GAI HW4 Accelerating DDPM with DIP-based Initial Priors

一、Resource:

dip：參考作者的 github

ddpm：參考別人的實作模型

（dip 和 ddpm 皆以 Unet 為架構實作）

資料集選擇：FASHIONMINST

環境：Colab T4

二、模型架構：

DIP:

```
Sequential(
  (1): Concat(
    (0): Sequential(
      (1): Sequential(
        (0): ReflectionPad2d((0, 0, 0, 0))
        (1): Conv2d(3, 4, kernel_size=(1, 1), stride=(1, 1))
      )
      (2): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (3): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (1): Sequential(
      (1): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(3, 128, kernel_size=(3, 3), stride=(2, 2))
      )
      (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (3): LeakyReLU(negative_slope=0.2, inplace=True)
      (4): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
      )
      (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (6): LeakyReLU(negative_slope=0.2, inplace=True)
      (7): Sequential(
        (1): Concat(
          (0): Sequential(
            (1): Sequential(
              (0): ReflectionPad2d((0, 0, 0, 0))
              (1): Conv2d(128, 4, kernel_size=(1, 1), stride=(1, 1))
            )
            (2): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (3): LeakyReLU(negative_slope=0.2, inplace=True)
          )
          (1): Sequential(
            (1): Sequential(
              (0): ReflectionPad2d((1, 1, 1, 1))
              (1): Conv2d(128, 128, kernel_size=(2, 2), stride=(2, 2))
            )
            (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (3): LeakyReLU(negative_slope=0.2, inplace=True)
            (4): Sequential(
              (0): ReflectionPad2d((1, 1, 1, 1))
              (1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
            )
            (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (6): LeakyReLU(negative_slope=0.2, inplace=True)
            (7): Sequential(
              (0): ReflectionPad2d((0, 0, 0, 0))
              (1): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
            )
            (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (9): LeakyReLU(negative_slope=0.2, inplace=True)
          )
          (2): Upsample(scale_factor=2.0, mode='bilinear')
        )
      )
      (2): BatchNorm2d(132, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (3): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(132, 128, kernel_size=(3, 3), stride=(1, 1))
      )
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): LeakyReLU(negative_slope=0.2, inplace=True)
      (6): Sequential(
        (0): ReflectionPad2d((0, 0, 0, 0))
        (1): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
      )
      (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (8): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (8): Upsample(scale_factor=2.0, mode='bilinear')
  )
)
(2): BatchNorm2d(132, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(3): Sequential(
  (0): ReflectionPad2d((1, 1, 1, 1))
  (1): Conv2d(132, 128, kernel_size=(3, 3), stride=(1, 1))
)
(4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(5): LeakyReLU(negative_slope=0.2, inplace=True)
(6): Sequential(
  (0): ReflectionPad2d((0, 0, 0, 0))
  (1): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
)
(7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(8): LeakyReLU(negative_slope=0.2, inplace=True)
(9): Sequential(
  (0): ReflectionPad2d((0, 0, 0, 0))
  (1): Conv2d(128, 1, kernel_size=(1, 1), stride=(1, 1))
)
(10): Sigmoid()
```

DDPM:

```
MyDDPM(
  (network): MyUNet(
    (time_embed): Embedding(800, 100)
    (te1): Sequential(
      (0): Linear(in_features=100, out_features=1, bias=True)
      (1): SiLU()
      (2): Linear(in_features=1, out_features=1, bias=True)
    )
    (b1): Sequential(
      (0): MyBlock(
        (ln): LayerNorm((1, 32, 32), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(1, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (1): MyBlock(
        (ln): LayerNorm((10, 32, 32), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (2): MyBlock(
        (ln): LayerNorm((10, 32, 32), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
    )
    (down1): Conv2d(10, 10, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (te2): Sequential(
      (0): Linear(in_features=100, out_features=10, bias=True)
      (1): SiLU()
      (2): Linear(in_features=10, out_features=10, bias=True)
    )
    (b2): Sequential(
      (0): MyBlock(
        (ln): LayerNorm((10, 16, 16), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(10, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (1): MyBlock(
        (ln): LayerNorm((20, 16, 16), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (2): MyBlock(
        (ln): LayerNorm((20, 16, 16), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
    )
    (down2): Conv2d(20, 20, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (te3): Sequential(
      (0): Linear(in_features=100, out_features=20, bias=True)
      (1): SiLU()
      (2): Linear(in_features=20, out_features=20, bias=True)
    )
    (b3): Sequential(
      (0): MyBlock(
        (ln): LayerNorm((20, 8, 8), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(20, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (1): MyBlock(
        (ln): LayerNorm((40, 8, 8), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (2): MyBlock(
        (ln): LayerNorm((40, 8, 8), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
    )
    (down3): Sequential(
      (0): Conv2d(40, 40, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
      (1): SiLU()
      (2): Conv2d(40, 40, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    )
    (te_mid): Sequential(
      (0): Linear(in_features=100, out_features=40, bias=True)
      (1): SiLU()
      (2): Linear(in_features=40, out_features=40, bias=True)
    )
    (b_mid): Sequential(
      (0): MyBlock(
        (ln): LayerNorm((40, 4, 4), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(40, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (1): MyBlock(
        (ln): LayerNorm((20, 4, 4), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (2): MyBlock(
        (ln): LayerNorm((20, 4, 4), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(20, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
    )
    (up1): Sequential(
      (0): ConvTranspose2d(40, 40, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
      (1): SiLU()
      (2): ConvTranspose2d(40, 40, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
    )
    (te4): Sequential(
      (0): Linear(in_features=100, out_features=80, bias=True)
      (1): SiLU()
      (2): Linear(in_features=80, out_features=80, bias=True)
    )
    (b4): Sequential(
      (0): MyBlock(
        (ln): LayerNorm((80, 8, 8), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(80, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (1): MyBlock(
        (ln): LayerNorm((40, 8, 8), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(40, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (2): MyBlock(
        (ln): LayerNorm((20, 8, 8), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
    )
    (up2): ConvTranspose2d(20, 20, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (te5): Sequential(
      (0): Linear(in_features=100, out_features=40, bias=True)
      (1): SiLU()
      (2): Linear(in_features=40, out_features=40, bias=True)
    )
    (b5): Sequential(
      (0): MyBlock(
        (ln): LayerNorm((40, 16, 16), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(40, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (1): MyBlock(
        (ln): LayerNorm((20, 16, 16), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(20, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (2): MyBlock(
        (ln): LayerNorm((10, 16, 16), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
    )
    (up3): ConvTranspose2d(10, 10, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (te_out): Sequential(
      (0): Linear(in_features=100, out_features=20, bias=True)
      (1): SiLU()
      (2): Linear(in_features=20, out_features=20, bias=True)
    )
    (b_out): Sequential(
      (0): MyBlock(
        (ln): LayerNorm((20, 32, 32), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(20, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (1): MyBlock(
        (ln): LayerNorm((10, 32, 32), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
      (2): MyBlock(
        (ln): LayerNorm((10, 32, 32), eps=1e-05, elementwise_affine=True)
        (conv1): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): SiLU()
      )
    )
    (conv_out): Conv2d(10, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
)
```

三、Theoretical Justification (30%):

DDPM（Denoising Diffusion Probabilistic Models）和 DIP（Deep Image Prior）是種不同的深度學習方式，主要用於圖像處理，如去除噪點、圖像重建等方面。

DDPM:

DDPM 是一种生成式模型，能夠有效地恢復嚴重損壞的圖象區域，生成高品質且具多樣性的圖片。它通過模擬反向擴散的過程，由高斯噪聲逐步生成需要的數值，最後的輸出不但具有較高的真實性，其應用也相對多元。此外，相較於傳統監督學習，DDPM 不需要相對應的訓練資料（同時擁有乾淨的圖和加入噪點的圖），因此，在不具備大量標記資料時，DDPM 依然具有一定價值。

即使如此，它在使用方面也有一些侷限。DDPM 的訓練和推理過程涉及多項步驟的生成過程，因此會需要較長的時間和計算資源。另外，由於此模型的性能高度依賴擴散步驟和噪音水平的正確設置，這些參數的調整皆需精細的實驗和經驗。

DIP:

DIP 利用卷積神經網路（CNN）的結構作為自然影像的 encode prior，證明了即使沒有大量訓練樣本，單一影像的深層特徵也能有效地用於影像復原任務。也就是說，DIP 不需要訓練資料集，直接對特定任務進行最佳化，簡化了模型的使用和部署，再加上它並不依賴預訓練模型，DIP 可以靈活應用於各種不同的影像修復和重建任務。

不過，基於 DIP 是針對單一影像最佳化，其泛化能力通常不如基於大數據訓練的模型，且如果最佳化時間過長，DIP 相當容易過度擬合到雜訊或損壞的特性。

考慮到兩者的優缺點，於此次實驗中，嘗試使用 DIP 模型為 DDPM 模型的訓練過程提供快速初始 prior。這種方法主要是透過利用特定於影像的 prior 來解決 DDPM 中緩慢的 backward 學習過程。

首先，我先在挑選出的特定資料集上，短暫的訓練 DIP 模型，以獲取影像中存在的 high-level 結構。然後將此經過訓練的 DIP 模型置於 DDPM 模型中，作為其修復影像的初始 prior，替代掉純雜訊的輸入。這種做法可以提供 DDPM 更多資

訊，使其有機會降低所需的擴散步驟數量，並加速 DDPM 模型的收斂，相較於 DIP 和 DDPM，不僅可以避免 DIP 過低的泛化能力，也能有效減少 DDPM 訓練中所需花費的運算資源。不過，結合兩種方法也有可能會造成計算負擔進一步的增加。此外，使用者也需細緻地條整框架，以有效結合優勢，避免過擬合的情況發生。

四、Experimental Verification (40%):

關於 DIP 的訓練，是透過 PSNR 之計算來表示訓練效果。當 iteration 數量上升時，可以發覺最初 PSNR 值也跟著上升，代表著去噪後出的圖有就好的表現。但是到了一定程度後，可以發現數值就停滯不前了，也就是 DIP 的極限，在後面就有一些 overfitting 的情況發生了。

	50	100	150	200	250	300	350	400
PSNR	6.71	8.33	8.79	8.64	8.25	8.24	8.32	8.43

關於 DDPM 與 綜合 DDPM 和 DIP 的比較：

下表顯示 DDPM 和 DIP+DDPM 等不同模型，於 step 等於 1000 時所花的訓練時間：

	Training Time
DDPM	34 min 18.67 sec
DIP + DDPM	36 min 8.40 sec

我們觀察發現，使用 DIP 當作初始的 Prior，會花費比純 DDPM 模型的時間多一些。

至於下表，則表示在不同的 step 下，DDPM 和 DIP+DDPM 三種模型生成圖片所花的時間(秒數)，以及訓練結果的 Loss 值。其中的 DIP 模型，是隨意從資料集中挑選一張訓練而成，迭代次數設為 3。可以發現的是，在 DDPM 方面，隨著 step 數目的增加，訓練一張圖的時間也會增加，但相對的，最後 Loss 收斂的速度及程度也會有一定提升，最後生成的圖片也擁有較好的品質；另外關於結合 DIP 和 DDPM 的部分卻不是如此，在 step = 200~1000 時可以發現，隨著 step 的提升，模型並沒有增進表現，loss 值並沒有下降的趨勢，反而隨著 step 一起上升。

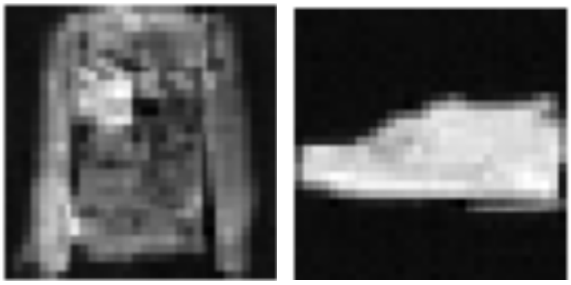
Time:

Step	200	400	600	800	1000
DIP+DDPM	1.25	2.63	3.87	4.83	6.26
DDPM	1.38	3.43	4.13	6.02	6.35

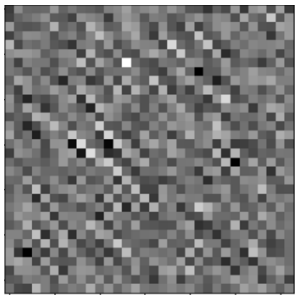
Loss:

Step	200	400	600	800	1000
DIP+DDPM	0.45	0.52	0.53	0.56	0.58
DDPM	0.065	0.050	0.043	0.039	0.032

下圖為 DDPM 中，step = 200 / 1000 產生出的圖片差異。可以看出，隨著 step 上升，確實模型去噪生成的圖能擁有更清晰均勻的畫質。



而關於這次實驗的模型，會發現雖然在訓練上看起來，它確實有收斂及學到了一些東西， 但展示出的圖片並未正確去除造點，只產生出了一些輪廓。



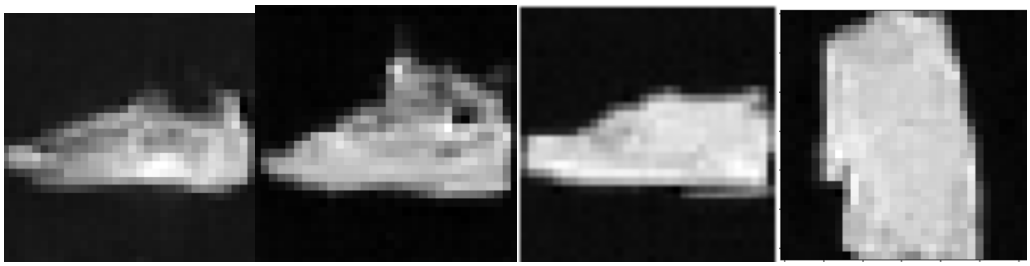
進一步比較這兩種模型，在 step = 1000 的情況下，DDPM 本身剛開始的收斂速度比起另一種還要快很多，在 epoch = 1 時， DDPM 的 loss 值是 0.078，遠小於 DIP+DDPM 的 0.63。原先預期隨著 epoch 增加，模型收斂後應該會提升些表現，但是後來會發現 DIP+DDPM 模型的收斂速度非常緩慢，且提升 epoch 並無法有效提升模型表現，我推測是因為 DIP 取得的特徵太單一，缺乏靈活性導致無法增進收斂。而在 step = 200 時，DIP+DDPM 的 loss 值反而低於 step = 1000，我認爲可能是因為在每一次的 step 中，DDPM 都會進行一次 DIP，然而 DIP 的特性是只針對單一圖像，所以在擷取其他資料集的 high-level 結構的時候，把重要資訊當成造點去掉了，因此成果反而越來越差。從實驗結果來看，DIP+DDPM 似乎無法生成較好的圖像品質。

五、Ablation Studies and Analysis (30%)：

從 DIP 的特性觀察，它僅針對某張圖片重組結構，因此，我最初做了一個設想，若是減少 DIP 對單一圖的訓練，以避免其過度擬合向某種特徵，是否能增進表現？我將訓練 DIP 的迭代次數減少，從 100、50、10、3 進行實驗，結果確實有一些不同。相較於在迭代次數 100 時產生的純粹佈滿噪音的圖像，迭代次數為 3 時能稍微辨別出一點輪廓，雖然表現皆不理想，也可以看出一些進步。所以在實驗上，我採用迭代 3 次的 DIP，作為 DDPM 的 prior。

至於 epoch 的選擇，我將模型從 epoch = 5~20 訓練了一次，後發現大約於 epoch = 15 時，模型的 loss 就沒再有下降的趨勢了，所以我選擇 15 epoch 當作實驗的數值。

下圖為 epoch = 5, 10, 15, 20 時，DDPM 所生成的圖像比較：



所以在這次實驗中，我以 DIP 迭代次數 3、epoch 為 15 來訓練模型並進行比較分析。

另一方面，針對 DIP+DDPM 模型表現不好的狀況，我猜測或許資料集佔了很大一部份的原因。在資料集中包含的圖片種類越多，僅能針對單一圖片進行處理的 DIP 效果就越是不好，所以在 FASHIONMNIST 這個包含了 10 種分類衣物的資料集中，它無法正常產生圖片結果好似也是可以預期的。所以我猜測，也許在某一種只包含較為單一圖像的資料集中，能取得較好的表現也不一定。