影像處理原理與應用作業

學號：R11631029 姓名：林正浩 日期：1111004

Part1:

|  |
| --- |
| page1image18342432 |
| page2image18043360 |
| page3image18190192 |
| page3image18190192 |
| page4image18046480 |
| page5image17916240 |
| page6image17942352 |
| page7image18033840 |
| page8image18040656 |
| page9image17867504 |

Part2:

|  |
| --- |
|  |
| UI 介面 |

|  |
| --- |
| 灰階處理 |
| Code: 依題意有兩種灰階處理計算方式。  def gray(img, eq\_type): # img: array, eq\_type: eq1 or eq2  if eq\_type == 1: # eq1  result = (img[:, :, 0] + img[:, :, 1] + img[:, :, 2]) / 3  if eq\_type == 2: # eq2  result = 0.299 \* img[:, :, 2] + 0.587 \* img[:, :, 1] + 0.114 \* img[:, :, 0]  return np.round(result) |
| Result: |

|  |
| --- |
| 二值化 |
| Code: 根據threshold 的值去進行0 255的分類  def img\_threshold(Matrix, threshold):  h, w = Matrix.shape  Matrix\_threshold = np.zeros(Matrix.shape)  for i in range(h):  for j in range(w):  if (Matrix[i][j] < threshold):  Matrix\_threshold[i][j] = 0  else:  Matrix\_threshold[i][j] = 255  return Matrix\_threshold |
| Result: |

|  |
| --- |
| Resize |
| Code: 用兩種方式去進行內差  Nearest neibor:  def nearest\_neighbor\_resize(img, side\_percent):  side\_percent = side\_percent / 100  h, w = img.shape  new\_h, new\_w = (round(h \* side\_percent), round(w \* side\_percent))  resize\_img = np.zeros((new\_h, new\_w))  s\_w = (w) / (new\_w)  s\_h = (h) / (new\_h)  for i in range(new\_h):  for j in range(new\_w):  p\_x = int(i \* s\_h)  p\_y = int(j \* s\_w)  resize\_img[i][j] = img[p\_x, p\_y]  return resize\_img  Bilinear:  def bilinear\_interpolation(image, side\_percent):  side\_percent = side\_percent / 100  h, w = image.shape  new\_h, new\_w = round(h \* side\_percent), round(w \* side\_percent)  s\_w = (w) / (new\_w)  s\_h = (h) / (new\_h)  new\_image = np.zeros((new\_h, new\_w), np.uint8)  for i in range(new\_h):  for j in range(new\_w):  y = (j + 0.5) \* (s\_w) - 0.5  x = (i + 0.5) \* (s\_h) - 0.5  x\_int = int(x)  y\_int = int(y)  # Prevent crossing  x\_int = min(x\_int, h-2)  y\_int = min(y\_int, w-2)  x\_diff = x - x\_int  y\_diff = y - y\_int  a = image[x\_int, y\_int]  b = image[x\_int, y\_int+1]  c = image[x\_int+1, y\_int]  d = image[x\_int+1, y\_int+1]  new\_image[i, j] = a \* (1 - x\_diff) \* (1 - y\_diff) + b \* (1 - x\_diff) \* (y\_diff) + c \* (x\_diff) \* (1 - y\_diff) + d \* x\_diff \* y\_diff  return new\_image |
| Result: |

|  |
| --- |
| Brightness and Contrast |
| Code: 套用網路上找到的公式。乘上常數會改變對比，加上常數會調整明暗  def adjust\_brightness\_contrast(img, brightness, contrast):  result = img \* (contrast/127 + 1) - contrast + brightness  result = np.clip(result, 0, 255)  result = np.uint8(result)  return result |
| Result: |

|  |
| --- |
| Equalization |
| Code: 計算機率(PD)🡪計算累計機率密度(CPD)🡪將pixel value X CPD 進行均值化  def cal\_probability(img): # probability  h, w = img.shape  area = h \* w  hist = cal\_hist(img)  p = hist / area  return p  def cal\_Cumulative\_probability(p): # Cumulative\_probability  CP = np.zeros(len(p))  count = 0  for index, value in enumerate(p):  count += value  CP[index] = count \* 255  return CP  def equalization(img):  CP = cal\_Cumulative\_probability(cal\_probability(img))  h, w = img.shape  equalization\_img = np.zeros(img.shape)  for i in range(h):  for j in range(w):  equalization\_img[i][j] = CP[int(img[i][j])]  return equalization\_img |
| Result: |