DLCV hw2

B10901041 林新晨

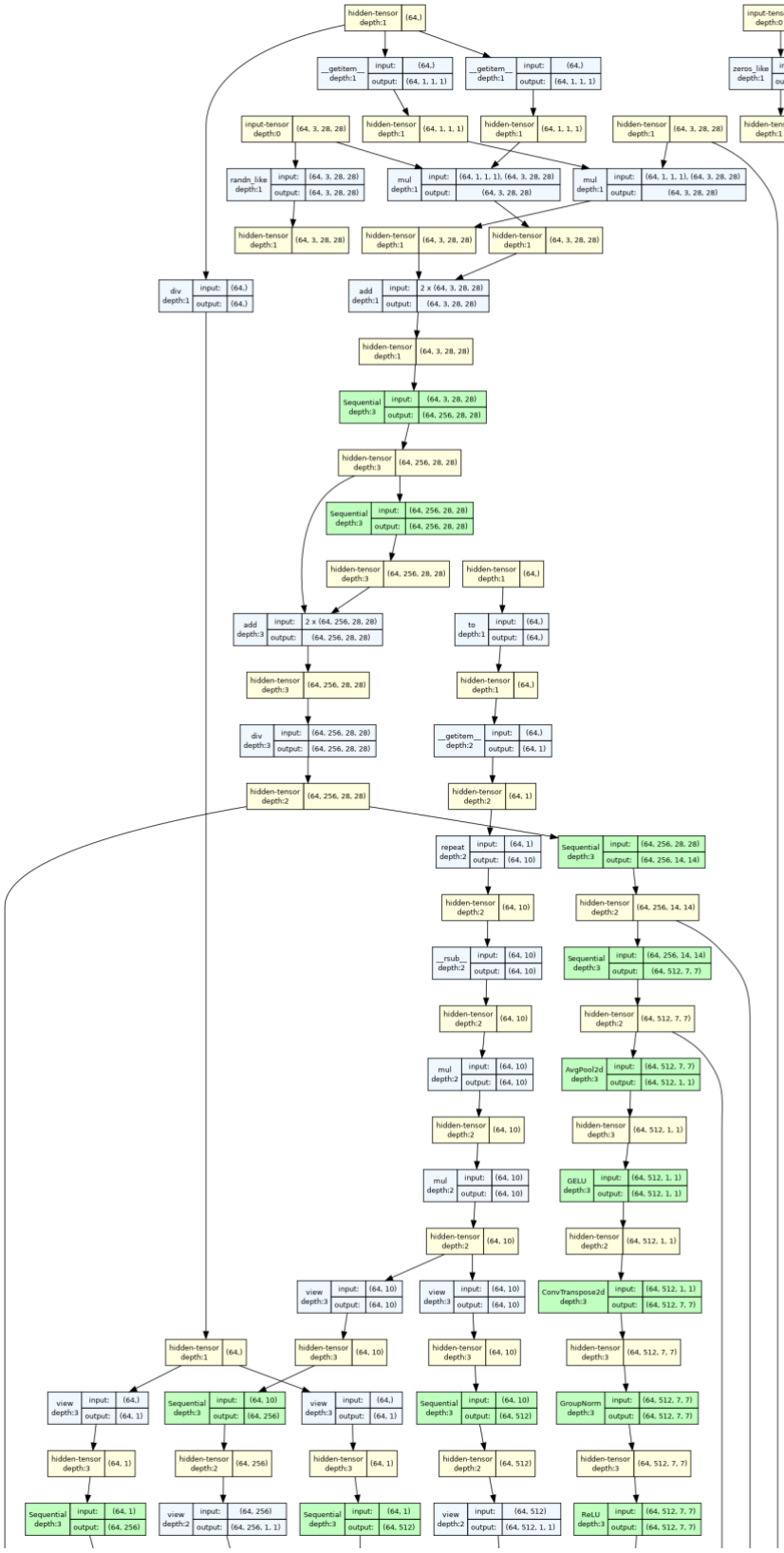
Problem 1

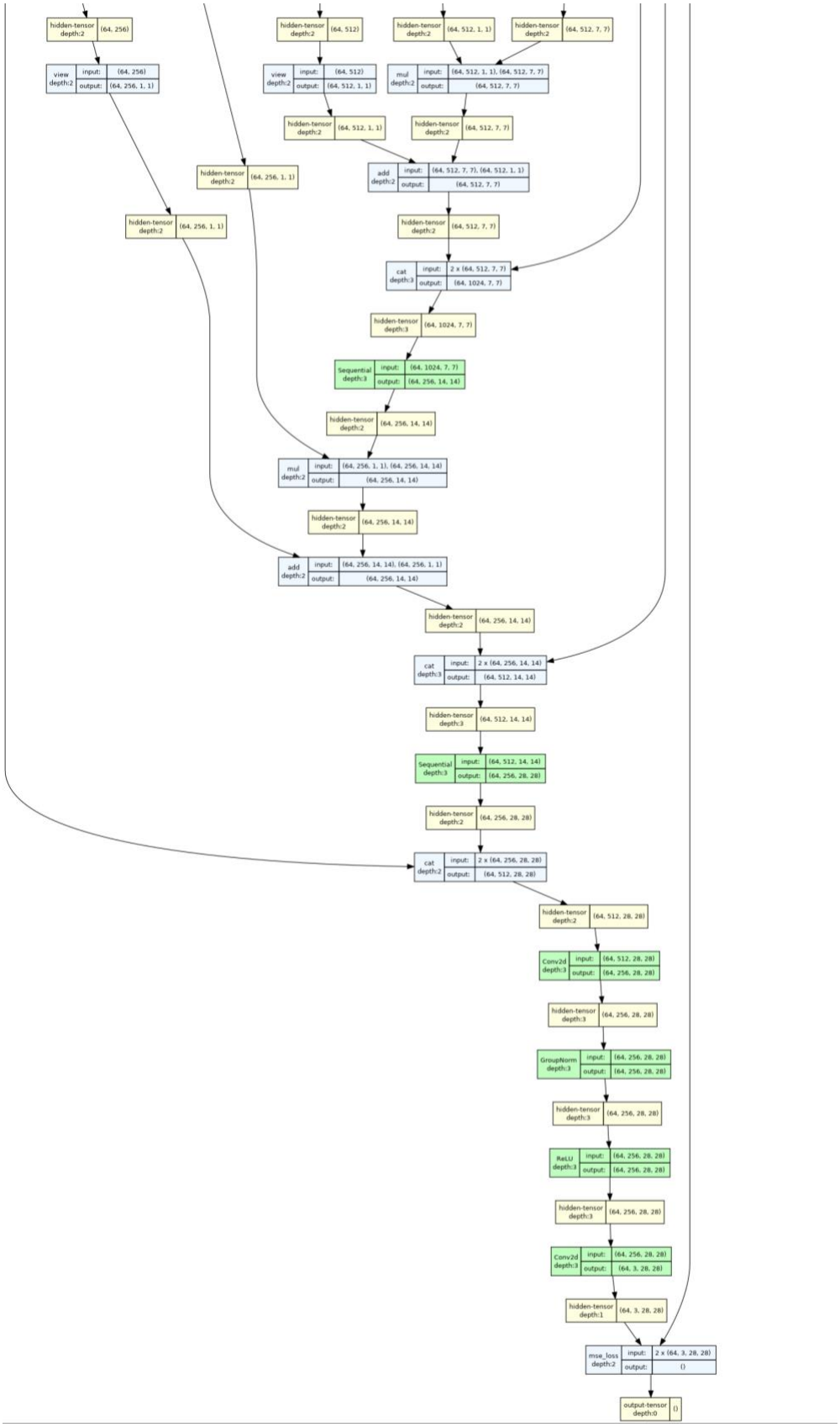
1. Model structure

Reference:

https://github.com/cloneofsimo/minDiffusion/blob/master/mindiffusion/unet.py

https://github.com/TeaPearce/Conditional_Diffusion_MNIST/blob/main/script.py





The model is based on https://github.com/TeaPearce/Conditional_Diffusion_MNIST/blob/main/script.py

DDPM: Schedule beta linearly.

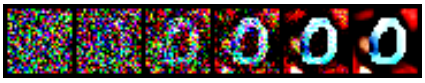
Condition method based on algorithm in https://arxiv.org/abs/2207.12598.

SiLU($f(x)=x\cdot\sigma(x)$) is used in EmbedFC, and it is both self-stabilizing and efficient.

2. **10 generated images for each digit (0-9)**

3. **First "0" in the grid with different time steps.**

   t = 0, 100, 200, 300, 400, 500



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

   The reverse diffusion process is quite time-consuming since we have to refine each sample step by step. The iterative nature of diffusion models means that generating samples takes longer compared to other methods. However, the high quality of sample from conditional diffusion model is undeniable. The generated images can be easily classified by the digit classifier. Also, pictures in different time steps shows clearly how the process works to turn images from noisy to clean, which is quite impressive!

Problem 2

1. **etas: [0.0, 0.25, 0.50, 0.75, 1.0]**

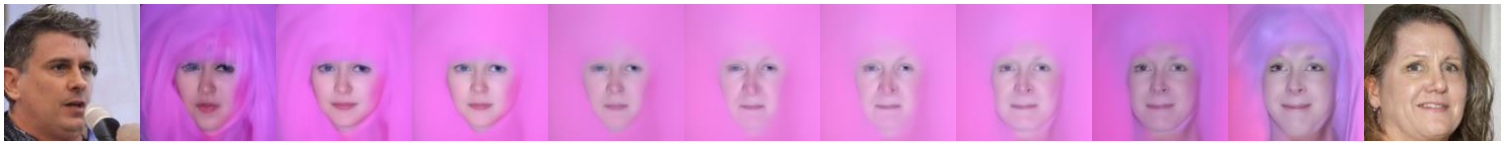
2. Interpolation

Spherical linear interpolation:

Alpha = { 0.0, 0.1, 0.2, …, 1.0 }

If simply use linear interpolation:

Alpha = { 0.0, 0.1, 0.2, …, 1.0 }



The transition between the faces looks more natural and smoother in slerp than in lerp because slerp is designed to work with angles and rotations. Lerp simply computes the average of the pixel values without considering the geometric structure. Since lerp treats the image as a flat array of pixels rather than a structured object with depth and form, the loss of detail is significant.

Reference:

https://github.com/huggingface/diffusers/blob/936cd08488260a9df3548d66628b83bc7f26bd9e/src/diffusers/schedulers/scheduling_ddpm.py

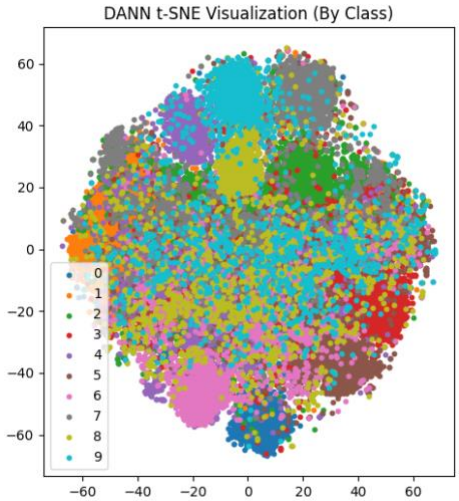
Problem 3

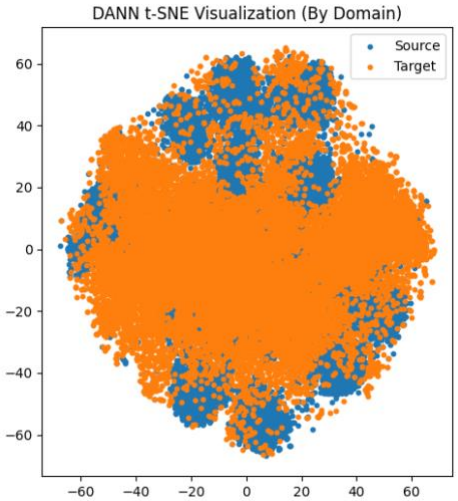
1. n_epoch = 200

	MNIST-M → SVHN	MNIST-M → USPS
Trained on source	0.33903	0.74402
Adaptation (DANN)	0.30250	0.72739
Trained on target	0.82684	0.96543

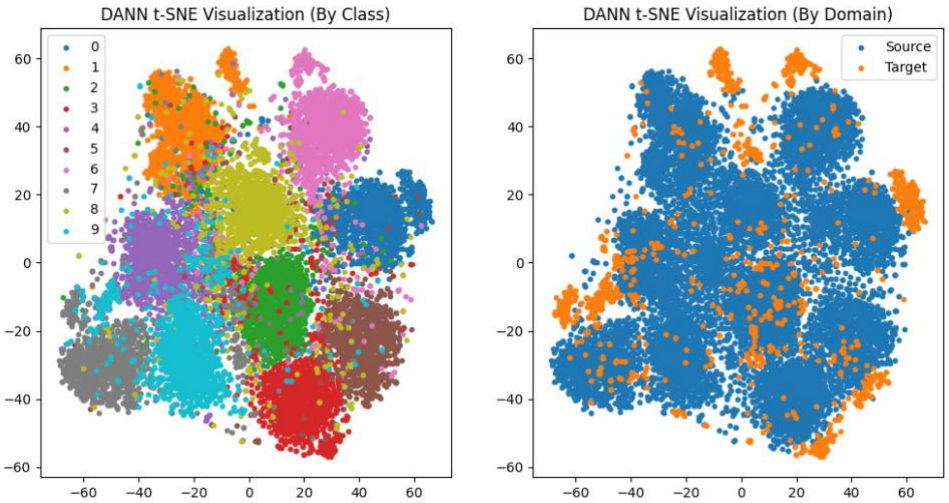
2. t-SNE

target set: svhn





target set: usps



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

Implementation: DANN model based on the model in
https://github.com/NaJaeMin92/pytorch_DANN/blob/master/utils.py
The model consists of feature extractor, class classifier, and domain classifier.
The feature extractor has 2 conv layers; class classifier has 3 fc layers; domain classifier has 2 fc layers.
Normalized all the data by calculating the mean and std respectively on all three datasets.

DANN offers a powerful framework for domain adaptation. From the t-SNE plot in the above images, we can see that the feature extractor learns to confuse the domain classifier while it still performs well on the main digit classification task. Also, the adaptation of MNIST-M → USPS performs better than MNIST-M → SVHN. I suppose it is because SVHN is composed of digit photos, while both MNIST-M and USPS data comprise handwritten digits. The similarity between MNIST-M and USPS may lead to the better adaptation result.

Reference:
https://github.com/fungtion/DANN/tree/master
https://github.com/NaJaeMin92/pytorch_DANN