Student Name: Vladislav Dubrovenski, Student ID: 16281273

Vlad Dubrovenski

CS 5582 Computer Vision

Homework 2

The complexity of this homework has increased a lot since homework 1. The most complex part of it was the fact that we used both SIFT and Dense SIFT, as well as VLAD and Fisher Vector, in attempt to compare different combinations of these 4 techniques for matching/non-matching pairs of images. The dataset was pretty large and the computations needed a lot of memory, so the author decided to use his local machine instead of UMKC Remote Desktop as he used for the last homework. The following specs were used:

32GBs Ram(2667MHz), 12 core Intel i7-8750H CPU @ 2.20GHz, SSD, GTX 1060 6GB

MATLAB R2020b(August)

[Q1, 20pts] Compute Image Features, create a matlab/python function that compute n x d features by calling vl_feat DSIFT and SIFT functions (notice that vl_feat also has Python version), implementing the following function:

The first question seems to be easy. However, that is a big mistake to think so as this question lays down the foundation for the rest of the questions (actual, Q2 and Q3 are also important for Q4 and Q5). In this question, I created a function that computes either SIFT or Dense SIFT features based on the option chosen. The following is the code:

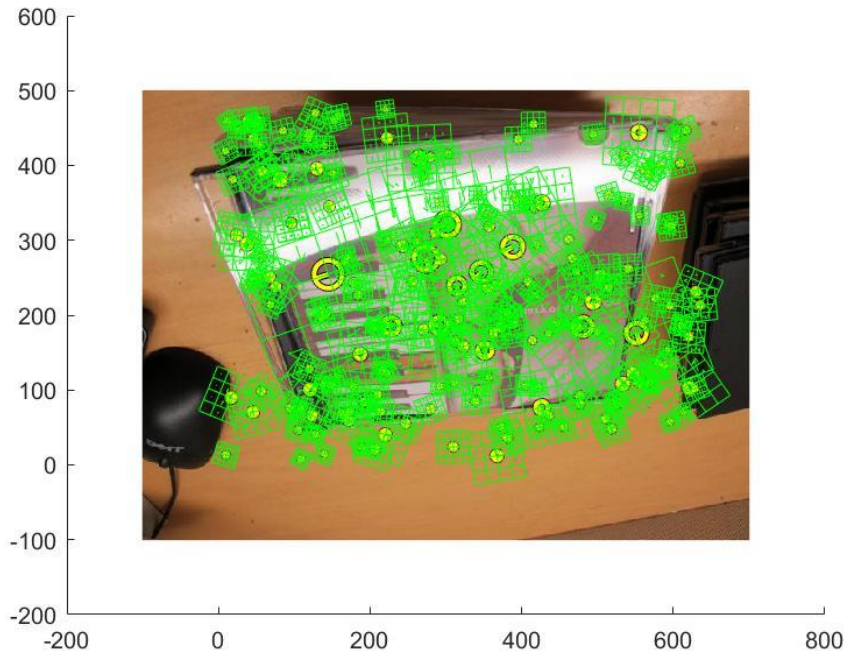
```
function [f,d]=getImageFeatures(im, opt)


if ismember('sift',opt.type)
[f,d] = vl_sift(im);
end
% variant of dense SIFT descriptors, extracted at multiple scales.
%https://www.vlfeat.org/overview/dsift.html
```

Student Name: Vladislav Dubrovenski, Student ID: 16281273

```
if ismember('dsft',opt.type)
    [f,d] = vl_dsift(im);
end
end
```

Note that I return both frames and descriptors. The following plot is those elements on the plot(SIFT for a single image), with green descriptors and yellow frames:



The code to plot it can be found below (reference: https://www.vlfeat.org/overview/sift.html):

```
function plotFeatures(im, f, d, numberOfFeatures)
%plot the features
perm = randperm(size(f,2)) ;
sel = perm(1:numberOfFeatures) ;
h1 = vl_plotframe(f(:,sel)) ;
h2 = vl_plotframe(f(:,sel)) ;
set(h1,'color','k','linewidth',3) ;
set(h2,'color','y','linewidth',2) ;
h3 = vl_plotsiftdescriptor(d(:,sel),f(:,sel)) ;
set(h3,'color','g') ;
hold on
```

Student Name: Vladislav Dubrovenski, Student ID: 16281273

```
I = imread(im);
h = image(xlim,ylim,I)
uistack(h,'bottom')
end
```

[Q2, 20pts] Compute VLAD and Fisher Vector models of image features, for this purpose you need to first compute a Kmeans model for DenseSIFT and SIFT. Use the CDVS data set given as both training and testing for convenience (not the right way in research though, should use a different data set, say FLICKR MIR, or ImageNet), implementing the following functions:

In this question, I had to compute models for VLAD and Fisher Vector that will be utilized in the question 3 for aggregation.

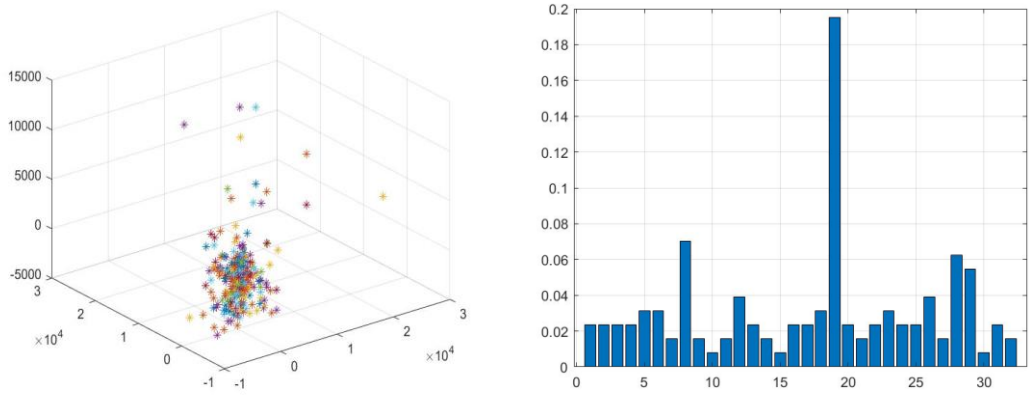
The code for Fisher Vector Model for SIFT matching looks like this:

```
for j=1:length(kd)
    for k =1:length(nc)
        fprintf('\n t=%1.2f: GMM: kd(%d)=%d, nc(%d)=%d ', cputime-t0, j, kd(j),
k, nc(k));
        x = double(d_sift_el1)*A_sift_el1(:,1:kd(j));
        [fv_gmm_sift_el1(j,k).m, fv_gmm_sift_el1(j,k).cov,
fv_gmm_sift_el1(j,k).p]=vl_gmm(x', nc(k), 'MaxNumIterations', 30);
    end
end
```

As we can see from the above, I actually create a matrix for each dimension and number of clusters described in Question 4. Each of those will have a model that would be aggregated to a fisher vector in question 3.

I plotted the means(3d) and priors(1d bar charts) and I include them all under ” HW2\OutputFigures\MeansPriorsGMM”. For the demonstration purposes, let me include the pair of means and priors plots for GMM with kd=24 and nc=32:

Student Name: Vladislav Dubrovenski, Student ID: 16281273



The file called generatePlotsForMeansAndPriors.m will generate the above plots for each pair of dimensions and clusters.

For VLAD model I used(reference: https://www.vlfeat.org/overview/encodings.html):

```
centers_sift = vl_kmeans(double(dataSift), numClusters);
kdtree = vl_kdtreebuild(centers_sift) ;
nn_sift = vl_kdtreequery(kdtree, centers_sift, double(dataSift))
;
```

As we can see, I first get the kmeans, the build kdtree, and then query this kdtree. You may read more about this in in the reference provided above. After this is done, we move further to aggregate the VLAD and Fisher Vector.

[Q3, 20pts] Compute VLAD and Fisher Vector Aggregation of Images, from the given VLAD and FV models, implementing the following functions. Notice that the feature nxd f need to be projected to the desired lower dimension via, f0=f*A(:,1:kd), to match the VLAD model dimension before calling this function.

This was the hardest question of this homework in the author's opinion. In fact, the bottleneck that the author encountered happened in this step, when he tried to save the whole data into the file instead of using return values. It is still not fixed in the main.m, but for the mainEXTRACREDIT.m it was fixed. So, if you encounter that bottleneck, just change the return

Student Name: Vladislav Dubrovenski, Student ID: 16281273

values inside the aggregation functions and in the main.m to be corresponding to the mainEXTRACREDIT.m, and comment out the save/load calls. After that, everything should work a lot faster(no need to wait for writing and reading from the disk). I needed that only for extra credit, but I was waiting for too long for the Question 4 due to this bottle neck.

Aggregation for Fisher Vector is computed in the following way(SIFT):

```
for j=1:length(kd)
    for k =1:length(nc)
        x = double(d_sift_el1)*A_sift_el1(:,1:kd(j));
        if k ==1 && j == 1
        encoding_temp = vl_fisher(x', fv_gmm_sift_el1(j,k).m,
fv_gmm_sift_el1(j,k).cov, fv_gmm_sift_el1(j,k).p);
        encoding1_sift = encoding_temp';
        continue;
        end
        encoding_temp = vl_fisher(x', fv_gmm_sift_el1(j,k).m,
fv_gmm_sift_el1(j,k).cov, fv_gmm_sift_el1(j,k).p);
        encoding1_sift = [encoding1_sift,encoding_temp'];
    end
end
```

And the aggregation for VLAD is computed as follows:

```
[A_sift,s,lat]=pca(double(dataSift));
t0=cputime;
globalCount = 1;
    for j=1:length(kd)
        for numClusters =1:length(nc)
            [nn_sift,centers_sift, ~,~] = getVladModel(dataSift,dataDSFT,
numClusters, 'sift');
            numDataToBeEncoded = length(nn_sift);
            assignments_sift = zeros(numClusters,numDataToBeEncoded);
            assignments_sift(sub2ind(size(assignments_sift), nn_sift,
1:length(nn_sift))) = 1;
            fprintf('\n t=%1.2f: VLAD: kd(%d)=%d, nc(%d)=%d ', cputime-t0, j,
kd(j), numClusters, nc(numClusters));
            x = double(dataSift)*A_sift(:,1:kd(j));
            if globalCount==1
```

Student Name: Vladislav Dubrovenski, Student ID: 16281273

```
            vlad_km_sift_temp=vl_vlad(x, centers_sift,assignments_sift);
            vlad_km_sift = vlad_km_sift_temp';
            disp(size(vlad_km_sift));
            globalCount = globalCount + 1;
            continue;
        end
         disp(1);
        disp(size(vlad_km_sift));
        disp(size(vlad_km_sift_temp));
        vlad_km_sift_temp=vl_vlad(x, centers_sift,assignments_sift);
        vlad_km_sift=[vlad_km_sift,vlad_km_sift_temp'];
        globalCount = globalCount + 1;
    end
end
```

[Q4, 20pts] Now benchmarking the TPR-FPR performance of various feature and aggregation scheme performance against the mini CDVS data set. For the SIFT and DenseSIFT features, let us have kd=[24, 48], nc=[32, 64, 96]. So for each image, we will have 2 x 6 x2 = 24 different feature + aggregation representations. For the total of N images in the mini CDVS dataset, we have M=N*(N-1)/2 total image pairs, and the matching pairs ground truth are given, we only care about the first 100 matching pairs and first 100 non-matching pairs in the fid.mat, which has two variables mp and nmp. Each has a row of two image filenames to their associated images, e.g, mp(1,:): mp_2.jpg and mp1_2.jpg are two matching pairs:

Let us use the Euclidean distance to on those features to compute the TPR-FPR ROCs and find out which one have the best performance. Last year's plots are attached below for example, you only need to plot the last row for 10 feature-aggregation combinations.

This question proved to be the most tedious one. To calculate the 'euclidean' distance, I used the following call(fisher with SIFT as example): `distanceFishingVectorSIFT = pdist2(fisherVectorSift1,fisherVectorSift2,'euclidean');`

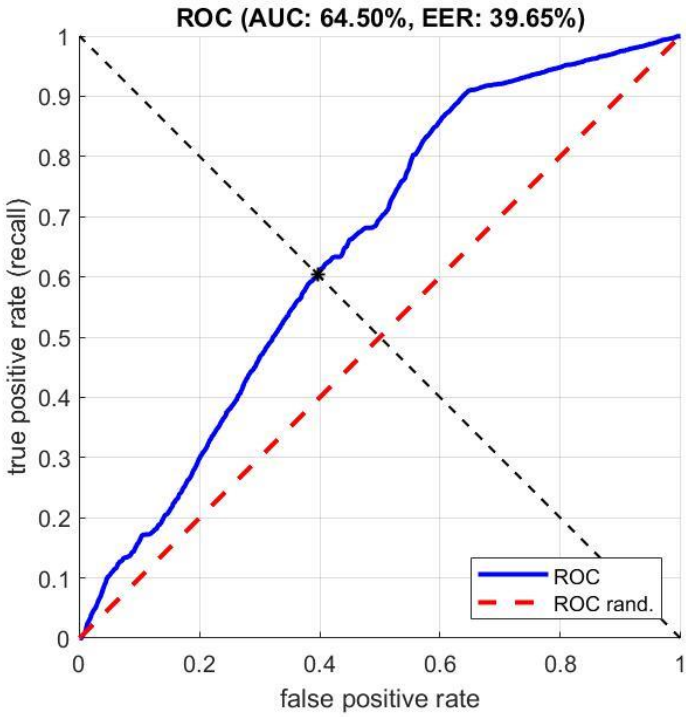
To plot all the diagrams, I used the following link as a reference: https://www.vlfeat.org/overview/plots-rank.html

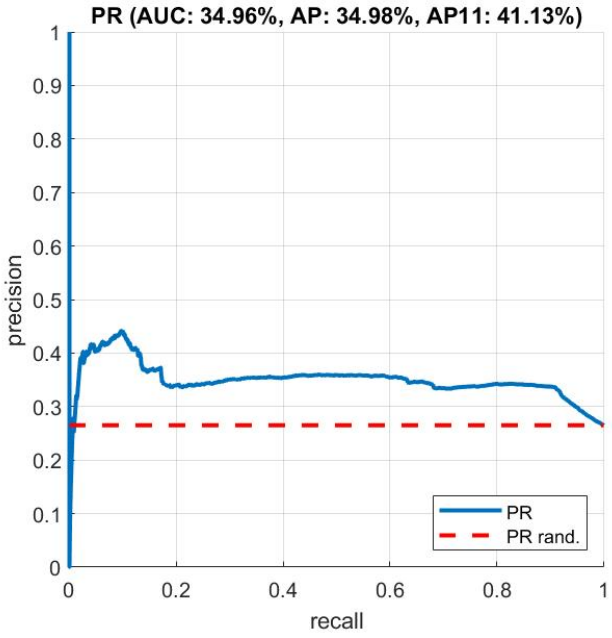
All plots can be found under:" HW2\OutputFigures". For the sake of demonstration, the author

Student Name: Vladislav Dubrovenski, Student ID: 16281273

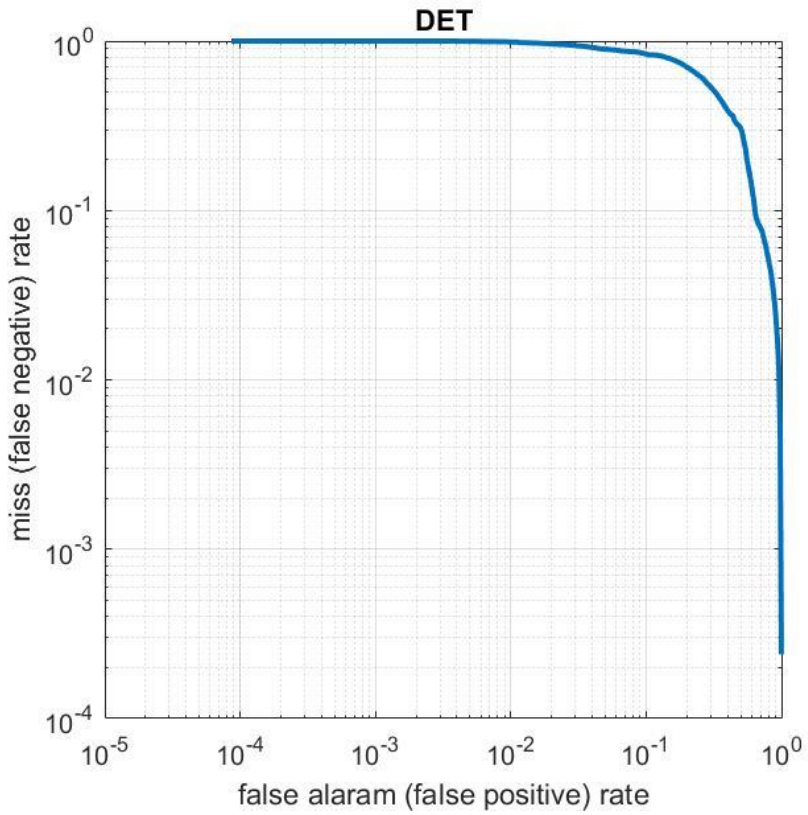
will provide some of the plots here:

1) Fisher SIFT Matching:



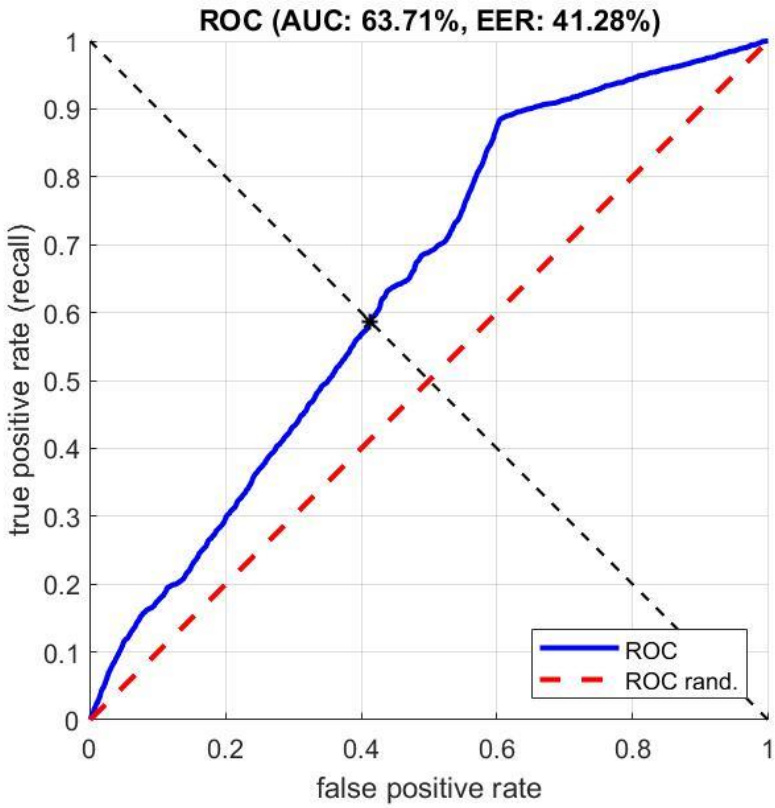


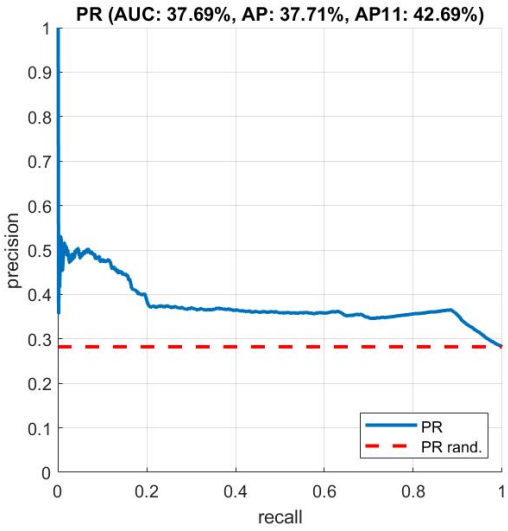
Student Name: Vladislav Dubrovenski, Student ID: 16281273



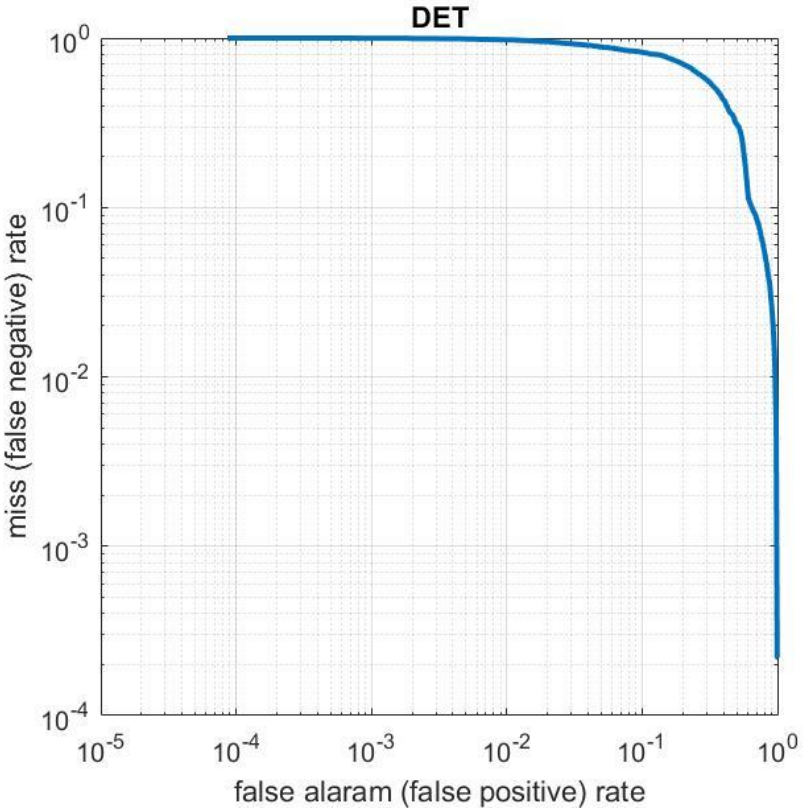
2) Fisher DSIFT MATCHING

Student Name: Vladislav Dubrovenski, Student ID: 16281273



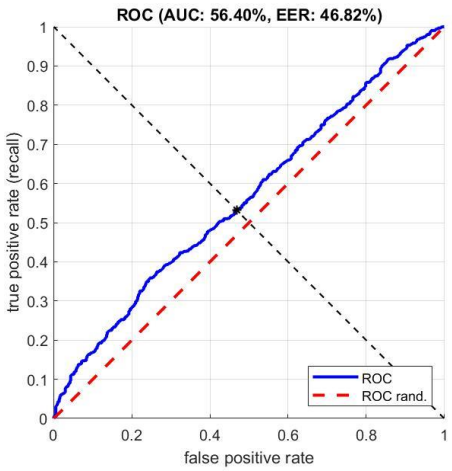


Student Name: Vladislav Dubrovenski, Student ID: 16281273

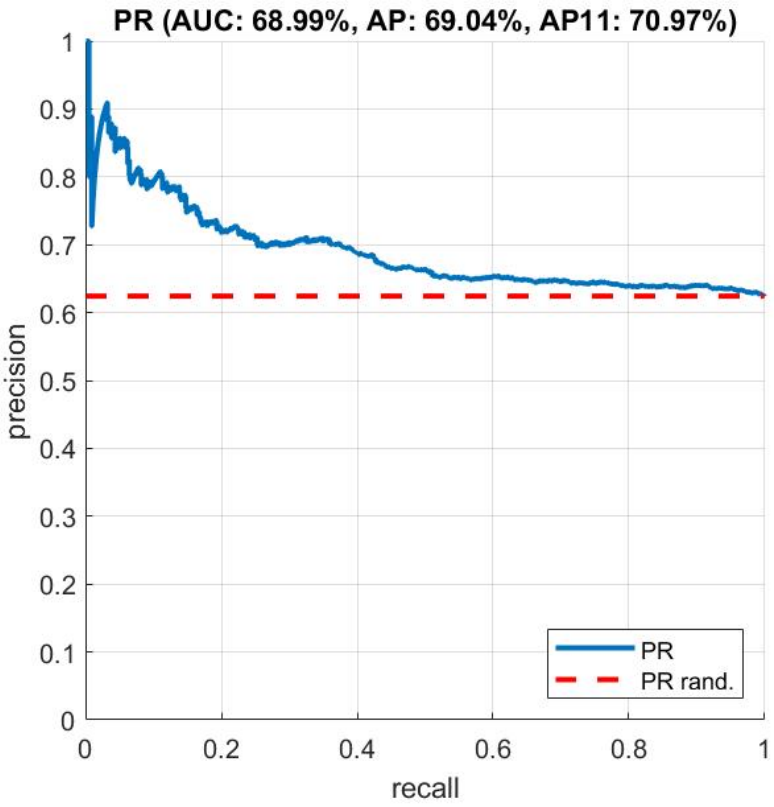


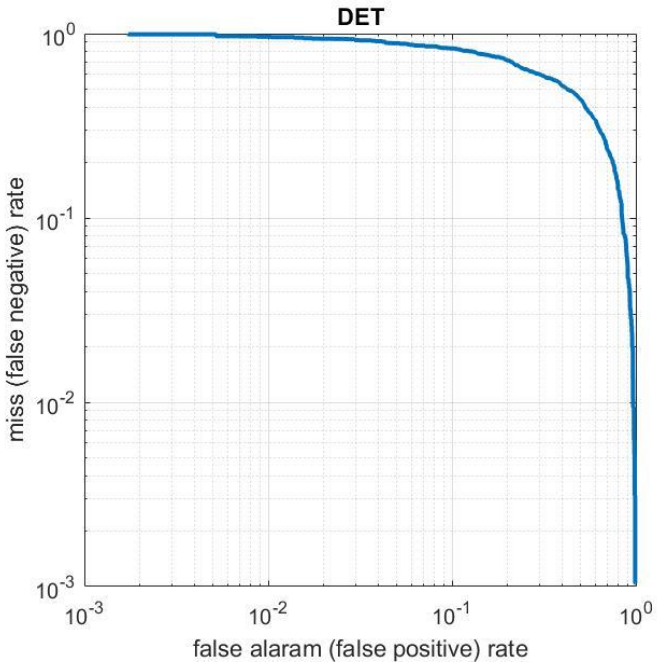
As we can see from the above, Fisher SIFT outperform Fisher DSIFT, even though not be much (within 2%).

3. VLAD SIFT MATCHING



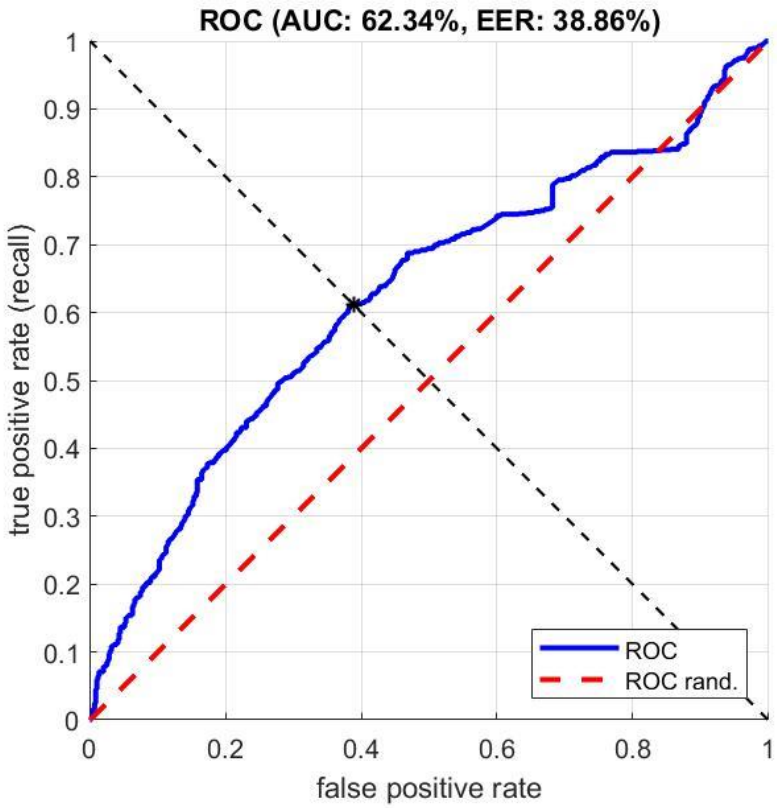
Student Name: Vladislav Dubrovenski, Student ID: 16281273



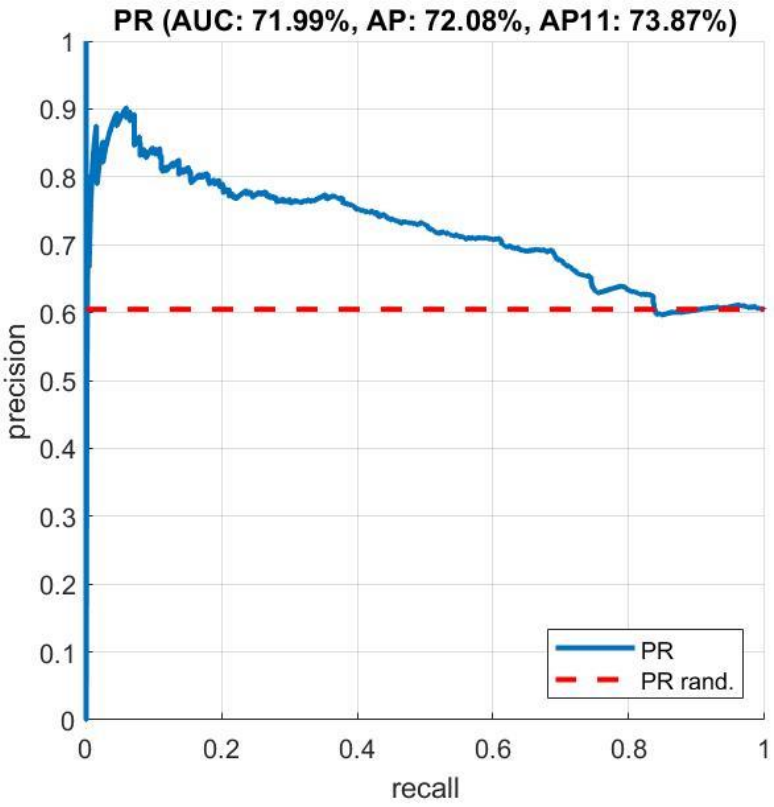


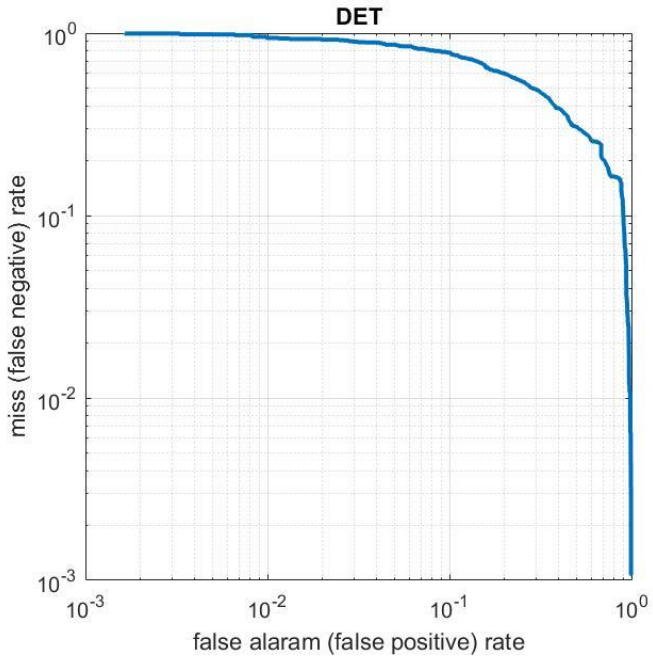
Student Name: Vladislav Dubrovenski, Student ID: 16281273

4. VLAD DSIFT MATCHING



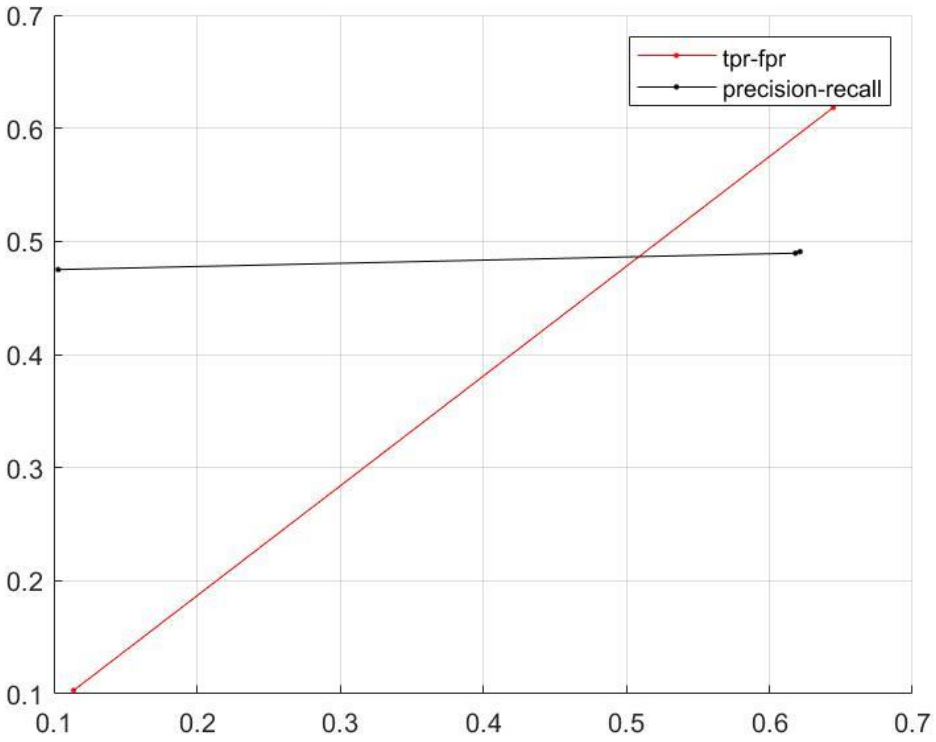
Student Name: Vladislav Dubrovenski, Student ID: 16281273

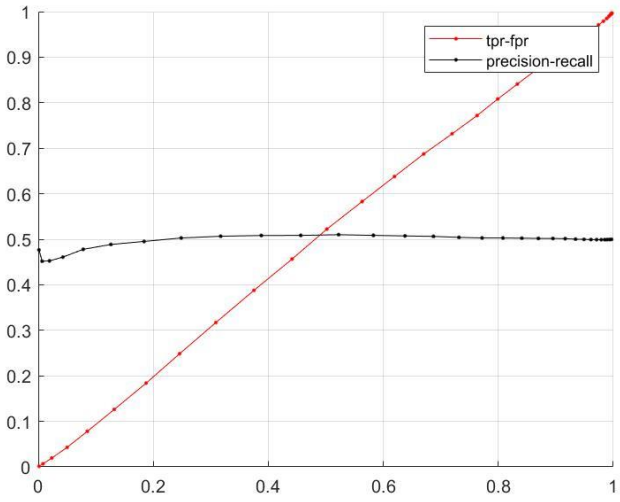




Student Name: Vladislav Dubrovenski, Student ID: 16281273

As we can see, the DSIFT version of VLAD outperforms the SIFT version of VLAD, but VLAD in general loses to Fisher. Here is the precision recall for VLAD(1) and Fisher(2)





Student Name: Vladislav Dubrovenski, Student ID: 16281273

5. The rest of the plots

I for the non-matching pairs, please consult the following folder(the sequence is the same as for matching): HW2\OutputFigures\outputNonMatching. The fisher with SIFT again outperformed the rest of competitors.

[Q5. bonus 30pts]: Compute SIFT and DenseSIFT for the NWPU Aerial image data set, and show the VLAD and FV aggregation results. Notice that you can choose your own PCA dimension and K for kmeans and GMM. To make this easier, only validate against the first 15 classes and 20 images per class. So you will have a 300x300 pdist2() matrix and those sharing the same semantic label will be considered a TP pair, while others TN pair, analyze which combination (2 feature x 12 aggregation) gives the best performance.

This question was complete and all the results can be found in: HW2\OutputFigures\outputMatchingExtraCredit
HW2\OutputFigures\outputNonMatchingExtraCredit

Instead of including all the plots, I would like to say that the Fisher DSIFT outperformed the rest of the competitors in this question. I do run behind and the deadline is pretty close, so let me explain how to run the code.

HOW TO RUN THE PROJECT:

Make sure, vlfeat is present inside the HW2. Also, you will need datasets inside this folder as well, as well as the file with names(I changed the name to names.mat, so change it to cdvs_sift_aggregation_test_data.mat in getImagesFeautres.m).

After the above is done, go to main.m. At the top, set “matching” to 0 or 1, depending on whether you want to get non-matching or matching results respectively.

You will need a lot of disk space for this part. There will be 16 resulting plots. After that is done, change “matching” to opposite value to have the other 16 plots.

For the extra credit part, as I explain in question 3, you will need to switch from saving and retrieving the data to be getting it as return values in the functions. Do not forget to do that step or it will take forever as the dataset is much larger than the normal credit portion. After that switch was done, head to mainEXTRACREDIT.m

Student Name: Vladislav Dubrovenski, Student ID: 16281273

and run it. Please note it does use different getter for images features called "getImagesFeaturesExtraCredit". I would not try running it without 16GB of RAM as my Matlab uses 8GB just when idle and 20-30GB when performing calculations.