DLCV HW2

資工AI所碩一 r11922a05 林聖硯

Problem1: GAN

1. Please print the model architecture of method A and B.

I use the same model structure (DCGAN) in both method A and method B. In method A, my model has the same structure as the one in pytorch tutorial (ref.) In method B, I use the following strategy to improve DCGAN.

- Enlarge the dimension of random noise from 100 to 300
- Use mode seek loss (from this paper), any GAN that uses this loss is called “MSGAN”

Model structiure of DCGAN

```
DCGAN_Discriminator(
  (dis): 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()
  )
)
DCGAN_Generator(
  (gen): Sequential(
    (0): ConvTranspose2d(300, 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()
  )
)
```

2. Please show the first 32 generated images of both method A and B then discuss the difference between method A and B.

- First 32 generated images from method A (fid = 34.23, face recognition = 89.60)

- First 32 generated images from method B (fid = 25.75, face recognition = 91.4)

Clearly, the image quality of method B is better than that of method A in general. In addition, the generated images in method B are more diverse (Not always facing the same direction, have

more facial expression, hair is more colorful, etc.) since I implemented mode seeking loss.

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

I also did the folloing experiments to improve my DCGAN.

a. Binary cross entropy loss v.s. Binary cross entropy loss + mode seeking loss

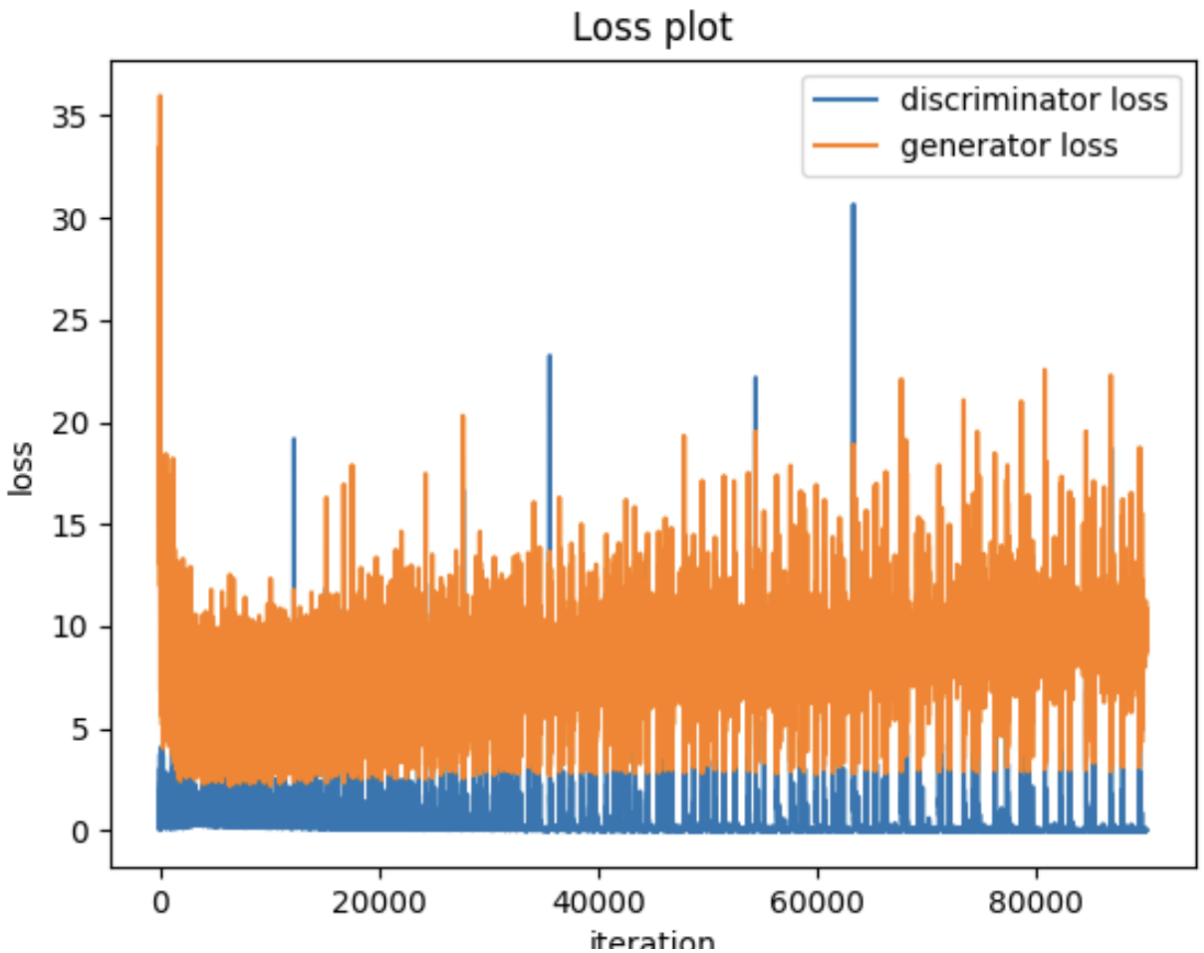
b. weight initialization in batch norm 2d: initialize with N(0, 0.02) v.s. initialize with N(1.0, 0.02)

c. size of random noice: 100 v.s. 200 v.s. 300

d. randomly generate 20000 pictures and pick the 1000 pictures with the highest score judged by discriminator (i.e. choose the top 1000 pictures that the discirminator couldn't tell it's real or fake)

Observation

- My DCGAN model never converge, the discriminator was always strong (i.e. $D(x) \rightarrow 1$) and the losses of both discriminator and generator are not stable



- Mode seeking loss helps get better fid and face recognition result
- The greater the size of random noise, the better the perfomance result. The feature space is larger, thus the generator has higher possibility to generate diverse images. Plus, with mode seeking loss, the performace of my model gets even better.
- Strategy (b) and strategy (d) didn’t make much improvement of my model

Problem2: diffusion model

1. Please print your model architecture and describe your implementation details.

I check the folloing reference to build my diffusion model.

- The annotated diffusion model
- Denoising diffusion probabilismodel

model architecture (Unet)

```
Unet(
  (init_conv): Conv2d(3, 18, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
  (time_mlp): Sequential(
    (0): SinusoidalPositionEmbeddings()
    (1): Linear(in_features=28, out_features=112, bias=True)
    (2): GELU(approximate=none)
    (3): Linear(in_features=112, out_features=112, bias=True)
  )
  (label_emb): Embedding(10, 112)
  (downs): ModuleList(
    (0): ModuleList(
      (0): ConvNextBlock(
        (time_mlp): Sequential(
          (0): GELU(approximate=none)
          (1): Linear(in_features=112, out_features=18, bias=True)
        )
        (label_mlp): Sequential(
          (0): GELU(approximate=none)
          (1): Linear(in_features=112, out_features=18, bias=True)
        )
        (ds_conv): Conv2d(18, 18, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=18)
        (net): Sequential(
          (0): GroupNorm(1, 18, eps=1e-05, affine=True)
          (1): Conv2d(18, 56, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (2): GELU(approximate=none)
          (3): GroupNorm(1, 56, eps=1e-05, affine=True)
          (4): Conv2d(56, 28, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        )
        (res_conv): Conv2d(18, 28, kernel_size=(1, 1), stride=(1, 1))
      )
      (1): ConvNextBlock(
        (time_mlp): Sequential(
          (0): GELU(approximate=none)
          (1): Linear(in_features=112, out_features=28, bias=True)
        )
        (label_mlp): Sequential(
          (0): GELU(approximate=none)
          (1): Linear(in_features=112, out_features=28, bias=True)
        )
        (ds_conv): Conv2d(28, 28, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=28)
        (net): Sequential(
          (0): GroupNorm(1, 28, eps=1e-05, affine=True)
          (1): Conv2d(28, 56, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (2): GELU(approximate=none)
          (3): GroupNorm(1, 56, eps=1e-05, affine=True)
          (4): Conv2d(56, 28, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        )
        (res_conv): Identity()
      )
```

```
  (2): Residual(
    (fn): PreNorm(
      (fn): LinearAttention(
        (to_qkv): Conv2d(28, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (to_out): Sequential(
          (0): Conv2d(128, 28, kernel_size=(1, 1), stride=(1, 1))
          (1): GroupNorm(1, 28, eps=1e-05, affine=True)
        )
      )
      (norm): GroupNorm(1, 28, eps=1e-05, affine=True)
    )
  )
  (3): Conv2d(28, 28, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
)
(1): ModuleList(
  (0): ConvNextBlock(
    (time_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=28, bias=True)
    )
    (label_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=28, bias=True)
    )
    (ds_conv): Conv2d(28, 28, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=28)
    (net): Sequential(
      (0): GroupNorm(1, 28, eps=1e-05, affine=True)
      (1): Conv2d(28, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (2): GELU(approximate=none)
      (3): GroupNorm(1, 112, eps=1e-05, affine=True)
      (4): Conv2d(112, 56, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (res_conv): Conv2d(28, 56, kernel_size=(1, 1), stride=(1, 1))
  )
  (1): ConvNextBlock(
    (time_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=56, bias=True)
    )
    (label_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=56, bias=True)
    )
    (ds_conv): Conv2d(56, 56, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=56)
    (net): Sequential(
      (0): GroupNorm(1, 56, eps=1e-05, affine=True)
      (1): Conv2d(56, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (2): GELU(approximate=none)
      (3): GroupNorm(1, 112, eps=1e-05, affine=True)
      (4): Conv2d(112, 56, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (res_conv): Identity()
  )
```

```
  (2): Residual(
    (fn): PreNorm(
      (fn): LinearAttention(
        (to_qkv): Conv2d(56, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (to_out): Sequential(
          (0): Conv2d(128, 56, kernel_size=(1, 1), stride=(1, 1))
          (1): GroupNorm(1, 56, eps=1e-05, affine=True)
        )
      )
      (norm): GroupNorm(1, 56, eps=1e-05, affine=True)
    )
  )
  (3): Conv2d(56, 56, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
)
(2): ModuleList(
  (0): ConvNextBlock(
    (time_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=56, bias=True)
    )
    (label_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=56, bias=True)
    )
    (ds_conv): Conv2d(56, 56, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=56)
    (net): Sequential(
      (0): GroupNorm(1, 56, eps=1e-05, affine=True)
      (1): Conv2d(56, 224, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (2): GELU(approximate=none)
      (3): GroupNorm(1, 224, eps=1e-05, affine=True)
      (4): Conv2d(224, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (res_conv): Conv2d(56, 112, kernel_size=(1, 1), stride=(1, 1))
  )
  (1): ConvNextBlock(
    (time_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=112, bias=True)
    )
    (label_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=112, bias=True)
    )
    (ds_conv): Conv2d(112, 112, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=112)
    (net): Sequential(
      (0): GroupNorm(1, 112, eps=1e-05, affine=True)
      (1): Conv2d(112, 224, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (2): GELU(approximate=none)
      (3): GroupNorm(1, 224, eps=1e-05, affine=True)
      (4): Conv2d(224, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (res_conv): Identity()
  )
```

```
    (2): Residual(
      (fn): PreNorm(
        (fn): LinearAttention(
          (to_qkv): Conv2d(112, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (to_out): Sequential(
            (0): Conv2d(128, 112, kernel_size=(1, 1), stride=(1, 1))
            (1): GroupNorm(1, 112, eps=1e-05, affine=True)
          )
        )
        (norm): GroupNorm(1, 112, eps=1e-05, affine=True)
      )
    )
    (3): Identity()
  )
)
(ups): ModuleList(
  (0): ModuleList(
    (0): ConvNextBlock(
      (time_mlp): Sequential(
        (0): GELU(approximate=none)
        (1): Linear(in_features=112, out_features=224, bias=True)
      )
      (label_mlp): Sequential(
        (0): GELU(approximate=none)
        (1): Linear(in_features=112, out_features=224, bias=True)
      )
      (ds_conv): Conv2d(224, 224, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=224)
      (net): Sequential(
        (0): GroupNorm(1, 224, eps=1e-05, affine=True)
        (1): Conv2d(224, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (2): GELU(approximate=none)
        (3): GroupNorm(1, 112, eps=1e-05, affine=True)
        (4): Conv2d(112, 56, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (res_conv): Conv2d(224, 56, kernel_size=(1, 1), stride=(1, 1))
    )
    (1): ConvNextBlock(
      (time_mlp): Sequential(
        (0): GELU(approximate=none)
        (1): Linear(in_features=112, out_features=56, bias=True)
      )
      (label_mlp): Sequential(
        (0): GELU(approximate=none)
        (1): Linear(in_features=112, out_features=56, bias=True)
      )
      (ds_conv): Conv2d(56, 56, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=56)
      (net): Sequential(
        (0): GroupNorm(1, 56, eps=1e-05, affine=True)
        (1): Conv2d(56, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (2): GELU(approximate=none)
        (3): GroupNorm(1, 112, eps=1e-05, affine=True)
        (4): Conv2d(112, 56, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (res_conv): Identity()
    )
```

```
  (2): Residual(
    (fn): PreNorm(
      (fn): LinearAttention(
        (to_qkv): Conv2d(56, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (to_out): Sequential(
          (0): Conv2d(128, 56, kernel_size=(1, 1), stride=(1, 1))
          (1): GroupNorm(1, 56, eps=1e-05, affine=True)
        )
      )
      (norm): GroupNorm(1, 56, eps=1e-05, affine=True)
    )
  )
  (3): ConvTranspose2d(56, 56, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
)
(1): ModuleList(
  (0): ConvNextBlock(
    (time_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=112, bias=True)
    )
    (label_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=112, bias=True)
    )
    (ds_conv): Conv2d(112, 112, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=112)
    (net): Sequential(
      (0): GroupNorm(1, 112, eps=1e-05, affine=True)
      (1): Conv2d(112, 56, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (2): GELU(approximate=none)
      (3): GroupNorm(1, 56, eps=1e-05, affine=True)
      (4): Conv2d(56, 28, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (res_conv): Conv2d(112, 28, kernel_size=(1, 1), stride=(1, 1))
  )
  (1): ConvNextBlock(
    (time_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=28, bias=True)
    )
    (label_mlp): Sequential(
      (0): GELU(approximate=none)
      (1): Linear(in_features=112, out_features=28, bias=True)
    )
    (ds_conv): Conv2d(28, 28, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=28)
    (net): Sequential(
      (0): GroupNorm(1, 28, eps=1e-05, affine=True)
      (1): Conv2d(28, 56, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (2): GELU(approximate=none)
      (3): GroupNorm(1, 56, eps=1e-05, affine=True)
      (4): Conv2d(56, 28, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (res_conv): Identity()
  )
```

```
    (2): Residual(
      (fn): PreNorm(
        (fn): LinearAttention(
          (to_qkv): Conv2d(28, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (to_out): Sequential(
            (0): Conv2d(128, 28, kernel_size=(1, 1), stride=(1, 1))
            (1): GroupNorm(1, 28, eps=1e-05, affine=True)
          )
        )
        (norm): GroupNorm(1, 28, eps=1e-05, affine=True)
      )
    )
    (3): ConvTranspose2d(28, 28, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  )
)
(mid_block1): ConvNextBlock(
  (time_mlp): Sequential(
    (0): GELU(approximate=none)
    (1): Linear(in_features=112, out_features=112, bias=True)
  )
  (label_mlp): Sequential(
    (0): GELU(approximate=none)
    (1): Linear(in_features=112, out_features=112, bias=True)
  )
  (ds_conv): Conv2d(112, 112, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=112)
  (net): Sequential(
    (0): GroupNorm(1, 112, eps=1e-05, affine=True)
    (1): Conv2d(112, 224, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (2): GELU(approximate=none)
    (3): GroupNorm(1, 224, eps=1e-05, affine=True)
    (4): Conv2d(224, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (res_conv): Identity()
)
(mid_attn): Residual(
  (fn): PreNorm(
    (fn): Attention(
      (to_qkv): Conv2d(112, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (to_out): Conv2d(128, 112, kernel_size=(1, 1), stride=(1, 1))
    )
    (norm): GroupNorm(1, 112, eps=1e-05, affine=True)
  )
)
(mid_block2): ConvNextBlock(
  (time_mlp): Sequential(
    (0): GELU(approximate=none)
    (1): Linear(in_features=112, out_features=112, bias=True)
  )
  (label_mlp): Sequential(
    (0): GELU(approximate=none)
    (1): Linear(in_features=112, out_features=112, bias=True)
  )
  (ds_conv): Conv2d(112, 112, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=112)
  (net): Sequential(
    (0): GroupNorm(1, 112, eps=1e-05, affine=True)
```

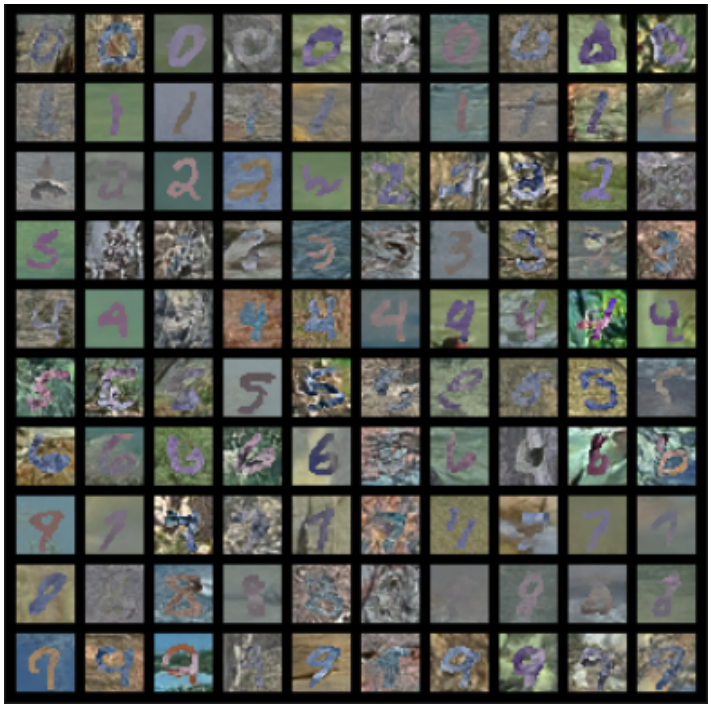
```
    (1): Conv2d(112, 224, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (2): GELU(approximate=none)
    (3): GroupNorm(1, 224, eps=1e-05, affine=True)
    (4): Conv2d(224, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (res_conv): Identity()
)
(mid_attn): Residual(
  (fn): PreNorm(
    (fn): Attention(
      (to_qkv): Conv2d(112, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (to_out): Conv2d(128, 112, kernel_size=(1, 1), stride=(1, 1))
    )
    (norm): GroupNorm(1, 112, eps=1e-05, affine=True)
  )
)
(mid_block2): ConvNextBlock(
  (time_mlp): Sequential(
    (0): GELU(approximate=none)
    (1): Linear(in_features=112, out_features=112, bias=True)
  )
  (label_mlp): Sequential(
    (0): GELU(approximate=none)
    (1): Linear(in_features=112, out_features=112, bias=True)
  )
  (ds_conv): Conv2d(112, 112, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=112)
  (net): Sequential(
    (0): GroupNorm(1, 112, eps=1e-05, affine=True)
    (1): Conv2d(112, 224, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (2): GELU(approximate=none)
    (3): GroupNorm(1, 224, eps=1e-05, affine=True)
    (4): Conv2d(224, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (res_conv): Identity()
)
(final_conv): Sequential(
  (0): ConvNextBlock(
    (ds_conv): Conv2d(28, 28, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=28)
    (net): Sequential(
      (0): GroupNorm(1, 28, eps=1e-05, affine=True)
      (1): Conv2d(28, 56, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (2): GELU(approximate=none)
      (3): GroupNorm(1, 56, eps=1e-05, affine=True)
      (4): Conv2d(56, 28, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (res_conv): Identity()
  )
  (1): Conv2d(28, 3, kernel_size=(1, 1), stride=(1, 1))
)
```

Implementation detail

- Data transform: ToTensor ((H,W,C) → (C,W,H) and divide pixel value by 255)) and (pixel value*2) - 1 (recommend by the paper)
- model: Unet
- optimizer: Adam

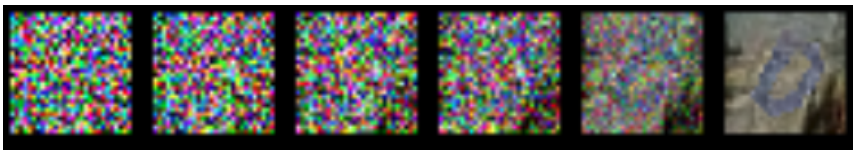
- learning rate: 0.001
- number of running epoch: 300
- loss function: smooth l1 loss (ref)
- batch size: 128
- timesteps: 200 (T in the original papaer)

2. Please show 10 generated images for each digit (0-9) in your report. You can put all 100 outputs in one image with columns indicating different noise inputs and rows indicating different digits.



3. Visualize total six images in the reverse process of the first “0” in your grid in (2) with different time steps.

time step = 0, 40, 80, 120, 160, 200



4. Please discuss what you’ve observed and learned from implementing conditional diffusion model.

I also did the folloing experiments to improve my diffusion model.

- beta scheduler: linear, cosine, quadratic, sigmoid
- loss function: l1 loss, mse loss, smooth l1 loss (huber loss)
- changing batch size

Observation:

- Deta scheduler has big impact on performance result (linear > quadratic > cosine > sigmoid)
- Loss function and changes in batch size have only impact on performance result
- At the beggining, I used multi-head self attention (which takes quadratic time) in my U-net modle. However, since the the training time was way too long, I used the linear attention (which requires only linear time to calculate the attention) in the end. Finally, it took around 6 mins to train one epoch and the whole training process took me a day (~1700mins).

Problem3: DANN

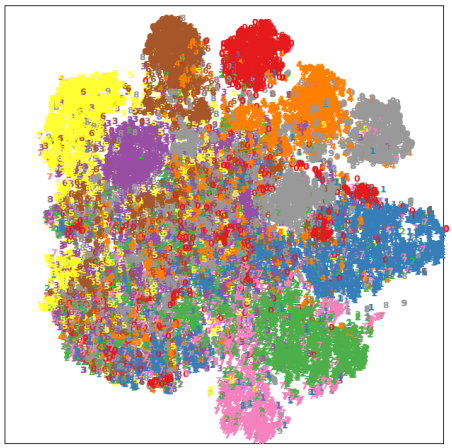
1. Please create and fill the table with the following format in your report:

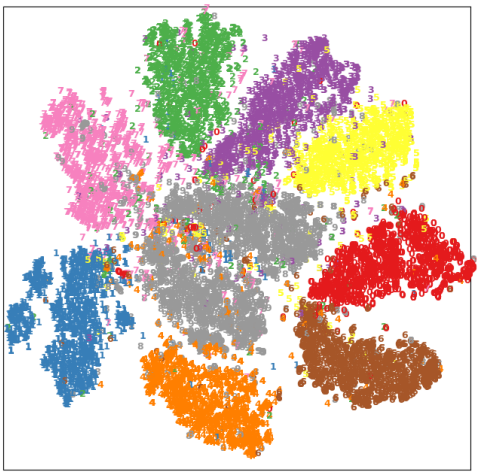
	MNIST-M → SVHN	MNIST-M → USPS
Trained on source	39.01%	68.91%
DANN	48.37%	87.37%
Trained on target	70.52%	93.85%

2. Please visualize the latent space of DANN by mapping the validation images to 2D space with t-SNE. For each scenario, you need to plot two figures which are colored by digit class (0-9) and by domain, respectively. (Note that you need to plot the figures of both 2 scenarios, so 4 figures in total.)

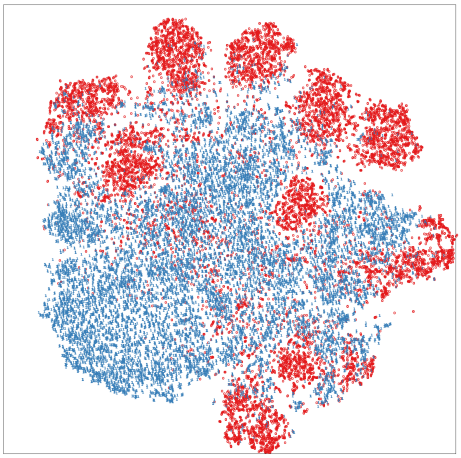
MNIST-M → SVHN **MNIST-M → USPS**

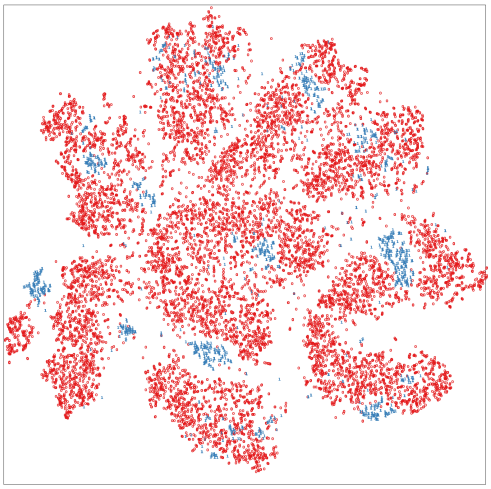
by class





by domain





3. Please describe the implementation details of your model and discuss what you've observed and learned from implementing DANN.

I also did the folloing experiments to improve my DANN model.

- adjust learning rate (1e-2 ~ 1e-5)
- adjust optimizer (SGD / Adam)
- use learning rate scheduler ReduceLROnPlateau(optimizer, 'max', eps=1e-7)
- try different model strcture
    - model 1

```python
self.feature_extractor = nn.Sequential(
    nn.Conv2d(3, 64, 5),
    nn.BatchNorm2d(64),
    nn.MaxPool2d(2),
    nn.ReLU(inplace=True),
    nn.Conv2d(64, 50, 5),
    nn.BatchNorm2d(50),
    nn.Dropout2d(), #change to dropout 1d?
    nn.MaxPool2d(2),
    nn.ReLU(inplace=True)
)

self.domain_clf = nn.Sequential(
    nn.Linear(50*4*4, 100),
    nn.BatchNorm1d(100),
    nn.ReLU(inplace=True),
    nn.Linear(100, 1), #If we want to use BCELoss, the output layer should be 1
    nn.Sigmoid()
)

self.label_clf = nn.Sequential(
    nn.Linear(50*4*4, 100),
    nn.BatchNorm1d(100),
    nn.ReLU(inplace=True),
    nn.Dropout2d(), #change to dropout 1d?
    nn.Linear(100, 100),
    nn.BatchNorm1d(100),
    nn.ReLU(inplace=True),
    nn.Linear(100, 10), #Use BCE loss in p3_train.py
    nn.LogSoftmax(dim=1)
    #nn.CrossEntropyLoss() = nn.NLLLoss() + nn.LogsoftMax()
)
```

- model 2

```
self.feature_extractor = nn.Sequential(
    nn.Conv2d(3,64,5),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(3,2),
    nn.Conv2d(64,64,5),
    nn.ReLU(inplace=True),
    nn.Conv2d(64,128,5),
    )

self.domain_clf = nn.Sequential(
            nn.Linear(1152,1024),
             nn.ReLU(inplace=True),
             nn.Linear(1024,1024),
             nn.ReLU(inplace=True),
             nn.Linear(1024,1),
             nn.Sigmoid())

self.label_clf = nn.Sequential(
    nn.Linear(1152,3072),
    nn.ReLU(inplace=True),
    nn.Linear(3072,2048),
    nn.ReLU(inplace=True),
    nn.Linear(2048,10),
    nn.LogSoftmax(dim=1)
)
```

Observation

- SGD + greater learning rate + decrease learning rate during training achieve better performance
- Learning rate scheduler doesn’t help a lot in perfomance result
- model 2 is better for both tasks (dropout layers affect the performance negatively in both tasks)
- During the training process, the domain loss usually decreases smoothing while classification tend to have unstable trend