CS 480 Programming Assignment # 01
Applications of Singular Value Decomposition

Deev Patel

March 3, 2019

Problem 1: Mapping Facial Features

Problem Discussion & Technical Overview

Naturally acquired images often have a high degree of variability in camera orientation, zoom, and resolution. As a result, even if we take images of the same underlying item things can appear very differently. This variability can cause many computer vision applications to fail. Thus, as a pre-processing step, we often normalize the images to a fixed size and map certain common features to fixed locations.

For this specific problem, we will take images of various human faces and transform them onto a 48×40 domain in such a way that four common features (the left eye center, the right eye center, the nose tip, & the mouth center) appear at fixed locations. While we cannot map every point exactly to its desired location, we can get close by computing a least squares solution via Singular Value Decomposition (SVD).

We can start by modeling a set of 2-D affine transformations to transform a given 2-D image into a 48×40 plane. We can do this by using a generic 2×2 matrix, A, to represent scale, rotation, and shear in conjunction with a generic 2×1 matrix, b, to represent translation. Then, we can take any point P in the original image and find its corresponding point, $\hat{P}$, in the 48×40 image.

$$\hat{P} = AP + b \tag{1}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad , \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Since we want to fix the positions of four features (mentioned earlier) in the new image, we can write the following system of linear equations, where P_1, P_2, P_3, P_4 represent the x, y coordinates of the features in the original image, and $\hat{P}_1, \hat{P}_2, \hat{P}_3, \hat{P}_4$ represent the fixed location we want those points to end up at.

$$\begin{aligned} \hat{P}_1 &= AP_1 + b \\ \hat{P}_2 &= AP_2 + b \\ \hat{P}_3 &= AP_3 + b \\ \hat{P}_4 &= AP_4 + b \end{aligned} \tag{2}$$

Now, to make things more clear, we can rewrite the system above in matrix form using homogeneous coordinates and expand all the points in terms of their x, y coordinates.

$$\begin{bmatrix} \hat{x_1} & \hat{y_1} \\ \hat{x_2} & \hat{y_2} \\ \hat{x_3} & \hat{y_3} \\ \hat{x_4} & \hat{y_4} \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ b_1 & b_2 \end{bmatrix} \tag{3}$$

It is now clear that we have an over-determined set of equations. There are eight equations but only six unknowns $(a_{11}, a_{21}, a_{12}, a_{22}, b_1, b_2)$. So, chances are unlikely that we will be able to find an exact solution. However, we can still try to estimate the system so that $\hat{P_1}, \hat{P_2}, \hat{P_3}, \hat{P_4}$ end up as close to the target values as possible. We can accomplish by viewing Eq (3) as two separate over determined linear systems, each with four equations and three unknowns.

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ b_1 \end{bmatrix} = \begin{bmatrix} \hat{x_1} \\ \hat{x_2} \\ \hat{x_3} \\ \hat{x_4} \end{bmatrix} \quad , \quad \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{bmatrix} \begin{bmatrix} a_{21} \\ a_{22} \\ b_2 \end{bmatrix} = \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \hat{y_3} \\ \hat{y_4} \end{bmatrix} \tag{4}$$

Here, we now have two systems of over-determined linear equations in the form $Cz = d$. Ordinarily, if C were square, we could try and solve this by multiplying by C^{-1} from the left. However, since C is 4×3, it cannot have an inverse. We can get around this by using the fact that any matrix times its transpose yields a square matrix. So, we can multiply by C^T from the left. Then, we can solve the system (ie: obtain its least-sqaure solution), by multiplying by the inverse of the now square matrix. Thus, we get $z = (C^T C)^{-1} C^T d$. Here, $(C^T C)^{-1} C^T$ represents the sudo-inverse of C and is written as C^+. Thus, Eq (4) becomes the following.

$$\begin{bmatrix} a_{11} \\ a_{12} \\ b_1 \end{bmatrix} = \left(\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{bmatrix} \right)^+ \begin{bmatrix} \hat{x_1} \\ \hat{x_2} \\ \hat{x_3} \\ \hat{x_4} \end{bmatrix} \quad , \quad \begin{bmatrix} a_{21} \\ a_{22} \\ b_2 \end{bmatrix} = \left(\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{bmatrix} \right)^+ \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \hat{y_3} \\ \hat{y_4} \end{bmatrix} \tag{5}$$

However, now the problem that arises is what happens if $(C^T C)$ is singular. That is $(C^T C)^{-1}$ does not exist. This is where we can use Singular Value Decomposition (SVD) and get a usable estimate in all cases. From SVD, we know that any matrix can be written in the form UDV^T, where U is a column

orthonormal matrix containing the eigenvectors of AA^T, V is a orthonormal matrix containing the eigenvectors of A^TA, and D is a diagonal matrix containing the singular values. Now, it can be shown that the sudo-inverse of the matrix can be estimated by $VD^{-1}U^T$, where D^{-1} is the diagonal matrix containing the reciprocal of every singular value greater than some $\epsilon > 0$ and 0 everywhere else.

Looking back at Eq (5), we can see that the full affine transformation can be estimated by computing an SVD approximation of a single homogeneous coordinate matrix describing all the ground truth feature points and using it to perform two different multiplications.

Now that we have the affine transformation, we can create the normalized image. Theoretically, this is fairly simple and involves using Eq (2) on every point P in the original image and finding its corresponding $\hat{P}$ in the normalized image. However, this is practically unfeasible because of the discrete nature of images. Since we only have certain discrete values of P, we cannot guarantee that every desired point in the normalized image will be given a value. Thus, in order to create the normalized image, we must start with the points that we want to compute on the normalized image, $\hat{P}$, and work our way backwards to find the P that each $\hat{P}$ came from. Here, if P is not a value that we have sampled (ie: it is not an integer in the domain size), we can simply use the nearest value that we actually have.

$$\begin{bmatrix}\hat{X}\\ \hat{Y}\end{bmatrix} = \begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}\begin{bmatrix}X\\ Y\end{bmatrix} + \begin{bmatrix}b_1\\ b_2\end{bmatrix} \qquad \begin{bmatrix}\hat{X}\\ \hat{Y}\end{bmatrix} = \hat{P}, \begin{bmatrix}X\\ Y\end{bmatrix} = P$$

$$\begin{bmatrix}X\\ Y\end{bmatrix} = \begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}^{-1}\begin{bmatrix}(\hat{X} - b_1)\\ (\hat{Y} - b_2)\end{bmatrix} \tag{6}$$

With Eq (6), we can practically recreate the full normalized image, even if we have to estimate/interpolate the solution in many cases. Note that Eq (6) represents a linear system and can thus be solved via an SVD decomposition. Using SVD is efficient because, in creating the new image, we will keep the matrix being decomposed the same, and only change the column matrix on the right hand side.

So, we now have a methodology, involving various SVD decompositions, to normalize images features into certain specified locations.

Implementation Notes

The code used to generate the results in this report, is attached alongside this document. Consult the README for more details.

Before embarking on the project, I manually found the four required features on all 20 given images. The locations can be found in the "*annotated_locations.txt*" file. Based on these manually chosen locations, I determined where I wanted each feature to map onto the new 48×40 image. Note that these points represent the $\hat{P}$ values in Eq (2).

$$\text{Desired location of left eye center } (\hat{P}_1) : \begin{bmatrix} 12 & 19 \end{bmatrix}^T$$
$$\text{Desired location of right eye center } (\hat{P}_2) : \begin{bmatrix} 27 & 19 \end{bmatrix}^T$$
$$\text{Desired location of nose tip } (\hat{P}_3) : \begin{bmatrix} 20 & 27 \end{bmatrix}^T$$
$$\text{Desired location of lip center } (\hat{P}_4) : \begin{bmatrix} 20 & 38 \end{bmatrix}^T$$

Once I choose the desired points, I hard coded their x, y coordinates into two 4×1 floating point cv::Mat variables, following in the spirit of Eq (4). Then, I wrote a function to use the OpenCV SVD class to solve/estimate systems of linear equations via Singular Value Decomposition. Notably, it was efficiently designed so a single SVD computation could be used to solve multiple systems that only differed in their right hand sides. This function formed the basis of my approach in that it was used to compute the affine transformation and create the normalized image. Finally, I wrote a few other helper functions to parse the annotation data file for features, read the images, and to display/save the results.

Once all the necessary helper functions were created, I moved onto the actual process of normalizing the images. This involved performing two steps on each individual image, (1) computing the parameters of the affine transformation and (2) using the computed transformation to produce a normalized image.

(1) To find the affine transformation, I essentially followed Eq (5). I first created a 4×3 cv::Mat to hold the homogeneous coordinates of the four feature points. Then I used the two already existing matrices with desired locations to solve the two over-determined systems from Eq (5). Here I made use of the SVD function mentioned earlier since only the homogeneous coordinate matrix had to be decomposed. Taken together, the solutions provided me with the full affine transformation.

(2) With the affine transformation fully computed, I went about the task of normalizing each image into the new 48×40 domain. I followed the inverse mapping approach mentioned at the end of the technical discussion. This approach, shown in Eq (6), required me to create one 2×2 matrix containing the A part of the affine transformation and a series of other column vectors. The column vectors were created by taking the locations I needed to be mapped (0 to 39 for x and 0 to 47 for y) and subtracting the b part of the affine transformation. This amounted to $48 \times 40 = 1920$ column vectors, one for every pixel value in the normalized image. Then, I simply ran the SVD decomposition function on the 2×2 A matrix and passed all 1920 column vectors as the right hand sides. Finally, I filled in each pixel value in the normalized image by taking the nearest corresponding value (via rounding) in the original image.

Once the normalization process was done, I added functionality to compute the normalization error. This was done by using the forward mapping to find the closest location where each of the feature points went to. Then, I simply took the average of the norms between those value and the desired values to compute the average shown in the results.

Results & Discussion

Before performing any transformations, the manually extracted features were visualized. From these features, shown in the first and third rows of Fig. 1 & Fig. 2, we can see that their absolute and local locations tend vary from image to image. This variation is exactly what we hope to reduce via the normalization process.

The visual results of the normalization process are shown in the second and fourth rows of Fig. 1 & Fig. 2. From the figures, it is evident that the normalization process, while not perfect, did a fairly nice job of mapping the features to the desired locations. Furthermore, the normalized images more or less maintained the faces. That being said, in some images the face did end up be being squished or deformed. Additionally, sometimes parts of the face, especially the ears, ended up looking motion blurred. While this is not desirable, it is to be expected as some faces are bound to have feature locations different from the average. Moreover, since the discrete nature of images makes it near impossible to fully apply the affine transformation without some sort of interpolation, there will always be some errors. When looking at the normalized images, it is important to note that they seem

blurred/pixelated because they are actually 48×40. For future cases, it may be better to map the normalized images/features onto a bigger domain.

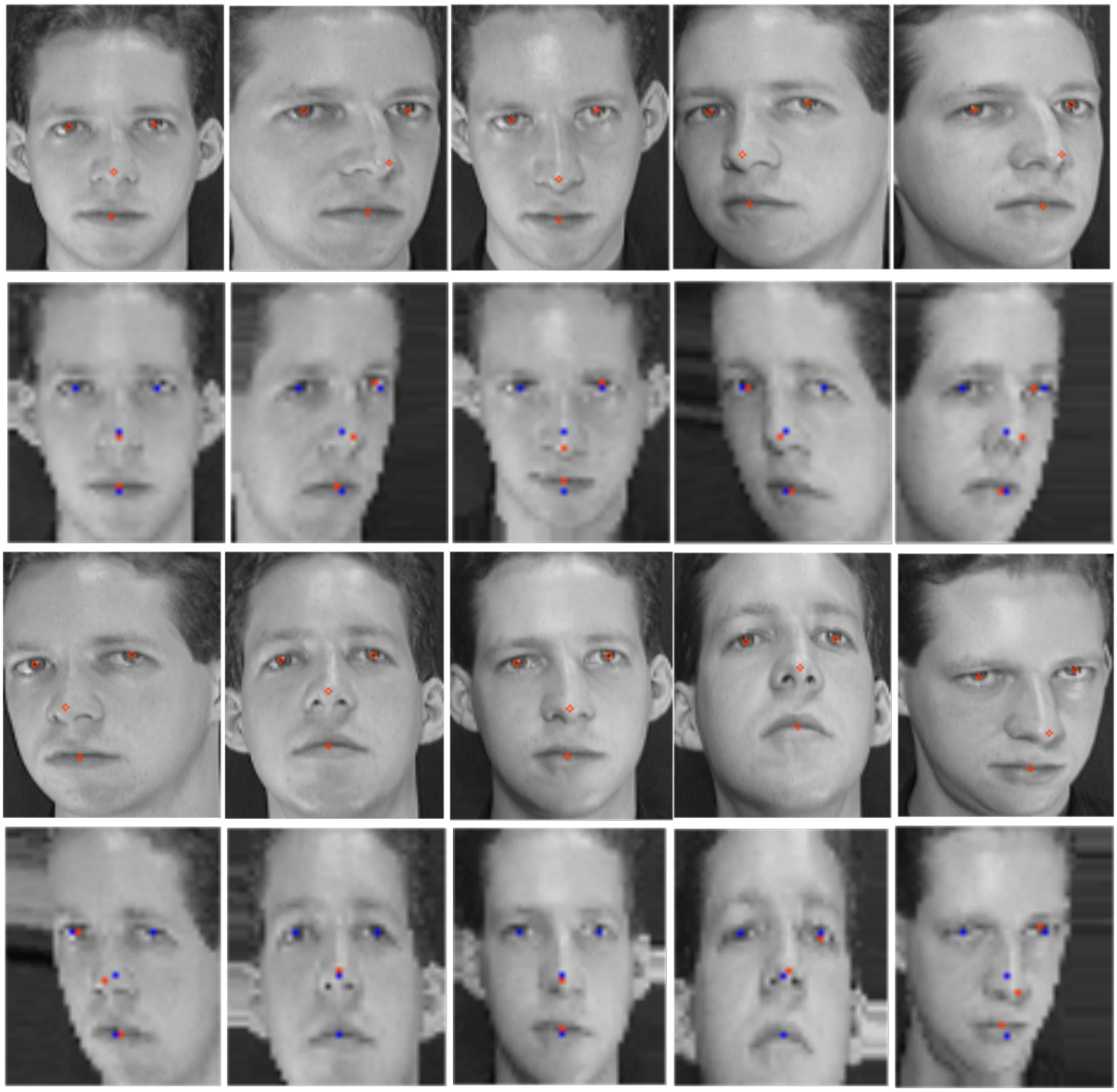

Figure 1: Visual results of feature normalization on data set "S1". The first and third rows show the original images with the hand-determined features shown in orange. The second and fourth rows show the corresponding 48×40 normalized image. The blue points represent the, fixed, desired feature locations, while the orange points represent the actual features obtained via the forward mapping. Note that if there is no orange dot for a feature, then it ended up at the exact desired location.

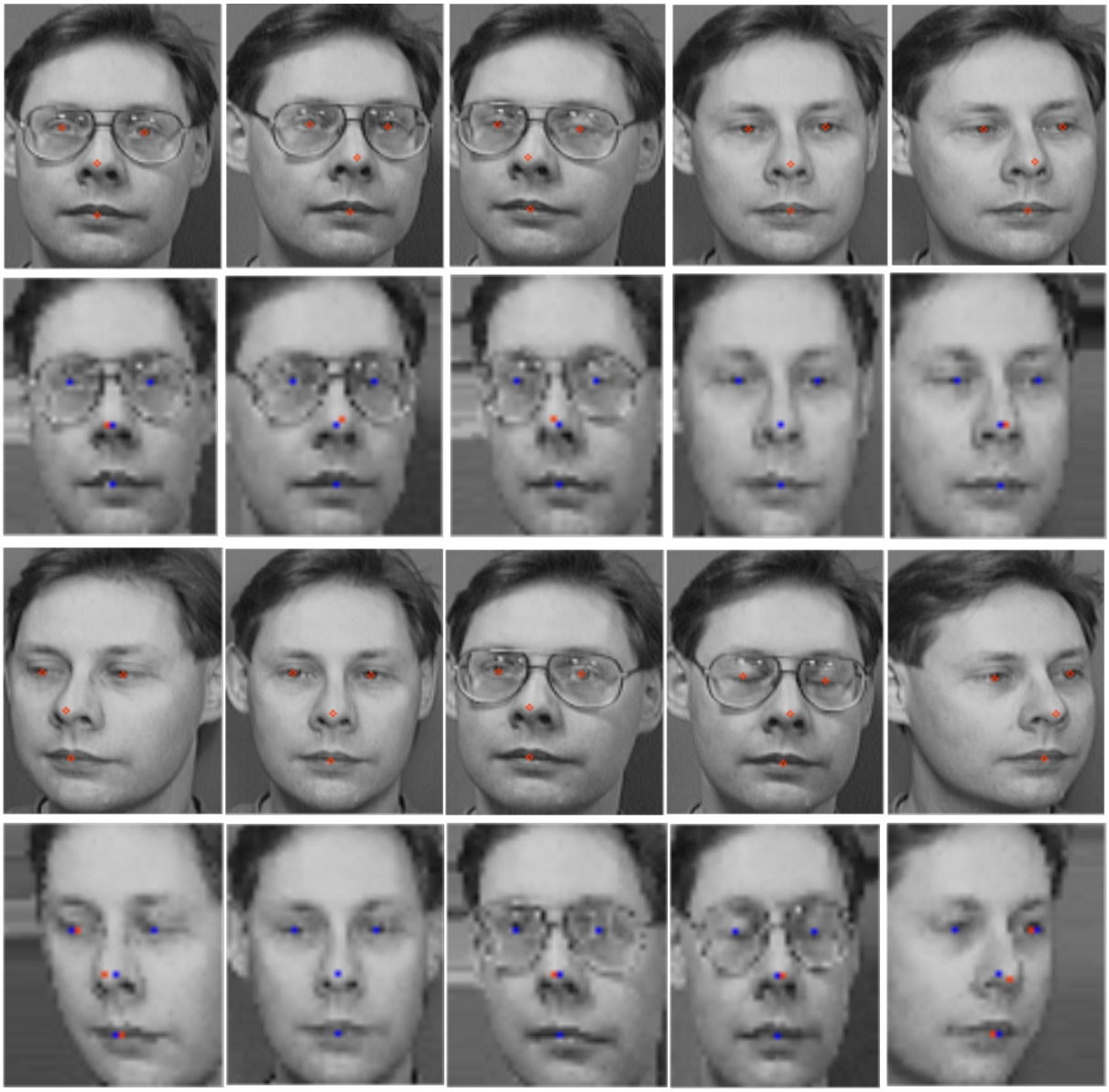

Figure 2: Visual results of feature normalization on data set "S2". The first and third rows show the original images with the hand-determined features shown in orange. The second and fourth rows show the corresponding 48×40 normalized image. The blue points represent the, fixed, desired feature locations, while the orange dots represent the actual features obtained via the forward mapping. Note that if there is no orange dot for a feature, then it ended up at the exact desired location.

The numerical results of the normalization process are given in Fig. 3 & Fig. 4. They show the parameters of the affine transformation and error rate of the images in Fig. 1 & Fig. 2 respectively. The error rate was computed by finding the average of the distances between where each point ended and where it was supposed to go. From the error rates, it is evident that the SVD decomposition to the over determined system did a nice job of finding the optimal solution. For most of the images, the feature points ended up within 1 or 2 pixels of the desired location. The exact parameters can be found in the log files provided with the report.

A	$\begin{bmatrix} .42 & .01 \\ .01 & .49 \end{bmatrix}$	$\begin{bmatrix} .31 & -.04 \\ -.01 & .44 \end{bmatrix}$	$\begin{bmatrix} .42 & -.01 \\ .04 & .41 \end{bmatrix}$	$\begin{bmatrix} .35 & .05 \\ .05 & .46 \end{bmatrix}$	$\begin{bmatrix} .32 & -.07 \\ .01 & .44 \end{bmatrix}$
b	$\begin{bmatrix} 0.52 \\ -6.32 \end{bmatrix}$	$\begin{bmatrix} 4.10 \\ 0.23 \end{bmatrix}$	$\begin{bmatrix} 2.00 \\ -1.95 \end{bmatrix}$	$\begin{bmatrix} 4.81 \\ -2.91 \end{bmatrix}$	$\begin{bmatrix} 4.19 \\ -0.89 \end{bmatrix}$
$\lvert Err \rvert$	0.500	1.266	1.500	0.853	1.54

A	$\begin{bmatrix} .35 & .03 \\ .05 & .46 \end{bmatrix}$	$\begin{bmatrix} .39 & .01 \\ .03 & .51 \end{bmatrix}$	$\begin{bmatrix} .39 & -.01 \\ .03 & .46 \end{bmatrix}$	$\begin{bmatrix} .39 & -.02 \\ .02 & .52 \end{bmatrix}$	$\begin{bmatrix} .35 & -.02 \\ .004 & .44 \end{bmatrix}$
b	$\begin{bmatrix} 6.59 \\ -3.65 \end{bmatrix}$	$\begin{bmatrix} 2.80 \\ -4.57 \end{bmatrix}$	$\begin{bmatrix} 1.76 \\ -3.26 \end{bmatrix}$	$\begin{bmatrix} 0.75 \\ -0.38 \end{bmatrix}$	$\begin{bmatrix} 1.47 \\ -3.82 \end{bmatrix}$
$\lvert Err \rvert$	1.059	0.250	0.500	0.604	1.814

Figure 3: Numerical results of feature normalization on data set "S1". The top table corresponds to the images in the top half of Fig. 1. The bottom table corresponds to the images in the bottom half of Fig. 1. The error is calculated by find the distances between the desired and actual features and averaging them.

A	$\begin{bmatrix} .42 & .05 \\ -.03 & .53 \end{bmatrix}$	$\begin{bmatrix} .45 & .01 \\ -.01 & .52 \end{bmatrix}$	$\begin{bmatrix} .42 & .06 \\ -.03 & .54 \end{bmatrix}$	$\begin{bmatrix} .45 & -.01 \\ .02 & .53 \end{bmatrix}$	$\begin{bmatrix} .43 & -.01 \\ .02 & .53 \end{bmatrix}$
b	$\begin{bmatrix} -0.28 \\ -7.49 \end{bmatrix}$	$\begin{bmatrix} -4.23 \\ -7.05 \end{bmatrix}$	$\begin{bmatrix} 1.14 \\ -7.64 \end{bmatrix}$	$\begin{bmatrix} -2.27 \\ -9.84 \end{bmatrix}$	$\begin{bmatrix} -4.32 \\ -9.97 \end{bmatrix}$
$\lvert Err \rvert$	0.250	0.354	0.354	0.000	0.250

A	$\begin{bmatrix} .42 & .08 \\ -.01 & .53 \end{bmatrix}$	$\begin{bmatrix} .45 & .02 \\ -.02 & .52 \end{bmatrix}$	$\begin{bmatrix} .42 & .07 \\ -.02 & .53 \end{bmatrix}$	$\begin{bmatrix} .42 & .02 \\ -.03 & .53 \end{bmatrix}$	$\begin{bmatrix} .43 & -.06 \\ .02 & .54 \end{bmatrix}$
b	$\begin{bmatrix} 1.69 \\ -8.57 \end{bmatrix}$	$\begin{bmatrix} -1.74 \\ -7.60 \end{bmatrix}$	$\begin{bmatrix} -0.51 \\ -7.81 \end{bmatrix}$	$\begin{bmatrix} -2.67 \\ -8.00 \end{bmatrix}$	$\begin{bmatrix} -4.20 \\ -11.02 \end{bmatrix}$
$\lvert Err \rvert$	1.000	0.000	0.250	0.250	1.059

Figure 4: Numerical results of feature normalization on data set "S2". The top table corresponds to the images in the top half of Fig. 2. The bottom table corresponds to the images in the bottom half of Fig. 2. The error is calculated by find the distances between the desired and actual features and averaging them.

Problem 2: Lighting Correction (Extra Credit)

Problem Discussion & Technical Overview

In addition to using SVD to normalize image features, we can also use it to try and remove many kinds of degradation. In this specific case, we will look at a simple model for pixel intensities in hopes of reducing the effects of uneven illumination.

We start by proposing a non-linear model to define the intensity values of an image in terms of each pixel's x, y coordinates. Note that a, b, c, d represent arbitrary parameters that differ from image to image.

$$f(x, y) = ax + by + cxy + d \tag{7}$$

In order find the parameters of this model — a, b, c, d — we can leverage the fact that we know $f(x, y)$ for every pair of discretely sampled x, y pairs in the image. So, for any $n \times m$ image, we can use the already known pixel values to create an over-determined system with four unknowns and $n \times m$ equations. To make things simpler mathematically, we can put the system into matrix form. Note that $x_{a,b}$ represents the x coordinate at location (a, b) in the image. Similarly $y_{a,b}$ represents the y coordinate.

$$\begin{bmatrix} f(x_{0,0}, y_{0,0}) \\ f(x_{0,1}, y_{0,1}) \\ \vdots \\ f(x_{0,n-1}, y_{0,n-1}) \\ f(x_{1,0}, y_{1,0}) \\ f(x_{1,1}, y_{1,1}) \\ \vdots \\ f(x_{m-1,n-1}, y_{m-1,n-1}) \end{bmatrix} = \begin{bmatrix} x_{0,0} & y_{0,0} & x_{0,0}y_{0,0} & 1 \\ x_{0,1} & y_{0,1} & x_{0,1}y_{0,1} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{0,n-1} & y_{0,n-1} & x_{0,n-1}y_{0,n-1} & 1 \\ x_{1,0} & y_{1,0} & x_{1,0}y_{1,0} & 1 \\ x_{1,1} & y_{1,1} & x_{1,1}y_{1,1} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{m-1,n-1} & y_{m-1,n-1} & x_{m-1,n-1}y_{m-1,n-1} & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \tag{8}$$

Now that we have fully defined the over determined system, we can go ahead and use SVD decomposition to estimate the four parameters. This can be done by decomposing the $nm \times 4$ matrix and multiplying by the $nm \times 1$ matrix from the right. This will allow us to estimate the best values for the four parameters of Eq (7).

Once we have found the best parameters for the model, we can go ahead and "correct" our original image and remove illumination discrepancies. This can be down by computing the expected pixel values and subtracting them from the actual ones.

Implementation Notes

The code used to generate the results in this report, is attached alongside this document. Consult the README for more details.

The implementation of problem 2 was fairly straightforward. I simply took the feature normalized images from problem 1 and computed the two non-parameter matrices in Eq (8). I then used the SVD function from problem 1 and solved the system in Eq (8) for the parameters a, b, c, d. Then, I used Eq (7) to compute the model at every pixel value. Finally, I subtracted the model from the original image to get the light normalized image. Here, I had to be careful in that the subtraction often resulted in negative values. Thus, I had the store the results of the subtraction in a temporary signed integer array and remap the values into normal range of [0, 255]. Note that this whole process was only implemented for gray scale images. However, everything could easily be converted to the RGB domain. It would just require the over determined system to have more equations or operate on vectors.

Results & Discussion

The visual results of the lighting normalization can be seen in Fig. 5 & Fig. 6. While the changes between the original and lighting normalized images are not always evident at first glance, we can see that the process tends to affect darker and lighter areas. For instance, in many of the lighting normalized images, there a drastic difference in the background shade. In some cases, we can even see more variation background of the final result. In terms of the actual faces themselves, the only differences noticeable to the human eye seem to appear near the hot-spots on the forehead of certain faces. Even then, the changes are barely evident because the images have such a low resolution. For a better understanding of the effects of lighting correction, it may have been better to choose higher quality images that were known to be affected by uneven illumination. Since these images are only 48×40 and probably do not suffer from lighting illumination, the changes are hard to view.

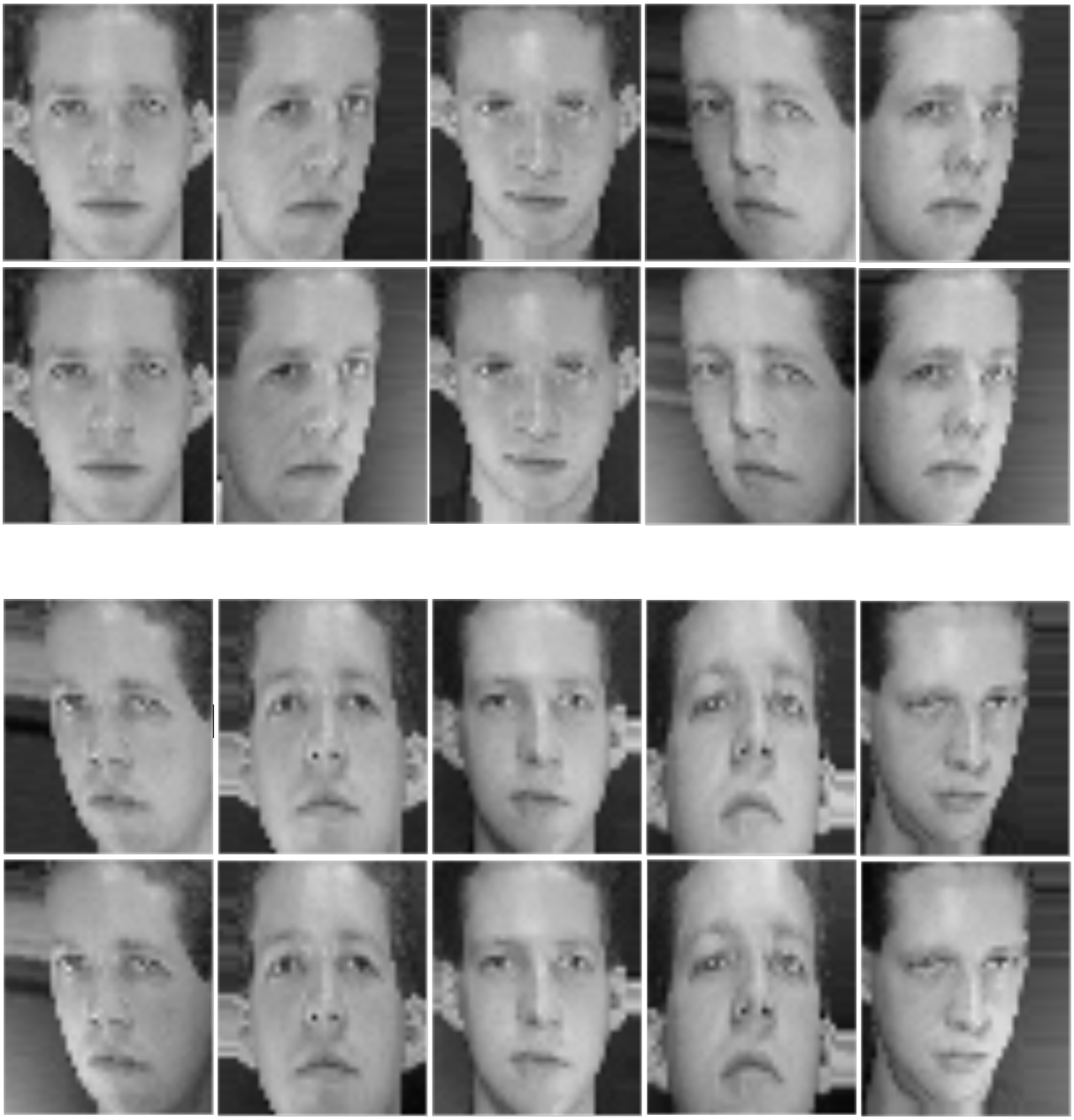

Figure 5: The visual results of lighting normalization on data set "S1". The first and third rows show the feature normalized images from Fig.1. The second and fourth rows show the corresponding lighting normalized image created by subtracting the model in Eq (7).

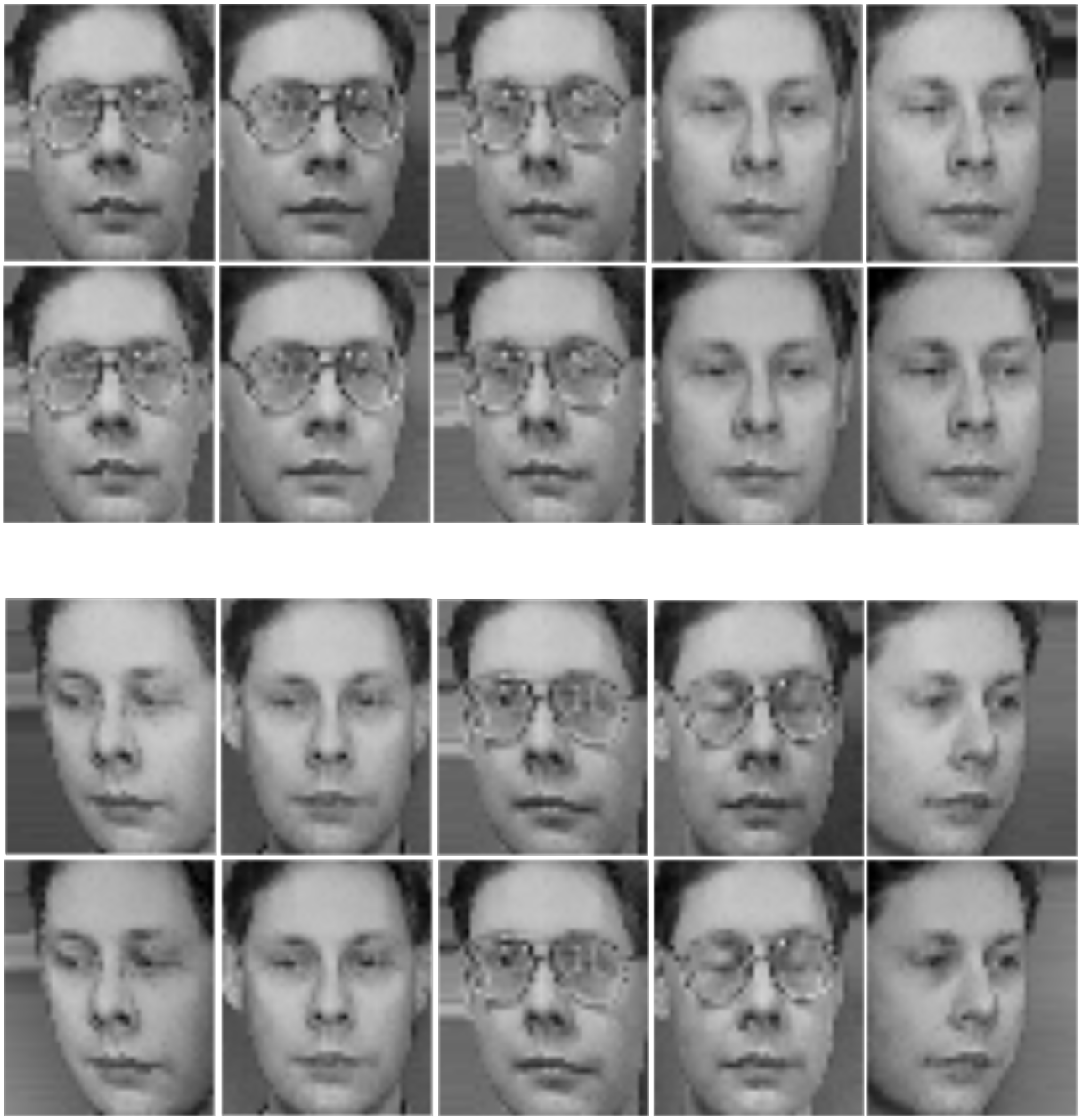

Figure 6: The visual results of lighting normalization on data set "S2". The first and third rows show the feature normalized images from Fig.2. The second and fourth rows show the corresponding lighting normalized image created by subtracting the model in Eq (7).

The numerical results of the lighting normalization are given in Fig. 7 & Fig. 8. They give the recovered parameters from Eq (7) for each of the images in Fig. 5 & Fig. 6.

a	-0.586	-0.782	-0.622	0.365	-0.428
b	-0.554	1.803	-0.725	-2.614	0.736
c	0.024	-0.095	0.010	0.117	-.062
d	145.514	137.644	153.075	121.228	139.641

a	-1.431	-0.507	0.308	-0.244	-0.136
b	-2.905	-0.073	-0.940	-0.285	-0.102
c	0.138	0.028	0.019	0.022	-0.044
d	160.072	131.667	131.287	122.907	142.66

Figure 7: The numerical results of lighting normalization on data set "S1". The top table corresponds to the images in the top half of Fig. 5. The bottom table corresponds to the images in the bottom half of Fig. 5.

a	-0.138	-0.334	-0.225	-0.339	-1.952
b	-0.083	0.134	-0.535	-0.602	-0.963
c	-0.015	-0.035	0.0005	0.005	0.027
d	135.350	140.300	144.350	152.394	182.333

a	0.975	0.167	-0.349	-0.266	-1.448
b	-1.525	-0.583	-0.649	0.011	-0.499
c	0.034	0.002	0.010	-0.034	-0.009
d	135.689	140.910	145.221	140.580	177.600

Figure 8: The numerical results of lighting normalization on data set "S2". The top table corresponds to the images in the top half of Fig. 6. The bottom table corresponds to the images in the bottom half of Fig. 7