

hw-gan

May 30, 2023

. Generative adversarial networks

GAN

```
[1]: import os
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
import torchvision.transforms as tt
import torch
import torch.nn as nn
import cv2
from tqdm.notebook import tqdm
from torchvision.utils import save_image
from torchvision.utils import make_grid
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set(style='darkgrid', font_scale=1.2)
```

0.1 1. (1)
Flickr Faces, (1024 1024).

,
DataLoader
(1024 , 128)
Google Drive. data loader

```
[2]: from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
[3]: sample_dir = 'data_gan'
os.makedirs(sample_dir, exist_ok=True)
```

```
[4]: !unzip "/content/drive/MyDrive/Colab Notebooks/GUN/archive.zip" -d '/content/
↳data_gan/'
```

Archive: /content/drive/MyDrive/Colab Notebooks/GUN/archive.zip

inflating: /content/data_gan/faces_dataset_small/00055.png
inflating: /content/data_gan/faces_dataset_small/00237.png
inflating: /content/data_gan/faces_dataset_small/00240.png
inflating: /content/data_gan/faces_dataset_small/00241.png
inflating: /content/data_gan/faces_dataset_small/00242.png
inflating: /content/data_gan/faces_dataset_small/00243.png
inflating: /content/data_gan/faces_dataset_small/00244.png
inflating: /content/data_gan/faces_dataset_small/00245.png
inflating: /content/data_gan/faces_dataset_small/00246.png
inflating: /content/data_gan/faces_dataset_small/00247.png
inflating: /content/data_gan/faces_dataset_small/00248.png
inflating: /content/data_gan/faces_dataset_small/00249.png
inflating: /content/data_gan/faces_dataset_small/00253.png
inflating: /content/data_gan/faces_dataset_small/00255.png
inflating: /content/data_gan/faces_dataset_small/00257.png
inflating: /content/data_gan/faces_dataset_small/00258.png
inflating: /content/data_gan/faces_dataset_small/00259.png
inflating: /content/data_gan/faces_dataset_small/00260.png
inflating: /content/data_gan/faces_dataset_small/00261.png
inflating: /content/data_gan/faces_dataset_small/00262.png
inflating: /content/data_gan/faces_dataset_small/00263.png
inflating: /content/data_gan/faces_dataset_small/00264.png
inflating: /content/data_gan/faces_dataset_small/00265.png
inflating: /content/data_gan/faces_dataset_small/00266.png
inflating: /content/data_gan/faces_dataset_small/00267.png
inflating: /content/data_gan/faces_dataset_small/00268.png
inflating: /content/data_gan/faces_dataset_small/00269.png
inflating: /content/data_gan/faces_dataset_small/00270.png
inflating: /content/data_gan/faces_dataset_small/00271.png
inflating: /content/data_gan/faces_dataset_small/00272.png
inflating: /content/data_gan/faces_dataset_small/00273.png
inflating: /content/data_gan/faces_dataset_small/00274.png
inflating: /content/data_gan/faces_dataset_small/00275.png
inflating: /content/data_gan/faces_dataset_small/00276.png
inflating: /content/data_gan/faces_dataset_small/00277.png
inflating: /content/data_gan/faces_dataset_small/00278.png
inflating: /content/data_gan/faces_dataset_small/00279.png
inflating: /content/data_gan/faces_dataset_small/00280.png
inflating: /content/data_gan/faces_dataset_small/00281.png
inflating: /content/data_gan/faces_dataset_small/00282.png

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```

inflating: /content/data_gan/faces_dataset_small/04969.png
inflating: /content/data_gan/faces_dataset_small/04970.png
inflating: /content/data_gan/faces_dataset_small/04971.png
inflating: /content/data_gan/faces_dataset_small/04972.png
inflating: /content/data_gan/faces_dataset_small/04973.png
inflating: /content/data_gan/faces_dataset_small/04974.png
inflating: /content/data_gan/faces_dataset_small/04975.png
inflating: /content/data_gan/faces_dataset_small/04976.png
inflating: /content/data_gan/faces_dataset_small/04977.png
inflating: /content/data_gan/faces_dataset_small/04978.png
inflating: /content/data_gan/faces_dataset_small/04979.png
inflating: /content/data_gan/faces_dataset_small/04980.png
inflating: /content/data_gan/faces_dataset_small/04981.png
inflating: /content/data_gan/faces_dataset_small/04982.png
inflating: /content/data_gan/faces_dataset_small/04983.png
inflating: /content/data_gan/faces_dataset_small/04984.png
inflating: /content/data_gan/faces_dataset_small/04985.png
inflating: /content/data_gan/faces_dataset_small/04986.png
inflating: /content/data_gan/faces_dataset_small/04987.png
inflating: /content/data_gan/faces_dataset_small/04988.png
inflating: /content/data_gan/faces_dataset_small/04989.png
inflating: /content/data_gan/faces_dataset_small/04990.png
inflating: /content/data_gan/faces_dataset_small/04991.png
inflating: /content/data_gan/faces_dataset_small/04992.png
inflating: /content/data_gan/faces_dataset_small/04993.png
inflating: /content/data_gan/faces_dataset_small/04994.png
inflating: /content/data_gan/faces_dataset_small/04995.png
inflating: /content/data_gan/faces_dataset_small/04996.png
inflating: /content/data_gan/faces_dataset_small/04997.png
inflating: /content/data_gan/faces_dataset_small/04998.png
inflating: /content/data_gan/faces_dataset_small/04999.png

```

```

        ,
        DataLoader

```

```

[5]: def get_dataloader(image_size, batch_size):
    """
    Builds dataloader for training data.
    Use tt.Compose and tt.Resize for transformations
    :param image_size: height and width of the image
    :param batch_size: batch_size of the dataloader
    :returns: DataLoader object
    """
    # TODO: resize images, convert them to tensors and build dataloader

    INP_DIR = '/content/data_gan'

    transform_img = tt.Compose([

```

```

        tt.Resize(image_size),
        tt.CenterCrop(image_size),
        tt.ToTensor(),
        tt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

data_set = ImageFolder(root=INP_DIR, transform=transform_img)

data_loader = DataLoader(data_set, batch_size, shuffle=True, num_workers=2)

return data_loader

```

```

[6]: #TODO: build dataloader and transfer it to device
image_size = 128
batch_size = 32

train_dl = get_dataloader(image_size, batch_size)

```

(32 .)

```

[7]: def denorm(img_tensors):
        return img_tensors * 0.5 + 0.5

def show_images(images, nmax=64):
    fig, ax = plt.subplots(figsize=(13, 13))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(denorm(images.detach()[:nmax]), nrow=8).permute(1, 2, 0))

def show_batch(dl, nmax=64):
    for images, _ in dl:
        show_images(images, nmax)
        break

```

```

[8]: show_batch(train_dl)

```



```

0.2      2.      (2      )
image_size)      ,      : *      (      3 x image_size x
      •      latent_size x 1 x 1      3 x image_size
      x image_size

```

```

[9]: device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
     print(f"Device: {device}.")

```

Device: cuda.

: Conv2d, LeakyReLU, BatchNorm2d, ConvTranspose2d, ReLU.

= 100

```

[10]: discriminator = nn.Sequential(

    # in: 3 x 128 x 128
    nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1, bias=False),
    nn.LeakyReLU(0.2, inplace=True),
    # out: 64 x 64 x 64

    nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(128),
    nn.LeakyReLU(0.2, inplace=True),
    # out: 128 x 32 x 32

```

```

nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1, bias=False),
nn.BatchNorm2d(256),
nn.LeakyReLU(0.2, inplace=True),
# out: 256 x 16 x 16

nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1, bias=False),
nn.BatchNorm2d(512),
nn.LeakyReLU(0.2, inplace=True),
# out: 512 x 8 x 8

nn.Conv2d(512, 1024, kernel_size=4, stride=2, padding=1, bias=False),
nn.BatchNorm2d(1024),
nn.LeakyReLU(0.2, inplace=True),
# out: 1024 x 4 x 4

nn.Conv2d(1024, 1, kernel_size=4, stride=1, padding=0, bias=False),
# out: 1 x 1 x 1

nn.Sigmoid()
)

```

```

[11]: latent_size = 100 # choose latent size

generator = nn.Sequential(
    # in: 128 x 1 x 1

    nn.ConvTranspose2d(latent_size, 1024, kernel_size=4, stride=1, padding=0,
↳bias=False),
    nn.BatchNorm2d(1024),
    nn.ReLU(True),
    # out: 1024 x 4 x 4

    nn.ConvTranspose2d(1024, 512, kernel_size=4, stride=2, padding=1,
↳bias=False),
    nn.BatchNorm2d(512),
    nn.ReLU(True),
    # out: 512 x 8 x 8

    nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1,
↳bias=False),
    nn.BatchNorm2d(256),
    nn.ReLU(True),
    # out: 256 x 16 x 16

    nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1,
↳bias=False),
    nn.BatchNorm2d(128),

```

```

nn.ReLU(True),
# out: 128 x 32 x 32

nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1, bias=False),
nn.BatchNorm2d(64),
nn.ReLU(True),
# out: 64 x 64 x 64

nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1, bias=False),
# out: 3 x 128 x 128

nn.Tanh()
)

```

```

GAN. : 1. : *
1 * 0 *
2. :
• 0
• ,
-
Adam,
BCELoss.
Learning rate 0.0002.

```

```

[12]: lr = 2e-4

gunD = discriminator.to(device)
gunG = generator.to(device)

criterion = nn.BCELoss()

optimizerD = torch.optim.Adam(discriminator.parameters(),
                               lr=lr, betas=(0.5, 0.999))
optimizerG = torch.optim.Adam(generator.parameters(),
                               lr=lr, betas=(0.5, 0.999))

```

```

[13]: def fit(mod_discriminator, mod_generator, opt_discriminator, opt_generator,
             criterion, epochs, lr):
    # Losses & scores
    losses_g = []
    losses_d = []
    real_scores = []
    fake_scores = []

```

```

for epoch in range(epochs):
    loss_d_per_epoch = []
    loss_g_per_epoch = []
    real_score_per_epoch = []
    fake_score_per_epoch = []

    for real_images, _ in tqdm(train_dl):
        # Train discriminator
        mod_discriminator.zero_grad()

        real_images = real_images.to(device)
        batch_s = real_images.size(0)

        label = torch.full((batch_s, ), 1, dtype=torch.float, device=device)
        real_preds = mod_discriminator(real_images).view(-1)
        real_loss = criterion(real_preds, label)
        cur_real_score = torch.mean(real_preds).item()
        real_loss.backward()

        # Generate fake images
        latent = torch.randn(batch_s, latent_size, 1, 1, device=device)
        fake_images = mod_generator(latent)

        # Pass fake images through discriminator
        label.fill_(0)
        fake_preds = mod_discriminator(fake_images.detach()).view(-1)
        fake_loss = criterion(fake_preds, label)
        fake_loss.backward()
        cur_fake_score = torch.mean(fake_preds).item()

        real_score_per_epoch.append(cur_real_score)
        fake_score_per_epoch.append(cur_fake_score)

        # Update discriminator weights
        loss_d = real_loss + fake_loss
        opt_discriminator.step()
        loss_d_per_epoch.append(loss_d.item())

        # Train generator
        mod_generator.zero_grad()
        label.fill_(1)

        # Try to fool the discriminator
        preds = mod_discriminator(fake_images).view(-1)
        loss_g = criterion(preds, label)
        loss_g.backward()

```



```

        # Update generator weights
        loss_g_per_epoch.append(loss_g.item())
        opt_generator.step()

    # Record losses & scores
    losses_g.append(np.mean(loss_g_per_epoch))
    losses_d.append(np.mean(loss_d_per_epoch))
    real_scores.append(np.mean(real_score_per_epoch))
    fake_scores.append(np.mean(fake_score_per_epoch))

    with torch.no_grad():
        z = torch.randn(batch_size, latent_size, 1, 1, device=device)
        fake_images = mod_generator(z)
        save_image(denorm(fake_images).view(fake_images.size(0), 3, 128, 128),
        ↪f"generated_images_{epoch+1}.png")

    # Log losses & scores (last batch)
    print("Epoch [{}/{}], loss_g: {:.4f}, loss_d: {:.4f}, real_score: {:.4f},
    ↪fake_score: {:.4f}".format(
        epoch+1, epochs,
        losses_g[-1], losses_d[-1], real_scores[-1], fake_scores[-1]))

    return losses_g, losses_d, real_scores, fake_scores, mod_discriminator,
    ↪mod_generator

```

6 : 50,70,80,90,100 110.
colab

```

[14]: checkpoint = torch.load('/content/drive/MyDrive/models_110.pth')
gunD.load_state_dict(checkpoint['discriminator'])
gunG.load_state_dict(checkpoint['generator'])

```

[14]: <All keys matched successfully>

```

[15]: data = torch.load('/content/drive/MyDrive/data_110.pth')
losses_g_110 = data['losses_g']
losses_d_110 = data['losses_d']
real_scores_110 = data['real_scores']
fake_scores_110 = data['fake_scores']

```

!

```

[ ]: losses_g, losses_d, real_scores, fake_scores, trained_discriminator,
    ↪trained_generator = fit(gunD, gunG, optimizerD, optimizerG, criterion, 10,
    ↪lr)

```

0%| | 0/99 [00:00<?, ?it/s]

Epoch [1/10], loss_g: 7.6455, loss_d: 0.1348, real_score: 0.9540, fake_score: 0.0431

0%| | 0/99 [00:00<?, ?it/s]

Epoch [2/10], loss_g: 7.4841, loss_d: 0.2255, real_score: 0.9549, fake_score: 0.0524

0%| | 0/99 [00:00<?, ?it/s]

Epoch [3/10], loss_g: 7.5663, loss_d: 0.0714, real_score: 0.9729, fake_score: 0.0259

0%| | 0/99 [00:00<?, ?it/s]

Epoch [4/10], loss_g: 9.7180, loss_d: 0.1539, real_score: 0.9578, fake_score: 0.0406

0%| | 0/99 [00:00<?, ?it/s]

Epoch [5/10], loss_g: 8.6273, loss_d: 0.2062, real_score: 0.9475, fake_score: 0.0532

0%| | 0/99 [00:00<?, ?it/s]

Epoch [6/10], loss_g: 8.4781, loss_d: 0.1055, real_score: 0.9649, fake_score: 0.0347

0%| | 0/99 [00:00<?, ?it/s]

Epoch [7/10], loss_g: 7.6398, loss_d: 0.0533, real_score: 0.9778, fake_score: 0.0217

0%| | 0/99 [00:00<?, ?it/s]

Epoch [8/10], loss_g: 9.1176, loss_d: 0.0818, real_score: 0.9727, fake_score: 0.0271

0%| | 0/99 [00:00<?, ?it/s]

Epoch [9/10], loss_g: 10.1473, loss_d: 0.3046, real_score: 0.9362, fake_score: 0.0638

0%| | 0/99 [00:00<?, ?it/s]

Epoch [10/10], loss_g: 9.1865, loss_d: 0.0982, real_score: 0.9663, fake_score: 0.0344

p.s.: 110 .

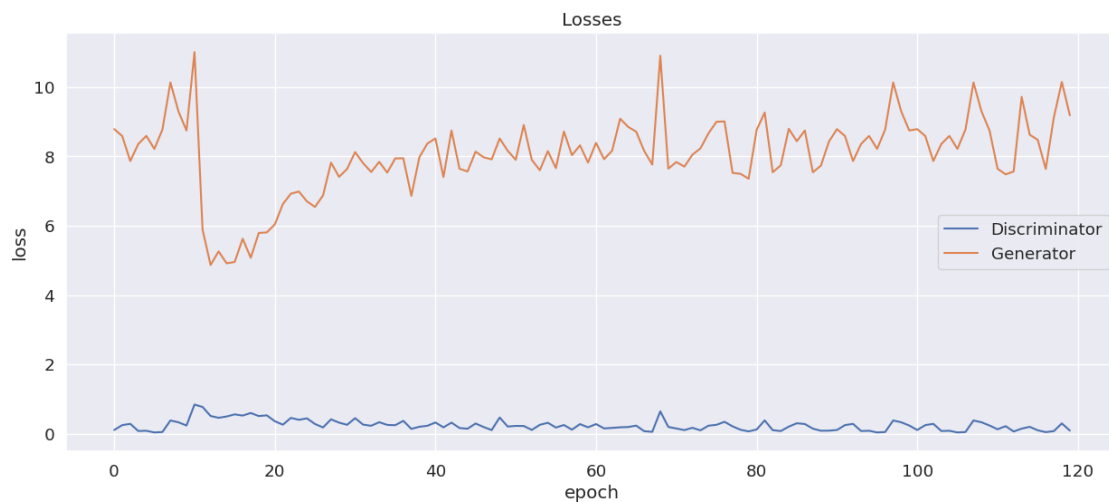
```
[ ]: gunD = trained_discriminator
     gunG = trained_generator
```

```
[16]: trained_discriminator = gunD
      trained_generator = gunG
```

```
[ ]: losses_g_110 = losses_g_110 + losses_g
      losses_d_110 = losses_d_110 + losses_d
      real_scores_110 = real_scores_110 + real_scores
      fake_scores_110 = fake_scores_110 + fake_scores
```

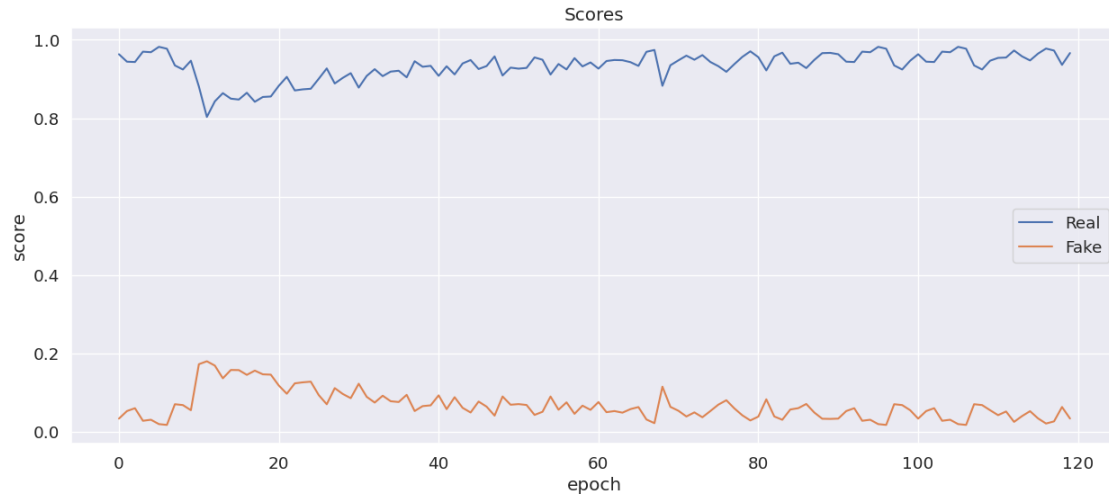
?

```
[ ]: plt.figure(figsize=(15, 6))
      plt.plot(losses_d_110, '-')
      plt.plot(losses_g_110, '-')
      plt.xlabel('epoch')
      plt.ylabel('loss')
      plt.legend(['Discriminator', 'Generator'])
      plt.title('Losses');
```



```
[ ]: plt.figure(figsize=(15, 6))

      plt.plot(real_scores_110, '-')
      plt.plot(fake_scores_110, '-')
      plt.xlabel('epoch')
      plt.ylabel('score')
      plt.legend(['Real', 'Fake'])
      plt.title('Scores');
```



0- , 1- . loss , 1-

0.3 3. (1)

```
[ ]: n_images = 4

fixed_latent = torch.randn(n_images, latent_size, 1, 1, device=device)
fake_images = gunG(fixed_latent)

[ ]: def show_images(generated):
    # TODO: show generated images
    fig, axes = plt.subplots(nrows=1, ncols=generated.shape[0], figsize=(10, 5))
    for i, ax in enumerate(axes):
        img = np.transpose(generated[i].detach().cpu().numpy(), (1, 2, 0))
        img = (img + 1) / 2 # [-1, 1] [0, 1]
        ax.imshow(img)
        ax.axis('off')
    plt.show()

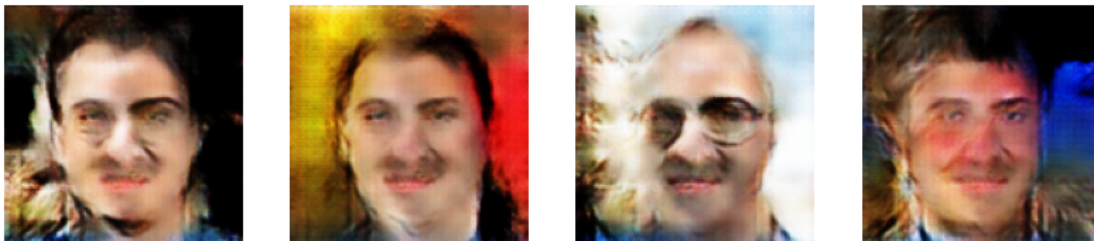
[ ]: print(fake_images.shape)
```

torch.Size([4, 3, 128, 128])

, 70, 110.

70 :

```
[ ]: show_images(fake_images.to('cpu'))
```



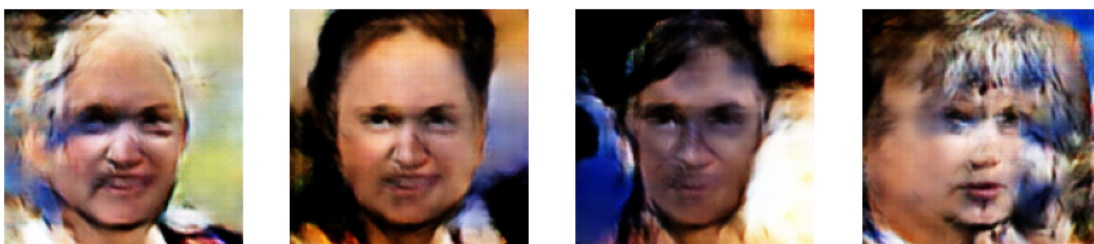
80 :

```
[ ]: show_images(fake_images.to('cpu'))
```



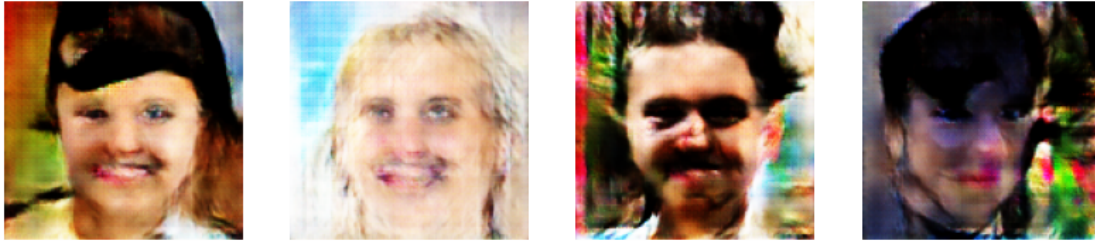
90 :

```
[ ]: show_images(fake_images.to('cpu'))
```



100 :

```
[ ]: show_images(fake_images.to('cpu'))
```



110 :

```
[ ]: show_images(fake_images.to('cpu'))
```



```
[ ]: torch.save({
    'discriminator': trained_discriminator.state_dict(),
    'generator': trained_generator.state_dict(),
}, 'models_110.pth')
```

```
[ ]: torch.save({
    'losses_g': losses_g + losses_g_110,
    'losses_d': losses_d + losses_d_110,
    'real_scores': real_scores + real_scores_110,
    'fake_scores': fake_scores + fake_scores_110,
}, 'data_110.pth')
```

?

0.4 4. Leave-one-out-1-NN classifier accuracy (6)

0.4.1 4.1. accuracy (4)

```

: *
0, - 1. * leave-one-out : 1NN Classifier
(sklearn.neighbors.KNeighborsClassifier(n_neighbors=1))
(accuracy) sklearn.model_selection.LeaveOneOut
```

```
[ ]: !zip -r '/content/trained_pictures_110.zip' '/content/trained_pictures/'
```

```
adding: content/trained_pictures/ (stored 0%)
adding: content/trained_pictures/generated_images_6.png (deflated 0%)
adding: content/trained_pictures/generated_images_5.png (deflated 0%)
adding: content/trained_pictures/generated_images_2.png (deflated 0%)
adding: content/trained_pictures/generated_images_7.png (deflated 0%)
adding: content/trained_pictures/generated_images_3.png (deflated 0%)
adding: content/trained_pictures/generated_images_10.png (deflated 0%)
adding: content/trained_pictures/generated_images_8.png (deflated 0%)
adding: content/trained_pictures/.ipynb_checkpoints/ (stored 0%)
adding: content/trained_pictures/generated_images_4.png (deflated 0%)
adding: content/trained_pictures/generated_images_1.png (deflated 0%)
adding: content/trained_pictures/generated_images_9.png (deflated 0%)
```

git-

accuracy

```
[17]: from sklearn.model_selection import LeaveOneOut
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.metrics import accuracy_score
```

```
[18]: n_images = 1600

      fixed_latent = torch.randn(n_images, latent_size, 1, 1, device=device)
      fake_images = trained_generator(fixed_latent)
```

```
[19]: reshaped_fake = fake_images.reshape(fake_images.shape[0], -1)

      real_images = [i[0] for i in tqdm(train_dl)][:int(n_images/batch_size)]
```

100%| | 99/99 [02:06<00:00, 1.27s/it]

```
[20]: reshaped_real = torch.stack(real_images).reshape(fake_images.shape[0], -1)
```

```
[21]: assert reshaped_real.shape == reshaped_fake.shape
```

```
[25]: fake_labels = np.zeros(n_images)
      real_labels = np.ones(n_images)
      labels = np.concatenate([fake_labels, real_labels])

      features = torch.concat([reshaped_fake.cpu(), reshaped_real.cpu()]).detach().
        ↪numpy()

      loo = LeaveOneOut()
      knn = KNeighborsClassifier(n_neighbors=1)
```

```
[26]: %%time
      # leave-one-out
      accuracies = []
      for train_index, test_index in loo.split(features):
          print("\tTRAIN:", train_index, "TEST:", test_index)
          X_train, X_test = features[train_index], features[test_index]
          y_train, y_test = labels[train_index], labels[test_index]
          knn.fit(X_train, y_train)
          y_pred = knn.predict(X_test)
          accuracies.append(accuracy_score(y_test, y_pred))
```

```
TRAIN: [ 1  2  3 ... 3197 3198 3199] TEST: [0]
TRAIN: [ 0  2  3 ... 3197 3198 3199] TEST: [1]
TRAIN: [ 0  1  3 ... 3197 3198 3199] TEST: [2]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [3]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [4]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [5]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [6]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [7]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [8]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [9]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [10]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [11]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [12]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [13]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [14]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [15]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [16]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [17]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [18]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [19]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [20]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [21]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [22]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [23]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [24]
TRAIN: [ 0  1  2 ... 3197 3198 3199] TEST: [25]
```


[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```

TRAIN: [ 0 1 2 ... 3197 3198 3199] TEST: [3194]
TRAIN: [ 0 1 2 ... 3197 3198 3199] TEST: [3195]
TRAIN: [ 0 1 2 ... 3197 3198 3199] TEST: [3196]
TRAIN: [ 0 1 2 ... 3196 3198 3199] TEST: [3197]
TRAIN: [ 0 1 2 ... 3196 3197 3199] TEST: [3198]
TRAIN: [ 0 1 2 ... 3196 3197 3198] TEST: [3199]

```

CPU times: user 41min 12s, sys: 24min 4s, total: 1h 5min 17s

Wall time: 52min 56s

```
[27]: mean_accuracy = np.mean(accuracies)
      print("Accuracy:", mean_accuracy)
```

Accuracy: 0.8690625

```

          ?      accuracy          ?
          .
          (      1181      1600      )
          .
          ,
          .
          ,
          .

```

0.4.2 4.2. (2)

```

, TSNE) ,
(

```

```
[ ]: from sklearn.manifold import TSNE
      import plotly.express as px
```

```
[ ]: tsne = TSNE(n_components=2)
      tsne_results = tsne.fit_transform(X)
```

```

, 70 ,
.

```

110 :

```
[ ]: fig = px.scatter(tsne_results, x=0, y=1, color=y.astype(str), labels={'0': 'tsne-2d-one', '1': 'tsne-2d-two'})
      fig.show()
```

100 :

```
[ ]: fig = px.scatter(tsne_results, x=0, y=1, color=y.astype(str), labels={'0': 'tsne-2d-one', '1': 'tsne-2d-two'})
      fig.show()
```

90 :

```
[ ]: fig = px.scatter(tsne_results, x=0, y=1, color=y.astype(str), labels={'0': 'tsne-2d-one', '1': 'tsne-2d-two'})
      fig.show()
```

80 :

```
[ ]: fig = px.scatter(tsne_results, x=0, y=1, color=y.astype(str), labels={'0': 'tsne-2d-one', '1': 'tsne-2d-two'})
fig.show()
```

70 :

```
[ ]: fig = px.scatter(tsne_results, x=0, y=1, color=y.astype(str), labels={'0': 'tsne-2d-one', '1': 'tsne-2d-two'})
fig.show()
```

:

GUN, accuracy

accuracy ,

:

90 80

, 70

110 100

BCELoss MSELoss,

git- , loss, ..

p.s.: , gun gan

!