EA-1 Lab Project: Least Squares

This part of the Lab Project will focus on the solution of Least Squares problems. You will learn how to apply Least Squares in order to reconstruct images with missing information (image inpainting). Furthermore, you will test how image corruption can affect face recognition, based on the student image database we constructed in the Lab Introduction. We remind here that all material from the three parts of the Lab Project should be submitted together. There will be a single due date for the Lab Project during the last week of classes. Detailed instructions on submitting the Lab Project can be found at the end of this document, in Section 5.

1 Image Reconstruction using a Dictionary of Patches

Image reconstruction is one of the core problems of Image Processing and it usually involves finding the solution to a linear system of equations $\mathbf{y} = A\mathbf{x}$, where $\mathbf{x}$ represents the unknown vectorized image, A correspods to a linear measurement operation and $\mathbf{y}$ denotes the vector of observed data from which the actual image $\mathbf{x}$ must be inferred. The solution to the aforementioned problem can seem trivial, at first glance, and one could claim that it could be calculated by $\mathbf{x} = A^{-1}\mathbf{y}$ if the matrix A is square, or by using the least squares solution $\mathbf{x} = (A^T A)^{-1} A^T \mathbf{y}$, otherwise. However, often times, such solution might not be tractable due to the matrix A or $A^T A$ being singular. In such cases, assumptions concerning the image $\mathbf{x}$ can be used in order to constraint the solution space.

One commonly used approach in the Image Processing literature is that a small image block (patch) can be represented as a linear combination of image patches selected by a set of images in a database. The underlying idea is that a big selection of image patches appropriately selected by a set of images can sufficiently describe all possible edges or smooth areas in an image and therefore could form a good basis for any image block representation. As such, one could utilize this image patch database and use it to reconstruct any image that could be corrupted by noise, blurring effects or other types of degradation. An overview of this idea is shown in Figure 1.

In mathematical terms, we denote:

- $\mathbf{x}_p$: $N \times 1$ vectorized unknown image patch.
- D : $N \times M$ dictionary of patches (each patch represents an $N \times 1$ column in D).
- $\mathbf{a}_p$: $M \times 1$ vector of coefficients such that each image patch $\mathbf{x}_p$ of image $\mathbf{x}$ can be described as $\mathbf{x}_p = D\mathbf{a}_p$.
- T_p : A transformation matrix that transforms a set of P image patches $\mathbf{x}_p$, $p = 1 \dots P$, to the vectorized image $\mathbf{x}$.

Therefore the total observation model can be described as:

$$\mathbf{y} = A \cdot T_p \cdot D \cdot \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_P \end{bmatrix} = A \cdot T_p \cdot \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_P \end{bmatrix} = A \cdot \mathbf{x}. \tag{1}$$

As a result, if the total observation matrix $M = A \cdot T_p \cdot D$ is invertible or, at least, well-behaved in the Least Square sense, one could first calculate the unknown set of vectors $\begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_P \end{bmatrix}$, and then obtain the unknown image by applying the equation $\mathbf{x} = T_p \cdot D \cdot \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_P \end{bmatrix}$. Strictly speaking, the Least Squares method might not always be appropriate for such types of image reconstruction and there are multiple advanced methods for solving image reconstruction problems for all sorts of degradation types. Here, we will consider the case of image inpainting, where Least Squares and Dictionary-based techniques can prove very effective.

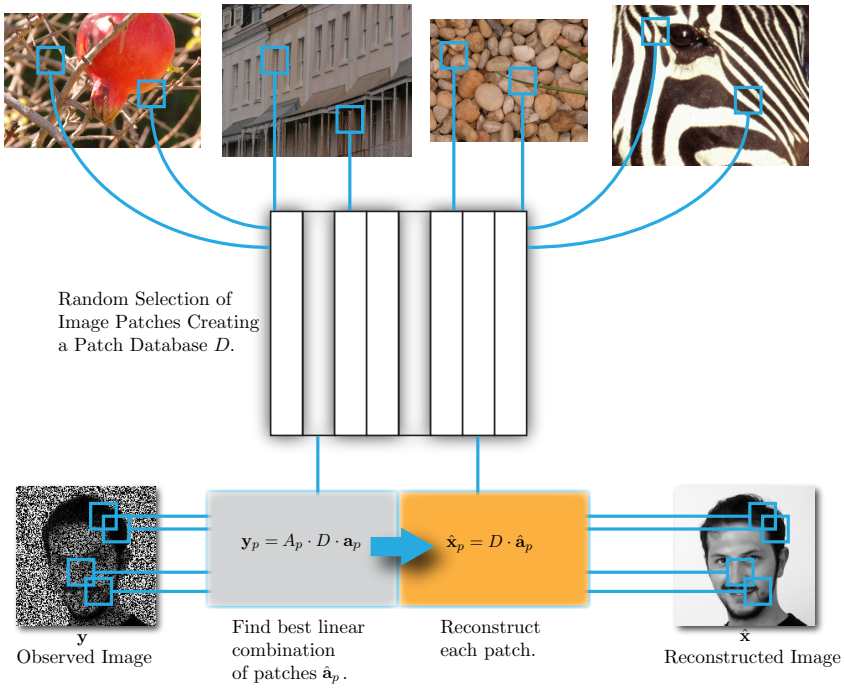

Figure 1: Illustration of main idea behind image reconstruction using a dictionary of patches.

2 Image Inpainting

Image Inpainting refers to the problem when one aims at reconstructing an image when some of its pixels have been erased. In this case, the measurement matrix A in equation (1) would be a diagonal matrix with entries 1 or 0 on the diagonal, depending on whether a pixel has been measured or not, respectively. We can simplify equation (1) to a patch level by writing,

$$\mathbf{y}_p = M_p \cdot \mathbf{a}_p = A_p \cdot D \cdot \mathbf{a}_p = A_p \cdot \mathbf{x}_p, \tag{2}$$

where $\mathbf{y}_p : N \times 1$ represents a patch in the measured image and $A_p : N \times N$ correponds to the entries of the diagonal matrix A that measure the specific patch p. In simpler words, matrix A_p can be viewed as another diagonal matrix with entries 1 or 0 but only corresponding to the pixels within a patch in the image $\mathbf{x}$. Consider an example where,

$$\mathbf{y}_p = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \quad A_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad M_p = A_p \cdot D = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 0 & 0 & 0 \end{bmatrix} \tag{3}$$

Obviously, matrices A_p and M_p are not invertible. However, we are interested in finding a solution without taking into account the equations that have $\mathbf{y} = 0$ since this constitutes a missing pixel in our image and is not informative. Instead, we want to find a linear combination of vectors in D that give us the best solution for the non-zero part of $\mathbf{y}$. If D has been constructed such that it contains information of commonly appearing consecutive pixels in an image (e.g., dictionary of patches from images), we expect that by finding a solution that satisfies the non-zero part of the equation, we will be able to retrieve the missing information at the zero locations.

Thererefore, we can solve the reduced linear system of equations in (4) in order to find a good

estimate of the correct solution,

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \hat{\mathbf{a}}_p = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \rightarrow \hat{\mathbf{a}}_p = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{3} \end{bmatrix} \rightarrow \hat{\mathbf{x}}_p = D \cdot \hat{\mathbf{a}}_p = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}. \tag{4}$$

Hence the estimated vector $\hat{\mathbf{x}}_p$ represents the unknown image which is best described by using the dictionary D and based on the observation $\mathbf{y}$. The main idea behind image inpainting is presented in Figure 1. By repeating the above procedure for all image patches $p = 1 \dots P$, one can estimate the original image as $\hat{\mathbf{x}} = T_p \begin{bmatrix} \hat{\mathbf{x}}_1 & \hat{\mathbf{x}}_2 & \dots & \hat{\mathbf{x}}_P \end{bmatrix}$. Usually, for improved performance, overlapping patches are being used and subsequently averaged in order to smooth-out possible outliers in the estimated solution vectors.

3 Problem Description

3.1 Goal of this Project

The goal of this part of the Lab Project is to create an algorithm for image reconstruction for images that have been degraded by erasing a percentage of their pixels. You will be using the methods described in Section 2. Furthermore, the image reconstruction functions will be tested on the student images and the database created at the Introduction of the Lab Project to study how face recongition can be affected due to the images being degraded.

We provide you with the following:

- Function `corruptStudentImage(x,studentNumber,corruptionPercentages)` which performs corruption of the image in `x` by erasing a percentage of its pixels selected at random. The function performs corruption for all percentages stored in the vector `corruptionPercentages`. The function returns as output the matrix `corruptedStudentImages` which contains the corrupted images vectorized in its columns.

- Function `nd2col(img,blockSize,method)` which rearranges each distinct block of size `blockSize` in the image `img` into a column of the output. The input `method` should be set to the string `'sliding'` which vectorizes all overlapping blocks of the image into columns. For more details see help of function `im2col()`. `nd2col()` is a slight modification of this function.

- Function `col2nd(cols,blockSize,imgSize,method)` which rearranges each column of the input matrix `cols` into a block of size `blockSize` in the image of size `imgSize` that is returned as an output. If the input `method` is set to the string `'sliding'` the portions of blocks that overlap are averaged. `cols` is the output of the function `nd2col()` and function `col2nd()` is being used to retransform the columns into an image.

- Script `LabProject.m` which contains the main script you would need to run for this Lab Project. Note that this is an extension of the same file provided to you at the previous parts of the Lab Project. Except for selecting the student number that corresponds to you and writing your group information (names of students as comments), you **should not** make any other changes to this script. The script has been designed to run up to different sections once you have written the appropriate functions, as described in Section 4.

- The `.mat` file, `DictionaryData.mat`, which contains a database of vectorized 8×8 patches selected at random from a set of training images. You will not need to do anything to extract the data from this file. The loading functions are provided to you in the `LabProject.m` script.
- Finally, the function `identifyImages2()` which uses the results of the previous steps of the algorithm to summarize the findings in a nice figure.

Note: Functions `corruptStudentImages()`, `nd2col()`, `col2nd()` and `identifyImages2()` are provided in `.p` file format. This format is essentially the same as a MATLAB `.m` file but cannot be accessed for editing or viewing.

The main steps of the algorithm are:

1. Corrupt the student image by erasing a percentage of its pixels selected at random. The number of the resulting corrupted images is equal to the sequence of selected percentages in the vector `corruptionPercentages`.
2. Reconstruct each corrupted image using the provided patch database and Least Squares reconstruction. The image might still be degraded especially if the performed corruption was severe.
3. Calculate reconstruction performance and create a plot using the PSNR metric.
4. Identify each corrupted image with a student in the `correctedDatabase` created in the Introduction part of the Lab Project.
5. Identify each reconstructed image with a student in the `correctedDatabase` created in the Introduction part of the Lab Project.
6. Present the identification results before and after reconstructing the images.

3.2 To Do Items

1. Copy and paste the new provided files in the same folder where you have the files for the two previous parts of the Lab Project. If you wish, for the file `LabProject.m`, you might only copy and paste the new part of the code under the `LAB PROJECT - PART 2 - LEAST SQUARES`. If, instead, you choose to copy and replace the whole file, you will again need to set the `studentNumber` variable to the number that corresponds to your student image from the folder `Student_Images` as well as write the name of the students in your group at the beginning of the file, in comments.
2. Try to run the provided script `LabProject.m`. You will notice that you get a set of `warnings`. This is due to the fact that some functions are missing in order for the code to run properly.
3. Implement and test the functions discussed in Section 4 one-by-one and test the script `LabProject.m` until all the warnings dissappear. Your whole script should only run completely (warning/error-free) once you have implemented all the functions correctly.
4. Once you have finished writing all the functions test the whole script `LabProject.m`. You should get two figures as an output, summarizing the results. The first one depicts the calculated PSNRs between the original image and each one of the reconstructed images, as

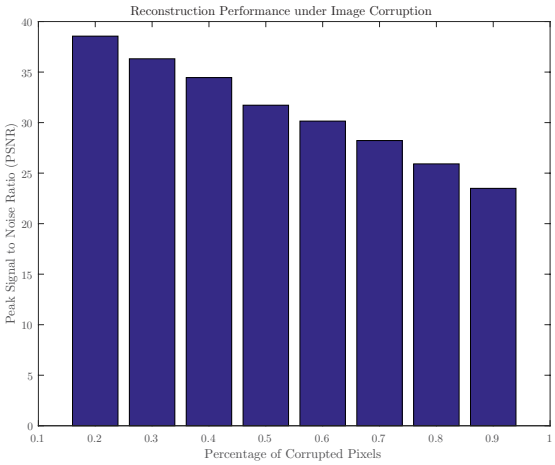

Figure 2: Example of calculated PSNRs for different percentages of corrupted pixels.

the one presented in Figure 2. The second one will be a big cumulative figure containing 4 rows of images. The first one depicts 8 corrupted images of the same student for different percentages of erased pixels, the second one represents the face recognition results when the corrupted images are used as input, the third one represents the corresponding reconstructed images while the last one shows the face recognition results when the reconstructed images are used as input. An example is presented in Figure 3.

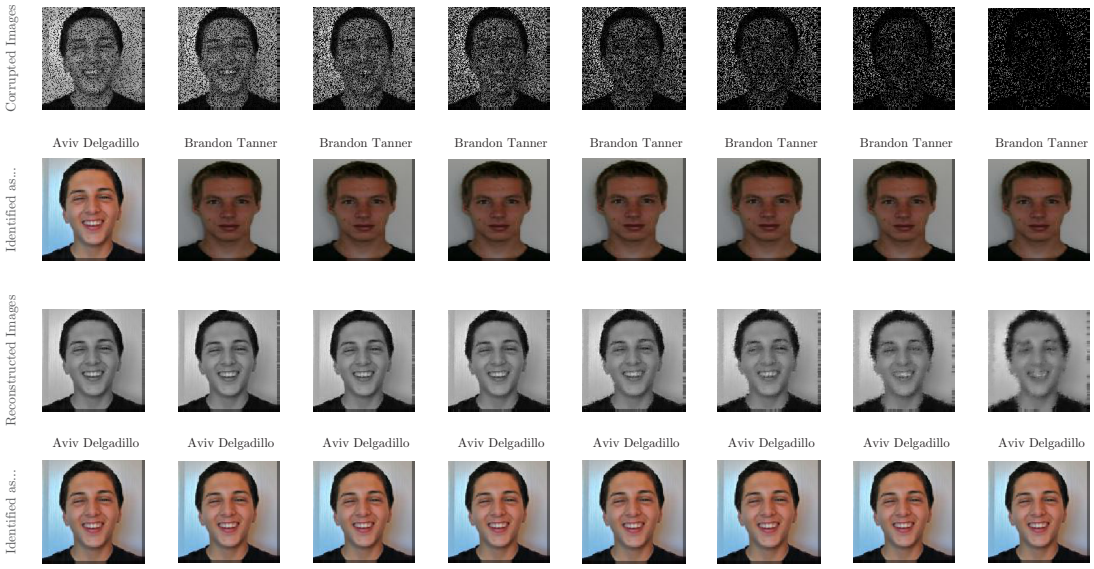

Figure 3: Example of final face recognition results before and after reconstruction.

4 Set of Functions to be Implemented

You will create a set of functions to perform the reconstruction operation as well as see how corruption affects face recognition.

- `function x = solveModifiedLS(A,y)`

  This function acceps a matrix `A` and the column vector `y` as inputs. It returns as an output the Least Squares solution to the linear system of equations $A\mathbf{x} = \mathbf{y}$. However, this should be a"modified" Least Squares solution. Specifically you should find the Least Squares solution only for the linear system of equations that is produced by **erasing** the equations for which $\mathbf{y} = \mathbf{0}$, as described in section 2 and equation (4).

  For example, given the inputs,

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 0 \end{bmatrix},$$

  the output of the function should be the solution of the linear system of equations which results by erasing the last equation in the system, whose corresponding $\mathbf{y}$ is 0. The solution is the vector $\mathbf{x} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^T$.

  In this function, you should:

  1. Make `y` a column vector, if it is not already one.
  2. Check whether the dimensions of the matrix `A` and the vector `y` are consistent in order to be able to find a Least Squares solution, otherwise report an error.
  3. Find the "modified" Least Squares solution, as discussed above.

- `function y = reconstructPatch(A,x)`

  This function acceps a matrix `A` and the column vector `x` as inputs. It first checks whether the dimesions of `A` and `x` are consistent, otherwise it reports an error. Alternatively, it simply calculates the output vector `y`, as $\mathbf{y} = A\mathbf{x}$.

- `function y = reconstructImage(img,dictionary,blockSize)`

  This function accepts as inputs an image (matrix) `img`, a dictionary of patches (matrix) `dictionary`, and a row vector `blockSize` which stores the dimensions of the patches in the dictionary. For example, for the provided dictionary, whose patches have size 8×8, `blockSize` should be the row vector `[8 8]`. The function returns as output the Modified Least Squares reconstruction of the image when using the provided dictionary.

  More specifically, in this function you should:

  1. Check if the `blockSize` is consistent with the size of the provided dictionary, otherwise report an error.
  2. Convert each image block of size `blockSize` to a column and store all these columns in a matrix. You should use the provided function `nd2col()`. For details, please read Section 3.

3. For all the image blocks (patches), find the modified Least Squares solution using the provided dictionary and the reconstruction resulting from this Least Squares solution. You should use the functions `solveModifiedLS()` and `reconstructPatch()` you just created and store all the reconstructed patches in a matrix.
4. Use the resulting matrix from step 3 and convert the reconstructed patches into overlapping blocks of size `blockSize` in an image. You should use the provided function `col2nd()`. For details, please read Section 3.
5. Finally, store the resulting image as a column vector to the output `y` of the function.

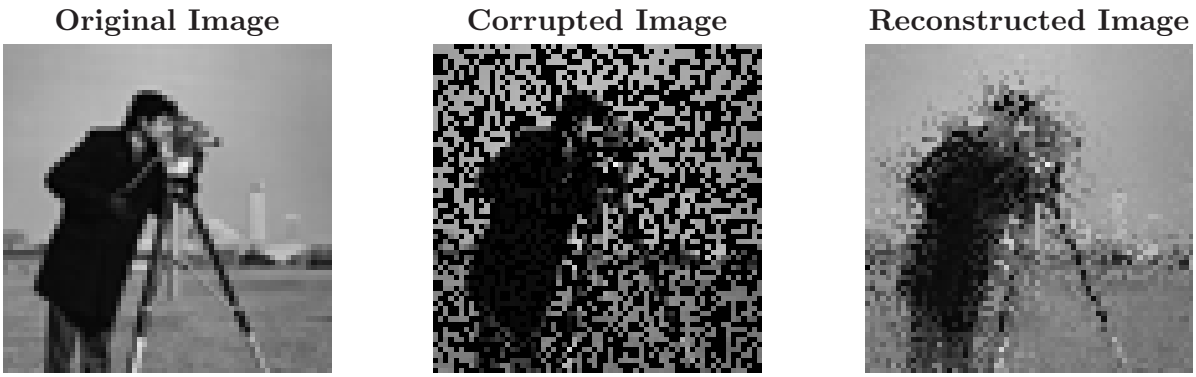

Figure 4: Example reconstruction using the script provided below.

Before proceeding, test this function using the sequence of commands:

```
x = im2double(imread('cameraman.tif'));
x = imresize(x,[64,64]);
z = x;
z(randperm(numel(z),round(0.5*numel(z)))) = 0;
dictionary = [ones(64,1) rand(64,64)];
y = reconstructImage(z,dictionary,[8,8]);
figure, subplot(1,3,1); imshow(x); title('Original Image');
subplot(1,3,2); imshow(z); title('Corrupted Image');
subplot(1,3,3); imshow(reshape(y,[64,64])); title('Reconstructed Image');
```

The result should look similar to the result presented in Figure 4. Note that the reconstruction performance is not very good, since we are not using patches and the dictionary is essentially a dictionary of random vectors. Nevertheless, one can observe that the missing pixels are filled with meaningful values to best represent the original image.

- `function [reconstructedImages,PSNRs] = ...`
  `reconstructStudentImages(corruptedImages,dictionary,blockSize,origImage)`

  This function accepts as inputs the matrix `corruptedImages` whose columns contain vectorized versions of a set of corrupted images, the dictionary of patches `dictionary`, the specific `blockSize` for the provided dictionary and the original image (matrix) `origImage` we are trying to reconstruct. It returns two outputs. The first one is the matrix `reconstructedImages` whose columns contain the reconstructed images and the row vector `PSNRs` which stores the PSNRs between each reconstructed image and the `origImage` which is provided as input to the function.

In this function, you should:

1. Create a new dictionary only with the columns of the provided dictionary that are linearly independent. (It is ok to use the built-in function `rref`).
2. Pre-append a column of all ones to the updated dictionary. This is necessary because the columns of the dictionary are normalized and they are only able to capture variations around a mean value. This column of ones serves as a way of being able to estimate the mean intensity of the image, on which the variations captured by the dictionary will be added to best represent the final image. In mathematical terms this is equivalent to,

$$\mathbf{x}_p = D \cdot \mathbf{a}_p + c = \begin{bmatrix} \mathbf{1} & D \end{bmatrix} \cdot \begin{bmatrix} c \\ \mathbf{a}_p \end{bmatrix}, \tag{5}$$

   where c denotes a constant representing the mean intensity in the image patch $\mathbf{x}_p$.
3. Reconstuct each one of the corrupted images and store the reconstructed images in a matrix of the same size as `corruptedImages`. Again, each column of the resulting matrix should be one reconstructed image. You should use the function `reconstructImage()` you just created. **Note**: This function accepts the image as a matrix and returns it as a vector.
4. Finally, compute the PNSR between all reconstructed images and the original image `origImage` and store the result to the output row vector `PSNRs`. You can use the function `computePSNRs()` that you created in the Introduction part of the Lab Project.

5 Lab Project - Submission Instructions

Once you are finished with the implementation of all parts of the Lab Project you will need to submit the Lab Project as a whole. **Note**: Each part of the Lab Project is independent, so if you haven't done some of the steps you can still run subsequent steps without the code crashing, so if you are not able to complete everything, submit what you have, **BUT make sure that the submitted files are able to run without reporting any errors**. It is better to omit a function completely than including it, if it reports an error.

Your submission should include the following:

1. A zipped (compressed) folder containing all the files that are required to run the Lab Project. You may only omit the `Student_Images` folder that we provided you with.

2. A published version of your code in html format. In order to produce this, go to the "PUBLISH" tab of MATLAB and click on the "Publish" button. Let the code run and produce all the results (this might take some time). Now, you should have a folder named "html" under your MATLAB working directory. Rename this folder as "html_your_name", zip (compress) it and submit this compressed folder as well. **Note**: **For students working in teams you should submit an html folder for each person in your team**. Therefore, you should publish your code using a different student number for each student in your group and then submit both resulting and properly renamed html folders.

3. For groups of students, only one student should submit all material of the Lab Project for all students in the group under his Canvas account. The grades will be assigned accordingly to all students in the group.