# Blurred Image Transform With Least Square Methods

Raaghav Thatte, Reese Karo, William Mahnke

May 2024

## 1 Introduction

In today's digital era, facial recognition technology is widely used in smartphones, such as Apple's products, for unlocking devices, authorizing payments, and more. It also plays a significant role in security systems.

One way to alter images, such as for enhancing privacy, is through image blurring. This can be achieved by converting an image into a matrix using the OpenCV package in Python. The dimensions of this matrix depend on the image's orientation—a tall photo results in a tall, skinny matrix, while a wider photo produces a shorter, wider matrix. Our project aims to apply Gaussian and median noise to blur an image. We will then attempt to reverse this blurring effect to restore the original image. The objective is to compare different least squares methods to determine which one most accurately reconstructs the original image. Though these facial recognition systems use more developed and advanced methods, we want to show that similar technique can be used with least squares.

Will Mahnke took the role of coding, ensuring we could see the transformations using different methods such as the normal equation, QR factorization, and householder reflectors. Raaghav Thatte took the role of creating the presentation for the slides, and Reese Karo took the role of writing the report. Overall, as a group, worked with eachother to help develop each part of the project to ensure quality and accuracy in the coding, presentation, and report, therefore we are all leaders of the project.

## 2 Methodology

The *least squares solution* is a tactical mathematical tool used to find the closest possible answer to a set of overdetermined equations. An overdetermined system of equations is a set that has no exact solution, where $x \in \mathbb{R}^n$ in the following form:

$$Ax = b$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. In matrix form, this reads as:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

The least squares solution, $\bar{x}$, comes from minimizing the difference in Ax = b, which we can call the *normal equation*.

$$\arg\min_{x \in \mathbb{R}} \|\mathbf{Ax} - \mathbf{b}\|_2^2 = (Ax - b)^T (Ax - b) \tag{1}$$

From above, we can retrieve the least squares solution by taking the gradient of 1 and setting it to 0. We have,

$$\bar{x} = (A^T A)^{-1} A^T b \tag{2}$$

This method, coupled with the QR factorization of our blurred image matrix A, is the other method we will be choosing to transform the blurred matrix back to the original. QR (full) factorization comes from splitting the matrix A into Q, a square orthogonal matrix of size mxm, and R which is the same size of A, where the first nxn submatrix is upper triangular, and the lowr k rows(k=m-n) and n columns are 0's. Thus we can decompose A as $A = QR$. Lastly we use household reflectors on the QR factorization of the matrix we have blurred to determine if a blurred matrix could be ill conditioned, and could return a more accurate result. The main idea to note, is the fact we need the true image solution in order for this method to work. When the true solution is unknown, more advanced methods are needed, and are discussed in 6.1

# 3 Computational Setup

The OpenCV package allows us to edit and transform an image in many ways, including the blurring effect on an image. This is better known as smoothing or filtering images. The OpenCV package has documentation that can be accessed for more information. We used this documentation to help create the code to transform the image as a blurred image. The blurring, or smoothing effect comes from using a normalized box filter. The cv.blur(image, ksize = (x,y)) function takes in an image that will blurred, and convolves the matrix with the kernel K, such that

$$K = \frac{1}{x * y} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{3}$$

We will not go into detail of convolving with two matrices, but can be found here, as it is ouside the scope of this class. Essentially, the matrix is convolved

by the following equation, where K is our Kernel, and our image matrix A(x,y):

$$K * A(x,y) = \sum_{i=-x}^{x} \sum_{j=-y}^{y} K(i,j)A(x-i, y-j) \tag{4}$$

The code below shows how one can use OpenCV to transform an image into a matrix variable to be changed. By using cv.cvtColor(image, cv.COLOR_BGR2GRAY), we can ensure the image is in black and white, which ensures the array is two-dimensional.

```
# importing packages
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
from functions import *

# importing image, changing to grey scale
image = cv.imread('fozzybear.jpg') # or any file name

# changing the image to all grey to mitigate working with colors
gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
print(gray_image.shape)
```

Listing 1: Image transformation with OpenCV

We will be working with the "fozzybear" image from the Muppets shown in figure 1.


Figure 1: Gray Fozzybear

# 4 Code

We first performed sanity checks using the 'np.linalg.lstsq' on the full matrix at once. Looking at the documentation for the function, we performed a second

sanity check, using the same function for each column of the matrix, as a proof of concept. Once we saw that the method worked via iterating through each column of the original (gray) image, we wrote a separate Python file including functions to find the least squares solution via the normal equation, reduced QR factorization, full QR factorization, and QR factorization using the Householder reflector. Once we obtained the matrix (of column vectors of least squares solutions) we used right-multiplied the matrix with the blurred image to produce our restored image. For each initial test we used the method with the blurring set to (25,25) [‘cv.blur(image, (25,25)‘], a significant even blurring in both axes.

When testing the normal equation method, the restored image was essentially a bunch of randomly scattered pixels that didn't resemble the initial image. Our initial thought was to decrease the blurring amount. After testing the method multiple times with increasingly less blurred photos (all the way down to (5,5)), we received the same results every time. This suggests that the conditioning of the matrix, and problem as a whole, makes the use of the normal equation nonviable.

Our initial code for reduced QR factorization, heavily inspired by Professor Park's notes, showed success for de-blurring the gray image up to a blur setting of (27,27). However, any blurring past that resulted in the transformed photo sharing the same qualities as the original photo but the color (relative to the gray-scale) being off. The improvement to the limits of using the normal equation, as low as the bar was, show that reduced QR factorization has better conditioning than the normal equation. To improve our runtime, we substituted the for loop for ‘np.linalg.qr‘ since using built-in packages significantly improves runtime. To further speed up the process we also only performed the factorization once before iterating through the columns where we just had to use ‘np.linalg.solve‘ to find each vector. The changes to the code showed massive improvement, with the runtime being cut from 17 minutes to less than 10 seconds.

Implementing full QR factorization, inspired by Professor Park's notes, showed better conditioning than the reduced method. At around (49,49) "blurring", the transformed image begins to look slightly faded and resembles the discoloration that appeared when testing reduced QR. This faded effect was magnified until the method breaks at (64,64) where it looks like the results of using the normal equation. The process of streamlining the code followed the same steps as streamlining the reduced QR code, with the runtime from 20 minutes to less than 10 seconds.

Implementing QR factorization using the Householder reflector, again inspired by Professor Park's notes, showed similar conditioning than full QR factorization. At around (36,36) "blurring", the transformed image starts looking faded similar to the results from full QR with the method finally breaking around (57,57). To ease the implementation of the QR using the reflector, we created a helper function to perform the QR factorization and then iterated through the columns with the matrices. The process of streamlining the code was pretty much identical to that of reduced and full QR, where we cut the runtime from about 3 hours to less than 20 seconds.

# 5 Results

In this section, we will show different results for different blurring strengths for each of the methods.

## 5.1 Normal Equation Results

When the blurring is set to (5,5) we see that the normal equation no longer holds due to the ill-conditioned image matrix.: Due to the immediate failure

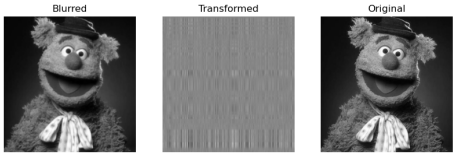


Figure 2: Normal equation, Blurring set to (5,5)

in transformation from the normal equation, this motivates the usage of QR factorization.

## 5.2 Reduced & Full QR

Blurring strength set to (27,27) for reduced QR, where Q $\in R^{mxn}$, the same size as our blurred matrix, and R $\in R^{nxn}$

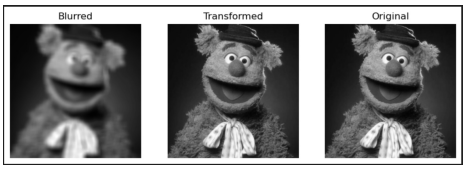


Figure 3: Reduced QR with filter size (27,27)


Figure 4: Full QR with filter size (27,27)

From the two figures 3 and 4, the results ended up being the same, and they both returned the original image. Compared to just the least squares normal equation using 2, this was a much stronger method.


Figure 5: Full QR with filter size (49,49)

For the last full QR factorization, we convolve the gray image with a filter of size 49 by 49, and we see the image is slightly fainter than the original, but we still get the original shape with sharpness.

### 5.3 Household Reflectors

lastly, we take a look at the results of Household reflectors.


Figure 6: Household Reflectors (36, 36)

It can easily be seen from figure 6, we had to use a lower sized filter to achieve similar results to the full QR with a filter size of (49,49).

## 6 Conclusion

We conclude this report with the understanding that different least squares methods result in varying transformations. Notably, the full QR factorization method consistently delivers the closest results in terms of replicating the sharpness of the original image. Smoothing the image through convolution with a smoothing or Gaussian kernel—dependent on the size of the image filter—aligns with our theoretical predictions of restoring the original image. Results from the normal equations approach are generally less reliable due to issues with ill-conditioned matrices. However, this issue is effectively mitigated by decomposing the image matrix into Q and R components and reformulating the normal equations. This QR method provides better results and proves superior to the Householder method in terms of accuracy and stability.

### 6.1 Future Work

The limits of the least squares solution that a vector is able to transform a blurry image back to its natural state with no blurriness. This turns our at-

tention to the use of other applications to de-blur images with Singular value decomposition (SVD), and neural networks (NN's). Recent studies have delved into sharpening blurry images by training neural networks using lightly blurred and highly blurred images, using a numerical distance to judge how close the de-blurred image is to the ground truth image. This creates a demand for creating innovative methods to de-blur images. Another note would be to try repeated QR and household reflectors to get closer to the original image for the transformed image, iteravely.

We also want to thank Professor Jea-Hyun Park for inspiration, and householders, and his countless hours put into the class for our learning.