DLCV

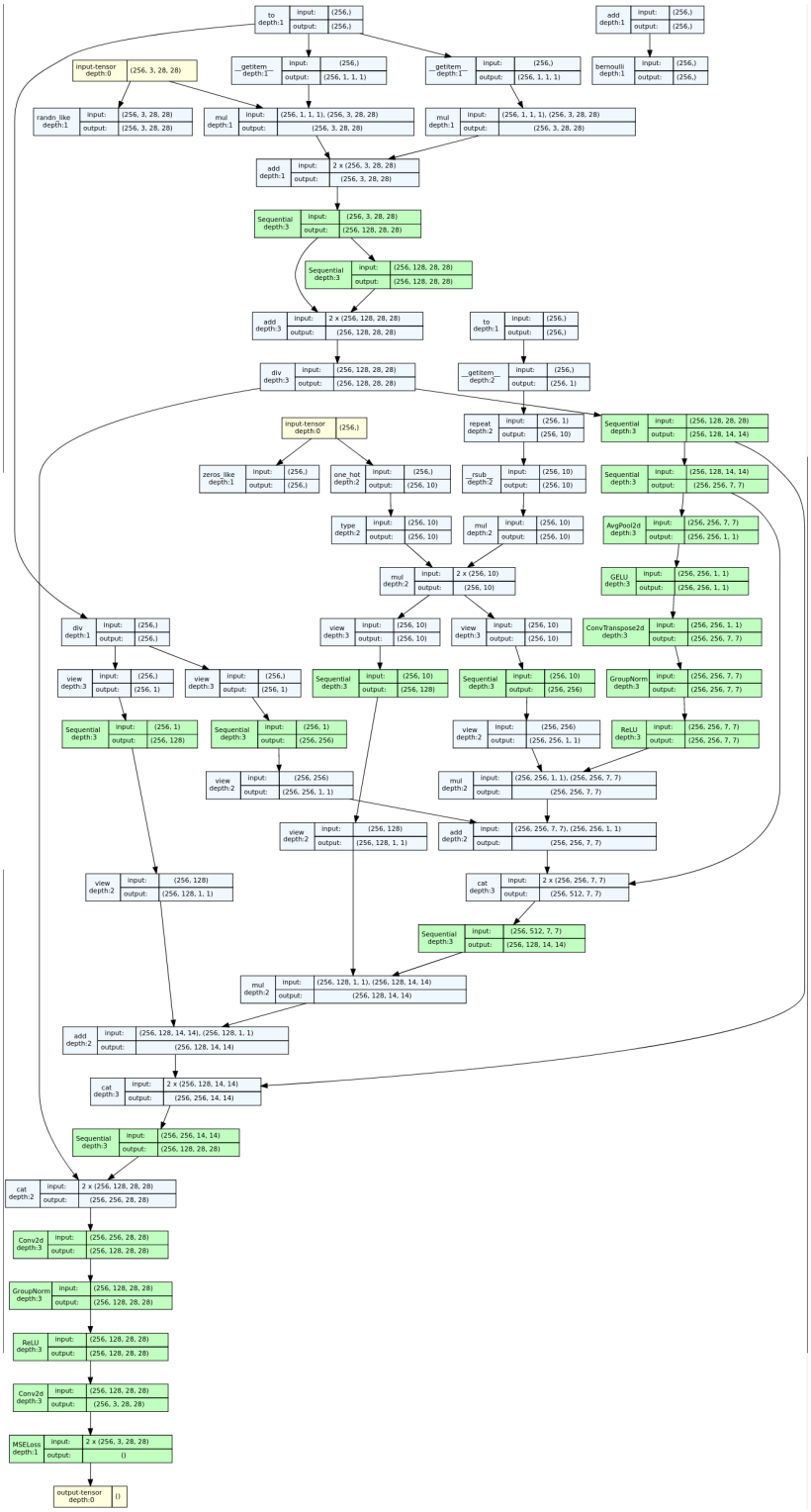
# Homework 2

R12922147 魏子翔

**Problem 1.**

1. Follow the Github Example to draw your model architecture and describe your implementation details.

   I modified the structure of diffusion process from TeaPearce/Conditional_Diffusion_MNIST



with time step = 500

epochs = 20

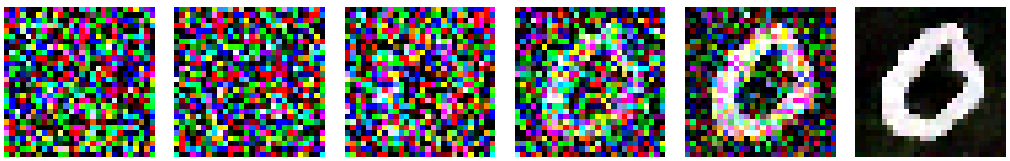
batch size = 256

learning rate = 1e-4

optimizer = Adam

2. 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. Visualize total six images in the reverse process of the first "0" in your grid in (2) with different time steps.



(a) t=0 (b) t=100 (c) t=200 (d) t=300 (e) t=400 (f) t=500

4. Please discuss what you've observed and learned from implementing conditional diffusion model.

   Although I've learned about diffusion models for a few times in other class, this is still the first time for me to see the detailed structures of the code implementation. Also, the concept of adding the class condition embedding into the time embedding is a informative way for me to across from DDPM to conditional DDPM. I've learned a lot from this experience.

**Problem 2.**

1. Please generate face images of noise 00.pt 03.pt with different eta in one grid. Report and explain your observation in this experiment.

   The rows of the following image are $\eta = 0, 0.25, 0.5, 0.75, 1$, respectively, and the columns of the image are noises from 00.pt to 03.pt.

   When $\eta = 0$, the generated images are very closed to the given ground truth images. With $\eta$ is getting larger, the randomness of the generated images are also getting more obvious. Until $\eta = 1$, the noises are totally randomly generated so I could not find any relationship between the ground truth and the results.

2. Please generate the face images of the interpolation of noise 00.pt 01.pt.

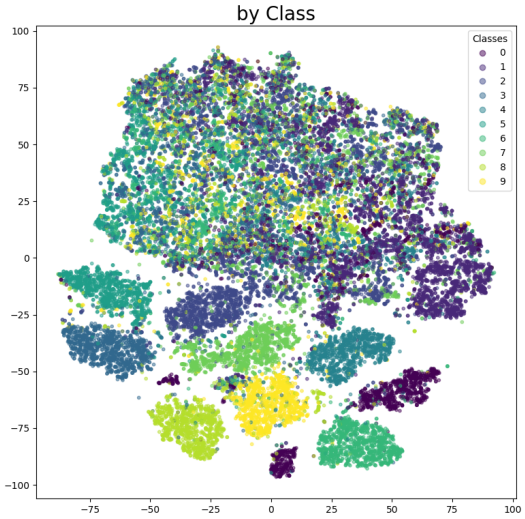
   The upper one is slerp interpolation, and the lower one is linear interpolation. Since the input noise is a 3-dimension data for all pixels, using slerp interpolation is the better way to deal with the high dimensional data, while the linear interpolation may destroy the relationship between the dimensions.

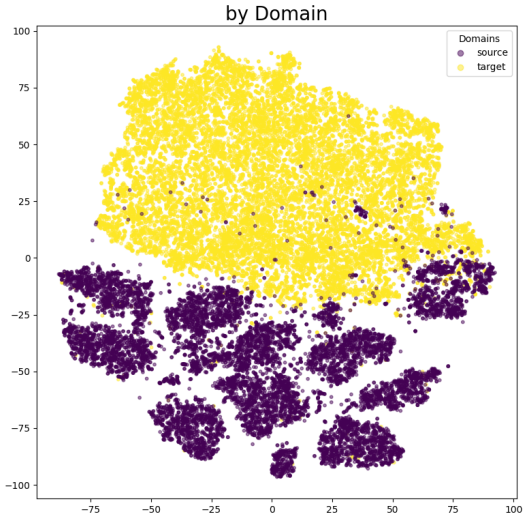
**Problem 3.**

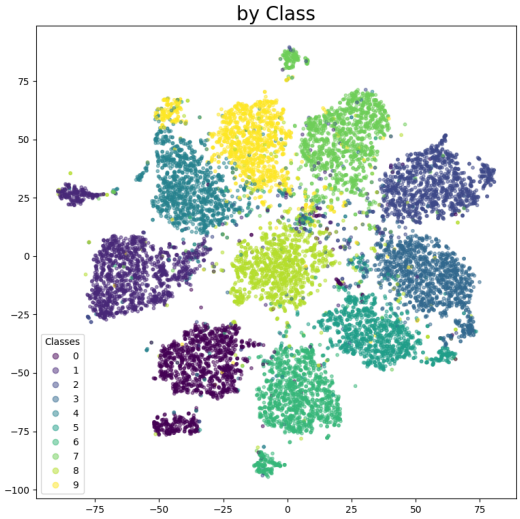
1. Please create and fill the table with the following format in your report:

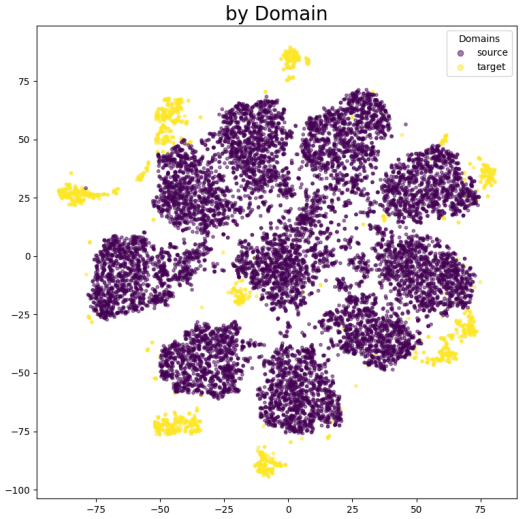
| | MNIST-M → SVHN | MNIST-M → USPS |
|---|---|---|
| Trained on source | 0.19443 | 0.25332 |
| Adaptation (DANN) | 0.50305 | 0.8918 |
| Trained on target | 0.90155 | 0.97804 |

2. 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.




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

   I modified the structure of DANN model from NaJaeMin92/pytorch_DANN

with the following model structures:

```
Extractor(
  (extractor): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Dropout2d(p=0.5, inplace=False)
    (7): ReLU()
    (8): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  )
)
Classifier(
  (classifier): Sequential(
    (0): Linear(in_features=128, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=1024, out_features=256, bias=True)
    (4): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): Linear(in_features=256, out_features=10, bias=True)
  )
)
Discriminator(
  (discriminator): Sequential(
    (0): Linear(in_features=128, out_features=256, bias=True)
    (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=256, out_features=2, bias=True)
  )
)
```

Also, the training details are as follows: epochs = 100

batch size = 512

learning rate = 0.1

transforms = normalize to mean and std of the training set

optimizer = SGD

loss function for class and domain = cross entropy loss

lr scheduling = $\frac{\mu_0}{(1+\alpha \cdot p)^\beta}$ with $\alpha = 10, \beta = 0.75$

For the model trained on the source set and validation on the target set, the performance of both target set are poor. However, using DANN method without any target domain labels, the accuracy of SVHN dataset can reach 50% and the USPS dataset can even get 89%, only 8% from the performance of training on target data. It is an amazing result on transfer learning.