2021 DLCV fall hw2 r09942091 謝承軒

***** Problem 1 *****

1. Use **DCGAN**

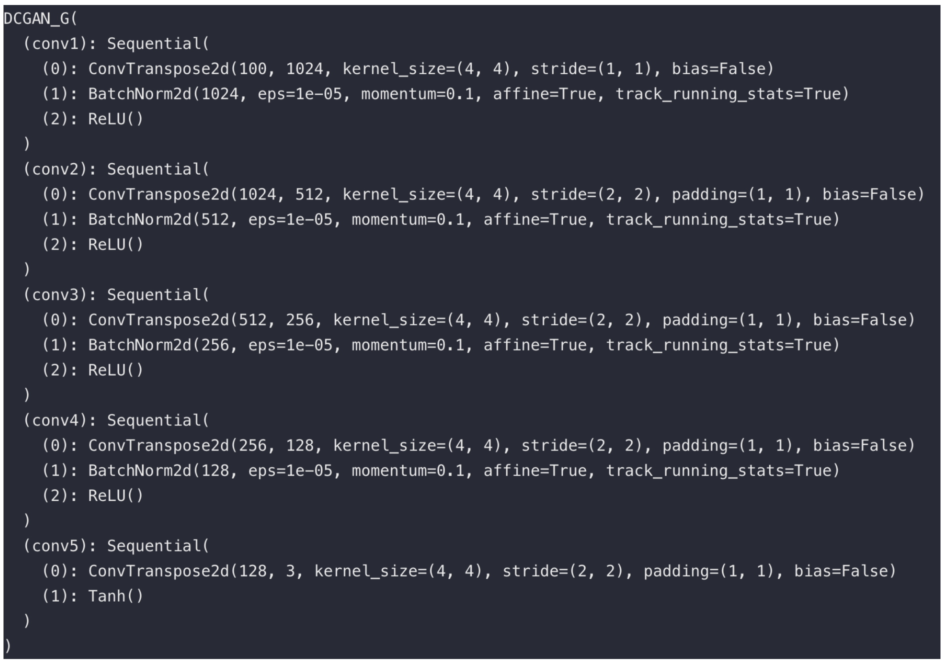
   details:

   Batch_size=128, epochs=110, optimizer=Adam(betas=(0.5, 0.999)), lr=2e-4

   noise dim.: 100, image normalize: mean=(0.5,), std=(0.5,)

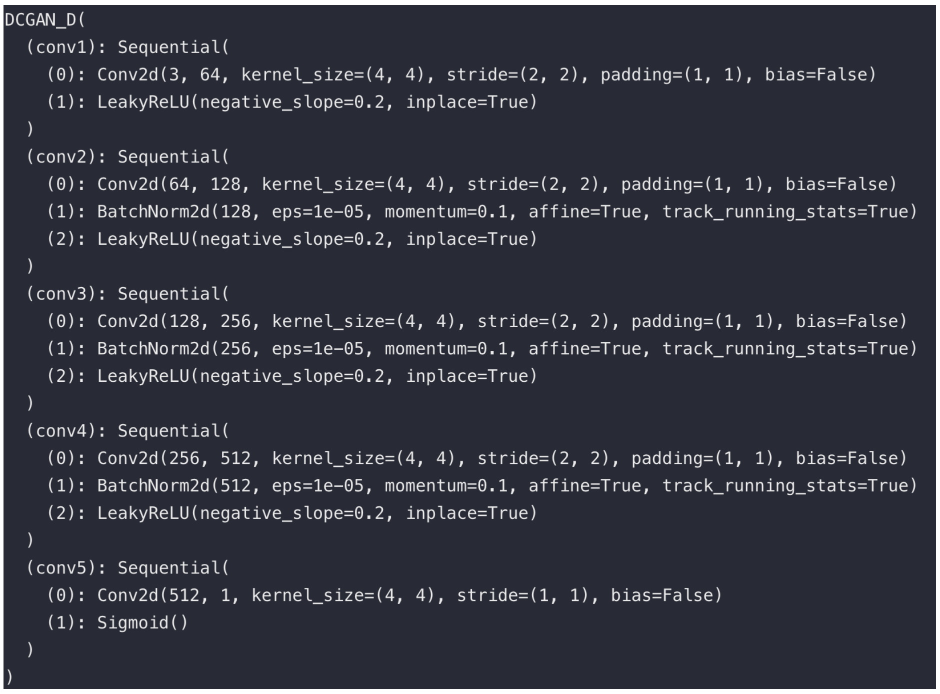
   Training seed: 999, inference seed: 0, Loss function: BCELoss

   Genetator:



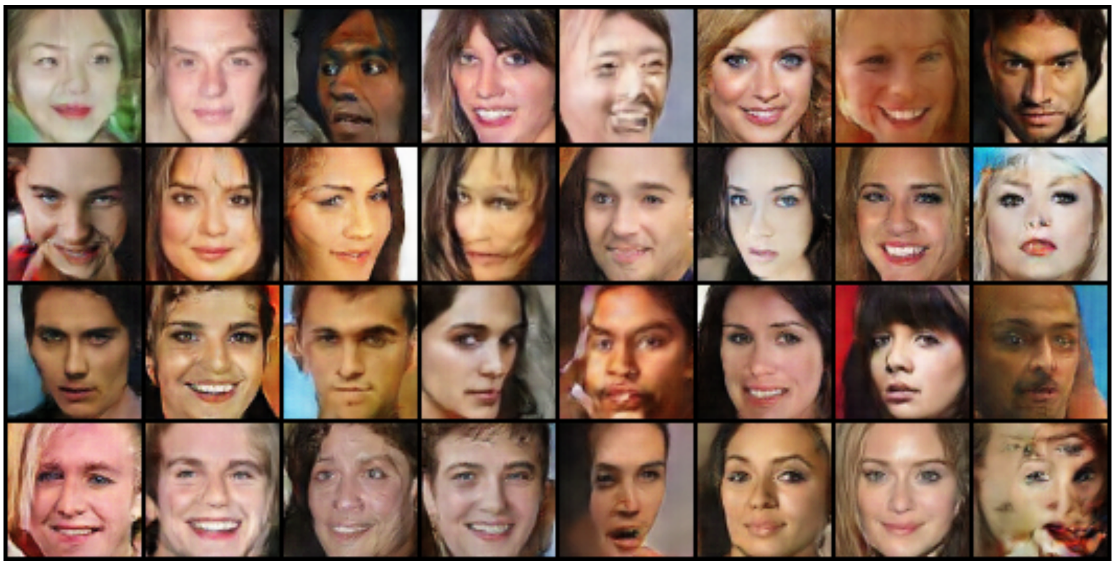
```
DCGAN_G(
  (conv1): Sequential(
    (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv2): Sequential(
    (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv3): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv4): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv5): Sequential(
    (0): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): Tanh()
  )
)
```

   Discriminator:



```
DCGAN_D(
  (conv1): 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)
  )
  (conv2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (conv3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (conv4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (conv5): Sequential(
    (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): Sigmoid()
  )
)
```

2.



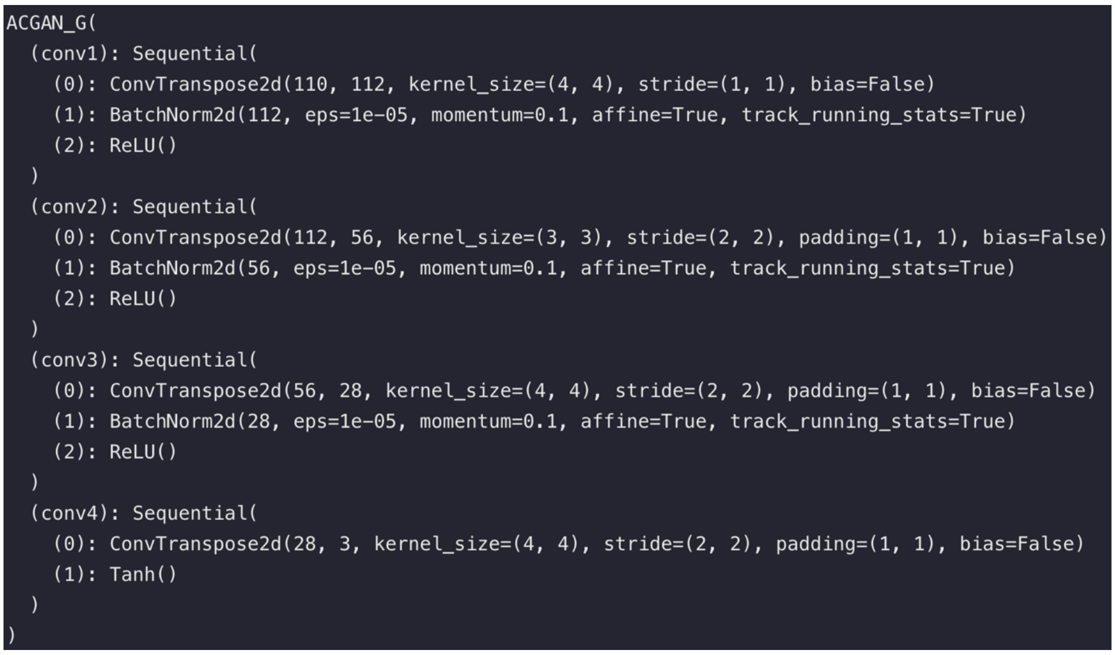
3. FID: 27.088
   IS: 2.117

4. In my training process, after 20 epoch the fid score is wandering, and slowly converge.
   The 96 epoch get the best fid score.
   Also, wgan’s training process(train D 5 times in every iteration) takes too long so I used dcgan.

***** Problem 2 *****

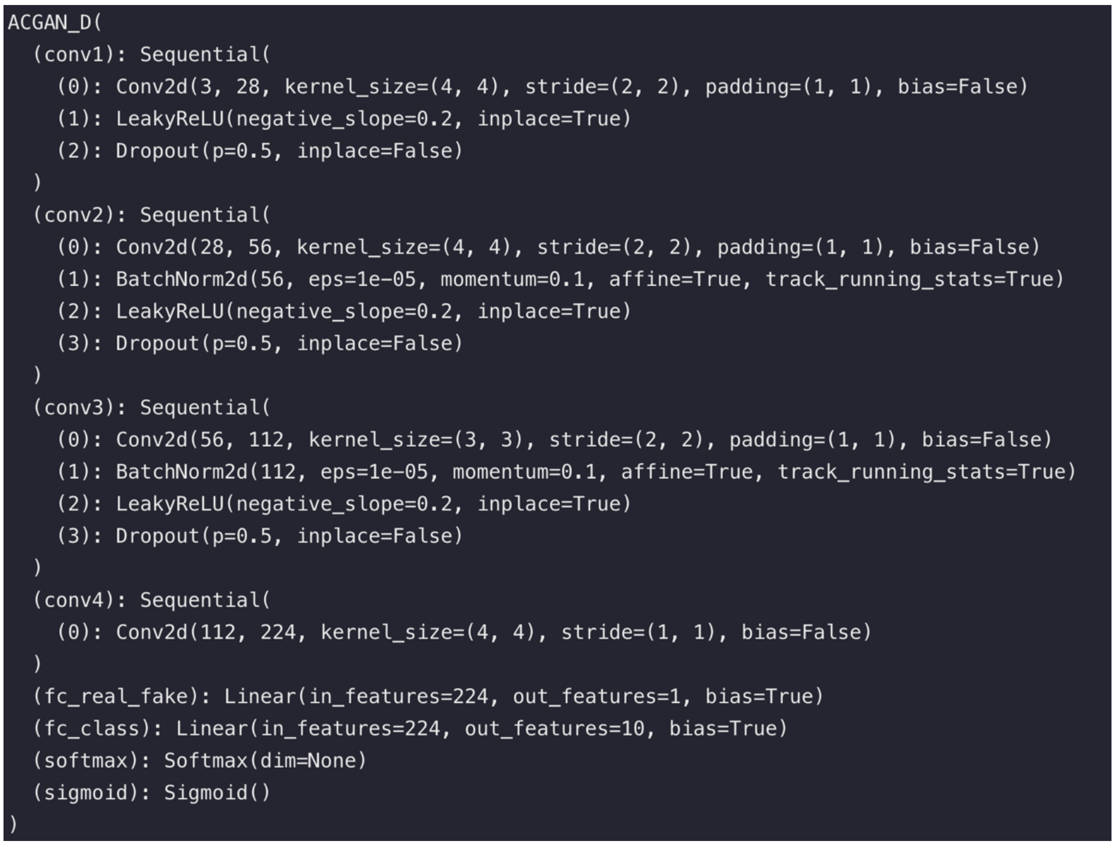
1. Batch_size=64, epochs=40, optimizer=Adam(betas=(0.5, 0.999)), lr=2e-4
   noise dim.: 110 (first ten are one hot), image normalize: mean=(0.5,), std=(0.5,)
   Training/inference seed: 999/2022, Loss function: BCELoss and NLLLoss

   Generator



```
ACGAN_G(
  (conv1): Sequential(
    (0): ConvTranspose2d(110, 112, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv2): Sequential(
    (0): ConvTranspose2d(112, 56, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(56, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv3): Sequential(
    (0): ConvTranspose2d(56, 28, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(28, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv4): Sequential(
    (0): ConvTranspose2d(28, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): Tanh()
  )
)
```

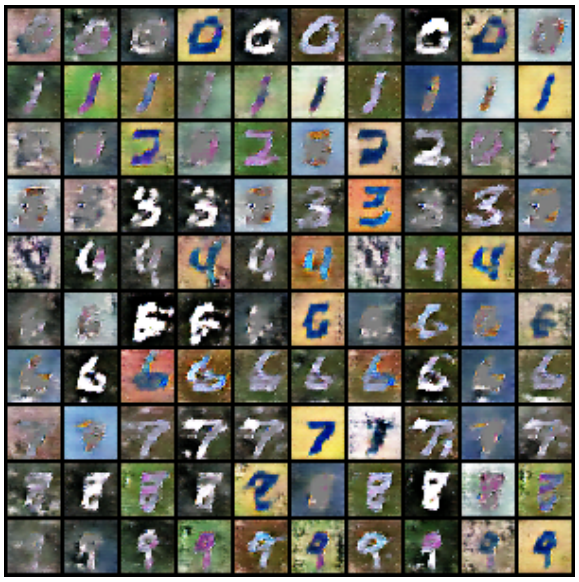
   Discriminator:



```
ACGAN_D(
  (conv1): Sequential(
    (0): Conv2d(3, 28, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Dropout(p=0.5, inplace=False)
  )
  (conv2): Sequential(
    (0): Conv2d(28, 56, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(56, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Dropout(p=0.5, inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(56, 112, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Dropout(p=0.5, inplace=False)
  )
  (conv4): Sequential(
    (0): Conv2d(112, 224, kernel_size=(4, 4), stride=(1, 1), bias=False)
  )
  (fc_real_fake): Linear(in_features=224, out_features=1, bias=True)
  (fc_class): Linear(in_features=224, out_features=10, bias=True)
  (softmax): Softmax(dim=None)
  (sigmoid): Sigmoid()
)
```

2. Acc: 81.2%

3.



How I generate 110dim*10class noise:

```
fixed_noise = torch.randn(1000, hidden_dim, 1, 1)
fake_noise = np.random.normal(0, 1, (1000, hidden_dim))
fake_onehot = np.zeros((1000, 10))
for i in range(10):
    fake_classes = np.ones(100, dtype=int)*i
    fake_onehot[np.arange(i*100, (i+1)*100), fake_classes] = 1
    fake_noise[np.arange(i*100, (i+1)*100), :10] = fake_onehot[np.arange(i*100, (i+1)*100)]
fake_noise = (torch.from_numpy(fake_noise))
fixed_noise.data.copy_(fake_noise.view(1000, hidden_dim, 1, 1))
fixed_noise = fixed_noise.to(device)
```

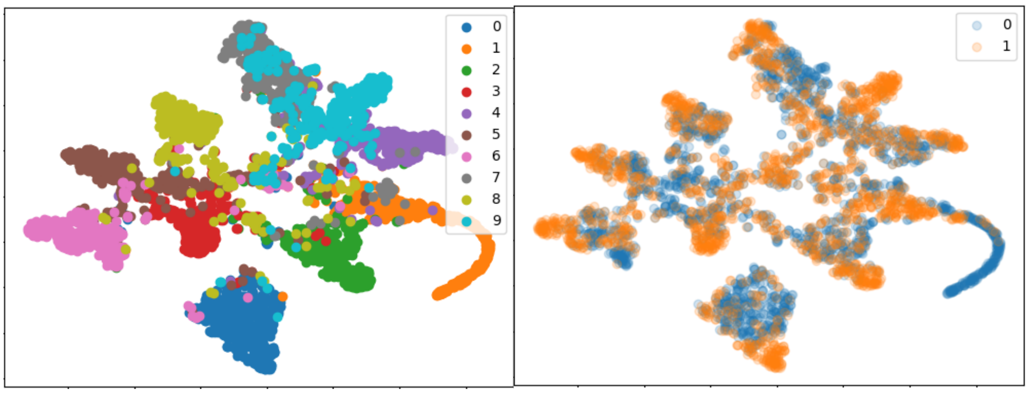
***** Problem 3 *****

1.

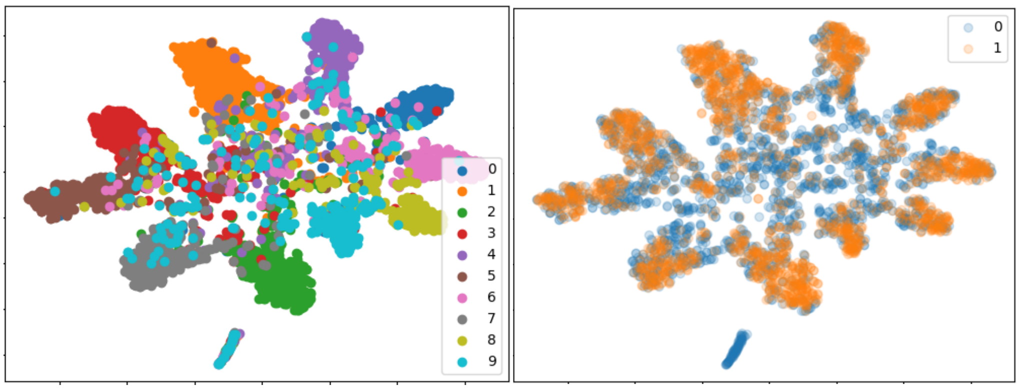
	MNIST-M -> USPS	SVHN -> MNIST-M	USPS -> SVHN
Trained on source	72.45	46.13	13.44
Adaptation	**78.57**	**54.45**	**29.76**
Trained on target	96.61	89.76	90.67

2. 每個 dataset 我都 random 取 2000 筆，看起來較平均一點

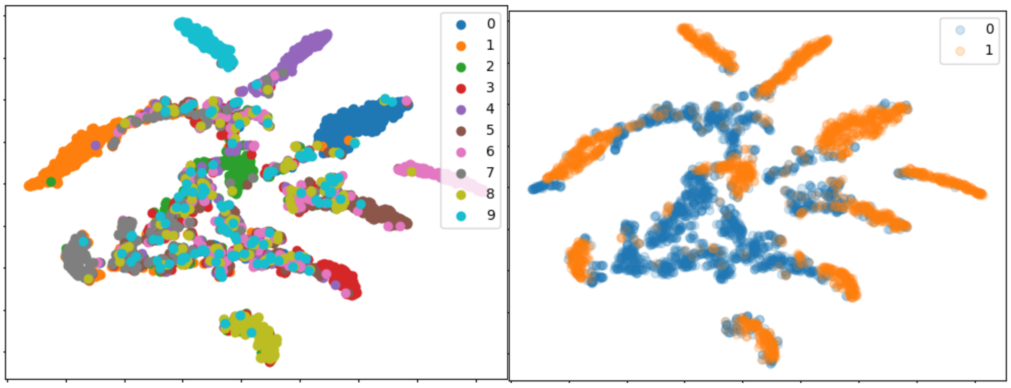
mnisim(1) -> usps(0)



svhn(1) -> mnistm(0)



usps(1) -> svhn(0)



3. Actually DANN does really improved, however, my latents pics don’t look really good.

My training details: (seed: 2022)

class predicted loss: Classes_loss – 0.02*lambda*domain_loss
lamda: 2/(1+e^(-10*p) – 1
p: epoch/MAX_EPOCH

Classes_loss: CrossEntropyLoss
domain_loss: BCEWithLogitsLoss

MAX_EPOCH: 25
optimizer & lr: Adam / 1e-3

also, I add clip_grad_norm_ with feature extractor