Group III

Image Analysis Group **Report**

CS3IA16: Image Analysis

University of Reading

Created by:

Shavin Croos: 27015244

Dan Keep: 28016852

Uthman Maigari: 28017661

Date of Completion: 27/10/2021

Actual hrs spent: 9 Hours

Assignment Evaluation:

(1) The assignment was fairly easy to understand and comprehend for the concepts.

(2) One thing that could be better about this assignment is if one of the lecturer's faces was heavily distorted and we would have to try to find out who the lecturer was by carrying out image analysis.

Abstract

The objective of the coursework is to enhance a distorted image which was caused by a combination of periodic noise and random noise. This coursework reviews the fundamentals of image analysis and aims to provide basic information regarding the features, application, and tools. An algorithm (Fast Fourier Transformation) was used to transform images into frequency domain by compressing the image to cosine and sines, which the image is transformed into an array of it's values from low to high. Removing the period noise is done by completing a filter process then implementing the inverse of the fourier transformation. Next the random noise is removed by taking the filtered image from the periodic noise removal process then by performing a gaussian blur on the image. Median filter is applied to the image which will sharpen the image.

Introduction

Image analysis has been a part of the computer science field since the 1950s, initially as a branch of artificial intelligence and robotics [1]. Image analysis is where useful information is drawn out from images through the usage of image processing techniques [1]. The field was developed between 1950 and 1970 [1]. The aim of this coursework is to enhance and build up our understanding, the process used for image enhancement in the spatial and frequency domain.

The main objective is to develop an algorithm in both spatial and frequency domain to improve the quality of the distorted image by diminishing the noise (periodic noise and random noise). The mean square error (MSE) will evaluate the result and it is used to compare image compression quality. It represents the cumulative squared error between compressed and original images. The two filters that were used for the coursework are the median filter and high-pass filter.

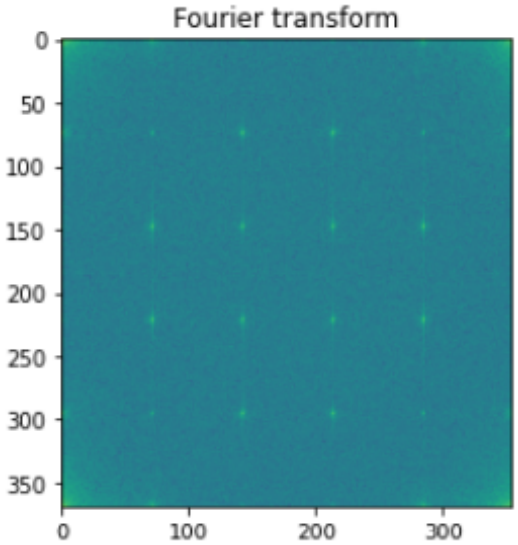
Methodology

In order to transform and enhance the distorted image to almost be like the original image, the fast fourier transformation (FFT) was first used. The FFT is an algorithm that transforms images into the frequency domain by decomposing an image into sines and cosines of varying amplitudes and phases, showing patterns that repeat in the image [2]. To begin with, the FFT functions are imported from the FFT package in the line `from scipy import fftpack`. The distorted image is then imported and assigned to the variable `im`. After this is done, the distorted image is called into the FFT function and is assigned to the variable as `im_fft = fftpack.fft2(im)`, where the image is converted into an array of its sine and cosine frequency values from low to high.

At the same time, LogNorm is imported as `from matplotlib.colors import LogNorm` to normalise the values in-between vmin and vmax to 0 - 1 on the log scale. If the vmin and vmax values are not provided, `autoscale_None(value)` is called on the first use, which will initialise the values. After this, the FFT graph is produced via the `plt.imshow()` function by calling two variables inside that function, which are `np.abs(im_fft)` and `norm=LogNorm()`. `np.abs(im_fft)` provides the absolute value of every value in the array `im_fft` and

`norm=LogNorm()` plots the transformation onto a graph, which is shown as a logarithmic colour spectrum map. The resulting graph shows the concentration of light at the corners and the noise is shaped like a grid throughout the image.

```
# Import disorted image and save as im.
im = plt.imread('dogDistorted.bmp')

# Use fast fourier transform to convert the image into an array of its sine and cosine
frequency values from low to high.
from scipy import fftpack
im_fft = fftpack.fft2(im)

# Import LogNorm to normalize value inbetween vmin and vmax to 0-1 on the log scale
# If vmin and vmax are not provided, autoscale_None(value) is called on the first use
which will initialize the values.
from matplotlib.colors import LogNorm

# np.abs provides the absolute value of each value in the array im_fft, we must use
LogNorm to plot the transformation to a graph.
plt.figure()
plt.imshow(np.abs(im_fft), norm=LogNorm())
plt.title('Fourier transform')
# We can see the concentration of light at the corners and the noise shaped like a grid
throughout the image.
```



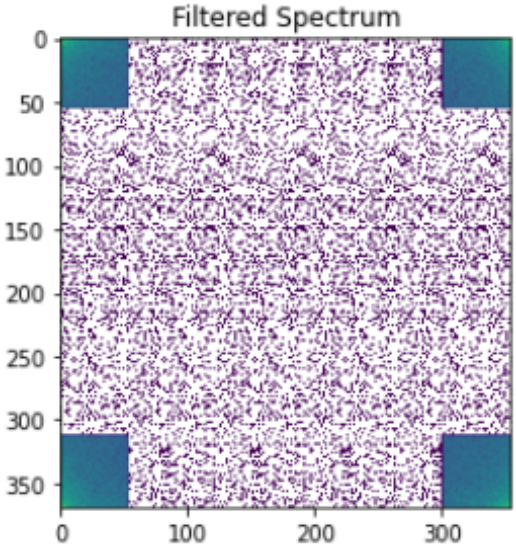
After this is done, the image is then passed through a frequency domain filter that is designed to remove the periodic noise. To begin with, the amount of light that is to be kept at the edge of the image is defined as a percentage in the line `keep = 0.15`. This means that 15% of the light will be kept. After this is done, the `.shape` function will obtain the number of dimensions and the number of values in each dimension of the 2D array `im_fft` and assign them to the frequency

indices of `u` and `v` respectively. Utilising the values `u` and `v`, the sections are sliced out of the array for each frequency index.

For the index `u`, the section sliced out is between the values of `u` size of the array times the keep value, 0.15, which gives 15%, and `u` size of the array times 1 subtracted by the keep value, giving 75%. The dimensions inside that slice of the array are set to 0. The process is repeated with the frequency index `v`, only this time, a colon is included to search through all dimensions (u) and slice the values in each of these dimensions like we have previously, setting everything between 15% and 75% to 0. These steps effectively remove all the periodic noise that was present on the image.

The filtered image is displayed as a graph once again by using the same aforementioned function and its called variables as before, `plt.imshow(np.abs(im_fft), norm=LogNorm())`.

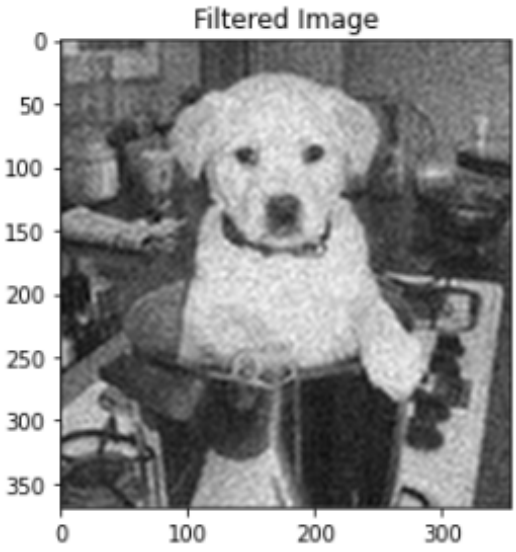
```
# Define the amount we keep at the edge of the image as a percentage.
keep = 0.15

# Use .shape to get the dimensions and values of the 2D array im_fft, u v for
frequency index.
u, v = im_fft.shape

# Slice out section of array between (u size of array x 0.15 gives 15%) and
# u times (1 - 0.15) = 75% of the image and set these values to 0
# cast as an integer as it cannot slice floating point.
im_fft[int(u*keep):int(u*(1-keep))] = 0

# We do the same for v except we include a colon to look through
#all dimensions (u) and slice the values in each of these dimensions.
im_fft[:,int(v*keep):int(v*(1-keep))] = 0

# Display the transform after removing the noise frequencies.
plt.figure()
plt.imshow(np.abs(im_fft), norm=LogNorm())
plt.title('Filtered Spectrum')
```



Once the filter process is complete, the filtered graph is transformed back into the filtered image by implementing the inverse of the fourier transformation in the line `im_new = fftpack.ifft2(im_fft).real`. The filtered image is then displayed using the function `plt.imshow(im_new, plt.cm.gray)`.

.

```
#Get the inverse of the fourier transformation
im_new = fftpack.ifft2(im_fft).real

#Dislay filtered image
plt.figure()
plt.imshow(im_new, plt.cm.gray)
plt.title('Filtered Image')
```



One all the periodic noise has been removed, only the random noise remains and is the next noise to be removed before we get to the final result. The process of removing the random noise begins by taking the filtered image from the periodic noise removal process and performing a Gaussian Blur on the image using the function and assigning it to a variable as `blur = cv2.GaussianBlur(im_new,(5,5),cv2.BORDER_DEFAULT)`. This is to ensure that any noise remaining is completely removed.

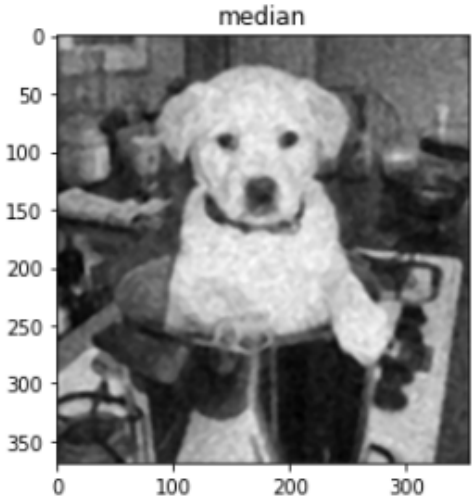
When this is done, the sharpening kernel is initialised by utilising the standard sharpening matrix in the array as shown in the line `kernel = np.array([[0,-1,0], [-1,5,-1], [0,-1,0]])`. Next, the function `filter2D` in the line `im = cv2.filter2D(blur, -1, kernel)` takes a kernel and calculates the averages of every pixel within a blurred image to enhance the detail. The sharpened image is then displayed in the line `plt.imshow(im, plt.cm.gray)`.

```
# Perform a GaussianBlur which smooths the image to further remove any noise.
blur = cv2.GaussianBlur(im_new,(5,5),cv2.BORDER_DEFAULT)
# We use the sharpening kernel.
kernel = np.array([[0,-1,0], [-1,5,-1], [0,-1,0]])
# filter2D takes a kernel and calculates the average of each pixel in a blurred
image to enhance detail.
im = cv2.filter2D(blur, -1, kernel)

# Display sharpened image.
plt.figure()
plt.imshow(im, plt.cm.gray)
plt.title('sharpen')
```

The final step in almost reaching the original undistorted image is to apply the median filter to the image. This is done by saving the sharpened image from the previous step as an array of type uint8 called `noise_img` in order to be in the correct format for the median filter. After this is done, the line `median = cv2.medianBlur(noise_img, 5)`, which contains the function `medianBlur`, takes `noise_img` as the input and calculates the median of each pixel inside the kernel of a specified size, which is 5 in this case. Finally, when the median filter has been applied, the fully transformed image is displayed in the line `plt.imshow(median, plt.cm.gray)`.

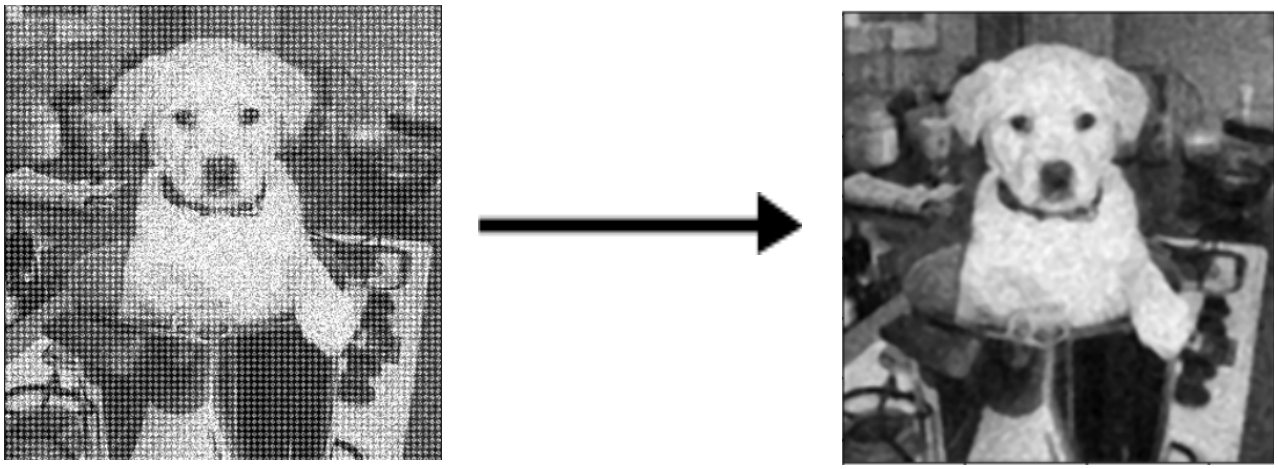
```
# Save our sharpened image as an array called noise_img of type uint8 for
correct format for medianBlur.
noise_img = np.array(im, dtype = 'uint8')

# Here median blur takes the input image noise_img and calculates the median of
each pixel inside the kernel of a specified size, here 5.
median = cv2.medianBlur(noise_img, 5)
```

```
plt.figure()
plt.imshow(median, plt.cm.gray)
plt.title('median')
# Final image
```



Results and Discussion

At the end of the transformation and enhancement processes, this is what the final image turned out as:



As we can see, the final image of the transformation looks much sharper, cleaner and better than the distorted image that was used in the beginning. The final image (right) is then compared to the original, undistorted dog image (left) as shown below:

The final product is almost spot on. The image could have been further improved by using some other filters to make the look more enhanced. After applying all the transformations and enhancements to the distorted image, the final image produced is evaluated and compared to the given original image. This evaluation and comparison is made by measuring what is known as the Mean Square Error (MSE) of the image. As shown in the undistorted image using the high pass filter, it has the effect of enhancing the edges in the images, which are associated with the high spatial frequencies [3]. Sharpening is a fundamental aspect of high pass filters in the frequency domain [3].

When calculating the Mean Square Error (MSE), we first convert the images (8 bit integer that goes from 0-255) to float point avoid it from wrapping around after reaching 255 or it will overflow, then take the dimensions of the image and times it by the number of elements, each dimension will get a total number of pixels in the image then divide by the sum of the square difference which will lead to the mean squared error. The value of the MSE we had obtained from this coursework is 1879.0096950265277, which tells us that the final product is pretty close in comparison to the original, undistorted image.

```
import cv2
from google.colab.patches import cv2_imshow
import numpy as np

temp =  [3,3]
ex = 3 // 2
ey = 3 // 2

image = cv2.imread('dogDistorted.bmp', 0)
cv2_imshow(image)
m, n = image.shape
arr = np.array(image)

filteredImage = np.zeros([m, n])
```

```
for x in range(ex, m - ex):
   for y in range(ey, n - ey):
       for fx in range(0, 3):
          for fy in range(0, 3):
              temp.append(arr[x + fx - ex] [y + fy - ey])
       temp = sorted(temp)
       filteredImage[x][y] = temp[4]
       temp = []

cv2_imshow(filteredImage)
```

Here is an implementation of the median filter. We create an empty array using np.zeros of the same size as the original image. We iterate through the pixels along the x and y axis of the image 3 from the start and up until 3 from the end to avoid overflow when applying our median array, then for each pixel iterate over our temp array and get the average of all values for each pixel. We then rebuild the picture of each median filtered pixel at a time by saving the x and y values into the `filteredImage[x][y]` array on each iteration of x and y. The temp folder is cleared afterwards ready for the next pixel.

By applying the fourier transform, we were able to visualise the noisy high frequency signals and remove them from the image. Converting the image back into the spatial domain, we could alter the pixels RGB values directly to remove further noise from the image and enhance detail through the use of median filter, gaussian blur and a sharpening kernel.

Conclusion

Based on the findings of this coursework, it can be said that the fundamentals of image analysis had been grasped. These fundamentals can be used in further applications such as detecting people over CCTV, enhancing and detecting medical anomalies or identifying number plates. As shown with the work given, image analysis can make an image available in any format requested by the user, which in our case was to improve the quality of the image. This means that information can be analysed and extracted for use in a huge variety of further tasks.

Appendix

[1] Wikipedia contributors. (2021, September 14). *Image analysis*. Wikipedia. Retrieved 27 October 2021, from https://en.wikipedia.org/wiki/Image_analysis

[2] *Fast Fourier Transform (FFT) Background*. (n.d.). L3Harris Geospatial. Retrieved 20 October 2021, from https://www.l3harrisgeospatial.com/docs/backgroundfastfouriertransform.html

[3] School of Physics and Astronomy - The University of Edinburgh. (2007). *Digital Filtering* (Revised ed.) [E-book] from https://www2.ph.ed.ac.uk/~wjh/teaching/dia/documents/filtering.pdf. School of Physics and Astronomy - The University of Edinburgh.

Link to Google Colab Code:
https://colab.research.google.com/drive/1Pd17Bfjh4b1YmN6gC-JVXsagUSXZSCjx?usp=sharing

Effort Allocation Sheet

Group: 3

Name	Contribution (%)	Note (briefly explain the contribution)	Signature
Shavin Croos	100	Wrote the Methodology and some parts of the other sections like Introduction, Results & Discussion and Conclusion.	**T.S.Croos**
Uthman Maigari	100	Documented the Abstract, Introduction and Discussion on the report.	***bm.uthman***
Dan Keep	100	Did the code for the image analysis coursework, wrote some of the Results & Discussion and Conclusion sections.	**D.Keep**