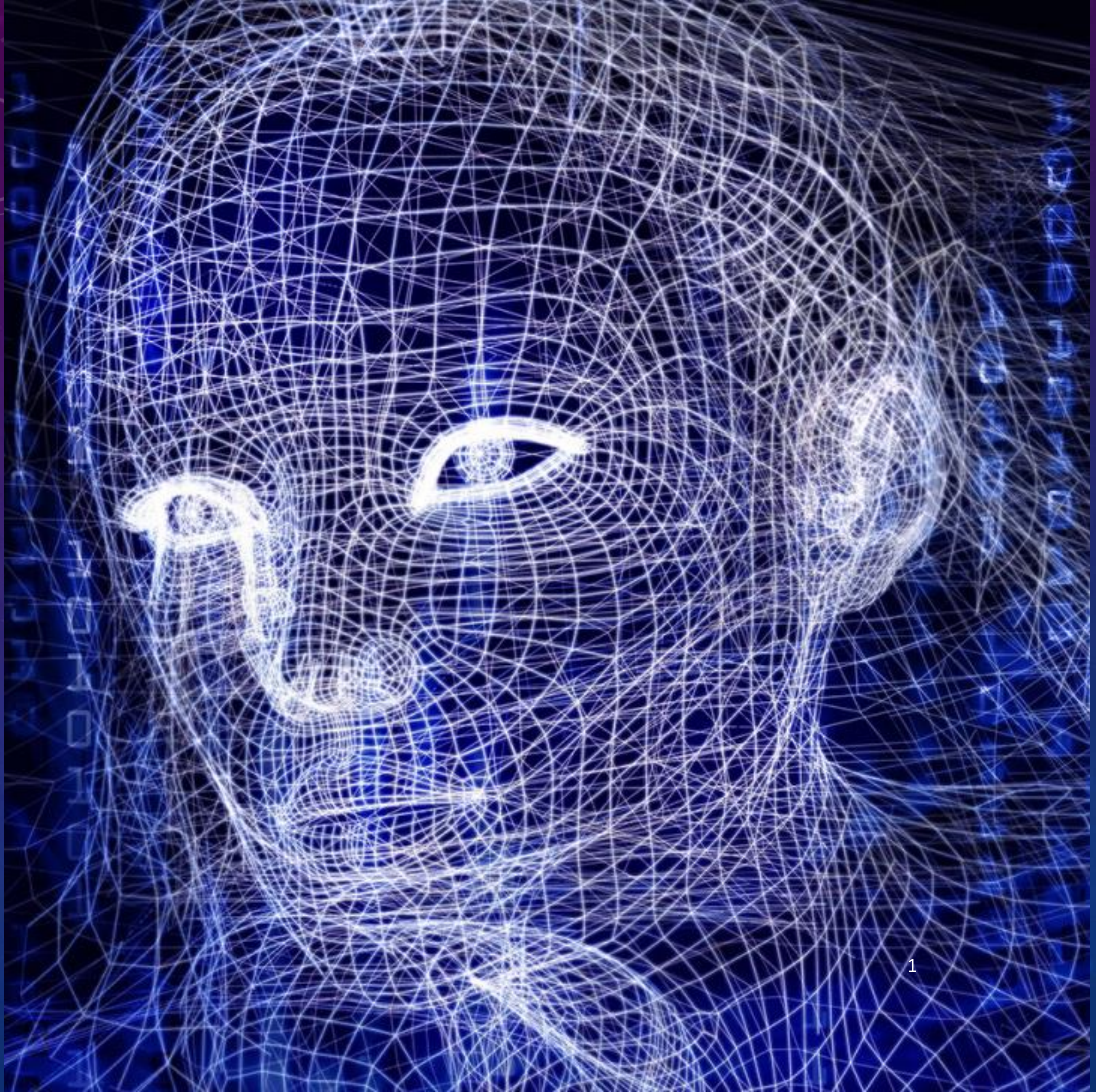
COMPUTER VISION AND ITS APPLICATIONS

ANOMALY DETECTION EXAMPLE AND EXERCISE

徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of Science and Technology



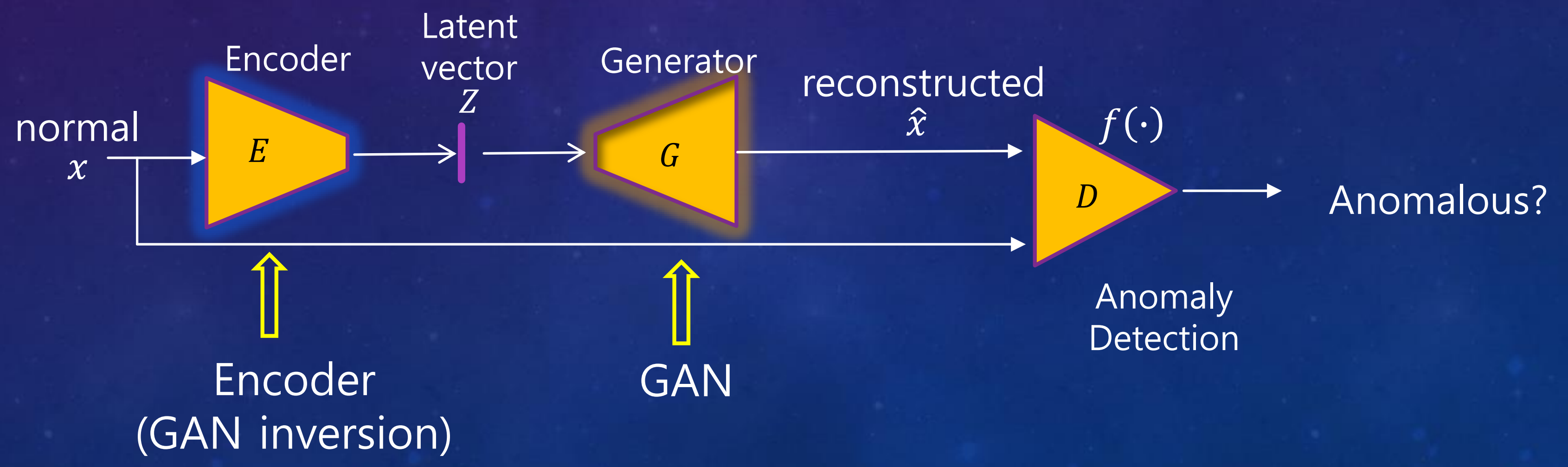
Recall : f-AnoGAN

- The generator learns the representation of the normal data

  1. Select large volume of normal images x.
  2. Train the generative model, i.e., generator G and discriminator D, on the normal images x..
  3. Do GAN inversion. Train an encoder E which can map the image to the latent vector z.
  4. The encoder E and the generator G are used to generate the synthetic normal data $\hat{x}$.
  5. Compare test image and its reconstructed image to get anomaly score.

  Salient feature: Use GAN inversion to find the mapping of query image to latent space

  Anomaly score = feature residual error + image reconstruction error



Recall : f-AnoGAN

- The generator learns the representation of normal image distribution
- The discriminator outputs the Wasserstein distance between generated and real data distribution, not a measure of realness of a given image.
- Use normal image only for the WGAN-GP training
- Latent vector follows the normal distribution.
- Generator & discriminator follow the ResNet network. (In exercise, DCGAN are used instead.)

Normal image　Latent vector

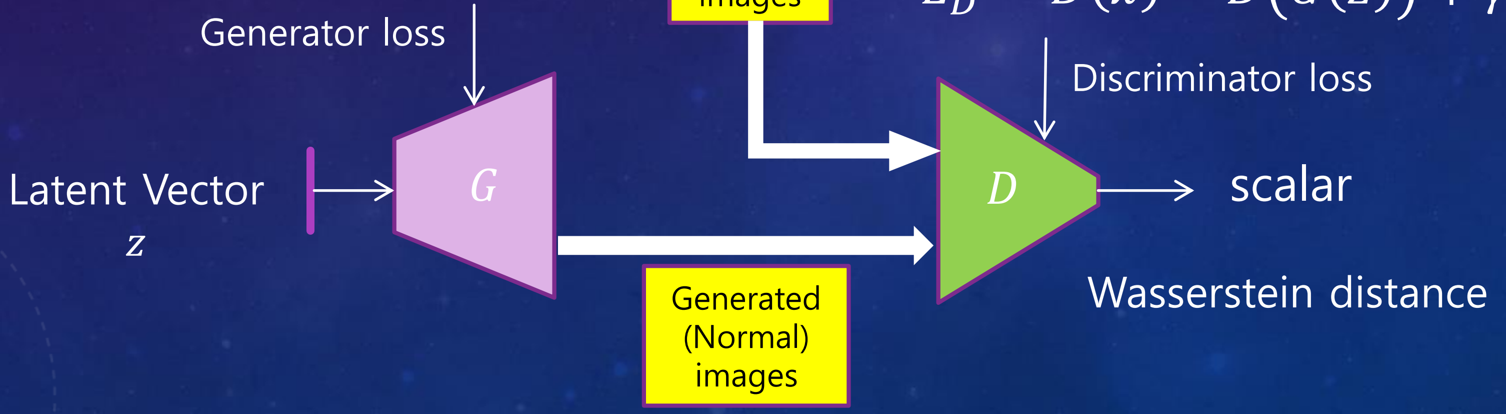
$random\ \epsilon$,
0< ϵ <1

$$\hat{x} = \epsilon x + (1-\epsilon)G(z)$$

$$L_G = D(G(z))$$

$$L_D = D(x) - D\big(G(z)\big) + \gamma(\|\nabla_{\hat{x}} D(\hat{x})\| - 1)^2$$

Gradient Penalty

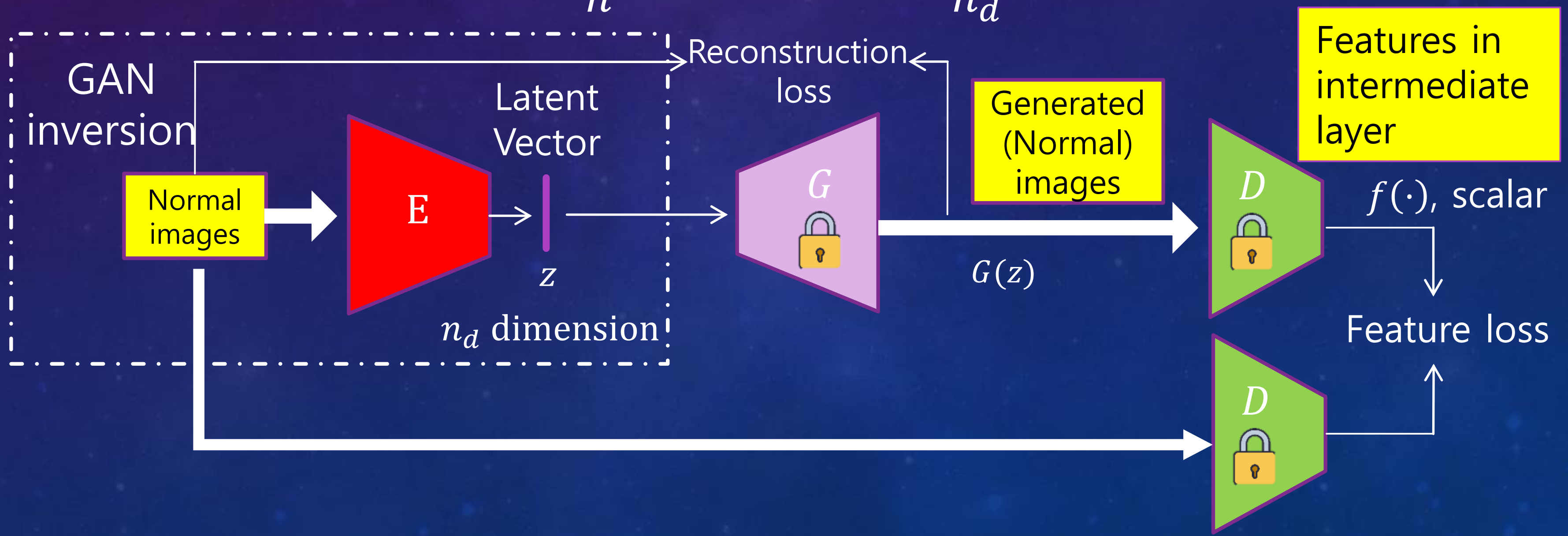


Recall : f-AnoGAN

- Learn the mapping of image to a vector in latent space where the generator is trained from.
- Parameters in generator and discriminator are fixed.
- The generated image shall closely resemble to the input normal image
- The discriminator outputs the features in the intermediate layer, not Wasserstein distance nor a measure of image realness.

Encoder loss — Reconstruction loss — Feature loss

$$L_E(x) = \frac{1}{n} \cdot \|x - G(z)\|^2 + \frac{\kappa}{n_d} \cdot \|f(x) - f(G(z)\|^2$$

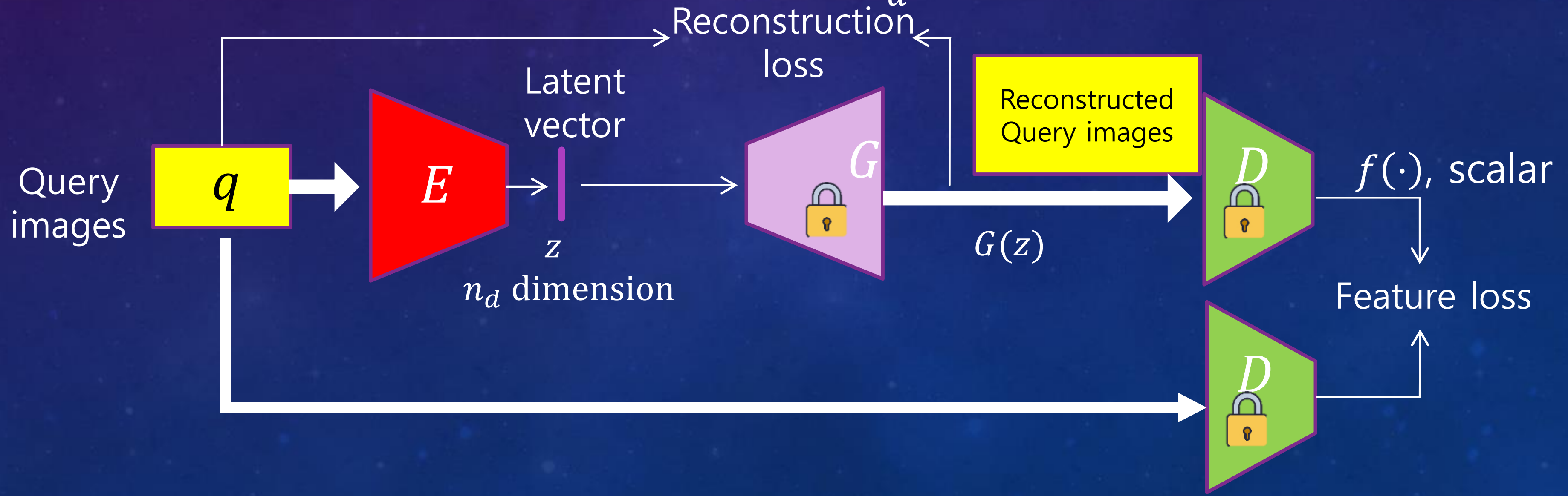


Recall : f-AnoGAN

- Anomaly score (AS) for image-level detection is the sum of reconstruction loss and feature loss of query image.
- If the query image is a normal image, the AS shall be as smaller as possible.
- The AS of anomalous query image shall be significantly different from the AS of normal images.
- For the pixel-level anomaly localization, we use the absolute value of pixel-wise residual, $\hat{A}_R = |x - G(z)|$

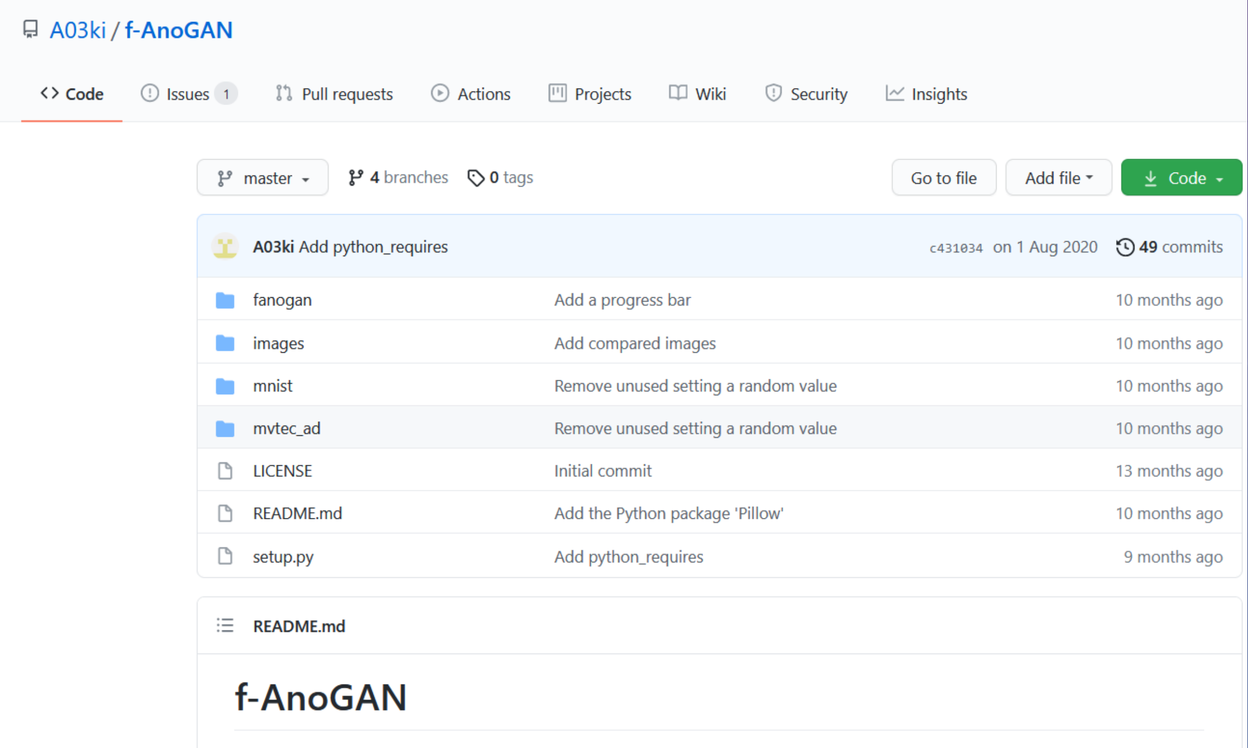
Anomaly Score, Reconstruction loss, Feature loss

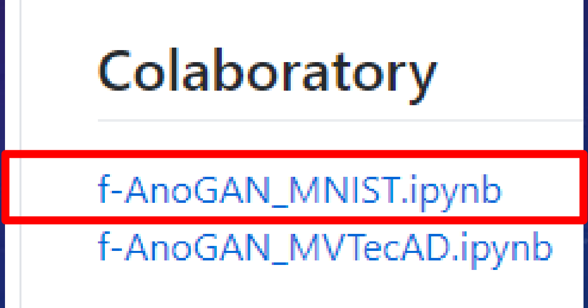
$$AS(q) = \frac{1}{n} \cdot \|x - G(E(q))\|^2 + \frac{\kappa}{n_d} \cdot \|f(q) - f(G(E(q))\|^2$$



Exercise_MNIST : f-AnoGAN

- Please visit https://github.com/A03ki/f-AnoGAN .
- Scroll down to the bottom of page and enter “f-AnoGAN_MNIST.ipynb”





Exercise_MNIST : f-AnoGAN - Pre-requisites

Pre-requisites :

Follow the instruction to download the codes of f-AnoGAN and setup the environments.

Step: 0

Please push "Open in playground" and run below in order.

1.

```
[23] !git clone https://github.com/A03ki/f-AnoGAN.git

Cloning into 'f-AnoGAN'...
remote: Enumerating objects: 194, done.
remote: Counting objects: 100% (194/194), done.
remote: Compressing objects: 100% (125/125), done.
remote: Total 194 (delta 108), reused 139 (delta 67), pack-reused 0
Receiving objects: 100% (194/194), 220.66 KiB | 3.87 MiB/s, done.
Resolving deltas: 100% (108/108), done.
```

2.

```
[24] %cd f-AnoGAN

/content/f-AnoGAN/mnist/f-AnoGAN/f-AnoGAN
```

3.

```
[25] !python setup.py install
```

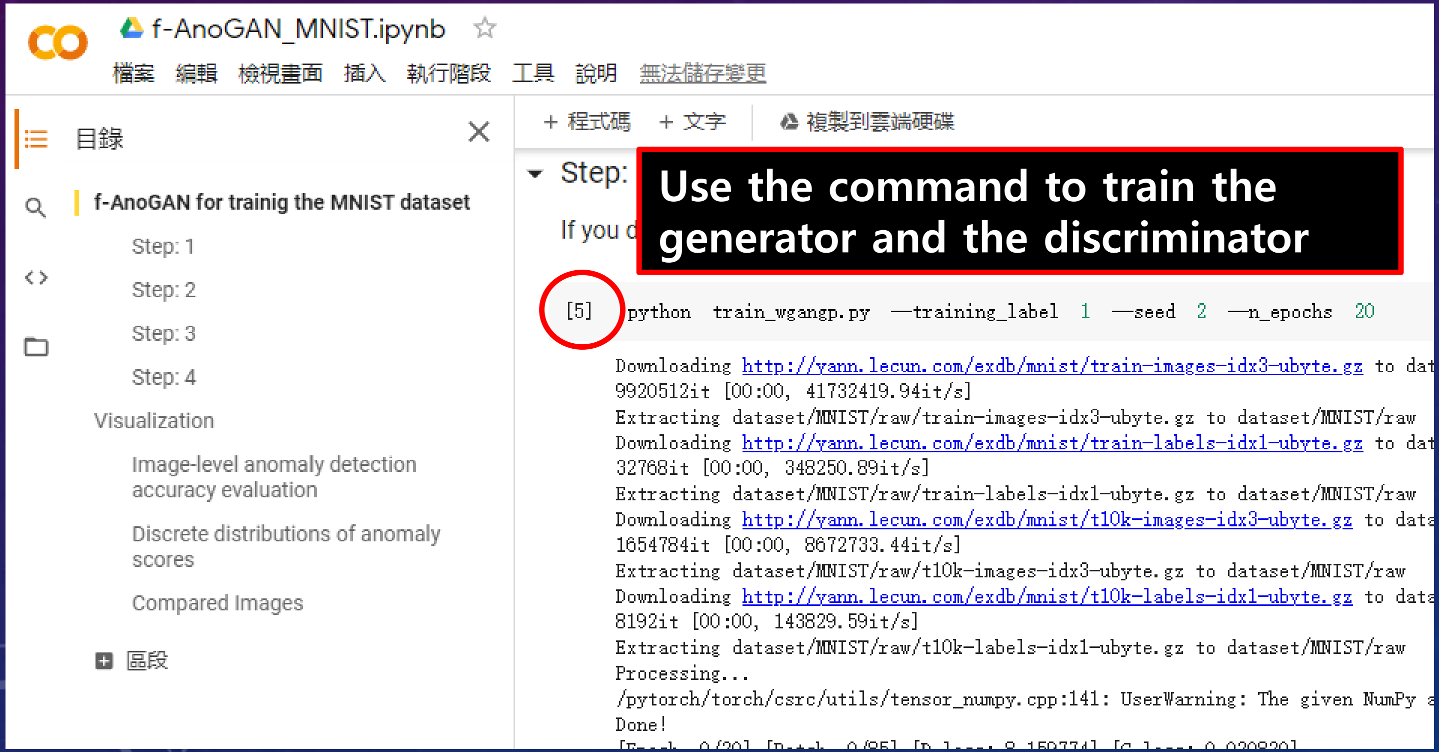
```
[4] %cd mnist

/content/f-AnoGAN/mnist
```

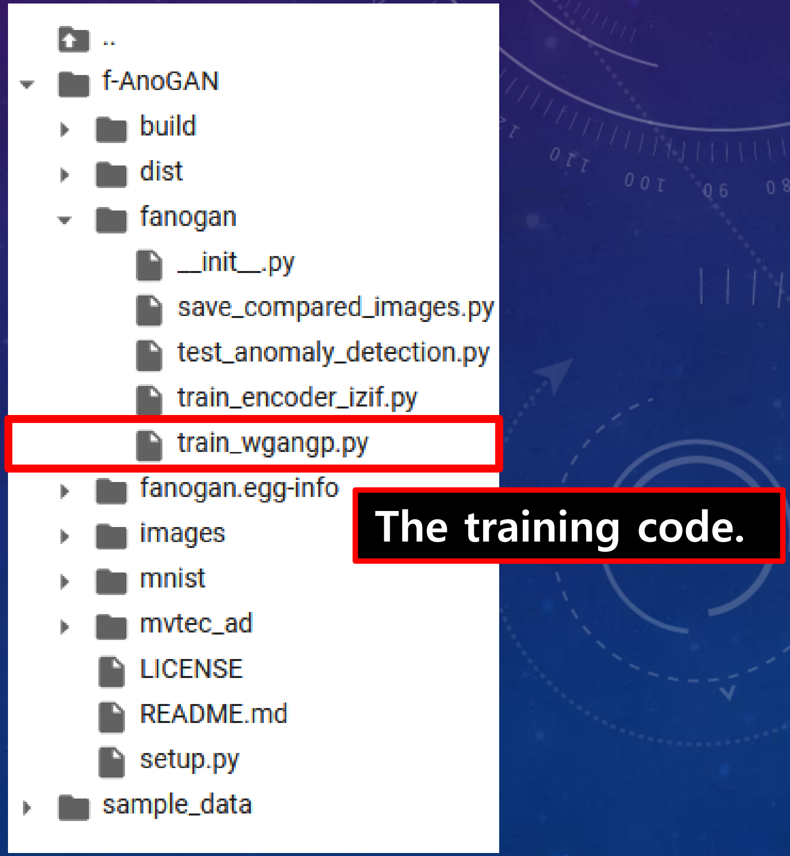
4.

Exercise_MNIST : f-AnoGAN – Train WGANGP (1/5)

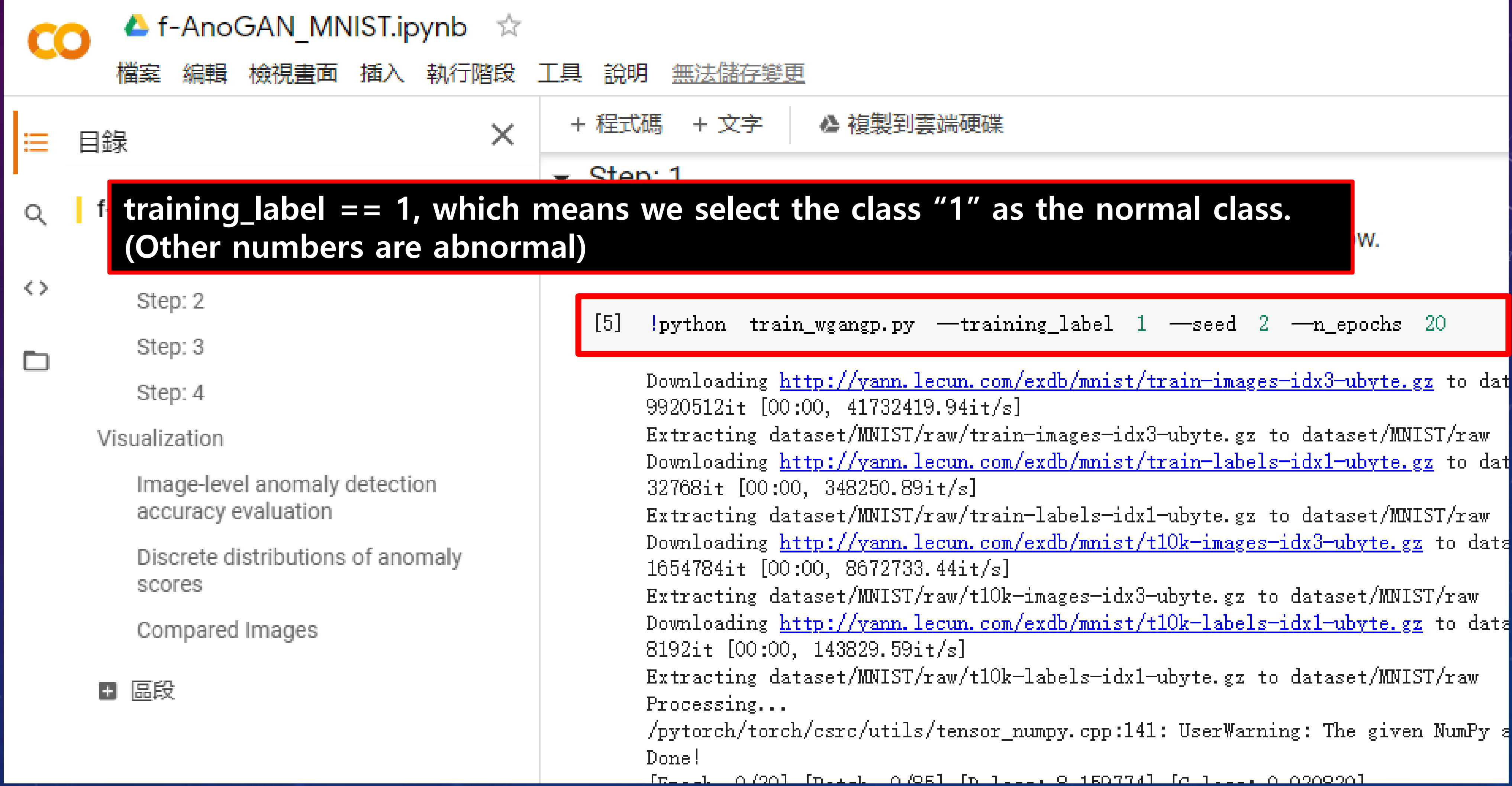
Step 1. Train WGANGP model

Use the command “python train_wgangp.py … ” to train the generator and the discriminator. All the codes can be visualized in “f-AnaGAN/fanongan/train_wgangp.py”





Exercise_MNIST : f-AnoGAN – Train WGANGP (2/5)



Exercise_MNIST : f-AnoGAN – Train WGANGP (3/5)

```
optimizer_D.zero_grad()

# Sample noise as generator input
z = torch.randn(imgs.shape[0], opt.latent_dim, device=device)

# Generate a batch of images
fake_imgs = generator(z)
```

During training the generator and the discriminator, we use the random noise z to generate the synthesized normal data.

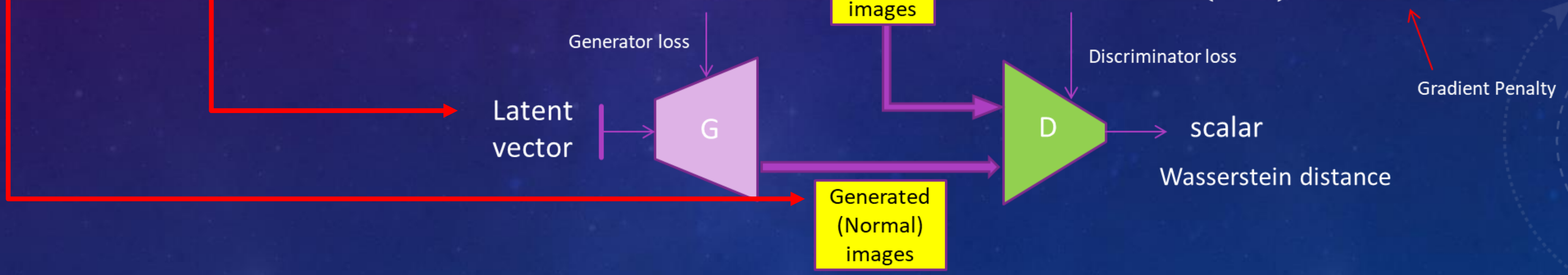
The generator G aims to synthesize the normal data from the random noise z

Normal image　Latent vector　$random\ \epsilon$, $0 < \epsilon < 1$

$$\hat{x} = \epsilon x + (1-\epsilon)G(z)$$

$$L_G = D(G(z))$$

$$L_D = D(x) - D\big(G(z)\big) + \gamma(\|\nabla_{\hat{x}} D(\hat{x})\| - 1)^2$$



Exercise_MNIST : f-AnoGAN – Train WGANGP (4/5)

Train the discriminator.

```
# Real images
real_validity = discriminator(real_imgs)
# Fake images
fake_validity = discriminator(fake_imgs.detach())
# Gradient penalty
gradient_penalty = compute_gradient_penalty(discriminator,
                                            real_imgs.data,
                                            fake_imgs.data,
                                            device)
# Adversarial loss
d_loss = (-torch.mean(real_validity) + torch.mean(fake_validity)
          + lambda_gp * gradient_penalty)

d_loss.backward()
optimizer_D.step()

optimizer_G.zero_grad()
```

$D(x)$ denotes the output of the real image.

$D(G(x))$ denotes the output of the generated image.

The loss function, WGAN-GP

Normal image Latent vector

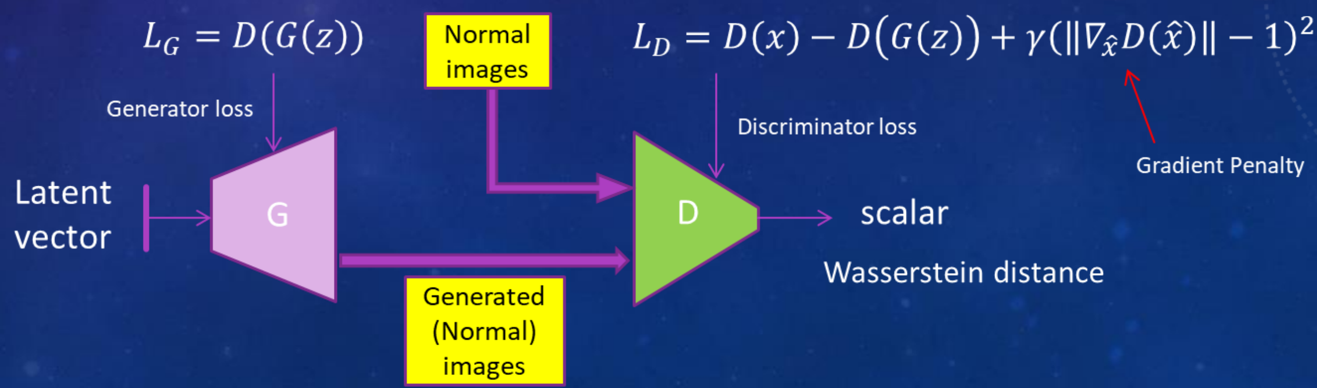
$random\ \epsilon$, $0 < \epsilon < 1$

$$\hat{x} = \epsilon x + (1-\epsilon)G(z)$$

$$L_G = D(G(z))$$

$$L_D = D(x) - D\big(G(z)\big) + \gamma(\|\nabla_{\hat{x}} D(\hat{x})\| - 1)^2$$

Gradient penalty is $\nabla_{\hat{x}} D(\hat{x})$



Exercise_MNIST : f-AnoGAN – Train WGANGP (5/5)

Train the generator

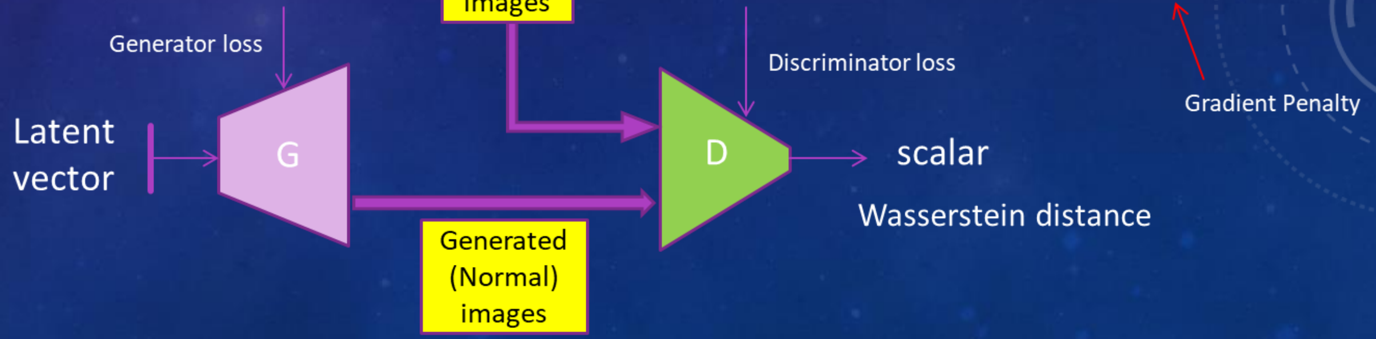
```python
# Train the generator and output log every n_critic steps
if i % opt.n_critic == 0:

    # -----------------
    #  Train Generator
    # -----------------

    # Generate a batch of images
    fake_imgs = generator(z)
    # Loss measures generator's ability to fool the discriminator
    # Train on fake images
    fake_validity = discriminator(fake_imgs)
    g_loss = -torch.mean(fake_validity)

    g_loss.backward()
    optimizer_G.step()

    print(f"[Epoch {epoch:{padding_epoch}}/{opt.n_epochs}] "
          f"[Batch {i:{padding_i}}/{len(dataloader)}] "
          f"[D loss: {d_loss.item():3f}] "
          f"[G loss: {g_loss.item():3f}]")

    if batches_done % opt.sample_interval == 0:
        save_image(fake_imgs.data[:25],
                   f"results/images/{batches_done:06}.png",
                   nrow=5, normalize=True)

    batches_done += opt.n_critic
```

G tries to fool D → $L_G = D(G(z))$

Normal image　Latent vector　*random* ϵ , $0 < \epsilon < 1$

$$\hat{x} = \epsilon x + (1-\epsilon)G(z)$$

$$L_G = D(G(z)) \qquad L_D = D(x) - D(G(z)) + \gamma(\|\nabla_{\hat{x}} D(\hat{x})\| - 1)^2$$

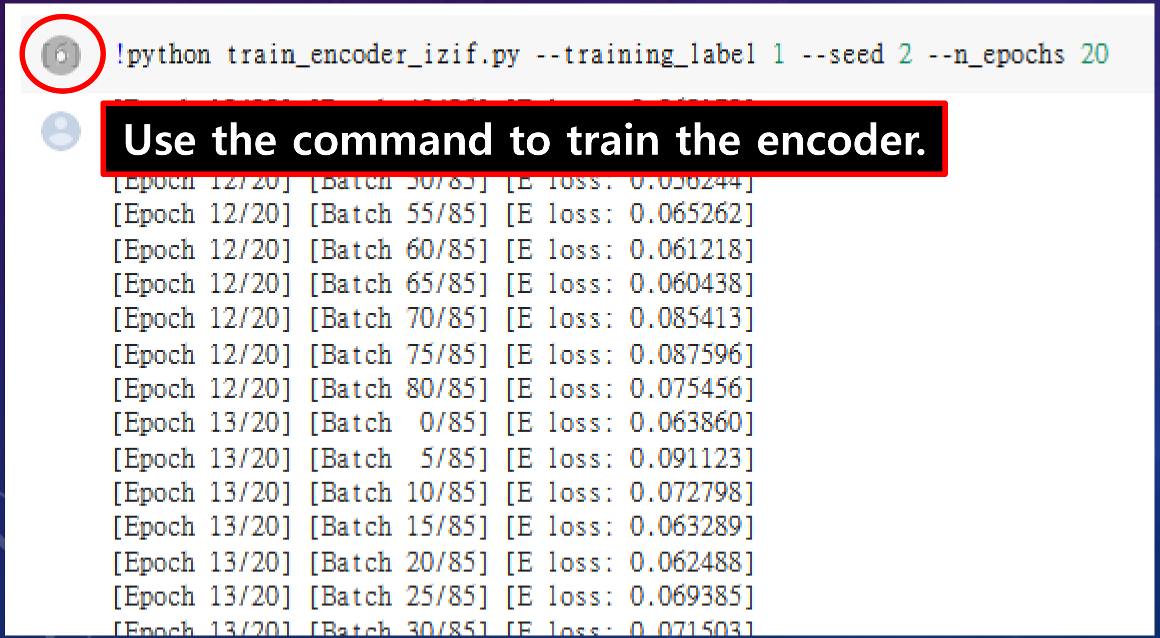


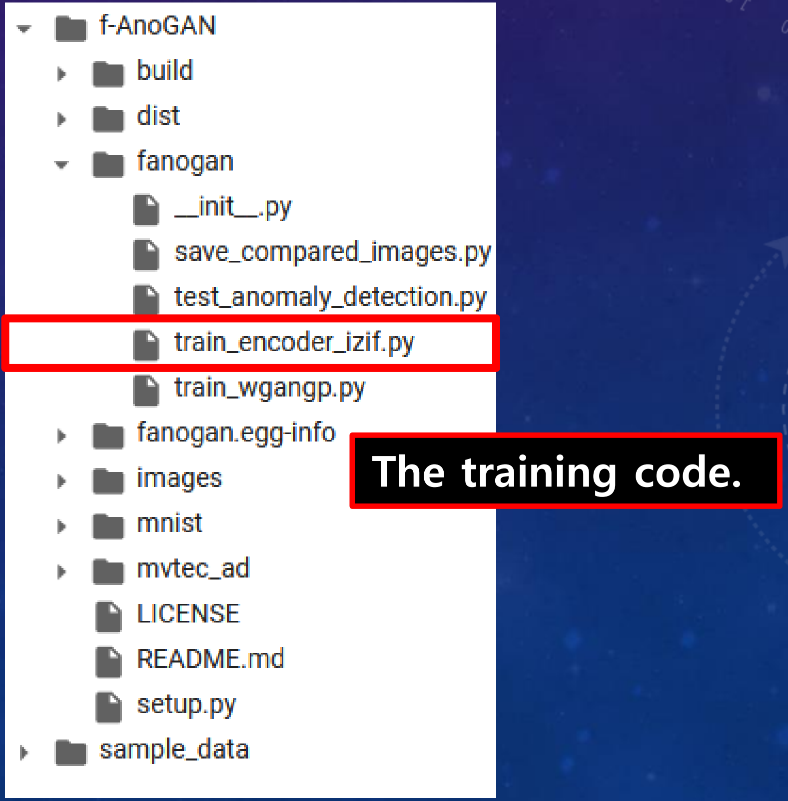
Exercise_MNIST: f-AnoGAN – Train Encoder (1/2)

Step 2. Train encoder model

The encoder aims to learn how to map the real image to a latent vector, which can be used to generate the images by the generator.

Use the command "python train_encoder_izif.py … " to train the encoder. The training codes can be visualized in "f-AnaGAN/fanongan/train_encoder_izif.py"





Exercise_MNIST: f-AnoGAN – Train Encoder (2/2)

We use two objective functions to train the encoder:

1) Reconstruction loss
2) Feature loss.

```
# Real features
real_features = discriminator.forward_features(real_imgs)
# Fake features
fake_features = discriminator.forward_features(fake_imgs)

# izif architecture
loss_imgs = criterion(fake_imgs, real_imgs)
loss_features = criterion(fake_features, real_features)
e_loss = loss_imgs + kappa * loss_features

e_loss.backward()
optimizer_E.step()
```

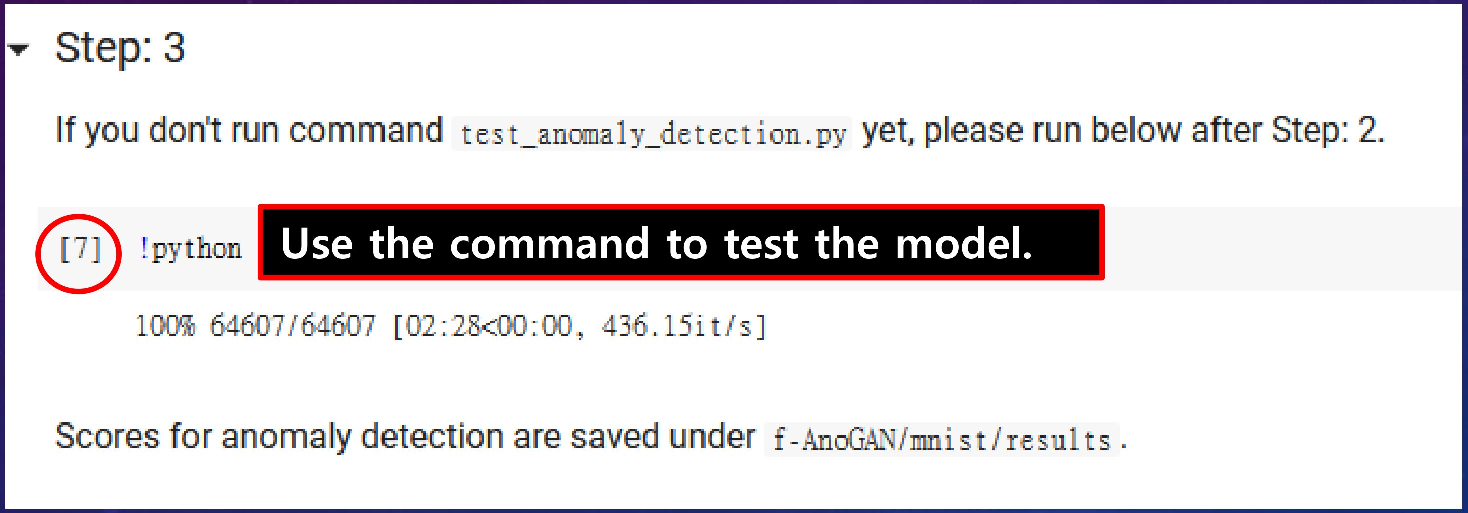
Reconstruction loss

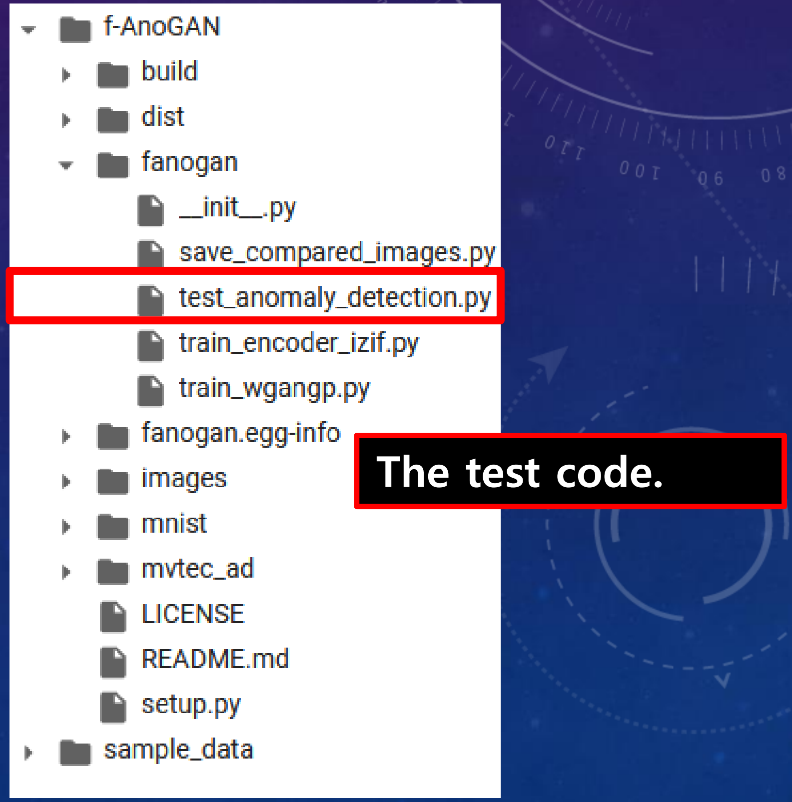
Feature loss

Exercise_MNIST: f-AnoGAN – Inference (1/2)

Step 3. Inference

Use the command “test_anamoly_detection.py … ” to test the performance. The inference codes can be visualized in “f-AnaGAN/fanongan/ test_anamoly_detection.py”





Exercise_MNIST: f-AnoGAN – Inference (2/2)

```python
def test_anomaly_detection(opt, generator, discriminator, encoder,
                           dataloader, device, kappa=1.0):
    generator.load_state_dict(torch.load("results/generator"))
    discriminator.load_state_dict(torch.load("results/discriminator"))
    encoder.load_state_dict(torch.load("results/encoder"))

    generator.to(device).eval()
    discriminator.to(device).eval()
    encode

    criterion = nn.MSELoss()

    with open("results/score.csv", "w") as f:
        f.write("label,img_distance,anomaly_score,z_distance\n")
```

Load the trained weights

```python
    for (img, label) in tqdm(dataloader):

        real_img = img.to(device)

        real_z = encoder(real_img)
        fake_img = generator(real_z)
        fake_z = encoder(fake_img)

        real_feature = discriminator.forward_features(real_img)
        fake_feature = discriminator.forward_features(fake_img)

        # Scores for anomaly detection
        img_distance = criterion(fake_img, real_img)
        loss_feature = criterion(fake_feature, real_feature)
        anomaly_score = img_distance + kappa * loss_feature

        z_distance = criterion(fake_z, real_z)
```

Calculate the error metrics
(distances and anomaly scores)

Exercise_MNIST: f-AnoGAN – Visualize (1/3)

Step 4. Obtain the test results and analyze, the codes can be visualized in "f-AnaGAN/fanongan/save_compared_images.py"

Step: 4

If you don't run command `save_compared_images.py` yet, please run below after Step: 2.

```
[8] !python save_compared_images.py --seed 4 --n_iters 0 --n_grid_lines 10
```

1.

Compared images are saved under `f-AnoGAN/mnist/results/images_diff`.

Visualization

Please run below after Step: 1~3.

```
[9] import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd
    from sklearn.metrics import roc_curve, precision_recall_curve, auc
```

2.

```
[10] df = pd.read_csv("results/score.csv")
     df
```

3.

```
df = pd.read_csv("results/score.csv")
df
```

4.

"1" is normal class

	label	img_distance	anomaly_score	z_distance
0	1	0.072017	0.129134	0.008652
1	1	0.033213	0.036116	0.003864
2	1	0.034159	0.042097	0.028874
3	1	0.030439	0.035003	0.003410
4	1	0.067693	0.095846	0.012227
...	...	...	...	...
64602	2	0.521851	0.815865	0.779736
64603	3	0.482058	0.622129	0.382944
64604	4	0.436425	0.515557	0.188704
64605	5	0.376133	0.488694	0.144208
64606	6	0.638045	0.930252	0.156074

64607 rows × 4 columns

others are abnormal

Exercise_MNIST: f-AnoGAN – Visualize (2/3)

Calculate the results.

5.

```
[26] trainig_label = 1
     labels = np.where(df["label"].values == trainig_label, 0, 1)
     anomaly_score = df["anomaly_score"].values
     img_distance = df["img_distance"].values
     z_distance = df["z_distance"].values
```

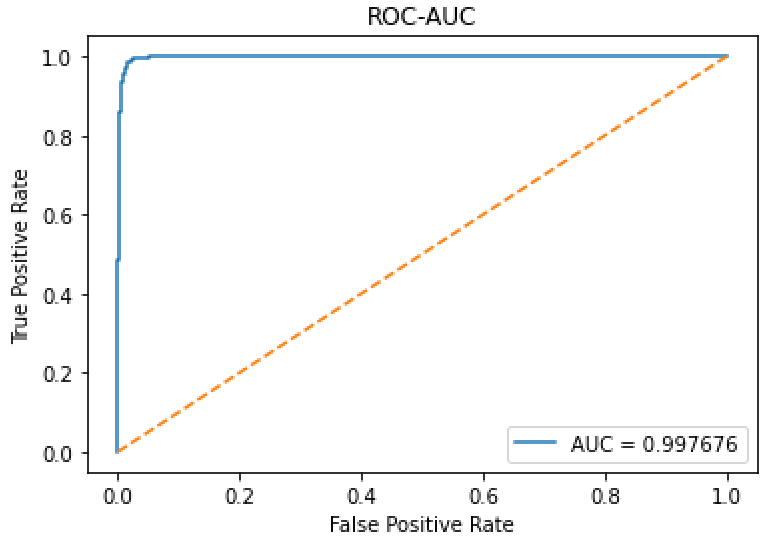
6.

```
fpr, tpr, _ = roc_curve(labels, img_distance)
precision, recall, _ = precision_recall_curve(labels, img_distance)
roc_auc = auc(fpr, tpr)
pr_auc =  auc(recall, precision)
```

Image-level anomaly detection accuracy evaluation

7.

```
[13] plt.plot(fpr, tpr, label=f"AUC = {roc_auc:3f}")
     plt.plot([0, 1], [0, 1], linestyle="--")
     plt.title("ROC-AUC")
     plt.xlabel("False Positive Rate")
     plt.ylabel("True Positive Rate")
     plt.legend()
     plt.show()
```

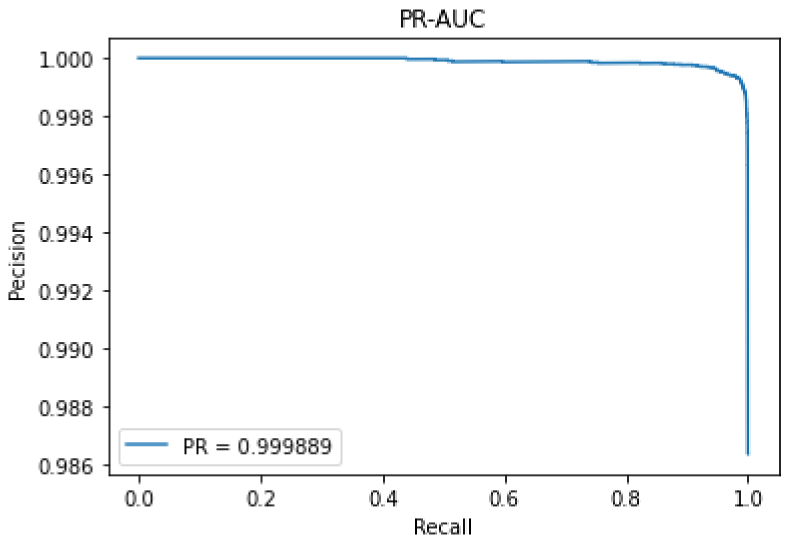


Exercise_MNIST: f-AnoGAN – Visualize (3/3)

Draw the ROC curve, data distribution and the real/generated sample test.

8.

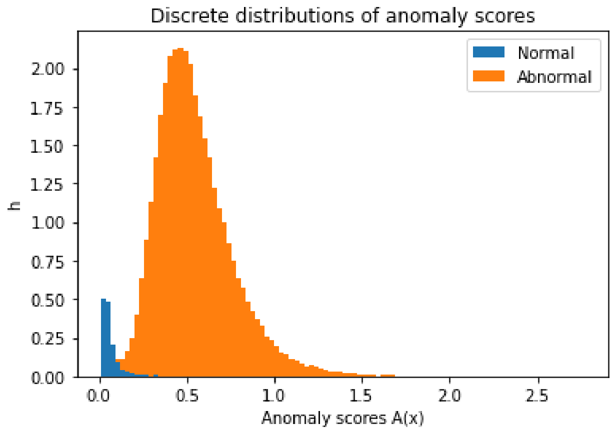
```
[14] plt.plot(recall, precision, label=f"PR = {pr_auc:3f}")
     plt.title("PR-AUC")
     plt.xlabel("Recall")
     plt.ylabel("Pecision")
     plt.legend()
     plt.show()
```



Discrete distributions of anomaly scores

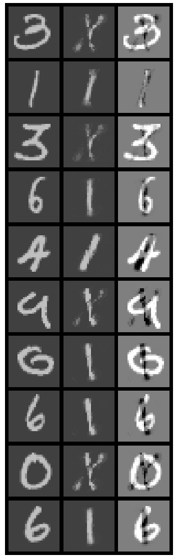
9.

```
[15] plt.hist([anomaly_score[labels == 0], anomaly_score[labels == 1]],
              bins=100, density=True, stacked=True,
              label=["Normal", "Abnormal"])
     plt.title("Discrete distributions of anomaly scores")
     plt.xlabel("Anomaly scores A(x)")
     plt.ylabel("h")
     plt.legend()
     plt.show()
```



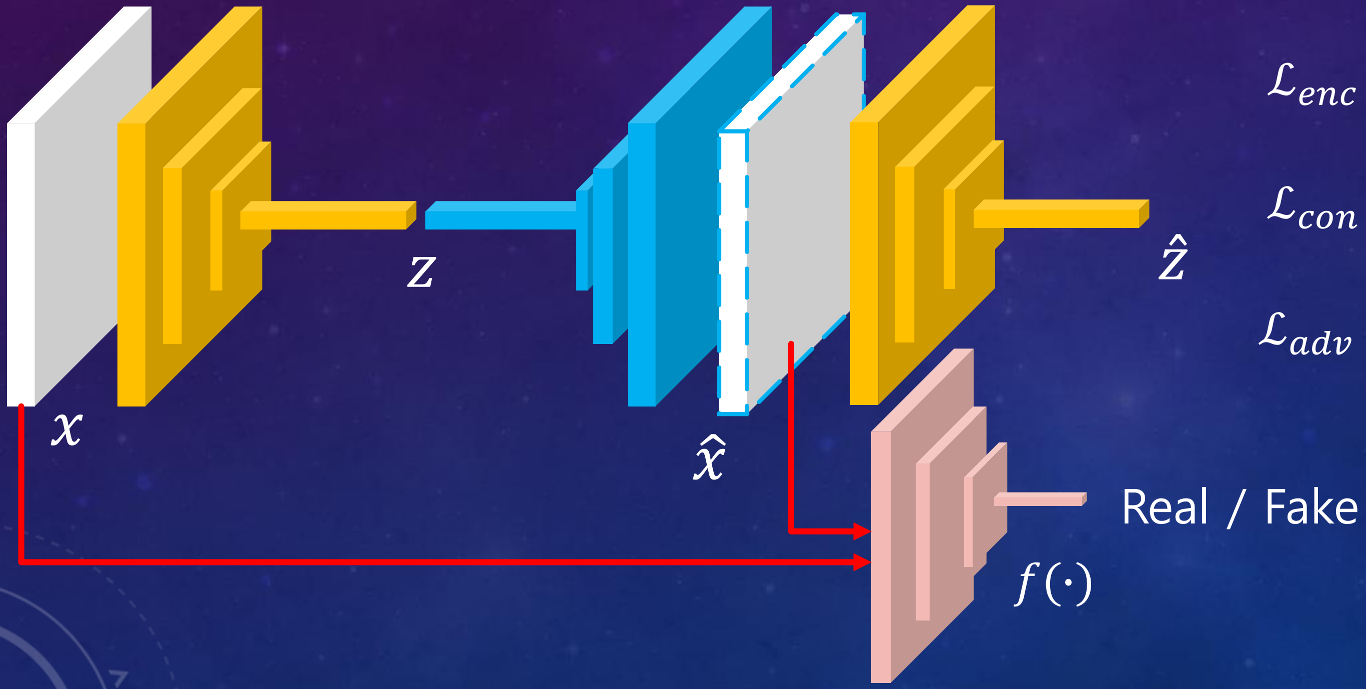

```
[17] from PIL import Image

     Image.open("results/images_diff/000010.png")
```

10.



Homework : GANomaly

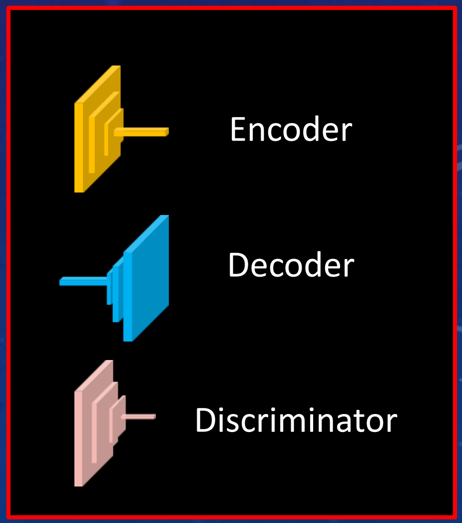
- GANomaly detects anomalies with a novel encoder-decoder-encoder structure.
- The detection criterion is similar to f-AnoGAN.
- The distance of latent vector z and $\hat{z}$ is defined as the anomaly score.



$$\mathcal{L}_{enc} = \|z - \hat{z}\|$$

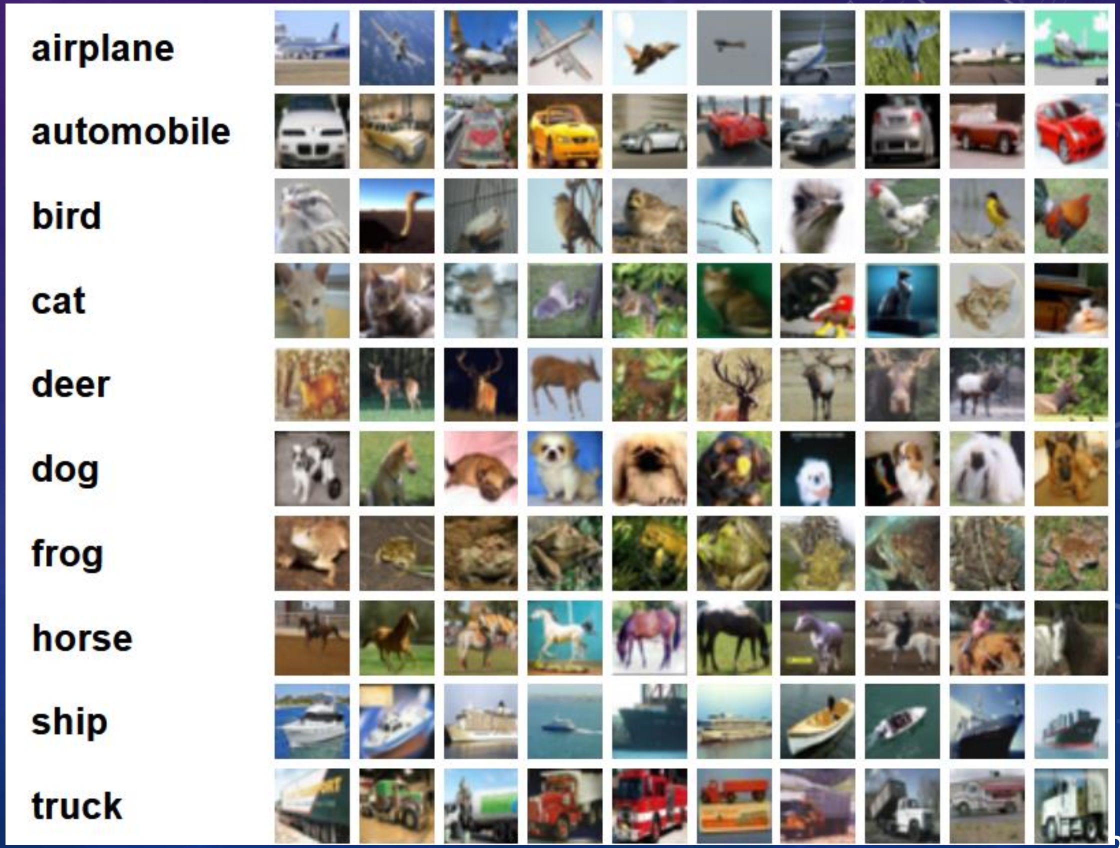
$$\mathcal{L}_{con} = \|x - \hat{x}\|_1$$

$$\mathcal{L}_{adv} = \|f(x) - f(\hat{x})\|_2$$



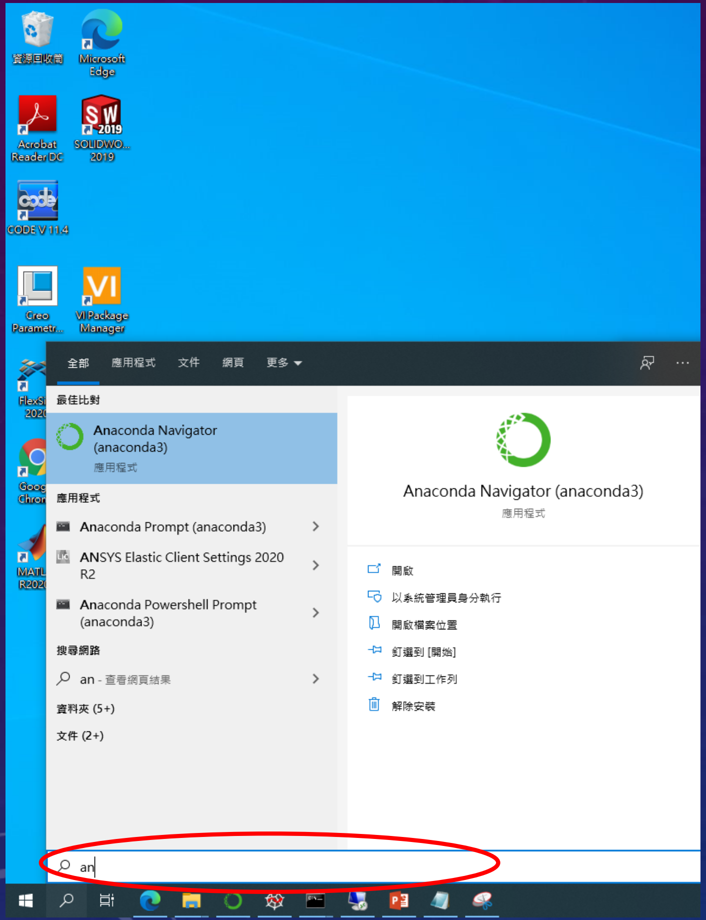
Homework : GANomaly

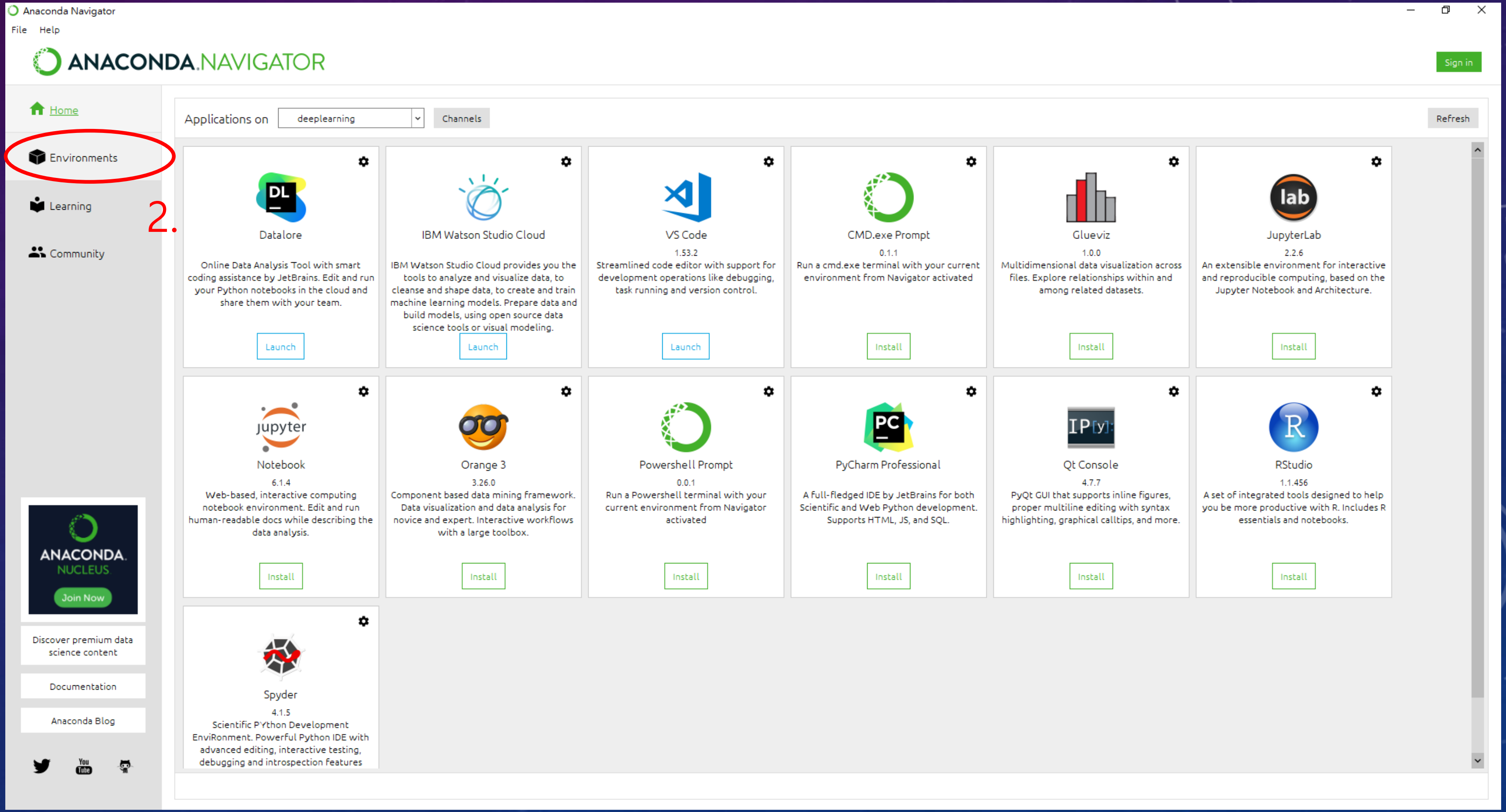
The code use Cifar10 dataset as default.
The dataset consists of 60k 32x32 color images in 10 classes, with 6k per class. (5k training / 1k test)
GANomaly define 'car' as the abnormal class (All 6000 images)
The other classes follow 5k training 1k test protocol.
Training : 45,000 normal images
Test : 6,000 abnormal + 9,000 normal images



Homework：GANomaly

0. Please download “ganomaly-master_modified.zip” from moodle.

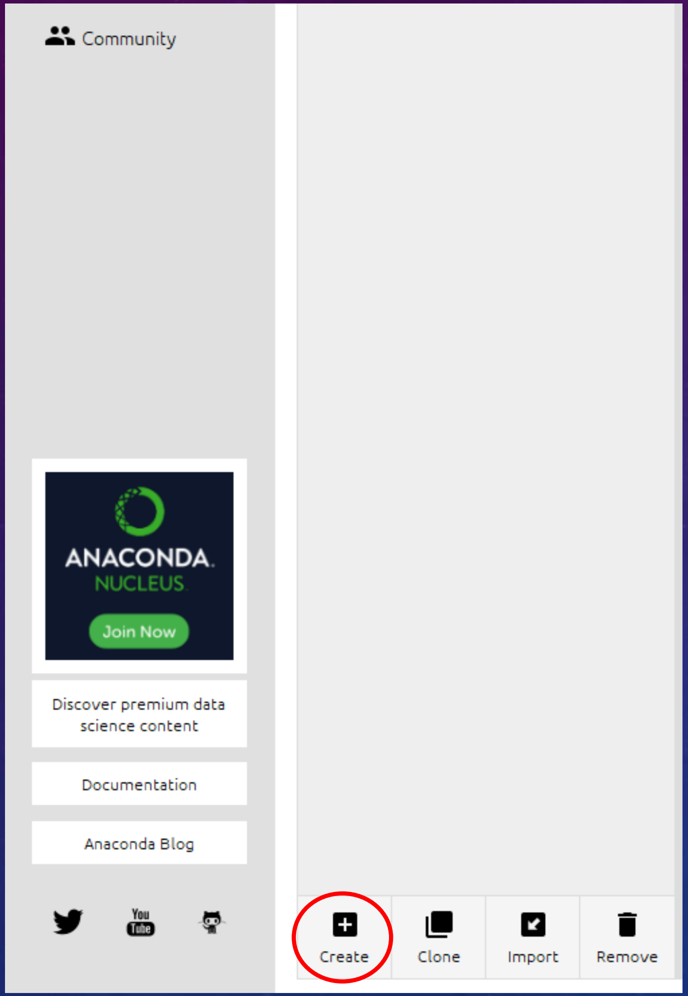




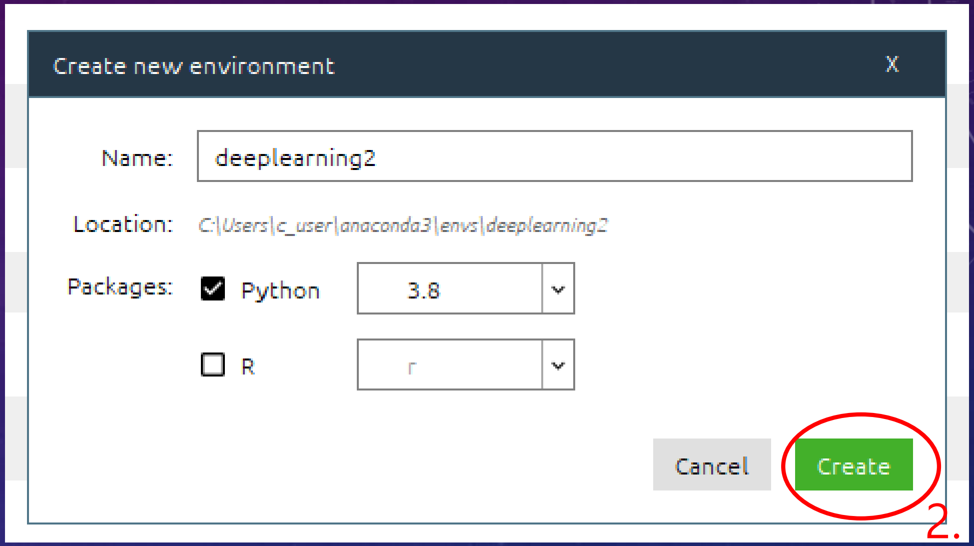
2.

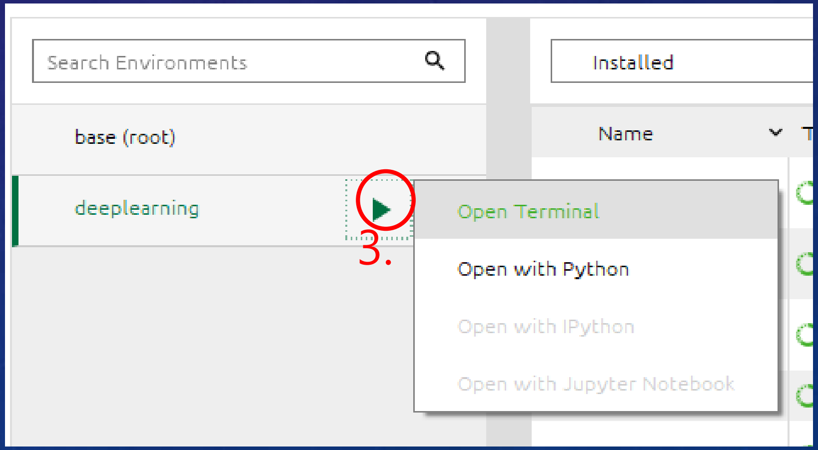
1. Enter “Anaconda”

Homework : GANomaly



1.





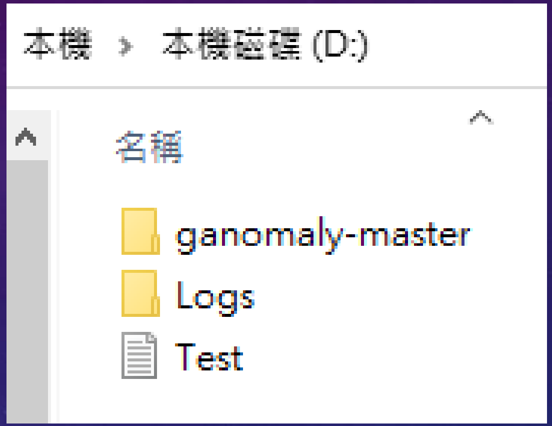
Homework : GANomaly

C:\Windows\system32\cmd.exe

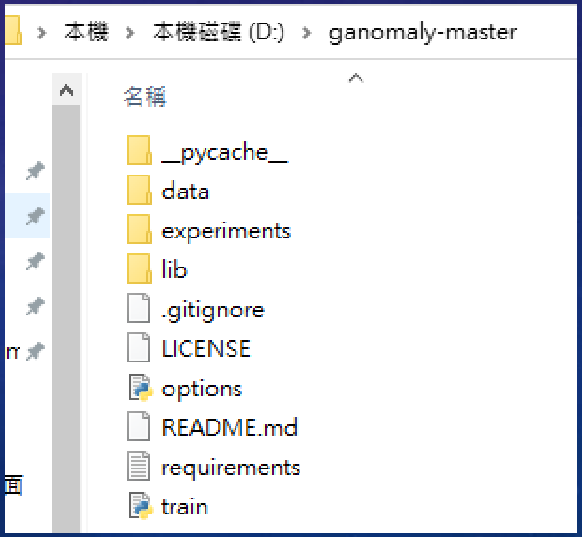
```
(deeplearning) C:\Users\c_user>D:
```

```
(deeplearning) D:\>cd ganomaly-master
```

```
(deeplearning) D:\ganomaly-master>
```



Change directory



Homework : GANomaly

本機 › 本機磁碟 (D:) › ganomaly-master

名稱

- __pycache__
- data
- experiments
- lib
- .gitignore
- LICENSE
- options
- README.md
- requirements
- train

Please remove these items due to version issues.
"torch" and "scipy"

1.

```
asn1crypto==0.24.0
certifi==2019.6.16
cffi==1.12.3
chardet==3.0.4
cryptography==2.7
cycler==0.10.0
idna==2.8
joblib==0.13.2
kiwisolver==1.1.0
matplotlib==3.1.0
numpy==1.16.4
olefile==0.46
Pillow>=7.1.0
pycparser==2.19
pyOpenSSL==19.0.0
pyparsing==2.4.0
PySocks==1.7.0
python-dateutil==2.8.0
pytz==2019.1
pyzmq==18.0.2
requests==2.22.0
scikit-learn==0.21.2
scipy==1.3.0
six==1.12.0
torch==1.2.0
torchfile==0.1.0
torchvision==0.4
tornado==6.0.3
tqdm==4.33.0
urllib3==1.25.3
visdom==0.1.8.8
websocket-client==0.56.0
```

2. After 1., we install the required packages

2-1.

```
(deeplearning) D:\ganomaly-master>pip install -r requirements.txt
```

2-2.

```
(deeplearning) D:\ganomaly-master>
conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
```

2-3.

```
(deeplearning) D:\ganomaly-master>pip install tqdm scipy tensorboardX tensorflow
```

Homework : GANomaly

After steps above, training is ready.

```
(deeplearning) D:\ganomaly-master>python train.py
```

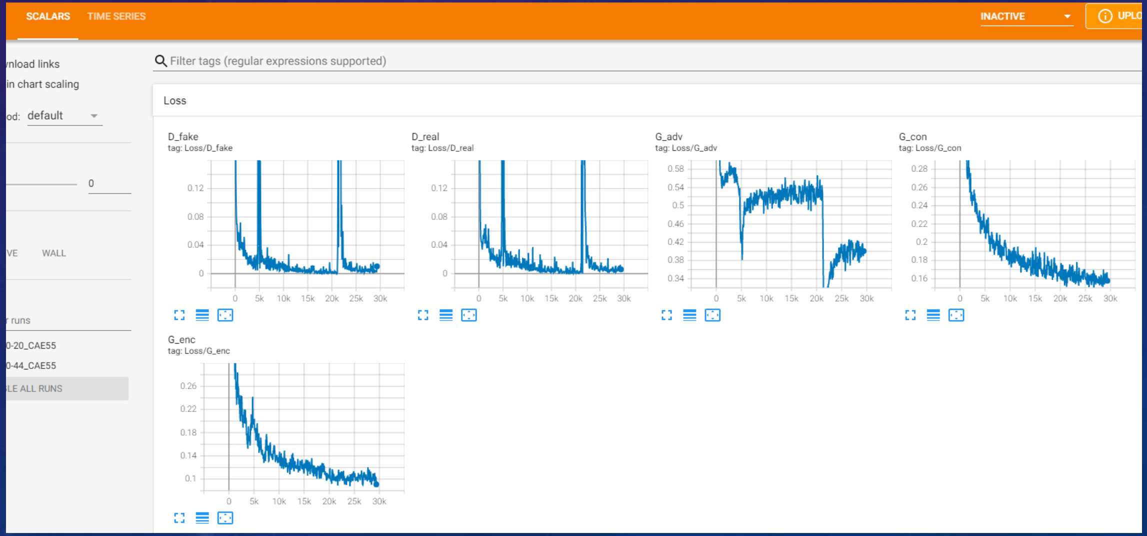
Please open another terminal and start Tensorboard. (After training is started.)

```
(deeplearning) D:\ganomaly-master>cd runs

(deeplearning) D:\ganomaly-master\runs>tensorboard --logdir . --port 6006
```

```
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.5.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

*Tensorboard is a tool that collects the loss values for further analysis

Please copy the address and paste it to your browser.



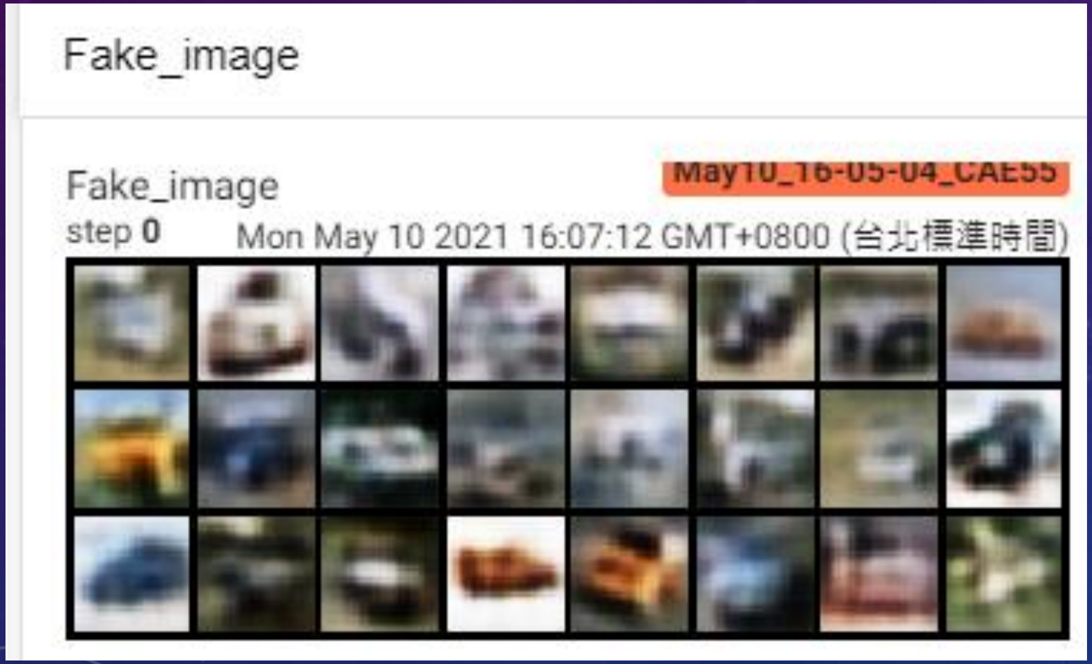
Homework 4-2: GANomaly

Please show your training loss curve, and discuss the peaks of the graphs.
(Hint : The relation between G and D)

What happened at the blue circles ?
(same iteration assumed)

Homework : GANomaly

Please show the images of epoch 1, 10 or more epochs. And discuss what may be happening? Is there any relation between loss and fake images?



Homework : GANomaly

Please compare your results with the pre-trained model.

The pertained model is already in “output/ganomaly/cifar10/train/weights/”


```
(pytorch_high) D:\ganomaly-master>python train.py --load_weights --phase test
Files already downloaded and verified
Files already downloaded and verified
>> Training model Ganomaly.
   Loaded weights.
OrderedDict([('Avg Run Time (ms/batch)', 9.974679946899414), ('roc', 0.692968462962963)])
```

Phase test for pretrain model

```
(pytorch_high) D:\ganomaly-master>python train.py --load_weights --phase valid
Files already downloaded and verified
Files already downloaded and verified
>> Training model Ganomaly.
   Loaded weights.
```

Phase valid for your model

< Your results >

Homework : GANomaly

Given a pre-trained model shown in last page, please compare the anomaly scores and the generated image between the pre-trained model and the model your trained, and write down the observation in your .docx file.

1. Please compare the generated image from your model and the pretrained model
2. Please compare the anomaly scores between these two models.
3. Write down what have you been observed.

Anomaly Score

Pre-trained

Yours