tags: `dlcv`

HW2

P1

```
StyleGAN2(
  (S): StyleVectorizer(
    (net): Sequential(
      (0): EqualLinear()
      (1): LeakyReLU(negative_slope=0.2, inplace=True)
      (2): EqualLinear()
      (3): LeakyReLU(negative_slope=0.2, inplace=True)
      (4): EqualLinear()
      (5): LeakyReLU(negative_slope=0.2, inplace=True)
      (6): EqualLinear()
      (7): LeakyReLU(negative_slope=0.2, inplace=True)
      (8): EqualLinear()
      (9): LeakyReLU(negative_slope=0.2, inplace=True)
      (10): EqualLinear()
      (11): LeakyReLU(negative_slope=0.2, inplace=True)
      (12): EqualLinear()
      (13): LeakyReLU(negative_slope=0.2, inplace=True)
      (14): EqualLinear()
      (15): LeakyReLU(negative_slope=0.2, inplace=True)
    )
  )
  (G): Generator(
    (initial_conv): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
    (blocks): ModuleList(
      (0): GeneratorBlock(
        (to_style1): Linear(in_features=512, out_features=512, bias=True)
        (to_noise1): Linear(in_features=1, out_features=512, bias=True)
        (conv1): Conv2DMod()
        (to_style2): Linear(in_features=512, out_features=512, bias=True)
        (to_noise2): Linear(in_features=1, out_features=512, bias=True)
        (conv2): Conv2DMod()
        (activation): LeakyReLU(negative_slope=0.2, inplace=True)
        (to_rgb): RGBBlock(
          (to_style): Linear(in_features=512, out_features=512, bias=True)
          (conv): Conv2DMod()
          (upsample): Sequential(
            (0): Upsample(scale_factor=2.0, mode=bilinear)
            (1): Blur()
          )
        )
      )
      (1): GeneratorBlock(
        (upsample): Upsample(scale_factor=2.0, mode=bilinear)
        (to_style1): Linear(in_features=512, out_features=512, bias=True)
        (to_noise1): Linear(in_features=1, out_features=256, bias=True)
        (conv1): Conv2DMod()
        (to_style2): Linear(in_features=512, out_features=256, bias=True)
        (to_noise2): Linear(in_features=1, out_features=256, bias=True)
        (conv2): Conv2DMod()
        (activation): LeakyReLU(negative_slope=0.2, inplace=True)
        (to_rgb): RGBBlock(
          (to_style): Linear(in_features=512, out_features=256, bias=True)
```

```
      (conv): Conv2DMod()
      (upsample): Sequential(
        (0): Upsample(scale_factor=2.0, mode=bilinear)
        (1): Blur()
      )
    )
  )
  (2): GeneratorBlock(
    (upsample): Upsample(scale_factor=2.0, mode=bilinear)
    (to_style1): Linear(in_features=512, out_features=256, bias=True)
    (to_noise1): Linear(in_features=1, out_features=128, bias=True)
    (conv1): Conv2DMod()
    (to_style2): Linear(in_features=512, out_features=128, bias=True)
    (to_noise2): Linear(in_features=1, out_features=128, bias=True)
    (conv2): Conv2DMod()
    (activation): LeakyReLU(negative_slope=0.2, inplace=True)
    (to_rgb): RGBBlock(
      (to_style): Linear(in_features=512, out_features=128, bias=True)
      (conv): Conv2DMod()
      (upsample): Sequential(
        (0): Upsample(scale_factor=2.0, mode=bilinear)
        (1): Blur()
      )
    )
  )
  (3): GeneratorBlock(
    (upsample): Upsample(scale_factor=2.0, mode=bilinear)
    (to_style1): Linear(in_features=512, out_features=128, bias=True)
    (to_noise1): Linear(in_features=1, out_features=64, bias=True)
    (conv1): Conv2DMod()
    (to_style2): Linear(in_features=512, out_features=64, bias=True)
    (to_noise2): Linear(in_features=1, out_features=64, bias=True)
    (conv2): Conv2DMod()
    (activation): LeakyReLU(negative_slope=0.2, inplace=True)
    (to_rgb): RGBBlock(
      (to_style): Linear(in_features=512, out_features=64, bias=True)
      (conv): Conv2DMod()
      (upsample): Sequential(
        (0): Upsample(scale_factor=2.0, mode=bilinear)
        (1): Blur()
      )
    )
  )
  (4): GeneratorBlock(
    (upsample): Upsample(scale_factor=2.0, mode=bilinear)
    (to_style1): Linear(in_features=512, out_features=64, bias=True)
    (to_noise1): Linear(in_features=1, out_features=32, bias=True)
    (conv1): Conv2DMod()
    (to_style2): Linear(in_features=512, out_features=32, bias=True)
    (to_noise2): Linear(in_features=1, out_features=32, bias=True)
    (conv2): Conv2DMod()
    (activation): LeakyReLU(negative_slope=0.2, inplace=True)
    (to_rgb): RGBBlock(
```

```
          (to_style): Linear(in_features=512, out_features=32, bias=True)
          (conv): Conv2DMod()
        )
      )
    )
    (attns): ModuleList(
      (0): None
      (1): None
      (2): None
      (3): Sequential(
        (0): Residual(
          (fn): PreNorm(
            (fn): LinearAttention(
              (nonlin): GELU()
              (to_q): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=
              (to_kv): DepthWiseConv2d(
                (net): Sequential(
                  (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padd
                  (1): Conv2d(128, 1024, kernel_size=(1, 1), stride=(1, 1), bia
                )
              )
              (to_out): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1))
            )
            (norm): ChanNorm()
          )
        )
        (1): Residual(
          (fn): PreNorm(
            (fn): Sequential(
              (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
              (1): LeakyReLU(negative_slope=0.2, inplace=True)
              (2): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
            )
            (norm): ChanNorm()
          )
        )
      )
      (4): Sequential(
        (0): Residual(
          (fn): PreNorm(
            (fn): LinearAttention(
              (nonlin): GELU()
              (to_q): Conv2d(64, 512, kernel_size=(1, 1), stride=(1, 1), bias=F
              (to_kv): DepthWiseConv2d(
                (net): Sequential(
                  (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), paddin
                  (1): Conv2d(64, 1024, kernel_size=(1, 1), stride=(1, 1), bias
                )
              )
              (to_out): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1))
            )
            (norm): ChanNorm()
          )
```

```
          )
          (1): Residual(
            (fn): PreNorm(
              (fn): Sequential(
                (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1))
                (1): LeakyReLU(negative_slope=0.2, inplace=True)
                (2): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1))
              )
              (norm): ChanNorm()
            )
          )
        )
      )
      (SE): StyleVectorizer(
        (net): Sequential(
          (0): EqualLinear()
          (1): LeakyReLU(negative_slope=0.2, inplace=True)
          (2): EqualLinear()
          (3): LeakyReLU(negative_slope=0.2, inplace=True)
          (4): EqualLinear()
          (5): LeakyReLU(negative_slope=0.2, inplace=True)
          (6): EqualLinear()
          (7): LeakyReLU(negative_slope=0.2, inplace=True)
          (8): EqualLinear()
          (9): LeakyReLU(negative_slope=0.2, inplace=True)
          (10): EqualLinear()
          (11): LeakyReLU(negative_slope=0.2, inplace=True)
          (12): EqualLinear()
          (13): LeakyReLU(negative_slope=0.2, inplace=True)
          (14): EqualLinear()
          (15): LeakyReLU(negative_slope=0.2, inplace=True)
        )
      )
      (GE): Generator(
        (initial_conv): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
        (blocks): ModuleList(
          (0): GeneratorBlock(
            (to_style1): Linear(in_features=512, out_features=512, bias=True)
            (to_noise1): Linear(in_features=1, out_features=512, bias=True)
            (conv1): Conv2DMod()
            (to_style2): Linear(in_features=512, out_features=512, bias=True)
            (to_noise2): Linear(in_features=1, out_features=512, bias=True)
            (conv2): Conv2DMod()
            (activation): LeakyReLU(negative_slope=0.2, inplace=True)
            (to_rgb): RGBBlock(
              (to_style): Linear(in_features=512, out_features=512, bias=True)
              (conv): Conv2DMod()
              (upsample): Sequential(
                (0): Upsample(scale_factor=2.0, mode=bilinear)
                (1): Blur()
              )
            )
```

```
    )
    (1): GeneratorBlock(
      (upsample): Upsample(scale_factor=2.0, mode=bilinear)
      (to_style1): Linear(in_features=512, out_features=512, bias=True)
      (to_noise1): Linear(in_features=1, out_features=256, bias=True)
      (conv1): Conv2DMod()
      (to_style2): Linear(in_features=512, out_features=256, bias=True)
      (to_noise2): Linear(in_features=1, out_features=256, bias=True)
      (conv2): Conv2DMod()
      (activation): LeakyReLU(negative_slope=0.2, inplace=True)
      (to_rgb): RGBBlock(
        (to_style): Linear(in_features=512, out_features=256, bias=True)
        (conv): Conv2DMod()
        (upsample): Sequential(
          (0): Upsample(scale_factor=2.0, mode=bilinear)
          (1): Blur()
        )
      )
    )
    (2): GeneratorBlock(
      (upsample): Upsample(scale_factor=2.0, mode=bilinear)
      (to_style1): Linear(in_features=512, out_features=256, bias=True)
      (to_noise1): Linear(in_features=1, out_features=128, bias=True)
      (conv1): Conv2DMod()
      (to_style2): Linear(in_features=512, out_features=128, bias=True)
      (to_noise2): Linear(in_features=1, out_features=128, bias=True)
      (conv2): Conv2DMod()
      (activation): LeakyReLU(negative_slope=0.2, inplace=True)
      (to_rgb): RGBBlock(
        (to_style): Linear(in_features=512, out_features=128, bias=True)
        (conv): Conv2DMod()
        (upsample): Sequential(
          (0): Upsample(scale_factor=2.0, mode=bilinear)
          (1): Blur()
        )
      )
    )
    (3): GeneratorBlock(
      (upsample): Upsample(scale_factor=2.0, mode=bilinear)
      (to_style1): Linear(in_features=512, out_features=128, bias=True)
      (to_noise1): Linear(in_features=1, out_features=64, bias=True)
      (conv1): Conv2DMod()
      (to_style2): Linear(in_features=512, out_features=64, bias=True)
      (to_noise2): Linear(in_features=1, out_features=64, bias=True)
      (conv2): Conv2DMod()
      (activation): LeakyReLU(negative_slope=0.2, inplace=True)
      (to_rgb): RGBBlock(
        (to_style): Linear(in_features=512, out_features=64, bias=True)
        (conv): Conv2DMod()
        (upsample): Sequential(
          (0): Upsample(scale_factor=2.0, mode=bilinear)
          (1): Blur()
        )
```

```
        )
      )
      (4): GeneratorBlock(
        (upsample): Upsample(scale_factor=2.0, mode=bilinear)
        (to_style1): Linear(in_features=512, out_features=64, bias=True)
        (to_noise1): Linear(in_features=1, out_features=32, bias=True)
        (conv1): Conv2DMod()
        (to_style2): Linear(in_features=512, out_features=32, bias=True)
        (to_noise2): Linear(in_features=1, out_features=32, bias=True)
        (conv2): Conv2DMod()
        (activation): LeakyReLU(negative_slope=0.2, inplace=True)
        (to_rgb): RGBBlock(
          (to_style): Linear(in_features=512, out_features=32, bias=True)
          (conv): Conv2DMod()
        )
      )
    )
    (attns): ModuleList(
      (0): None
      (1): None
      (2): None
      (3): Sequential(
        (0): Residual(
          (fn): PreNorm(
            (fn): LinearAttention(
              (nonlin): GELU()
              (to_q): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=
              (to_kv): DepthWiseConv2d(
                (net): Sequential(
                  (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padd
                  (1): Conv2d(128, 1024, kernel_size=(1, 1), stride=(1, 1), bia
                )
              )
              (to_out): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1))
            )
            (norm): ChanNorm()
          )
        )
        (1): Residual(
          (fn): PreNorm(
            (fn): Sequential(
              (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
              (1): LeakyReLU(negative_slope=0.2, inplace=True)
              (2): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
            )
            (norm): ChanNorm()
          )
        )
      )
      (4): Sequential(
        (0): Residual(
          (fn): PreNorm(
            (fn): LinearAttention(
```

```
                (nonlin): GELU()
                (to_q): Conv2d(64, 512, kernel_size=(1, 1), stride=(1, 1), bias=F
                (to_kv): DepthWiseConv2d(
                  (net): Sequential(
                    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), paddin
                    (1): Conv2d(64, 1024, kernel_size=(1, 1), stride=(1, 1), bias
                  )
                )
                (to_out): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1))
              )
              (norm): ChanNorm()
            )
          )
          (1): Residual(
            (fn): PreNorm(
              (fn): Sequential(
                (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1))
                (1): LeakyReLU(negative_slope=0.2, inplace=True)
                (2): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1))
              )
              (norm): ChanNorm()
            )
          )
        )
      )
    )
  )
```

I used stylegan2 with 2 attention layers, and trained 150000 steps. However the checkpoints around 8200~9200 steps generate best images.

2.

3~4.

IS:2.1766
FID:27.3224

5.

I tried DCGAN, WGAN-GP and SNGAN. SNGAN > WGAN-GP > DCGAN, but stylegan is the best.
In stylegan, when generating images, the trunc_psi should be closer to 1.0 to have better image diversity.

P2

1.

I built a generator and discriminator where both of them have convolutional layers. And my discriminator will return 2 results, label results and validity results. The image should be resized to 32, otherwise the accuracy will be low.

```
Generator(
  (label_emb): Embedding(10, 100)
  (l1): Sequential(
    (0): Linear(in_features=100, out_features=8192, bias=True)
  )
  (conv_blocks): Sequential(
    (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
    (1): Upsample(scale_factor=2.0, mode=nearest)

    (2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_sta
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Upsample(scale_factor=2.0, mode=nearest)
    (6): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_stat
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Conv2d(64, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): Tanh()
  )
)
```

```
Discriminator(
  (conv_blocks): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Dropout2d(p=0.25, inplace=False)
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Dropout2d(p=0.25, inplace=False)
    (6): BatchNorm2d(32, eps=0.8, momentum=0.1, affine=True, track_running_stat
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Dropout2d(p=0.25, inplace=False)
    (10): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_sta
    (11): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (12): LeakyReLU(negative_slope=0.2, inplace=True)
    (13): Dropout2d(p=0.25, inplace=False)
    (14): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_st
  )
  (adv_layer): Sequential(
    (0): Linear(in_features=512, out_features=1, bias=True)
    (1): Sigmoid()
  )
  (aux_layer): Sequential(
    (0): Linear(in_features=512, out_features=10, bias=True)
    (1): Softmax(dim=0)
  )
)
```

2~4.

0.95

5.

P3

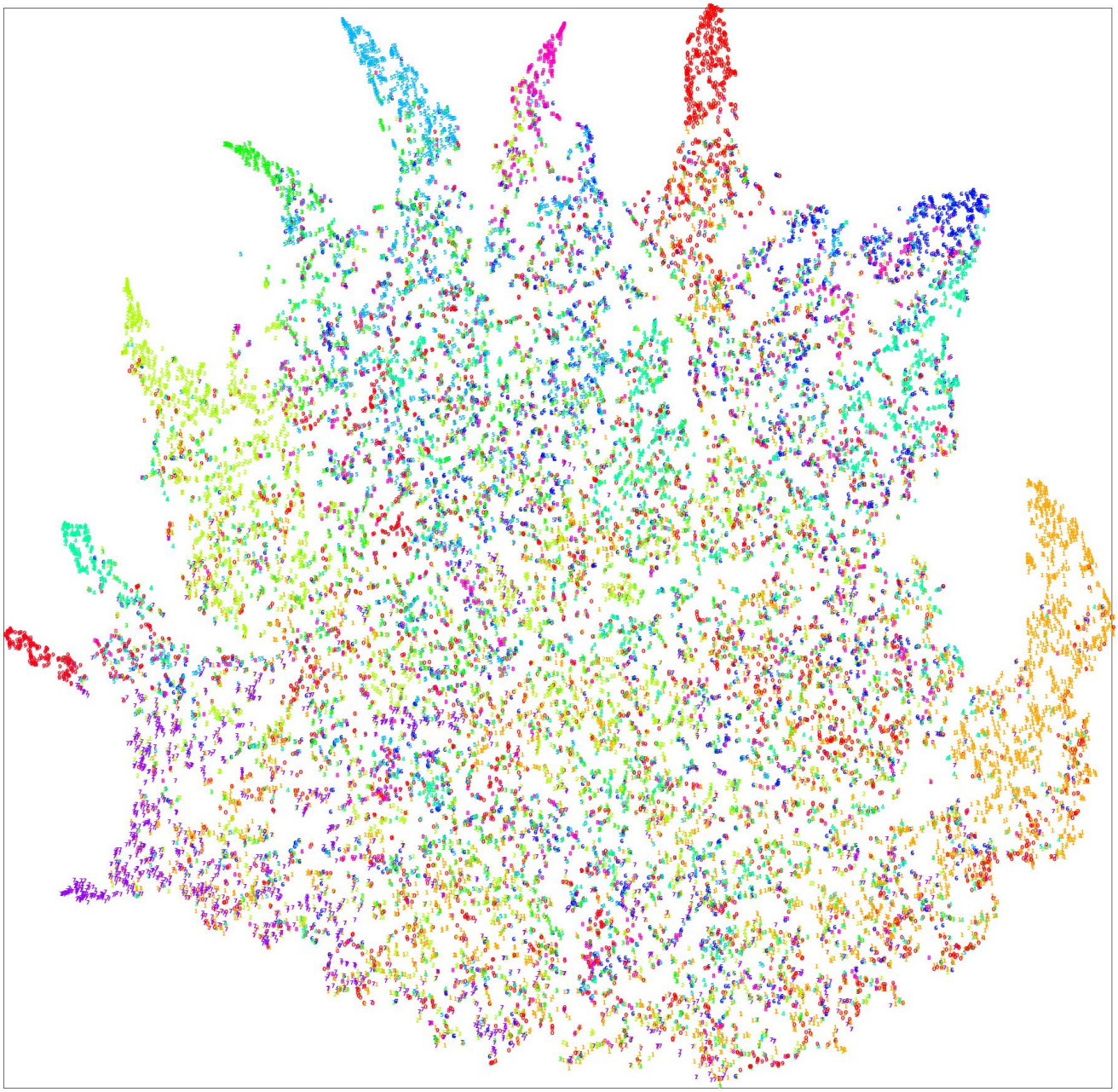
1~3 .

	MNSITM → USPS	SVHN → MNIST-M	USPS → SVHN
On source	0.7045	0.4443	0.2127
DANN	0.7643	0.4918	0.3041
target	0.9583	0.9513	0.8722

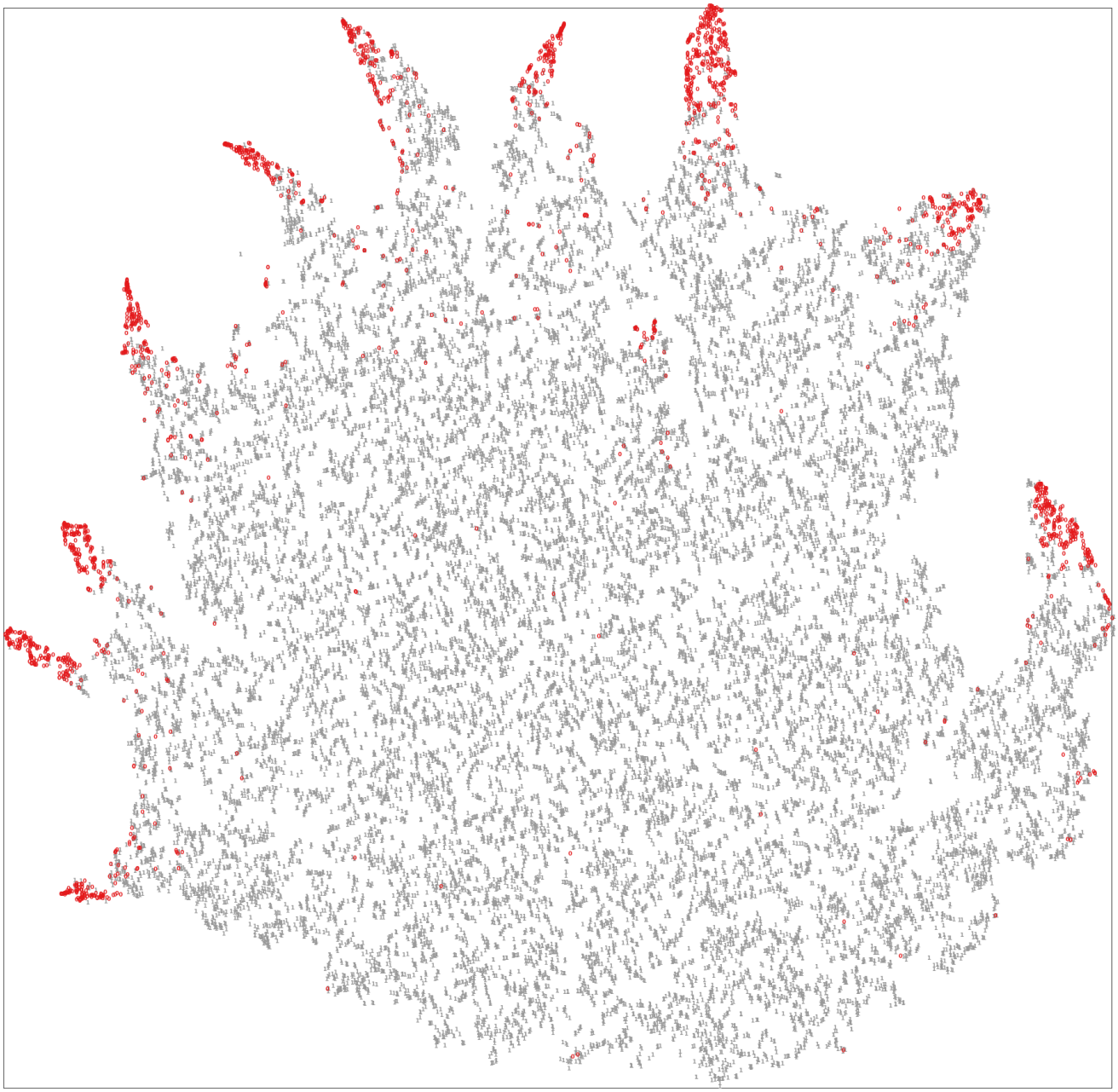
4.

- usps->svhn

  (a)

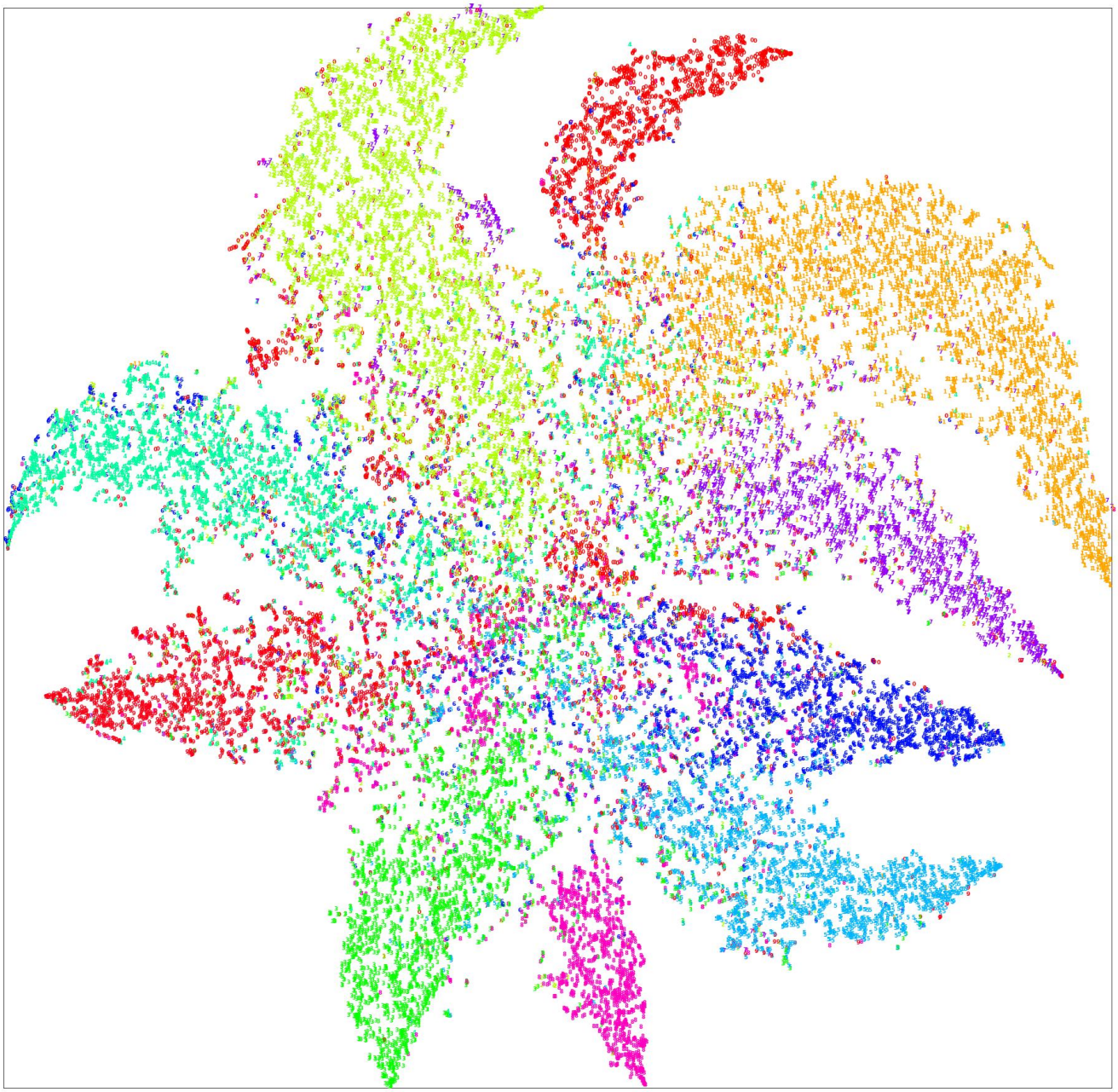


(b)

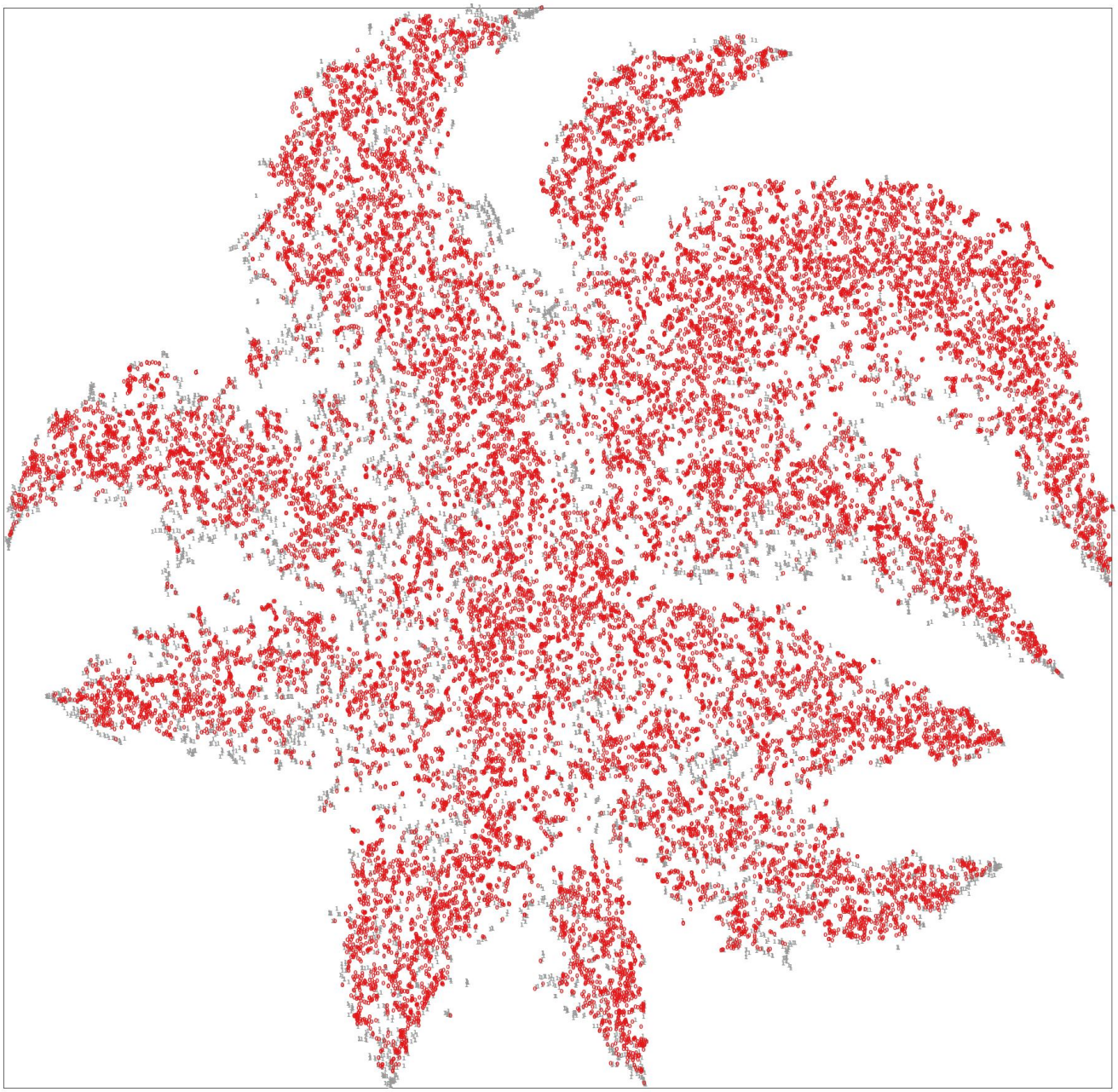


- svhn->mnistm

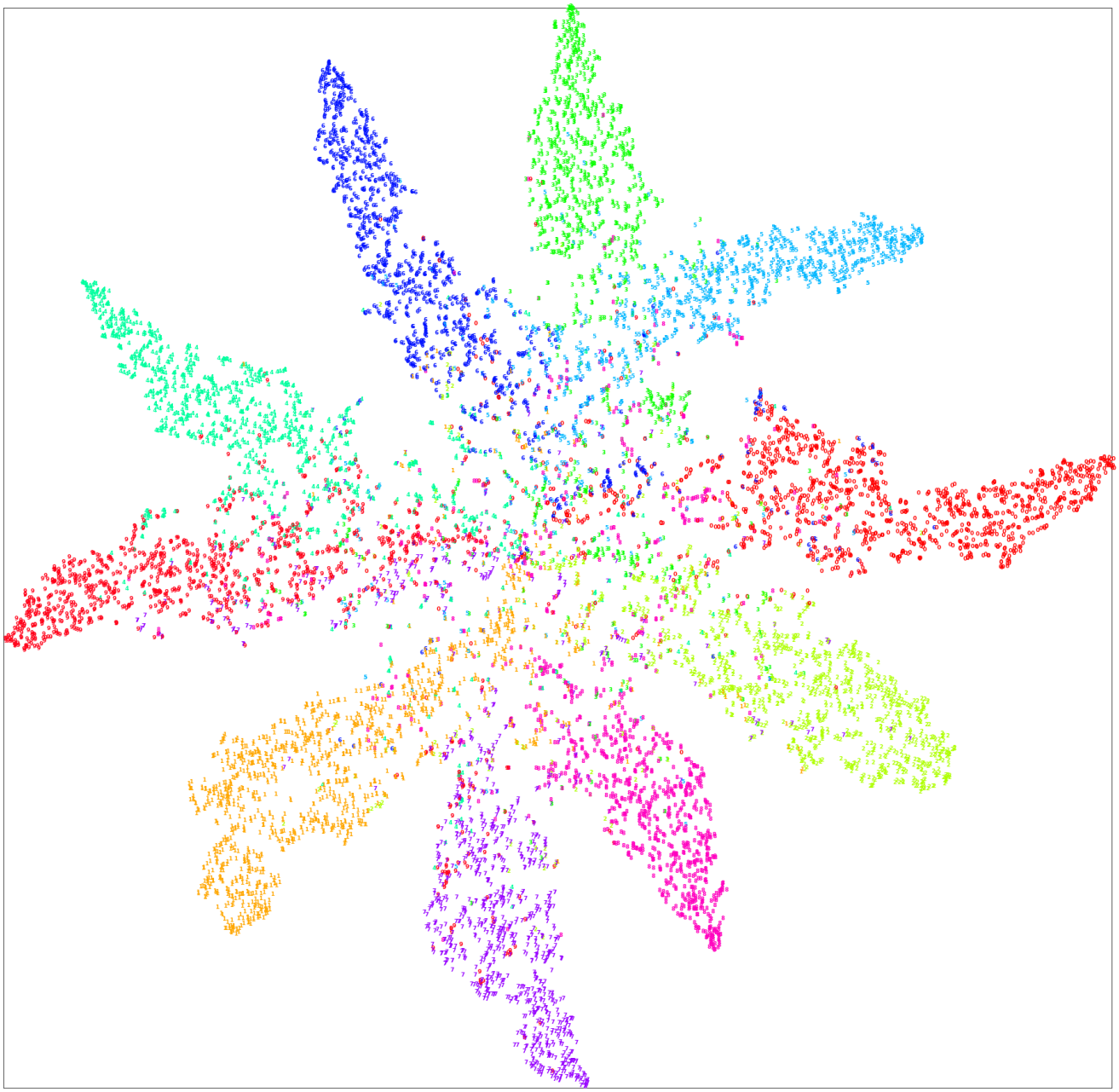
  (a)



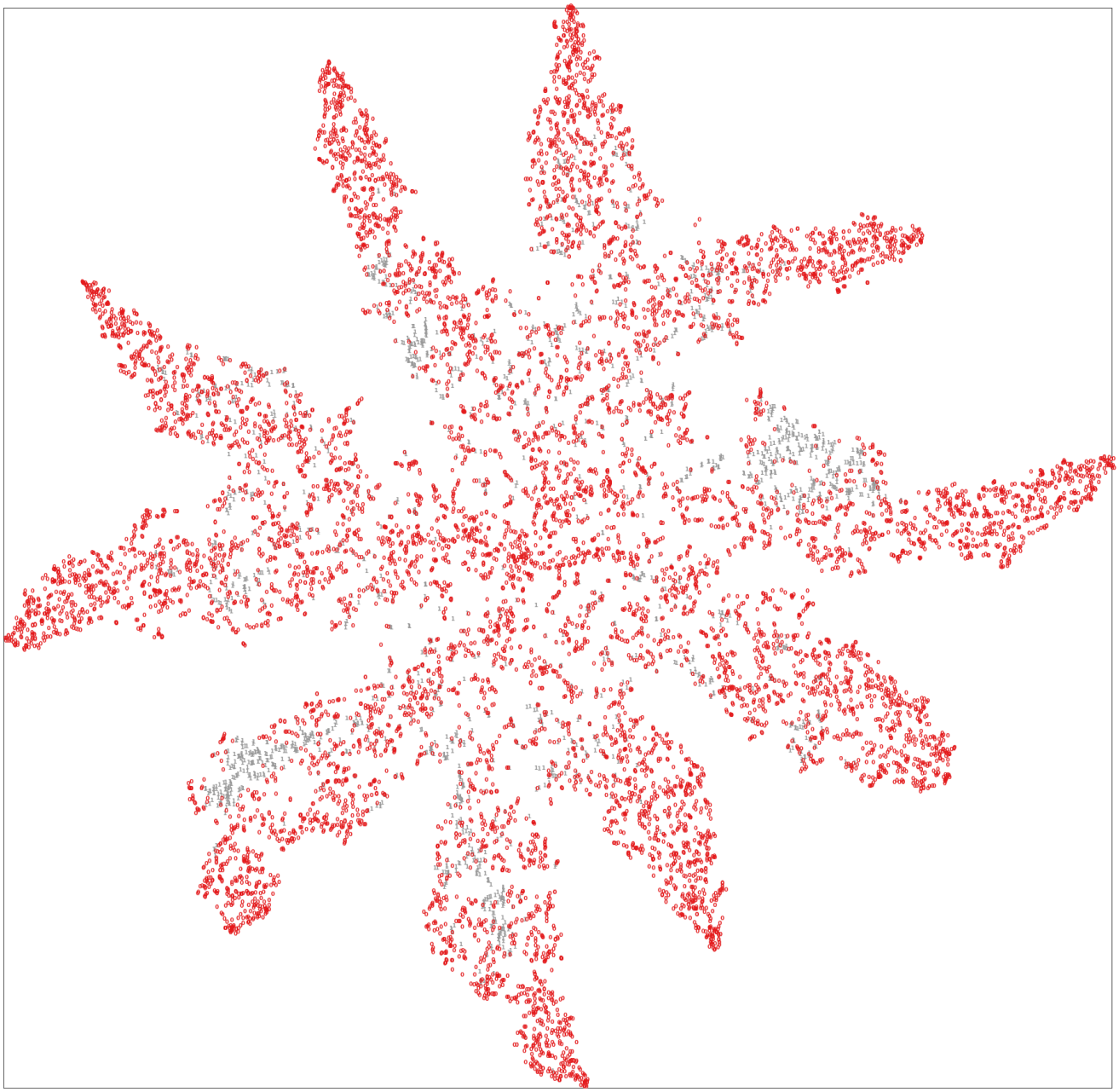
(b)



- mnistm->usps
  (a)



(b)



5.

On usps->svhn, if we use models that are too deep, the performance would be bad. I grayscaled all images if source or domain is usps. I used a model that has featureExtractor, domainClassifier and labelPredictor, and a GRL layer. The model is simple but enough to pass all baselines.

Bonus

1.

	MNSITM → USPS	SVHN → MNIST-M	USPS → SVHN
DANN	0.7643	0.4918	0.3041
improve	0.9147	0.6805	0.4810

2.

I used auto-augment and I implemented a MCD model from this https://github.com/mil-tokyo/MCD_DA (https://github.com/mil-tokyo/MCD_DA).
I guess the reason that the new one improves because the model and loss function are better and autoaugment let the distribution of source and target more likely to be aligned.