

Deep Learning for Computer Vision HW3 Report

資工所碩二 R08922086 蔡承運

Problem 1: Variational Auto-Encoder

(1) VAE model architecture

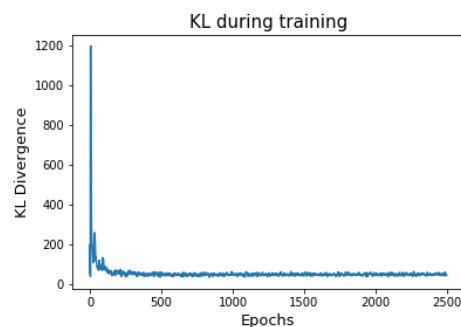
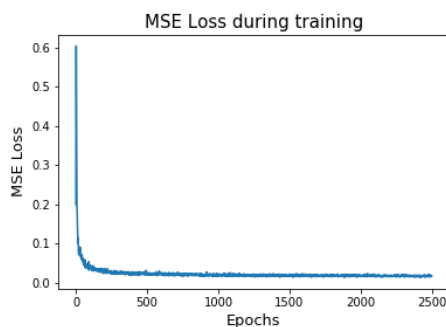
```
VAE(  
  (encode_block): Sequential(  
    (0): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): LeakyReLU(negative_slope=0.01, inplace=True)  
    (3): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (5): LeakyReLU(negative_slope=0.01, inplace=True)  
    (6): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (8): LeakyReLU(negative_slope=0.01, inplace=True)  
    (9): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (10): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (11): LeakyReLU(negative_slope=0.01, inplace=True)  
  )  
  
  (decode_block): Sequential(  
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): LeakyReLU(negative_slope=0.01, inplace=True)  
    (3): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (5): LeakyReLU(negative_slope=0.01, inplace=True)  
    (6): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (7): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (8): LeakyReLU(negative_slope=0.01, inplace=True)  
    (9): ConvTranspose2d(32, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
  )  
  (fc_mu): Linear(in_features=4096, out_features=2048, bias=True)  
  (fc_logvar): Linear(in_features=4096, out_features=2048, bias=True)  
  (fc_decode): Linear(in_features=2048, out_features=4096, bias=True)  
)
```

Training settings:

- Epochs: 50
- Batch size: 128
- Optimizer: Adam
- Learning rate: 0.001
- Weight for KL Divergence: 0.0001
- No data augmentation and learning rate scheduling is used

Other hyper-parameters follow the default values in PyTorch

(2) Learning curve (both reconstruction loss and KL divergence)



(3) Randomly choose 10 testing images and get the reconstructed images

I simply pick the first 10 images in the test dataset and pass them into VAE to get the reconstructed images and their corresponding MSE.

(Images are normalized to [0, 1], and the loss is computed accordingly.)






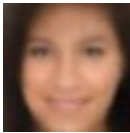
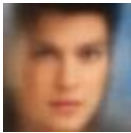

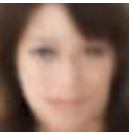






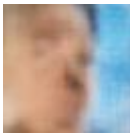
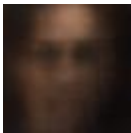

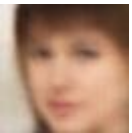

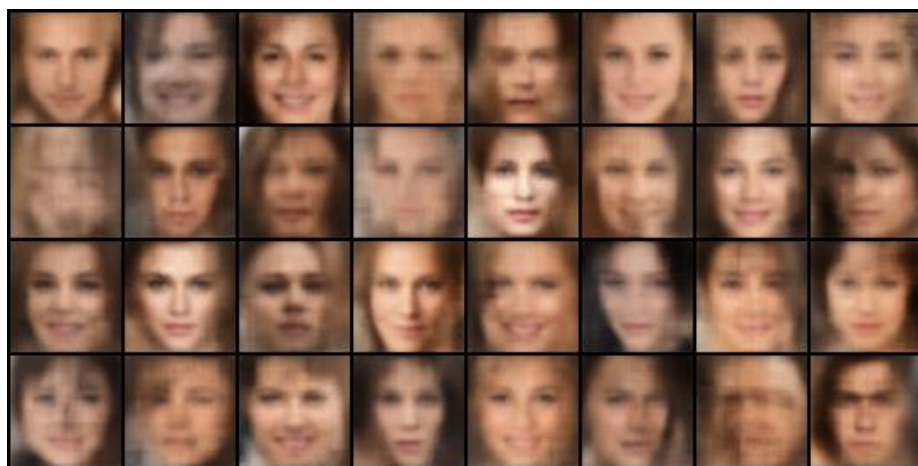
Image ID	40000	40001	40002	40003	40004
Testing Img.					
Recon. Img.					
MSE	0.009510	0.000128	0.000681	0.012881	0.009892

Image ID	40005	40006	40007	40008	40009
Testing Img.					
Recon. Img.					
MSE	0.002279	0.017329	0.007697	0.002114	0.008815

(4) Randomly sample 32 latent vectors from normal distribution and plot the reconstruct the images from the latent vectors.

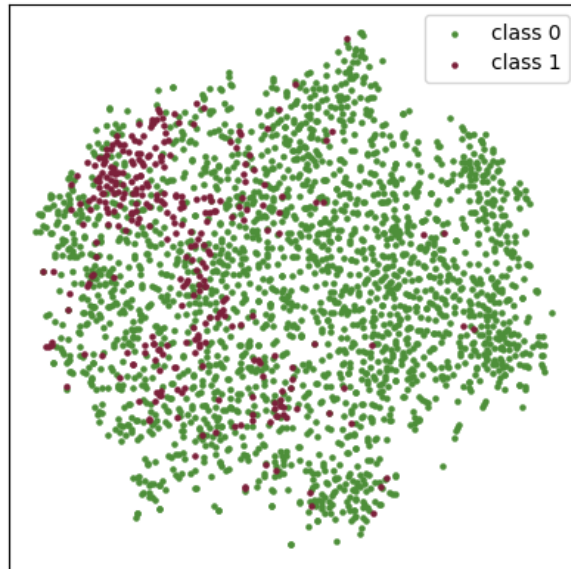


(5) Analyze the latent space of VAE using t-SNE

Attribute: Blond_Hair

Class 0: has no blond hair

Class 1: has blond hair



(6) Discuss what you have observed and learned from VAE

Variational auto-encoder learns to encode images into mean and variance vectors and reconstruct images from the latent space, parametrized by mean and variance vectors. The reconstruction results demonstrate its effectiveness and show the strong reconstruction ability in terms of MSE. However, even though the MSE is quite low, the contour of the reconstructed images is blurred and are not clear enough.

Specifically, during sampling from the latent space, we are going to sample vectors having Gaussian distribution of some specific mean and variance. However, sampling from this kind of distribution cannot be differentiated in back propagation in training. We have to apply “**re-parametrization trick**”, sampling vectors from normal distribution (mean=0, variance=1) and multiplying by the designated mean and the variance. This allows us to train our model in an end-to-end fashion and update all the parameters at the same time.

Problem 2: Generated Adversarial Network

(1) Print the model architecture of GAN (DCGAN)

```
Generator(  
  (main module): Sequential(  
    (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1))  
    (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))  
    (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))  
    (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))  
    (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))  
  )  
  (output): Tanh()  
)
```

```
Discriminator(  
  (main module): Sequential(  
    (0): Conv2d(3, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): LeakyReLU(negative_slope=0.2, inplace=True)  
    (3): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (5): LeakyReLU(negative_slope=0.2, inplace=True)  
    (6): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (7): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (8): LeakyReLU(negative_slope=0.2, inplace=True)  
    (9): Conv2d(1024, 2048, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (10): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (11): LeakyReLU(negative_slope=0.2, inplace=True)  
  )  
  (output): Sequential(  
    (0): Conv2d(2048, 1, kernel_size=(4, 4), stride=(1, 1))  
  )  
)
```

Training settings:

- Epochs: 150
- Batch size: 128
- Optimizer: Adam (beta1=0.5, beta2=0.999)
- Learning rate: **0.002**
- No data augmentation and learning rate scheduling is used
- Other hyper-parameters follow the default values in PyTorch.

The model architecture follows the design in DCGAN paper.

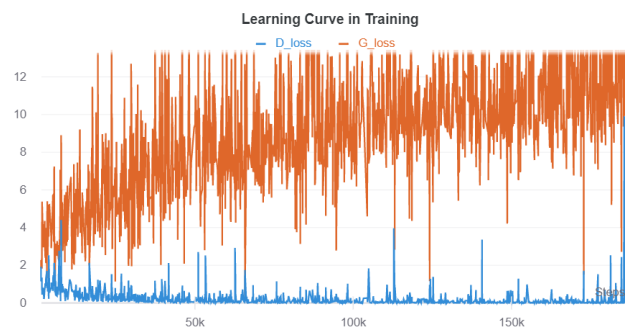
(2) Sample 32 vectors from Normal distribution and generate by the Generator



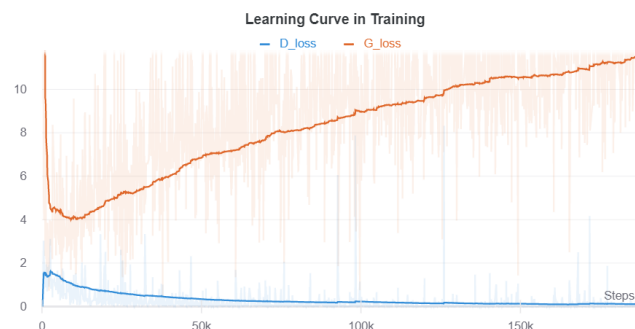
(3) Discuss what you have observed and learned from GAN

Regarding the implementation of the GAN, the architecture is the most important factors in successful training of GAN. Following the suggestion by the author of DCGAN (and WGAN), I add BatchNorm in both the generator and discriminator. If they were not added, the model is surprisingly hard to being successfully trained. Also, one should use LeakyReLU in every block of the discriminator and use Tanh in the last layer of the generator.

With respect to the training of DCGAN, the loss of both generator and discriminator is incredibly unstable, as shown in the below figure.



After a smoothing factor of 0.99 is applied, we can then finally see the progress of the learning.



Also, it is noteworthy that during training, the loss of generator (G_loss) is first dropped significantly, while the discriminator loss arises because the discriminator is too weak to distinguish fake or real images and, as a result, it is quite easy for generator to generate fake images to fool the discriminator. Later on, as the discriminator become stronger and stronger, the generator is incapable of generating images that can fool the discriminator and thus the generator loss become higher and higher.

(4) Compare the difference between image generated by VAE/GAN and discuss

Contrary to VAE, GAN generated superb images with clear contour. It is most probably owing to the fact that GAN has two network to fight against each other, while VAE only learns to encode and decode from a latent vectors, and the quality of the latent vectors cannot be supervised by another network.

Problem 3: Domain Adversarial Neural Network

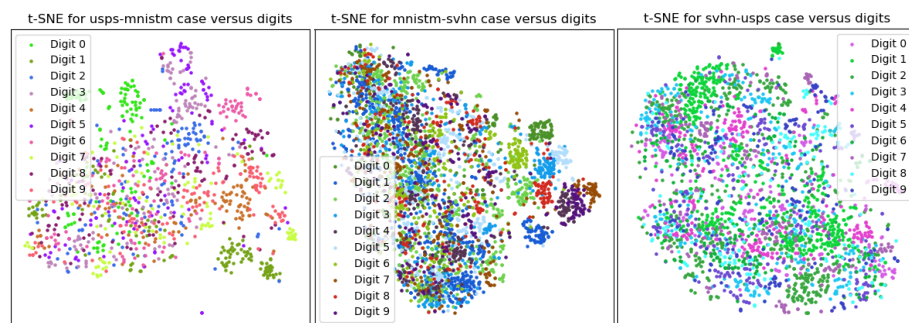
(1) Accuracy of different scenarios

	usps -> mnistm	mnistm -> svhn	svhn -> usps
Source only	30.20%	42.3195%	58.2090%
Domain Adaption	42.15%	47.8795%	57.1998%
Oracle	94.8167%	92.5115%	95.5224%

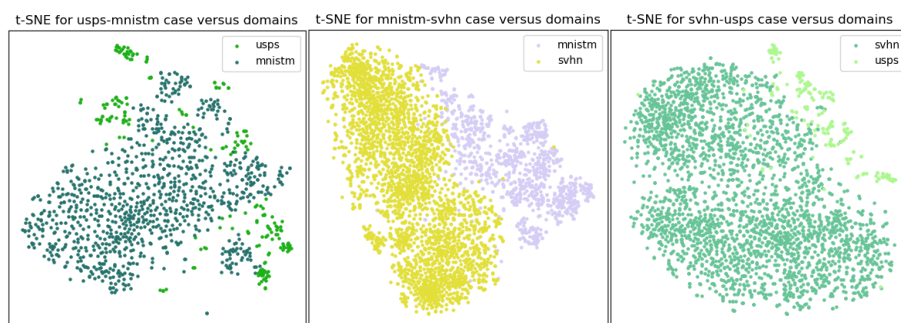
*Oracle: train on target and evaluate on target

(2) t-SNE

- Digits



- Domains



(3) Model Architecture of DANN

```
DANN(  
  (feature_extractor): Sequential(  
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))  
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace=True)  
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (4): Dropout2d(p=0.25, inplace=False)  
    (5): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))  
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (7): ReLU(inplace=True)  
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (9): Dropout2d(p=0.3, inplace=False)  
    (10): Flatten()  
  )  
)
```

```
(label_classifier): Sequential(  
  (0): Linear(in_features=3200, out_features=256, bias=True)  
  (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
  (2): ReLU(inplace=True)  
  (3): Linear(in_features=256, out_features=128, bias=True)  
  (4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
  (5): ReLU(inplace=True)  
  (6): Linear(in_features=128, out_features=10, bias=True)  
)  
(domain_classifier): Sequential(  
  (0): Linear(in_features=3200, out_features=128, bias=True)  
  (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
  (2): ReLU(inplace=True)  
  (3): Linear(in_features=128, out_features=1, bias=True)  
)  
)
```

Training settings:

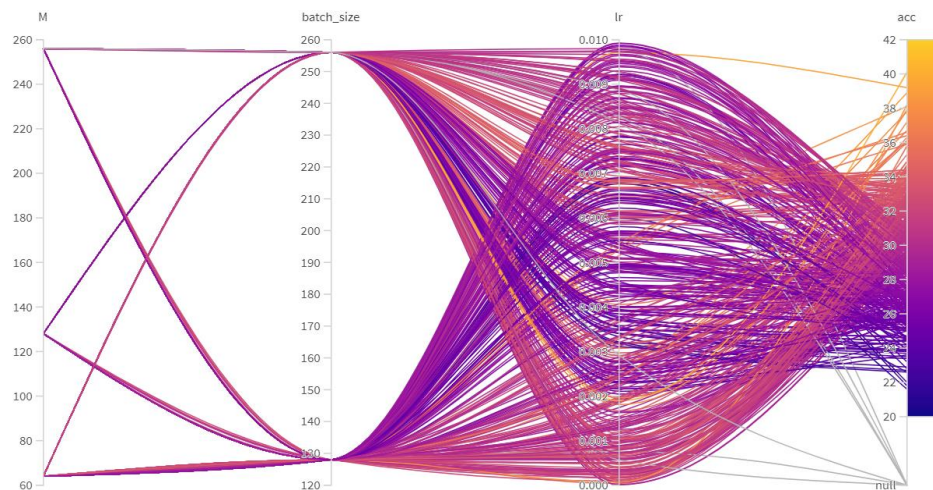
- Epochs: 20
- Batch size: 128
- Optimizer: Adam
- Learning rate: 0.034503325 (for usps->mnistm) / 0.001 (for other cases)
- No data augmentation and learning rate scheduling is used

Other hyper-parameters follow the default values in PyTorch.

(4) Discuss what you've observed and learned from DANN

The idea of DANN, though being similar to GAN, is quite novel. It **explicitly forces the model to learn the “universal” features that can be applied to both source and target domains**. As mentioned above, the lack of supervision of the quality of the latent vectors produced by VAE leads to weaker representation and poorer reconstruction. With the introduction of a “Domain classifier” in DANN, it is able to force model and supervise the learned representation and the features, thus having better quality.

Despite saying much of the advantages, DANN is surprisingly hard to train. The hyper-parameters are hard to choose and the training failed most of the time. For instance, in the “usps to mnist-m” scenario, if I use the very same model used in “mnistm to svhn” or “svhn to usps”, the model almost failed all the times. I use Bayesian hyper-parameter tuning method to run 512 times and still fail to surpass the baseline (40%) except only one of them get 40.1% accuracy on target dataset. I presume the singularity of USPS results in these tragic results, as it has only one channel and was pre-processed to convert to three channels. In the end, I change the model architecture for “usps to mnist-m” by reducing the input channels from three to one, as the three channels are actually identical. This reduces the model complexity and guides the model to learn the essence of the USPS data.



Problem 4: Improved Unsupervised Domain Adaption

In this section, I implemented “**Sliced Wasserstein Discrepancy for Unsupervised Domain Adaptation**” (SWD), published on CVPR 2019, as my improved unsupervised domain adaption model.

Specifically, SWD has a generator and two classifiers. The generator acts as a feature extractor and two classifiers sharing the same architecture aims to classify the given image to the class it belongs to.

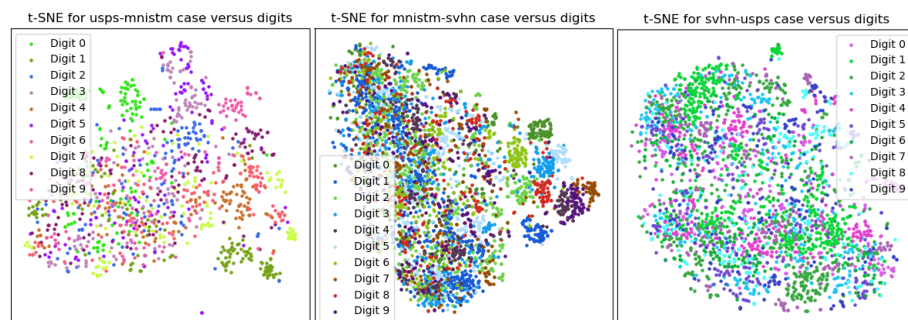
(1) Accuracy of different scenarios

	usps -> mnistm	mnistm -> svhn	svhn -> usps
Source only	24.8%	36.0983%	61.1940%
Oracle*	96.1%	91.3594%	93.0348%
Domain Adaption by DANN	42.15%	47.8795%	57.1998%
Domain Adaption by SWD	50.0800%	48.6824%	73.7917%

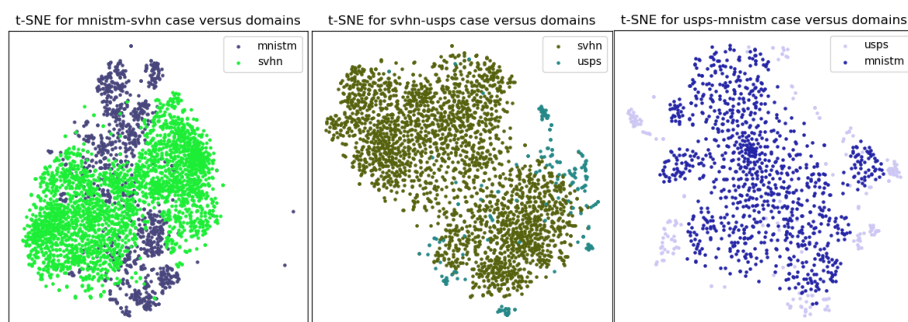
*Oracle: train on target and evaluate on target

(2) t-SNE

- Digits



- Domains



(3) Model Architecture of SWD

(for MNIST-M→SVHN, SVHN→USPS)

```

Generator(
  (conv_blocks): Sequential(
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=1, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=1, dilation=1, ceil_mode=False)
    (8): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (9): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU(inplace=True)
    (11): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (12): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): ReLU(inplace=True)
  )
  (fc): Sequential()
)

Classifier(
  (main_module): Sequential(
    (0): Linear(in_features=8192, out_features=3072, bias=True)
    (1): BatchNorm1d(3072, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=3072, out_features=2048, bias=True)
    (5): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): Linear(in_features=2048, out_features=10, bias=True)
  )
)

```

(for USPS→MNIST-M)

```

Generator(
  (block1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=(1, 1), ceil_mode=False)
  )
  (block2): Sequential(
    (0): Conv2d(32, 48, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=(1, 1), ceil_mode=False)
  )
)

Classifier(
  (dropout1): Dropout(p=0.0, inplace=False)
  (block): Sequential(
    (0): Linear(in_features=768, out_features=100, bias=True)
    (1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Linear(in_features=100, out_features=100, bias=True)
    (4): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): Linear(in_features=100, out_features=10, bias=True)
  )
)

```

Training settings:

	MNIST-M→SVHN	SVHN→USPS	USPS→MNIST-M
Batch size	128	256	256
Learning rate	5e-5	1e-4	5e-3
M (number of radial projections)	128	256	128
<ul style="list-style-type: none"> - Epochs: 15 - k (number of steps taken to maximize discrepancy): 10 - Optimizer: Adam - No data augmentation and learning rate scheduling is used <p>Other hyper-parameters follow the default values in PyTorch.</p>			

(4) Discuss what you've observed and learned from the improved UDA model

SWD aims to **minimize the dissimilarity of probability between two classifiers**. This additional step better **aligns the distributions of source and target** by **utilizing the task-specific decision boundaries**. The decision boundary of models is thus enhanced, leading to better results than DANN.

Plotting the t-SNE of the original data, we can see that the original distribution is **separated** from each other. From this point of view, it is reasonable that the model only trained on source performs bad on target data. After domain adaption model, either DANN or SWD, is applied, the extracted features from two domains are much better entangled between two data sources. As indicated by the figures below, **SWD adapts and entangles the data distribution and clusters different digits far better than that by DANN**.

