110612117 張仲瑜 HW1

I. Method

可以將整個作業分成三個部分看

分別是中心旋轉、放大、以及三種插值法的撰寫(頁數限制只節錄部分)

之後再互相組合

A. 中心旋轉

```
10      ori_height, ori_width = image.shape[:2]
11      center = (ori_width//2, ori_height//2)
12      rotation_matrix = np.array([[np.cos(np.deg2rad(-angle)), -np.sin(np.deg2rad(-angle))],
13                                  [np.sin(np.deg2rad(-angle)), np.cos(np.deg2rad(-angle))]])
14      new_image = np.zeros((ori_height, ori_width, 3), np.uint8)
15      for i in range(ori_height):
16          for j in range(ori_width):
17              new_i, new_j = np.array([i - center[1], j - center[0]]) @ rotation_matrix + center
18              if 0 <= new_i < ori_height and 0 <= new_j < ori_width:
```

11~12: 取得中心的座標、設立旋轉矩陣

17: 因為是根據中心點旋轉，將原本的坐標扣掉中心座標後再旋轉，

之後再平移回本來的座標軸

18: 只保留原圖框內部分

B. 放大:乘上倍率倒數後無條件捨去即可取得原本的索引座標

C. Nearest Neighbor interpolation

```
77      for i in range(new_height):
78          for j in range(new_width):
79              new_image[i, j] = image[min(int(i/scale), ori_height - 1), min(int(j/scale), ori_width - 1)]
```

75~76: 採用反向投射，需要遍歷整張新圖

77: 將新圖每個索引除以放大倍率(如果需要)後無條件捨去（取整數）

找出原圖最接近的索引，再透過 min 函數限制範圍以防邊緣部

分超出索引值

D. Bilinear interpolation

```
94      new_image[i,j, channel] = (np.array([x2 - x, x - x1])
95                                 @ np.array([[image[x1, y1, channel], image[x1, y2, channel]],[image[x2, y1, channel], image[x2, y2, channel]]]
96                                 @ np.array([[y2 - y], [y - y1]]))
```

雙內插的式子可以提出共同項後整理乘矩陣形式

(1 * 2 @ 2 * 2 @ 2 * 1) = 常數

E. Bicubic interpolation

```
41  def cubic_interpolation(p0, p1, p2, p3, x):
42      p0 = p0.astype(np.float32)
43      p1 = p1.astype(np.float32)
44      p2 = p2.astype(np.float32)
45      p3 = p3.astype(np.float32)
46
47      value = ((-0.5) * p0 + 1.5 * p1 - 1.5 * p2 + 0.5 * p3) * pow(x, 3) + (p0 - 2.5 * p1 + 2 * p2 - 0.5 * p3) * pow(x, 2) + ((-0.5) * p0 + 0.5 * p2)
48      for i in range(3):
49          value[i] = max(0, min(value[i], 255))
50      return value.astype(np.uint8)
```

42~45: 先轉小數避免後續自動轉型出錯

47~50: 將助教給的公式套入計算三次函數的值後限制在 0~255 之間

```
114                 p0 = cubic_interpolation(image[max(0, x1-1), max(0, y1-1)], image[x1, max(0, y1-1)], image[min(ori_width-1,x1+1), max(0, y1-1)], ima
115                 p1 = cubic_interpolation(image[max(0, x1-1), y1], image[x1, y1], image[min(ori_width-1,x1+1), y1], image[min(ori_width-1,x1+2), y1],
116                 p2 = cubic_interpolation(image[max(0, x1-1), min(ori_height - 1, y1+1)], image[x1, min(ori_height - 1, y1+1)], image[min(ori_width-1
117                 p3 = cubic_interpolation(image[max(0, x1-1), min(ori_height - 1, y1+2)], image[x1, min(ori_height - 1, y1+2)], image[min(ori_width-1
118                 new_image[i, j] = cubic_interpolation(p0, p1, p2, p3, y%1)
```

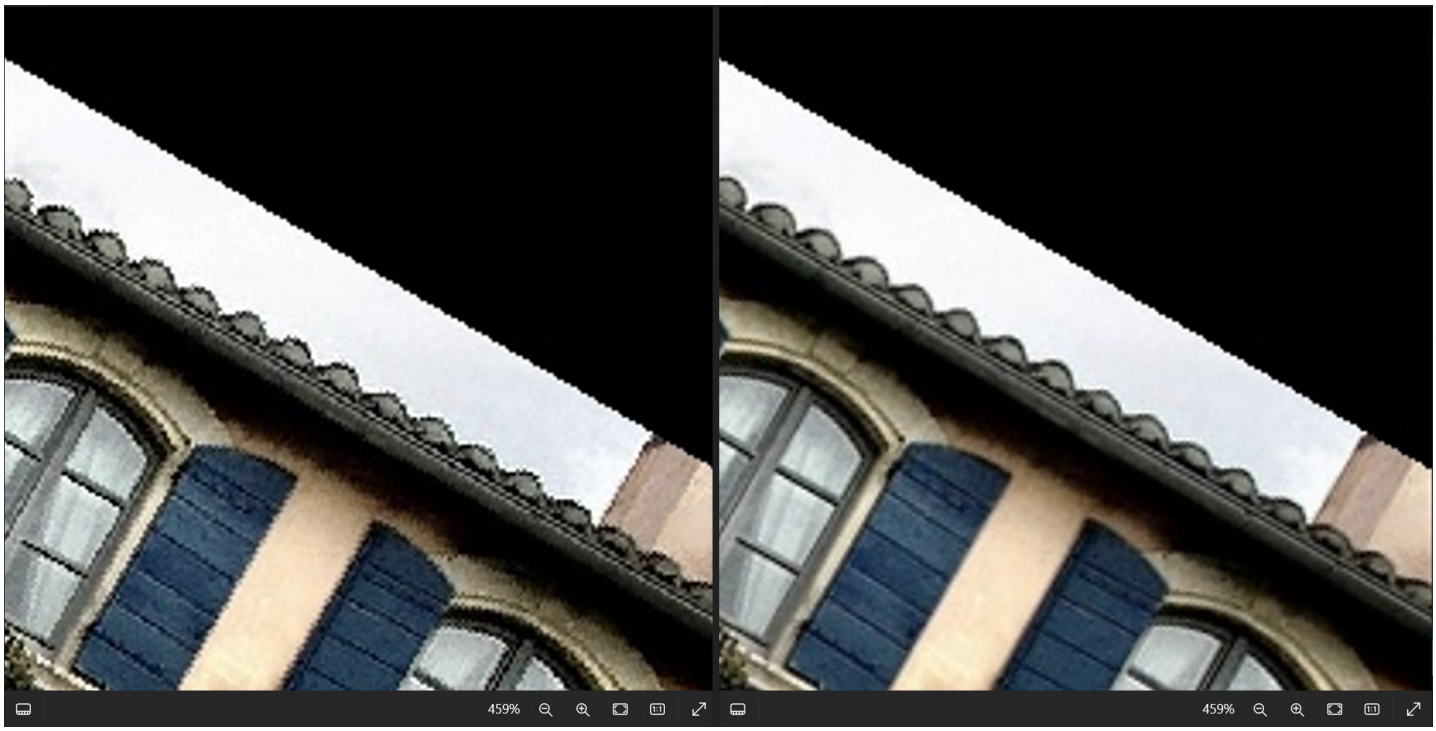
114~117: 先分別對 4 橫排 x 做三次內插，記得要限制索引範圍

118: 再對 1 直排 y 做三次內插後填入新圖

II. Result

A. Rotation 的右上角局部圖（左上 NN，右上 Bilinear，中左 Bicubic）

B. Enlarge 的右上角局部圖（中右 NN，左下 Bilinear，右下 Bicubic）



III. Feedback: 這次實作了常見的插值方式，也在過程裡收穫不少寶貴的經驗，例如提前轉型避免錯誤、條件限制避免邊界出錯等，可說是受益量多。但因為自己反順序完成，沒有一開始就意識到，透過函式對接可以避免重複撰寫，希望下次可以改進，讓程式更精簡。