NVLab Summer School 2023 HW1

112061517 王浩

Abstract

在本次作業裡，Part1, Part2 我們先使用手寫影像處理的基礎function 搭配Hybrid image，得到Color/Luminance Manipulation, Image Filter 初步的影像實作結果，並且在Part3, Part4 嘗試呼叫Albumentations, Torchvision libraries，研究其相關問題。

1 Part 1. Color/Luminance Manipulation

1.1 Code implementation

1.1.1 Gamma Mapping

```
def gamma_correction(img, gamma):
    img_float = img.astype(np.float32)
    img_float = img_float / 255.0
    corrected_img = np.power(img_float, 1/gamma)
    corrected_img = corrected_img * 255.0
    corrected_img = corrected_img.astype(np.uint8)
    return corrected_img
```

1.1.2 Histogram Equalization

先利用CDF 計算出pixel value 的顏色分佈累積狀態，再重新分配到全部的顏色區間裡。

$$h(v) = round(\frac{cdf(v) - cdf_{min}}{cdf_{max} - cdf_{min}} * (L-1))$$

```
def histogram_equalization(img):
    img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    # extract V channel (luminance)
    img_hsv[:, :, 2] = cv2.equalizeHist(img_hsv[:, :, 2])
    img_equalized = cv2.cvtColor(img_hsv, cv2.COLOR_HSV2BGR)
    return img_equalized
```

1.2 Discussion

1.2.1 Part1 a. Compare the corrected results with gamma = 0.55, 1 and 2.2 .

以下三張圖片分別代表設定Gamma 為0.55 1, 1 2 與2.2 3 的不同情況。

1.2.2 Part1 b. Show the original image and the processed output as well as their histograms.

以下三張圖片分別代表原圖4、原圖的v channel histogram5, 經過histogram equalization 後的圖6 與經過histogram equalization 後的v channel histogram 7 。

Figure 1: This picture's gamma is 0.55.

Figure 2: This picture's gamma is 1.

Figure 3: This picture's gamma is 2.2.

Figure 4: Original picture

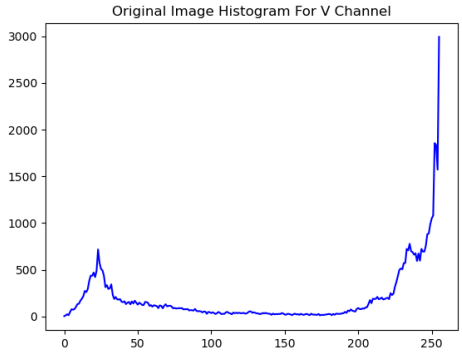

Figure 5: Original picture's v channel histogram

Figure 6: Picture after histogram equalization

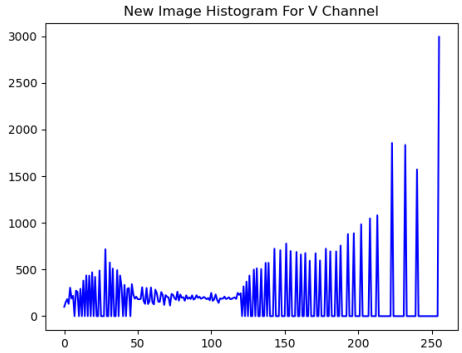

Figure 7: Histogram after histogram equalization

2 Part 2. Image Filter

2.1 Code implementation

2.1.1 Convolution Implementation

```
def my_conv(img, kernel):
    '''
    function: convolution

    img: (H*W*C) numpy array
    kernel: (K*K) numpy array
    output: (H*W*C) numpy array
    '''
    pad_size = kernel.shape[0]//2
    (height, width, channel) = img.shape
    # Zero-padding
    pad_img = cv2.copyMakeBorder(
                    img, pad_size, pad_size, pad_size, pad_size
                    , cv2.BORDER_REPLICATE
                    )
    # Convolution
    output_img = np.zeros(img.shape)
    for i in range(height):
        for j in range(width):
            for c in range(channel):
                output_img[i, j, c] =
                    np.sum(
                    pad_img[i:i+kernel.shape[0], j:j+kernel.shape[1], c]
                    * kernel
                    )

    return output_img
```

2.1.2 Hybrid Image

```
def my_GaussianBlur(size, sigma):
    '''
    function: gaussian blur kernel generating

    size: kernel size, should be an odd number
    sigma: standard deviation of gaussina distribution
    g: gaussian kernel
    '''
    x, y = np.mgrid[-size//2 + 1:size//2 + 1, -size//2 + 1:size//2 + 1]
    g = np.exp(-((x**2 + y**2)/(2.0*sigma**2)))
    g = g/g.sum()
    return g

def main():
    LPF_kernel = my_GaussianBlur(20, 5)
    lpf_img = my_conv(l_img, LPF_kernel)
    hpf_img = h_img - my_conv(h_img, LPF_kernel)
    hybrid_img = lpf_img + hpf_img
return hybrid_img
```

Figure 8: Box blur result

Figure 9: Sharpen result

2.2 Discussion

2.2.1 Part2 a. Use $my_conv()$ to process the input image with different kernels (t least blur and sharpen)

兩張圖片分別使用$my_conv()$，搭配

$$sharpen\ kernel: \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, box\ blur\ kernel: \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

最後得到box blur picture 8 與sharpen picture 9。

2.2.2 Part2 b. Try to use different parameters for $my_GaussianBlur()$, what do you observe?

我使用了三種的Gaussian blur kernel size、sigma 組合，來觀察Hybrid image 的最後效果，分別是$\{size: 10, sigma: 2\}$10、$\{size: 20, sigma: 5\}$11 與$\{size: 50, sigma: 10\}$ 12。

這三者的kernel size 與sigma 數值逐次遞增。可以觀察出low pass picture 的blur 狀態是隨著數值上升，變得更加的「模糊」，所以狗的型態在距離近時，變得更難直接看出。high pass picture 則是愈加明顯，輪廓隨著size and sigma 增加，變得更加强烈清晰，所以貓的型態在靠近時，相當明顯。

2.2.3 Part2 b. Try different pairs of pictures in the folder /resources or gather your own picture pairs , show your results of hybrid results and what you discover during the process.

共使用了五種不同情況的組合，分別是dog/cat 13, Marilyn/Einstein 14, bicycle/motorcycle 15, fish/submarine 16, bird/plane 17，在過程中，可以在近的距離內，看到比較清楚的High pass result，而隨著距離拉遠，我們逐漸會發現Low pass result 會愈加明顯。

而且發現某些圖片的Gaussian blur sigma 和kernel size 依然需要調整，如第二組的Marilyn and Einstein 目前的參數選擇為kernel size:20, Gaussian blur sigma:5, 但Einstein (high pass filter result)的輪廓就相較而言高出許多，並不如其他組合結果合適；相對而言，Bicycle and motorcycle hybrid

Figure 10: Hybrid image with Gaussian size:10 and sigma:2

Figure 11: Hybrid image with Gaussian size:20 and sigma:5

Figure 12: Hybrid image with Gaussian size:50 and sigma:10

Figure 13: Dog and cat hybrid image

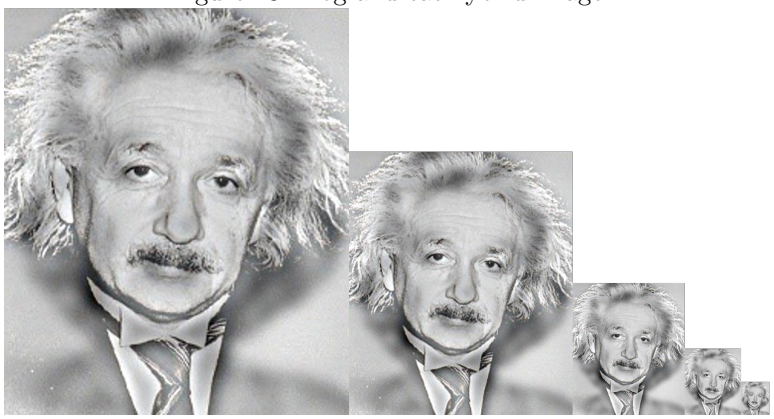

Figure 14: Marilyn and Einstein hybrid image

image 的low pass filter result 的腳踏車，就是一個相反的例子，他的low pass 結果就變成比較明顯，所以參數的選擇還是需要根據每一組圖片的狀態去動態調整，才整找出最好的範例。

2.2.4 Part2 b. Describe what kind of information we acquire from two filters in Hybrid Image.

Hybrid image 是從兩種不同的spatial frequencies 中的資訊組合，而我們就可以從觀看的距離差異，去接收到來源自兩張不同圖片的不同資訊。而分為Low-pass filter 與High-pass filter。

- Low-pass filter 就是讓一張圖片中的low frequency component 通過，block high frequency 的component。效果是smooth 圖片中的細節、blur 尖銳的邊緣、保持整體圖片的大致結構。
- High-pass filter 的效果相反，讓圖片中的high-frequency component 通過，block low-frequency component。能凸顯出圖片中的細節、邊緣，可以看到銳利的高頻特徵，而忽略整體的架構低頻資訊。

在距離遠的時候，我們比較容易觀察出low-pass filter 過濾出來的資訊，而在距離近的時，就可以看到高頻過濾的結果。

2.2.5 Part2. What applications do you think this technology can be used for?

可以拿來運用Hybrid image Encryption and Steganography. 這是一種視覺上的加密方式，將重要資訊藏在high-pass filter，就可以不引起其他人視覺上的注意，安全地將視覺資訊傳出。

- Hybrid image encryption 表示可以先將一張圖片在hybrid 前，就先對其做加密，等到decryption process 在將重要或敏感的資訊提取出來。
- Steganography 表示我們再傳重要資訊時，可以利用其他的medium「覆蓋」在重要的資訊上面，而我們就可以用另外一張圖片來當作是這個medium，把重要的資訊藏起來。

因此如果利用Hybrid image 的技術，就應用到視覺方面的資訊安全。

Figure 15: Bicycle and motorcycle hybrid image

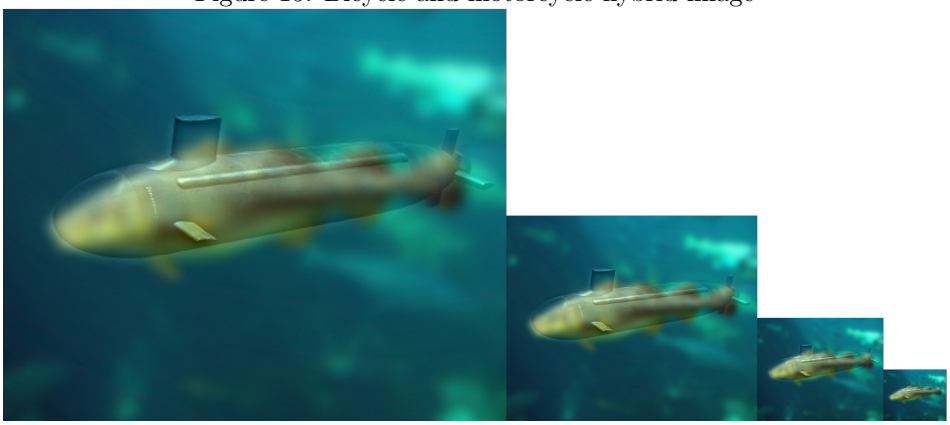

Figure 16: Fish and submarine hybrid image

Figure 17: Bird and plane hybrid image

3 Part3 & Part4 Albumentations and Torchvision

3.1 Code implementation

3.1.1 Albumentations

```
import albumentations as A
transform = A.Compose([
    A.Blur(),
    A.Rotate(limit=30),
    A.Flip(),
    A.Resize(width=256, height=256),
])
albu_aug_img = transform(image=img)['image']
```

3.1.2 Torchvision

```
import torchvision.transforms as transforms
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.GaussianBlur(kernel_size=5),
    transforms.RandomRotation(degrees=30),
    transforms.RandomHorizontalFlip(),
    transforms.Resize((256, 256)),
])

RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
transformed_image = transform(RGB_img)
```

3.2 Discussion

3.2.1 Part 3. Use Albumentation to do the image process(e.g. blur, rotation, flip, resize, etc.)(at least 3 kinds)

我們利用Albumentation library 中的blur, rotation, flip, resize 四種augmentation 的方式，實作出18

3.2.2 Part 4. Use Torchvision to do the image process(e.g. blur, rotation, flip, resize, etc.)(at least 3 kinds)

也利用Torchvision library 中的blur, rotation, flip, resize 四種augmentation 的方式，實作出19

3.2.3 Discuss the difference of doing image processing using convolution techniques in Part2.a, using Albumentation and Torchvision.

1. Part 2.a 的self-written code 的優點是我們能夠完全控制、撰寫執行convolution 的每一個細節，包含kernel 的size, padding, stride 等等內容，有最高的flexibility。缺點是time-consuming，我們也容易在撰寫的過程中出現錯誤，也需要花最多的精力去研究、掌握所有的細節。

2. Albumentations 是專門做image augmentation 的library，優點是high-level interface 跟pre-defined 的常見影像處理的transformations function，而且他能夠整合、串接各種augmentation 的任務，可以順利的執行複雜的augmentation task。

3. Torchvision 除了Albumentation 的優點外，Torchvision 也是Pytorch framework 下的module，所以很好的整合了Pytorch's data loading utilities，能夠在我們要訓練模型前，做好data 的process。

Figure 18: Augmented picture by Albumentation

Figure 19: Augmented picture by Torchvision