Lear, Michelle (Stud. DKE)

i6240491

**Image and video processing**

**Project 2**

Table of Contents

[1. Image Degradation 2](#_Toc104628194)

[1.1. Degrading the Image And Adding Noise 2](#_Toc104628195)

[1.2. Removing Noise 2](#_Toc104628196)

[1.2.1. Remove Motion Blur – Direct Inverse Filtering 2](#_Toc104628197)

[1.2.2. Remove Motion and Additive Noise 3](#_Toc104628198)

[1.2.3. Minimum Mean Squared Error – Motion Blur 4](#_Toc104628199)

[1.2.4. Minimum Mean Squared Error – Motion Blur and Gaussian Noise 4](#_Toc104628200)

[2. Image DCT – Hidden Message 5](#_Toc104628201)

[2.1. Watermark Insertion 5](#_Toc104628202)

[2.2. Watermark Detection 8](#_Toc104628203)

[3. Morphology 9](#_Toc104628204)

[3.1. Counting 9](#_Toc104628205)

[3.1.1. Counting Oranges On A Clear Surface 9](#_Toc104628206)

[3.1.2. Counting Oranges on a Tree 9](#_Toc104628207)

[3.2. Granulometry 9](#_Toc104628208)

[4. PCA – Recognition 10](#_Toc104628209)

[4.1. Eigenfaces 10](#_Toc104628210)

[4.2. Reconstruction 12](#_Toc104628211)

[4.3. Reconstruction With Different Eigenface 14](#_Toc104628212)

# Image Degradation

## Degrading the Image And Adding Noise

To degrade an image, it is necessary to have a degradation function. In this case, where are variables from a mesh grid made from the width and height of the image, are constants that are chosen – these constants can be any number greater than 0, the higher the greater the motion blur – and is an imaginary number. The values chosen were and . These values were found to apply a visible motion blur without degrading the image too much.

To actually apply the degradation function, the image has to undergo certain changes first; it has to be of a numerical type so the function can be applied to its pixels. Then, we can apply the degradation function to get some value . After getting , we need to find the Fourier Transform of the image to multiply with . applies the degradation to the image and gives us an image with motion blur.

Graphical user interface

Description automatically generated with medium confidenceThe next step was to apply gaussian noise to the new blurred image. This was done using the inbuilt function, “random\_noise” with and . These values for mu and sigma were found to apply the (objectively) perfect amount of noise to the image; the noise is visible but not overpowering.

Figure 1: Original, Motion and Gaussian Blur

Figure 1 above shows the original, motion and gaussian blurred images. The motion blur is visible in the middle plot of the figure which indicates that the steps mentioned in the above paragraphs were successful. Furthermore, after adding the gaussian blur, the image becomes darker but the motion blur is still visible.

## Removing Noise

I had initially done the whole of question 1 in grayscale and then I reread the description and it specifies that it must not be in grayscale. I tried to do the filtering in colour but I was unsuccessful so I decided to leave in what I had managed to do previously with the grayscale. The code used for grayscale can be found in the file “ImageDegradation1.py” and the attempted colour code can be found in the file “ImageDegradation.py”.

### Remove Motion Blur – Direct Inverse Filtering

To remove the motion blur, a direct inverse filtering method was used. This is a type of high pass filter so it is sensitive to noise but for this task there is no noise so that should not be a problem. The formula, where is the blurry image and is the degradation function, gives us a deblurred image which hopefully resembles the original image. In this A bird flying over a tree

Description automatically generated with low confidencecase, the following image was obtained after applying the formula to the blurred image.

Figure 2: Direct Inverse Filtering for Motion Blur

It appears to still be blurred after the transformation which is strange because the code should work. I am not sure why this is but I have attached the image to show what I tried.

### Remove Motion and Additive Noise

A close-up of a bat

Description automatically generated with medium confidenceWhen there is additive noise in the image, the formula for inverse filtering becomes where is the added noise. This extra variable in the formula becomes a problem when is zero because becomes very large which causes an amplification of noise even if it was not there initially. This can be seen in the figure below. The expected deblurred image seen in Figure 3 is actually extremely noisy. This is the effect of the added variable in the formula. The Wiener filter is more suited for deblurring images with additive noise.

Figure 3: Denoised Motion and Gaussian

It is quite difficult to see the contents of the image as it is very dark but if you are in a dark environment, you can see that the bird in the image. The image appears to be black and this is because of the amplification of noise described above.

### Minimum Mean Squared Error – Motion Blur

The formula for the minimum mean squared error is the Wiener filter’s transfer function , where is the degradation function, is the noise power spectrum and is the original images power spectrum. These values are easily calculated using Pythons numpy library.

A bird flying over a plant

Description automatically generated with medium confidenceFigure 4 shows the motion blurred image (seen in Figure 1) after Wiener filtering. The image has significant decrease of motion blur and seems to have unnoticeable difference from the original image.

Figure 4: Wiener Filtering of Motion Blur And Gaussian Noise

### Minimum Mean Squared Error – Motion Blur and Gaussian Noise

The same transfer function is used for the image with motion blur and gaussian noise. The only difference is that we include a constant K which approximated the ratio of the noise power spectrum and the original power spectrum. K is calculated by finding the average of their true ratios using the formula for signal-to-noise ratio , where is the original image and is the noise image. In the end, the formula for the transfer function is . This resulted in the following image.

A picture containing chart

Description automatically generatedFigure 5is quiet noisy, more so than before the filtering. I am not sure why this is the outcome of the image as I expected clearer image as in Section 1.2.3. Although, the output is not the best, it is better than with the direct inverse filtering which is expected.

Figure 5: Wiener filtering of Motion and Gaussian Blur

# Image DCT – Hidden Message

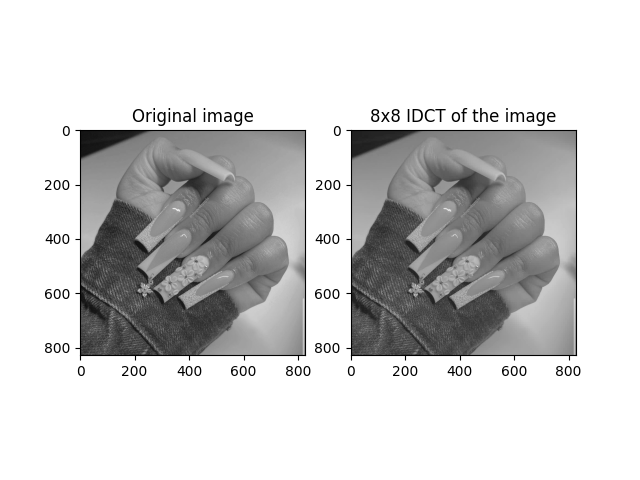
## Watermark Insertion

Chart

Description automatically generated with medium confidenceTo create an invisible watermark in an image, the following steps were taken: firstly, compute a block wise 2D Discrete Cosine Transform (DCT) on a grayscale image. Figure 6 shows the DCT of the entire image on the left and on the right, a zoomed in portion of the DCT.

Figure 6: DCT

After computing the DCT, I had to choose K coefficients with the largest magnitude to keep their frequencies and the rest would be changed to 0. This was done via thresholding. A series of threshold values were tested and ultimately, a threshold of 0.01 was found to result in an inverse DCT that looked the same as the original image (as seen in Figure 7).

Diagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generated with low confidenceFor arguments sake, Figure 10, Figure 8, and Figure 9 show the IDCT for thresholds 0.02, 0.1 and 0.5 respectively.

Figure 7: Original and IDCT with threshold = 0.01

Figure 8: IDCT threshold = 0.1

Figure 9: IDCT threshold = 0.02

Figure 10: IDCT threshold = 0.5

At initial glance Figure 9 appears to be the same as the original but upon further inspection, one can see that the colours are slightly lighter than the original. Figure 8 is slightly blurry and so a threshold of 0.1 was found to not be appropriate and finally Figure 10 is extremely blurry and has a higher contrast than the original.

Next, I created a pseudo-randomly generated set of numbers with a gaussian distribution with and . This set of random numbers was used as a watermark for the image. The watermark was embedded into the image using the following equation where is a measure of intensity for the watermark, is the original DCT coefficient and are the randomly generated set of numbers. The next step was to decide on a value for . Through trial and error, the final value used was ; this is because for values greater than 0.02, the watermark was visible. Figure 14 and Figure 15 show the watermark and the image with the watermark embedded. As seen, the watermark is invisible after being embedded which shows that the value of alpha is appropriate.

Diagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generated with medium confidenceChart

Description automatically generatedDiagram

Description automatically generatedTo demonstrate what other values of alpha produce, below show figures (Figure 11, Figure 12 and Figure 13) with alpha values 0.05, 0.4 and 1 respectively. For Figure 11, the watermark is not very obvious but when analysing thoroughly, a grid can be seen over the image. In Figure 12, the watermark is fairly obvious as it creates a mesh-like coverage on the image and also causes a bit of blur and finally, Figure 13 most obviously shows the watermark as the image is darker than the original and the grid can be seen very easily.

Figure 11: Alpha = 0.05

Figure 12: Alpha = 0.4

Figure 13: Alpha = 1

Figure 14: Watermark

Figure 15: Image with watermark

A picture containing text

Description automatically generatedTo have a better overview of the processes applied above, a difference image was calculated. A difference image determines the changes between two images by calculating the difference between each pixel in the images and generating an image based on the result (Wikipedia, 2016).

Figure 16: Comparison

Chart

Description automatically generatedIt is quite hard to see, but the third image in Figure 16, a vague outline of the hand seen in the original image and watermarked image can be seen. This shows that there is a very small difference between the two images and further illustrates that the alpha value for the watermarking is appropriate for an invisible watermark.

Figure 17: Image Histograms

Figure 17 shows the histogram for the original image, the image with a watermark and the difference image (left to right). The watermarked image has a more dense plot than the original image. This makes sense because the watermarked image has more data than the original image even though the watermark is not visible. The difference image histogram shows that the range of difference is low therefore the inverse DCT transformation was successful as the goal is to have an image that is as similar to the original as possible.

The watermark is inserted on the non-DC coefficients for this exact reason, if the watermark is inserted where there are DC-coefficients, it will create noise and the image will appear different because it will overfit the coefficients.

## Watermark Detection

# Morphology

## Counting

### Counting Oranges On A Clear Surface

Background pattern

Description automatically generatedTo count the number of similar objects in an image, I first converted the images to black and white images. This helped with having a clear indication of the objects. Next, dilate the image further to remove possible distortions from the image (See Figure 18). Then, find the contours in the image to locate the edges of the objects and use that as a count for the image.

Figure 18: Dilated Oranges

After applying all these steps to the original image, the code outputted that there were 24 oranges in the image. This is correct which shows that the code is functional.

### Counting Oranges on a Tree

Counting the number of oranges, or any fruit for that matter, on a tree can be difficult because there is, what we can consider as, noise in the image. Meaning, in the image, we want to focus on the oranges, but there are leaves and other objects in the image that make it difficult to specify what to count. A way to get around this is by using morphology techniques and a range of colours that we specify.

In this case, we are looking for orange objects in the image, therefore we have a range of [1,170,100] to [18,255,255]. This range will help filter out the other colours and noise in the image. After this, we apply the same erosion, dilation and contour techniques mentioned in the previous section. An extra step here is to filter out the smaller contours in the image to ignore the noise. This left us with Figure 19 and it counted 58 oranges in the image.

A picture containing nature, rain

Description automatically generated

Figure : Orange Tree After Changes

## Granulometry

Granulometry “deals with determining the size distribution of particles in an image” (Gonzalez & Woods, 2018). This is useful when particles are not completely separated and would therefore be more difficult to count. With the use of morphology, we can find the frequencies of these particles “without having to identify and measure individual particles”. (Gonzalez & Woods, 2018).

A picture containing indoor, colorful, several, light

Description automatically generatedChart, line chart

Description automatically generatedIn order to do this, one has to take the following steps: first we need to define the approximated diameter, the increasing factor for the diameter and the number of iterations. After defining those, one can begin with the changes. The image needs to be eroded, this removes certain pixels from the object boundaries and helps with distinguishing between different particles. Next, dilate the eroded image, this adds pixels back to the particles but the particles still have distinct boundaries that were found in the previous step. The surface area of the particles will be the sum of the image and the frequencies will be the absolute value of the difference between adjacent elements. Apply these steps for the number of iterations that were previously defined and the frequencies have been calculated.

Figure : Lights Image for Granulometry

Figure : Granulometry

Figure 20 shows the plot of the frequencies of the lights in the image (Figure 19). The plot seems to be accurate as there are several large lights in the background as well has smaller ones in the foreground which are shown in the graph as it staggers up and down in frequency.

# PCA – Recognition

## Eigenfaces

A picture containing text, gallery

Description automatically generatedAn eigenface is an image of a person’s face that can be added to the mean of their face to create a new facial image (Mallick, 2018). This can be calculated by getting the average of a set of images of one person’s face, subtracting this from each image in the set and then using this result to calculate the covariance matrix of the face. From the covariance matrix, the eigenvalues and eigenvectors can be found which finally give us the eigenfaces.

Figure 22: Eigenfaces for Kid

Figure 21 shows the eigenfaces for the kid image. Each of the images in the figure have different focuses but one common visible feature are the glasses.

Graphical user interface, application

Description automatically generated Figure 22 shows the eigenfaces for the old man. These eigenfaces seem to have more data on the face shape than the kid image. This may be because the original image of the man was more centred than the kid image and thus the eigenvalues were higher for the old man images.

Figure 23: Eigenfaces for Old Man

A picture containing text, gallery

Description automatically generatedFinally, Figure 23 shows the eigenfaces for the woman. These see to be the noisiest and the most merged. This may be because the background of the original image had similar colours to that of the woman’s skin. A lot of features are mixed up between the eigenfaces, for example, the eigenface for the image where the woman has long hair, the main feature is a beard (which is not seen in the image).

Figure 24: Eigenfaces for the Woman

## Reconstruction

Reconstruction of a facial image is done by adding one or multiple eigenfaces to the mean of an image. When this is done with the three facial images given, the following are the results.

Figure 25: Reconstruction of Kid's face



Figure 24 shows (from left to right) the original, reconstruction with 1 eigenface and reconstruction with 6 eigenfaces of the kids face. As we can see, the two reconstructions do not vary a lot but the original does not have a lot of the features seen in the reconstruction. This is because the reconstruction is done using the mean of the images, therefore it has data from other images that the original does not have. This is not a great example as the mean of the kid’s image is filled with data which creates a bias for the reconstruction.

Next, the reconstruction of the old man’s face can be seen in Figure 26. Like the kid’s reconstruction, very little difference can be seen between the two reconstructions (1 eigenface – left, and 6 eigenfaces – right). But a difference that can be seen here is that not all of the eigenfaces features can be seen in the reconstructions. This suggests that the eigenfaces used to reconstruct the facial images have a lower eigenvalue than the kid’s images as there is less data present in the images.

Figure 26: Reconstruction of the woman's face

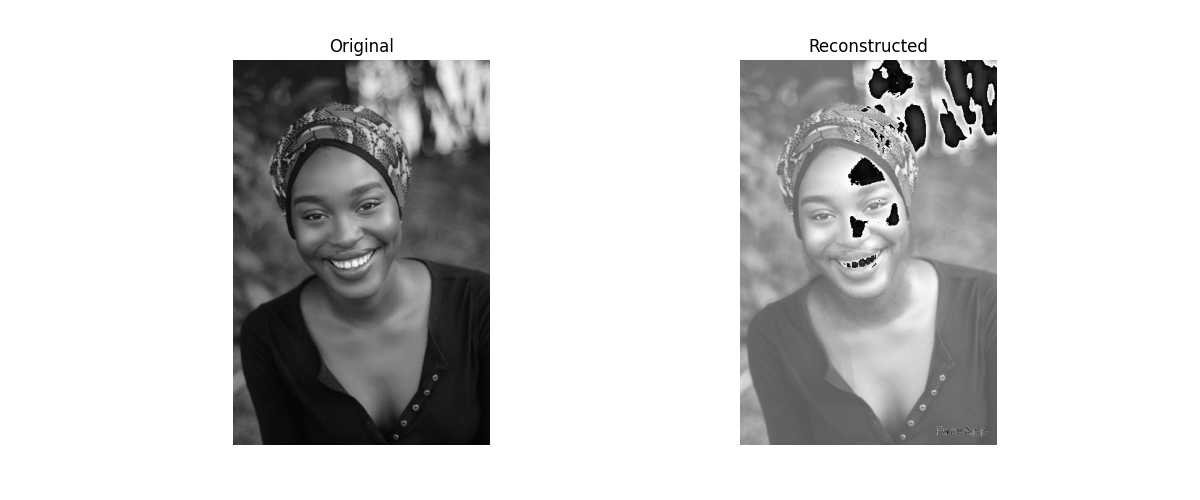
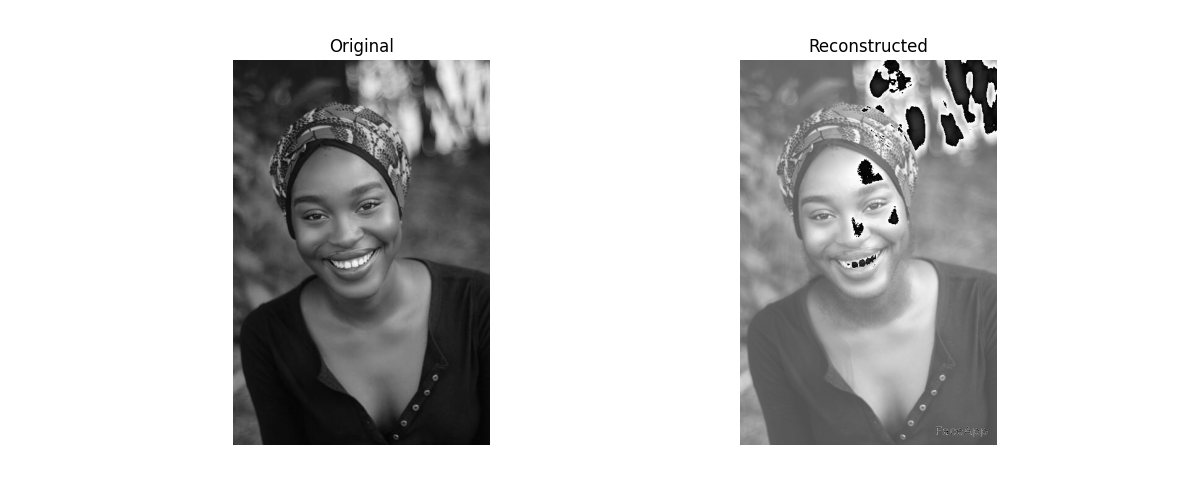
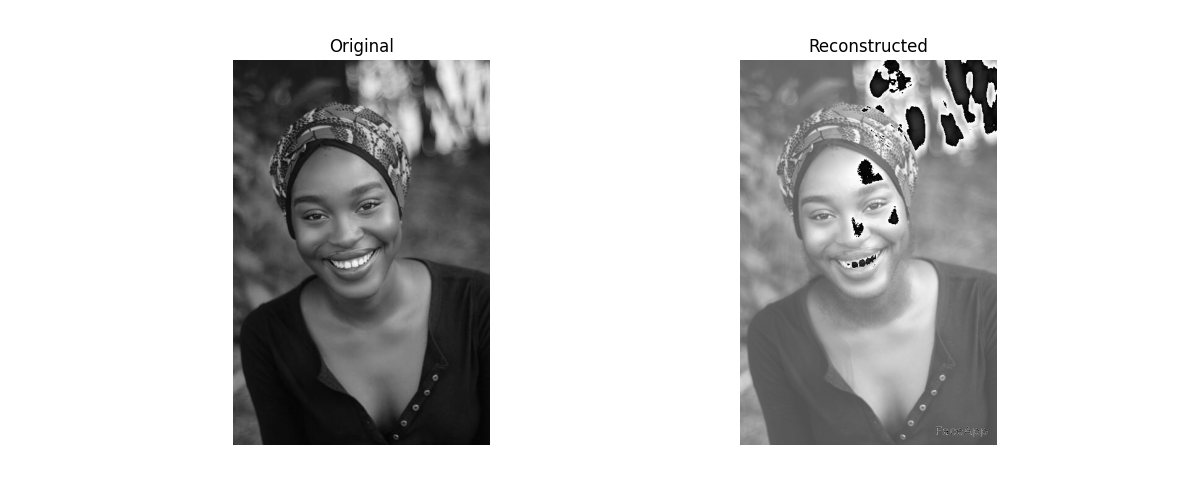


Figure 27: Reconstruction of the old man's face

Lastly, Figure 25 is the reconstructions of the woman’s facial images. These reconstructions were the most peculiar as the same program was used for all three sets of images but only this one came out with noise. The reconstructions show black marks on parts of the images that are not present in the original image nor in the eigenfaces. I do not know why this happens. On the reconstruction with 6 eigenfaces – the image on the far right – there is more of this black noise visible. This may be because all of the eigenfaces combined with the mean of the image amplify this.

## Reconstruction With Different Eigenface

For each eigenvector of a facial image, there is a corresponding eigenvalue. This eigenvalue tells us how much data is stored in the eigenvector so the eigenvector with the highest eigenvalue is the principal component where the most data lies. Taking this into account, when attempting to reconstruct a facial image with a different set of eigenfaces the eigenface with the highest eigenvalue will be used to reconstruct with a different face.