Deep Learning for Computer Vision – HW#2

宋體淮, R09921135, Electrical Engineering

Problem 1: GAN

1. **Build your generator and discriminator from scratch and show your model architecture in your report. Then, train your model on the face dataset and describe implementation details.**

   Generator:

```
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()
  )
)
```

   Discriminator:

```
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()
  )
)
```

   Implementation detail of the GAN model:

   (1) I implement the original DCGAN generator [1], consisting of five transposed convolution layers and follow the guideline propose in the paper.

   (2) I use label smoothing by replace the real label to a random number between 0.9-1, and replace the fake label to a random number between 0-0.1。

   (3) I make the input real image noisy before into discriminator. Because if the real label is a number between 0.9-1, that means the input real image is not that real. In this way, it would become a kind of soft label technique and also make

discriminator harder to train.

2. **Please samples 1000 noise vectors from Normal distribution and input them into your Generator. Save the 1000 generated images in the assigned folder path for evaluation, and show the first 32 images in your report.**

3. **Evaluate your 1000 generated images by FID and IS score.**

FID	IS
29.73	**2.08**

:

4. **Discuss what you've observed and learned from implementing GAN.**
   (1) The training results are highly affected by the random seed you choose because the input of GAN is random noise.
   (2) We can check if *D* and *G* performance matches from the value of *D(x)* and *D(G(z))*, they should be close to 0.5.
   (3) Training more epochs would be helpful (100 in my case), although you may think the results are getting worse during the process.

Problem 2: ACGAN

1. **Build your ACGAN model from scratch and show your model architecture in your report. Then, train your model on the mnistm dataset and describe implementation details.**

   Generator:

```
_netG(
  (fc1): Linear(in_features=110, out_features=384, bias=True)
  (tconv2): Sequential(
    (0): ConvTranspose2d(384, 192, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (tconv3): Sequential(
    (0): ConvTranspose2d(192, 96, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (tconv4): Sequential(
    (0): ConvTranspose2d(96, 48, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (tconv5): Sequential(
    (0): ConvTranspose2d(48, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): Tanh()
  )
)
```

Discriminator:

```
_netD(
  (conv1): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Dropout(p=0.5, inplace=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Dropout(p=0.5, inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Dropout(p=0.5, inplace=False)
  )
  (conv4): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Dropout(p=0.5, inplace=False)
  )
  (conv5): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Dropout(p=0.5, inplace=False)
  )
  (conv6): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Dropout(p=0.5, inplace=False)
  )
  (fc_dis): Linear(in_features=8192, out_features=1, bias=True)
  (fc_aux): Linear(in_features=8192, out_features=10, bias=True)
  (softmax): Softmax(dim=None)
  (sigmoid): Sigmoid()
)
```

Implementation detail of the ACGAN model:

(1) I use one-hot encoding for the input class label instead of just a one category value, and then concatenate it with the input noise. Because just one category value may not be prominent for model to learn category condition.

(2) I also use the label smoothing like in problem 1.

2. **Evaluate your generated output by the classification accuracy with a pretrained digit classifier.**

Accuracy
0.903

3. **Show 10 images for each digit (0-9) in your report.**

Problem 3: DANN

1. **Compute the accuracy on target domain, while the model is trained on source domain only. (lower bound)**

SVHN→MNIST-M	MNIST-M→USPS	USPS→SVHN
0.4563	0.7154	0.2571

2. **Compute the accuracy on target domain, while the model is trained on source and target domain. (domain adaptation)**

SVHN→MNIST-M	MNIST-M→USPS	USPS→SVHN
0.4717	**0.7947**	**0.2809**

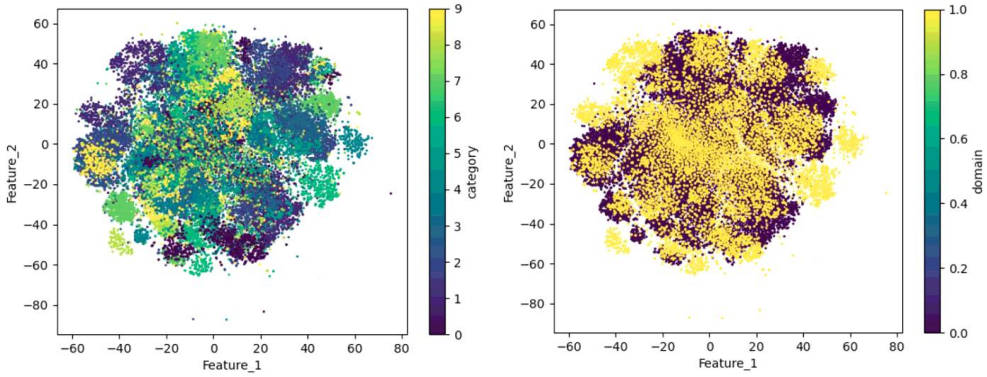
3. **Compute the accuracy on target domain, while the model is trained on target domain only. (upper bound)**

SVHN→MNIST-M	MNIST-M→USPS	USPS→SVHN
0.9561	0.9671	0.9172

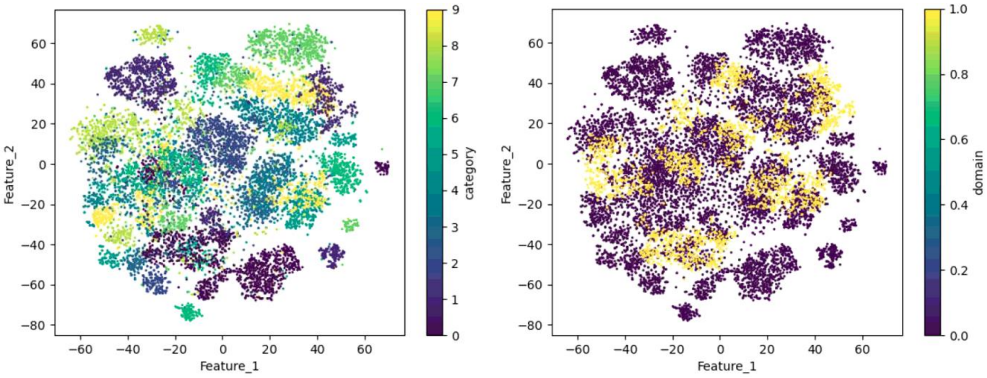
4. Visualize the latent space by mapping the testing images to 2D space with t-SNE and use different colors to indicate data of (a) different digit classes 0-9 and (b) different domains.

For the following figures, purple color represent source domain data and yellow color represents target domain data.

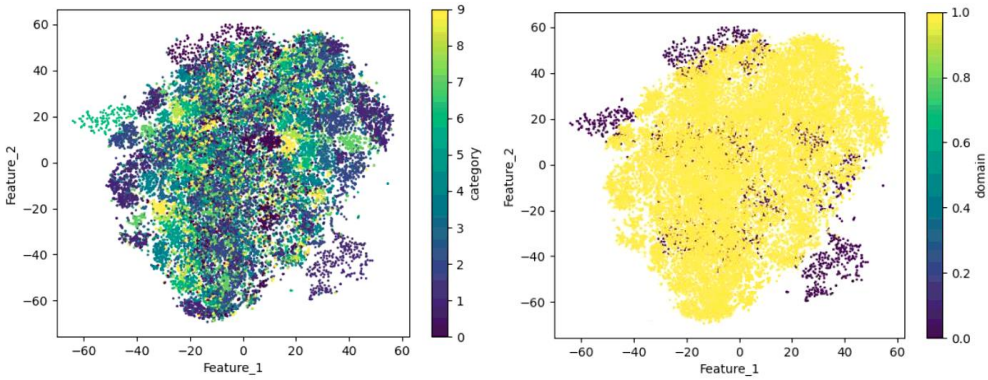
(1) SVHN→MNIST-M



(2) MNIST-M→USPS



(3) USPS→SVHN



Bonus: Improved UDA model

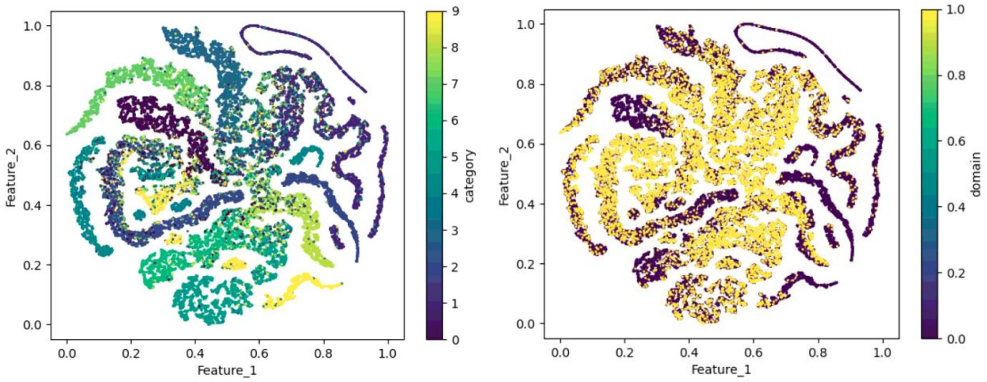
1. **Compute the accuracy on target domain, while the model is trained on source and target domain. (domain adaptation)**

SVHN→MNIST-M	MNIST-M→USPS	USPS→SVHN
0.6103	**0.8794**	**0.2970**

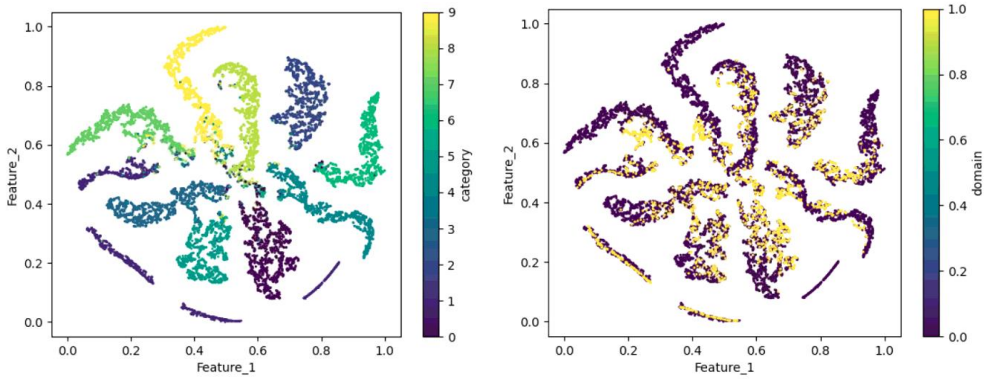
2. **Briefly describe implementation details of your model and discuss what you've observed and learned from implementing your improved UDA model.**

I use the pre-trained resnet34 followed by three transposed convolution layers to build an encoder-decoder-like feature extractor. From the t-SNE results shown below, we can clearly see that the distribution between the two domains are more aligned together and clustered better.

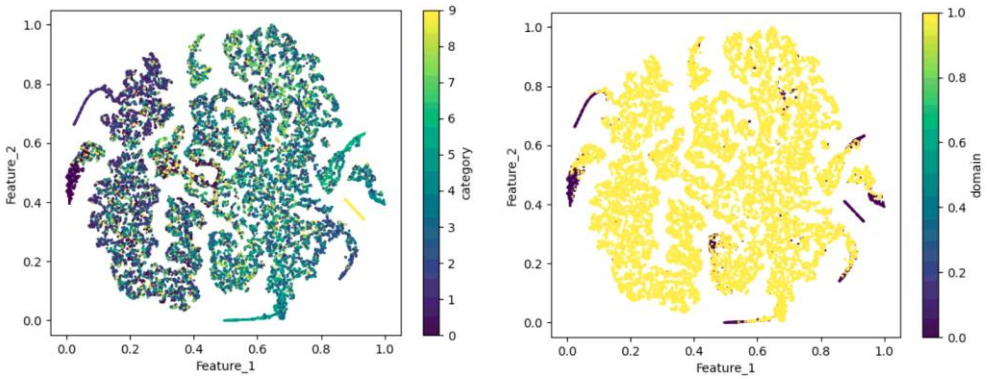
(1) SVHN→MNIST-M



(2) MNIST-M→USPS



(3) USPS→SVHN



Reference

[1] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks."