HW1

1. Image reading

- **讀取 RAW 圖檔 ：**

  用 np.fromfile 把 3 個檔案(goldhill、lena、peppers)讀進來，而原本讀進來的 RAW 檔為一維陣列，所以建立一個 function 並在裡面建立一個新的 3 維陣列去把原本讀進來一維陣列的值一一存到三維陣列，圖片回傳的 shape 為 (512, 512, 1)，最後再用 img[251:261,251:261] 的方式讀取 512* 512 圖片中間 10 * 10 pixel 的值。

- **讀取 bmp 圖檔 ：**

  用 cv2.imread 把 3 個檔案(baboon、boat、F16)讀進來，再用 img[251:261,251:261] 的方式讀取 512* 512 圖片中間的 10 * 10 pixel 的值

2. Image enhancement toolkit

- **Log Transformations**

  定義一個名為 Log 的 fuction(**Log(im, chan)**)，需輸入的參數為"讀進來的圖檔" 和 "圖片的 channel 數 (bmp 圖檔為 3, raw 圖檔為 1) "。讀取 bmp 檔時 chan 帶 3，讀取 raw 檔時 chan 帶 1，再將圖片每個像素值帶入 Log Transformations 公式。

  [公式] :

  $s = T(r) = c\log(r+1)$ ， $c = 255/\log(r_{max}+1)$ (r 表象素值)

  通常圖片中像素最大值為 255，所以 c 也可以直接取 255/ log(256)，不過還是採取找出圖片值像素值中最大值帶入 c 公式。

- **Gamma Transformation**

  定義一個名為 ga 的 fuction(**ga(im, chan)**)，需輸入的參數為"讀進來的圖檔" 、"gamma 值(都設為 3) "、"圖片的 channel 數 (bmp 圖檔為 3, raw 圖檔為 1) "。將圖片每個像素值除以 255，正規化到 [0,1]，再用 gamma 數值做指數的調整，最後乘以 255 用 Maximum 做還原。

  [公式] : (r / 255)^gamma * 255　(r 表像素值)

- **image negative**

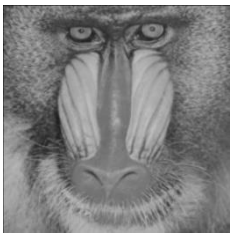
  定義一個名為 negative 的 fuction(**negative(im, chan)**) ，需輸入的參數為"讀進來的圖檔" 和 "圖片的 channel 數 (bmp 圖檔為 3, raw 圖檔為 1) " 。讀取 bmp 檔時 chan 帶 3，讀取 raw 檔時 chan 帶 1。再將圖片每個像素值用整張圖片最大的像素質(通常是 255) 減去原本的像素質。

  [公式] : r' = 255 – r (r'為 image negative 後的像素值，r 為原本的像素質)

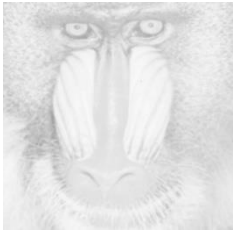
- **Comparision**

transformation	調整目的
Log	將影像較暗的地方，透過對數轉換，提升成較亮的像素質，調整後整張圖明顯明亮許多。
Gamma	能將過曝的照片調暗，曝光不足的照片調亮，而 gamma 值設定越大，圖片就越暗。
Negative	調整後黑白、深淺對調，能讓埋藏在黑暗影像中的白色或灰色線條更清楚，像狒狒的鬍鬚經過負片轉換後能更明顯看到。

Origin

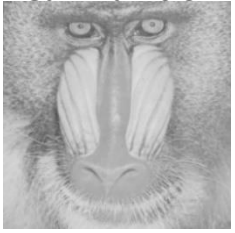


Log

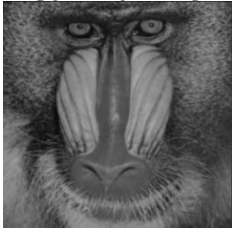


Negative

Gamma = 0.5



Gamma = 1.5



Gamma = 3

- **difficulties I met**
  在理解公式原理的時候花了一點時間，再將公式轉換成 Code 時也試了好幾次才成功。另外，在以上三個 Transformations 方法中，其中 image negative 有兩種做法，一種是用 255 去減掉原像素值，另一種則是用圖片像素值中的最大值去減掉原像素值，因不確定題目是指哪種方式，但網路上大多是用 255 去減掉像素值，所以採用了此方法。

3. Image downsampling and upsampling

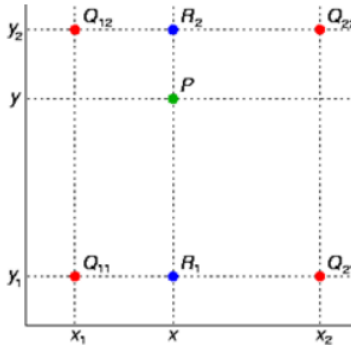
- **Nearest Neighbor interpolation**
  利用原圖片的長寬去除以目標圖片的長和寬得到比例後，就可計算出縮放後的點的位置，選擇離他最近的點，去代表他的值。
  [公式]:

$$\left.\begin{aligned} x\_ratio &= \frac{w_1}{w_2} \\ y\_ratio &= \frac{h_1}{h_2} \end{aligned}\right\} w_2, h_2 \neq 0$$

- **Bilinear interpolation**(以 baboon 這張圖當例子)
  1) 定義一個名為 GetBilinearPixel 的 fuction(GetBilinearPixel(imArr, posX, posY)) 去計算圖片每個點經過下面公式計算後的值，參數為 “ 讀進來的圖檔 ”、“ P 點的 X 位置 ”、 “ P 點的 Y 位置 ”

  $$Q_{11} = (x_1, y_1), Q_{12} = (x_1, y_2), Q_{21} = (x_2, y_1), Q_{22} = (x_2, y_2)$$

  方法:



先在 x 方向進行線性插值

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \quad \text{Where} \quad R_1 = (x, y_1),$$
$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \quad \text{Where} \quad R_2 = (x, y_2).$$

然後在 y 方向進行線性插值

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$

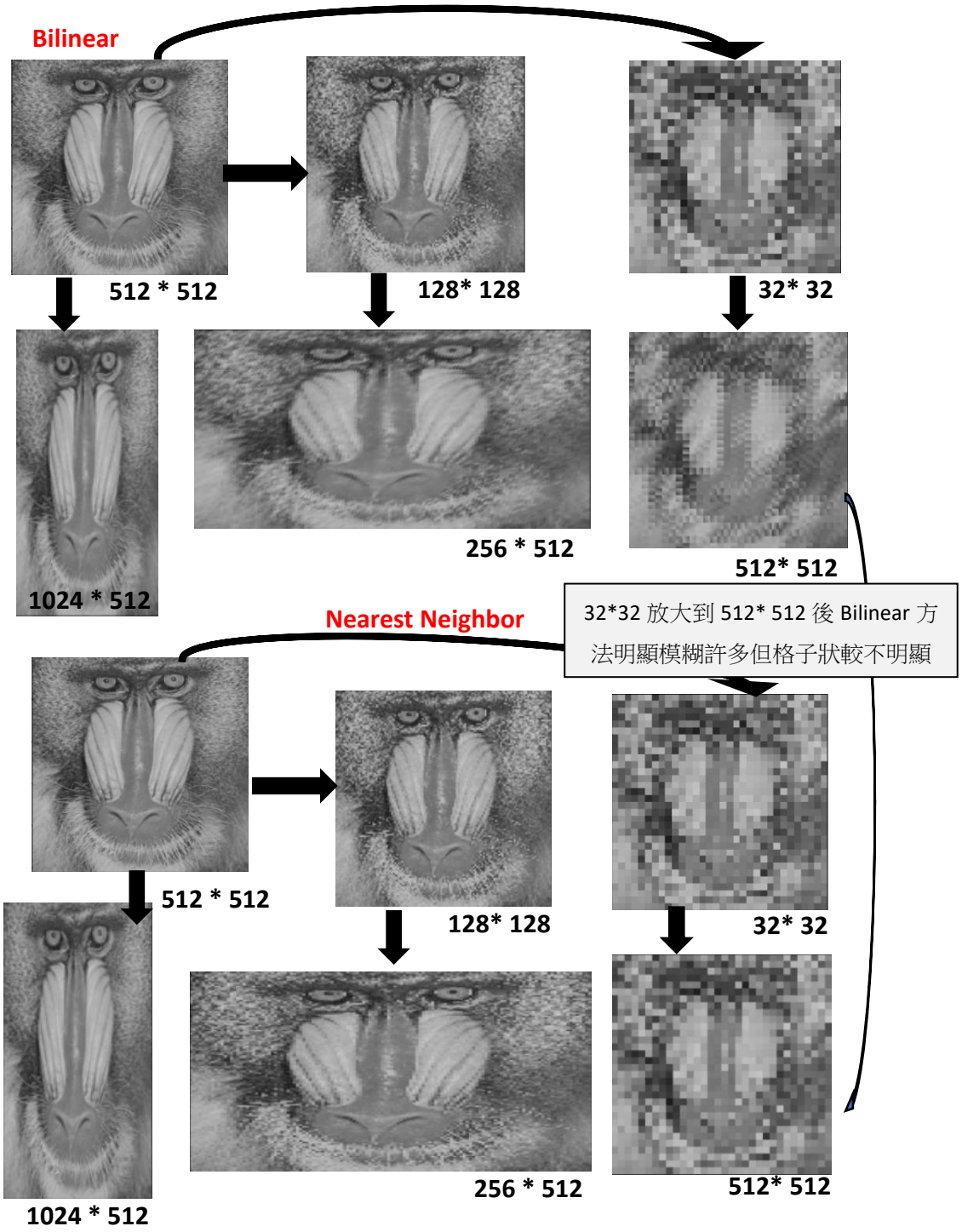
就可得到 f(x, y)為

$$f(x,y) \approx \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y)$$
$$+ \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1).$$

2) 建一個主要用來轉換圖片大小的 fuction (**bili(img, new_h, new_w)**)，參數為目標圖像的長和寬。再將剛剛第一步建的 GetBilinearPixel fuction 拿進來套用，並將目標圖像的第（i,j）個像素點（i 行 j 列）通過邊長比對應回源圖像，其對應坐標為 “ i*m/a,j*n/b “(假設源圖像大小為 mxn，目標圖像為 axb)

➢ Comparision

	運算複雜度	縮放狀況
Nearest Neighbor	較簡單，程式執行時間較短	較容易會有鋸齒狀及格子狀的現象發生。
Bilinear	較複雜，程式執行時間較長	較容易產生模糊狀，e.g. (32x32) -> (512x512)模糊十分嚴重



- **difficulties I met**

  在理解 Bilinear interpolation 公式原理的時候花最多時間，然後再將公式轉換成 Code 時也試了好幾次才成功。另外第一次實際用 code 跑後發現圖片不是正的，而是歪了 90 度，後來將圖片像素點 x，y 位置對調後，圖片才轉正，這是遇到比較匪夷所思的地方。