

DIP Homework 3 Image Enhancement

在這次作業中，輸入有 4 張看起來不太漂亮的幾張圖片，而所要求的輸出為將輸入圖片做「Image Enhancement」，輸出修正後的圖片，而主要的評估指標有：對比、亮度、雜訊、顏色。以下會分別討論我是如何處理這 4 張圖片，其主要分為：成品展示、觀察與問題發現、處理步驟與演算法。

Input 1

1. 成品展示

輸入圖片	輸出圖片
	

2. 觀察與問題發現

這張圖片中最大的問題違逆光而拍，因此這張照片在於牆內的對比不明顯，導致牆內許多細節都看不清楚；又因為中間有洞可以使光線透進來，在光與牆壁之間的對比又太高。並在仔細觀察，發現在右下角中有些許雜訊。故這張圖片的問題歸納為：牆壁間**對比**、**雜訊**。

3. 處理步驟與演算法

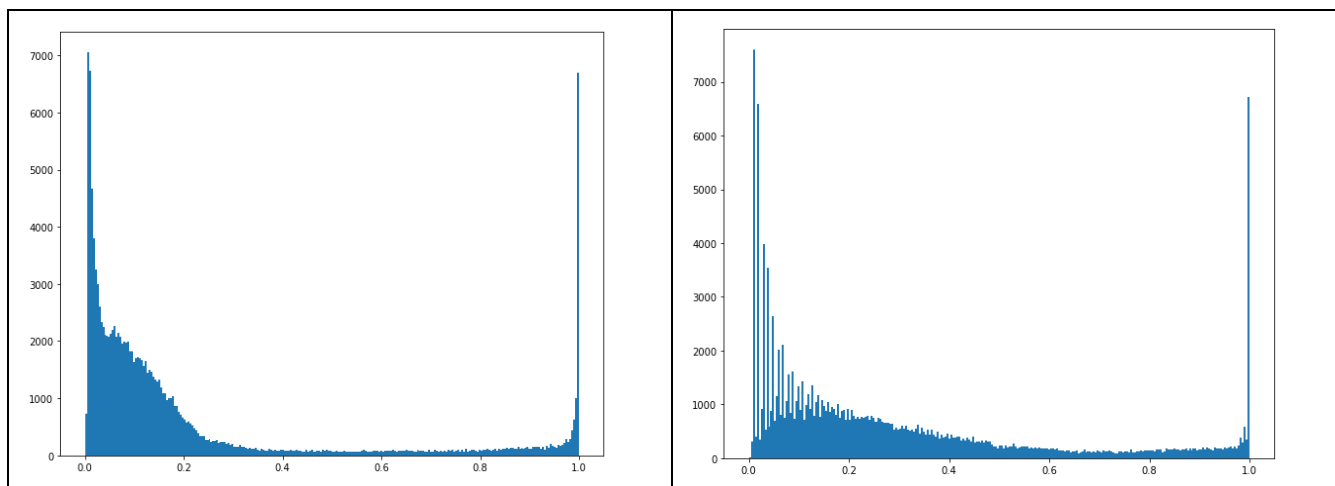
步驟	說明
讀取圖片	讀取 bmp 檔案，其資料為 sRGB
去除雜訊	使用一個 kernel 為[2, 2]的 median filter 來去除雜訊
將 sRGB 轉為 HSV	為了方便之後的直方圖等化所做的步驟(*1)
對 V 空間執行直方圖等化	在這裡我使用區域直方圖等化(kernel size =[100, 50])，而不是全域直方圖等化，此原因在之後解釋(*2)
輸出檔案	將圖片匯出，寫成 bmp 格式

(*1) 直方圖等化必須執行在圖片的亮度上，但在 **sRGB** 色彩空間中，並沒有有關亮度的資訊，因此我將色彩空間轉為 **HSV**，其中的 **V** 即代表著明度。另也有其他色彩空間可以選擇，像是 **HSL** 的 **L** 或是 **Lab** 的 **L** 等等。

(*2)

下表是執行直方圖等化前後 **V** 的直方圖：

執行前	執行後
-----	-----





而下表為執行區域直方圖等化與全域直方圖等化的結果比較：

區域直方圖等化	全域直方圖等化
	

由兩張圖的結果比較可以看出雖然使用全域直方圖等化可以使牆內的細節更加明顯，但整張圖看來有明顯失真的情況，像是在中下位置的行人整體變為紅色。因此在兩者比較下，我選擇左圖做為輸出，因為它比原圖多了牆壁的細節，但沒有明顯的失真。

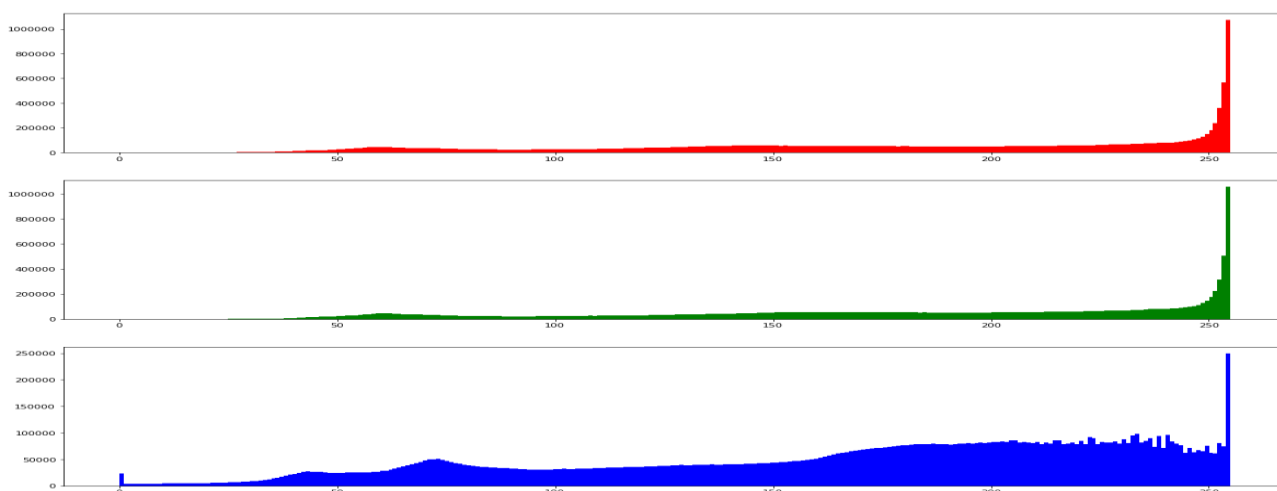
Input 2

1 成品展示

輸入圖片	輸出圖片
	

2 觀察與問題發現

這張圖片最大的問題就是過曝，使得原來的顏色都偏白，並且某些區塊的顏色已經完全無法區別，像是位於左下角的水面與植物，幾乎是相同的顏色。下圖為輸入圖片的 RGB 直方圖：



透過觀察可以發現在 RG 這兩個顏色，大多數的數值都位於較高的位置，甚至多數都位於 255 這個數值，這表示幾乎所有的像素點都處於過曝的狀態，完全沒了原來的資訊。但只有 B 這個顏色的分布是比較正常的。

3 處理步驟與演算法

3.1 去除雜訊：用一個 median filter(kernel size=[10, 10])去除雜訊。

3.2 轉換色彩空間：將圖片轉為 HSV 空間，並把 V 通道中每個像素的數值用 RGB 中 B 通道的數值覆蓋($V = B/255$)。此舉是透過上述的觀察中，B 通道的數值分布是相對正常的，因此將 B 通道覆蓋到 V 通道上，應該可以將亮度正常化。

3.3 直方圖等化：在 V 通道上執行全域直方圖等化，加強對比；比例增加 S 通道上的數值，將飽和度提升。

3.4 輸出圖片

Input 3

1 成品展示

輸入圖片	輸出圖片
	

2 觀察與問題發現

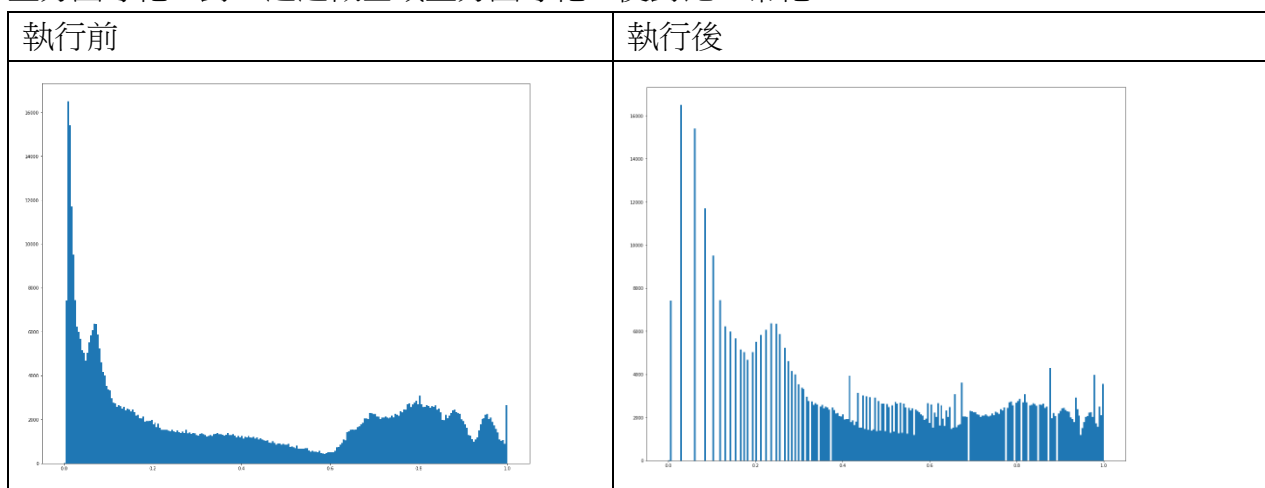
這張圖片的問題較為單純，一是有些雜訊，二則是由於逆光拍攝，導致人物的細節消失。因此主要的問題為雜訊、對比。

3 處理步驟與演算法

3.1 去除雜訊：用一個 median filter(kernel size=[3, 3])去除雜訊。

3.2 轉換色彩空間：將圖片轉為 HSV 空間

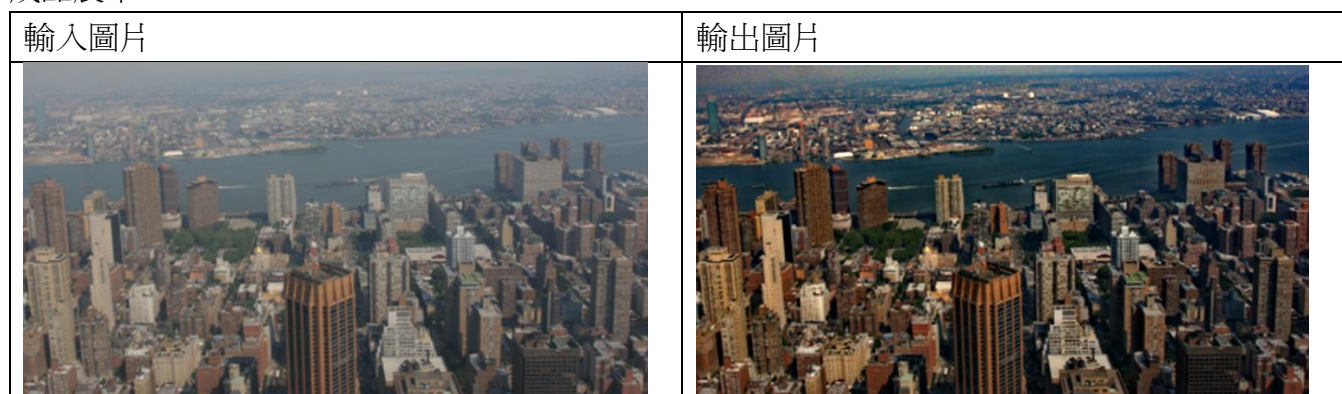
3.3 直方圖等化：對 V 通道做全域直方圖等化，使對比正常化。



3.4 輸出圖片

Input 4

1 成品展示



2 觀察與問題發現

這張圖片是 Dehaze/Defog 的一個經典範例，因此我是用參考[1]論文所提出的方法－「Single Image Haze Removal Using Dark Channel Prior」來完成此次作業。

3 演算法介紹

3.1 暗通道：對於任一個輸入的圖像，我們可以藉由下式取得他的暗通道：

$$J^{\text{dark}}(\mathbf{x}) = \min_{\mathbf{y} \in \Omega(\mathbf{x})} \left(\min_{c \in \{r, g, b\}} J^c(\mathbf{y}) \right), \quad (5)$$

若對此式做一個簡單的描述：將一張圖像的 RGB 中最低的值取出，在使用一個最小值濾波器對此值做濾波，kernel size 可自訂。

3.2 暗通道先驗：基於統計，可以歸納出一張無霧的圖像，其暗通道的直接接近 0

3.3 圖像形成模型：

$$\mathbf{I}(\mathbf{x}) = \mathbf{J}(\mathbf{x})t(\mathbf{x}) + \mathbf{A}(1 - t(\mathbf{x})), \quad (1)$$

$\mathbf{I}(\mathbf{x})$ 是現有的圖像， $\mathbf{J}(\mathbf{x})$ 是除霧後的圖像， $t(\mathbf{x})$ 是透射率， \mathbf{A} 為大氣光成分

3.4 透射率計算：根據參考[1]的推導，我們可以以下式計算透射率。

$$\tilde{t}(\mathbf{x}) = 1 - \min_{\mathbf{y} \in \Omega(\mathbf{x})} \left(\min_c \frac{I^c(\mathbf{y})}{A^c} \right). \quad (11)$$

3.5 除霧圖像：根據下式來還原圖像，此式是由 3.3 中的式子推導而來。

$$J(x) = \frac{I(x) - A}{\max(t(x), t_0)} + A. \quad (22)$$

A 值的選擇有許多種，我使用在暗通道中前 0.1% 的像素的數值平均作為 A。

3.6 優化：若只使用上述的方法，所得到的結果的確有達到除霧的效果，但有些物體旁會出現怪異的光圈現象；會出現這種現象，我推測是因為在計算暗通道時所使用的最小值濾波器所產生，該濾波器所產生的暗通道圖像黑白的邊界分明，這應就是產生光圈現象的主因。為了解決這個問題，我將暗通道再通過一個高斯濾波器。



4 實作步驟：

在實作中，我參考了 [meijieru/dehaze from github](https://github.com/meijieru/dehaze)，並依照前述演算法步驟執行。

4.1 取得暗通道

4.2 計算 A(大氣光成分)

4.3 藉由暗通道，計算折射率

暗通道數值(經高斯模糊)	折射率數值
	

4.4 藉由公式取得除霧後圖像

4.5 輸出圖片




Other Work

- 1 動機：事實上，一開始看到 Input 4 時，我想到的並不是使用 Dehaze 演算法，而是想將該圖片清晰化，因此便想使用 Super Resolution By Deep Learning 來使這張照片清晰化。以下實作由我與另一位同學楊詠翔共同完成。
- 2 我們參考了[3]，但我稍微簡化了一下模型。在[3]中，使用的輸入是一個尺寸較小的圖片，輸出是一個尺寸為原始大小 2 倍的圖片，我認為這樣的方法算是使用 neural network 來作 up-scale。而我們的目標為將模糊的圖片便清楚些，因此我稍微修改了模型，目標是輸入一張模糊的圖片，輸出一張清晰的圖片，網路架構如下：



Layer	Filter Number	Kernel Size	Activation Function
Convolution2D	128	(5, 5)	Relu
Convolution2D	64	(5, 5)	Relu
Convolution2D	32	(3, 3)	Relu
Convolution2D	16	(3, 3)	Relu
Convolution2D	8	(3, 3)	Relu
Convolution2D	3	(3, 3)	Sigmoid

- 3 訓練方法：使用 STL-10 dataset[4]，在這個模型中，我們只需要模糊的圖片與原始圖片，因此在訓練中，我是用的方法是將圖片先 down-sampling 再 up-sampling，來創造模糊的圖片。訓練 20 epoch，並使用 Adam 優化演算法。

4 成果展示：

原始圖片	先縮小再放大後作為輸入	直接作為輸入
		

由上表的第二欄可以得知若我們在 **testing** 的時候依照 **training** 時的做法，所得出來的結果相當不錯，幾乎與原圖無差異；但若我們直接將一張圖片輸入 **network**，他所得出來的結果便如第三欄所示，非常不理想，但這也是相當合理的結果，因為在 **training** 時我們輸入的都是模糊的圖片，在 **testing** 輸入一張清晰的圖片，那麼它的結果可預想到並不是太好。但是這樣的結果讓我們猜想是否這個神經網路只是一個較強的銳化圖片處理器呢？

Input 4	優化 Input 4
	

應用在 **Input 4** 上看起來有著不錯的效果，但似乎也就是對 **Input 4** 作銳化後的效果。

5 未來改進：

- 5.1 模糊圖片的產生來源增多，像是使用不同的 **up-scaling** 演算法、直接原始圖片作各種濾波(高斯、**median filter** 等等)，使這模型可以處理的模糊圖片增多。
- 5.2 嘗試減少 **layer** 的數量，[3]的作者只使用了 3 層網路即可將一張尺寸較小的圖片，轉為一張尺寸為 2 倍且品質比其他 **up-scaling** 演算法還好；對於我們這次的目標比原作者簡單，應該可以使用較小的網路。
- 5.3 在處理 **Input 4** 時，我使用[1]的方法，但其實現在有[5]的方法可以使用深度學習來作 **Dehaze**，稱為 **DehazeNet**，他使用神經網路的方法來提取暗通道、透射率，而不是[1]裡的統計方法。

參考資料

- [1] Kaiming He, Jian Su, and Xiaoou Tang. Single Image Haze Removal Using Dark Channel Prior, 2009
- [2] [meijieru/dehaze](https://github.com/meijieru/dehaze)
- [3] Wenzhe Shi, Jose Caballero, Ferenc Huszar, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, Zehan Wang. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network
- [4] [STL-10 dataset](https://arxiv.org/abs/1502.04397)
- [5] Bolun Cai, Xiangmin Xu, Kui Jia, Chunmei Qing, Dacheng Tao. DehazeNet: An End-to-End System for Single Image Haze Removal, 2016