影像處理 HW2

109550087 單宇晟

**原理介紹**

* Deep learning

深度學習是機器學習的一個子領域，藉由訓練具有多層的neuron network，以執行圖像識別、語音識別、自然語言處理等任務。深度學習模型由多層相互連接的節點組成，每個節點會對其input套用一個數學函式，並將結果傳遞給下一層。每一層都會從input的數據中提取越來越複雜的特徵，使整個model能夠做出更準確的預測或執行更複雜的任務。深度學習模型使用大量labeled過的數據進行train，並在train的過程中不斷改變節點之間的weight，以盡量減少預測結果的誤差。深度學習的優勢是它能夠從原始數據中自動學習特徵，而不是像傳統機器學習那樣依賴手動去label。

* Multilayer perceptron

多層感知器（MLP）也是一種neuron network，由多層相互連接的neuron組成，每一層的每個neuron都會收到前一層中所有neuron的輸入。MLP的第一層是輸入層，會接收到最原始的數據；中間層又被叫做隱藏層，最後一層則是輸出層，會產生整個網絡最後的output。和deep learning的概念一樣，MLP中的每個neuron會對其input套用一個函式，產生出的的output就會被當作input傳遞給下一層的所有neuron，同時在train的過程中調整neuron之間的weight，達到理想的預測結果。

* Implicit Neural Representation

隱式神經表示（INR）是深度學習的一種技術，能夠用緊湊和高效的方式表示複雜的幾何體或物體。與傳統的幾何表示不同，傳統的幾何表示需要明確的參數化或表面特徵，INR模型則是能透過neuron network去define。在建立INR model的時候，首先會將空間中的一個點作為input，然後output一個代表物體屬性的value，像是該點與表面的距離或物體的顏色。通過對空間中的許多點進行sampling，並在每個點上對model去進行evaluate，就可以重建物體的完整表徵。

* View synthesis

View synthesis是computer graphics的一種技術，會從一組現有的view中生成場景或物體的新的viewpoint。View synthesis的目標是創建一個可以在場景或物體周圍移動的虛擬camera，並生成新的視角，而不需要再做額外的input。View synthesis是基於幾何學、或是影像的組合方法來實現的。幾何方法包括從現有的view中計算場景或物體的三維結構，然後透過將三維結構投射到虛擬camera上去，然後render新的viewpoint；基於圖像的方法則是通過使用deep learning去學習input的view和output的view之間的映射，從現有view中合成新的viewpoint。

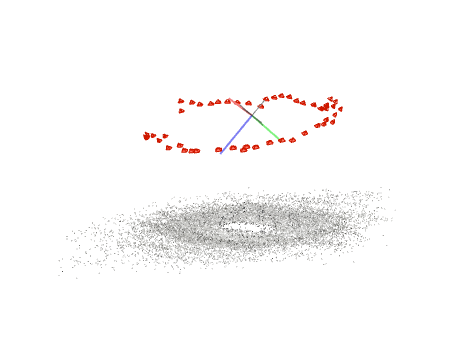
* NeRF (Neural Radiance Fields)

NeRF是一種比較新的深度學習方法，可以從二維圖像合成三維場景。NeRF使用一個叫做radiance field的連續函數來表示場景，它會透過train一個深度神經網絡來預測三維空間中每個點的顏色和不透明度。在train的過程中，會input一組從不同viewpoint拍攝的場景的二維圖像，之後就會進行優化，minimize預測的radiance和輸入圖像的radiance之間的差異。經過training之後，radiance field就能夠透過整合從相機位置到場景的三維空間射線的radiance，來render來自其他視角的新視圖。

**實作方法**

* Colmap

In the beginning, I follow the instructions on github to install COLMAP-3.8-windows-cuda, where it has a Colmap.bat file. Next, I take a series of photo, each of them should have an overlap between each other. Then, in Colmap, I first extract the feature. After this, we can start to match the features across the images. Next I sparse 3D reconstruction from the matched features, which will create a sparse 3D point cloud and camera poses from a set of 2D images. The point cloud represents the structure of the scene in 3D. Finally, we can get the following files: cameras.bin, images.bin, points3D.bin and project.ini.



* 深度學習網路

*train.py*

The whole program use only one class—NeRFSystem. The constructor initializes the model's hyperparameters, loss function, and two embedding layers, one for XYZ coordinates and the other for viewing directions. It also initializes NeRF model. Next is the decode\_batch(), it extracts rays and ground truth colors from a batch of data. And forward() performs batched inference on rays using NeRF model, where many hyperparameters are set. Then returns the predicted colors, depths, and other intermediate results. Fourth, prepare\_data() loads the training and validation datasets. Fifth, configure\_optimizers() just sets up the optimizer and learning rate.

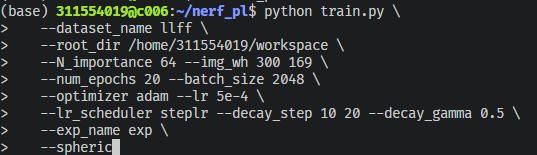
Next are train\_dataloader() and val\_dataloader(), they are very similar. Both of them return DataLoader objects for the training and validation datasets respectively. Then, comes training\_step() and validation\_step(), they define the forward pass for the model during training and validation respectively. The training\_step() calculates the loss, PSNR, and learning rate, and returns a dictionary. The validation\_step() also calculates the loss and PSNR and returns a dictionary. Last but not least, validation\_epoch\_end() calculates the mean validation loss and PSNR over all validation batches.

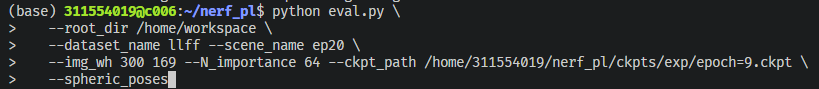
*Nerf.py*

This code has 2 class: Embedding and NeRF. They both contains only constructors and forward(). For Embedding, it embed input data x into a new representation by concatenating x with its sin() and cos() values. For NeRF, it implements a NeRF model, which takes input as an embedded vector of position and direction, then encodes it into a new representation. In the end, the model can predict the color and density at some points (other than input points). This NeRF model consists of multiple layers neural networks, which are used to encode the position and direction. And the output layers are for predicting color and density.

* 使用方法

基本上我都是照著github上的指令去做training，command如下：



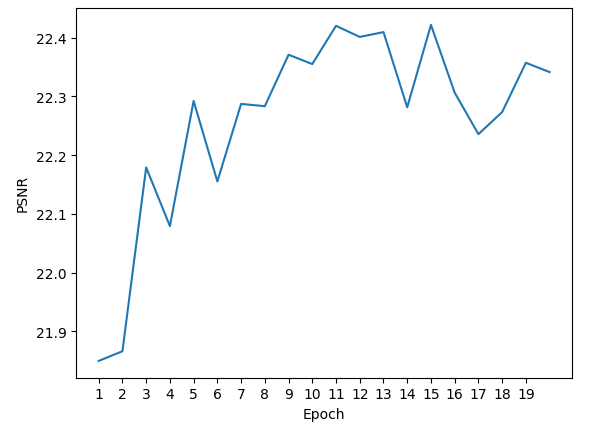
我首先更改了root\_dir，設成自己file的path、num\_epochs設成20、img\_wh則要跟自己拍攝的照片的height、width一樣、batch\_size我設成2048、最後，因為是做360 inward-facing，所以加上—spheric。

Test的部分，一樣把root\_dir設好，dataset\_name要改成llff，img\_wh也跟上面一樣，改成自己照片的size、並在ckpt\_path中設好這次要evaluate的ckpt檔、最後一樣加上—spheric就可以了。

**深度學習模型訓練及評估結果**

* Training

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch | 10 | 15 | 20 |
| PSNR (val) | 22.3551 | 22.4218 | 22.3415 |



在我training的過程中，可以發現，在前12個epoch左右的psnr是呈現一個上升的趨勢，然而過了epoch 12之後，psnr就開始震盪，並且有時還會降低，以下是我認為可能的原因：

(1) overfitting，過了epoch 12之後，model已經適當的train到一個段落了，之後的epoch就造成overfitting

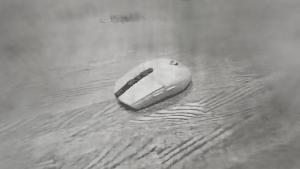
(2) learning rate，有可能learning rate 設太大，或是一開始不變，在psnr開始震盪的時候，learning rate要變小一點，model才不會圍繞optimal result震盪，無法收斂

(3) dataset，照片本身的解析度、大小(height, width)都會影響train的結果，拍攝的一連串照片之間的角度、遠近是否差距太大也會影響最終的結果；理想情況下，每張照片之間的角度差距應該要一樣、且和物體的距離也應該相同。

(4) 其他hyperparameter，NeRF這個model也有用到其他hyperparameter，有可能其他hyperparameter也要調整

* Testing

就肉眼觀察來說，我認為三張gif的表現其實差別不大，雖然epoch 12之後的psnr比較不穩定，但剛好我拿出來比較的epoch 15、20都沒有太大的變動，三者psnr的差距都在0.1之內，所以我覺得這結果是合理的。不過還是有一些比較明顯的差距：epoch 10的gif在環繞的時候，相比其他兩張會出現模糊的時候比較多，雖然epoch 15、20的gif有時候也會，但相比較少；另外，三張gif的上半部都出現模糊的情況(有點像烏雲，如圖)，我覺得這跟我拍攝的場景有關，有可能是因為巧拼的花紋導致的結果，也有可能是因為我的拍攝主體(滑鼠)和場景的顏色差距不太(白、灰)，造成這種結果。



**分析討論**

* INR

*Pros*

INR models are generally more compact and computationally efficient than NeRF models, since they can be learned with simpler neural networks.

*Cons*

However, INR models are usually more difficult to train than NeRF models, because their loss functions need to be carefully designed, and carefully regularized to avoid overfitting. Moreover, the result of INR models are usually worse than NeRF in showing details and lighting effects. In fact, INR are only trained to represent a single property of the scene.

* NeRF

*Pros*

NeRF models can represent very high-quality renderings of scenes with complex lighting effects and materials. NeRF models can be trained to represent multiple properties of the scene, just like this homework, it can show color, lighting, normal vector and etc.

*Cons*

On the other hand, NeRF models are computationally expensive to train and render, as they require the integration of a high-dimensional function along each ray passing through the 3D space. In fact, it took me almost 20 hours to train a model with 20 epochs.

這次homework主要是讓我們了解NeRF的運作，我遇到的第一個難題，就是架設環境，由於原作者使用的版本太舊了，所以光是架設環境就耗費我不少時間，之後在train的時候又遇到CUDA error: out of memory的問題，幸好我有來得及去借設備。另外，我這次其實不只拍照一次，在第一次準備dataset的時候，我沒有注意到拍攝的距離問題，相機忽遠忽近，結果導致model無法進行training，之後我才重拍一次，盡量用同等距離、高度去拍攝；另外照片的解析度也很重要，如果解析度太高，會讓colmap有feature exhaust的問題，之後我就把照片大小resize到600x338，但這樣的設置似乎還是不夠小，做training的每個epoch的時間太久(一個epoch大概100分鐘)，於是我就再重新resize成300x169，時間就快上許多(一個epoch大概40分鐘)；不過或許是解析度太低的問題，我train出來的結果都有些模糊，如果時間允許的話，我會想試試看用高解析度的dataset去做training，並提高dataset的size(我目前是53張)，這樣的結果應該會有所差異。