影像轉水墨畫

影像處理期末報告

409416541

二〇二二 年 六 月 二十 日 星期一

內容目錄

製作動機

水墨畫是華人文化中可以堪稱是華人畫作的代表，甚至在我國的國畫中，絕大部分的畫作皆是以水墨畫來呈現。可見水墨畫在我們心中據有多麼重要的地位。

而我出生於傳統華人家族，從小便接觸到了許多我們華人世界的各類藝術品，其中我對於水墨畫更是愛不釋手。古人運用一隻毛筆便可將山水風景盡收紙中，由此便可隨身攜帶各地美景，令我十分驚嘆，也因此我對於水墨畫可說是非常的熱衷，而其中只有黑白兩色的動物水墨畫更是我的偏好。單純的色調，卻又能夠簡潔而又有力地描繪出了事務的美好，能夠將我喜歡的動物畫成如此美麗的景象，讓我十分癡迷，當我發現它時，時常會呆立在原地欣賞它到忘我。

而在修習影像處理的課程時我發現到，雖然我沒有能力像張大千老師一樣將喜好的事物以水墨畫的形式表達出來，但是我可以運用我在影像處理的課程中所學到的知識，將我拍下來的照片轉換為我最為偏好的只有黑白兩色的水墨畫，由此我便可擁有屬於自己的水墨畫創作，這也是此次促使我做這項實作的動機。

功能介紹

使用者將喜好的影像給予程式讀取，該程式將生成只有黑白兩色的水墨畫出來，讓喜好水墨畫的人可以將喜歡的影像轉換為水墨畫並儲存下來，隨身攜帶專屬於自己的稀世珍寶。

工作原理

- cv2.COLOR_BGR2GRAY 將讀取進來的彩色影像轉為灰階影像。
- cv2.GaussianBlur 使用高斯濾波去除背景雜訊並生成新影像，因為高斯濾波會將影像進行平滑化，所以若是有雜訊的話會跟周圍的。
- cv2.Canny 取得影像的邊緣資訊。
- change(img) 因為邊緣是黑色的話我比較習慣，但是用 canny 的話邊緣會是白色的，所以自己寫一個函式將邊緣改成黑色。
- cv2.adaptiveThreshold 將影像透過高斯法的適應性閥值化來將灰階影像的灰階值變為只有 0 和 255。
- reduce_noise(img, edges, radius) 為了透過邊緣資訊來將需要的像素點保留，所以我寫了一個函式來進行判斷，在各像素點的半徑(radius)周圍的邊緣資訊(edges)是否已值，有值則保留，沒值則將灰階值設為 255(白色)，以此來確認哪些像素點的灰階值應該被保留下來，其餘的灰階值為 255，以此來生成成品。

程式流程

1. 讀取想要轉換為水墨畫的影像，並存在 img 內
2. 將該 img 轉換為灰階影像，並存在 gray_img 內
3. 將 gray_img 透過高斯濾波處理背景雜訊，並存在 gaussian_img 內。
4. 透過 cv2.Canny 讀取 gaussian_img 的邊緣資訊，而後再透過函式 change(img) 來將邊緣資訊的顏色做倒數，並存在 edges_img 內。
5. 將 gray_img 透過高斯法的適應性閥值化來將影像二值化，並存在 adaptive_theshold_gaussian 內。
6. 透過 reduce_noise(img, edges, radius)來判斷若灰階影像中各相素點半徑(radius)內邊緣影像(edges)有值則將該像素點的灰階值保留，若否則將該像素點的灰階值設為 255(白色)，並將結果存在 reduce_noise_img 內，而這個也就是最終成品。而在這裡,img 帶 adaptive_theshold_gaussian,edges 帶 edges_img，radius 帶 6，來進行判斷。
7. 最後透過 cv2.imwrite 將轉換好的水墨畫保存起來。

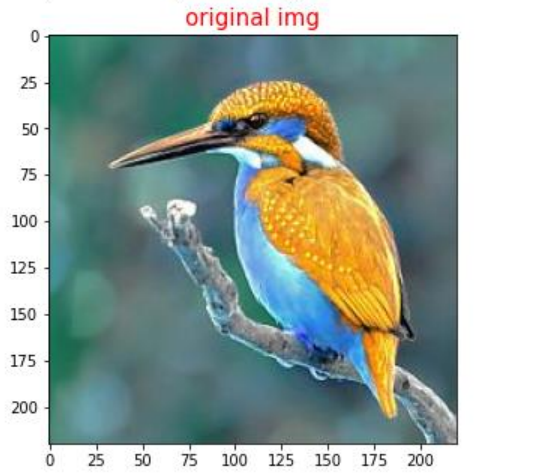
程式碼

讀取影像

讀取影像

```
[2]  1 import numpy as np
     2 import cv2
     3 from google.colab.patches import cv2_imshow
     4 from matplotlib import pyplot as plt
     5 from google.colab import drive
     6 from google.colab.patches import cv2_imshow
     7
     8 drive.mount('/content/drive')
     9 img = cv2.imread("/content/drive/My Drive/Colab Notebooks/image_processing/week14.jpg", -1)
    10 print(img.shape)
    11
    12 plt.figure(figsize = (5, 5))
    13
    14 plt.title('original img', fontsize = 15, color = 'r')
    15 plt.imshow(img, cmap = 'gray')
```

```
(220, 220, 3)
<matplotlib.image.AxesImage at 0x7ff11e1cc5d0>
```

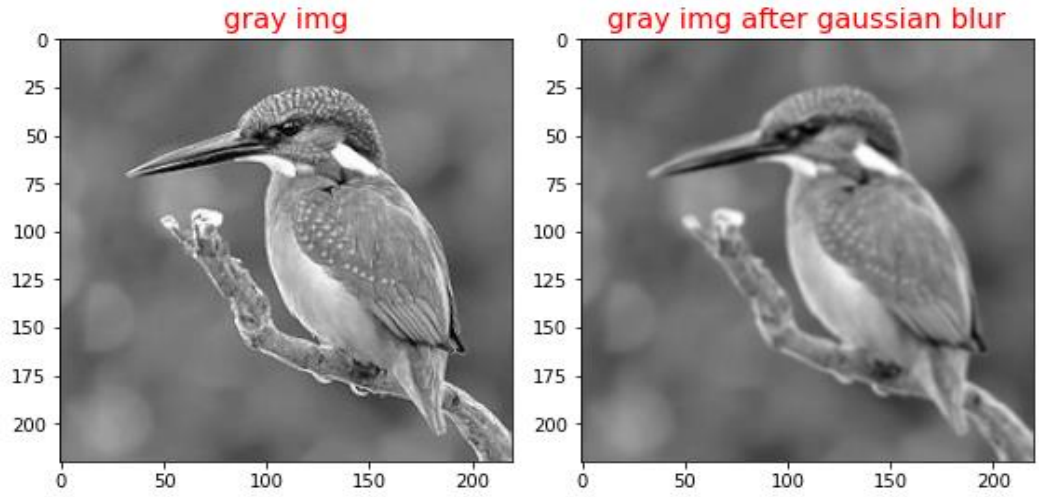


- 讀取想要轉換成水墨畫的影像。

生成灰階影像

生成灰階影像

```
[3]  1 plt.figure(figsize = (8, 8))
     2
     3 gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #將讀取的影像轉換為灰階影像
     4 plt.subplot(1, 2, 1)
     5 plt.title("gray img", fontsize = 15, color = 'r')
     6 plt.imshow(gray_img, cmap = 'gray')
     7
     8 gaussian_img = cv2.GaussianBlur(gray_img, (5, 5), 0)    #將讀取的影像透過高斯濾波將影像平滑化，以此來減少背景躁點
     9 plt.subplot(1, 2, 2)
    10 plt.title("gray img after gaussian blur", fontsize = 15, color = 'r')
    11 plt.imshow(gaussian_img, cmap = 'gray')
    12
    13 plt.tight_layout()
```

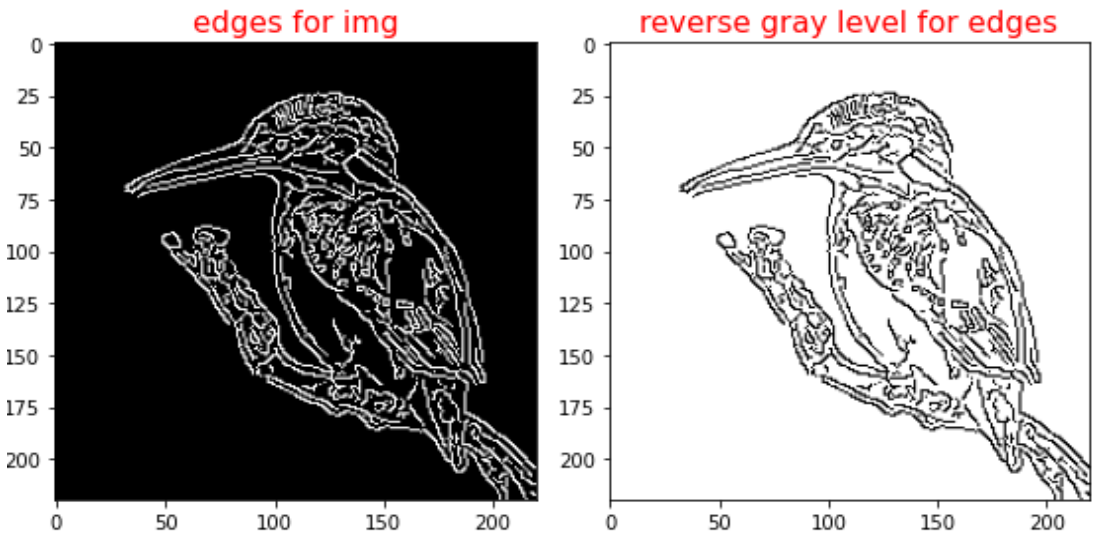


- 生成灰階影像(左邊的)，並額外再儲存一個有通過高斯濾波進行背景降噪的灰階影像(右邊的)。

生成影像邊緣

生成影像邊緣

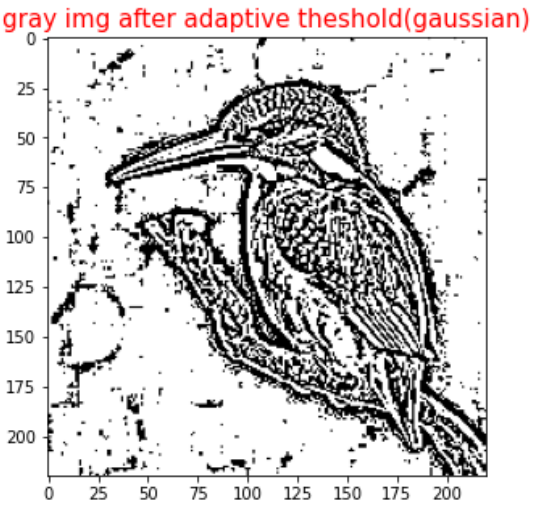
```
[4]  1 def  change(img):    #對影像灰階值做倒數
     2         nr,  nc  =  img.shape[:2]
     3         new_img  =  np.zeros((nr,  nc),  np.uint8)    #建立一新影像且灰階值為0
     4
     5         for  x  in  range(nr):
     6                 for  y  in  range(nc):
     7                         if  img[x][y]  ==  0:    #若傳進來的影像目前位置灰階值為0，則新影像同樣位置灰階值設為1
     8                                 new_img[x][y]  =  255
     9
    10         return  new_img    #回傳該影像
    11
    12 plt.figure(figsize  =  (8,  8))
    13
    14 edges_img  =  cv2.Canny(gaussian_img,  5,  90)    #影像邊緣偵測
    15 plt.subplot(1,  2,  1)
    16 plt.title("edges  for  img",  fontsize  =  15,  color  =  'r')
    17 plt.imshow(edges_img,  cmap  =  'gray')
    18
    19 edges_img  =  change(edges_img)    #因為Canny所生成之邊緣影像的邊緣為白色背景為黑色，故將其顛倒
    20 plt.subplot(1,  2,  2)
    21 plt.title("reverse  gray  level  for  edges",  fontsize  =  15,  color  =  'r')
    22 plt.imshow(edges_img,  cmap  =  'gray')
    23
    24 plt.tight_layout()
```



- 透過 cv2.Canny 讀取有通過高斯濾波的灰階影像的邊緣資訊(左邊的)，而後透過函示 change(img) 來將邊緣資訊的影像值倒轉(右邊的)，原因是我個人習慣白底黑邊，我認為這樣看上去比較直觀。

將未經過高斯濾波的灰階影像二值化

```
adaptive_theshold_gaussian = cv2.adaptiveThreshold(gray_img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)    #將影像透過高斯法的適應性閥值化來將灰階影像變為只有黑白兩色

plt.figure(figsize = (5, 5))

plt.title('gray img after adaptive theshold(gaussian)', fontsize = 15, color = 'r')
plt.imshow(adaptive_theshold_gaussian, cmap = 'gray')
```



- 透過高斯法的適應性閥值化來將沒有經過高斯濾波的灰階影像二值化。

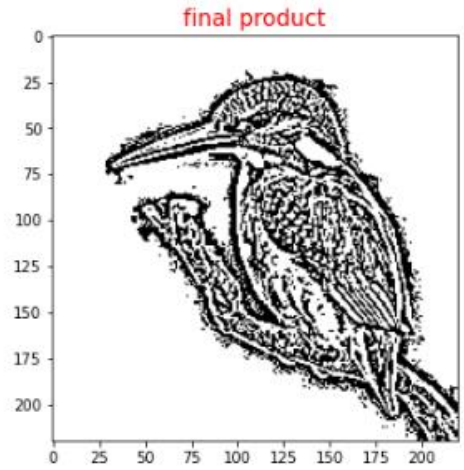
生成成品

透過邊緣資訊來將二值化後的影像中不必要的地方刪除

用經過高斯濾波的邊緣資訊以及未經過高斯濾波的二值化影像生成影像

```
def reduce_noise(img, edges, radius):    #若灰階影像中各相素點半徑(radius)內邊緣影像有值則將其保留，若否則將該相素點刪除
    nr, nc = img.shape[:2]
    new_img = img.copy()    #複製一新影像
    has_edges = 0    #檢測半徑內是否有邊緣資訊
    edges_x = edges_y = 0    #紀錄半徑內的X,Y軸的值

    for x in range(nr):
        for y in range(nc):
            if img[x][y] == 0:
                for round_x in range(2 * radius + 1):    #檢測目前位置加上或減去半徑後是否會超過影像範圍
                    for round_y in range(2 * radius + 1):
                        if (x - radius + round_x) < 0:
                            edges_x = 0

                        elif (x - radius + round_x) >= nr:
                            edges_x = nr - 1

                        else:
                            edges_x = x - radius + round_x

                        if (y - radius + round_y) < 0:
                            edges_y = 0

                        elif (y - radius + round_y) >= nc:
                            edges_y = nc - 1

                        else:
                            edges_y = y - radius + round_y

                        if edges[edges_x][edges_y] == 0:    #若在半徑內有邊緣資訊，則記錄有邊緣
                            has_edges = 1

                if has_edges == 0:    #若半徑內無邊緣資訊則將新影像的該點的灰階值設為255
                    new_img[x][y] = 255

                has_edges = 0

    return new_img

reduce_noise_img = reduce_noise(adaptive_theshold_gaussian, edges_img, 6)    #透過邊緣資訊來將不必要的地方刪除

print(reduce_noise_img.shape)

plt.figure(figsize = (5, 5))

plt.title('final product', fontsize = 15, color = 'r')
plt.imshow(reduce_noise_img, cmap = 'gray')

cv2.imwrite('/content/drive/My Drive/Colab Notebooks/image_processing/endterm project.bmp', img)    #將轉好的影戲存起來
```

```
(220, 220)
True
```



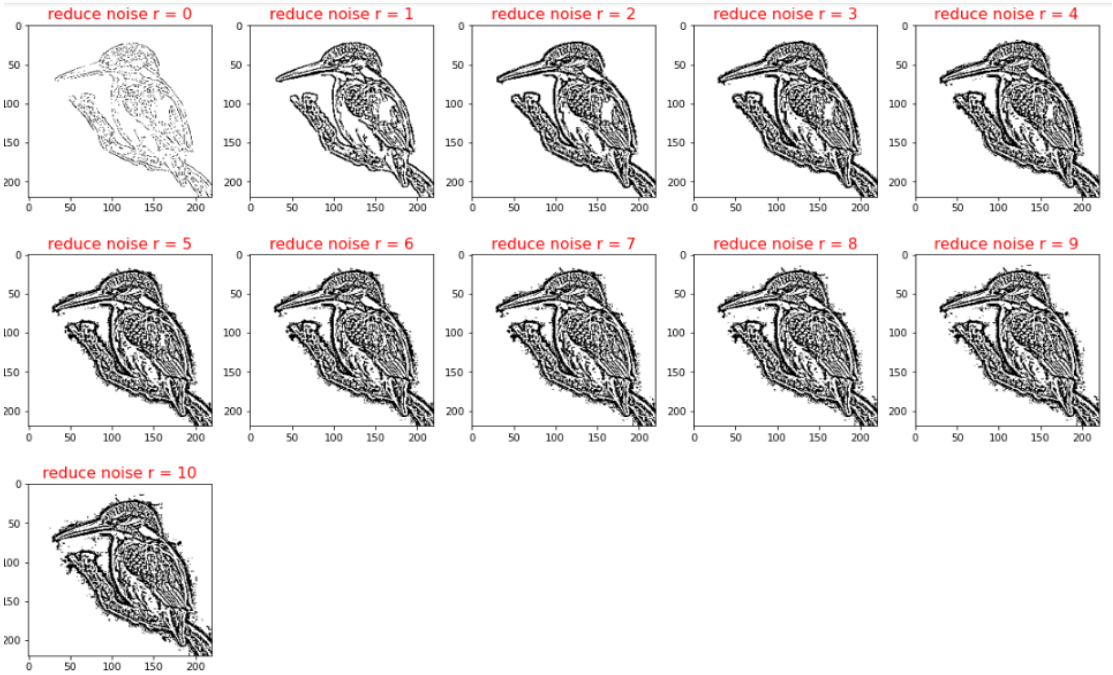
- 透過函式 reduce_noise(img, edges, radius) 用有經過高斯濾波的邊緣資訊來判斷未經過高斯濾波的二值化影像上面哪些像素點的周圍(半徑為 6)有無邊緣資訊，若有則保留該像素點的灰階值，若否則將該像素點的灰階設為 255(白色)

問題討論與心得

如何決定半徑(radius)要帶多少

當灰階影像的半徑(r)含有邊緣影像時，半徑0-10所生成之影像

```
[ ]  1 r0 = reduce_noise(adaptive_theshold_gaussian, edges_img, 0)
     2 r1 = reduce_noise(adaptive_theshold_gaussian, edges_img, 1)
     3 r2 = reduce_noise(adaptive_theshold_gaussian, edges_img, 2)
     4 r3 = reduce_noise(adaptive_theshold_gaussian, edges_img, 3)
     5 r4 = reduce_noise(adaptive_theshold_gaussian, edges_img, 4)
     6 r5 = reduce_noise(adaptive_theshold_gaussian, edges_img, 5)
     7 r6 = reduce_noise(adaptive_theshold_gaussian, edges_img, 6)
     8 r7 = reduce_noise(adaptive_theshold_gaussian, edges_img, 7)
     9 r8 = reduce_noise(adaptive_theshold_gaussian, edges_img, 8)
    10 r9 = reduce_noise(adaptive_theshold_gaussian, edges_img, 9)
    11 r10 = reduce_noise(adaptive_theshold_gaussian, edges_img, 10)
    12
    13
    14 imgs = [r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10]
    15 titles = ['reduce noise r = 0', 'reduce noise r = 1', 'reduce noise r = 2', 'reduce noise r = 3', \
    16               'reduce noise r = 4', 'reduce noise r = 5', 'reduce noise r = 6', 'reduce noise r = 7', \
    17               'reduce noise r = 8', 'reduce noise r = 9', 'reduce noise r = 10']
    18 plt.figure(figsize = (15, 10))
    19
    20 for i in range(11):
    21         plt.subplot(3, 5, i + 1)
    22         plt.title(titles[i], fontsize = 15, color = 'r')
    23         plt.imshow(imgs[i], cmap = 'gray')
    24
    25 plt.tight_layout()
    26 plt.show()
```

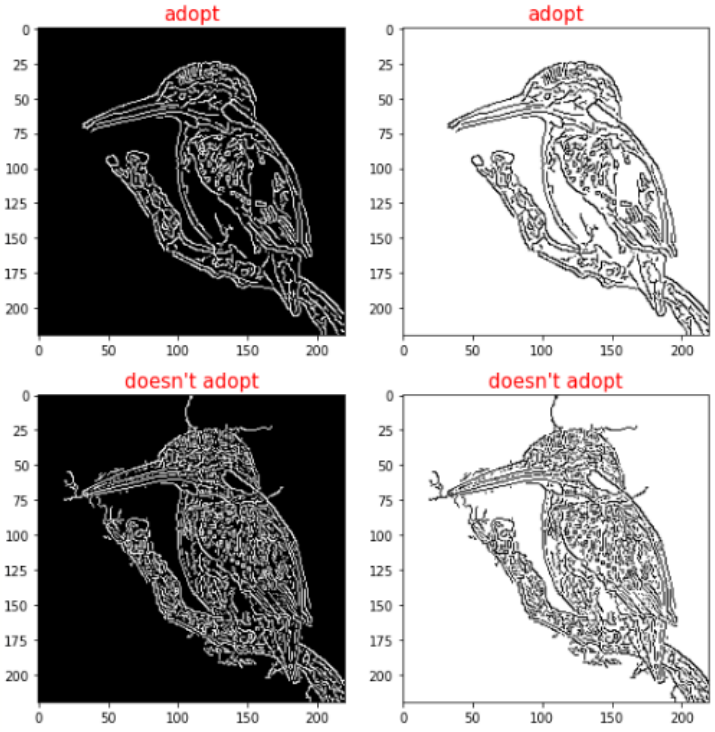


- 上圖為半徑 0~10 所呈現出成果，可以發現到，當半徑為 0 時圖像十分不清楚，當半徑小於 4 時鳥的周圍沒有任何毛邊我認為那樣不像毛筆畫，而半徑小於 6 時可以清楚的發現到羽毛中間禿一塊而當半徑大於 7 後，鳥的周圍毛邊變得過多顯得不好看，而半徑為 6 跟 7 時我認為是最好的，但因為半徑為 7 周圍毛邊略比半徑為 6 的多，所以我最後選擇半徑為 6。

若邊緣資訊使用的是未經過高斯濾波處理的灰階影像

使用未經過高斯濾波的灰階影像的邊緣資訊

```
plt.figure(figsize = (8, 8))

#採納的
edges_img = cv2.Canny(gaussian_img, 5, 90)    #影像邊緣偵測
plt.subplot(2, 2, 1)
plt.title("adopt", fontsize = 15, color = 'r')
plt.imshow(edges_img, cmap = 'gray')

edges_img = change(edges_img)
plt.subplot(2, 2, 2)
plt.title("adopt", fontsize = 15, color = 'r')
plt.imshow(edges_img, cmap = 'gray')

#未採納的
nonadopt_edges_img = cv2.Canny(gray_img, 5, 90)    #影像邊緣偵測(未經高斯濾波處理的)
plt.subplot(2, 2, 3)
plt.title("doesn't adopt", fontsize = 15, color = 'r')
plt.imshow(nonadopt_edges_img, cmap = 'gray')

nonadopt_edges_img = change(nonadopt_edges_img)
plt.subplot(2, 2, 4)
plt.title("doesn't adopt", fontsize = 15, color = 'r')
plt.imshow(nonadopt_edges_img, cmap = 'gray')
```

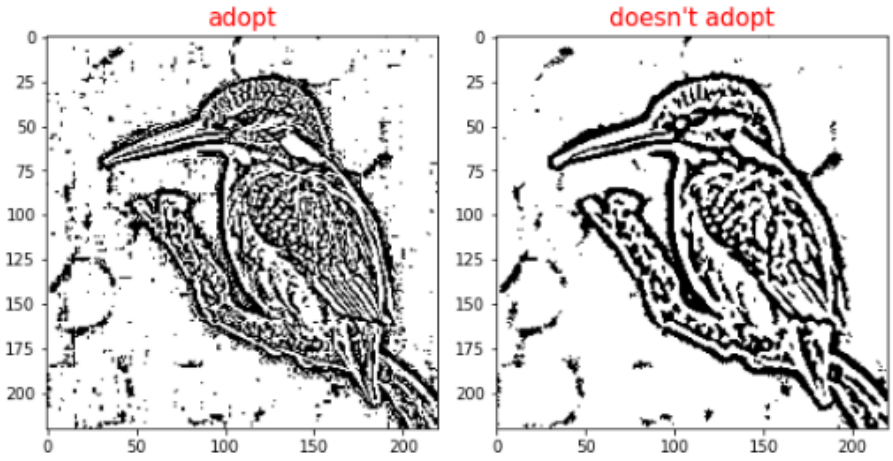


- 從下圖可以看出，若邊緣資訊使用的是未經過高斯濾波的影像來取得的話，其背景中的一些雜訊也會被判斷為是邊緣，跟我所採用的(上圖)產生明顯對比，在鳥的身體周圍多出了許多條邊緣，也因此我最終將影像經過高斯濾波後再取得他的邊緣資訊。

若將經過高斯濾波後的灰階影像二值化

將經過高斯濾波後的灰階影像二值化

```
[ ]  1 plt.figure(figsize = (8, 8))
     2
     3 plt.subplot(1, 2, 1)
     4 plt.title("adopt", fontsize = 15, color = 'r')
     5 plt.imshow(adaptive_theshold_gaussian, cmap = 'gray')
     6
     7 nonadopt_adaptive_theshold_gaussian = cv2.adaptiveThreshold(gaussian_img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
     8
     9 plt.subplot(1, 2, 2)
    10 plt.title("doesn't adopt", fontsize = 15, color = 'r')
    11 plt.imshow(nonadopt_adaptive_theshold_gaussian, cmap = 'gray')
    12
    13 plt.tight_layout()
```

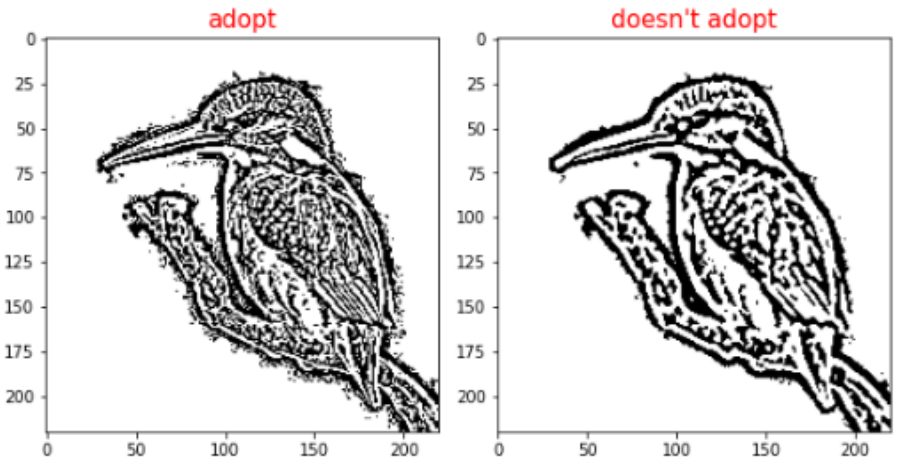


- 從右圖中可以看到，若使用經過高斯濾波處理的二值化影像，鳥的精細度非常的低，這也是為甚麼我在將經過高斯濾波處理的灰階影像額外儲存(gray_img)而非只使用(gaussian_img)來自做成品。

若使用經過高斯濾波的邊緣資訊以及經過高斯濾波的二值化影像生成成品

用經過高斯濾波的邊緣資訊以及經過高斯濾波的二值化影像生成影像

```
[ ]  1 plt.figure(figsize = (8, 8))
     2
     3 plt.subplot(1, 2, 1)
     4 plt.title("adopt", fontsize = 15, color = 'r')
     5 plt.imshow(reduce_noise_img, cmap = 'gray')
     6
     7 nonadopt_reduce_noise_img1 = reduce_noise(nonadopt_adaptive_theshold_gaussian, edges_img, 6)
     8
     9 plt.subplot(1, 2, 2)
    10 plt.title("doesn't adopt", fontsize = 15, color = 'r')
    11 plt.imshow(nonadopt_reduce_noise_img1, cmap = 'gray')
    12
    13 plt.tight_layout()
```

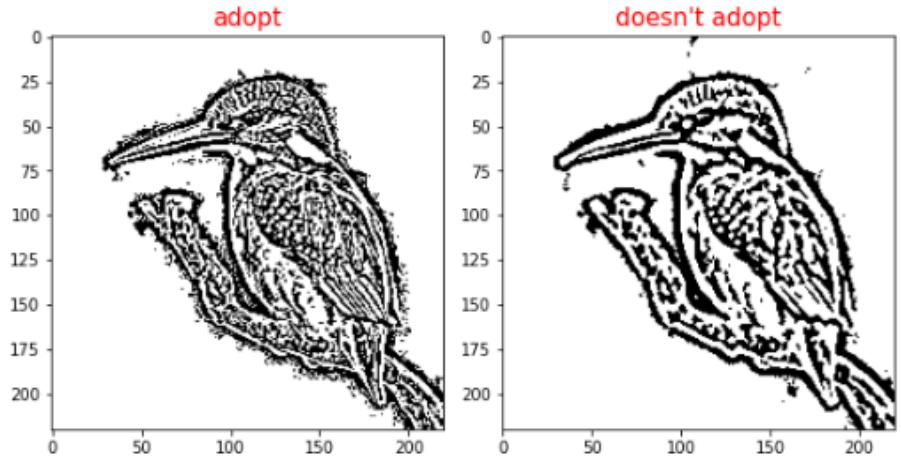


- 若使用的是經過高斯濾波後的影像生成邊緣資訊來生成成品，可以很明顯地看到鳥的背景是乾淨的，但是因為使用的二值化影像是經過高斯後的，所以鳥的精細度很差。

若使用未經過高斯濾波的邊緣資訊以及經過高斯濾波的二值化影像生成成品

用未經過高斯濾波的邊緣資訊以及經過高斯濾波的二值化影像生成影像

```
[ ]  1 plt.figure(figsize = (8, 8))
     2
     3 plt.subplot(1, 2, 1)
     4 plt.title("adopt", fontsize = 15, color = 'r')
     5 plt.imshow(reduce_noise_img, cmap = 'gray')
     6
     7 nonadopt_reduce_noise_img2 = reduce_noise(nonadopt_adaptive_theshold_gaussian, nonadopt_edges_img, 6
     8
     9 plt.subplot(1, 2, 2)
    10 plt.title("doesn't adopt", fontsize = 15, color = 'r')
    11 plt.imshow(nonadopt_reduce_noise_img2, cmap = 'gray')
    12
    13 plt.tight_layout()
```

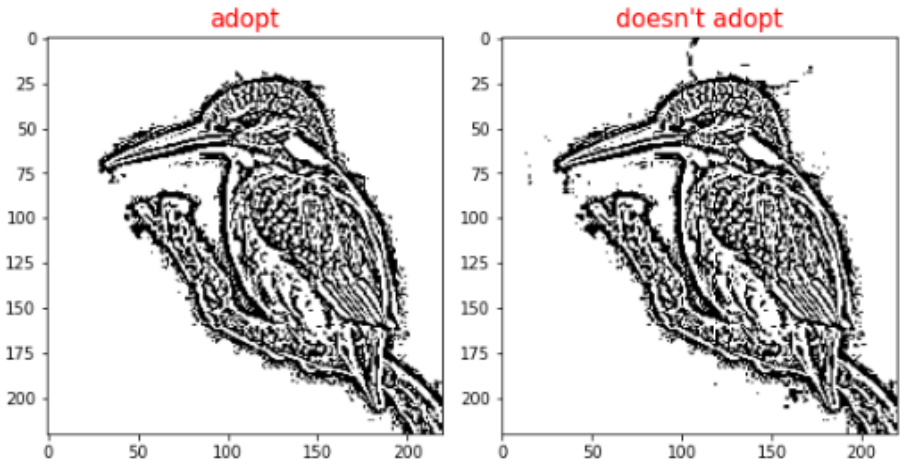


- 若使用的是未經過高斯濾波的邊緣資訊，以及經過高斯濾波處理後的二值化影像來生成成品的話，從上圖中可以很明顯地看到，鳥的背景有許多噪點，且影像精細度非常差。

若使用未經過高斯濾波的邊緣資訊以及未經過高斯濾波的二值化影像生成影像

用未經過高斯濾波的邊緣資訊以及未經過高斯濾波的二值化影像生成影像

```
plt.figure(figsize = (8, 8))

plt.subplot(1, 2, 1)
plt.title("adopt", fontsize = 15, color = 'r')
plt.imshow(reduce_noise_img, cmap = 'gray')

nonadopt_reduce_noise_img3 = reduce_noise(adaptive_theshold_gaussian, nonadopt_edges_img, 6)

plt.subplot(1, 2, 2)
plt.title("doesn't adopt", fontsize = 15, color = 'r')
plt.imshow(nonadopt_reduce_noise_img3, cmap = 'gray')

plt.tight_layout()
```



- 若使用的是經過高斯濾波的邊緣資訊，以及經過高斯濾波處理後的二值化影像來生成成品的話，從上圖中可以看到，鳥的精細度很高，但是鳥的背景還是有許多噪點。

心得

在老師宣布這一次報告的時候我還沒有甚麼頭緒，很迷茫要做些甚麼，但當我回到家的時候看到我蒐藏的動物水墨畫，我瞬間就有了靈感，為甚麼我要蒐藏別人的作品何不蒐藏自己的呢?也因此就有了這次的研究主題。

剛開始我對於要如何實踐時毫無頭緒，但當老師教到影像二值化和影像邊緣檢測的時候我突然靈光一閃，對阿，為甚麼我不依照檢測到的邊緣來將二值化後的影像變成水墨畫呢?若是依照邊緣資訊來製作，不僅可以免去不必要的背景，還可以生成出栩栩如生的水墨畫，一想到這我回家後就開始著手進行準備，雖然中間遇到過許多挫折，但是我還是克服困難將其完成了，對於成品的完成我非常的興奮，畢竟可以實踐自己的夢想，那是誰都最為可望的事情，以後我可以運用這次的成果，來蒐藏更多自己喜愛的作品，非常謝謝老師這次的報告給了我這個機會可以實踐夢想。

而在這次報告中，我學到了非常多東西，更是對於影像檢測有了更進一步的認知，而剛好我跟同學大三的專題實驗也是要做影像檢測相關的，我們的目標是打造出一個可以辨識垃圾的是屬於哪一類的機器，在有了對這些影像檢測技術有更加深刻的認知後，相信對我們的專題是非常有幫助的。

而我也期許我在外來任何事情上，都能保有現在的求知精神，不斷的摸索與嘗試，碰壁了也試圖從中找到突破口，不輕言放棄。

成果展示

成果影片連結

程式碼連結