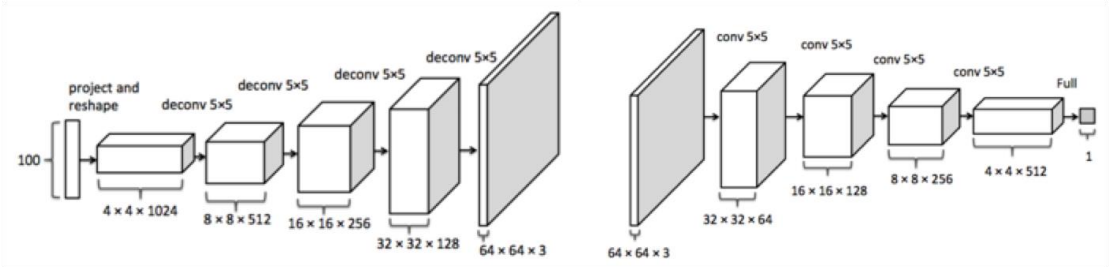
DLCH HW3 Report

余奇安 r07921010 電機碩一

Problem1 : GAN (DCGAN)

1. Describe the architecture & implementation details of your model.



Ref: http://www.timzhangyuxuan.com/project_dcgan/

Generator structure:

```
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)
    (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)
    (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)
    (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)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
```

Discriminator structure:

```
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)
    (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)
    (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)
    (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)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Implementation detail:

Batchsize:128, noise dimension:100, epochs:200, learning rate:0.0002,

optimizer: Adam betas=(0.5, 0.999)

loss function: BCEloss

all the image data is normalize with mean: 0.5, std: 0.5

2. Plot 32 random images generated from your model.[fig1_2.jpg]

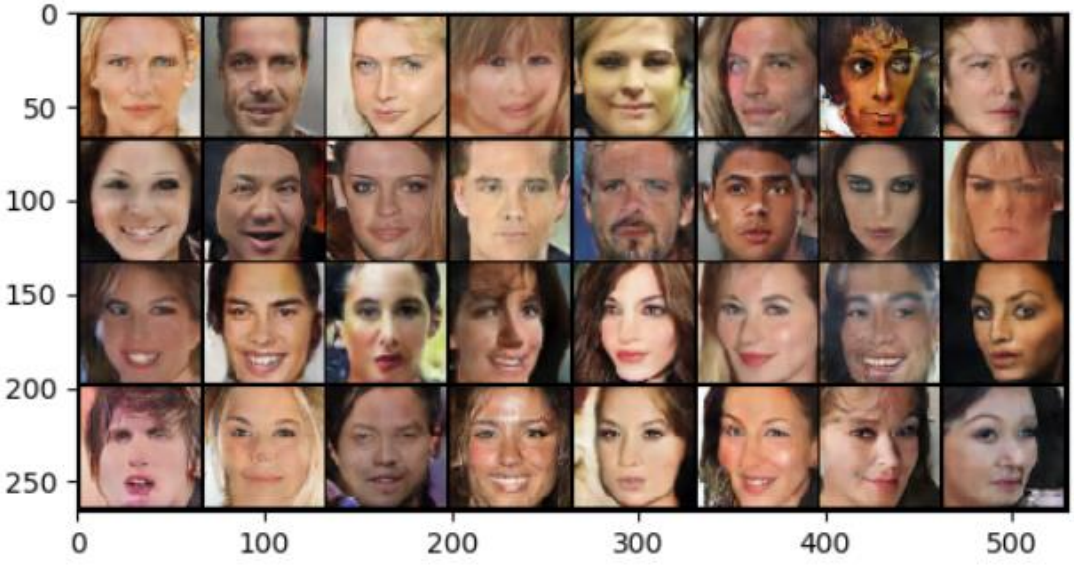
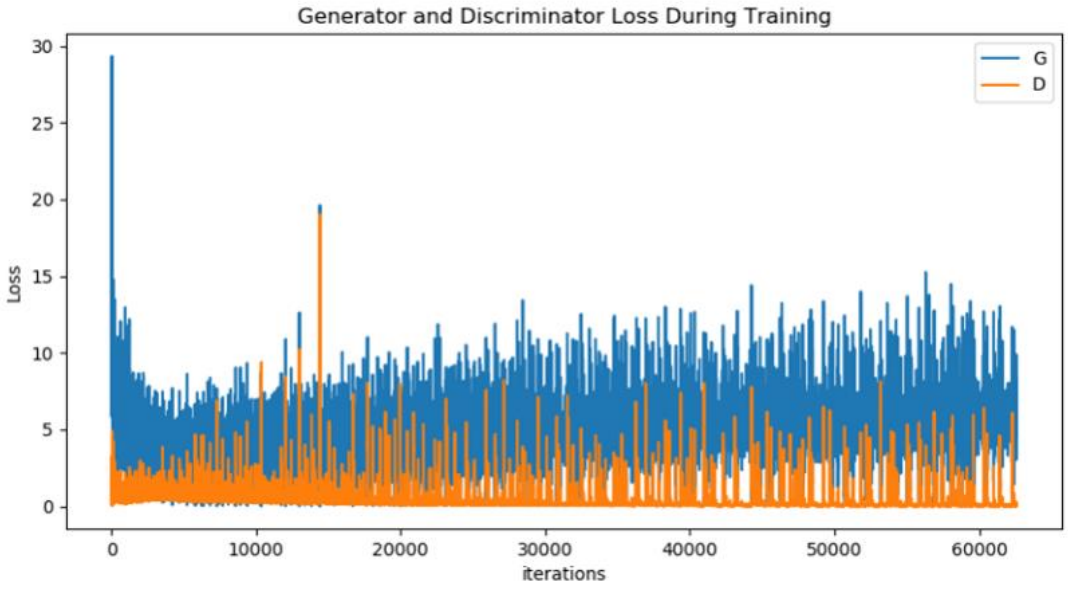


Fig1_2.jpg

3. Discuss what you've observed and learned from implementing GAN



We can see that the Discriminator loss can be very low, while the Generator loss usually 2 times than Discriminator. I do some experiment try to minimize Generator loss to compete to Discriminator loss. For example, I update Generator 5 times to Discriminator, but the image quality do not improve very much.

In the contrary, updating Discriminator 5 times to Generator, the image quality improves a little. I view it very weird but my explanation is the image quality depends on Discriminator much more than Generator. Base on loss function,

$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}[\log(1 - D\ (G(x)))] + \mathbb{E}[\log D(y)]$$

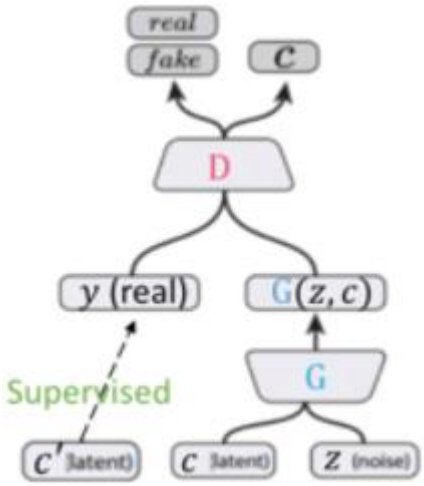
The total loss is depend on log D, so maybe we should find the optimal discriminator,

meanwhile minimize log(1-D(G(x))).
Eventually, I try 0 to 999 random seeds. I found that the noise affects the image quality a lot!

Problem2: ACGAN

I choose smiling attributes

1. Describe the architecture & implementation details of your model.
   Follow the acgan slide from TA, ACGAN structure:



Generator:

```
Generator(
  (decoder): Sequential(
    (0): ConvTranspose2d(101, 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)
    (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)
    (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)
    (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)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Discriminator:

```
Discriminator(
  (decoder): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace)
    (11): Conv2d(512, 64, kernel_size=(4, 4), stride=(1, 1))
  )
  (fc_dis): Linear(in_features=64, out_features=1, bias=True)
  (fc_aux): Linear(in_features=64, out_features=1, bias=True)
  (softmax): Softmax()
  (sigmoid): Sigmoid()
)
```

Implementation detail:

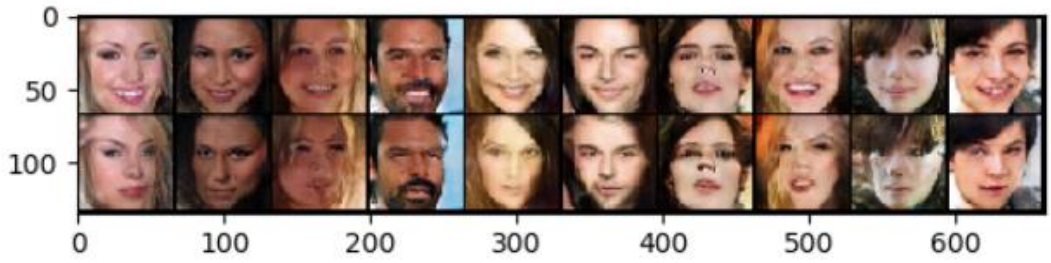
Batchsize:128, noise dimension:100, epochs:200, learning rate:0.0002,

optimizer: Adam betas=(0.5, 0.999)

loss function: BCEloss

all the image data is normalize with mean: 0.5, std: 0.5

2.Plot 10 random pairs of generated images from your model, where each pair should be generated from the same random vector input but with opposite attribute. This is to demonstrate your model's ability to disentangle features of interest.



[fig2_2.jpg]

3. Discuss what you've observed and learned from implementing ACGAN.

During the training process, I found that the image quality with attribute generally better than image without attribute. I view it reasonable because smiling attribute becomes kinds of constrains. With this constrain and supervised learning, the generator knows more information about how to generate images.

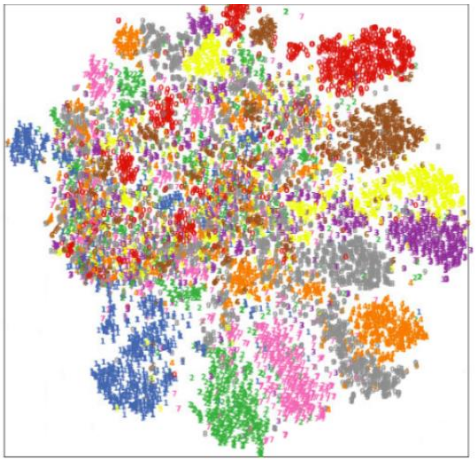
Problem3: DANN

1.2.3Accuracy:

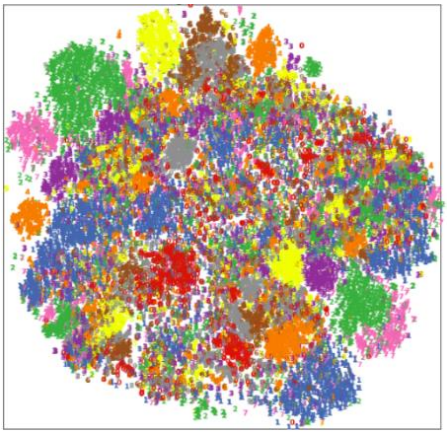
	USPS to MNIST-M	MNIST-M to SVHN	SVHN to USPS
Trained on source	17%	29%	60%
Adaption	38.43%	45.43%	51.42%
Trained on target	90%	90%	96%

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 digits classes and (b) different domains

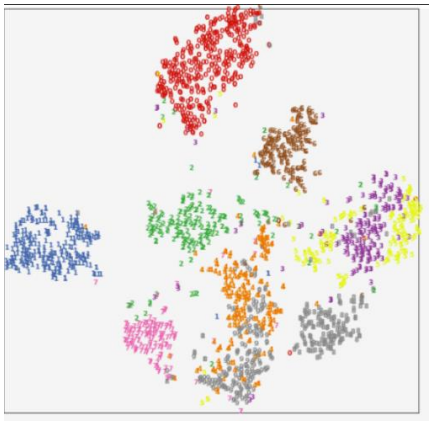
(1)USPS to MNIST-M



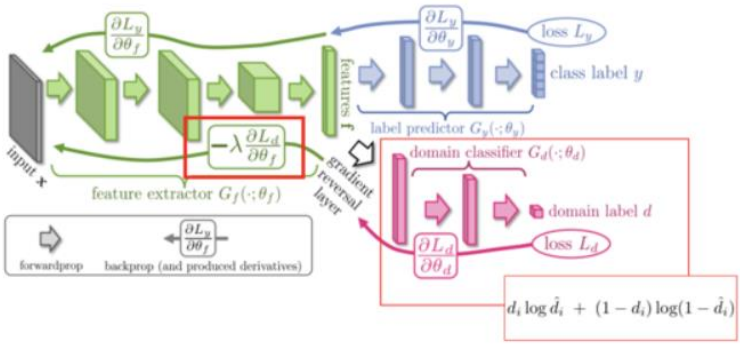
(2) MNISTM to SVHN



(3) SVHN to USPS



5.Describe the architecture & implementation detail of your model.



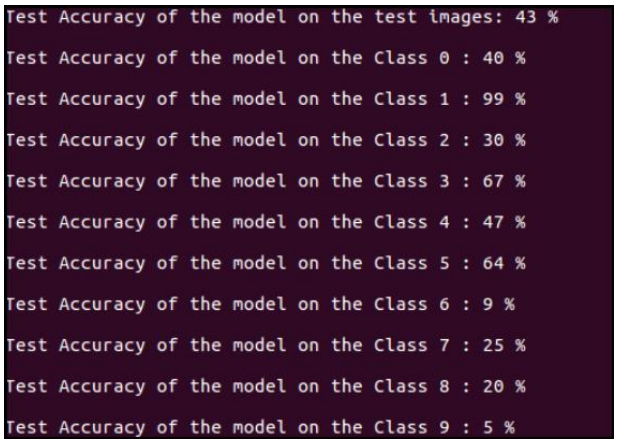

```
CNNModel(
  (feature): Sequential(
    (f_conv1): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1))
    (f_bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (f_pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (f_relu1): ReLU(inplace)
    (f_conv2): Conv2d(64, 50, kernel_size=(5, 5), stride=(1, 1))
    (f_bn2): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (f_drop1): Dropout2d(p=0.5)
    (f_pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (f_relu2): ReLU(inplace)
  )
  (class_classifier): Sequential(
    (c_fc1): Linear(in_features=800, out_features=100, bias=True)
    (c_bn1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (c_relu1): ReLU(inplace)
    (c_drop1): Dropout2d(p=0.5)
    (c_fc2): Linear(in_features=100, out_features=100, bias=True)
    (c_bn2): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (c_relu2): ReLU(inplace)
    (c_fc3): Linear(in_features=100, out_features=10, bias=True)
    (c_softmax): LogSoftmax()
  )
  (domain_classifier): Sequential(
    (d_fc1): Linear(in_features=800, out_features=100, bias=True)
    (d_bn1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (d_relu1): ReLU(inplace)
    (d_fc2): Linear(in_features=100, out_features=2, bias=True)
    (d_softmax): LogSoftmax()
  )
)
```

6.Discuss what you've observed and learned from implementing DANN.

DANN use only one CNN to get the features of images from two different domains, and train a discriminator to distinguish where this feature from.

We can see one special event that without domain adaptation, SVHN to USPS case performs very well (60% accuracy) rather than two other cases (17%, 29%). In my opinion, this is kind of coincidence because SVHN and USPS training data has similar distribution. SVHN dataset is multiply digit number in image with blue background.

Maybe CNN local feature is very similar in feature domain. I also test MNIST-M to USPS and also get 43% accuracy. I think another reason is USPS is quite and grayscale image without too much noise, so any kinds of CNN extraction can encoder enough feature from image and make suitable classification.



Mnist to USPS

Another point is about after training DANN the SVHN to USPS case perform worse than origin. The explanation is that the discriminator is kind of generalization and extract the same features from two different domains. However, this kind of features may not be suitable to be classified at the feature domain. For example, the CNN may extract the same background from USPS and SVHN, but this kind of information can result in misclassification.

Problem4: Improved UDA model

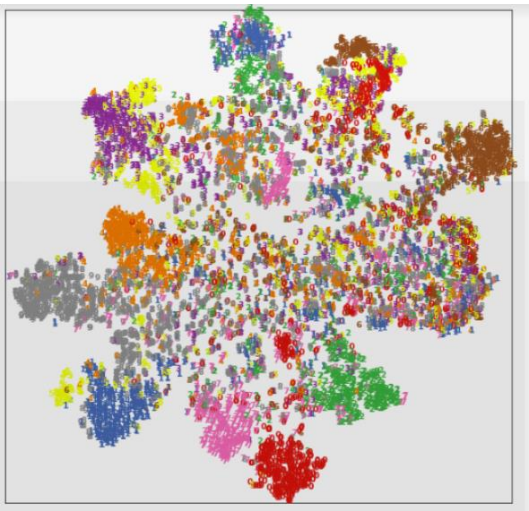
I choose Adversarial Discriminative Domain Adaptation(ADDA) as improved model.

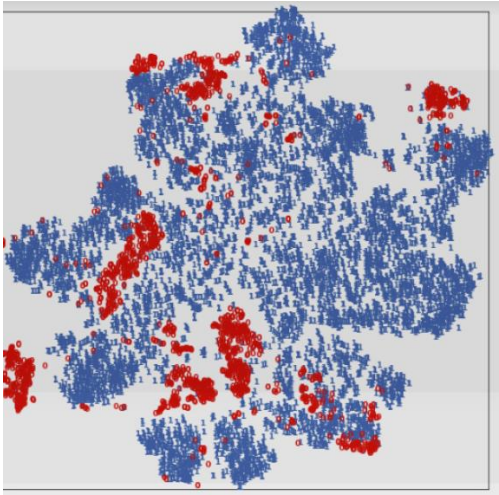
1.

	USPS to MNIST-M	MNIST-M to SVHN	SVHN to USPS
Adaption	40%	47%	53.114%

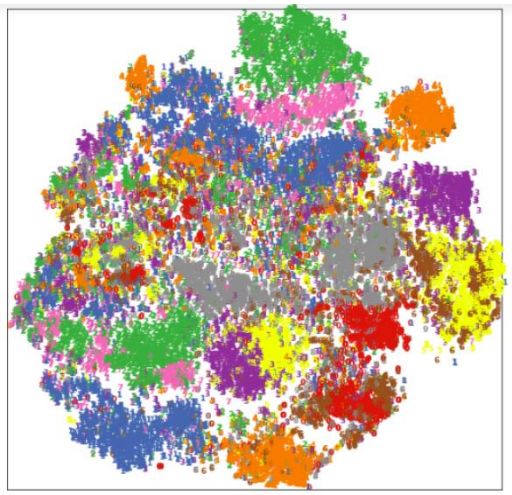
2. Visualize the latent space by mapping the testing images to 2D space(with t-SNE) and use different colors to indicate data of (a) different digits classes and (b) different domains

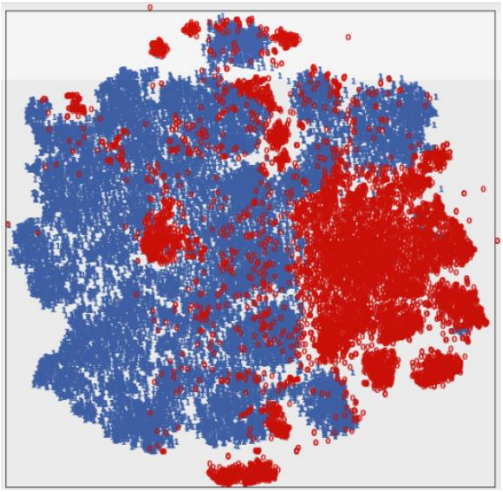
(1) USPS to MNIST



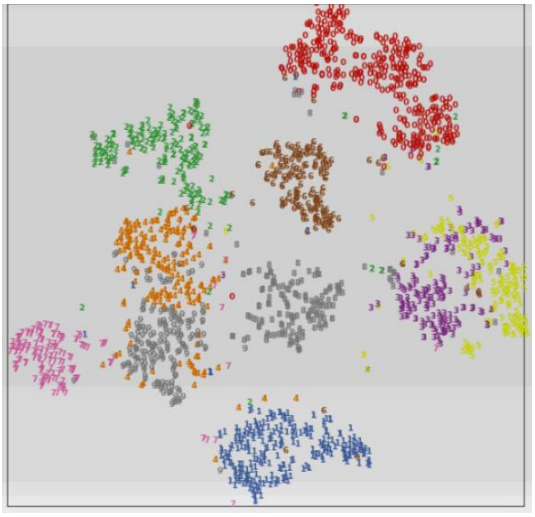


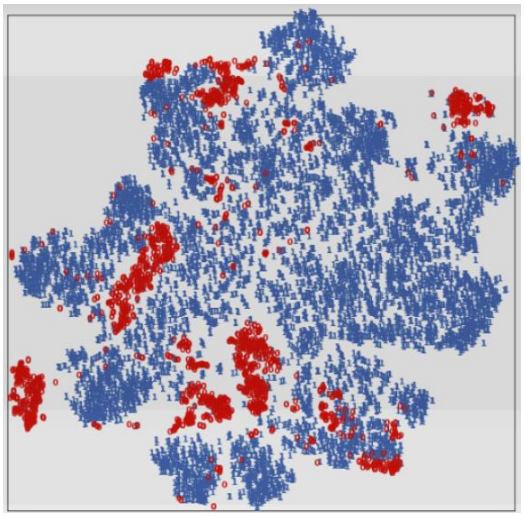
(2)MNIST_M to SVHN



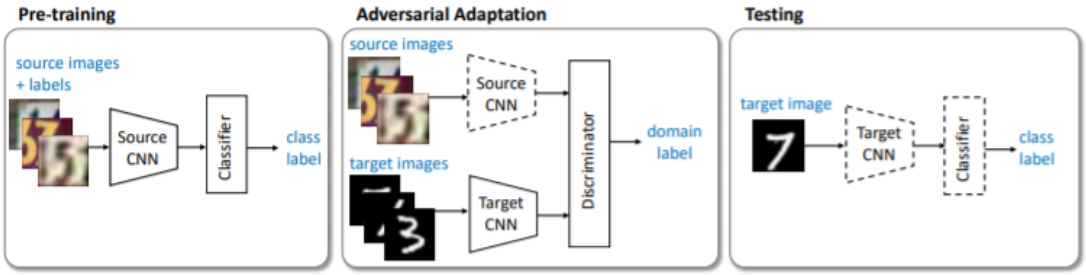


(3)SVHN to USPS





3.Describe the architecture & implementation detail of your mode.




```
CNN(
  (conv1): Conv2d(1, 20, kernel_size=(5, 5), stride=(1, 1))
  (bn1): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(20, 50, kernel_size=(5, 5), stride=(1, 1))
  (bn2): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=800, out_features=500, bias=True)
  (bn3): BatchNorm1d(500, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
CNNClassifier(
  (fc2): Linear(in_features=500, out_features=10, bias=True)
)
Discriminator(
  (fc1): Linear(in_features=500, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=500, bias=True)
  (fc3): Linear(in_features=500, out_features=2, bias=True)
  (bn1): BatchNorm1d(500, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm1d(500, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

Bach size:128

Learning rate:2e-4

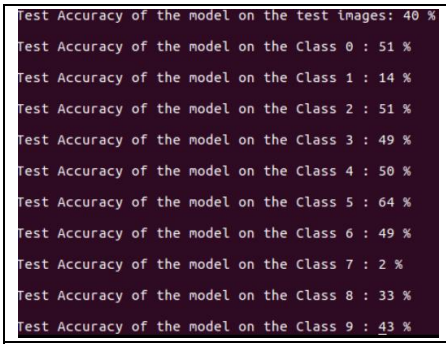
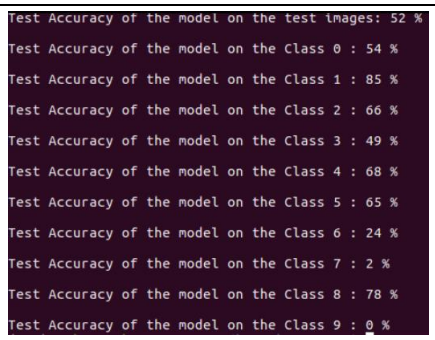
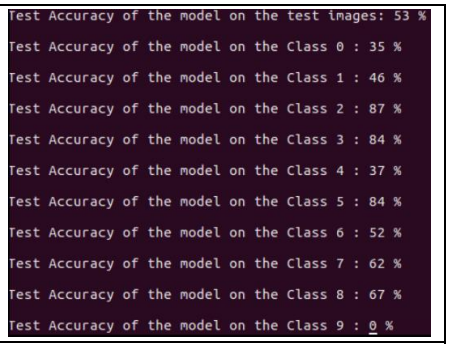
Weight decay:2.5e-4

Iteration:500

Epoch: Iteration /min(target_len, source_len)

4. Discuss what you've observed and learned from implementing your improved UDA model.

The ADDA model applies two different feature extractor on source and target domain respectively. This is more reasonable because there are always some unique features on source and target domain. At this case, the discriminate will be more generalized on high level features. We can see that the performance has slight improved. The DANN shared the classifier on source and target domain and I think maybe we can find tuning the classifier in ADDA training process to reach better performance.

		
Usps to mnist	Mnist to svhn	Svhn to usps

We can see that class 1 7 9 accuracy will not be high at the same time. I think that because there feature is quite similar at high level domain, and the classifier classifies one of them to other two classes. For example, on Mnist to svhn and svhn to usps case, the class 9 accuracy is 0 and the class 1 or class can reach 85% accuracy.

Reference:

https://github.com/aabbas90/ADDA-PyTorch?fbclid=IwAR0-lEB2KqLUM0OPx_ad6xUQ8go6LIy-jJdhGJtY1hjuWZUjCLGEOEPBofg (ADDA)
https://github.com/GakkiAogSan/DANN?fbclid=IwAR0LWiPoDOsI8Sbt_7YM0xWmy5hfMKEi9x2tmpvXS7FSkJ91-BEa-jxs6Hg (DANN)
https://github.com/pytorch/examples/tree/master/dcgan?fbclid=IwAR2RSfMiXXbJBoC2nCLl59rZeZzedcgReK-P1k1qquIk48j4CfJp_e-S_SM (DCGAN)
https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html?fbclid=IwAR0KjJ0UklBJYyK9rPlPq7Np2kMSJ4BFO1KipnYhV_nm2ScJcavrMWj7tu8
https://arxiv.org/abs/1702.05464
https://arxiv.org/pdf/1505.07818.pdf