

DLCV HW2 Report

tags: DLCV

Course	Student ID	Name	Date
2022 Fall NTU DLCV	R10943109	Shiuan-Yun Ding	2022-10-30

HackMD Link: <https://hackmd.io/MwBnUqJCQzSI9bAJJTtnVQ>

Problem 1: GAN

Metric	Simple Baseline	Strong Baseline	Model A	Model B
FID	30.00 (2.5%)	27.00 (25%)	26.82%	24.19%
Face Recognition	85.00% (2.5%)	90.00% (2.5%)	89.10%	90.30%

(5%) Model Architectures

Model A

- Basic DCGAN copied from [PyTorch DCGAN TUTORIAL](#)

Discriminator	<pre>Discriminator((main): Sequential((0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (1): LeakyReLU(negative_slope=0.2, inplace=True) (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (4): LeakyReLU(negative_slope=0.2, inplace=True) (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (7): LeakyReLU(negative_slope=0.2, inplace=True) (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (10): LeakyReLU(negative_slope=0.2, inplace=True) (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False) (12): Sigmoid()))</pre>
Generator	<pre>Generator((main): Sequential((0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False) (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): ReLU(inplace=True) (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (5): ReLU(inplace=True) (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (8): ReLU(inplace=True) (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (11): ReLU(inplace=True) (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (13): Tanh()))</pre>

Model B

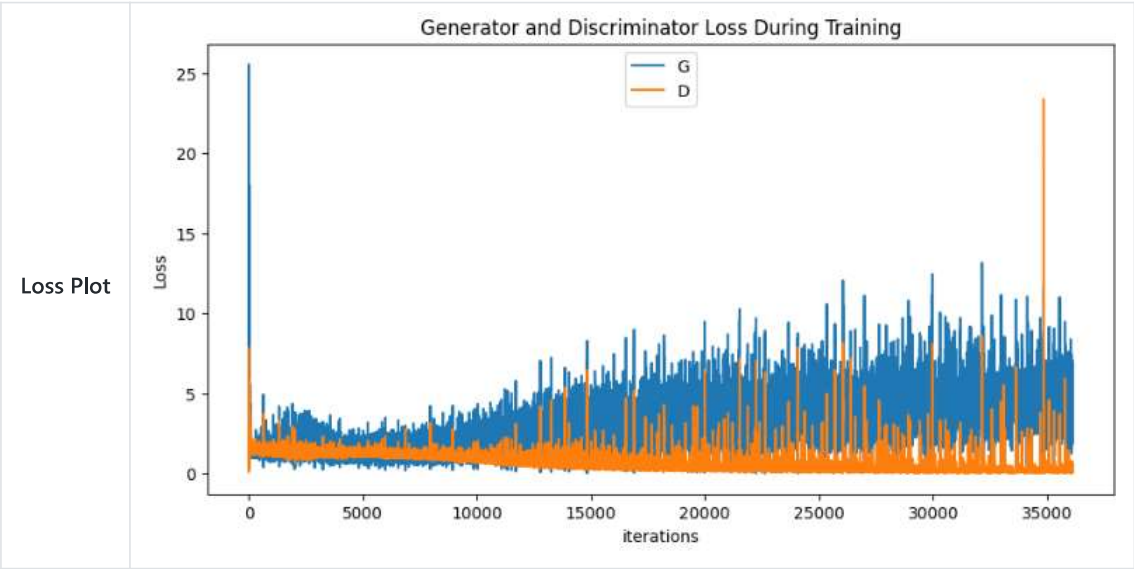
- Difference : I doubled the number of channels.

Discriminator	<pre>Discriminator((main): Sequential((0): Conv2d(3, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (1): LeakyReLU(negative_slope=0.2, inplace=True) (2): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (4): LeakyReLU(negative_slope=0.2, inplace=True) (5): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (6): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (7): LeakyReLU(negative_slope=0.2, inplace=True) (8): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (9): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (10): LeakyReLU(negative_slope=0.2, inplace=True) (11): Conv2d(1024, 1, kernel_size=(4, 4), stride=(1, 1), bias=False) (12): Sigmoid())))</pre>
Generator	<pre>Generator((main): Sequential((0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False) (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): ReLU(inplace=True) (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (5): ReLU(inplace=True) (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (8): ReLU(inplace=True) (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (11): ReLU(inplace=True) (12): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (13): Tanh())))</pre>

Implementation Details

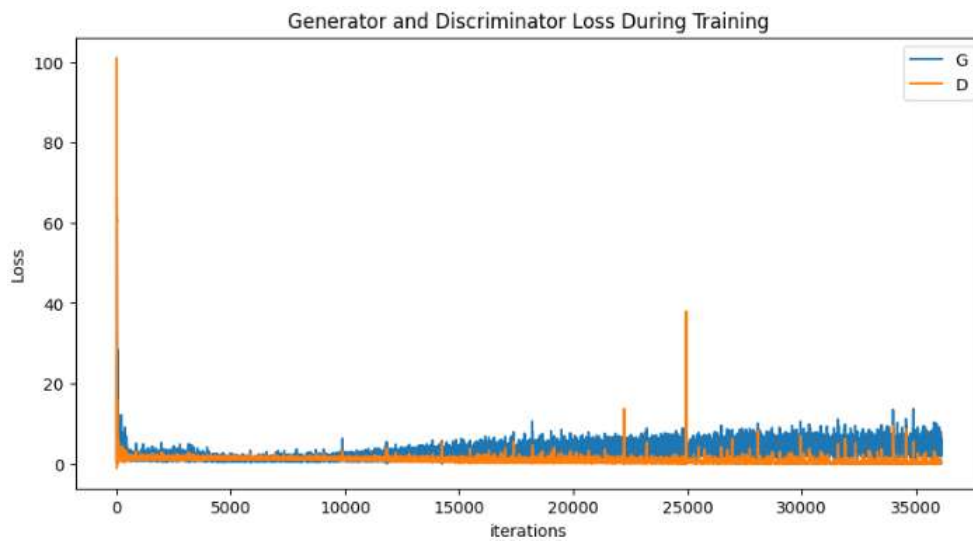
	Optimizer	Learning Rate	Beta1
Discriminator	Adam	0.0005	0.5
Generator	Adam	0.005	0.5

Model A



Model B

Loss Plot



(5%) Generated Images

	Model A	Model B
64		
gif		

(5%) Observed & Learned

- Mode collapse happens when the discriminator is too strong. Tune the learning rates to make sure the discriminator and the generator learn at a competitive speed.
- Increase the number of channels to improve FID.
- Examine the losses of discriminator and generator to find the cause of failure.
- Pytorch dataloader sometimes fails. Change the num_worker to 0 will fix the problem.
- GAN is a metaphysics, sometimes it fails and sometimes it succeeds.



Reference

- https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
- <https://clay-atlas.com/blog/2019/10/24/pytorch-%E6%95%99%E5%AD%B8-image-dcgan-%E5%88%A9%E7%94%A8%E7%94%9F%E6%88%90%E5%B0%8D%E6%8A%97%E7%B6%B2%E8%B7%AF%E7%94%9F%E6%88%90%E5%9C%96%E7%89%87/>
- https://github.com/thtang/DLCV2018SPRING/blob/master/hw4/train/GAN_train.ipynb
- <https://github.com/soumith/ganhacks>

Problem 2: Diffusion Models

Metric	Simple Baseline	Strong Baseline	My Result
Accuracy	80.00%	88.00%	95.70%

(5%) Model Architecture & Implementation Details

Model Architecture

Input Convolution Layer	<pre>(init_conv): ResidualConvBlock((conv1): Sequential((0): Conv2d(3, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)) (conv2): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)))</pre>
Unet Down Layer	<pre>(down1): UnetDown((model): Sequential((0): ResidualConvBlock((conv1): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)) (conv2): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none))) (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False))) (down2): UnetDown((model): Sequential((0): ResidualConvBlock((conv1): Sequential((0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)) (conv2): Sequential((0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none))) (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)))</pre>
Pooling Layer	<pre>(to_vec): Sequential((0): AvgPool2d(kernel_size=7, stride=7, padding=0) (1): GELU(approximate=none))</pre>

Input Convolution Layer	<pre>(init_conv): ResidualConvBlock((conv1): Sequential((0): Conv2d(3, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)) (conv2): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)))</pre>
Fully-Connected Layer	<pre>(timeembed1): EmbedFC((model): Sequential((0): Linear(in_features=1, out_features=256, bias=True) (1): GELU(approximate=none) (2): Linear(in_features=256, out_features=256, bias=True))) (timeembed2): EmbedFC((model): Sequential((0): Linear(in_features=1, out_features=128, bias=True) (1): GELU(approximate=none) (2): Linear(in_features=128, out_features=128, bias=True))) (contextembed1): EmbedFC((model): Sequential((0): Linear(in_features=10, out_features=256, bias=True) (1): GELU(approximate=none) (2): Linear(in_features=256, out_features=256, bias=True))) (contextembed2): EmbedFC((model): Sequential((0): Linear(in_features=10, out_features=128, bias=True) (1): GELU(approximate=none) (2): Linear(in_features=128, out_features=128, bias=True)))</pre>
Unet Up Layer	<pre>(up0): Sequential((0): ConvTranspose2d(256, 256, kernel_size=(7, 7), stride=(7, 7)) (1): GroupNorm(8, 256, eps=1e-05, affine=True) (2): ReLU()) (up1): UnetUp((model): Sequential((0): ConvTranspose2d(512, 128, kernel_size=(2, 2), stride=(2, 2)) (1): ResidualConvBlock((conv1): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)) (conv2): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none))) (2): ResidualConvBlock((conv1): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)) (conv2): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)))))</pre>

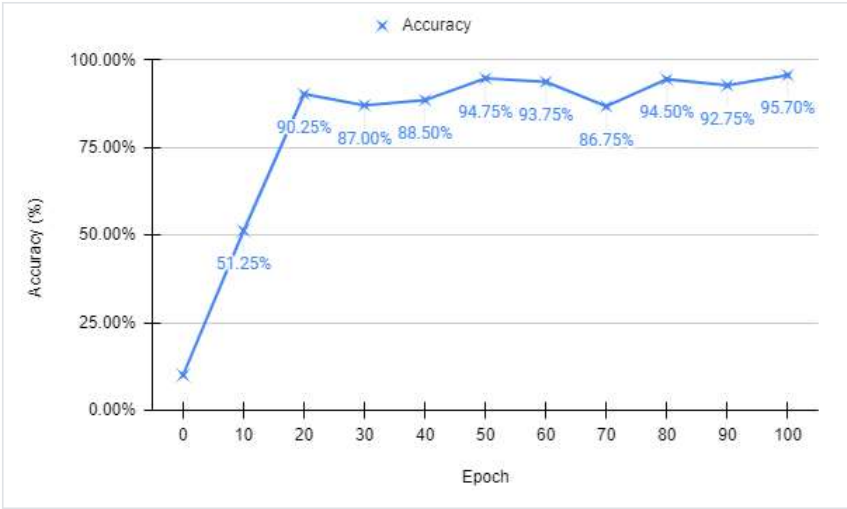
Input Convolution Layer	<pre>(init_conv): ResidualConvBlock((conv1): Sequential((0): Conv2d(3, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)) (conv2): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)))</pre>
	<pre>(up2): UnetUp((model1): Sequential((0): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2)) (1): ResidualConvBlock((conv1): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)) (conv2): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)))) (2): ResidualConvBlock((conv1): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none)) (conv2): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): GELU(approximate=none))))</pre>
Output Convolution Layer	<pre>(out): Sequential((0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): GroupNorm(8, 128, eps=1e-05, affine=True) (2): ReLU() (3): Conv2d(128, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)))</pre>

Implementation Details



- | Optimizer | Learning Rate | Beta1 |
|-----------|---------------|---------|
| Adam | 0.0001 | default |

- | Batch Size | Epochs |
|------------|--------|
| 256 | 100 |

- Accuracy Grow



(5%) 10 x 10 Generated Digit Images

My Generated Images	MNIST-M
	

(5%) Reverse Process

timestep	t = 0	t = 5	t = 20	t = 80	t = 160	t = 400
Image						

(5%) Observed & Learned

- Basic concepts and implementation of DDPM.
- Detail construction of U-Nets
- Compared to GAN, diffusion model is quite simple (only has one model) and successful.



Reference

- https://github.com/TeaPearce/Conditional_Diffusion_MNIST
- https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/annotated_diffusion.ipynb
- <https://zhuanlan.zhihu.com/p/572161541>

Problem 3: DANN

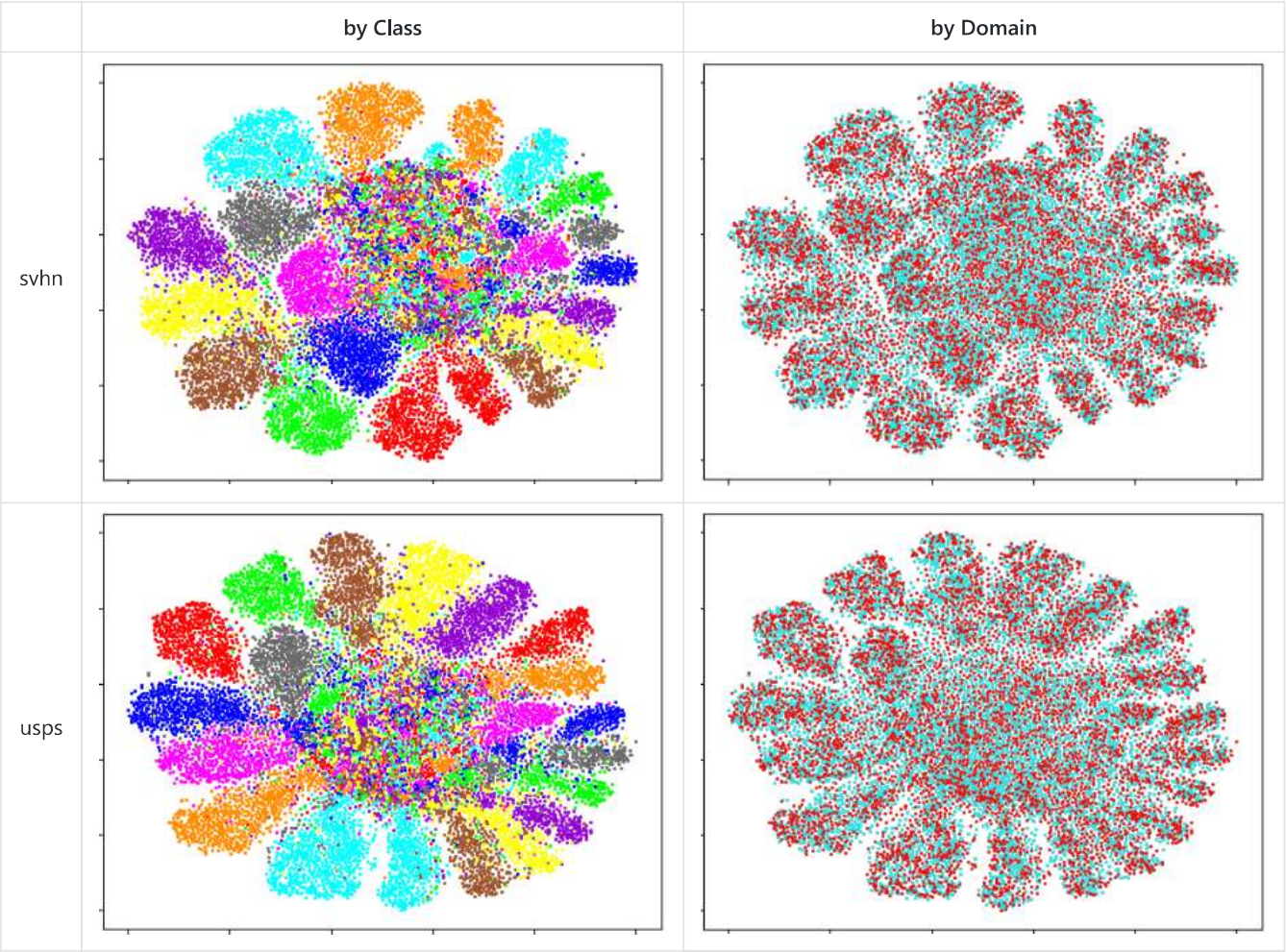
	MNIST-M -> SVHN	MNIST-M -> USPS
Baseline	40%	76%
My Results	40.0579%	76.8817%

(5%) Table

	MNIST-M -> MNIST-M	MNIST-M -> SVHN	MNIST-M -> USPS
Trained on Source (Lower Bound)	90.0893%	35.5888%	73.3871%
Adaptation (DANN)	x	40.0579%	76.8817%
Trained on Target (Upper Bound)	90.0893%	90.6779%	98.3871%

Source \ Target	MNIST-M	SVHN	USPS
MNIST-M	90.0893%	35.5888%	73.3871%
SVHN	54.0804%	90.6779%	65.7258%
USPS	32.2679%	19.3044%	98.3871%

(8%) t-SNE



(10%) Details & Learned & Observed

Model Architectures

Model	Architecture
-------	--------------

Model	Architecture
Feature Extractor (Encoder)	<pre> FeatureExtractor((conv): Sequential((0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): ReLU() (3): Dropout2d(p=0.5, inplace=False) (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) (5): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (6): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (7): ReLU() (8): Dropout2d(p=0.5, inplace=False) (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) (10): Flatten(start_dim=1, end_dim=-1) (11): Linear(in_features=3136, out_features=1024, bias=True) (12): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (13): ReLU())) </pre>
Label Predictor (Decoder)	<pre> LabelPredictor((fc): Sequential((0): Linear(in_features=1024, out_features=50, bias=True) (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): ReLU() (3): Linear(in_features=50, out_features=10, bias=True))) </pre>
Domain Classifier (Discriminator)	<pre> DomainClassifier((fc): Sequential((0): Linear(in_features=1024, out_features=50, bias=True) (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): ReLU() (3): Linear(in_features=50, out_features=1, bias=True))) </pre>

Upper & Lower Bound Implementation Details

Optimizer	Criterion	Batch Size	Epochs
Adam	CrossEntropyLoss	256	25

DANN Implementation Details

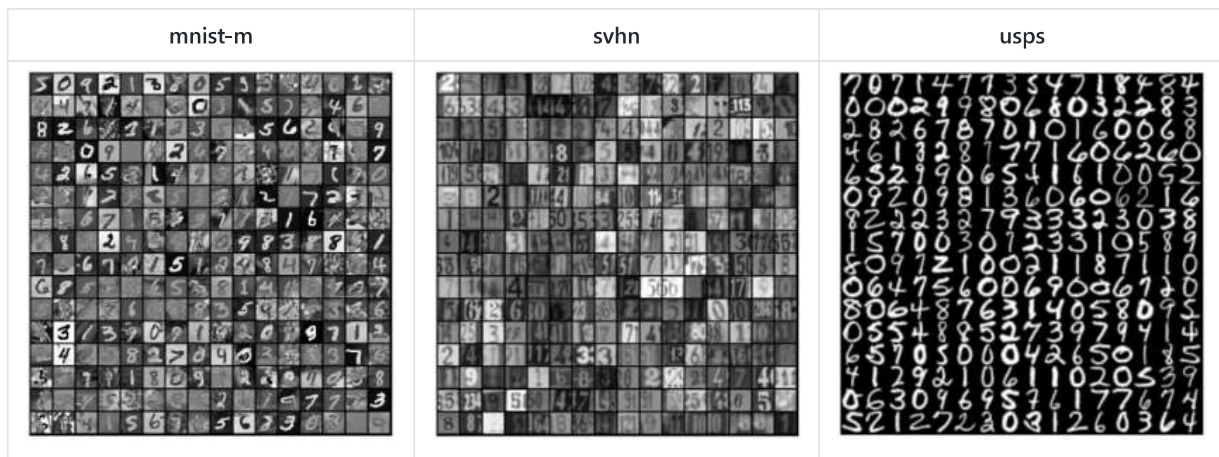
	Optimizer	Learning Rate	Betas
Feature Extractor	Adam	default	default
Label Predictor	Adam	default	default
Domain Classifier	Adam	default	default

Batch Size	Best Epoch of svhn	Best Epoch of usps
256	24	2

Data Preprocessing

- Turn all into gray scale images.

mnist-m	svhn	usps
---------	------	------



Observed & Learned

- More training cannot enhance the performance. In fact, the best accuracy of DANN on MNIST-M to SVHN happens on the second epoch.
- By t-SNE, we can see that there is still a mess in the middle. But the colors outer space by classes are very separated.

Reference

- https://blog.csdn.net/qq_41858347/article/details/105400606