

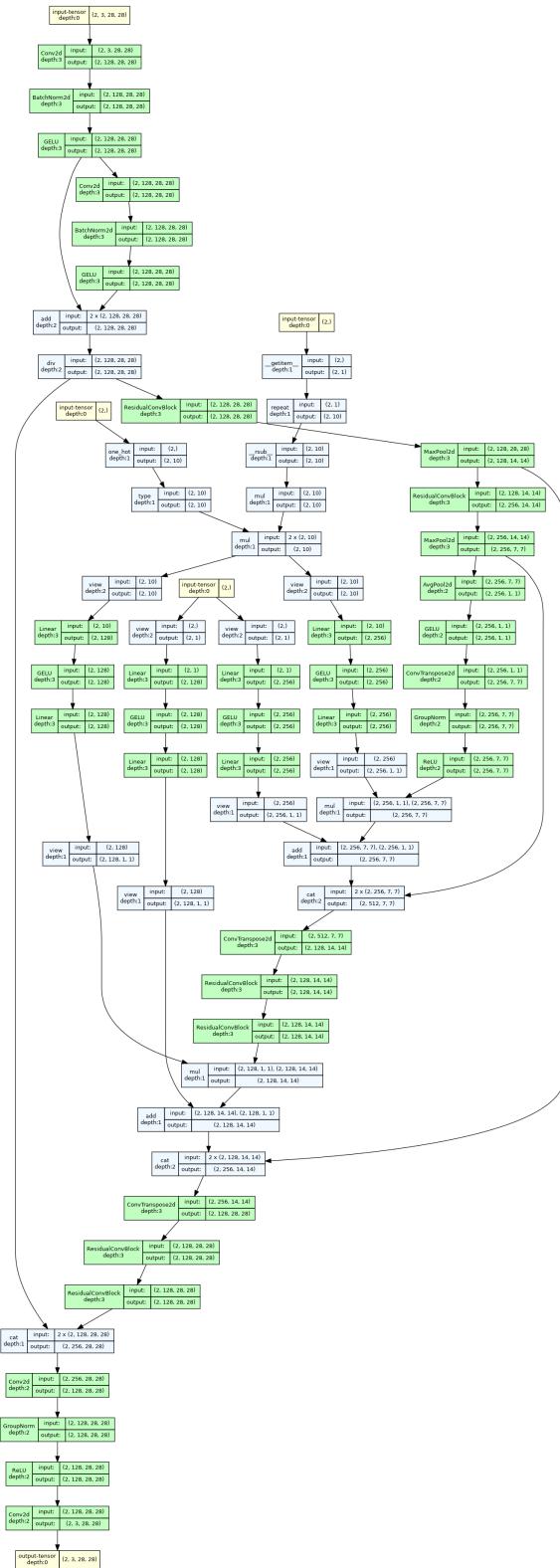


# DLCV-Hw2 Report

R11942180 電信碩二 黃湛元

## Problem 1: Report (20%)

1. (5%) Follow the [Github Example](#) to draw your model architecture and describe your implementation details.



Model 架構我是使用了

[https://github.com/TeaPearce/Conditional\\_Diffusion\\_MNIST/tree/main](https://github.com/TeaPearce/Conditional_Diffusion_MNIST/tree/main) 這篇的 model 架構，

以及 DDPM 的流程。並修改 Dataset 以及 Training Loop 來實作。其中 UNet 主要包含了 Encoder, Decoder 和 Embedding block 三個部分。而在 sampling 時，參數 guided\_w 則是決定最後混合的權重。

Training hyper-parameter detail:

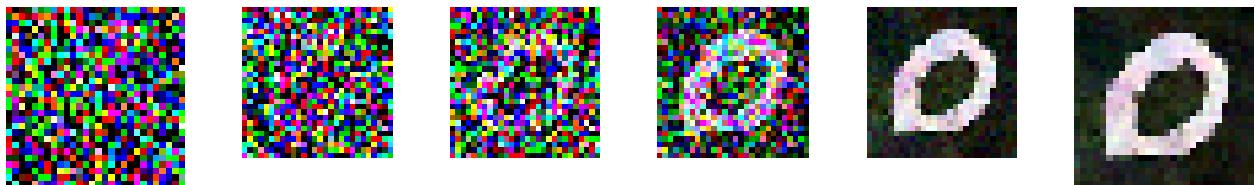
```
# ----- Training -----
n_epoch = 50
batch_size = 512
n_T = 400 # Timestep
n_classes = 10
n_feat = 128
lrate = 1e-4
w = 2.0
```

2. (5%) Please show 10 generated images **for each digit (0-9)** in your report. You can put all 100 outputs in one image with columns indicating different noise inputs and rows indicating different digits.



3. (5%) Visualize total six images in the reverse process of the **first “0”** in your grid in (2) **with different time steps**.

Timestep 由左到右分別是 0, 80, 160, 240, 320, 400。



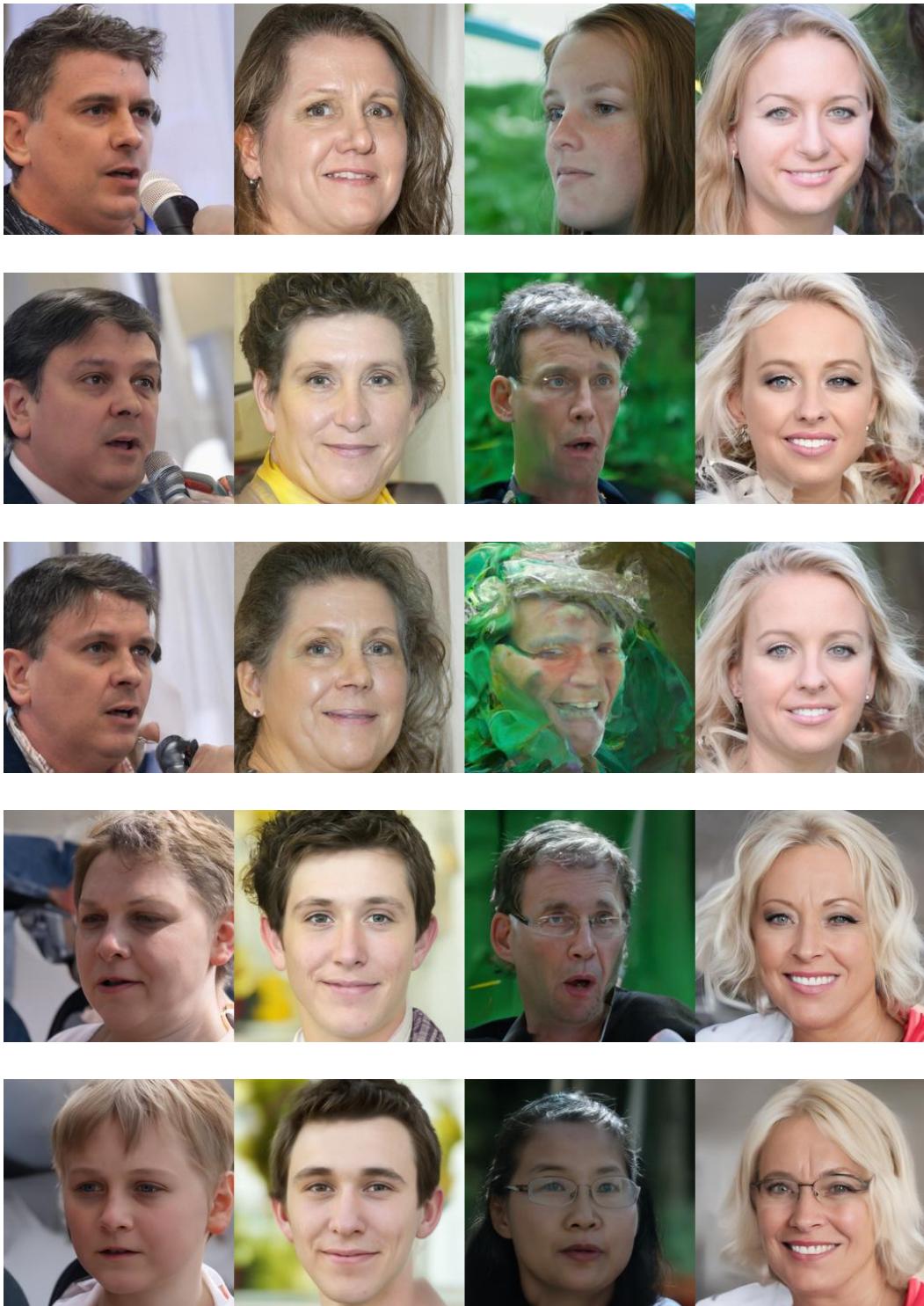
4. (5%) Please discuss what you've observed and learned from implementing conditional diffusion model.
- 原始的 diffusion model 是一種無條件的生成模型,它只透過反向擴散過程生成新資料，但不能明確控制生成內容。
  - 而 conditional diffusion model 通過在生成過程中引入條件資訊，能夠控制和引導圖片生成的結果。例如標籤、文字描述、圖片等,來指導生成過程。這樣可以明確地指定模型應該生成怎樣的結果。
  - 在實作上，我學到了一般 diffusion model 和 conditional diffusion model 之間的差異，以及如何訓練，需要對原始模型進行修改，使其能夠條件化地計算反向擴散中的參數，將條件通過 Embedding 後，與圖片特徵進行拼接，讓模型可以學習條件化的映射 function。

## Problem 2: DDIM (35%)

第二題中我使用了 <https://github.com/xiaohu2015/nngen> 中 ddim sampling 的相關程式碼，並使用助教預先提供的 `UNet.py` 和 `UNet weight` 來生成圖片。

- (7.5%) Please generate face images of noise **00.pt ~ 03.pt with different eta** in one grid. Report and explain your observation in this experiment.
  - DDIM 中， $\eta$  控制著擴散過程中的隨機性程度，它調節了模型在從 noise data 恢復出原始數據的過程中，每一步恢復的確定性和隨機性的比例。
  - 在 DDIM 中， $\eta$  的範圍通常設定在 0 到 1 之間：
    - 當  $\eta=0$  時，生成過程是完全確定性的。這意味著從相同的起始 noise data 出發，會重複生成相同的圖像，無論進行多少次嘗試。
    - 當  $\eta$  增加時，生成過程引入了隨機性。每個步驟都會在去噪過程中加入一定量的隨機 noise data，這允許生成過程探索更多的可能性，從而產生更多樣化的結果。
    - 當  $\eta=1$  時，生成過程是完全隨機的。這就像是在原始的去噪擴散概率模型 (DDPM) 中每步都進行隨機抽樣一樣。
  - 我的結果中：
    - 隨著  $\eta$  值的增加，從上到下的圖像變得越來越不同，顯示出更多的變化和創造性。

- 高  $\eta$  值導致圖像失真或者在某些情況下，完全不同於原始意圖的生成結果。
- 圖片中的五行對應於五個不同的  $\eta$  值，從上到下依次為 0、0.25、0.5、0.75 和 1。



2. (7.5%) Please generate the face images of the interpolation of noise **00.pt ~ 01.pt**. The interpolation formula is **spherical linear interpolation**, which is also known as **slerp**.

$$\mathbf{x}_T^{(\alpha)} = \frac{\sin((1-\alpha)\theta)}{\sin(\theta)} \mathbf{x}_T^{(0)} + \frac{\sin(\alpha\theta)}{\sin(\theta)} \mathbf{x}_T^{(1)}$$

where  $\theta = \arccos \left( \frac{(\mathbf{x}_T^{(0)})^\top \mathbf{x}_T^{(1)}}{\|\mathbf{x}_T^{(0)}\| \|\mathbf{x}_T^{(1)}\|} \right)$ . These values are used to produce DDIM samples.

in this case,  $\alpha = \{0.0, 0.1, 0.2, \dots, 1.0\}$ .

What will happen if we simply use linear interpolation? Explain and report your observation.



如果使用 linear interpolation，代表直接對應到源點和目標點之間的直線路徑。Linear interpolation 在這樣的高維空間中可能不會保持點的規範，導致插值過程中臉部特徵的比例和位置出現比較奇怪的變化。尤其是當兩個點之間的直線路徑不完全位於數據分佈的支持範圍內時，這種變化會更加顯著。

從我的生成結果來看，slerp 的每一步的過渡都相當平滑，這表明 slerp 為生成的 DDIM 樣本提供了很好的過渡方式。當我們改成直接使用線性插值，可以圖像中的中間圖像顯示非自然過渡，如特徵的不連續變化、不對稱性、或者在特徵空間中不存在的臉部組合。

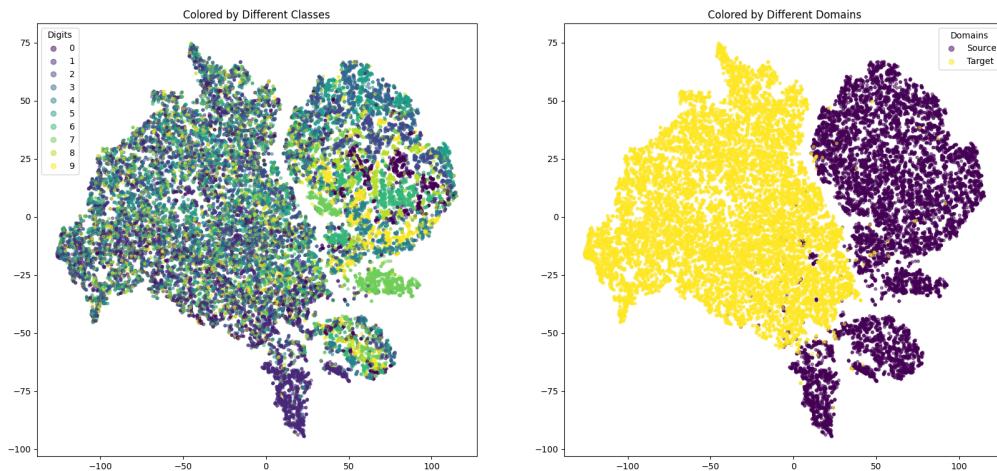
## Problem 3: Report (23%)

1. (10%) Please create and fill the table with the following format **in your report**:

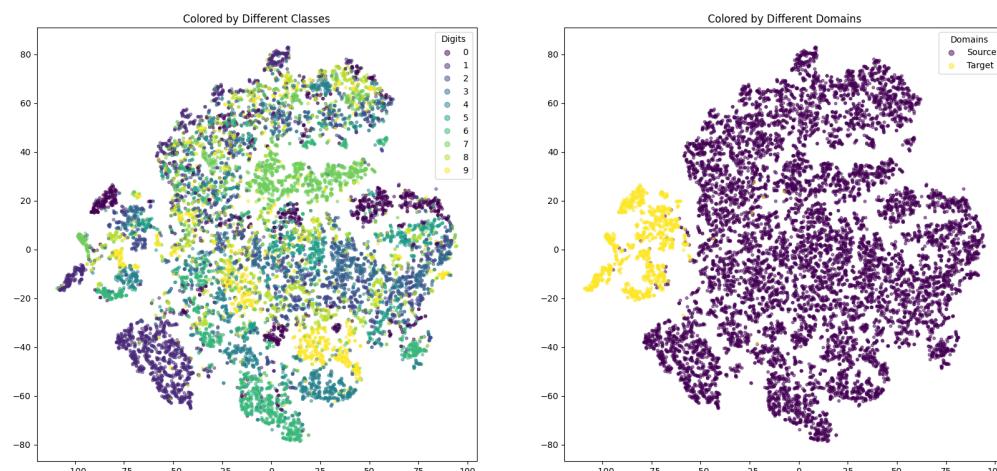
	MNIST-M → SVHN	MNIST-M → USPS
Trained on source	0.31786	0.78305
Adaptation (DANN)	0.45606	0.85765
Trained on target	0.93273	0.98963

2. (10%) Please visualize the latent space (**output of CNN layers**) of DANN by mapping the **validation** images to 2D space with t-SNE. For each scenario, you need to plot two figures which are colored **by digit class (0-9)** and **by domain**, respectively.

## MNISTM → SVHN



## MNISTM → USPS



3. (3%) Please describe the implementation details of your model and discuss what you've observed and learned from implementing DANN.

我的模型架構如下（參考 ChatGPT 撰寫的程式碼）：

```
class DomainClassifier(nn.Module):
    def __init__(self):
        super(DomainClassifier, self).__init__()
        self.layer = nn.Sequential(
```

```

        nn.Linear(256, 256), # 512 is the output size from resnet18's penultimate layer
        nn.ReLU(),
        nn.Linear(256, 2)      # 2 outputs for source and target domains
    )

class DANN(nn.Module):
    def __init__(self):
        super(DANN, self).__init__()
        # self.resnet18 = models.resnet34(pretrained=False)
        # Remove the last FC layer for customization
        # self.backbone = nn.Sequential(*list(self.resnet18.children())[:-1])
        self.backbone = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.Dropout2d(),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.Dropout2d(),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),

            nn.Conv2d(256, 256, kernel_size=3),
        )
        self.grl = GradientReversalLayer(alpha=1.0)

        self.classifier = nn.Sequential(
            nn.Linear(256, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),

            nn.Linear(256, 128),
            nn.BatchNorm1d(128),
            nn.ReLU(),

            nn.Linear(128, 10) # 10 classes for digits
        )
        self.domain_classifier = DomainClassifier()

```

Training hyper-parameter 如下：

```

"""# Configurations"""
# Initialize a model, and put it on the device specified.
model = DANN().to(device)
# The number of batch size.
batch_size = 512

```

```
# The number of training epochs.  
n_epochs = 100  
# If no improvement in 'patience' epochs, early stop.  
patience = 20  
# For the classification task, we use cross-entropy as the measurement of performance.  
label_loss = nn.CrossEntropyLoss()  
domain_loss = nn.BCEWithLogitsLoss()  
# Initialize optimizer, you may fine-tune some hyperparameters such as learning rate on your own.  
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-4, weight_decay=1e-5)  
# Learning rate scheduler  
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)
```

且訓練 MNISTM → USPS 和 MNISTM → SVHN 都使用相同的模型架構和訓練參數。

## Reference

1. ChatGPT 幫忙撰寫程式碼 (Problem 1~3)
2. [https://github.com/TeaPearce/Conditional\\_Diffusion\\_MNIST/tree/main](https://github.com/TeaPearce/Conditional_Diffusion_MNIST/tree/main)
3. <https://github.com/xiaohu2015/nngen>
4. DANN 參考：[https://github.com/fungtion/DANN\\_py3](https://github.com/fungtion/DANN_py3)
5. Training / Validation/ Inference 相關的部分則參考了：  
**HUNG-YI LEE (李宏毅) ML Source code** <https://speech.ee.ntu.edu.tw/~hylee/ml/2023-spring.php>
6. Discussion: R11521701, R10625016