CS 231A Computer Vision (Autumn 2012)
Problem Set 1

Solution Set

Due: Oct. 9th, 2012 (2:15 pm)

1 Finding an Approximate Image Basis – EigenFaces (25 points)

In this problem you will implement a solution to a facial recognition task. Specifically, you will determine whether a given image contains a face that belongs to a given set of people. The method you will explore is discussed in Lecture 2, and is known as EigenFaces. EigenFaces relies upon the idea of Principle Component Analysis (PCA).

In this problem, we have high dimensional data and believe that valuable information can be represented in a lower dimensional space. This dimensionality reduction is a typical application of PCA. We will start by representing our data as a line and then decrease the dimensions of our representation.

The images in this problem are in the form of vectors, where $x^{(i)}$ denotes the vector corresponding to the image i. Our representation will consist of a set of linearly independent vectors $a^{(i)}$. The set of vectors, $a^{(i)}$ is known as the facespace. Given a test image as a vector y, we will project it onto the range of the facespace. For our projection, we aim to find a set of weights w to solve

$$\text{min. } \|Aw - y\|_2^2$$

where $A = \left[a^{(1)} \dots a^{(n)}\right]$, or in other words $a^{(i)}$ are the columns of A. Our projection is given by $y_{proj} = Aw_{opt}$. Once we have this projection, we can determine whether y is in our given set of people.

First consider the approximation to our high dimensional data by a line set, L, where $a \in \mathbb{R}^n$, $b \in \mathbb{R}^n$ and a is a unit vector. The set

$$L = \{\ az + b \mid z \in \mathbb{R}\}$$

is a line in $\mathbb{R}^n$. For any vector $x \in \mathbb{R}^n$ the distance between x and L is defined by

$$d(x, L) = \min_{y \in L} \|x - y\|_2$$

where $\|x\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2}$ (l_2 norm). We have m vectors, and we would like to find the line L that minimizes the total squared distance

$$d_t = \sum_{i=1}^m d(x^{(i)}, L)^2$$

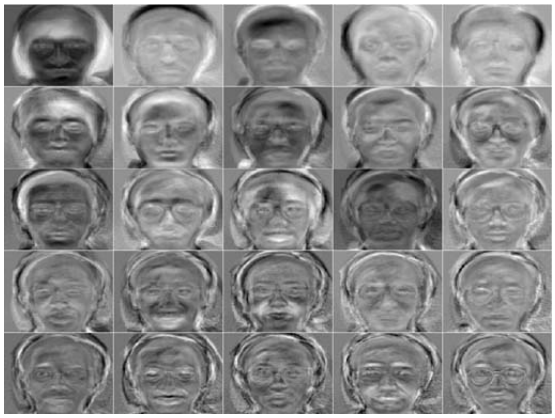

Figure 1: Example facespace

In other words we'd like to fit the points to a straight line, as close as possible such that d_t is minimized.

(a) Show that $d(x, L) = \|(I - aa^T)(x - b)\|_2$. *Hint: you might consider using a least squares approach*

(b) Using the result from (a), the total distance squared is given by

$$d_t = \sum_{i=1}^{m} \|(I - aa^T)(x^{(i)} - b)\|_2^2$$

Show that the optimal b, which minimizes d_t while all other variables are held constant, is given by

$$b_{\text{opt}} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

That is b_{opt} is the mean of the vectors $x^{(i)}$. *Hint:* $P = (I - aa^T)$ *is a projection matrix.*

(c) Using this choice of $b = b_{opt}$, find the optimal value of a which minimizes d_t, in terms of $x^{(i)}$ and b_{opt}? *Hint: Try to formulate the problem as a maximization problem of the form* $\max(a^T \Sigma a)$.

The remaining parts of this problem require programming. We have provided skeleton code in `PS1_data.zip` in the folder `eigenFaces`. The provided code can be executed using Matlab on the computing clusters/personal machines. The appropriate `.m` files have been commented to indicate where your code should be placed and the variables provided by us. All your code for this problem should be located in `eigenface.m`.

(d) Run the script `readYaleFaces`. This will load a matrix A, whose columns are the reshaped images of the data set, into the Matlab workspace. Find the mean of these images (as in the Lecture 2 notes) and subtract it from all the images to form a matrix B (of same dimension as A). Write code that finds the r largest singular values and the corresponding left singular vectors of B. These singular vectors are the eigenfaces corresponding to our data set. Turn in the 5 largest singular values. Please put your code is the m-file `eigenface.m`. *Note: that this is a big matrix, and just typing* `svd` *may not do what you want. Check the help for the* `svd` *command in Matlab.*

(e) For facial recognition, we project the given image onto the facespace. If the resulting error is less than a given threshold, the image is recognized as belonging to the facespace. For this part use 25 eigenfaces, *i.e.* choose $r = 25$ above. You are given twenty images, i.e. `image1.mat` ... `image20.mat`. Images `image1.mat`, `image3.mat`, `image6.mat`, `image12.mat`, and `image14.mat` are not in the facespace. To determine if an image y is in the facespace we use a threshold τ such that if $\|y - y_{proj}\|_2^2 \geq \tau$ the given image is not in the facespace and if $\|y - y_{proj}\|_2^2 < \tau$ the given image is in the facespace. This threshold τ should maximize the number of correct classifications. To find a value for τ use the following threshold values and report the value which yields the maximum correct classifications $\tau = \{0.009\text{E}8,\ 0.09\text{E}8,\ 0.9\text{E}8,\ 9\text{E}8\}$. Also report the number of correct detections this optimal value of τ achieves. In the skeleton code there is code that will calculate the number of correct detections for you.

Solution:

(a) We have

$$d = \min_z \|az + b - x\|_2$$

and so the least squares solution is

$$z = (a^T a)^{-1} a^T (x - b) = a^T (x - b)$$

substituting back in gives the result.

(b) Letting $P = (I - aa^T)$, we have

$$\begin{aligned} d_t &= \sum_i \|P(x^{(i)} - b)\|^2 \\ &= \sum_i (x^{(i)} - b)^T P^T P (x^{(i)} - b) \\ &= \sum_i (x^{(i)} - b)^T P (x^{(i)} - b) \end{aligned}$$

notice that P is a projection matrix, thus $P^T = P$ and $PP = P$. To find b_{opt} find the gradient with respect to b denoted by ∇_b.

$$\begin{aligned} \nabla_b\ d_t &= \sum_i \nabla_b (x^{(i)T} P x^{(i)} - 2x^{(i)T} P b + b^T P b) \\ &= \sum_i (2Pb - 2Px^{(i)}) \\ &= P \sum_i (b - x^{(i)}) \\ \Rightarrow b_{opt} &= \frac{1}{m} \sum_i x^{(i)} \end{aligned}$$

(c) The optimal a can be chosen as follows

$$\begin{aligned} d_t &= \sum_{i=1}^{m} \|(I - aa^T)(x^{(i)} - b_{\text{opt}})\|^2 \\ &= \sum_{i=1}^{m} (x^{(i)} - b_{\text{opt}})^T (I - aa^T)(x^{(i)} - b_{\text{opt}}) \\ &= \sum_{i=1}^{m} \|x^{(i)} - b_{\text{opt}}\|^2 - \sum_{i=1}^{m} \left((x^{(i)} - b_{\text{opt}})^T a \right)^2 \\ &= \sum_{i=1}^{m} \|x^{(i)} - b_{\text{opt}}\|^2 - a^T \Sigma a \end{aligned}$$

where

$$\Sigma = \sum_{i=1}^{m} (x^{(i)} - b_{\text{opt}})(x^{(i)} - b_{\text{opt}})^T$$

Since the first term is constant, the optimal a can be chosen so as to maximize $a^T \Sigma a$. Thus, the optimal a is the unit norm eigenvector corresponding to $\lambda_{max}(\Sigma)$.

(d) See matlab code

```
First 5 Singular values are along the diagonal (Old solution)
ans =

   1.0e+05 *

    1.2663         0         0         0         0
         0    0.8434         0         0         0
         0         0    0.6743         0         0
         0         0         0    0.6096         0
         0         0         0         0    0.4901
```

(e) The accuracy is 19/20, which is obtained using $\tau = 0.9\text{E}8$

Note: For parts (d) and (e), depending on which machines you ran on and how you ran the svd command, it is possible to attain different singular values and accuracies. We did not deduct points if your code was right.

Matlab Code

```
% CS 231A
% Problem Set 1
% Problem 1

% load data
```

```
readYaleFaces;

% part d -- find the mean
mean_image = mean(A,2);

% subtract the mean image from all the images
B = zeros(size(A,1),size(A,2));
for i = 1:size(A,2)
    B(:,i) = A(:,i) - mean_image;
end

% % form a smaller matrix to take the eigenvalues of
% B_sm = B'*B;
% [V_sm,D_sm] = eig(B_sm);
% % arrange the singular values in decreasing order
% D = zeros(size(D_sm,1),size(D_sm,2));
% V = zeros(size(V_sm,1),size(V_sm,2));
% for i = 1:size(D_sm,1)
%     V(:,size(V_sm,1)+1-i) = V_sm(:,i);
%     D(size(D_sm,1)+1-i,size(D_sm,1)+1-i) = sqrt(D_sm(i,i));
% end
% % the matrix B is not full rank, thus we need to make
% % the matrices for the
% % svd thin
% D_hat = D;
% V_hat = V;
% find the matrix U of the left singular vectors
%to be used for eigenfaces
%U_hat = B*pinv(D_hat*V_hat');

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

% report first 5 singular values and the coressponding 5
fprintf('First 5 Singular values are along the diagonal')
S(1:5,1:5)


% part e -- determine accuracy and plot projections
clear e
e = zeros(size(image1,1)*size(image1,2),1);
x_proj = zeros(size(image1,1)*size(image1,2),1);

for i = 1:20
    clear x
    testFileName = ['image' num2str(i)];
    x = eval(testFileName);
    x = double(reshape(x,size(x,1)*size(x,2),1));
```

```
    r = 25;
    V_r = U(:,1:r);
    alpha_opt = pinv(V_r)*(x-mean_image);
    x_proj(:,i) = mean_image + V_r*alpha_opt;
    e(:,i) = x - x_proj(:,i);
    norm_e(i) = e(:,i)'*e(:,i);
end

numGndTrth = 20;
gnd_truth = ones(numGndTrth,1);
gnd_truth(1) = 0;
gnd_truth(3) = 0;
gnd_truth(6) = 0;
gnd_truth(12) = 0;
gnd_truth(14) = 0;

thres = 10^8*[.009 .09 .9 9];
num_correct = 0;
for i = 1:length(thres)
    for j = 1:length(gnd_truth)
        if ( norm_e(j) < thres(i) && gnd_truth(j) == 1)
            num_correct = num_correct + 1;
        elseif (norm_e(j) > thres(i) && gnd_truth(j) == 0)
            num_correct = num_correct + 1;
        end
    end
    num_correct
    num_correct = 0;
end
```

2 Steerable filters (25 points)

Images can often be characterized by an aggregation of local information contained in the derivative or gradient. State of the art image descriptors often rely on such local gradients. In this problem, you will derive one local operator, steerable filters, which provides the directional derivative of an image. Due to the efficiency of local operators they are often used in real-time computer vision tasks such as motion estimation and face recognition. Steerable filters also provide the flexibility to look for derivatives you might be expecting in a specific direction.

Let $G^0(x, y)$ be some 2-dimension Linear Shift Invariant (LSI) filter, a function of the cartesian coordinates x and y. Let $G^\theta(x, y)$ be a rotation of $G^0(x, y)$ by θ radians about the origin in the counter-clockwise direction.

(a) Show that

$$G^\theta(x, y) = G^0(r\cos(\phi - \theta), r\sin(\phi - \theta))$$

where $r = \sqrt{x^2 + y^2}$ and $\tan\phi = y/x$.

(b) Using the fact that $G^\theta(x,y)$ can be written as

$$G^\theta(x,y) = G^0\left(r\cos(\phi-\theta), r\sin(\phi-\theta)\right)$$

write an expression for $G^\theta(x,y)$ in terms of r, θ, ϕ, given that $G^0(x,y) = -2xe^{-(x^2+y^2)}$.

Let $F^\theta(x,y) = I(x,y) \star G^\theta(x,y)$ where $\star$ denotes convolution. Using your expression for $G^\theta(x,y)$ show that $F^\theta(x,y) = a(I(x,y) \star G^0(x,y)) + b(I(x,y) \star G^{\pi/2}(x,y))$ where $a, b \in \mathbb{R}$ i.e. $F^\theta(x,y)$ can be written as a linear combination of $I(x,y) \star G^0(x,y)$ and $I(x,y) \star G^{\pi/2}(x,y)$. State the value of a, b explicitly.

(c) Find the direction of maximum response at a point (x,y) of the image $I(x,y)$ to the steerable filter $G^\theta(x,y)$. The direction of maximum response is the θ, such that $F^\theta(x,y)$ has the largest magnitude. Give your answer in terms of the image $I(x,y)$ and the responses $F^0(x,y)$, $F^{\pi/2}(x,y)$ to the two steerable basis filters $G^0(x,y)$, $G^{\pi/2}(x,y)$.

Remark: Once this maximum response is found you can use it to steer your filter $G^\theta(x,y)$ such that it will produce a large response when an image similar to the original $I(x,y)$ passes through the filter.

Solution:

(a) We have two sets of coordinate systems centered at the same origin. The first coordinate system is described by the cartesian coordinates (x,y) and corresponds to $G^\theta(x,y)$, the second is described by the cartesian coordinates (x',y') and corresponds to $G^0(x,y)$. We can express the first coordinate system in terms of its polar coordinates

$$\begin{aligned} x &= r\cos\phi \\ y &= r\sin\phi \end{aligned}$$

where $r = \sqrt{x^2+y^2}$ and $\phi = \arctan(y/x)$. Since the two coordinate systems share the same origin and the (x',y') coordinate system is a clock-wise rotated version of the (x,y) coordinate system we can write

$$\begin{aligned} x' &= r\cos(\phi-\theta) \\ y' &= r\sin(\phi-\theta) \end{aligned}$$

Thus we have

$$G^\theta(x,y) = G^0\left(r\cos(\phi-\theta), r\sin(\phi-\theta)\right)$$

(b) Starting with

$$G^0(x,y) = -2xe^{-(x^2+y^2)}$$

and using part (a) we can write

$$\begin{aligned} G^\theta(x,y) &= G^0(r\cos(\phi-\theta), r\sin(\phi-\theta)) \\ &= -2(r\cos(\phi-\theta))e^{-(x^2+y^2)} \end{aligned}$$

Recalling that $r^2 = x^2 + y^2$ and the trigonometric identity $\cos(\phi - \theta) = \cos\phi\cos\theta + \sin\phi\sin\theta$ we can write

$$\begin{aligned} G^\theta(x,y) &= -2r\cos\phi\cos\theta e^{-r^2} - 2r\sin\phi\sin\theta e^{-r^2} \\ &= -2x\cos\theta e^{-r^2} - 2y\sin\theta e^{-r^2} \end{aligned}$$

where we use the fact that

$$\begin{aligned} r\cos\phi &= x \\ r\sin\phi &= y \end{aligned}$$

realizing that

$$G^{\frac{\pi}{2}}(x,y) = -2ye^{-r^2}$$

we reach the linear combination that was asked for

$$G^\theta(x,y) = \cos\theta G^0(x,y) + \sin\theta G^{\frac{\pi}{2}}(x,y)$$

Given an input image $I(x,y)$ we can write the response $F^\theta(x,y)$ to the filter $G^\theta(x,y)$ as

$$\begin{aligned} F^\theta(x,y) &= I(x,y) \star G^\theta(x,y) \\ &= I(x,y) \star \left[\cos\theta G^0(x,y) + \sin\theta G^{\frac{\pi}{2}}(x,y)\right] \\ &= \cos\theta I(x,y) \star G^0(x,y) + \sin\theta I(x,y) \star G^{\frac{\pi}{2}}(x,y) \end{aligned}$$

where the last line follows from the linearity property of convolution.

(c) To find direction of maximum response at a given point (x,y) we take the partial derivative of $F^\theta(x,y)$ with respect to θ and set it equal to zero.

$$\begin{aligned} \frac{\partial}{\partial\theta}F^\theta(x,y) &= \frac{\partial}{\partial\theta}\cos\theta F^0(x,y) + \frac{\partial}{\partial\theta}\sin\theta F^{\frac{\pi}{2}}(x,y) && = 0 \\ &= -\sin\theta F^0(x,y) + \cos\theta F^{\frac{\pi}{2}}(x,y) && = 0 \end{aligned}$$

solving for θ we find

$$\begin{aligned} \theta_{max} &= \arctan\left(\frac{F^{\frac{\pi}{2}}(x,y)}{F^0(x,y)}\right) \\ &= \arctan\left(\frac{I(x,y) \star G^{\frac{\pi}{2}}(x,y)}{I(x,y) \star G^0(x,y)}\right) \end{aligned}$$

3 Digital Matting (25 points)

In digital matting, a foreground element is extracted from a background image by estimating a color and opacity for the foreground element at each pixel. Thus we can think of digital matting as a form of background subtraction. The opacity value at each pixel is typically called its α. Matting is used in order to composite the foreground element into a new scene. Matting

and compositing were originally developed for film and video production. The most common compositing operation is the over operation, which is summarized by the compositing equation

$$C = \alpha F + (1 - \alpha)B$$

where $C \in \mathbb{R}^3$, $F \in \mathbb{R}^3$, and $B \in \mathbb{R}^3$ are the pixel's composite, foreground, and background color vectors , respectively, and $\alpha \in \mathbb{R}$ is the pixel's opacity component used to linearly blend between foreground and background.

For the development that follows, we will assume that our input image has already been segmented into three regions: background, foreground, and unknown, with the background and foreground regions having been delineated conservatively. The goal then, is using local image statistics to solve for the foreground color F, background color B, and opacity α given the observed color C for each pixel within the unknown region of the image.

The method you will derive uses a continuously sliding window for local neighborhood definitions, marches inward from the foreground and background regions, and utilizes nearby computed F, B, and α values (in addition to these values from known regions) in constructing oriented Gaussian distributions. Using a statistical framework we will leverage these distributions to improve the accuracy of the foreground and background within the initially unknown region. Specifically we will use maximum a posteriori (MAP) estimation to improve our foreground and background segmentation. The MAP estimate is given by

$$\arg\max_{F,B,\alpha} \; P(F, B, \alpha \mid C)$$

Remark : If you need a refresher on MAP estimation, log likelihood functions, and/or maximum likelihood estimation you can refer to Chapter 3 of Pattern Classification Duda, Hart, and Stork which we have posted on the course webpage. We also gave a brief introduction to maximum likelihood estimation in Problem Set 0 Problem 5.

(a) Generally we don't know the distribution $P(F, B, \alpha \mid C)$. In this problem we will use the simplifying assumption that F, B and α are independent. Using this assumption rewrite the MAP estimate as the sum of log-likelihood functions, $\arg\max_{F,B,\alpha} \ell(F, B, \alpha \mid C) =$?. Make sure there are no terms in your solution that are probabilities given C.

(b) We can model the log-likelihood function of our observation C as

$$\ell(C \mid F, B, \alpha) = -\frac{\|C - \alpha F - (1 - \alpha)B\|^2}{\sigma_C^2}$$

(a) Input (b) Opacity α (c) Composite

Figure 2: Example of digital matting

where we know σ_C. Also we model the log-likelihood function of foreground color F as

$$\ell(F) = -(F - \bar{F})^T \Sigma_F^{-1} (F - \bar{F})$$

where for the purposes of this problem you can assume $\bar{F}$ and Σ_F have already been computed. The log-likelihood for the background B is

$$\ell(B) = -(B - \bar{B})^T \Sigma_B^{-1} (B - \bar{B})$$

again, for the purposes of this problem you can assume $\bar{B}$ and Σ_B have already been computed. Also, Σ_F^{-1} and Σ_B^{-1} are symmetric matrices. To solve for the MAP estimate, we need to split this problem into two sub-problems. To solve the first sub-problem we hold α constant while optimizing $\ell(F, B, \alpha)$ over F and B. For this part of the problem hold α constant and show that the optimal F and B are given by the linear equation

$$\begin{bmatrix} \Sigma_F^{-1} + I\alpha^2/\sigma_C^2 & I\alpha(1-\alpha)/\sigma_C^2 \\ I\alpha(1-\alpha)/\sigma_C^2 & \Sigma_B^{-1} + I(1-\alpha)^2/\sigma_C^2 \end{bmatrix} \begin{bmatrix} F \\ B \end{bmatrix} = \begin{bmatrix} \Sigma_F^{-1}\bar{F} + C\alpha/\sigma_C^2 \\ \Sigma_B^{-1}\bar{B} + C(1-\alpha)/\sigma_C^2 \end{bmatrix}$$

(c) Now, we solve the second sub-problem by optimizing $\ell(F, B, \alpha)$ over α while holding F and B constant. In this problem we assume that $\ell(\alpha)$ is a constant. Show that, under this assumption, the optimal α is given by

$$\alpha = \frac{(B - F)^T (B - C)}{\|B - F\|^2}$$

Remark: To obtain the final estimates for F,B and α in this problem we must iterate between these two solutions.

Solution:

(a) Using Bayes' rule we can rewrite our MAP problem as

$$\arg\max_{F,B,\alpha} P(F, B, \alpha \mid C) = \arg\max_{F,B,\alpha} \frac{P(C \mid F, B, \alpha) P(F) P(B) P(\alpha)}{P(C)}$$

we can rewrite this as a sum by taking the log of the right hand side of the above equation. Since log is monotonically increasing we will find the same optimal values for our variables. The result is

$$\arg\max_{F,B,\alpha} \ell(F, B, \alpha) = \arg\max_{F,B,\alpha} \ell(C \mid F, B, \alpha) + \ell(F) + \ell(B) + \ell(\alpha)$$

(b) Plugging our distributions into our log-likelihood function from (b) (we drop the variables in ℓ for notational brevity) we get

$$\begin{aligned} \ell(\ldots) &= -\frac{\|C - \alpha F - (1-\alpha)B\|^2}{\sigma_C^2} - (F - \bar{F})^T \Sigma_F^{-1} (F - \bar{F}) - (B - \bar{B})^T \Sigma_B^{-1} (B - \bar{B}) \\ &= -\frac{(C - \alpha F - (1-\alpha)B)^T (C - \alpha F - (1-\alpha)B)}{\sigma_C^2} \quad \cdots \\ &\quad - (F - \bar{F})^T \Sigma_F^{-1} (F - \bar{F}) - (B - \bar{B})^T \Sigma_B^{-1} (B - \bar{B}) \end{aligned}$$

Now we will solve the first sub problem by taking the partials of $\ell(\dots)$ with respect to F and B and setting them equal to zero. We start by finding the partial with respect to F of the first term in the above equation

$$
\begin{aligned}
&- \nabla_F \frac{(C - \alpha F - (1-\alpha)B)^T (C - \alpha F - (1-\alpha)B)}{\sigma_C^2} \\
&= \nabla_F \frac{(-C^T + \alpha F^T + (1-\alpha)B^T)(C - \alpha F - (1-\alpha)B)}{\sigma_C^2} \\
&= \nabla_F \frac{-C^T C + 2\alpha C^T F + 2(1-\alpha)C^T B - \alpha^2 F^T F - 2\alpha(1-\alpha)F^T B - (1-\alpha)^2 B^T B}{\sigma_C^2} \\
&= \frac{2\alpha C - 2\alpha^2 F - 2\alpha(1-\alpha)B}{\sigma_C^2}
\end{aligned}
$$

We now find the partial of the second term with respect to F

$$
\begin{aligned}
&- \nabla_F \left((F - \bar{F})^T \Sigma_F^{-1} (F - \bar{F}) \right) \\
&= -\nabla_F \left(F^T \Sigma_F^{-1} F - 2\bar{F}^T \Sigma_F^{-1} F + \bar{F}^T \Sigma_F^{-1} \bar{F} \right) \\
&= -2\Sigma_F^{-1} F + 2\Sigma_F^{-1} \bar{F}
\end{aligned}
$$

The third term in $\ell(\dots)$ depends solely on B and thus goes to zero when taking the partial with respect to F. Thus combining all the partials, setting them to zero and rearranging we have the following equation

$$(I\alpha^2/\sigma_C^2 + \Sigma_F^{-1})F + \alpha(1-\alpha)B/\sigma_C^2 = \Sigma_F^{-1}\bar{F} + \alpha C/\sigma_C^2$$

We now take the partial of $\ell(\dots)$ with respect to B starting with the first term of $\ell(\dots)$ we have

$$
\begin{aligned}
&\nabla_B \frac{-C^T C + 2\alpha C^T F + 2(1-\alpha)C^T B - \alpha^2 F^T F - 2\alpha(1-\alpha)F^T B - (1-\alpha)^2 B^T B}{\sigma_C^2} \\
&= \frac{2(1-\alpha)C - 2\alpha(1-\alpha)F - 2(1-\alpha)^2 B}{\sigma_C^2}
\end{aligned}
$$

When evaluating the partial with respect to B the second term goes to zero and the third term is the same as the partial of the second term when evaluating with respect to F only replacing the F with B yielding

$$-2\Sigma_B^{-1} B + 2\Sigma_B^{-1}\bar{B}$$

Combining all the partials, setting them to zero and rearranging we have the following equation

$$(1-\alpha)C/\sigma_C^2 + \Sigma_B^{-1}\bar{B} = \left((1-\alpha)^2 I/\sigma_C^2 + \Sigma_B^{-1}\right) B + \alpha(1-\alpha)F/\sigma_C^2$$

putting this in a single linear equation we have

$$
\begin{bmatrix} \Sigma_F^{-1} + I\alpha^2/\sigma_C^2 & I\alpha(1-\alpha)/\sigma_C^2 \\ I\alpha(1-\alpha)/\sigma_C^2 & \Sigma_B^{-1} + I(1-\alpha)^2/\sigma_C^2 \end{bmatrix} \begin{bmatrix} F \\ B \end{bmatrix} = \begin{bmatrix} \Sigma_F^{-1}\bar{F} + C\alpha/\sigma_C^2 \\ \Sigma_B^{-1}\bar{B} + C(1-\alpha)/\sigma_C^2 \end{bmatrix}
$$

(c) Considering the second sub problem we first note that the only term to be optimized here is the first term in $\ell(\ldots)$ this gives us the following optimization problem where α is the variable we are optimizing over

$$\text{maximize} \qquad -\frac{\|C - \alpha F - (1-\alpha)B\|^2}{\sigma_C^2}$$

we can convert this to an equivalent optimization problem over α

$$\text{minimize} \qquad \|C - \alpha F - (1-\alpha)B\|^2$$

rearranging

$$\text{minimize} \qquad \|(B-F)\alpha - (B-C)\|^2$$

noticing this is a least squares problem and assuming that $(B-F)$ is skinny and full rank the optimal α is given by

$$\alpha = \left[(B-F)^T(B-F)\right]^{-1}(B-F)^T(B-C)$$

because B and F are vectors we can write

$$\alpha = \frac{(B-F)^T(B-C)}{\|B-F\|^2}$$

4 Content-Aware Image Resizing (25 points)

For this exercise, you will implement a version of the content-aware image resizing technique described in Shai Avidan and Ariel Shamirs SIGGRAPH 2007 paper, Seam Carving for Content-Aware Image Resizing. In this problem all the code and data can be found in `PS1_data.zip` in the `seamCarving` folder. First read through the paper, with emphasis on sections 3, 4.1, and 4.3.

(a) Write a Matlab function which computes the energy of an image, where Energy is defined to be

$$e(x,y) = \sum_{c \in \{R,G,B\}} \left|\frac{\partial I_c(x,y)}{\partial x}\right| + \left|\frac{\partial I_c(x,y)}{\partial y}\right|$$

where $I_c(x,y)$ is the pixel in color channel c located at (x,y). This function should be written in `computeEnergy.m`, take an RGB image in and return a 2D array of equal size which contains the derivative. In this case, we will use Sobel filters to take the horizontal and vertical derivatives, then create our energy from the sum of the absolute value. The sobel filters are defined as:

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Also we have included a sample image `parta_sampleinput.jpg`, and the output of our version of `computeEnergy.m`, `parta_sampleoutput.mat` to help you debug your code. You can use the `testEnergy.m` to show the difference between your output and our output. *Hint: you may find the Matlab function* `filter2` *useful.*

(b) The energy of the optimal seam is defined in Equation 4 of Avidan et al as:

$$s^* = \min_z \sum_{i=1}^{n} e(\mathbf{I}(s_i))$$

where $\mathbf{I}(si)$ are the pixels of a path of a seam. In this problem we will compare two different methods of solving this problem. Turn in your code for both sub problems.

(i.) Implement a greedy algorithm for finding the optimal seam by sequentially adding each pixel in the path depending on the energies of immediate options. Use the skeleton code found in `findSeam_Greedy.m` to write your code. We have given you our seam carving results of the greedy algorithm on `stanford.jpg`, as `stanford_resize_gdy.jpg` to help you debug your final seam carving code in part(c).

(ii.) Implement the dynamic programming solution given in the paper with the following update step:

$$M(i,j) = e(i,j) + \min(M(i-1,j-1), M(i-1,j), M(i-1,j+1))$$

Where M is the cumulative energy function, as mentioned in the paper. Use the skeleton code found in `findSeam_Dyn.m` to help you write your code. The results of the dynamic programming is provided as `stanford_resize_dyn.jpg`. You can use it to debug your final code in part(c).

(c) Use the skeleton code provided in `reduceWidth.m` to combine your parts (a) and (b) into a functional seam carving application.

(d) Now test your code by running the following tests using both the dynamic programming and greedy techniques. Submit all images generated from this part:

```
reduceWidth('stanford.jpg', 200)
reduceWidth('dogs.jpg', 200)
reduceWidth('pool.jpg', 200)
```

We have given you our results on the `stanford.jpg` image as `stanford_resize_dyn.jpg` and `stanford_resize_gdy.jpg` for the dynamic programing and greedy algorithms respectively to help you debug your code. Comment on where the inferiority of the greedy methodology is most apparent and how this is solved by dynamic programming. Also

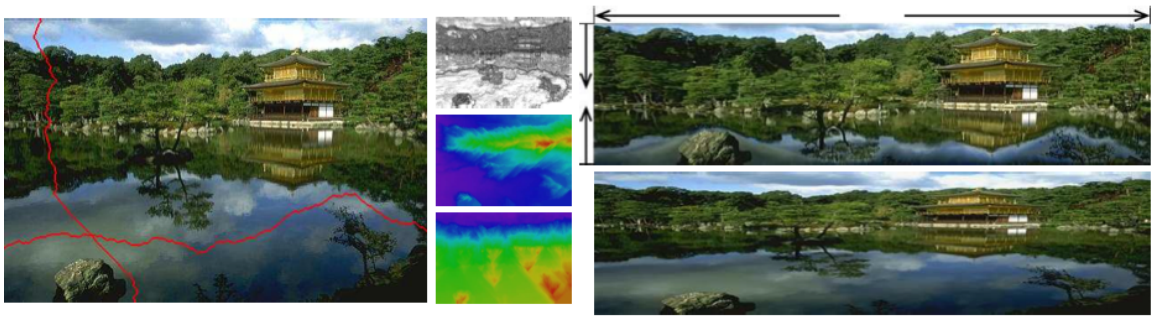

Figure 3: Example of content-aware image resize compared to standard rescale

comment on what types of images are sensitive to this manipulation and which images can have many lines removed without a perceptible difference. Find 2 images of your own and run experiments to show both a good and bad performance using the dynamic programming approach.

Solution:

This is our code for the `ComputeEnergy.m` function

```
function E = ComputeEnergy_sol(img)
    E = zeros(size(img,1),size(img,2));
    G_x = [1 0 -1; 2 0 -2; 1 0 -1];
    G_y = [1 2 1; 0 0 0; -1 -2 -1];

    for color = 1:3
       E = E + abs(filter2(G_x, img(:,:,color))) + abs(filter2(G_y, img(:,:,color)));
    end

end
```

This our code for the `findSeam_Greedy.m` function

```
function pix = findSeam_Greedy_sol(energy)

Pot = energy;
pix = [];
pix(1) = find(Pot(1,:) == min(Pot(1,:)),1);

for ii=2:size(Pot,1)
    %create list of next three options
    options = [ Pot(ii,pix(ii-1)) ];
    if pix(ii-1) == 1
        options = [ options 10000000 ];
    else
        options = [ options Pot(ii,pix(ii-1)-1) ];
    end
    if pix(ii-1) == size(Pot,2)
        options = [ options 10000000 ];
    else
        options = [ options Pot(ii,pix(ii-1)+1) ];
    end
    idx = find( options == min(options) );
    if idx ==1 %The lowest energy step is straight down
    pix(ii) = pix(ii-1);
    elseif idx ==2 %The lowest energy step is to the left
        pix(ii) = pix(ii-1) -1;
```

```
    else %The lowest energy step is to the right
        pix(ii) = pix(ii-1) +1;
    end

end
```

This is our code for the `findSeam_Dyn_sol.m` function

```
function pix = findSeam_Dyn_sol(energy)

%M is the cumulative energy function
M = zeros(size(energy));
%Prev_indices is a matrix that will point to the element in the (i-1)th row
%that an M(i,j) came from.
prev_indices = zeros(size(M));
M(1,:) = energy(1,:);

for i = 2:size(energy,1)
   %for columns not on the extreme right or left
   for j = 2:size(energy,2)-1
       [min_M,col_idx] = min( [M(i-1,j-1),M(i-1,j),M(i-1,j+1)] );
       prev_indices(i,j) = j+ (col_idx-2);
       M(i,j) = energy(i,j) +  min_M;
   end
   %for the right edge
   j = size(energy,2);
   [min_M,col_idx] = min([M(i-1,j-1),M(i-1,j)]);
   prev_indices(i,j) = j+ (col_idx-2);
   M(i,j) = energy(i,j) +  min_M;

   %for the left edge
   j = 1;
   [min_M,col_idx] = min([M(i-1,j),M(i-1,j+1)]);
   prev_indices(i,j) = j+ (col_idx-1);
   M(i,j) = energy(i,j) +  min_M;
end

%find the ending location of the path that contains the least energy
[min_M, col_idx] = min(M(end,:));

%re-trace the path
pix = zeros(size(M,1),1);

pix(end) = col_idx;
for i = length(pix)-1:-1:1
    pix(i) = prev_indices(i+1,pix(i+1));
```

```
end
```

Results:

```
reduceWidth('stanford.jpg', 200)
reduceWidth('dogs.jpg', 200)
reduceWidth('pool.jpg', 200)
```

The greedy algorithm's search for local minimum tends to have its start clustered in one region of the image. This leads to one area of the image being disproportionately deleted. The dynamic programming algorithm doesn't have this problem, and tends to delete seams more uniformly over the image.

However the greedy is better at following regular geometry, and following lines. This is seen in the result of the pool cue stick image.

Images that can have many seams removed without perceptible difference, are images that typically do not rely on fine grain detail and tend to be a larger composite scene, such as the stanford image.

(a) Dynamic Programming

(b) Greedy

Figure 4: Stanford Result

(a) Dynamic Programming

(b) Greedy

Figure 5: Dogs Result

(a) Dynamic Programming

(b) Greedy

Figure 6: Pool Result