Problem1 GAN

B09901104翁瑋杉

(5%) Please print the model architecture of method A and B.

```
Generator(
  (tconv1): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (tconv2): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (tconv3): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (tconv4): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (tconv5): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
)
Discriminator(
  (conv1): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (conv2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv4): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv5): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
)
```

DCGAN

```
WGAN_Discriminator(
  (l1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Sequential(
      (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (2): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_runn
ing_stats=False)
      (3): LeakyReLU(negative_slope=0.2)
    )
    (3): Sequential(
      (0): Conv2d(128, 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): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track_runn
ing_stats=False)
      (3): LeakyReLU(negative_slope=0.2)
    )
    (4): Sequential(
      (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (2): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=False, track_runn
ing_stats=False)
      (3): LeakyReLU(negative_slope=0.2)
    )
    (5): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1))
  )
)
```

Model B: WGAN-GP

This WGAN use same generator as DCGAN , with discriminator slightly different.

(5%) Please show the first 32 generated images of both method A and B then discuss the difference between method A and B.

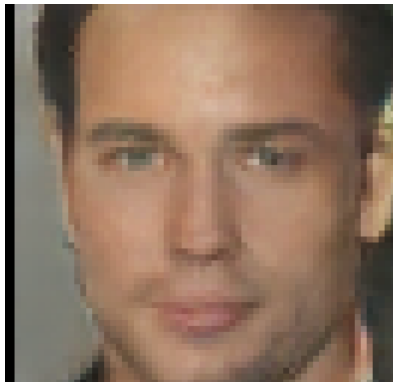
DCGAN

WGAN-GP

- The main difference between DCGAN and WGAN-GP is how to calculate loss. For WGAN-GP, techniques such as weight-clipping and gradient penalty are utilized to prevent discriminator from being too strong, and make sure that the training process is stable.

(5%) Please discuss what you've observed and learned from implementing GAN.

- Hyper-parameters provided by authors of paper are important, it is advised not to modify it.
- Sometimes, discriminator would become too accurate, which leads to "gradient vanish". They key point of training DCGAN and WGAN is to make sure that both discriminator and generator are well-matched.
- The model of discriminator is designed to be shallow, compared to typical classifier. This is intended to weaken discriminator.
- (I think I have generated pictures of Brad Pitt and Mark Zuckerberg lol.)




Problem2 Diffusion model

B09901104翁瑋杉

1. (5%) Please print your model architecture and describe your implementation details.

```
ContextUnet(
  (init_conv): ResidualConvBlock(
    (conv1): Sequential(
      (0): Conv2d(3, 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()
    )
    (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()
    )
  )
  (down1): UnetDown(
    (model): Sequential(
      (0): ResidualConvBlock(
        (conv1): 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()
        )
        (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()
        )
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
  (down2): UnetDown(
    (model): Sequential(
      (0): ResidualConvBlock(
        (conv1): Sequential(
          (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): GELU()
        )
        (conv2): Sequential(
          (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): GELU()
        )
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
```

```
(to_vec): Sequential(
  (0): AvgPool2d(kernel_size=7, stride=7, padding=0)
  (1): GELU()
)
(timeembed1): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=1, out_features=512, bias=True)
    (1): GELU()
    (2): Linear(in_features=512, out_features=512, bias=True)
  )
)
(timeembed2): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=1, out_features=256, bias=True)
    (1): GELU()
    (2): Linear(in_features=256, out_features=256, bias=True)
  )
)
(contextembed1): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=10, out_features=512, bias=True)
    (1): GELU()
    (2): Linear(in_features=512, out_features=512, bias=True)
  )
)
(contextembed2): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=10, out_features=256, bias=True)
    (1): GELU()
    (2): Linear(in_features=256, out_features=256, bias=True)
  )
)
(up0): Sequential(
  (0): ConvTranspose2d(512, 512, kernel_size=(7, 7), stride=(7, 7))
  (1): GroupNorm(8, 512, eps=1e-05, affine=True)
  (2): ReLU()
)
```

```
,
(up1): UnetUp(
  (model): Sequential(
    (0): ConvTranspose2d(1024, 256, kernel_size=(2, 2), stride=(2, 2))
    (1): ResidualConvBlock(
      (conv1): 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()
      )
      (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()
      )
    )
    (2): ResidualConvBlock(
      (conv1): 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()
      )
      (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()
      )
    )
  )
)
```

```
  (up2): UnetUp(
    (model): Sequential(
      (0): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
      (1): ResidualConvBlock(
        (conv1): 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()
        )
        (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()
        )
      )
      (2): ResidualConvBlock(
        (conv1): 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()
        )
        (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()
        )
      )
    )
  )
  (out): Sequential(
    (0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): GroupNorm(8, 256, eps=1e-05, affine=True)
    (2): ReLU()
    (3): Conv2d(256, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
)
```

Reference:
https://github.com/TeaPearce/Conditional_Diffusion_MNIST
https://arxiv.org/abs/2207.12598

Implement details:

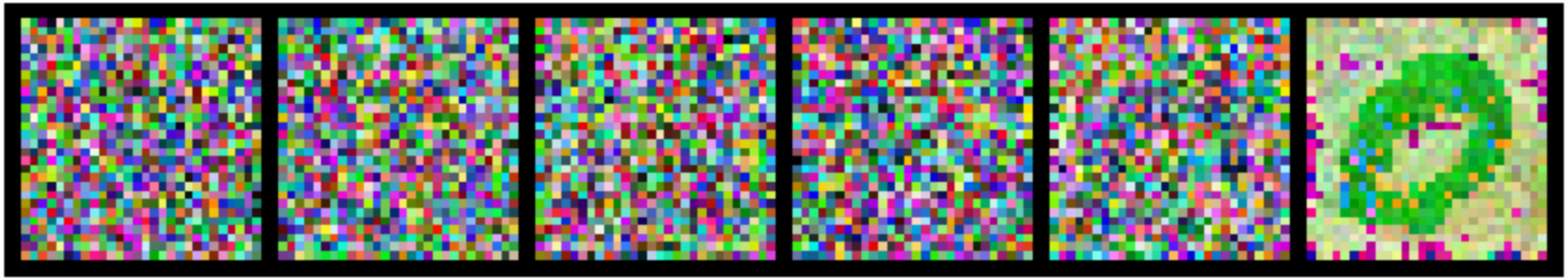
- The neural network architecture is a small U-Net.
- Classifier-Free Diffusion Guidance method is used for conditioning.
- Loss is MSE between added noise and predicted loss
- The context embedding may be dropped with probability of 0.1 during training.

```
config = {
    'data_dir': '../input/hw2-data/h
    'dfname': '../input/hw2-data/hw2
    'lr': 1e-4,
    'num_epochs': 100,
    'batch_size': 256,
    'num_classes': 10,
    'n_T': 300,
    'img_size': 28,
```

- (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. [see the below example]

- (5%) Visualize total six images in the reverse process of the **first "0"** in your grid in (2) **with different time steps**. [see the below example]

  T = 0 ,200, 400,600,800,1000



- (5%) Please discuss what you've observed and learned from implementing conditional diffusion model.

  - I learned that the idea behind diffusion model is pretty much similar to VAE but we only have to train network in reverse process. However, it is quite hard to implement.
  - I found that a good classifier is needed to guide the diffusion model.
  - The number of time steps is crucial to training time, but if noise step is too large, it becomes harder for model to reverse the process.

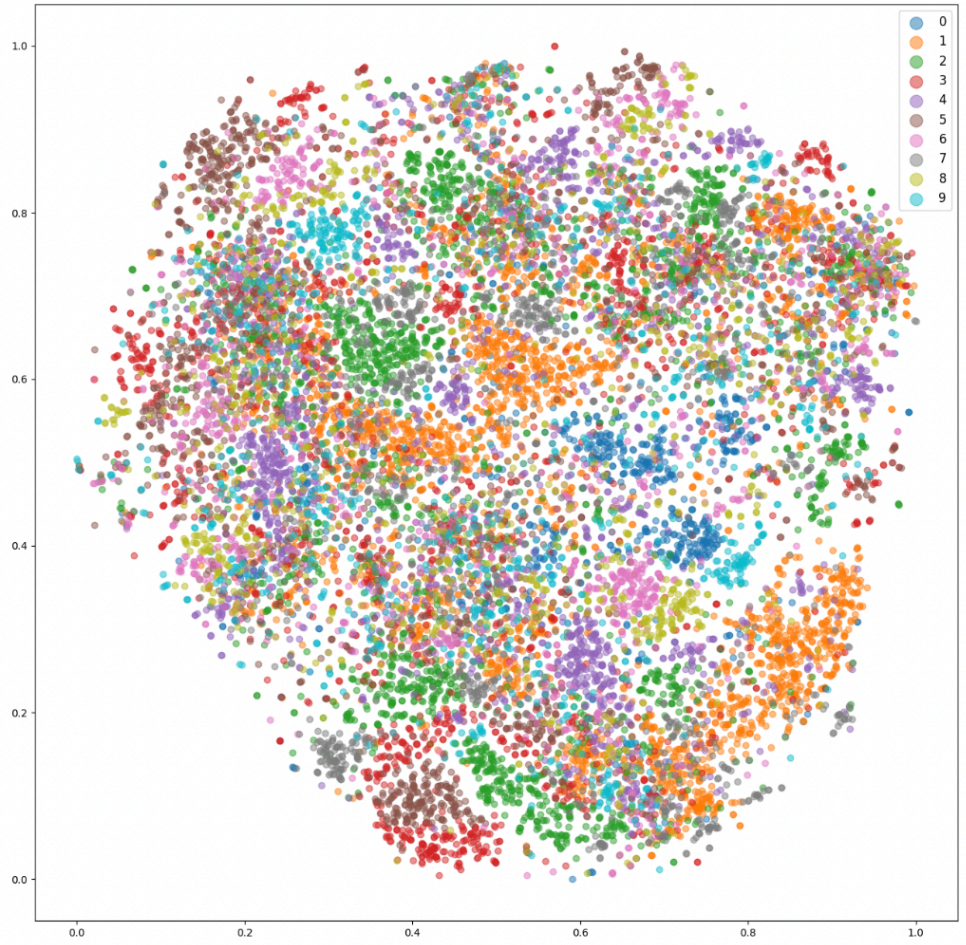
Problem3 Domain Adaption

B09901104翁瑋杉

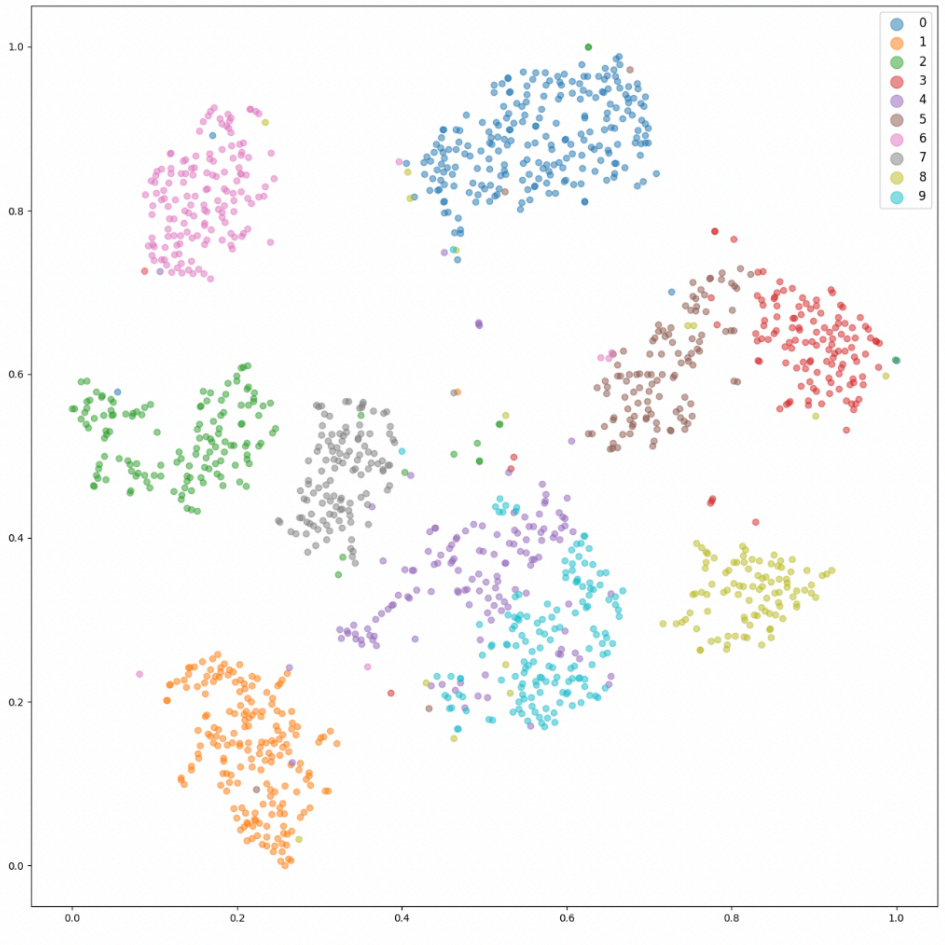
(5%) Please create and fill the table with the following format **in your report**:

	MNIST-M → SVHN	MNIST-M → USPS
Trained on source	0.281	0.732
Adaptation (DANN)	0.48	0.827
Trained on target	0.916	0.948

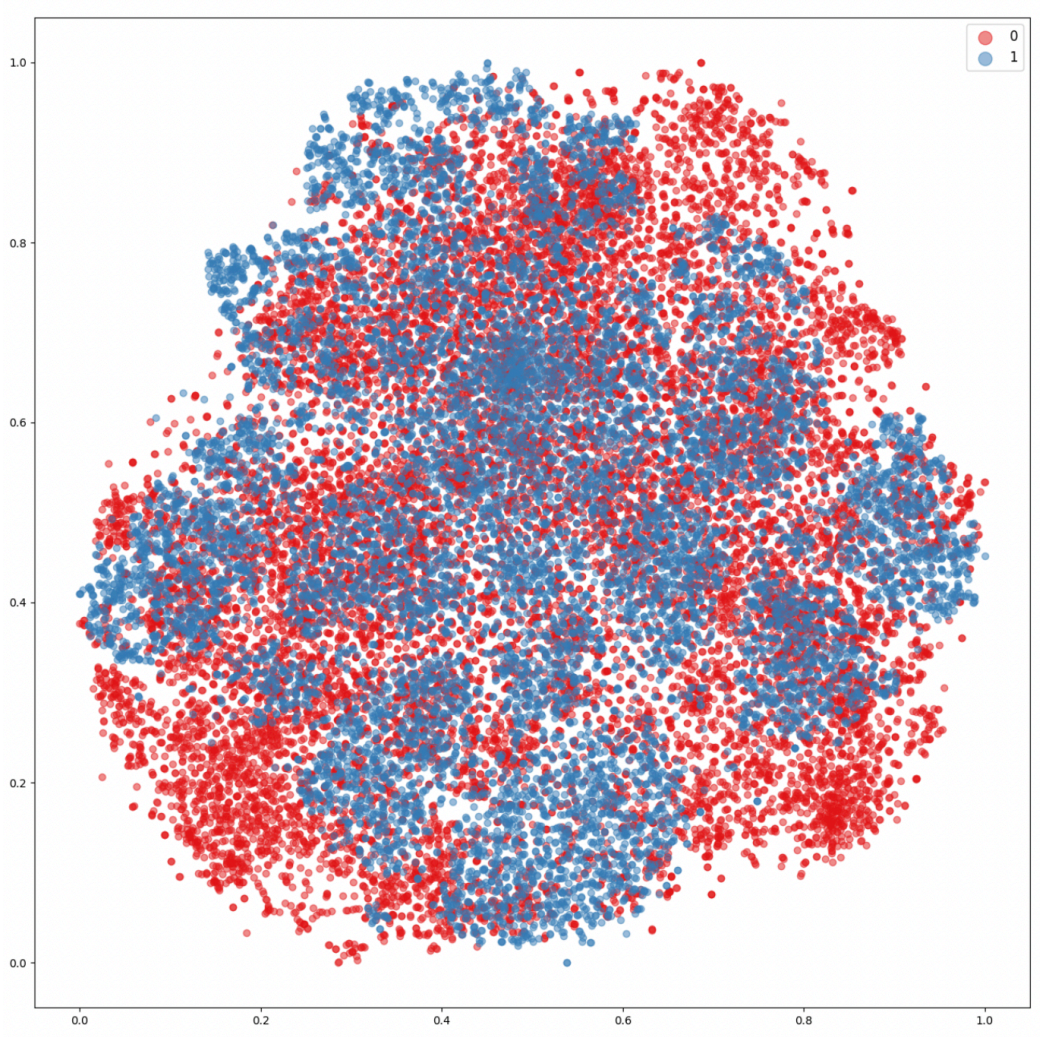
(8%) Please visualize the latent space of DANN by mapping the ***validation*** images to 2D space with t-SNE. For each scenario, you need to plot two figures which are colored **by digit class (0-9)** and **by domain**, respectively.



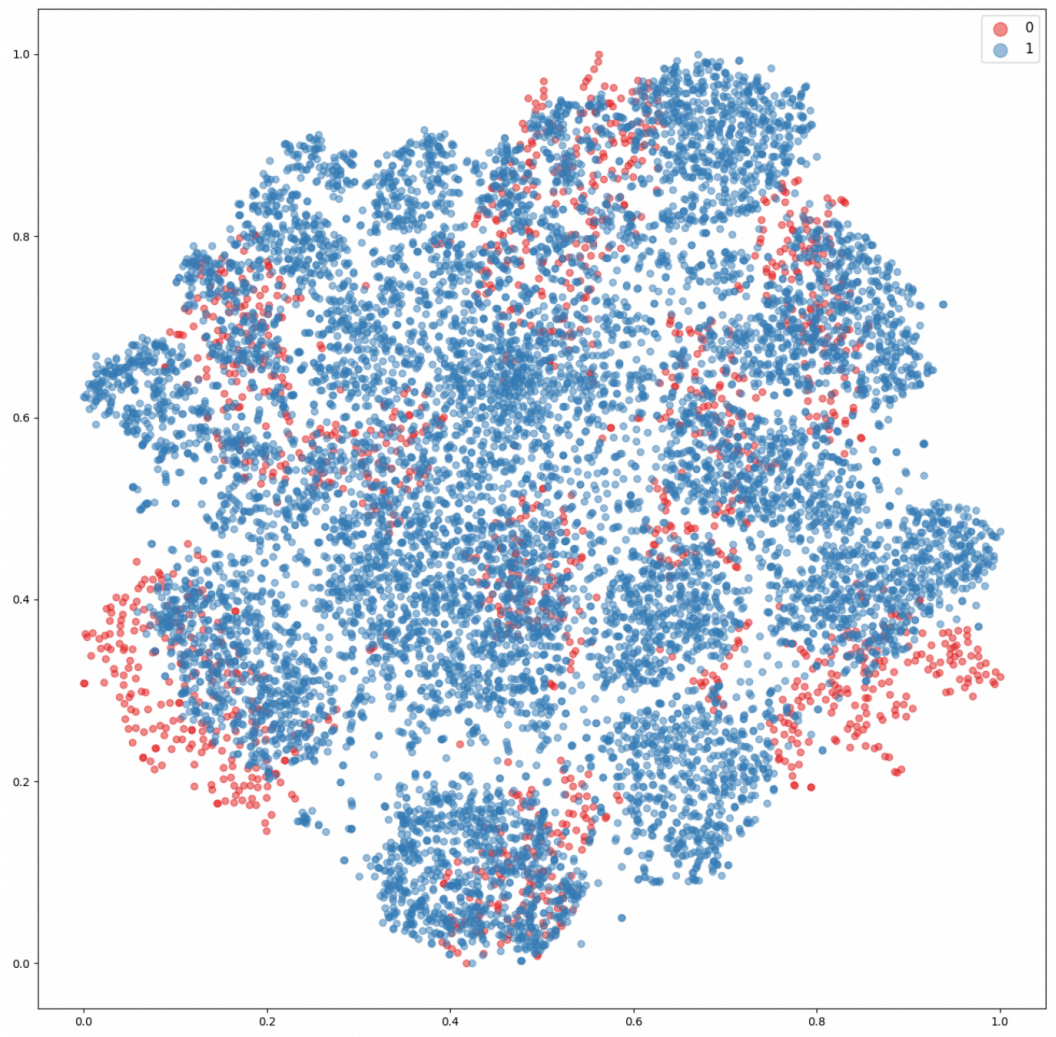
by digit class on SVHN



by digit class on USPS



by domain on SVHN



by domain on USPS

(10%) Please describe the implementation details of your model and discuss what you've observed and learned from implementing DANN.

Implement details:

- For images from SVHN and USPS, different transform and normalize are applied.
- Hyper-parameters :
  - Learning rate = 1e-4
  - batch_size = 256
- Optimizer is Adam, loss is NLLLoss
- The rest training flow basically follows https://arxiv.org/abs/1505.07818

- I've observed the result of t-SNE, and found that USPS dataset is better coherent with MNIST-M on both domain and digit class. The validation accuracy also shows that domain adaption from MNIST-M to USPS is very successful comparing to SVHN.
- I think that it was because images from MNIST-M and USPS are much more similar—they are composed only by digits and colored background, while SHVN contains high variation of format and light and shade.
- Still, the accuracy of DANN on SVHN could reach 0.5m which is a great leap from original. This is really impressive to me.