Helper Code Snippets

Problem 1

We provide some snippets of code to help you load and operate on images, perform **scanning** and computing **match scores** efficiently.

A. Reading in Images

The images in `problem_1/data/lfw_1000` are all dimension 64×64. The MATLAB command to read an image is:

```
image = double(imread(imagefile));
```

Note the “double()”. If you do not use it, MATLAB reads the data as uint8 and some operations cannot be performed.

B. Building A Matrix of Images

To learn eigen vectors, the collection of faces must be unravelled into a matrix. To unravel an image of any size, you can do the following:

```
[nrows, ncolumns] = size(image);
image = image(:);
```

The first line above is used only to retain the size of the original image. We will need it to fold an unravelled image back into a rectangular image. The second line converts the nrows $\times$ ncolumns image into a single nrows*ncolums x 1 vector. In our “training” data every image is the same size (64 x 64) and no rescaling of images is necessary.

To compose matrix from a collection of k images, the following matlab script can be employed (you can also do it your own way):

```
% Assuming your k images are stored in a cell image$
Y = [];
for i = 1:k
    Y = [Y image{i}(:)];
end
```

C. Computing Eigen faces

Eigen faces can be computed from Y, which is an (nrows*ncolumns) x nimages matrix. There are two ways to to compute eigen faces; one is using eigen analysis and the other is by singular value decomposition.

a. Eigen analysis

First compute the correlation matrix. You can compute eigen vectors and eigen values of the correlation matrix as:

```
corrmatrix = Y * Y';
[eigvecs, eigvals] = eig(corrmatrix);
```

The columns of the `eigenvecs` matrix (in MATLAB notation, the i-th column is given by `eigvecs(:,i)`) are the eigenvectors. The diagonal entries of the `eigvals` matrix are the eigenvalues. You can confirm that `corrmatrix = eigvecs * eigvals * eigvecs'`. To learn more about eigen analysis and the `eig()` command, type “help eig” into MATLAB.

You will find it instructive to plot the Eigen values: `plot(diag(eigvals));`. MATLAB's `eig()` function returns eigenvectors in order of increasing importance. The first eigenvector is the least important. The last one is the most important. The plot of eigen values will also show this -- the last eigenvalue is the largest.

If you have gone through the above exercise, you will quickly realize that it takes a long time to perform eigen analysis: `eig()` of a 4096 x 4096 correlation matrix takes a lot of time. If our images were larger (more pixels), it would take much much longer. One of the reasons is that `eig()` operates on a large correlation matrix. Another it that it computes all eigenvalues and eigenvectors, whereas we know that if we have k images in Y, only k of the Eigenvectors (and Eigenvalues) are relevant -- the others have zero importance.

b. Singular Value Decomposition

The faster way to achieve the same end is via SVD, which can be performed in matlab as

```
[U,S,V] = svd(Y,0);
```

Note that SVD is performed directly on Y and not on the correlation matrix. The “0” to the right states that we want a “thin” SVD. The thin SVD will give us an (nrows*ncolumns) x k matrix U, and a k x k diagonal matrix of singular values, S. The matrix V are the right singular values and we need not concern ourselves with it for this problem.

You can confirm that the k columns of U are the same as the final k columns of the Eigen vectors matrix returned by `eig(corrmatrix)`, only in reverse order. i.e. the first column of U is the most important eigen vector and are **not normalized**, the second column is the second-most important Eigen vector and so on. The diagonal entries of S are also the square roots of the final k diagonal entries of the eigvals matrix returned by `eig()`, only in reverse order.

Therefore, instead of running eigen analysis using `eig(corrmatrix)`, you can get the desired Eigen vectors using `svd(Y,0)` and it will be much faster.

`svd()` still computes k eigen vectors and singular values. For our problem we only want one (or, for Part 2, some K) eigenvectors. An even faster version of SVD is given by the `svds()` command in MATLAB which only computes exactly as many Eigenvectors as you want. You can learn more about these by typing `help svd` or `help svds` into MATLAB.

The eigenface will be in the form of a nrows*ncolumns x 1 vector. To convert it to an image, you must fold it into a rectangle of the right size. Matlab will do it for you with:

```
eigenfaceimage = reshape(eigenfacevector, nrows, ncolumns);
```

`nrows` and `ncolumns` are the values obtained when you read the image. eigenfacevector is the eigen vector obtained from the eigen analysis (or SVD).

D. Scanning an Image or Pattern

Let I be a P x Q image. Let E be an N x M Eigen face. The following MATLAB loop will compute the normalized dot product of every N x M patch of I as follows:

```
E = E/norm(E(:));
for i = 1:(P-N)
```

```
    for j = 1:(Q-M)
        patch = I(i:i+N-1,j:j+M-1);
        m(i,j) = E(:)'*patch(:) / norm(patch(:));
    end
end
```

`m(i,j)` is the normalized dot product, which represents the match between the eigen face E and the N x M patch of the image with its upper left corner at the (i,j)-th pixel. Note that we are normalizing both the eigen face and the patch in the above code. This will make sure that the results are invariant to both the size and lighting of the images.

E. Some Processing Tricks for Faster Results

I. Converting Color to Greyscale:

A few of the test images you have been given are a color image. This must be converted to grey scale. To do this, first read the image in as a color image. You can do this imply using `imread()`, just as you would read a greyscale image. `imread()` will give you three images, stored together in a three-dimensional array (the elements of which must be accessed using three indicies). So, for example, `image(i,j,1)` will give you the (i,j)-th pixel of the red image, `image(i,j,2)` will give you the corresponding green pixel and `image(i,j,3)` the blue pixel. To convert it to a greyscale image, the red, green and blue images must be added. This can be done by:

```
greyscaleimage = squeeze(mean(colorimage,3));
```

This sums the third dimension of the three-dimensional matrix that represents the color image to give you a greyscale image. We're actually using "mean" instead of "sum" to make sure everything is in the 8-bit range. Note the `squeeze()`. The result of sum will still have three indices (although the final index only takes the value 1), `squeeze()` eliminates the third index.

II. Scaling Images

Any P x Q image can be resized to an N x M image using the `imresize()` command as follows:

```
scaledimage = imresize(image,[N,M]);
```

Note that the above command does not consider the size of the original image explicitly. If N and M are not proportional to P and Q respectively, the image will get distorted. To ensure that the image retains its proportions, make sure that N/M = P/Q. To rescale the 64x64 Eigen face to 128x128, the command is:

```
neweigenface = imresize(eigenface,[128,128])
```

III. Viewing Images

You can view an image simply by `imagesc(image)`.

IV. Convolutions for **scanning**

Computation of the dot product of the eigenface with each patch of the test image is actually a convolution. This can be done really quickly using the convolution operator in MATLAB. Note the `flipud(fliplr())` operations here. This is required because convolution actually flips the eigenface prior to computing the dotproduct. By pre-flipping the eigenface, we're accounting for this.

```
tmpim = conv2(A, fliplr(flipud(E)));
convolvedimage = tmpim(N:end, M:end);
```

The result above has not accounted for the fact that we wanted to subtract out the mean of each patch. We will do that now:

```
sumE = sum(E(:));
patchscore = convolvedimage - sumE*patchmeanofA;
```

Problem 2 and 3

1. Computing the spectrogram

`problem_2/stft.m` computes the complex spectrogram of a signal. You can read a .wav file into MATLAB as follows:

```
[s,fs] = audioread('filename');
s = resample(s,16000,fs);
```

Above, we resample the signal to a standard sampling rate for convenience. Next, we can compute the complex short-time Fourier transform of the signal, and the magnitude spectrogram from it, as follows. Here we use 2048 sample windows, which correspond to 64ms analysis windows. Adjacent frames are shifted by 256 samples, so we get 64 frames/second of signal.

To compute the spectrogram of a recording, e.g. the music, perform the following.

```
spectrum = stft(s',2048,256,0,hann(2048));
music = abs(spectrum);
sphase = spectrum ./(abs(spectrum)+eps);
```

This will result in a 1025-dimensional (rows) spectrogram, with 64 frames(columns) per second of signal.

Note that we are also storing the phase of the original signal. We will need it later for reconstructing the signal. We explain how to do this later. The eps in this formula ensures that the denominator does not go to zero.

You can compute the spectra for the notes as follows. The following script reads the directory and computes spectra for all notes in it.

```
notesfolder = 'problem_2/data/notes15/';
listname = dir([notesfolder '*.wav']);
notes = [];
for k =1:length(listname)
    [s,fs] = audioread([notesfolder listname(k).name]);
    s = resample(s,16000,fs);
    spectrum = stft(s',2048,256,0,hann(2048));
    % Find the central frame
    middle = ceil(size(spectrum,2)/2);

    note = abs(spectrum(:,middle));
    % Clean up everything more than 40db below the peak
    note(find(note < max(note(:))/100)) = 0;
    note = note/norm(note); %normalize the note to unit length

    notes = [notes,note];
end
```

The “notes” matrix will have as many columns as there are notes (15 in our data). Each column represents a note. The notes will each be represented by a 1025 dimensional column vector.

2. Reconstructing a signal from a spectrogram

The recordings of the complete music can be read just as you read the notes. To convert it to a spectrogram, do the following. Let reconstructedmagnitude be the reconstructed magnitude spectrogram from which you want to compute a signal. In our homework we get many variants of this. To recover the signal we will use the `sphase` we computed earlier from the original signal.

```
reconstructedsignal = stft(reconstructedmagnitude.*sphase,2048,256,0,hann(2048));
```