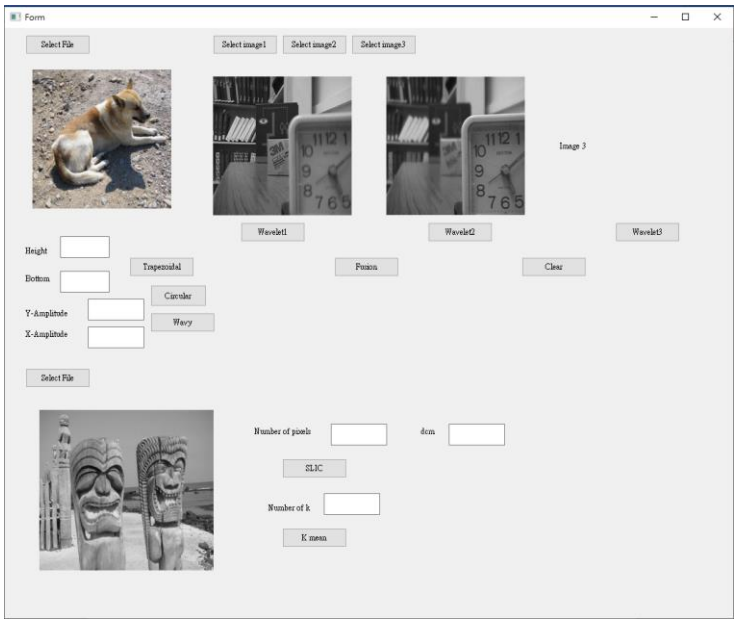
HW6

R12631015 易峻葦

介面:



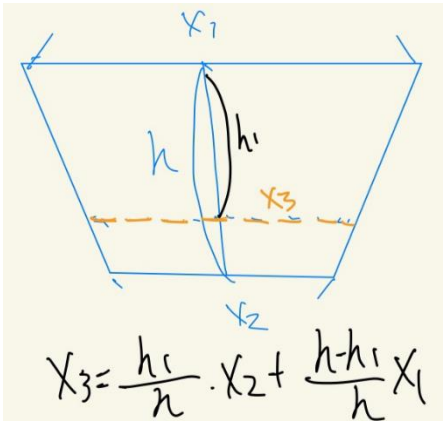
Part1

按下左上"Select File"按鈕後選擇狗狗的照片後就可進行各種轉換

(1)Trapezoidal Transform

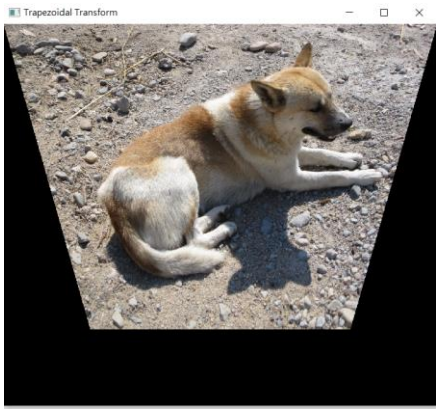
在 Height 空白欄中填入想要將圖片高度縮小的比例;Bottom 那一欄填入想要將圖片底部縮小的比例，按下 Trapezoidal 按鈕後即會出現。演算法採用的是根據兩欄的數字來進行線性的拉伸。

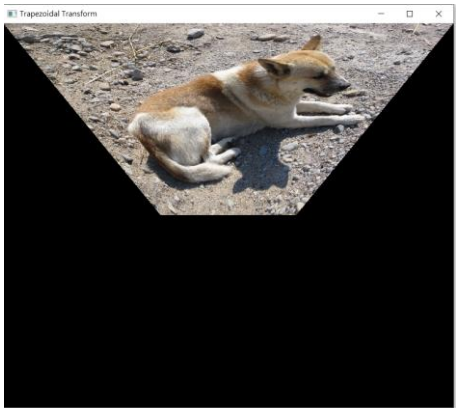
```
def Trapezoidal_transform(self):
    image= np.copy(self.im1)
    height = float(self.height.text())
    bottom = float(self.bottom.text())
    rows,cols =image.shape[:2]
    cx = int(cols/2)
    transform = np.zeros((rows,cols,3))

    for i in range(rows):
        new_h=int(height*i)
        transform[new_h,:,:]=image[i,:,:]
    h=int(height*rows)
    w=int(bottom*cols)
    for j in range(h):
        width=j/h*w+(h-j)/h*cols
        ratio=float(width/cols)
        for x in range(cols):
            transform[j,int(x*ratio),:]=transform[j,x,:]
            if(x>width):
                transform[j,x,:]=0
    final = np.zeros((rows,cols,3))
    for j in range(h):
        width=int(j/h*w+(h-j)/h*cols)
        for x in range(width):
            offset=cx-int(width/2)
            final[j,x+offset,:]=transform[j,x,:]
```



(Height =0.8 , Bottom =0.6)

(Height =0.5 , Bottom =0.3)





(2)Wavy Transform

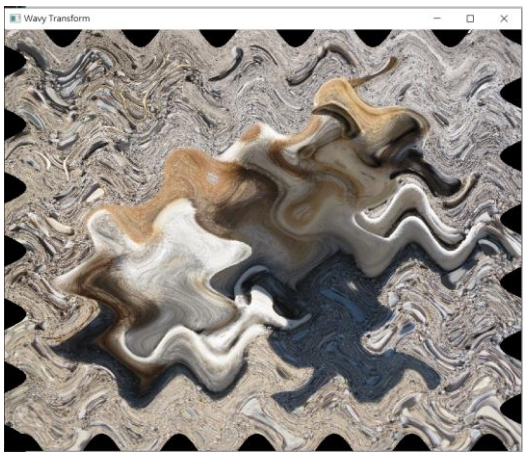
程式會創造一個與原圖片同樣大小的零矩陣，其中新矩陣像素點(x,y)與原來圖片像素點(i,j)關係為:

x $=X_{amp}\cos\left(\frac{2\pi*i}{180}\right)$, y=$Y_{amp}\sin\left(\frac{2\pi*j}{180}\right)$

在介面中 X-Amplitude、Y-Amplitude 欄中填入想要的振幅後按下 Wavy 鈕即可。

```
xamp=float(self.xamp.text())
yamp=float(self.yamp.text())
transform=np.zeros((image.shape))
rows, cols = image.shape[:2]
for i in range(rows):
    for j in range(cols):

        offset_x =  xamp * np.cos(2 * np.pi * i / 180)
        offset_y = yamp * np.sin(2 * np.pi * j / 180)

        if (j + offset_x < cols) and (i + offset_y < rows) and (j + offset_x > 0) and (i + offset_y > 0):
            transform[i, j,:] = image[int(i + offset_y) % rows, int(j + offset_x) % cols,:]

        else:
            transform[i, j,:] = 0
```

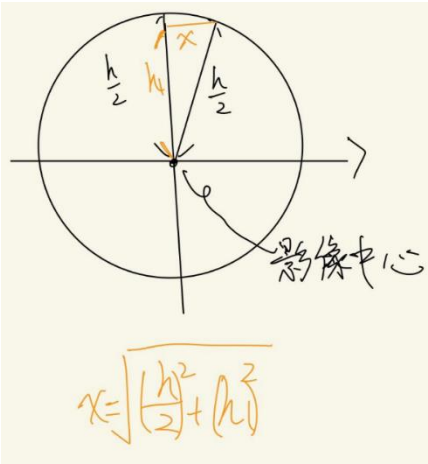
(Xamp=50,Yamp=50)

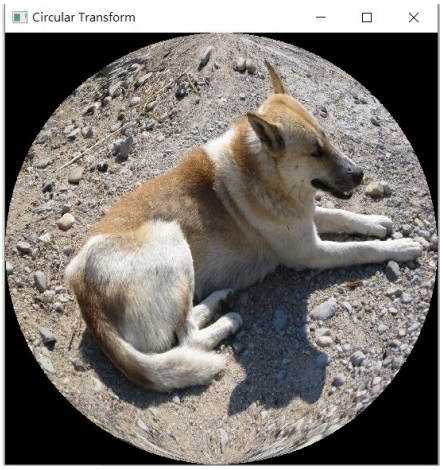


(Xamp=30,Yamp=30)

(3)Circular Transform

同樣是拉伸，想像以圖片為中心，直徑為影像高度 h 畫一個圓，計算原始圖片的每一個 row 與圖片中心的 row 相差 h1 後，以半徑 h/2 為斜邊，計算該 row 的長度該變成幾倍，以下圖為例，每個 row 都要縮成長度 2x:

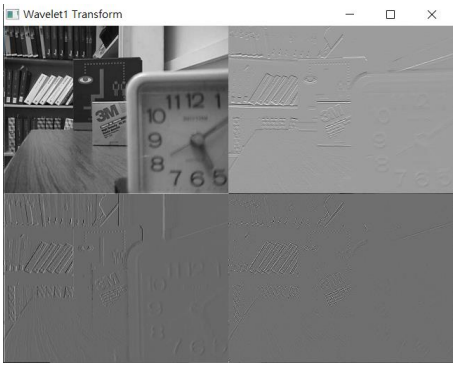


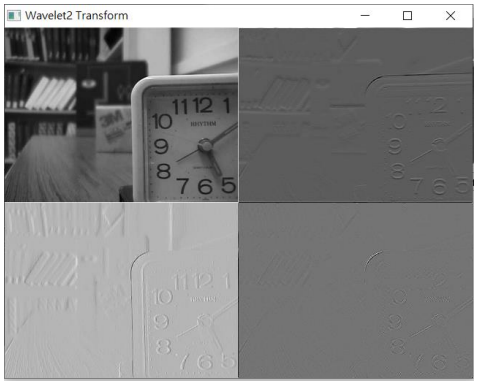


在介面中直接按下 Circular 按鈕即可出現結果

(Part2)

在介面中依序按下 Select image1、Select image2、(Select image3)後再依序按下 Wavlet1、Wavelet2、(Wavelet3)按鈕後就會依序出現兩(三)張圖片的小波轉換:



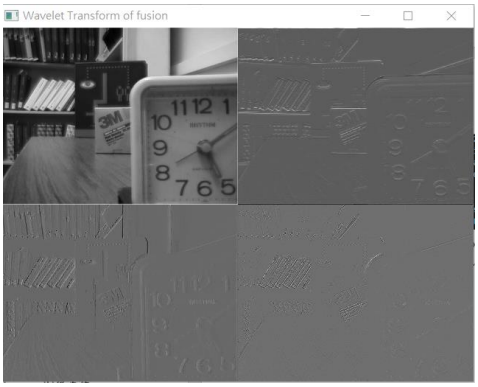


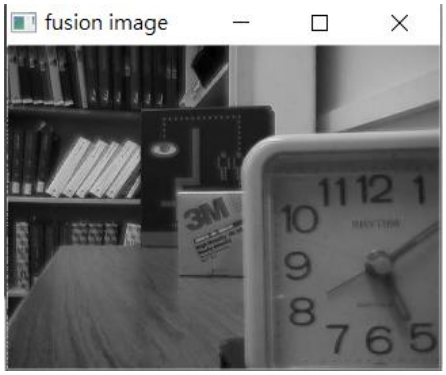
在按下不同 Wavelet 按鈕時，程式會順便將轉換後的圖片 LL、LH、HL、HH 合成一個 list 後再將其存進一個共用的 list:imgset，為之後影像合成所用:

```
image=np.copy(self.im21)
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
coeffs = pywt.dwt2(image, 'haar')
LL, (LH, HL, HH) = coeffs

wrows,wcols=LL.shape
output=np.zeros([wrows*2,wcols*2])
self.imgset.append([LL,LH,HL,HH])
```

想要合成圖片時，按下介面中 Fusion 按鈕即可，程式會將先前的 imgset 中的一個個元素拿出來比較它們的 LL、LH、HL、HH。LL 是取所有圖片 LL 的平均，
LH、HL、HH 則是取三者中的最大者:

```
rows,cols = self.imgset[0][0].shape
for  m in range(len(self.imgset)):
    if(m==0):
        LL_fusion = self.imgset[m][0]
        LH_fusion = self.imgset[m][1]
        HL_fusion = self.imgset[m][2]
        HH_fusion = self.imgset[m][3]
    else:
        for i in range(rows):
            for j in range(cols):
                LL_fusion[i][j]+=self.imgset[m][0][i][j]
                if(self.imgset[m][1][i][j]>LH_fusion[i][j]):
                    LH_fusion[i][j]=self.imgset[m][1][i][j]
                if(self.imgset[m][2][i][j]>LH_fusion[i][j]):
                    HL_fusion[i][j]=self.imgset[m][2][i][j]
                if(self.imgset[m][3][i][j]>LH_fusion[i][j]):
                    HH_fusion[i][j]=self.imgset[m][3][i][j]
LL_fusion = LL_fusion/(len(self.imgset))
fusion_image=LL_fusion+LH_fusion+HL_fusion+HH_fusion
```





從合成後的圖片可以發現原先兩個圖片中模糊的區域都被另外一張相較之下較不模糊的取代。要重新跟換一組新的圖片，按下 Clear 按鈕，程式就會把之前 imgset 中的圖片清除，接著再重新選擇新的圖片集即可:

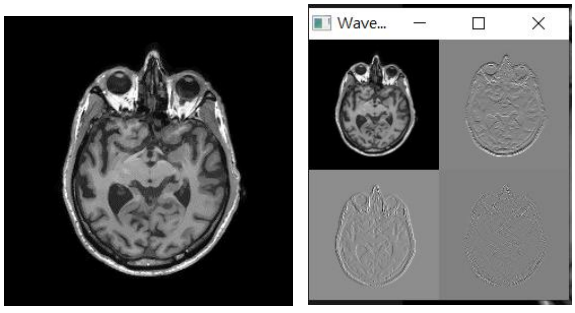


image1

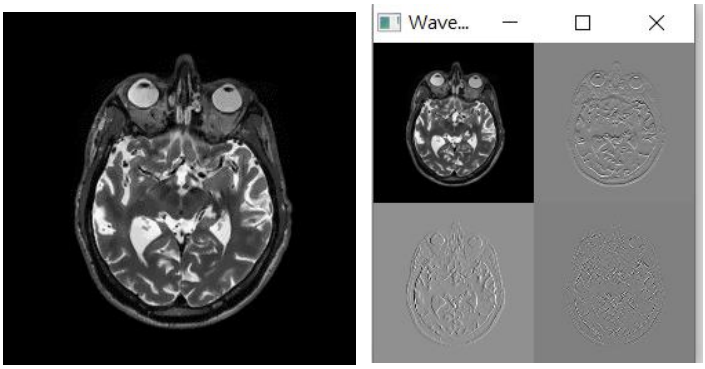
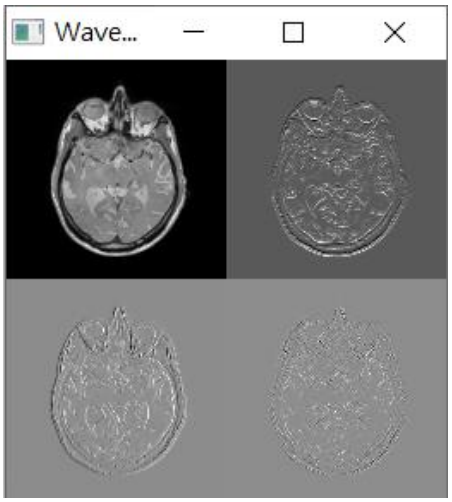
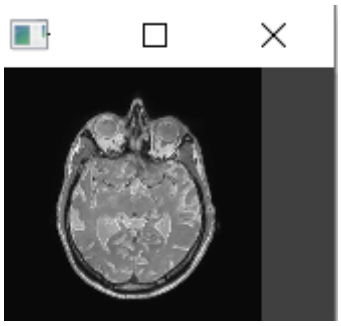


image2




fusion image

(Part3)

SLIC 演算法:

根據課本提供的資料以及提出該演算法的原始文獻，主要有兩個參數可以調，分別是決定像素的數目以及 label 時決定灰階強度比重的參數 dcm，而在原始論文有提到說通常這個演算法在重複計算 10 次後就會飽和，因此我是沒有設一個 criteria 讓每次 label 完後與其比較決定要不要做下一輪，而是直接讓這個演算法跑十次。

```
image=np.copy(self.im3)
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
rows,cols=image.shape
supixel = np.zeros(image.shape)
#print("total pixel number:",rows*cols)
sp = int(self.superpixel.text())
s = int(np.sqrt(rows*cols/sp))
#print(s)
dcm = int(self.Dcm.text())
center=[]
previous_mean = []
label= -np.ones((rows,cols),dtype=np.int16)
distance = 10**100*np.ones((rows,cols))
center=getcoordinate(center,rows,cols,s)
center=adjust_center(center,image)
for i in range(len(center)):
    x = center[i][0]
    y = center[i][1]
    previous_mean.append([image[x,y],x,y])#[Intensity,x,y]
```

```
def adjust_center(center,image):
    for i in range(len(center)):
        G=10**100
        x=center[i][0]
        y=center[i][1]
        for k in range(x-1,x+2):
            for m in range(y-1,y+2):
                gx=np.abs(image[k-1][m]-image[k+1][m])
                gy=np.abs(image[k][m-1]-image[k][m+1])
                if(G>np.sqrt(gx**2+gy**2)):
                    G=np.sqrt(gx**2+gy**2)
                    center[i][0]=k
                    center[i][1]=m
    return center
```

```
def getcoordinate(center,rows,cols,s):
    x_coordinate=[]
    y_coordinate=[]
    xs = int(s/2)
    ys = int(s/2)
    rx=int(rows/s)+1
    ry=int(cols/s)+1
    for i in range(rx):
        x_coordinate.append(xs+i*s)
    for j in range(ry):
        y_coordinate.append(ys+j*s)
    for k in range(len(x_coordinate)):
        for m in range(len(y_coordinate)):
            center.append([x_coordinate[k],y_coordinate[m]])
    return center
```

上方程式碼包含了將圖片依據 subpixel 的數量進行切割並決定每個區域的初始中心，但因為有時候 subpixel 的數量與原始數量的 pixel 並不等於整數，因此切割出來的數量多少會有點誤差。

```
for t in range(10):
    x_sum=np.zeros(len(center))
    y_sum=np.zeros(len(center))
    I_sum = np.zeros(len(center))
    count = np.zeros(len(center),dtype=np.int16)
    for c in range (len(center)):
        I_c=previous_mean[c][0]
        x_c=previous_mean[c][1]
        y_c=previous_mean[c][2]
        for x in range(int(x_c-s),int(x_c+s)):
            if(x<0 or x>=rows):
                continue
            for y in range(int(y_c-s),int(y_c+s)):
                if(y<0 or y>=cols):
                    continue
                euclidean = np.sqrt((x-x_c)**2+(y-y_c)**2)
                intensity = np.sqrt((int(image[x][y])-I_c)**2)
                D = np.sqrt((euclidean/s)**2+(intensity/dcm)**2)
                if (distance[x][y]>D):
                    distance[x][y]=D
                    label[x][y]=c
    for row in range(rows):
        for col in range(cols):
            cluster=label[row][col]
            count[cluster]+=1
            x_sum[cluster]+=x
            y_sum[cluster]+=y
            I_sum[cluster]+=int(image[row][col])
    for i in range(len(center)):
        Im=I_sum[i]/count[i]
        xm=x_sum[i]/count[i]
        ym=y_sum[i]/count[i]
        previous_mean[i] = [Im,xm,ym]
```

決定好初始 subpixel 中心的座標點後，重複去計算每個中心 2s 範圍每個 pixel 與其的距離 D= $[\left(\frac{dc}{dcm}\right)^2+\left(\frac{ds}{dsm}\right)^2]^{1/2}$，dc 為像素點語中心的灰階值差，ds 為兩者相差的距離。dsm 由 subpixel 的數量決定，dcm 為自設，其大小可以視為分群時參考顏色(這裡為灰階值差)的權重。

```
for i in range(rows):
        for j in range(cols):
            supixel[i][j]=previous_mean[label[i,j]][0]
            if(i == 0 or i==rows-1 or j==0 or j==cols-1):
                pass
            else:
                if(label[i][j]!=label[i+1][j] or label[i][j]!=label[i][j+1]):
                    image[i][j]=255
```

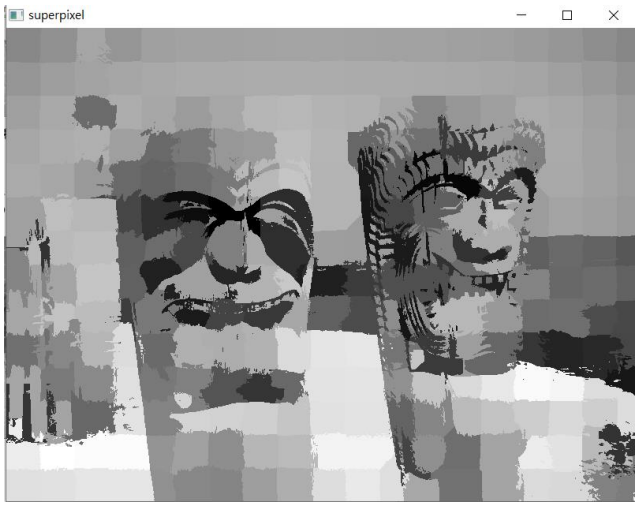
進行完分類後，將不同 label 間的區域畫出來

按下左下角的 Select File 的按鈕後即可選擇圖片，接著在 Number of pixels 以及 dcm 空白欄中輸入想要的數字後按下 SLIC 即可:

(Subpixel = 250 ,dsm=40)

從 Border 圖可發現，結果會有許多躁點，意思是在一大塊同個區域的 label 裡有一小塊卻是其他 label，在細節越多的地方越明顯。推測這是因為 dcm 的緣故，導致就算距離相對另一個中心較遠，但由於灰階值接近並且 dcm 不夠大，因此經過計算後被歸類在較遠的 label 中。

(Subpixel = 250 ,dsm=80)



當 dcm 越大時，躁點會越少，並且每個區域會越來越平滑，趨近於原始的正方形。Dcm 的大小要視目的而定，當要做影像閥值設定時，由於會考慮到色彩因素，因此 dcm 可能就要小點。與 K-mean 不同的點在於，由於 SLIC 還會參考空間資訊，因此出來的結果會有明顯的區塊，像馬賽克一樣。

K-mean=3: