorrect, 250 total)	2739 sec
COLOR	150x150	82.40% (44 correct, 206 incorrect, 250 total)	3470 sec

So from the statistics, we can see that larger subsample image needs more time to train because the feature vector is bigger. Also on average, COLOR subsample image generally gives better accuracy than GRAY image.

Part 2: Traditional Features

Haar Feature Extraction:

To model the haar filters, we have considered rectangles of various sizes. We then select a set of random points inside a rectangle and create a k-D [1] tree using the points. K-D tree assigns a splitting direction to each of the points. We then traverse the K-D tree inorder and

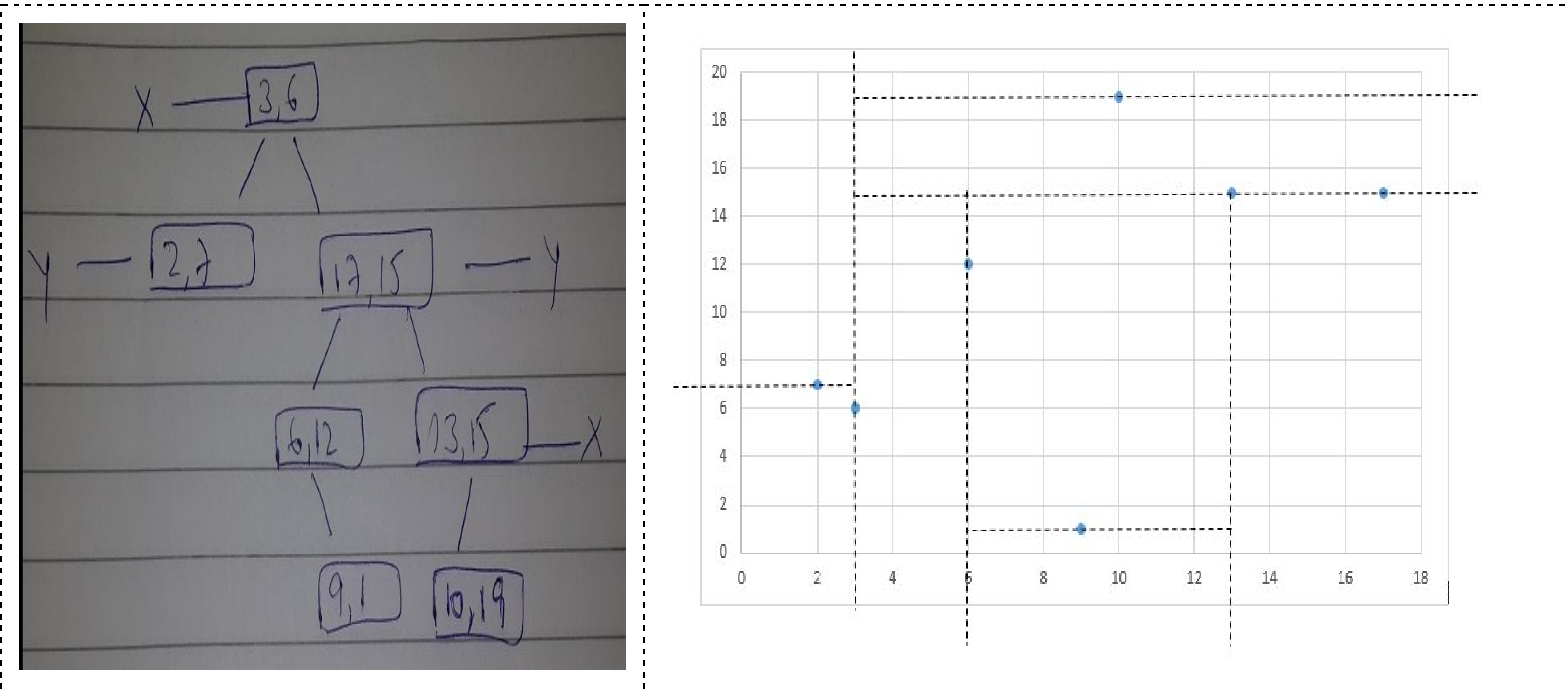

Figure: Creating a Haar filter using K-D tree. Image taken from [1]

split the rectangle recursively in the points by their assigned split direction. This divides a given rectangle into multiple random sub rectangles and then we randomly assign either black color or white color to the subrectangles to create the haar-like filters.

After creating the filters, we convert an given image to an integral image by taking the cumulative sum first at horizontal direction and then at vertical direction. Then, we apply the haar filters to the integral image and calculates the filter response for each individual subrectangle in a given haar filter using the following equation,

$$\text{Sub rectangle response} = d - (b+c) + a,$$

Where,

d = integral image value at the right-bottom pixel of given haar filter subrectangle

c = integral image value at the left-bottom pixel of given haar filter subrectangle

b = integral image value at the right-top pixel of given haar filter subrectangle

a = integral image value at the left-top pixel of given haar filter subrectangle

Finally to get the response for a full haar filter we subtract the summation of the black subrectangle responses from the summation of the white rectangle responses, as follows

Filter Response = ∑ white subrectangle responses - ∑ black subrectangle responses
Finally, we concatenate the responses from all of the haar-filters after applying them in a sliding window fashion on a given image to generate the feature vector representation of the whole image.

Result:
To get better result it is important to experiment with various size and number of haar-filters. But large number of filters create a very large size feature vector which takes too long to run, so we ran with 108 haar-filters and our result is as follows,

Image Type	**Image size**	**Test Error Rate**	**Total Training Time**
GRAY	20x20	90.40% (24 correct, 229 incorrect, 250 total)	415s

Train:
./a3 train haar fooddata2 GRAY

Test:
./a3 test haar fooddata2 GRAY

Part 3: Deep Features

Command to run:

```
From Build: make
From Train: ./a3 train deep fooddata2
                  mode  algorithm  folder
From Test: ./a3 test deep fooddata2
```

For Deep feature, we use subsample image size `231x231`. This is the lowest allowable image size for the simplest and faster kernel. Images smaller than this has to be resized and we do that in both training and test phase.

```
[SVM.h] static const int DEEP_SUBSAMPLE_SIZE = 231;
```

And for image size `231x231`, **Overfeat** package `4096` long feature vector. If the image size increase, the feature vector will be larger and always multiple of 4096 long.

```
[SVM.h] static const int DEEP_FEATURE_SIZE = 4096;
```

Train Steps:

1. Overfeat package takes input only images in feature extraction phase and we have to resize all the images. So we first resize each train images and save them back in name of `resized_image.jpg`. \
2. Then use this image to generate `4096` long feature vector for each image and store the values in `feature.txt` file.
3. From `feature.txt`, we extract the feature values and store in appropriate format in `train.dat`. This `train.dat` is necessary for `svm_multiclass` package to train.
4. `svm_multiclass` package use this `train.dat` file and generate `model` file after many iterations.

Test Steps:

1. First 2 steps are same as train steps.
2. From `feature.txt`, we extract the feature values and store in appropriate format in `test.dat`.
3. `test.dat` along with `model` file, `svm_multiclass` generate `prediction` file and gives the error rate details.

We renamed the following files,

```
svm_multiclass/train.dat   -----------> svm_multiclass/train_deep.dat
svm_multiclass/test.dat    -----------> svm_multiclass/test_deep.dat
svm_multiclass/model       -----------> svm_multiclass/model_deep
svm_multiclass/prediction -----------> svm_multiclass/prediction_deep
```

So for testing only the **Deep** Features, just rename these files.

Image Type	**Image size**	**Feature Vector size**	**Test Error Rate**	**Total Training Time**
COLOR (default)	231x231	4096	25.20% (187 correct, 63 incorrect, 250 total)	2568 s

** Also, the Pre-trained weight files of **Overfeat** are too large (>512MB) that we are not able to upload them in https://github.iu.edu/cs-b657-sp2017/snaha-islammdl-knayem-a3
For this reason, we upload our whole code in **google drive**.
https://drive.google.com/drive/folders/0ByA1mSrlcmG6ODUwNjNJSmRacGM?usp=sharing

References:

1. http://www.geeksforgeeks.org/k-dimensional-tree/