

ML 實戰演練 Report

2019.6.30

組員資訊

隊長：電機二 b06901007 戴子宜

隊員：電機二 b05901003 徐敏倩

資工五 b03902063 施皓平

電機二 b06901040 楊千毅

Introduction and Motivation

我們組別選擇的題目為第二題：Image Dehazing。在真實世界中，拍照時常常會因為各種各種懸浮微粒，包括灰塵、煙、水分子，吸收或折射或散射光線，導致所接收到的影像品質有非常大的下滑。因此，如何將這些 Hazy 的影像做 dehazing，讓其在 dehazing 過後能夠恢復到原本清晰的樣貌，便是本次 ML 實戰演練的主題。

在傳統模型之中，我們通常會將這些 hazed 影像以下列方程式來描述：

$$I(x) = J(x)t(x) + A(x)(1 - t(x))$$

其中， I 是 hazed 的影像， J 是本來的影像， A 是環境導致的干擾、 t 是轉換的 mapping， x 代表每一個 pixel。有些 dehaze 的方法是基於以上方程式，也就是為上述方程式找到最佳解，這種方法較簡單，但會被上述模型侷限住，且此方程式也存在許多未考慮到或者考慮錯誤的因素。

而現在，由於我們認為上述的方程式過於簡化真實的現象，所以我們使用方法則是希望透過 encoder 和 decoder 的模型架構，直接找到 I 跟 J 的 mapping，這樣就不會被傳統模型的方程式限制住，直接找出最好的 function。

而這次我們組還有一個需要克服的難題，就是運算資源不足：RAM 空間不足、GPU 速度不夠快、GPU 記憶體不足等等，所以我們組不僅需要找到最好的方法，還要找到能夠節省資源的方法。

Data Processing/ Feature Engineering

我們看了去年其他人的作法，決定使用兩種 model 的架構來訓練，分別是 PFFNet 還有 Perceptual Pyramid Deep Network（詳細 model 架構於 Methods 說明）。

由於本次官方給的 dataset 每張圖片太大，如果直接拿來 train 的話，會造成 model 太大及記憶體空間不足，所以原則上資料處理都必須將原圖切塊，再將切塊後的圖片分別拿來 train model 或 dehazing。但這樣的作法會造成各切塊

之間的關聯性被 model 忽略掉。也因此在此 data processing 時就必須要注意如何彌補這個損失。

1. 在 PFFNet 的模型中，我們將圖片擷取成的較小的圖片（在此 experiments 中會提到切出不同圖片大小對結果的影響），為了加入旋轉和鏡像，在此 dataset 裡面，每一張照片會隨機的被旋轉至 0、90、180、270 度，同時隨機取水平或鉛直兩種鏡像，也就是說，每一個小圖片經過 data augmentation 會得到 12 張新的 training 圖片。
2. 在 Perceptual Pyramid Deep Network 中，將原本的照片分別切為 512x512, 1024x1024, 1024x2048, 2048x2048 和原本的大小，再將照片調整為 640x640。

Methods

1. Multi-scale Single Image Dehazing using Perceptual Pyramid Deep Network 使用 encoder-decoder 類型的模型架構：

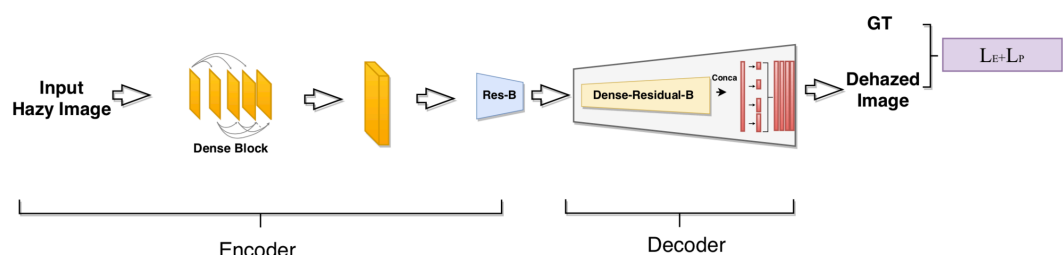
A. Encoder

利用 DenseNet 的 dense blocks 和 residual block，依照下圖的方式連接。Dense blocks 的部分參照 DenseNet-121，前三層分別有 12, 16, 24 個 densely-connected layers，而分別使用 DenseNet-121 model 的 pretrained weight。

B. Decoder

類似於 encoder 的架構，decode 同樣使用 residual 和 dense block，residual block 參考 ResNet，採 $u = F(v, W_i) + v$ 的關係式，其中， u, v 分別為特定層 input 和 output 的特徵， $F(x, W_i)$ 則是需要被 train 的 residual function。

Decoder 包含五個 dense-residual block 和 pyramid pooling module。每一個 dense-residual block 都包含兩層 dense block、一層 upsampling transition block 和兩層的 residual block。



圖（一）

C. Loss function

$$\text{Loss} = \lambda_E L_E + \lambda_P * L_P$$

L_E 就是一般的 MSELoss，而 L_P 是將 model 分別將生成的 dehaze image 和 Ground Truth 經過 VGG-16 的 model 的前幾層 layer，希望讓兩者的差異越小越好，就可以讓 perceptual difference 變小。我們使用的是 VGG-16 model 的 pretrained weights，並且使用的是 layer relu3_1 出來的 features。

D. Train parameters

Optimizer 使用 Adam，learning rate 為 2×10^{-3} ，batch size 為 1，總共跑 400000 個 iterations，並選定 $\lambda_E=1$ ， $\lambda_P=0.5$ 作為 loss 的參數。

E. Data post processing

利用 multi-scale ensemble strategy 將同一張照片經過 model 後的圖形合併。

i. Indoor

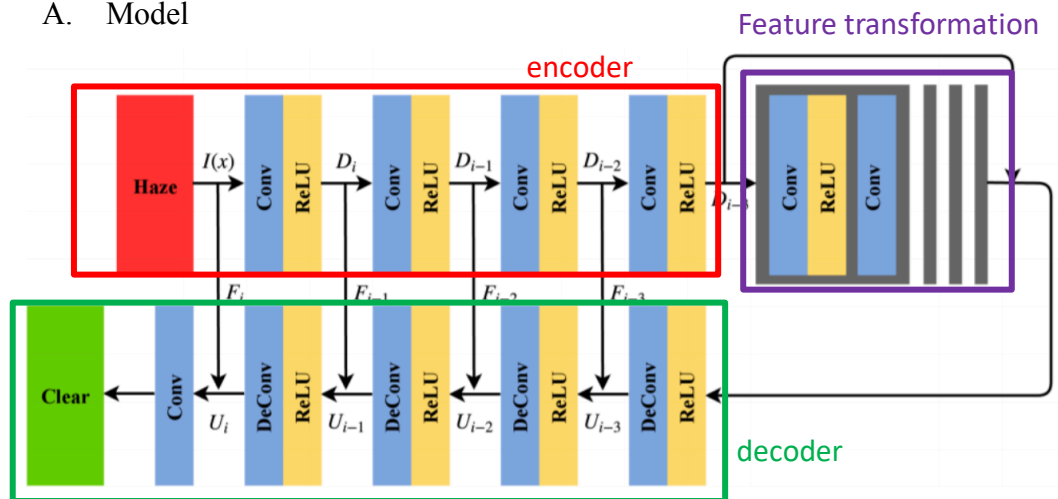
將原本的照片分割成 2048x1024 和 2048x2048 兩種，再分別將兩種的經過 model dehaze 完的圖片分別拼回完整的 output 圖形，並將兩張 output 圖形平均後得最後的 dehaze 圖片。

ii. Outdoor

將原本的照片分割成 3072x1536 和 1024x1024 兩種，再分別將兩種的經過 model dehaze 完的圖片分別拼回完整的 output 圖形，並將兩張 output 圖形平均後得最後的 dehaze 圖片。

2. PFFNet

A. Model



圖（二）

Model 的架構圖如圖（二），是 encoder-decoder 的架構。將 Haze Image 作為 input，接上 encoder 後再經過 feature transformation，最後經過 decoder 還原成 clear image。

Encoder 為 4 層 convolution layer，除了第一層的 stride 為 1，filter 個數為 16 以外，之後每一層的 stride 都是 2，且 filter 個數都是上一層的兩倍；

Feature transformation 是利用 18 個 residual network block（除了 convolution network 之外，還會將原本的 input 直接加到 output，這樣可以避免 layer 數增加後 train error 卻上升的問題）組成的；Decoder 則是和 encoder 對稱的結構，但是為了增加 multi-layer 之間的 information flow 還有確保 model 訓練的收斂性，所以 input 和 output 之間有 shortcut 相連，也就是圖中的 encoder 和 decoder 之間相連的部分 (F_i)。

B. Loss Function

在訓練時我們採用了不同的 loss function，一共使用了以下四種 Loss 的算法，詳細的算法和結果會在 Experiment 中說明：

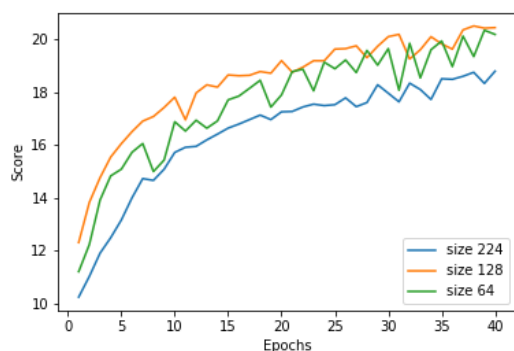
- i. MSELoss
- ii. PSNR * SSIM
- iii. MSELoss 加上 SSIM 的 regularization
- iv. MSELoss 加上 perceptual loss function
(參考 Perceptual Pyramid Deep Network 的方法)

C. Training parameters

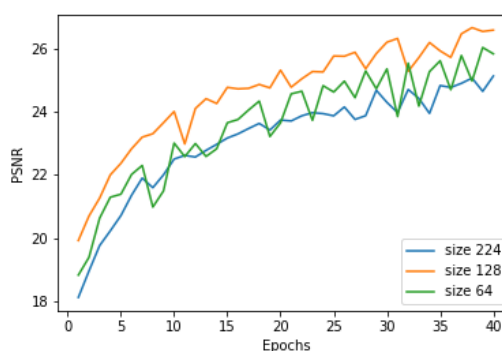
更新參數的方法是 ADAM，learning rate 初始值為 0.0001，大約到 40 個 epoch 左右會收斂。

Experiment and Discussion

1. 改變圖片大小



圖（三）



圖（四）

利用 PFFNet 的架構，改變 data preprocessing 時的圖片大小，分別將圖片 crop 成 224*224, 128*128, 64*64 的大小。圖（三）和圖（四）是 validation set 的 score (PSNR * SSIM) 和 PSNR 隨著 epoch 增加的變化，由結果可以發現，圖片大小對於結果會有影響，但沒有明顯的趨勢，因此我們最後決定將照片都切成 128*128。

2. 使用不同的 loss function

為了看不同的 Loss function 對於結果的影響，我們利用 PFFNet 的架構，使用以下四種不同的 Loss function。

- A. MSELoss：將 model 產生的 dehaze image 和原始圖片計算 MSELoss，也就是原本 paper 使用的方法。
- B. PSNR * SSIM：這是 JudgeBoi 計算分數的方法，因為是希望將這個數值最大化，所以在訓練時會加上負號。但這個方法訓練的結果不好，可能為了改變 SSIM 因此會影響到 MSELoss 的參數更新，最後得到的分數很低。
- C. MSELoss 加上 SSIM 的 regularization：因為直接使用 JudgeBoi 計算分數的方法，會為了 SSIM 反而讓 MSELoss 降不下來，所以決定先以 MSELoss 為主，等到 MSELoss 夠小才加入 SSIM 的影響，Loss Function 變為

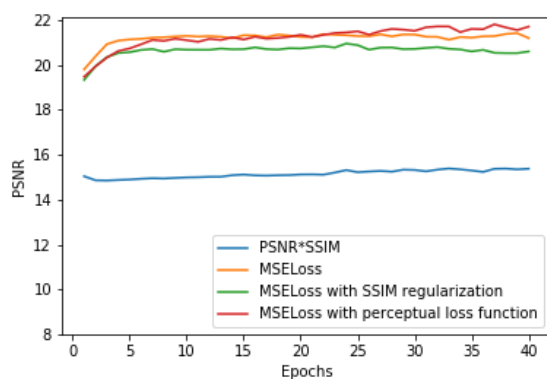
$$\text{MSELoss} - \mu * \text{SSIM}$$

其中的 μ 最後調整決定是 0.0001，也就是當 MSELoss 接近收斂時，上述兩項的大小會較接近。

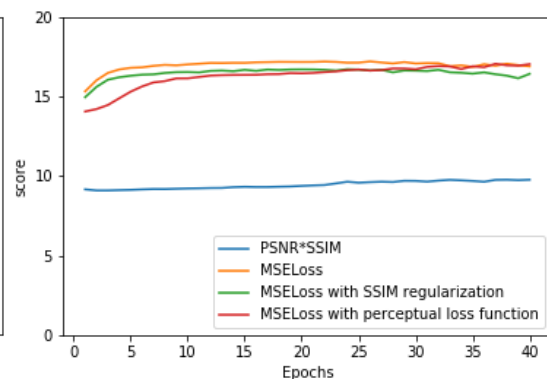
- D. MSELoss 加上 perceptual loss function：這個方法是參考 Perceptual Pyramid Deep Network 的方法，Loss Function 如下：

$$L_2 + \lambda * L_p$$

L_2 就是一般的 MSELoss，而 L_p 是將 model 分別將生成的 dehaze image 和 Ground Truth 經過 VGG-16 的 model 的前幾層 layer，希望讓兩者的差異越小越好，就可以讓 perceptual difference 變小。我們使用的是 VGG-16 model 的 pretrained weights，並且使用的是 layer relu3_1 出來的 features。 λ 取值為 0.5。



圖（五）





圖（六）

從結果可以發現，直接使用 JudgeBoi 的評分方式作為 Loss 反而得到的結果比較差，而如果用 MSELoss 加入 SSIM 的 regularization 的結果也不會比直接用 MSELoss 好。但如果使用 MSELoss 再加上 perceptual loss function，在 validation set 上一開始的分數會上升的比較慢，但最後會有辦法超越 MSELoss，因此我們選擇這種方式作為 Loss function。

3. 修改特定圖片

參照 final presentation 前三名組別所使用的方法，我們觀察利用 Perceptual Pyramid Deep Network（model 直接使用 pretrained model）dehaze 完的圖

片，可以看出 7.jpg dehaze 的效果較差，亦可發現在 training data 中找到 outdoor 的第五張圖片，與 7.jpg 十分相似，所以我們對於 7.jpg 做特別的處理。將 dehaze 完的 7.jpg 作為 source，outdoor 第五張的 ground truth 作為 reference，經過 histogram matching 後重新上傳。

Reference (05_outdoor_GT.jpg)	Source (7_dehaze.jpg)	Matched (7_matched.jpg)
		

表（一）

4. 比較不同方法的結果

由上述 1-3 點，我們可以利用調整 data processing 和 training method (loss function)，分別在 PFFNet 和 Perceptual Pyramid Deep Network 這兩個 model 架構獲得最好的結果，我們得到數據如表（二），因此最後選擇上傳 Perceptual Pyramid Deep Network 的方式到 github。

	PFFNet	Perceptual Pyramid Deep Network
JudgeBoi 分數	18.971644	19.914889

表（二）

Conclusion

我們利用分別利用 PFFNet 和 Perceptual Pyramid Deep Network 兩種 model 的架構，訓練出 decoder-encoder 的架構，生成 dehaze 的圖片。我們改變 data augmentation 和 loss 的方法，同時也對生成出來的圖片進行 data post processing，最後比較兩個上傳後的結果。

Reference

1. K. Mei, A. Jiang, J. Li, M. Wang, “Progressive Feature Fusion Network for Realistic Image Dehazing”
<https://arxiv.org/pdf/1810.02283.pdf>
2. “PFFNet” <https://github.com/MKFMiku/PFFNet>
3. H. Zhang, V. Sindagi, V. M. Patel, “Multi-scale Single Image Dehazing using Perceptual Pyramid Deep Network”

http://openaccess.thecvf.com/content_cvpr_2018_workshops/papers/w13/Zhang_Multi-Scale_Single_Image_CVPR_2018_paper.pdf

4. “Multi-scale Single Image Dehazing using Perceptual Pyramid Deep Network”
<https://github.com/hezhangsprinter/NTIRE-2018-Dehazing-Challenge>
5. “pytorch-ssim” <https://github.com/Po-Hsun-Su/pytorch-ssim>
6. “Histogram matching” https://scikit-image.org/docs/dev/auto_examples/transform/plot_histogram_matching.html