HW3
陳芃彣 R08942085

Problem1: VAE

1. Print model

```
VAE(
  (encoder): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 32, kernel_size=(3, 3), 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)
    )
    (1): Sequential(
      (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (2): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (3): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), 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=0.01)
    )
    (4): Sequential(
      (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
  )
  (fc_mu): Linear(in_features=2048, out_features=512, bias=True)
  (fc_var): Linear(in_features=2048, out_features=512, bias=True)
  (decode_input): Linear(in_features=512, out_features=2048, bias=True)
  (decoder): Sequential(
    (0): Sequential(
      (0): ConvTranspose2d(512, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (1): Sequential(
      (0): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (2): Sequential(
      (0): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (3): Sequential(
      (0): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (4): Sequential(
      (0): ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
  )
  (final_layer): Sequential(
    (0): Conv2d(32, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): Tanh()
  )
)
```

Encoder

Decoder

根據 Vanilla VAE ，我主要搭建了一個 Encoder 及 一個 Decoder，Encoder 用了五個 encoder block 組合的，Decoder 亦由五個 decoder block 組合，每個 encoder block 皆由一層 CNN＋

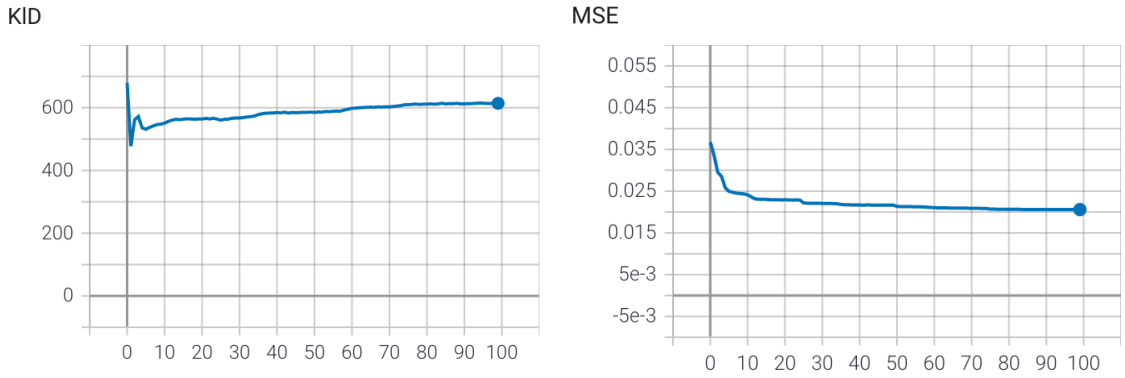
BatchNormalize＋LeakyReLU 組合， decoder block 則是由一層 CNN Transpose＋BatchNormalize＋LeakyReLU。

Encoder 的 output 會經由兩個 Linear Layer 得到 mu 和 var，目的為使用平均值與標準差從normal distribution sample 出一個 z vector，然後我有將 z 再過一層 Linear Layer 作為 Decoder 的 input。

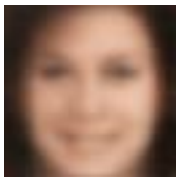
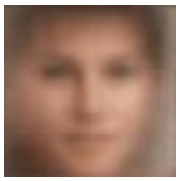
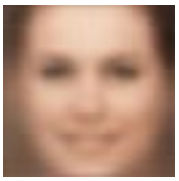
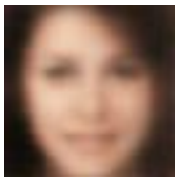
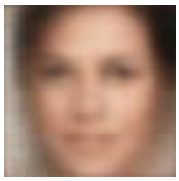
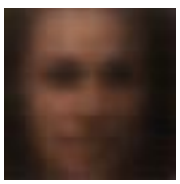
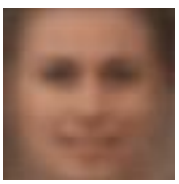
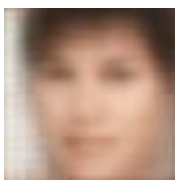
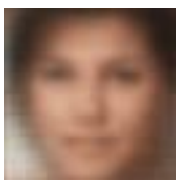
最後 Decoder 的 output 再過一層 CNN + Tanh 組合的 Final Layer。

Ref: [3] https://github.com/AntixK/PyTorch-VAE

2. Learning curve of KLD Loss and Reconstruction Loss

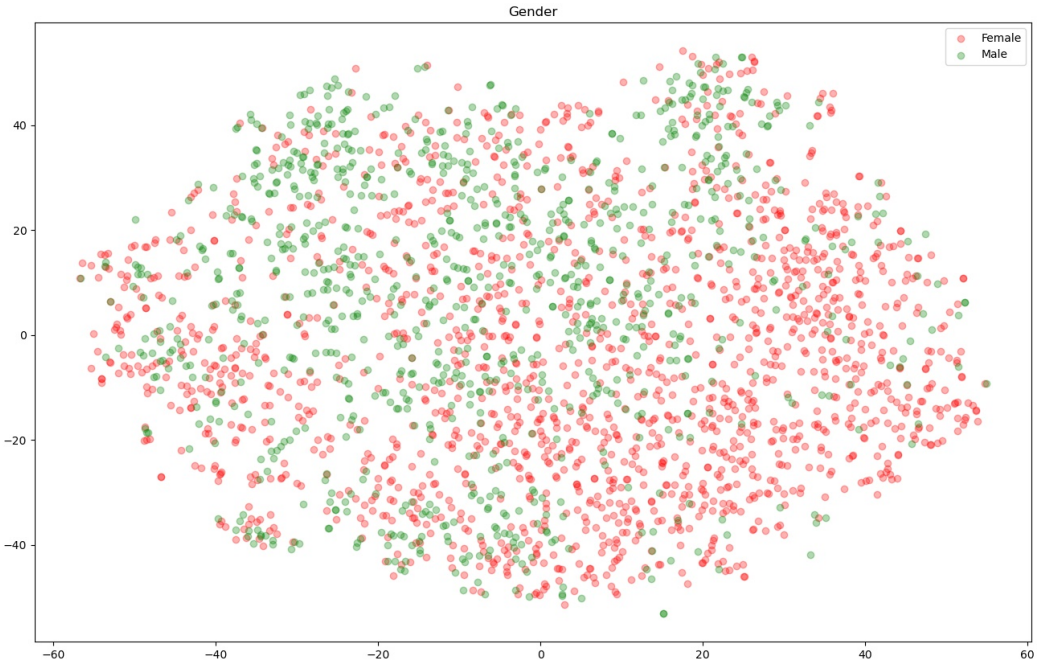


3. Reconstruct 10 testing images

Testing Image					
Recon. Image					
MSE	0.01529	0.02843	0.03939	0.02197	0.02013
Testing Image					
Recon. Image					
MSE	0.04402	0.00601	0.00964	0.01392	0.01775

4. Generate

5. TSNE: Gender



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

在實作過後發現 KLD Loss 與 MSE Loss 很難兩者兼顧，以本次作業為例，KLD Loss 我乘上了 1e-5 的權重才有確保 MSE Loss 可以降到大約 0.025，但是可以看到我的 KLD Loss 並沒有很低 （大約七百多），所以照片不管是重建還是生成，畫質都沒有很好，tsne 的話，可以算免強有分成男生與女生，其中女生分得比較好，男生比較分散。

Problem2 GAN

1. Print Model

Discriminator architecture

```
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()
  )
)
```

Generator architecture

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

Data: normalize to 0~1

Training: (DCGAN)

- Set 1 for real label, and 0 for fake label.
- Weight init both on Discriminator and Generator form a normal distribution with mean=0, stdev=0.2.
- Train the Discriminator first:
  - For Real: Feed the real image to the Discriminator, and calculate loss.
  - For Fake: Sample noise by torch.randn from normal distribution, and generate a fake image by the Generator. Feed the fake image to the Discriminator and calculate loss. Then, add the real loss and fake loss and then update once.
- Train the Generator:

  - Obtain the output of Discriminator that fed with the fake image, which generated from the Generator. Then, update the Generator once.
- Loss function: nn.BCELoss()
- Ref: [2] https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

2. Generate

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

- DCGAN can do better on image than vanilla GAN since it contains convolutional and transpose convolutional layers in Discriminator and Generator.
- When training a GAN, it is better to train the Discriminator first, since it is the Discriminator that help the Generator to learn to yield fake image.
- The Generator first learn to change the color, and then senses of a human face, such as eyes, nose and mouth. Finally the contour of a face. Moreover, we can see facial expression comes out after 20k iterations.

4. Compare the difference between image generated by VAE and GAN, discuss what you've observed.

- For updating, the generator in GAN updates according to the output of the discriminator. However, the VAE model updates its decoder (like generator) directly from the data.
- The human faces are more diverse generated by GAN, such as gender, facial expression, hair and skin color. But the outputs of VAE are mostly have the same facial expression.
- The contour of the outputs from GAN are more obvious than those from VAE.
- The outputs from VAE are more natural and stable. But in GAN, some of the output might be weird, which looks like people with a fail plastic surgery.

Problem3 DANN

1. **Lower bound:**

	usps -> mnist-m	mnist-m -> svhn	svhn -> usps
Accuracy	29.92	44.48	39.66

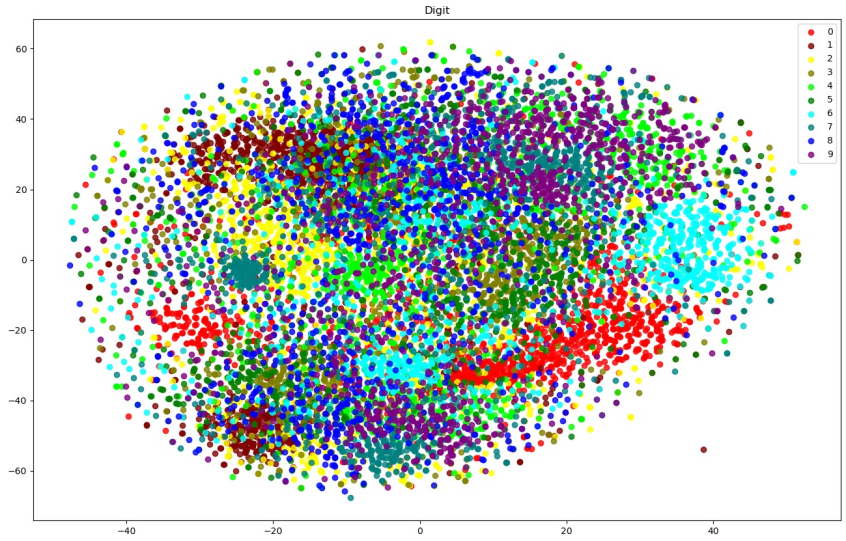
2. **DANN:**

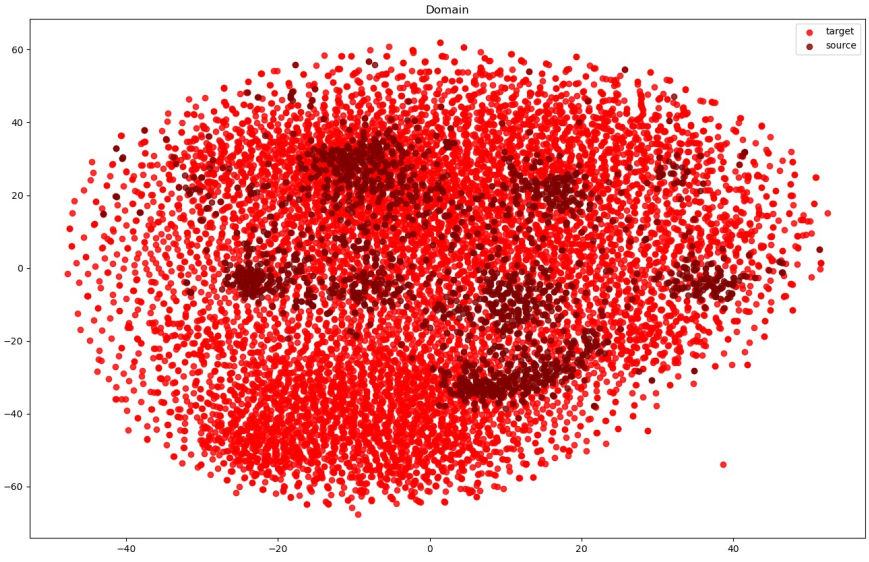
	usps -> mnist-m	mnist-m -> svhn	svhn -> usps
Accuracy	42.14	53.33	56.15

3. **Upper bound:**

	usps -> mnist-m	mnist-m -> svhn	svhn -> usps
Accuracy	89.71	85.84	94.76

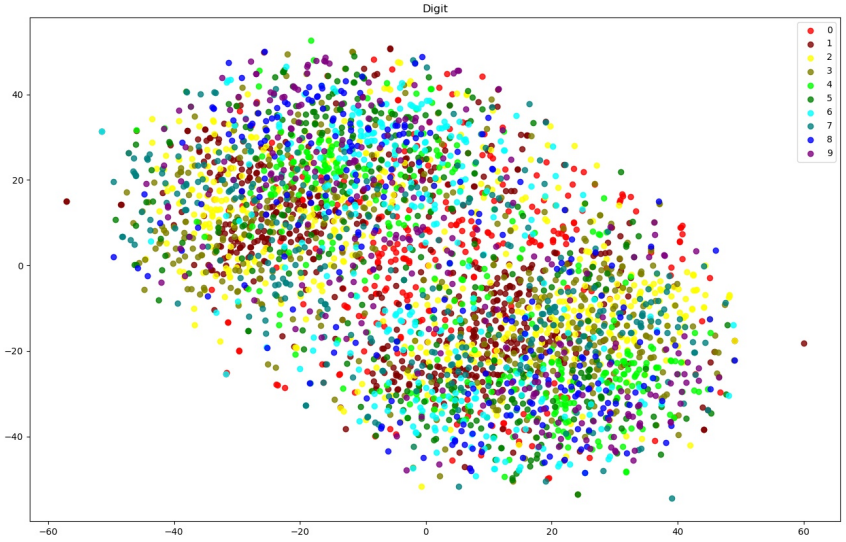
4. **TSNE:**
   - **usps -> mnistm**
     (Since the number of data of mnistm is larger than that of usps, the domain tsne is very unbalanced)

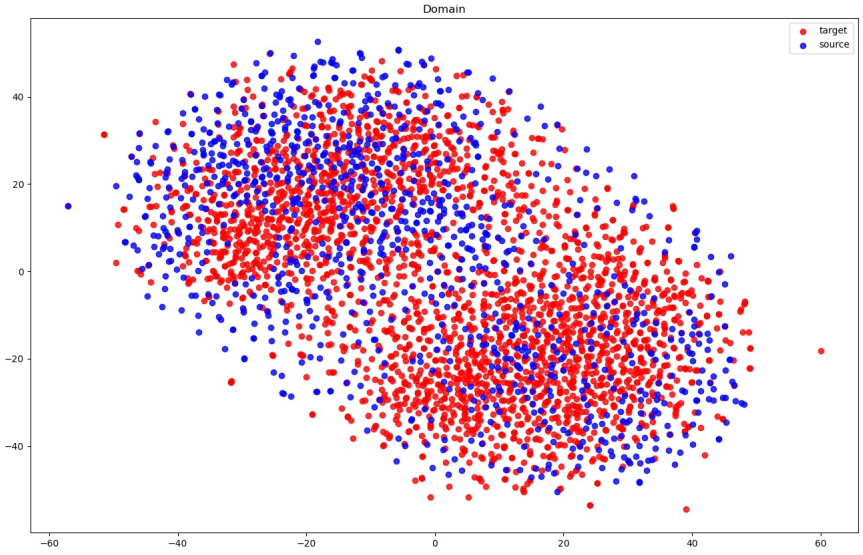




- minst-m -> svhn

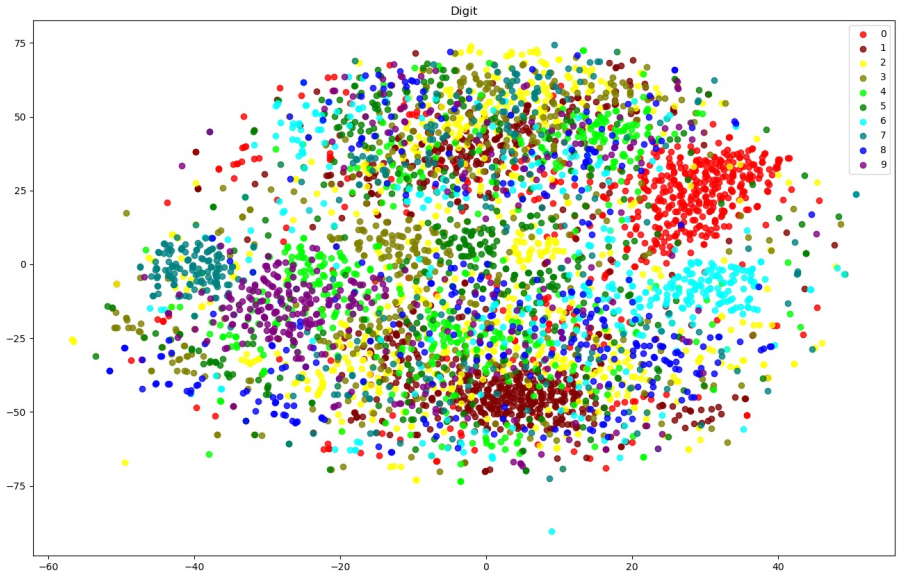
Since there are too many data in svhn and mnistm, I sampled the point for every 2560 steps.

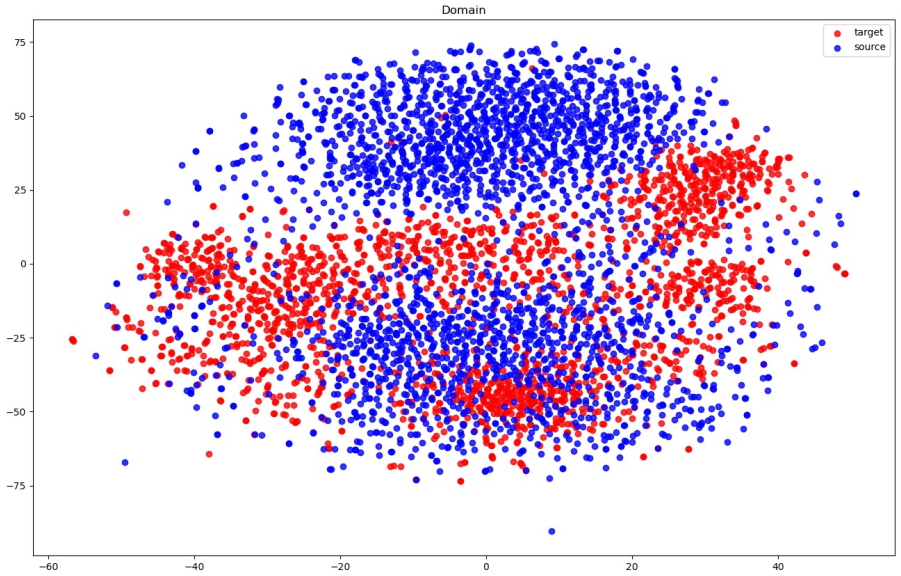




- svhn -> usps

Since there are too many data in svhn, I sampled the point of svhn for every 2560 steps.





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

I referred the model architecture from the GitHub[3]. The model consists of a Feature Extractor (CNN + BN + MaxPooling + ReLU + Dropout), a Label Classifier (FC + BN + ReLU + Dropout + Softmax at the end), and a Domain Classifier (FC + BN + ReLU + Softmax at the end).

```
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=True)
    (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, inplace=False)
    (f_pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (f_relu2): ReLU(inplace=True)
  )
  (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=True)
    (c_drop1): Dropout(p=0.5, inplace=False)
    (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=True)
    (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=True)
    (d_fc2): Linear(in_features=100, out_features=2, bias=True)
    (d_softmax): LogSoftmax()
  )
)
```

For the training, I have had tried to train the domain classifier first, but it would fail on the set of “usps -> mnist-m”, which didn’t pass the baseline score. Thus, I changed to train the whole model once by adding the loss functions (loss on target domain, loss on source domain, loss on source label) and updated the whole model once per step.

The weight that multiply on the reverse layer was zero at the first step, and it would get larger after a step and finally increased to one.

- Loss function: NLLLoss

Ref: [3] https://github.com/fungtion/DANN/blob/master/models/model.py

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

- DANN is not as stable as we think, since it’s sensitive to data. Take the three dataset as example. The set “usps - > mnistm ” is hard to be trained since the image of the two datasets have different channels, so we should pad the channels of a gray-scale image before training.
- It might be normal that we have a sense to train the domain classifier first, since we update the feature extractor base on the domain classifier result. However, it might not fit to all kinds of data, like the case “usps -> mnistm“.

Problem4 Improved UDA

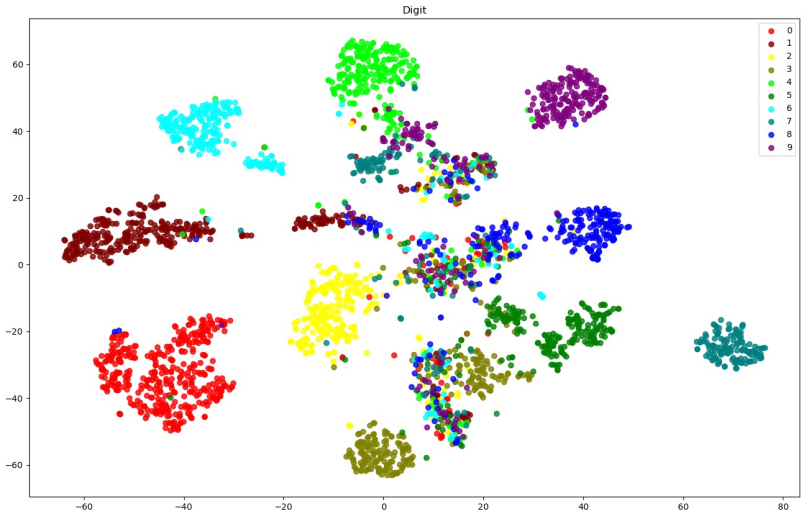
1. Improved UDA: Generate To Adapt

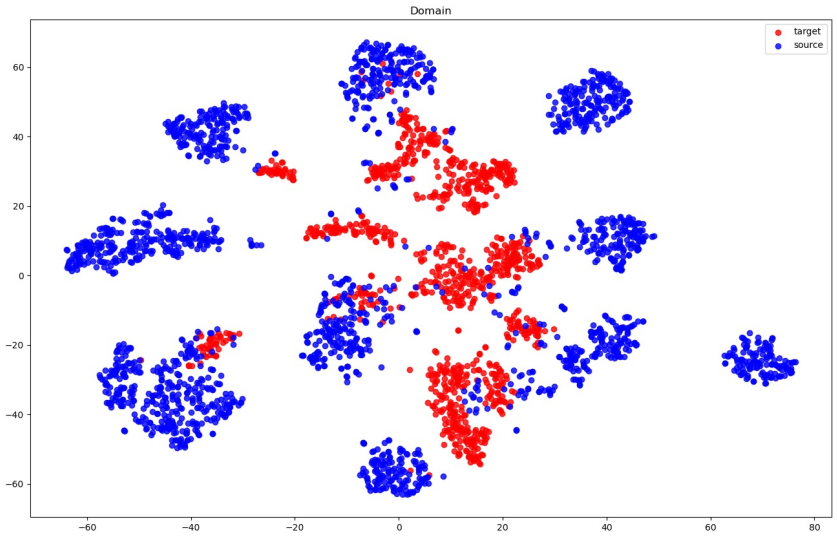
	usps -> mnist-m	mnist-m -> svhn	svhn -> usps
(DANN) Accuracy	42.14	53.33	56.15
(GTA) Accuracy	**48.05**	**61.45**	**63.22**

2. TSNE

- usps -> mnist-m

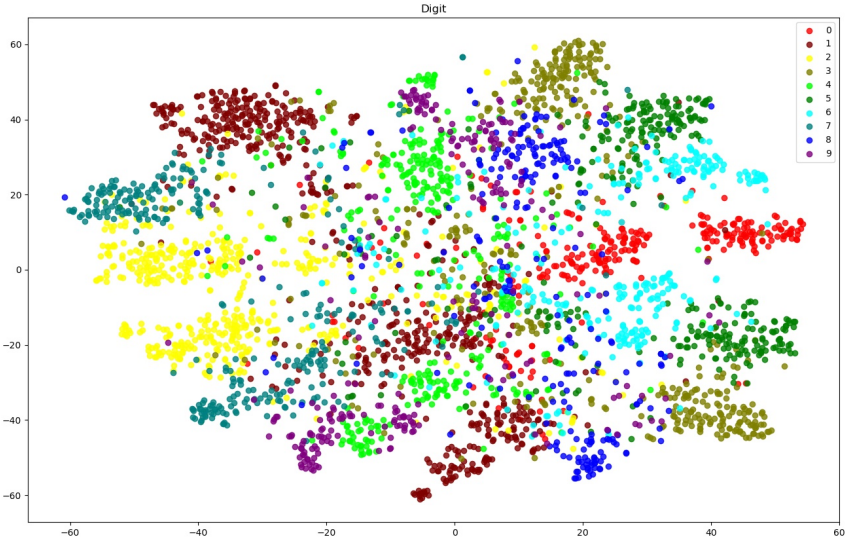
Since there are too many data in mnistm, I sampled the mnistm point for every 2560 steps.

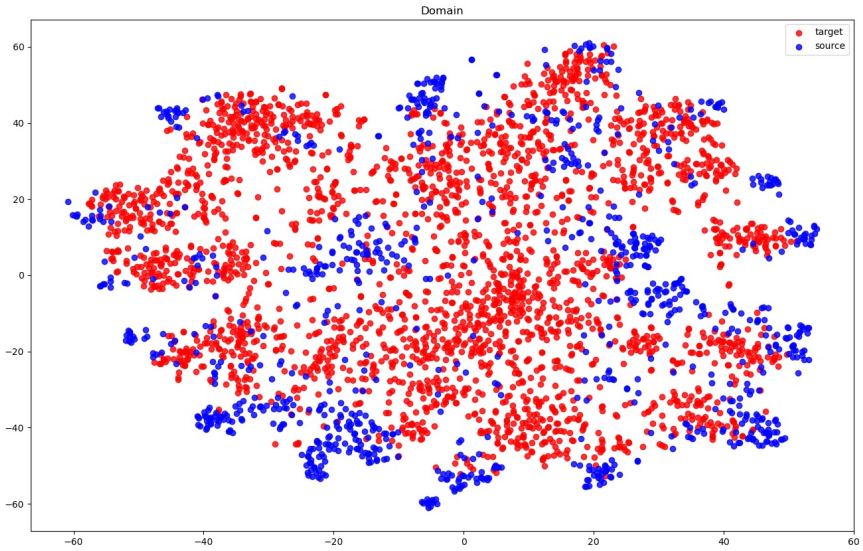




- **mnist-m ->svhn**

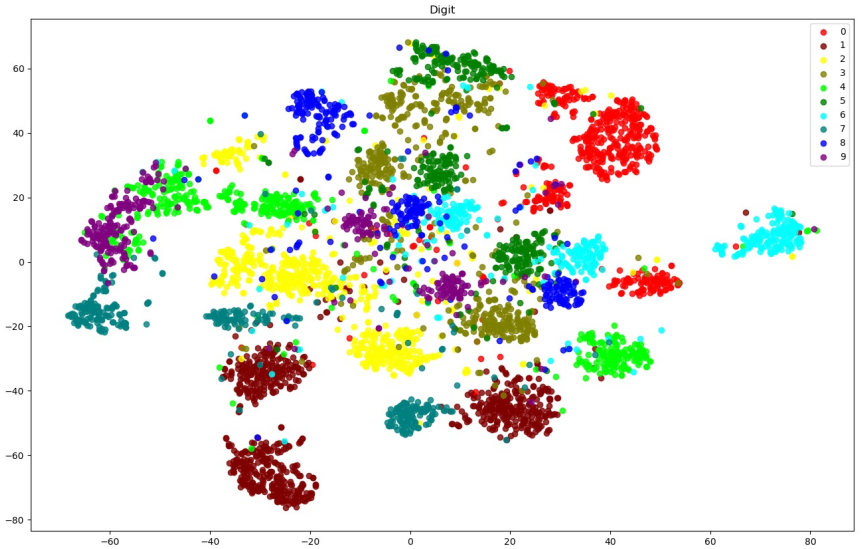
  Since there are too many data in svhn and mnistm, I sampled the point for every 2560 steps.

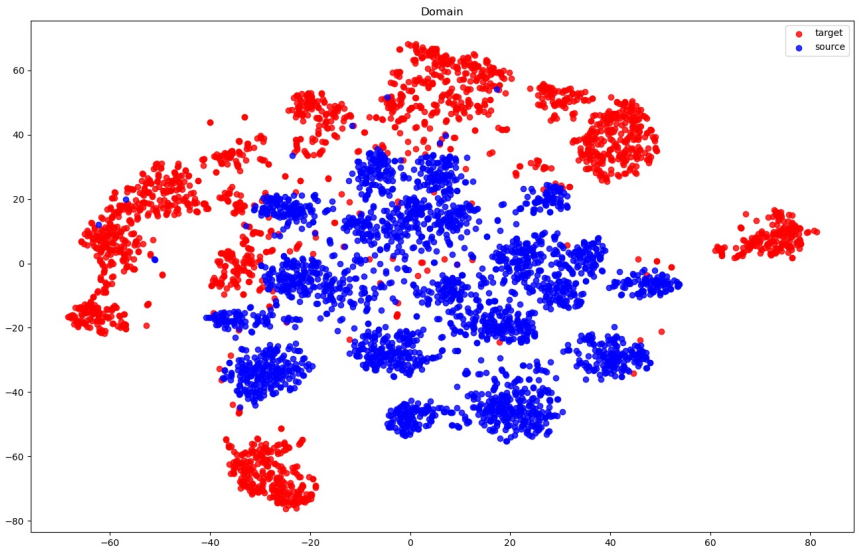




- **svhn -> usps**

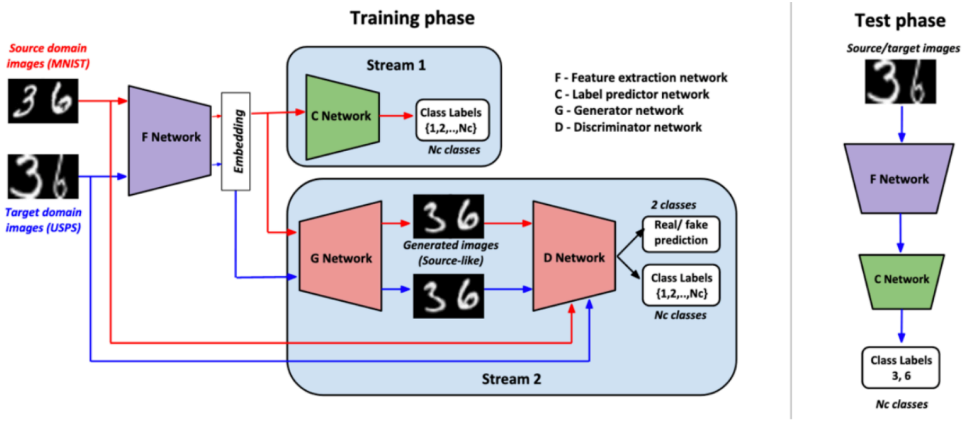
  Since there are too many data in svhn, I sampled the point of svhn for every 2560 steps.





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

- The main architecture from GTA:



- Comparing to the model architecture of DANN, GTA replace the naive domain classifier to a GAN like model. Thus, the architecture consists of a Feature Extractor, a Label Classifier and a GAN Domain Classifier.

```
_netF(
  (feature): Sequential(
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1))
    (7): ReLU(inplace=True)
  )
)
_netC(
  (main): Sequential(
    (0): Linear(in_features=128, out_features=128, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=128, out_features=10, bias=True)
  )
)
_netG(
  (main): Sequential(
    (0): ConvTranspose2d(651, 512, kernel_size=(2, 2), 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=(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=True)
    (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=True)
    (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=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

```
_netD(
  (feature): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): LeakyReLU(negative_slope=0.2, inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): LeakyReLU(negative_slope=0.2, inplace=True)
    (15): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier_c): Sequential(
    (0): Linear(in_features=128, out_features=10, bias=True)
  )
  (classifier_s): Sequential(
    (0): Linear(in_features=128, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

- The GAN Domain Classifier not only classify real/fake images (or domain), but also the labels of the image.
- Data:
  Since the GTA model is sensitive to data, when training on the set that both source and target images are 3 channels (“mnist-m -> svhn“), I first change the data to gray scale image, and then pad the channels back to 3 after resizing.
  For the set that source or target image have different channels, I resize and normalize both the source and target data.
- Training
  - I trained the Discriminator first. Thus, extract the features of both source and target images from the Feature Extractor, and then generate fake images by the generator. Feed both the real images and fake images to the Discriminator and update once.
  - Update the Generator by the result from the Discriminator with fake images.
  - Update the Label Classifier by the features from the Feature Extractor.
  - Finally, update the Feature Extractor by the result from both the Label Classifier and the Discriminator.

Ref: [4] https://github.com/yogeshbalaji/Generate_To_Adapt

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

- GTA is not stable because it is sensitive to data. Different from DANN, GTA is more sensitive on “mnist-m -> svhn”, which is the only set that have both 3 channels in source data and target data. Thus, I first convert both of then into gray-scale image, and padded them back to 3 channels after resizing.
- GTA is not stable. To be specific, we might get different score after training the same number of steps. Therefore, it is better to train more times on GTA.
- Although GTA have a better score than that of DANN, it only surpass little on the accuracy.