Course: DD2427 - Exercise Set 2

When you start *Matlab* remember to add your image directory to your path. Download from the course homepage the images for this exercise set. (These are image patches taken from the IMM face database.) They are contained in two separate directories `Aligned_Pics` and `Misaligned_Pics`. Using the command `montage` you can see both these directories contain images of eyes and noses. In the first directory all the images of the same part are centred at the same point while the in the second directory that is not the case (hence the title misaligned). In this exercise you will extract simple global feature vectors to summarize the content in these image patches and see how invariant to small shifts in translation.

Background 1: *Extract image template descriptor*

To get started, read in the image `Aligned_Pics/eye001.png` and transform it to a grayscale image from a colour image. The command `single` casts the grayscale values to single precision, while the last line normalizes the grayscale values and results in some robustness to illumination variations.

```
>> col_im = imread('Pics/eye001.png');
>> im = single(rgb2gray(col_im));
>> im = (im - mean(im(:)))/ std(im(:));
```

I have ensured that all the images you will load for this Exercise Set have the same size. Thus you can make a feature vector of size `4900`×`1` from the grayscale image `im` with the simple command:

```
>> fs = im(:);
```

This feature vector is known as the template image. As all the images have the same size each of the images will generate an `fs` of the same size and can be compared using standard distance functions such as the Euclidean distance. However, in general, this will not be the case and one has to choose a strategy for extracting `fs` of the same size - resize all images to a fixed size or extract a central sub-patches of a fixed size from each image.

You can plot `fs` using the command `stem`:

```
>> stem(fs(1:10:end);
```

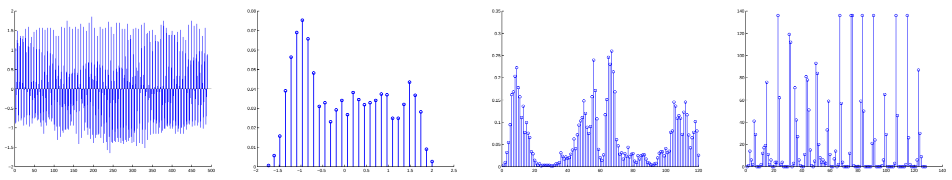
Note you have plotted just every 10th entry of `fs` otherwise the figure would be too cluttered. When you do this you should get the figure shown in 1(a).

Background 2: *Compute histogram of pixel intensities*

Next you will build an image feature descriptor based on the histogram of the pixel intensities in an image. Use the *Matlab* function `hist` to compute a histogram of the pixel intensities for the image `im`:

```
>> nbins = 30;
>> [fs, xx] = hist(im(:), nbins);
>> fs = fs(:)/sum(fs);
>> stem(xx, fs);
```

The command `hist` returns a histogram with `nbins` bins and the vector `H` contains the number of pixels which have grayscale values within each bin. The bin centres are stored in the vector `xx`. The command `fs(:)/sum(fs)` turns the vector `fs` into a column vector and normalizes the feature vector so that its elements sum to one. Once again you can use the function `stem` to plot `fs`. You should get the figure shown in 1(b) when this is applied to image `Aligned_Pics/eye001.png`. Note that a normalized histogram produces a feature vector of size `nbins`×`1` and with entries of the same magnitude irrespective of the size of the image.



(a) Template **(b)** Histogram **(c)** Grid histograms **(d)** Sift

Figure 1: **Descriptors of the image patch eye001.png.** Above are shown each of the descriptors which you calculate in this exercise to summarize a patch. Note the template feature has been sub-sampled by a factor of 10 for clarity.

Background 3: *Compute an* `ng`×`ng` *grid of histograms*

As stated in the lectures a histogram feature, while invariant to shifts in translation, destroys any spatial information within the image patch (ie there is a set of bright pixels in the upper left hand corner). One simple way to combat this is to split the image into a regular grid of `ng`×`ng` sub-patches, see figure 2, compute a histogram of the pixel intensities for each of these sub-regions and then concatenate these histograms to produce one single feature vector. The following code shows how to grab the top-left hand sub-region from a 2×2 grid and compute its histogram:

```
>> ng = 2;
>> xs = floor(linspace(1, size(im, 2)+1, ng+1));
>> ys = floor(linspace(1, size(im, 1)+1, ng+1));
>> ii = xs(1):xs(2)-1;
>> jj = ys(1):ys(2)-1;
>> pim = im(jj, ii);
>> fs = hist(pim(:), nbins);
>> fs = fs(:)/sum(fs);
```

Now you should write a function

```
function fs = ExtractGridHistogram(im, ng, nbins)
```

which returns a column vector `fs` of size `(ng*ng*nbins)`×`1` and should contain these nested `for` loops and perform the following.

```
fs = [];
for i=1:ng
  ii = xs(i):xs(i+1)-1;
  for j=1:ng
    jj = ys(j):ys(j+1)-1;
    Extract image patch defined by ii and jj
    Compute histogram of the patch's pixel intensities
    Normalize the histogram
    Concatenate this histogram to fs
  end
end
```

If you call this function with the image `Aligned_Pics/eye001.png`, `ng=2` and `nbins=30` one should get the descriptor shown in figure 1(c). Note you may visit the sub-regions in a different order to me, so vary this order if your feature vector does not look like the one shown.

Background 4: *Extract SIFT descriptor*

The final descriptor you extract from the image `im` is a SIFT descriptor. Luckily there is open-source *Matlab* code - VLFeat - which provides functions to compute this feature. Please download the package, unpack and install it. Follow the instructions given in VLfeat's Matlab install page. The following is code to compute the SIFT descriptor of the image `im`:

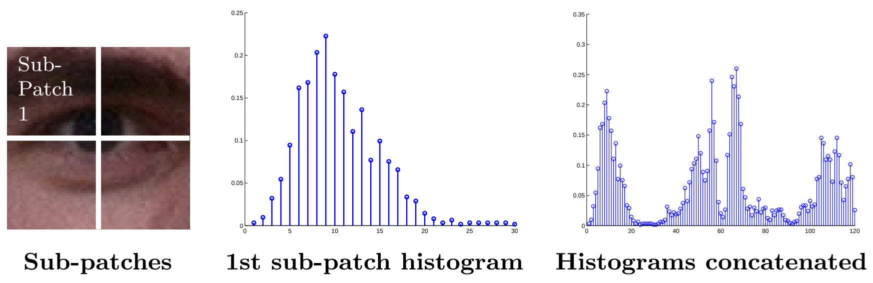


Figure 2: **Descriptor based on concatenation of histograms.** The image is divided into a regular grid of `ng`×`ng` sub patches. A histogram of the pixel intensities are used to describe each sub-patch. These histograms are then concatenated to form a final descriptor of the image patch.

```
w = size(im, 1);
sc = (w-2)/ 12;
fc = [w/2; w/2; sc; 0];
[fc, fs] = vl_sift(im, 'frames', fc);
fs = double(fs(:));
```

If you plot `fs` using `stem` then the SIFT feature should look as in 1(d). The 4 element vector `fc` contains the *frame* information of the *SIFT* feature. One can also visualize it via these commands (see figure 3)

```
>> figure(1)
>> hold off;imagesc(im); colormap(gray); axis equal;
>> hold on;
>> h3 = vl_plotsiftdescriptor(fs, fc);
>> set(h3, 'color', 'r', 'Linewidth', 3);
```

You should write a function

```
function fs = ExtractSiftDescriptor(im)
```

to wrap about the SIFT extraction code.

Exercise 1: *Extract the four types of image descriptors*

You will now write a function `ComputeDescriptors.m`. This function will read in the images you uploaded for this exercise and compute their feature vectors. Define it as:

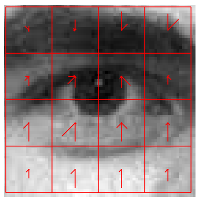


Figure 3: **The grid used in your Sift descriptor**. The histogram of the orientations of the image gradients is extracted from each sub-patch and are shown above.

```
function Fs = ComputeDescriptors(DirName, nbins, ng)
```

The variable `DirName` is the name of the directory containing the eye and nose images. To begin with set `DirName` to the `Aligned_Pics/`. The function's output `Fs` is a cell array of length 4. Each cell entry `Fs{i}` is a matrix of size `nf_i`$\times$`ni` where `ni` is the number of images in the directory and `nf_i` is the dimension of the `i`th extracted feature vector (`70*70` for the template , `nbins` for the histogram, `ng*ng*nbins` for grid histogram and `128` for the SIFT features). I assume that these images are in a directory with no other images. To get a list of images in a directory:

- Use the command `dir` to list all the image files in the directory and then load each image and save in a cell array as follows

  ```
  mystr = [DirName, '/*.png'];
  im_files = dir(mystr);
  ni = length(im_files);
  ims = cell(1, ni);
  addpath(DirName);
  for i=1:ni
      col_im = imread(im_files(i).name);
      im = single(rgb2gray(col_im));
      ims{i} = (im - mean(im(:)))/ std(im(:));
  end
  ```

- Declare the cell array `Fs = cell(1, 4)`
- Initialize each matrix `Fs{i}` to keep a record of the description of each image w.r.t. feature type `i`:

  ```
  Fs{i} = [];
  ```

- Run through all the images and using the functions you've previously written compute its

- template description (`fs1` of length `70*70`)
- histogram description (`fs2` of length `nbins=30`)
- grid histogram description (`fs3` of length `ng*ng*nbins=2*2*120`)
- SIFT description (`fs4` of length `128`)

and keep a record of each of these descriptors via a command like

```
Fs{1} = [Fs{1}, fs1];
```

Exercise 2: *Compare histogram feature vectors*

In this function you will compare the feature vector description for each image using the Euclidean distance. This is perhaps not the best distance to use when comparing histograms, but it is the easiest to compute. In case you've forgotten what this distance is; let me remind you. If `f1` and `f2` are vectors of length `nf` then the Euclidean distance between them in *Matlab* syntax is

```
sqrt(sum((f1 - f2).*(f1 - f2)))
```

A small Euclidean distance corresponds to the feature vectors being similar.

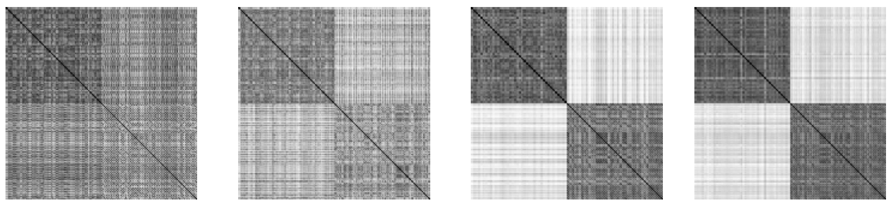
Your task is to write the function

```
function D = ComputeDistanceMatrix(Fs)
```

It will take the matrix of one type of features, `Fs{i}`, you have computed from the downloaded images using `ComputeDescriptors` and compute the Euclidean Distance between every pair of images. This information is then stored in a symmetric matrix D of size `ni`×`ni` where the entry $D(i, j)$ is the Euclidean distance between the feature vector on the `i`th column and the one on the `j`th column. Remember $D(i, j) = D(j, i)$.

You can then plot D using `imagesc`. If the images have been computed in the right order then you should get a D which exhibits some block structure. Note that if `DirName = 'Aligned_Pics/'` then the `D` returned from `ComputeDistanceMatrix(Fs{1})` should look as in figure 4(a).

Write a function which calls `ComputeDistanceMatrix(Fs{i})` for each of the feature types you have extracted and display the distance matrices you compute. Note how the histogram features produce distance matrices with less block structure than those extracted from the SIFT and template features, see figure 4.

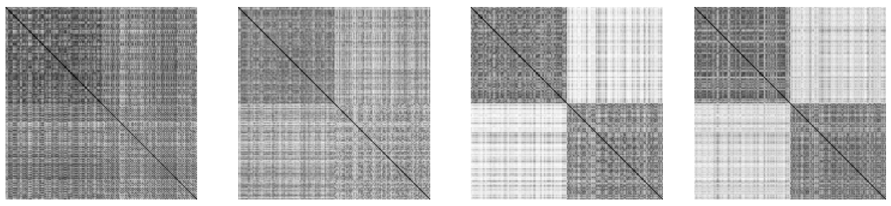


(a) Histogram **(b)** Grid histogram **(c)** Sift **(d)** Template

Figure 4: **Distance matrices computed from the different feature types computed from the *aligned* images.** In this case both the Sift and template descriptors can discriminate between the eye and nose images patches while the histogram features cannot.

Exercise 3: *Repeat process for the misaligned pictures*

Rerun the function `ComputeDescriptors(DirName, nbins, ng)` but this time using the misaligned image patches and also recompute the distance matrices. You should get results similar to those as shown in figure 5.



(a) Histogram **(b)** Grid histogram **(c)** Sift **(d)** Template

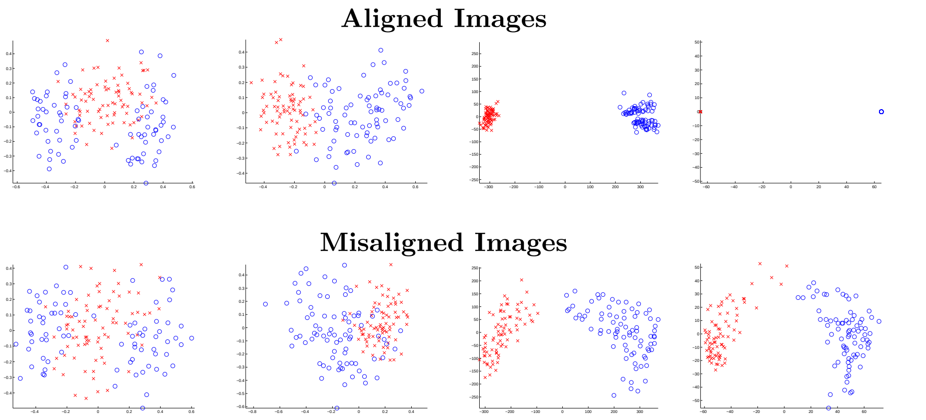
Figure 5: **Distance matrices computed from the different feature types computed from the *misaligned* images.** In this case both the Sift and template descriptors can discriminate between these eye and nose patches but not as well for the aligned patches.

Exercise 4: *Visualize feature vectors (optional)*

There is a method called multi-dimensional scaling which when given an `n`×`n` distance matrix `D`, such as those we have computed, finds `p` dimensional vectors whose pairwise distances produce a distance matrix close to the original one. This allows us to approximately visualize our high dimensional feature vectors in a low dimensional embedding. There is *Matlab* function which performs the multi-dimensional scaling. It is called as follows and one can also plot the embedded feature vectors. The colour coding in the plotting function is dependent on the fact that all the feature vectors from one class are followed by all the vectors from the other class, where `ni` is the number of images examined.

```
>> Y = mdscale(D, 2);
>> figure
>> plot(Y(1:ni/2, 1), Y(1:ni/2, 2), 'rx', 'MarkerSize', 10);
>> hold on
>> plot(Y(ni/2+1:end, 1), Y(ni/2+1:end, 2), 'bo', 'MarkerSize',
10);
>> axis equal
```

Run this code on the different distance matrices you have computed and note how the block structure in the distance matrices is translated to the points from the two different classes being separated in the 2 dimensional embedding, see figure 6. Also notice how the points are more spread out for the misaligned image patches.



(a) Histogram **(b)** Grid histogram **(c)** Sift **(d)** Template

Figure 6: **Two-dimensional points which have the *same* distance matrix as those computed from the different image patch descriptors.** The interesting point is that the template descriptor is perhaps less tolerant to shifts in translation. Also note if you well aligned images and not too much variation in appearance then template matching is as good as you can do.

For the lecture: 27th of March

Bring a print out of

- *the three functions you have written `ExtractGridHistogram`, `ComputeDescriptors` and `ComputeDistanceMatrix` and*
- *the pictures*

- *of the 4 distance matrices for the images in* `Aligned_Pics` *and*
- *of the 4 distance matrix for the images in* `Misaligned_Pics`*.*