

# Homework #2

## *Deep Learning for Computer Vision*

Due: 110/11/23 (Tue.) 03:00 AM

電機所 R09921102 曾翊維

---

### Problem 1: DCGAN

1. Build your generator and discriminator from scratch and show your model architecture in your report (You can use “print(model)” in PyTorch directly). Then, train your model on the face dataset and describe implementation details. (Include but not limited to training epochs, learning rate schedule, data augmentation and optimizer) (5%)

Learning rate = 0.0002

number of epochs = 50

Data augmentation: the following transforms are used.

transforms.RandomHorizontalFlip(p=0.5),

transforms.ColorJitter(brightness=0.3)

optimizer = optim.Adam(netG.parameters(), lr=lr, betas=(0.5, 0.999))

Besides, I use soft labels. (i.e. replace 1 with uniform(0.8, 1) and replace 0 with uniform(0, 0.3) according to the tips provided in <https://github.com/soumith/ganhacks>)

The architecture is shown below.

```
Generator(  
  (main): 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(inplace=True)  
    (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (5): ReLU(inplace=True)  
    (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (8): ReLU(inplace=True)  
    (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (11): ReLU(inplace=True)  
    (12): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (13): Tanh()  
  )  
)
```

```

Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(1024, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)

```

2. Now, we can use the Generator to randomly generate images. Please samples 1000 noise vectors from Normal distribution and input them into your Generator. Save the 1000 generated images in the assigned folder path for evaluation, and show the first 32 images in your report. (5%)



3. Evaluate your 1000 generated images by implementing two metrics:  
(1) Fréchet inception distance (FID) : use all “test” images in the face dataset as reference  
(2) Inception score (IS)

FID: 28.0806259371237

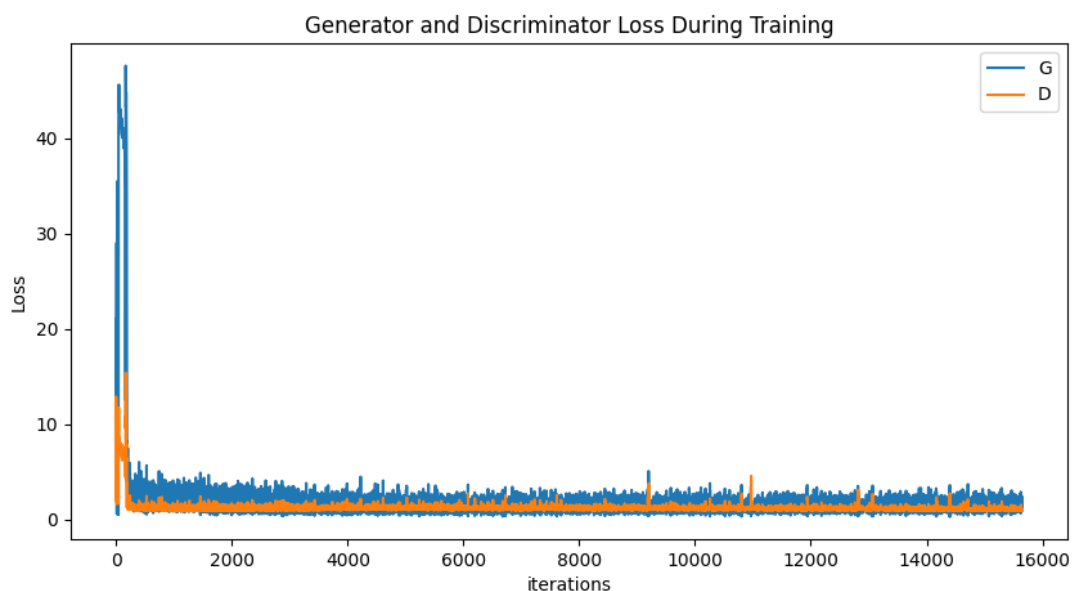
IS = 2.004525111160178

4. Discuss what you’ve observed and learned from implementing GAN.  
(5%)

When I set the ngf (i.e. depth of feature maps carried through the generator) and ndf (i.e. the depth of feature maps propagated through the discriminator) = 64, the result is unacceptable. After setting them = 128, the result gets better.

With soft labels, Generator can be trained more smoothly.

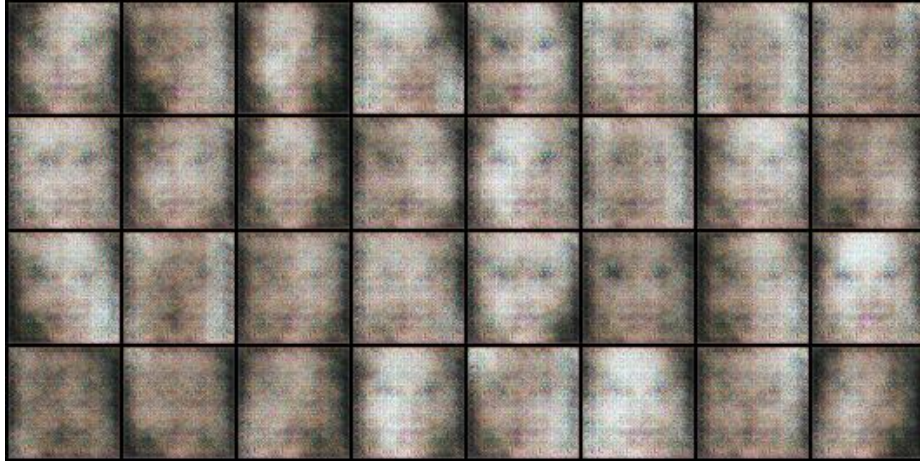
The loss during training is shown below.





I also save some samples after each epoch to see the progress.

Epoch 1



Epoch 10



Epoch 20



Epoch 30



Epoch 40



Epoch 50





## Problem 2: ACGAN


1. Build your ACGAN model from scratch and show your model architecture in your report (You can use “print(model)” in PyTorch directly). Then, train your model on the mnistm dataset and describe implementation details. (e.g. How do you input the class labels into the model?) (10%)

```
Generator(  
  (main): Sequential(  
    (0): ConvTranspose2d(100, 640, kernel_size=(4, 4), stride=(1, 1), bias=False)  
    (1): BatchNorm2d(640, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace=True)  
    (3): ConvTranspose2d(640, 320, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (4): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (5): ReLU(inplace=True)  
    (6): ConvTranspose2d(320, 160, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (7): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (8): ReLU(inplace=True)  
    (9): ConvTranspose2d(160, 80, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (10): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (11): ReLU(inplace=True)  
    (12): ConvTranspose2d(80, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (13): Tanh()  
  )  
)  
  
Discriminator(  
  (main): Sequential(  
    (0): Conv2d(3, 80, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): LeakyReLU(negative_slope=0.2, inplace=True)  
    (2): Conv2d(80, 160, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (3): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (4): LeakyReLU(negative_slope=0.2, inplace=True)  
    (5): Conv2d(160, 320, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (6): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (7): LeakyReLU(negative_slope=0.2, inplace=True)  
    (8): Conv2d(320, 640, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (9): BatchNorm2d(640, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (10): LeakyReLU(negative_slope=0.2, inplace=True)  
    (11): Conv2d(640, 80, kernel_size=(4, 4), stride=(1, 1), bias=False)  
  )  
  (dis_linear): Linear(in_features=80, out_features=1, bias=True)  
  (aux_linear): Linear(in_features=80, out_features=10, bias=True)  
  (sigmoid): Sigmoid()  
)
```

nz (dimension of the noise) = 100

num\_label = 10

Noise:  size = (batch\_size, 100, 1, 1)

Label:  size = (batch\_size, 10, 1, 1)

The first ten values in noise are replaced by the label one hot vector.

Noise with label: 

2. Sample random noise and generate 100 conditional images for each digit (0-9). Save total 1000 outputs in the assigned folder path for further evaluation. We will provide a digit classifier to evaluate your output images.
3. You should name your output digit images as the following format:  
(The first number of your filename indicates the corresponding digit label)
4. We will evaluate your generated output by the classification accuracy with a pretrained digit classifier, and we have provided the model architecture [digit\_classifier.py] and the weight [Classifier.pth] for you to test. (15%)

acc = 0.87500

5. Show 10 images for each digit (0-9) in your report. You can put all 100 outputs in one image with columns indicating different digits and rows indicating different noise inputs. [see the below example] (5%)



## Problem 3: DANN

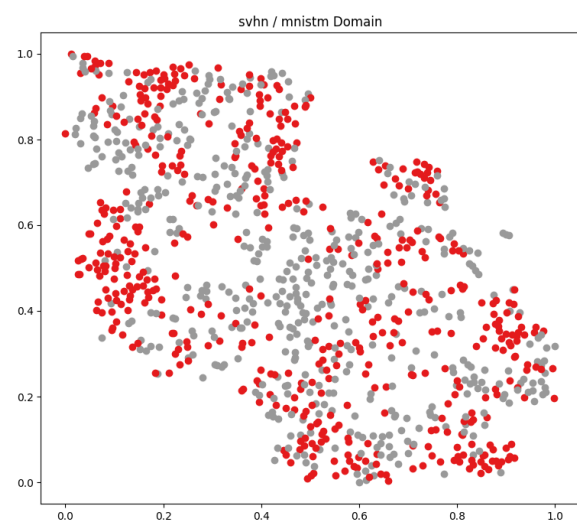
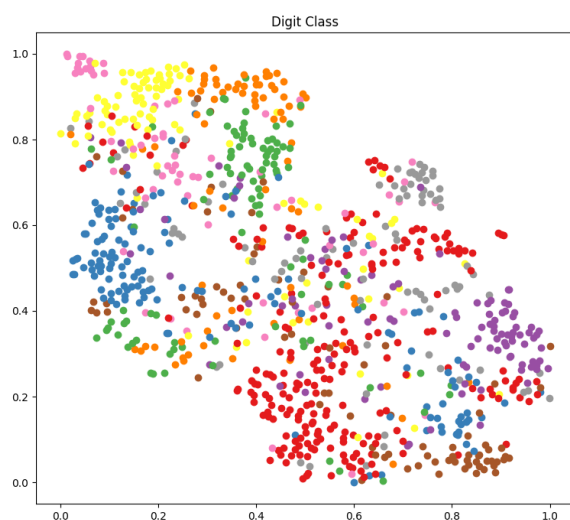
1. Compute the accuracy on target domain, while the model is trained on source domain only. (lower bound) (3%)
2. Compute the accuracy on target domain, while the model is trained on source and target domain. (domain adaptation) (4+9%)
3. Compute the accuracy on target domain, while the model is trained on target domain only. (upper bound) (3%)

	SVHN → MNIST-M	MNIST-M → USPS	USPS → SVHN
Trained on source	0.3872	0.1609	0.1045
DANN	0.4857	0.9068	0.1662
Trained on target	0.9847	0.9632	0.9437

4. Visualize the latent space by mapping the testing images to 2D space with t-SNE and use different colors to indicate data of (a) different digit classes 0-9 and (b) different domains (source/target). (9%)

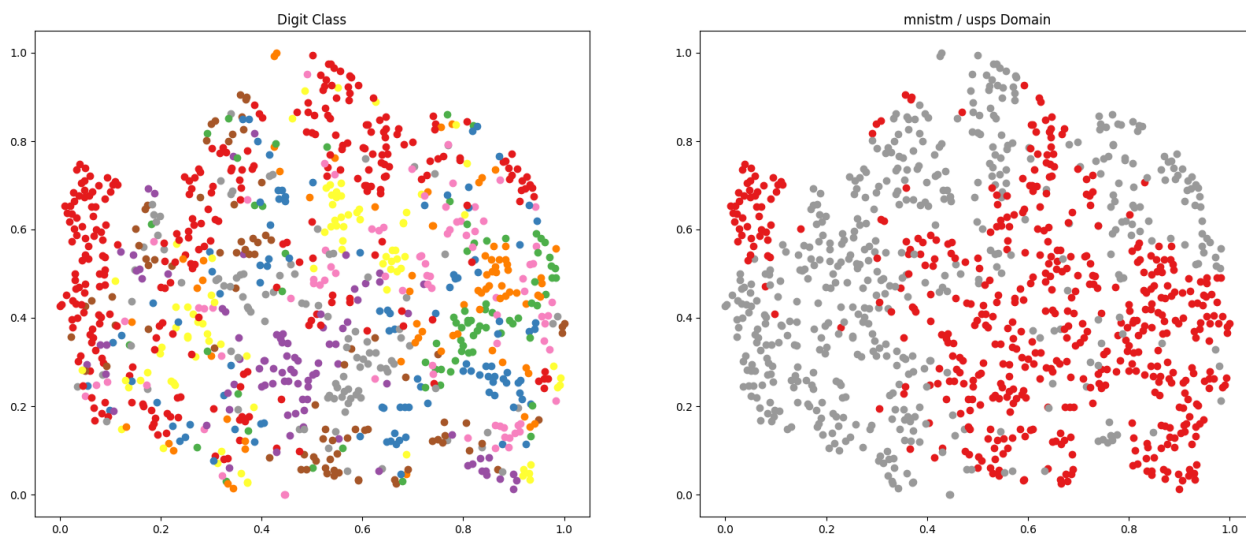
Run `tsne.py <target>` after downloading necessary models  
<target> is a string, either `mnistm`, `usps` or `svhn`.

(a) source: `svhn`, target: `mnistm`



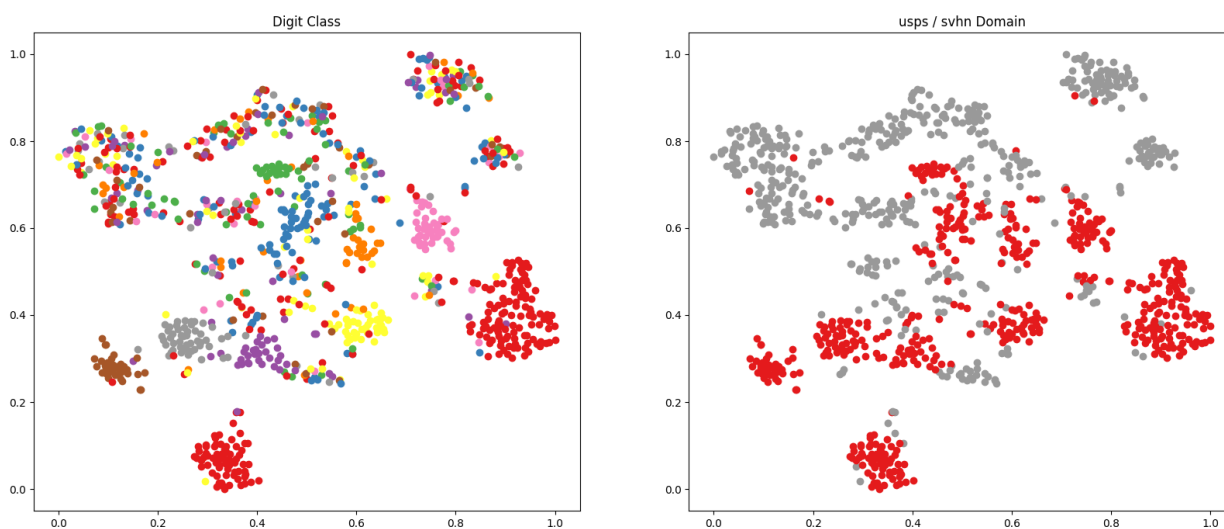


(b) source: mnistm, target: usps



(c) source: usps, target: svhn

As shown below, the feature extractor can still distinguish data from different domain, which leads to the bad performance on prediction of target testing data.



5. Describe the implementation details of your model and discuss what you've observed and learned from implementing DANN. (7%)

```
FeatureExtractor(  
  (conv): 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): ReLU(inplace=True)  
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (5): ReLU(inplace=True)  
    (6): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (8): ReLU()  
    (9): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (10): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (11): ReLU(inplace=True)  
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (13): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (14): ReLU(inplace=True)  
    (15): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (16): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (17): ReLU(inplace=True)  
    (18): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (19): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (20): ReLU(inplace=True)  
    (21): AdaptiveAvgPool2d(output_size=(1, 1))  
  )  
)  
  
LabelPredictor(  
  (layer): Sequential(  
    (0): Linear(in_features=256, out_features=512, bias=True)  
    (1): ReLU()  
    (2): Linear(in_features=512, out_features=512, bias=True)  
    (3): ReLU()  
    (4): Linear(in_features=512, out_features=10, bias=True)  
  )  
)  
  
DomainClassifier(  
  (layer): Sequential(  
    (0): Linear(in_features=256, out_features=512, bias=True)  
    (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU()  
    (3): Linear(in_features=512, out_features=512, bias=True)  
    (4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (5): ReLU()  
    (6): Linear(in_features=512, out_features=512, bias=True)  
    (7): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (8): ReLU()  
    (9): Linear(in_features=512, out_features=512, bias=True)  
    (10): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (11): ReLU()  
    (12): Linear(in_features=512, out_features=1, bias=True)  
  )  
)
```

Initially I started with a 5-layer feature extractor, and got bad results. Increasing number of epochs or number of kernels in each convolution layer didn't work for me.

Then I went deeper with 7 layers, and got better results.

However, the USPS dataset has less number of images than the others and has grayscale images, which led to the unsatisfactory performance in scenario 3 (usps -> svhn).

## References:

### DCGAN / ACGAN:

1. [https://pytorch.org/tutorials/beginner/dcgan\\_faces\\_tutorial.html](https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html)
2. <https://github.com/thtang/DLCV2018SPRING/tree/master/hw4>

### DANN

3. <https://github.com/fungtion/DANN>
4. [https://colab.research.google.com/drive/1cTdIDT\\_fsBWGbaljhPSmBI6gwkwtQ2H#scrollTo=b5cFq\\_TgWlQ](https://colab.research.google.com/drive/1cTdIDT_fsBWGbaljhPSmBI6gwkwtQ2H#scrollTo=b5cFq_TgWlQ)