Computer Vision

- Midterm Report -

Project Report

Eungbean Lee

Yonsei University
Department of Electrical & Electronic Engineering
Digital Image and Media Lab codes:
https://www.github.com/eungbean/computervisionpractice

Contents

Chapter 1

Transformation

Rotation, translation and skew are useful operations for matching and tracking. Write a function that takes as input an image I, rotates it with an angle θ_1 and horizontally skews it with a second angle, θ_2.

Example 1.1 (Transformation Matrix)
Write the matrix formulation for image rotation and skewing (define all the variables).

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & tan\theta_2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{skew}} \underbrace{\begin{bmatrix} cos\theta_1 & -sin\theta_1 & 0 \\ sin\theta_1 & cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{rotation}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where x', y' = output pixel position, x, y = input pixel position, θ_2 = horizontal skew angle from y axis.

Example 1.2 (Transformation Practice)
Create an image containing your name written in Arial, point 72, capital letters.
Rotate the image you created by θ = 30, 60, 120 and -50 degrees clockwise.
Skew the same image by θ = 10, 40 and 60 degrees.
Comment the results.

Inverse Mapping Inverse mapping is used in most of the practical implementation.

1. locate the output image pixel grid in output space.

2. for each output pixel on the grid:
   (a) apply the inverse spatial transformation to determine the corresponding location in input space: $(u_k, v_k) = T^{-1}(x_k, y_k)$
   (b) Using the input image pixels nearest to (u_k, v_k) interpolate to get an approximate value for the input image at (u_k, v_k).
   (c) Use that value for the k-th output pixel.

Bilinear Interpolation When performing image transformation and manipulation techniques, it is often necessary to employ some sort of interpolation or filtering in order to obtain a good image quality.

We can best understand bilinear interpolation by looking at the graphic below. The green P dot represents the point where we want to estimate the color. The four red Q dots represent the nearest pixels from the original image. The color of these four Q pixels is known. In this example, P lies closest to Q12, so it is only appropriate that the color of Q12 contributes more to the final color of P than the 3 other Q pixels. There are several ways equivalent ways to calculate the value of P. An easy way to calculate the value of P would be to first calculate the value of the two blue dots, R2, and R1. R2 is effectively a weighted average of Q12 and Q22, while R1 is a weighted average of Q11 and Q21.

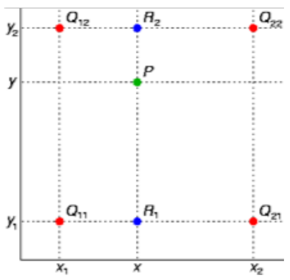


Figure 1.1: Bilinear interpolation

$$R_1 = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$R_2 = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

After the two R values are calculated, the value of P can finally be calculated by a weighted average of R1 and R2.

$$P = \frac{y_2 - y}{y_2 - y_1} R_1 + \frac{y - y_1}{y_2 - y_1} R_2$$

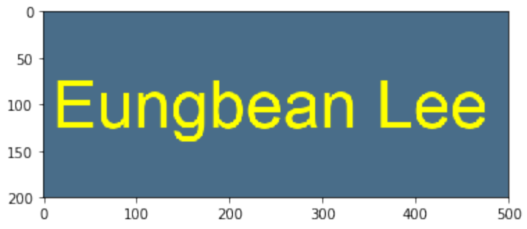


Figure 1.2: Input Image

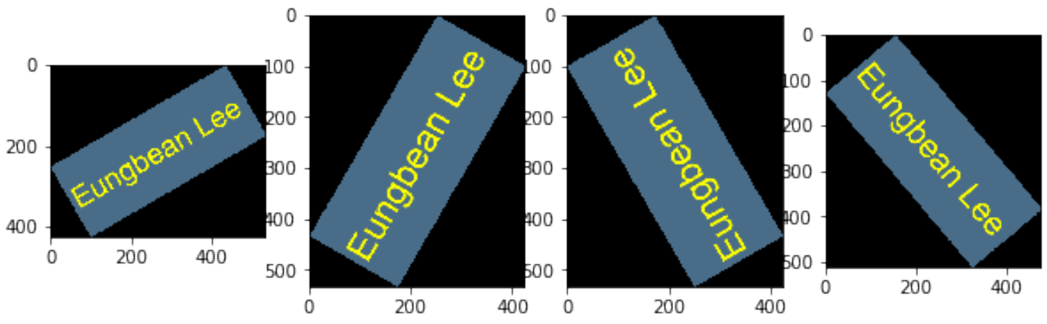


Figure 1.3: Rotation Result

Procedure

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} cos\theta_1 & -sin\theta_1 & 0 \\ sin\theta_1 & cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{rotation}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where θ_1 is a angle from x axis.

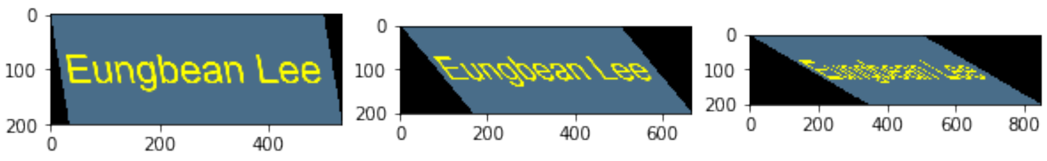


Figure 1.4: Skew Result

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & tan\theta_2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{skew}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where θ_2 is a angle from y axis.

Example 1.3 (Commutative Property)
Analyse the results when you change the order of the two operators i.e. R(S(I)) and S(R(I)), where R is the rotation and S is the skew.
Are the results of (i) and (ii) the same? Why?
Rotate the image by $\theta_1 = 20$ clockwise and then skew the result of $\theta_2 = 50$.
Skew the image of $\theta_2 = 50$ and then rotate the result by $\theta_1 = 20$ clockwise.

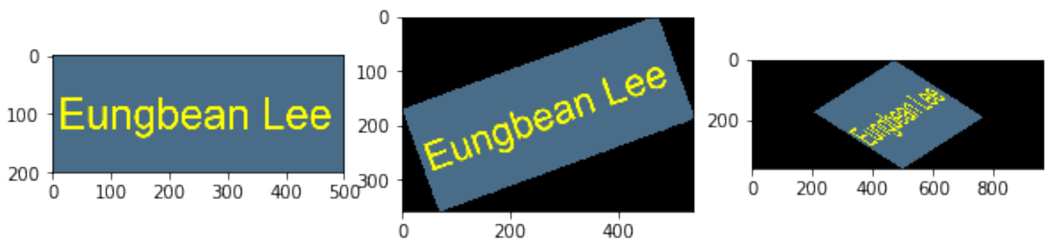


Figure 1.5: Skew after rotation result

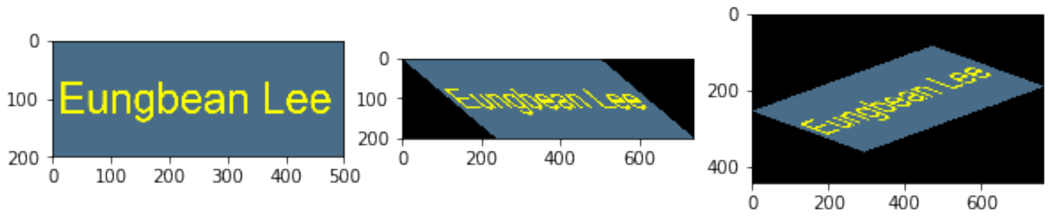


Figure 1.6: Rotation after skew result

As results above, The results are not same. Therefore, commutative property is not valid on skew and rotation transformation. For detail, the transformation matrix of skew after rotation and rotation after skew is below.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & tan\theta_2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{skew}} \underbrace{\begin{bmatrix} cos\theta_1 & -sin\theta_1 & 0 \\ sin\theta_1 & cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{rotation}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} cos\theta_1 + tan\theta_2 sin\theta_1 & -sin\theta_1 + tan\theta_2 cos\theta_1 & 0 \\ sin\theta_1 & cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{skew after rotation}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} cos\theta_1 & -sin\theta_1 & 0 \\ sin\theta_1 & cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} 1 & tan\theta_2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{skew}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} cos\theta_1 & cos\theta_1 tan\theta_2 - sin\theta_1 & 0 \\ sin\theta_1 & sin\theta_1 tan\theta_2 + cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{rotation after skew}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Thus, skew after rotation and rotation after skew transformation is not commutative.

Chapter 2

Filtering and Convolution

Image filtering allows you to apply various effects on images. In this assignment you will focus on filtering in the spatial domain, also known as convolution. Convolution provides a way of multiplying together two arrays of numbers, to produce a third array of numbers. In the image processing context, one of the input arrays is normally a grey-level image. The second array is usually much smaller, and is also two-dimensional (matrix), and is known as the convolution kernel. Depending on the designed filter and intended effect, the kernel can be a square matrix, e.g., of 3x3, 5x5 or 7x7 dimensions. Write a filtering function that takes an input image, performs convolution using a given kernel, and returns the resulting image.

Example 2.1 (Filtering function)
Design a convolution kernel that enables the computation of average intensity value in a 3x3 region for each pixel. Use this kernel and the filtering function above, save the resulting image.

The 3x3 convoluton kernel to compute average intensity value is defined as below.

$$k = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

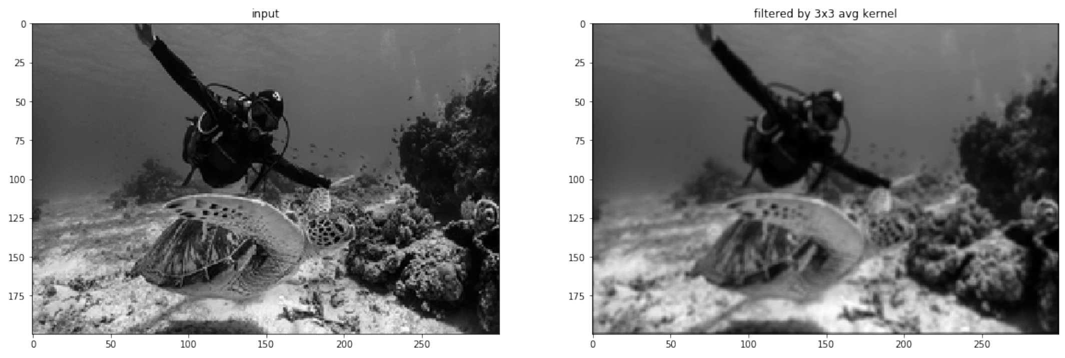


Figure 2.1: Input Image (left) and average image (right)

The result image is blurred. This kernel is known as 'box blur' in which each pixel in the resulting image has a value equal to the average value of its neighboring pixels in the input image. The kernel I applied has a property of using equal weights, so it is a form of low-pass simple version of low pass filter. This kernel is widely used to reduce image noise and reduce detail. In the result, we can also see reduction of details.

Example 2.2 (Convolution Practice)
Use the kernels provided below, apply the filtering function and save the resulting images. Comment on the effect each kernel has on the input image.

$$kernelA = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}, kernelB = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

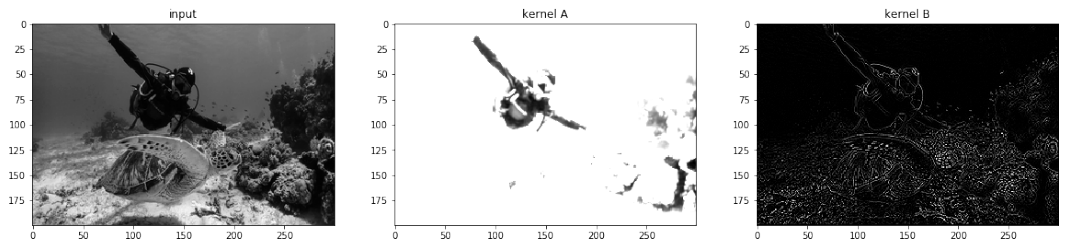


Figure 2.2: Input Image (left), kernel A (center), kernel B (right)

Kernel A has more weights on center pixel compared to the given kernel in example 1.1. However, most of image region appears to be white which is '255' in intensity. This means that image intensity value has been clipped because it is larger than 255 which is max value of uint8 type. The reason is that this kernel inflate the brightness of image because one pixel is a sum of intensities of 18 neighboring pixels. Therefore I modified this kernel with normalize.

$$kernelA = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \rightarrow Modified\ kernel\ A = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

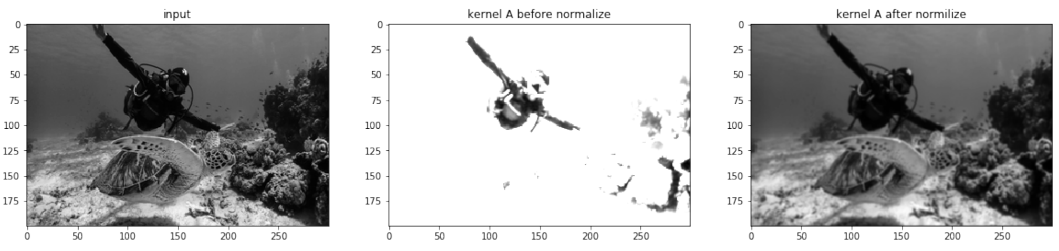


Figure 2.3: Input Image (left), kernel A (center), kernel A with Normalized (right)

The normalized kernel makes proper output of image. This kernel makes image blurry. This kernel is called 'gaussian blur' which is an approximation form of Gaussian function. Gaussian blur is a low-pass filter, attenuating high frequency signals.

Kernel B highlights the edges whose intensity value changes is large. This kernel is called 'laplacian operator'. The discrete Laplacian is defined as the sum of the second derivatives Laplace operator and calculated as sum of differences over the nearest neighbours of the central pixel. This operation is widely used in edge detection and estimation applications.

Example 2.3 (Commutative Property)
Use the filtering function for the following filtering operations:
(i) A followed by A;
(ii) A followed by B;
(iii) B followed by A.
Comment the results.

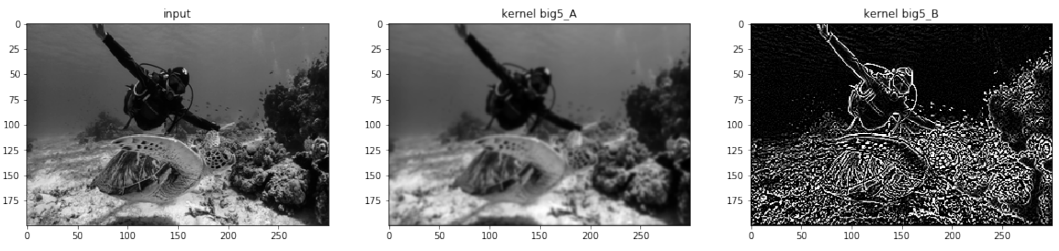


Figure 2.4: (from left) Input image, A followed by A, A followed by B, B followed by A

In this experiment, I use normalized kernel A rather than original version because original version of kernel A makes whole image white which is useless. In this result, we can see the applying gaussian kernel (A) twice is more blurrier than applying once. The reason is that overlapping convolution operation makes larger receptive field.

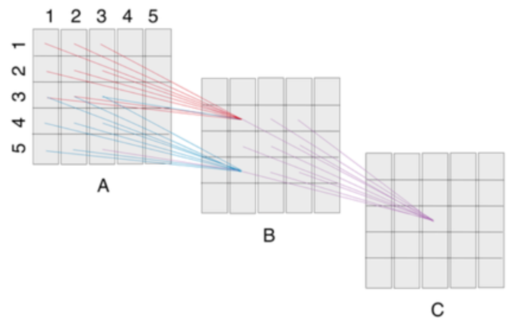


Figure 2.5: Overlapping gaussian function makes larger receptive field

Meanwhile, we can see that A followed by B and B followed by A outcome same result. This is because convolution operation has commutative property. So the consecutive operation makes laplacian image generated by kernel B more brighter by kernel A. We can observe the edges more easily with this operation.

Example 2.4 (Kernel Size)
Keeping the effect of the kernels in b) the same, discuss how to extend them to larger filter kernels 5x5 and 7x7.
Using the extended kernels repeat the operations in c).
Comment on the results and compare them with the ones obtained in c).

As the given kernel is 3x3 form of gaussian and laplacian function. So The 3x3 laplacian filter can be extend into 5x5 or 7x7 size by combination with gaussian function. This is called "Laplacian of Gaussian (LoG)" filter. The LoG or Laplacian of Gaussian filter is a two step process in which the image is first smoothed by a Gaussian kernel and then the edges are detected using the Laplacian operator. The response of the smoothing is dependent upon the size of the object and the size of the Gaussian kernel used. Because of this relationship, a 5X5, 7X7 or 9X9 kernel may be selected.

$$Gaussian\ kernel(x,y) = \frac{-1}{\pi\sigma^4}\frac{1-(x^2+y^2)}{2\sigma^2}\exp(\frac{-(x^2+y^2)}{2\sigma^2})$$

$$Laplacian\ kernel(x,y) = \frac{x^2+y^2-2\sigma^2}{\sigma^4}\exp(\frac{-(x^2+y^2)}{2\sigma^2})$$

5x5 kernel

$$Kernel\ A5 = \frac{1}{273}\begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix} Kernel\ B5 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

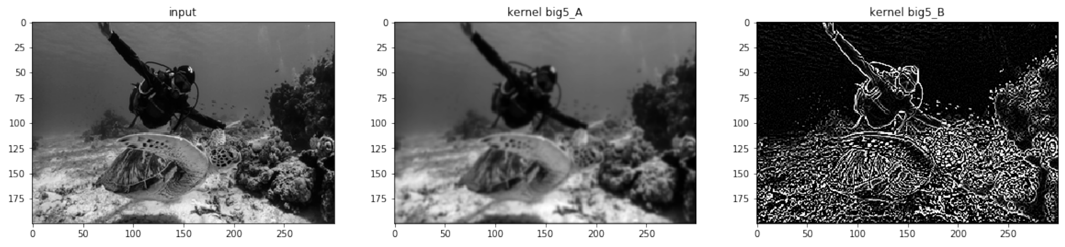


Figure 2.6: Input Image (left), kernel A5 (center), kernel B5 (right)

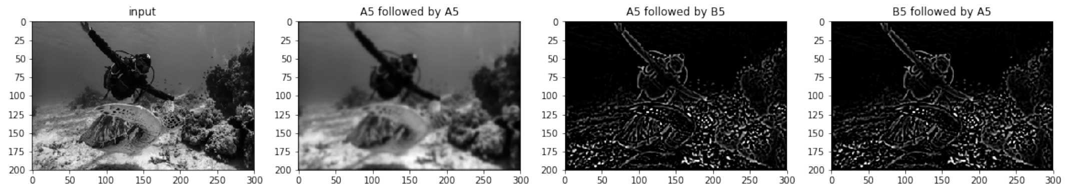


Figure 2.7: (from left) Input image, A5 followed by A5, A5 followed by B5, B5 followed by A5

7x7 kernel

$$Kernel\ A7 = \frac{1}{993}\begin{pmatrix} 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 3 & 13 & 22 & 13 & 3 & 0 \\ 1 & 13 & 49 & 97 & 59 & 13 & 1 \\ 2 & 22 & 97 & 159 & 97 & 22 & 2 \\ 1 & 13 & 59 & 97 & 59 & 13 & 1 \\ 0 & 3 & 13 & 22 & 13 & 3 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \end{pmatrix} Kernel\ B7 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 3 & 3 & 3 & 1 & 0 \\ 1 & 3 & 0 & -7 & 0 & 3 & 1 \\ 1 & 3 & -7 & -24 & -7 & 3 & 1 \\ 1 & 3 & 0 & -7 & 0 & 3 & 1 \\ 0 & 1 & 3 & 3 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

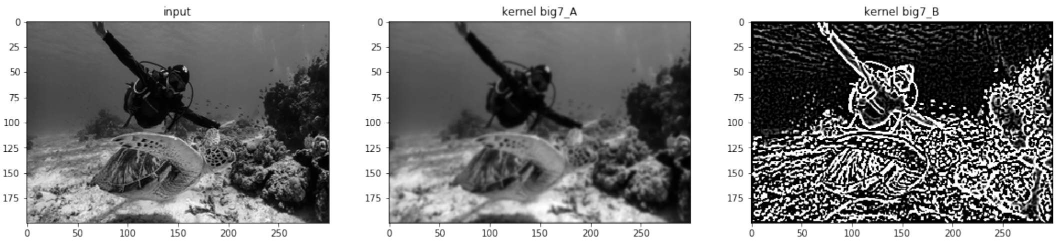


Figure 2.8: Input Image (left), kernel A7 (center), kernel B7 (right)

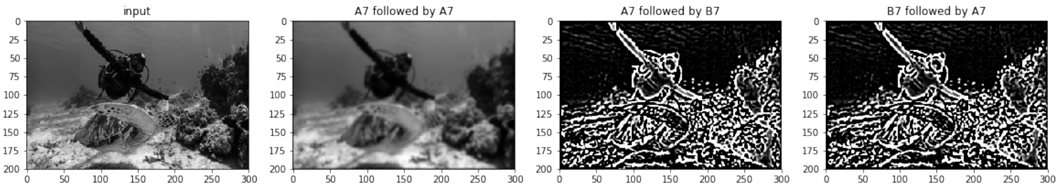


Figure 2.9: (from left) Input image, A7 followed by A7, A followed by B7, B7 followed by A7

In the result, we can observe that larger kernel make more blurry image and the edge is more highlighted. However, if the kernel is overly larger (7x7) the edge information is exaggerated. So selecting proper kernel size is important to process the image properly.

Chapter 3

Texture

Local binary pattern (LBP) is a feature used for texture representation and classification. The LBP operator describes the surroundings of a pixel by generating a bit-code from the binary derivatives of a pixel. The operator is usually applied to greyscale images and the derivative of the intensities. Note how a pixel with a $value > C$ is assigned '1' and a pixel with a $value \leq C$ is assigned '0'.

Example 3.1 (LBP)
Write a function that divides an input (greyscale) image into equally sized non-overlapping windows, and returns the feature descriptor for each window as distribution of LBP codes.

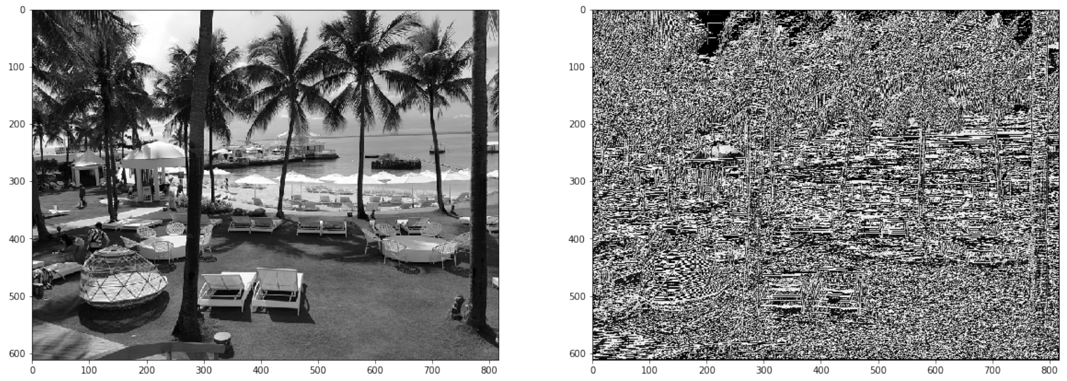


Figure 3.1: Input Image (left) and lbp (right)

In the result image, the shape of object in image barely observable. However, we can see that plane appears in black surface because there is no intensity change which is encoded in '0'. And we can also observe the horizontal and vertical edge in the LBP coded image because these edge tends to contain similar codes.

Example 3.2 (Global Descriptor)
Compute the histogram over the window, as the frequency of each "number" occurring. Normalize the histogram.

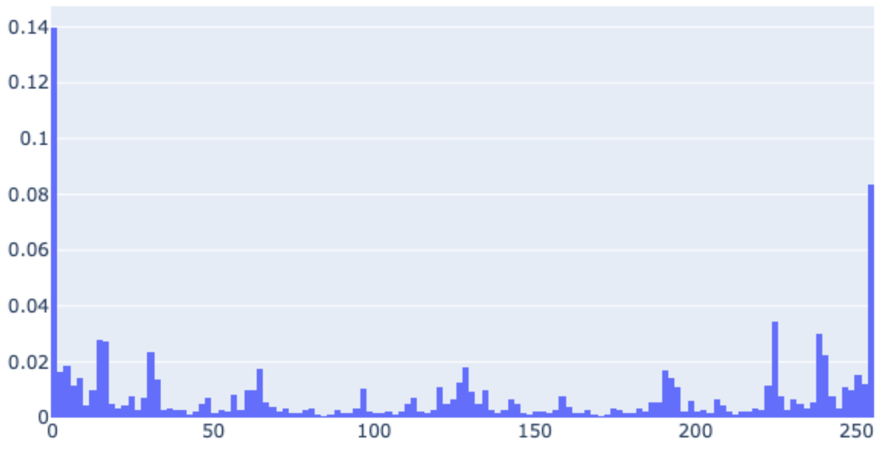
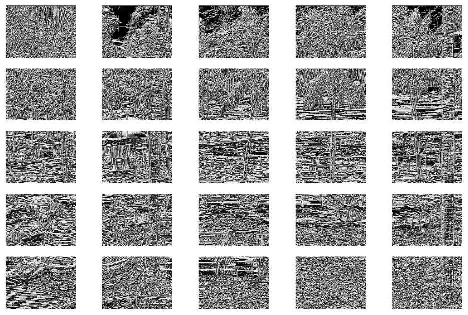


Figure 3.2: Normalized histogram of lbp image

This histogram visualize the distribution of LBP codes in image. the code 0 means no changes in neighbor pixel which stands for plain. However, once the noise is added to image, the plain cannot be encoded in zero because there is global ripples in intensity on the plain surface. So LBP coding method is too sensitive to noise.

Example 3.3 (Local Descriptor)
The histogram is now a feature descriptor representing the window at hand. Come up with a descriptor to represent the whole image as consisting of multiple windows. Hint: Think of combining several local descriptions into a global description.


(a) Image patches divide into 5x5

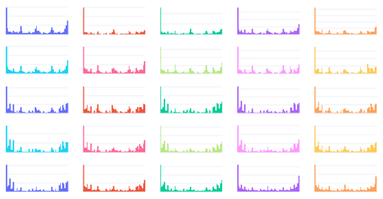
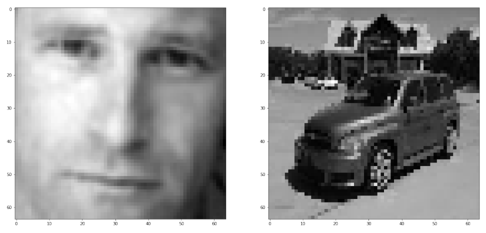

(b) Local histogram

Figure 3.3: Local histograms

This histogram stands local distribution of LBP coding. This expression contains local information comparing to global descriptor. For example, the sky region which usually composed in uniform color has more '0' codes than other parts. Therefore images which has locally biased features like face of air plane images can be expressed better with this methods.

Example 3.4 (Face Classifier)
Using the global descriptor you have created, implement a classification process to classify the images in the provided dataset into two categories: face images and non-face images. Comment the results.
Hint: You can use simple histogram similarities. Is the global descriptor able/unable to represent whole images of different types, e.g. faces vs. cars? Identify problems (if any). Suggest solutions if possible.

To make face classifier with LBP coding method, we can use histogram similarity. First, extract the local LBP histograms and average the whole histogram to make 1D vector to simplify the problem. Then set the 1D thresholds which has same shape in histogram vector to classify the face and car image.


(a) sample image of face and car dataset

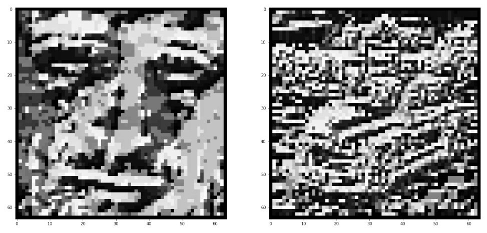

(b) LBP encoded image

Figure 3.4: LBP encoded image

And I set the threshold according to procedure below.

```
Result: Write here the result
initialization;
$face\_avg = Avg(\sum(histogram\ ofall\ face\ data)$;
$car\_avg = Avg(\sum(histogram\ of\ all\ cars)$;
threshold = (face_avg+car_avg) / 2;
if (histogram - threshold) > 0 then
    face;
else
    car;
end
finetune the thresholds by hand.;
```

Algorithm 1: How to write algorithms

This classifier successfully classify the 14 faces among the 20 faces. And classify 18 car images as non-face images among the 20 car images. Precision is 0.875 and recall is 0.7. This is significant result though simple process and applying many simplification. For better result, we can apply PCA component analysis rather than hand-tuned threshold setting.

Example 3.5 (Decrease window size)
Decrease the window size and perform classification again. Comment the results.

Smaller windows size, more local information is presented. So the histogram vector can contain more locally informations and the classification results improves than before.

Example 3.6 (Increase window size)
Increase the window size and perform classification again. Comment the results.

Larger windows size means it contains more general informations. So this representations of histogram may better to classify the data which has different amount of details like ocean and people.

Example 3.7 (Video Analysis)
Discuss how LBP can be used/modified for dynamic texture analysis, e.g. in a video.

dynamic texture contains repetition of similar textures so this information appears in periodic vector in LBP coded image. So after extracting LBP coded image, we can find similar objects in each frame to find similar vectors of segment the similar texture by analyzing LBP coded images.

Chapter 4

Colour Histogram

A colour histogram is a representation of the distribution of colours in an image. Given a discrete predefined colour space (e.g. red, green, blue), the colour histogram is obtained by counting the number of times each colour occurs in the image array. Histogram Intersection is a technique that can be used to match a pair of histograms.

Given a pair of histograms, e.g. of an input image I and a model M, each containing n bins, the intersection of the histograms is defined to be $\sum_{i,j} min(I, M)$ where j ranges over each colour in the histograms. The result of the intersection of a model histogram with an image histogram is the number of pixels from the model that have corresponding pixels of the same colour in the image.

Example 4.1 (3D colour histograms)
Write a histogram function that returns the 3D colour histogram of an input image. Visualize the 3D histogram and save the figure(s). For a given video sequence, use the above function to construct and visualise the histograms of each frame.

To analyse 3d histogram images, I set the bin as 16 and represents in bubble scatter plot. As my image is mostly blue and brown, so the 3d color histogram also biased in blue dots.

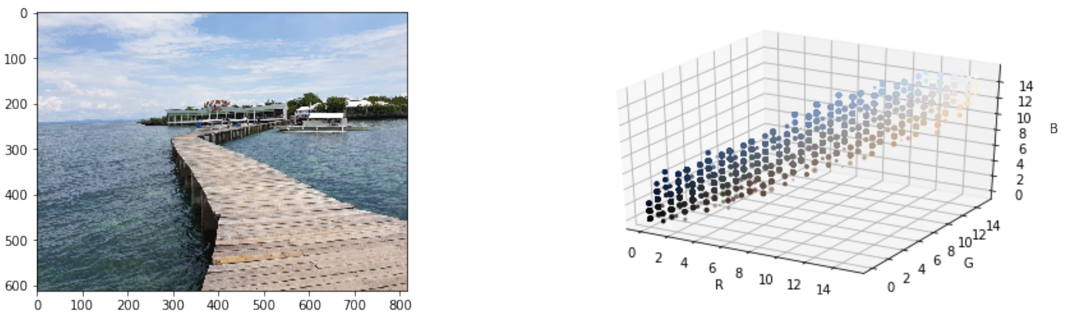


Figure 4.1: Input image I and corresponding 3d colour histogram

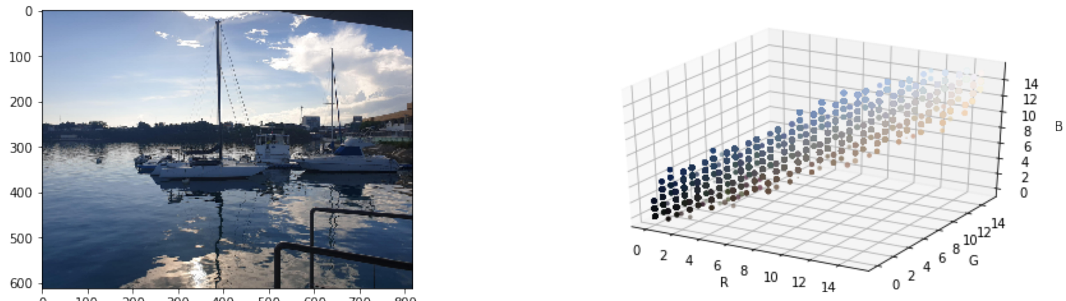


Figure 4.2: Input Model image M and corresponding 3d colour histogram

And I calculate histogram intersection.

```
def hist_intersection(img1,img2):

    hist1 = Histogram3d(img1)
    hist2 = Histogram3d(img2)

    histIntersection = np.minimum(hist1,hist2)
    son = np.sum(histIntersection)
    mom = min(np.sum(hist1),np.sum(hist2))
    result = son / mom

    return result
```

For these two image, the histogram intersection value is 0.74.

Example 4.2 (intersection of histograms)
Write a function that returns the intersection of a pair of histograms. For a given video sequence, use the histogram intersection function to calculate the intersection between the consecutive frames, e.g., between I_t and I_{t+1}, between I_{t+1} and I_{t+2} etc. Visualize and present the intersection values.
Find a way to normalize the intersection.
Does that cause any changes in the results?

Intersection of histogram can be easily obtained by minimum operation between two image. To normalize the intersection, we can divide the intersections by total amount of pixel. For my video sequence which has 337 frames in total has dramatic scene change in frame 210.

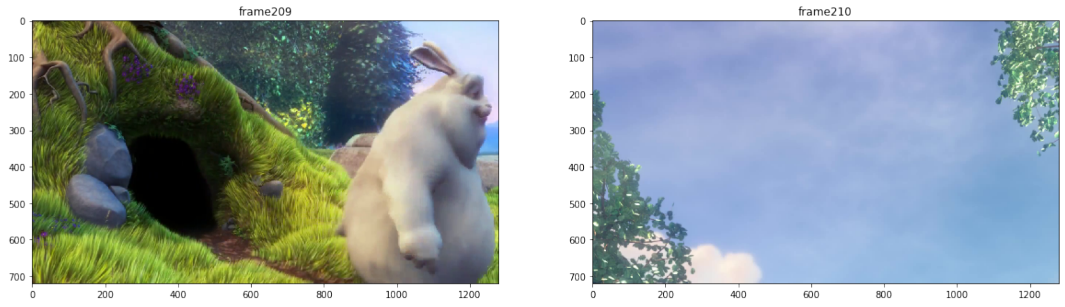


Figure 4.3: scene change between frame 209 and 210.

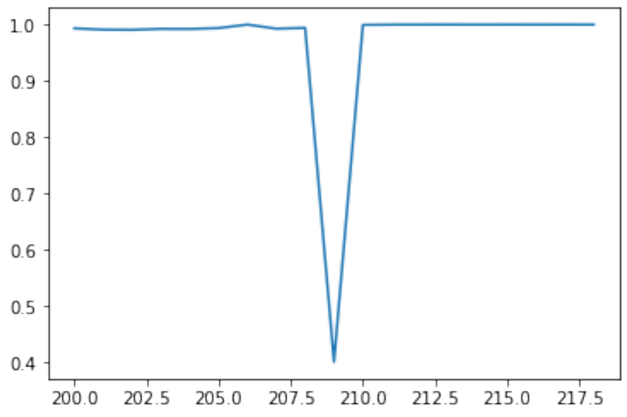


Figure 4.4: Histogram intersection around frame 210

We can observe significant value drop in frame 210. So we verify that meaningful change of histogram intersection value occurs in dynamic scene change.

Example 4.3 (Discussion)
What does the intersection value represent for a given input video?
Can you use it to make a decision about where the scene in the video changes?
How robust is the histogram intersection technique to changes?
Where does it fail?
What would be other application areas where histogram calculation and histogram intersection can be used?

Intersection value represents the similarity of colour information that contains in each frame. Therefore, if the intersection value is high the two frame may consecutive sequence in same scene. On the other hands, if the intersection value is low the tow frame is different scene. So we cans use this analysis method to segment the scene. We can divide the scene by selecting zero intersection value as scene-changing frame.

However, though this method is robust to change in dynamic scene changes, it cannot segment the minimal scene changes such as camera switch in same scene. And this methods is too sensitive in lighting condition in same scene. rapid changes in same scene such as car incoming with its head lights on may be segmented as diferrent scene with this method.

The most useful histogram intersection method can be used in video compression application. To compress the video, pixels with change from formal frame is saved. So histogram intersection analysis can be helpful to select the keyframes of the video.