

DLCV HW3 Report

Shuo-En Chang
R08922A02

November 29, 2020

1 VAE

1.1 My implement of VAE

In the encoder, I use 4 Conv2d layers with different input/output channels, and each filter size is set to 4×4 . There are BatchNorm between each of them and LeakyReLU is used as the activation function. I use stride=2 in the convolutional layer instead of pooling. That's because I experimented between using stride and pooling. And I found that if I choose to use stride as the downsampling method, I can get better results. I think this is because it will lose some important pixels when using pooling.

After the encoder are two parallel linear layers for converting features into logvar and mu, both of which use 1024 as the latent space dimension. And add a normal distribution noise as the input of the decoder.

In the decoder, it is just the transposition of the encoder, and finally, an image of the same size as the input image will be obtained. I use tanh as the activation function and the output will be divided by 2 and plus 0.5 to convert into the range $[0, 1]$ as the input image is.

Following are some hyperparameters using in the training process.

- Latent dimensional: 1024
- Batch size: 256
- λ_{KL} : $2e-4$
- Optimizer: Adam
- Learning rate: 0.00035
- Training epoch: 50
- Data preprocess: Normalize each picture to $[0, 1]$

```
VAE2(  
    (encoder): Sequential(  
      (0): Conv2d(3, 16, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (2): LeakyReLU(negative_slope=0.01, inplace=True)  
      (3): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
      (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (5): LeakyReLU(negative_slope=0.01, inplace=True)
```

```

(6): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
(7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(8): LeakyReLU(negative_slope=0.01, inplace=True)
(9): Conv2d(64, 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): LeakyReLU(negative_slope=0.01, inplace=True)
)
(logvar): Linear(in_features=2048, out_features=1024, bias=True)
(mu): Linear(in_features=2048, out_features=1024, bias=True)
(decoder_input): Linear(in_features=1024, out_features=2048, bias=True)
(decoder): Sequential(
  (0): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): LeakyReLU(negative_slope=0.01, inplace=True)
  (3): ConvTranspose2d(64, 32, kernel_size=(2, 2), stride=(2, 2))
  (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): LeakyReLU(negative_slope=0.01, inplace=True)
  (6): ConvTranspose2d(32, 16, kernel_size=(2, 2), stride=(2, 2))
  (7): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): LeakyReLU(negative_slope=0.01, inplace=True)
  (9): ConvTranspose2d(16, 3, kernel_size=(2, 2), stride=(2, 2))
)
(tanh): Tanh()
)

```

1.2 Learning curve of my model

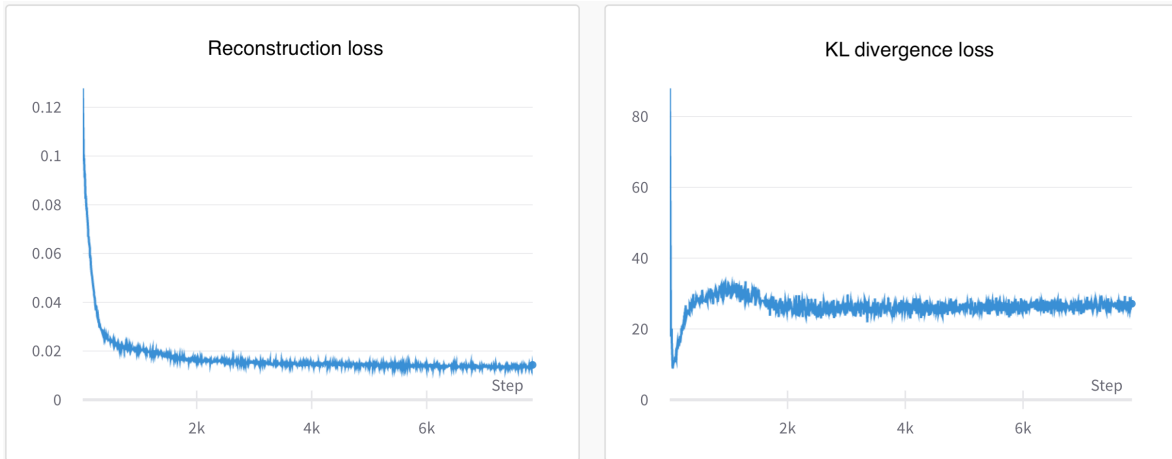





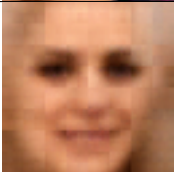
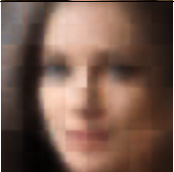
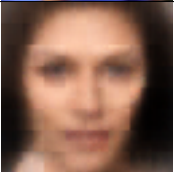
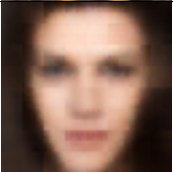
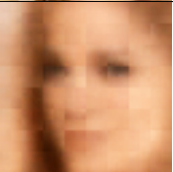





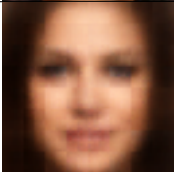
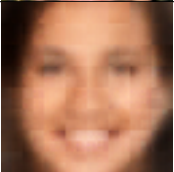
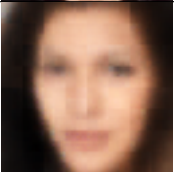
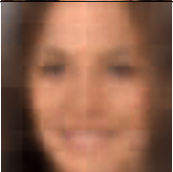
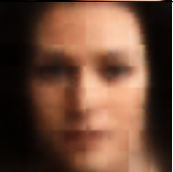


Figure 1: Reconstruction loss and KL divergence loss

1.3 Plot 10 testing images and their reconstructed results.

Table 1: Testing images and their reconstructed results.

Testing images					
Reconstructed					
MSE	0.017	0.009	0.010	0.008	0.011

Testing images					
Reconstructed					
MSE	0.006	0.012	0.008	0.007	0.009

1.4 Plot 32 random generated images from your model.

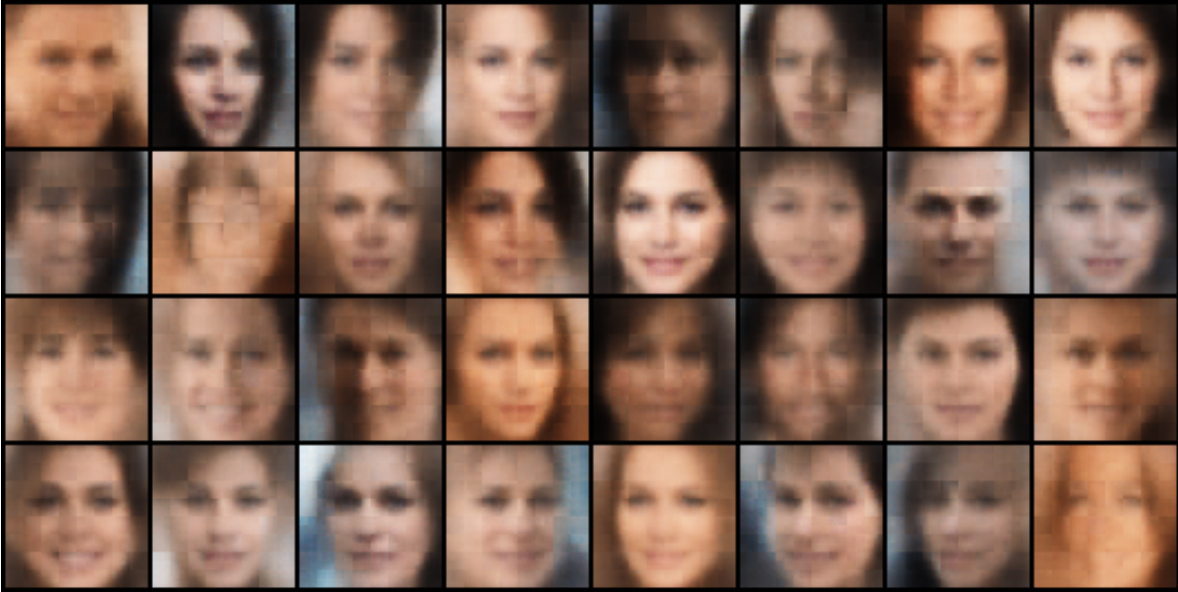


Figure 2: random generated images

1.5 Visualize the latent space

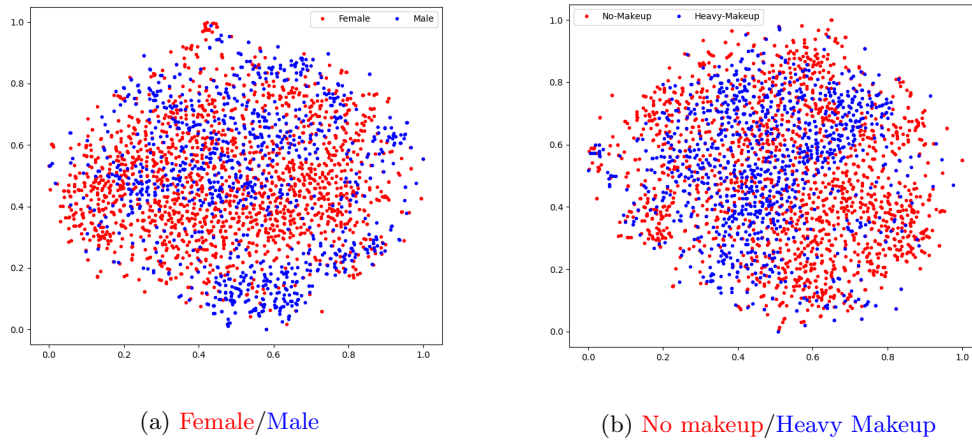


Figure 3: Visualize with different attribute

1.6 Observed and learned from implementing VAE.

I think the most important part of achieving good results in VAE is the value of λ_{KL} . If it is too large, good results cannot be obtained in the reconstructed image. If it is too small, you will not get good performance in randomly generated images, which means it just operates like an autoencoder. And compared with other methods, VAE is easy to implement due to the simplicity of the architecture.

2 GAN

2.1 My implement of GAN

In this problem, I use the architecture proposed in DCGAN[4]. The only difference between my model and DCGAN[4] is that I take 512 as my maximum channel to decrease the computing resource. Also, I choose WGAN[1] with weight clip version as my improved DCGAN[4], which set the clipping value to 0.01 in Discriminator.

Following are some hyperparameters using in the training process.

- Latent dimensional: 100
- Batch size: 64
- Optimizer: RMSprop
- Learning rate: 5e-5
- Training epoch: 1500
- Data preprocess: Normalize each picture to [-1, 1]

```
""" Generator """
Generator(
    (generator): 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=(5, 5), stride=(2, 2), padding=(2, 2),
        output_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=(5, 5), stride=(2, 2), padding=(2, 2),
        output_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=(5, 5), stride=(2, 2), padding=(2, 2),
        output_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=(5, 5), stride=(2, 2), padding=(2, 2),
        output_padding=(1, 1), bias=False)
      (13): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (14): Tanh()
    )
)

""" Discriminator """
Discriminator(
    (discriminator): Sequential(
      (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01, inplace=True)
      (3): Conv2d(64, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), bias=False)
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): LeakyReLU(negative_slope=0.01, inplace=True)
```

```

(6): Conv2d(128, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), bias=False)
(7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(8): LeakyReLU(negative_slope=0.01, inplace=True)
(9): Conv2d(256, 512, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), bias=False)
(10): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(11): LeakyReLU(negative_slope=0.01, inplace=True)
(12): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1))
(13): Flatten()
)
)

```

2.2 Plot 32 random generated images from your model.



Figure 4: Random generated images

2.3 Observed and learned from implementing GAN.

In GAN[3], we need to pay more attention to how to design our architecture compared to VAE. But with the help of DCGAN[4], I think it isn't a big problem for this assignment. Unlike VAE, we have no reconstruction loss to determine whether the model generates a good image. The only method is to justified by a human, so it is difficult for the model to know whether it learns the correct things.

2.4 Compare the difference between image generated by VAE and GAN.

I found that using GAN to generate images seems to have higher resolution and greater diversity. We can see that they are more colorful and have different facial expressions. But there are more artifacts in the image, some of which are really strange compared to the real image.

Using VAE gets some very blurry images. Moreover, it lacks diversity and most of them look similar. But on the other hand, they look more stable, and most of them are similar to human faces.

In my opinion, I like the images generated by GAN better.

3 DANN

3.1 Accuracy(trained on source domain only)

Please refer to Table 2.

3.2 Accuracy(trained on source and domain)

Please refer to Table 2.

3.3 Accuracy(trained on target domain only)

Please refer to Table 2.

Table 2: Accuracy with DANN

	USPS \rightarrow MNIST-M	MNIST-M \rightarrow SVHN	SVHN \rightarrow USPS
Trained on target	95.9%	90.7%	96.4%
Trained on both	42.4%	42.2%	58.2%
Trained on source	17.3%	30.2%	57.2%

3.4 Visualize the latent space

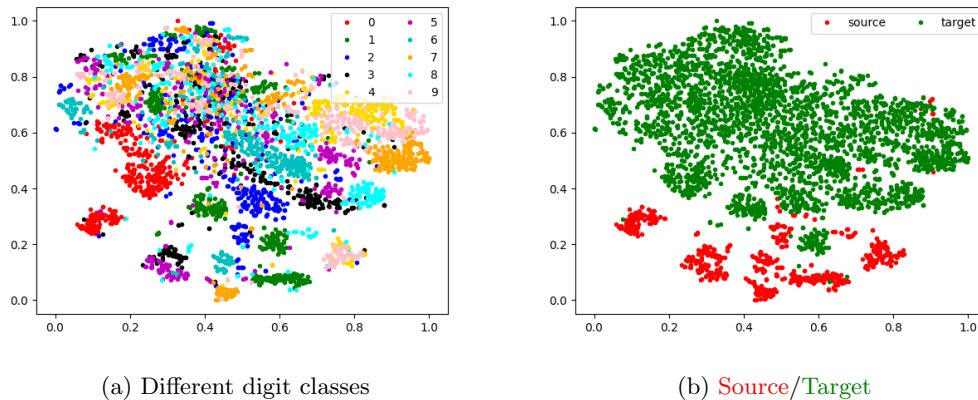


Figure 5: Visualization of USPS \rightarrow MNIST-M

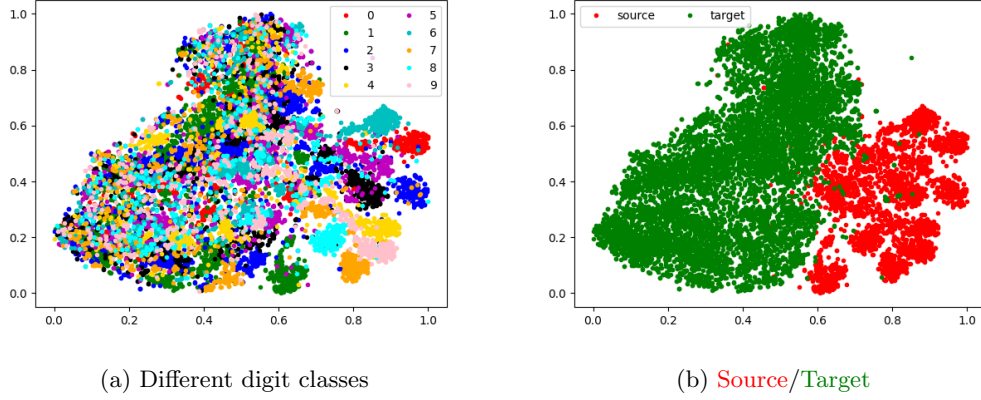


Figure 6: Visualization of MNIST-M \rightarrow SVHN

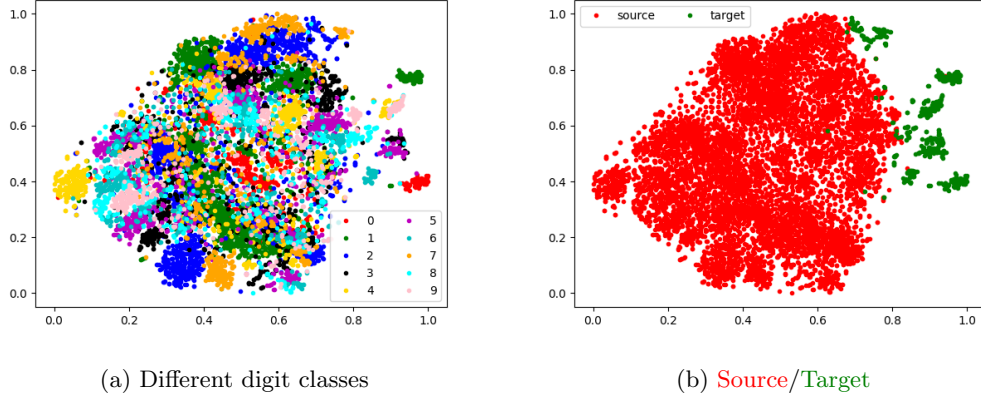


Figure 7: Visualization of SVHN \rightarrow USPS

3.5 My implement of DANN

In the feature extractor, I use 2 Conv2d layers with different input/output channels, and each filter size is set to 5×5 . There are BatchNorm between each of them and LeakyReLU is used as the activation function. I use stride=2 in the convolutional layer instead of pooling.

For label and domain classifier, I simply use two linear layers on both of them. Also, use LeakyReLU as the activation function after the first linear layer.

And I set λ in the GRL layer according to the method proposed in the origin paper DANN[2], which will dynamically be adjusted according to training step.

Following are some hyperparameters using in the training process.

- Batch size: 32
- Optimizer: Adam

- Learning rate: $1e-4$
- Training epoch: 30
- Data preprocess: Normalize each picture to $[0, 1]$

```

DANN(
  (feature_extractor): Sequential(
    (0): Conv2d(3, 32, kernel_size=(5, 5), stride=(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): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): LeakyReLU(negative_slope=0.01, inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (label_classifier): Sequential(
    (0): Linear(in_features=1024, out_features=256, bias=True)
    (1): LeakyReLU(negative_slope=0.01, inplace=True)
    (2): Linear(in_features=256, out_features=10, bias=True)
  )
  (domain_classifier): Sequential(
    (0): Linear(in_features=1024, out_features=256, bias=True)
    (1): LeakyReLU(negative_slope=0.01, inplace=True)
    (2): Linear(in_features=256, out_features=1, bias=True)
  )
)

```

3.6 Observed and learned from implementing DANN.

I think training the UDA model has a disadvantage. The performance of this kind of model strongly depends on the source data and target data, included but not limited to data amount and data feature.

This homework has three different tasks, some of them are easy to pass through the baseline, but some of them require changes to the architecture of the feature extractor and classifier.

And I found that in DANN[2], λ is a very important hyperparameter, which will seriously affect performance. If it is too small, it cannot perform well in target data. On the contrary, it even can't perform well in source data. Therefore, it is a good choice to dynamically adjust it during training.

4 Improved UDA

4.1 MCD Accuracy compare to DANN

Table 3: Accuracy with MCD

	USPS \rightarrow MNIST-M	MNIST-M \rightarrow SVHN	SVHN \rightarrow USPS
DANN[2]	42.4%	42.2%	58.2%
MCD[5]	49.0% (+6.6%)	45.2% (+3.0%)	66.3% (+8.1%)

4.2 Visualize the latent space

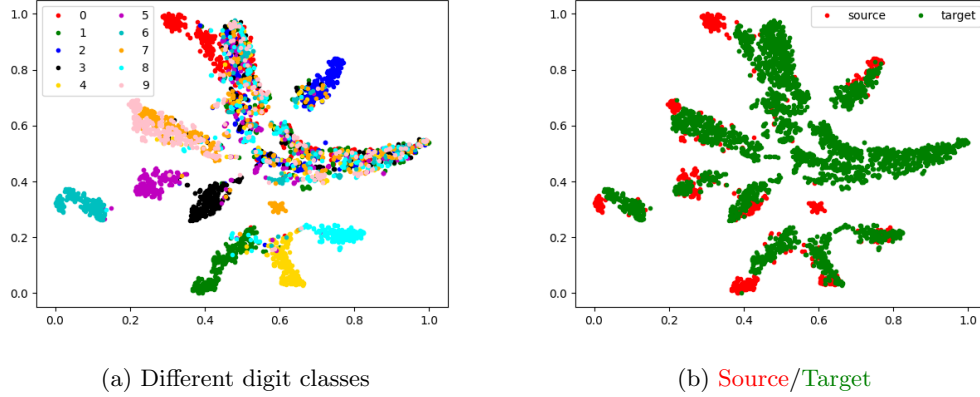


Figure 8: Visualization of USPS \rightarrow MNIST-M

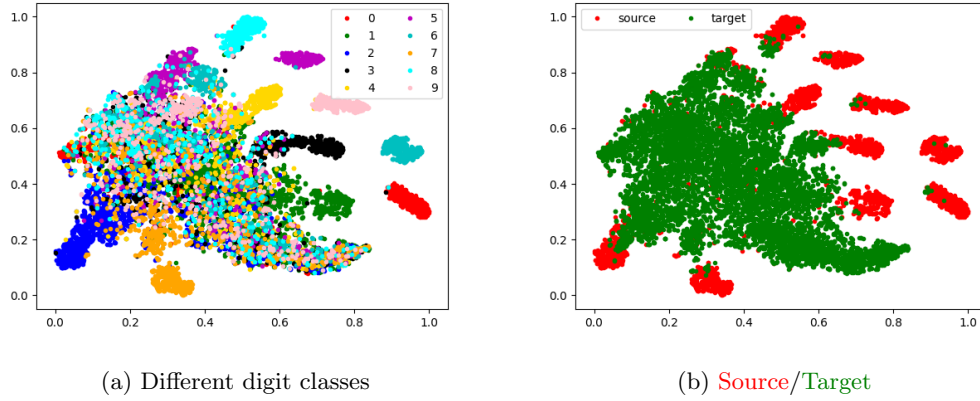


Figure 9: Visualization of MNIST-M \rightarrow SVHN

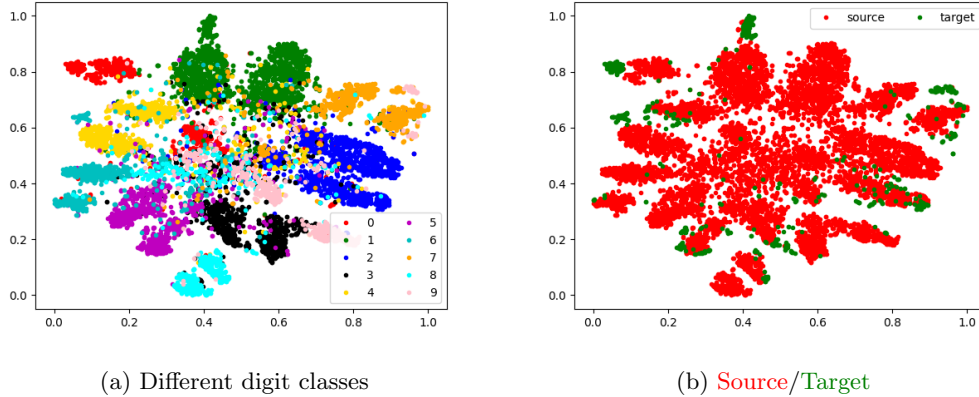


Figure 10: Visualization of SVHN → USPS

4.3 My implement of MCD

I use Maximum Classifier Discrepancy for Unsupervised Domain Adaptation[5](MCD) as my improved model. There are two modules in the method, the first is generator and the second is classifier.

In the feature extractor(or called generator in the paper), I use 3 Conv2d layers with different input/output channels, and each filter size is set to 5×5 . There are BatchNorm between each of them and ReLU is used as the activation function, following is a max-pooling. In the final stage, it will use a linear layer to transfer the feature into the latent space.

In the classifier, I simply use two linear layers. Also, use ReLU as the activation function after the first linear layer. During the training process, it will use two classifiers with the same architecture.

Following are some hyperparameters using in the training process.

- Batch size: 64
- Optimizer: RMSprop
- Learning rate: 5e-4
- Training epoch: 30
- Data preprocess: Normalize each picture to $[-1, 1]$, and resize to (32, 32)

""" Feature Extractor """

```
Feature(
  (feature): Sequential(
    (0): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1), bias=False)
    (1): BatchNorm2d(32, 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): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), bias=False)
    (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=0, dilation=1, ceil_mode=False)
    (8): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1), bias=False)
```

```

(9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(10): ReLU(inplace=True)
(11): Flatten()
(12): Linear(in_features=128, out_features=256, bias=True)
(13): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(14): ReLU(inplace=True)
(15): Dropout(p=0.5, inplace=False)
)
)

""" Classifier """
Classifier(
  (classifier): Sequential(
    (0): Linear(in_features=256, 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): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=256, out_features=10, bias=True)
  )
)

```

4.4 Observed and learned from implementing MCD

The novel idea they proposed in this paper is Discrepancy Loss. This model isn't a GAN-like model, so it is easier to train and also easier to implement compared to other UDA methods such as GTA[6], which I found out has bad performance on MNIST-M \rightarrow SVHN task.

And I found out that in the domain visualization figures(figs. 8b, 9b and 10b), the different domain is closer compare to which was in previous method(figs. 5b, 6b and 7b). I think that is the reason why MCD[5] has a better performance compare to DANN[2].

5 Collaboration

I had discussed this assignment with Tzu-Hsuan Weng(R08922144), Jie Ting(R08922130), Austin Tsai(R08922086).

References

- [1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.
- [2] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.
- [3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [4] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

- [5] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum classifier discrepancy for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3723–3732, 2018.
- [6] Swami Sankaranarayanan, Yogesh Balaji, Carlos D Castillo, and Rama Chellappa. Generate to adapt: Aligning domains using generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8503–8512, 2018.