УДК 004.8

Подделка лица персонажа аниме

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс использования модели глубокой сверточной генеративно-состязательной сети (DCGAN) для генерации изображения лица персонажа аниме. В работе использовалась библиотека Torch, и модель глубокой сверточной генеративно-состязательной сети для подделки лица персонажа. В наборе данных представлены лица персонажа аниме. В результате работы, модель DCGAN способна генерировать лицо персонажа аниме, а также была оценена модель для генерации изображения, в результате работы модели возможно получить новую комбинацию изображения сгенерированного персонажа.
Ключевые слова: GAN, DCGAN, Сверточные нейронные сети, Torch.

Anime character face fake

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes the process of using a deep convolutional generative adversarial network (DCGAN) model to generate an image of an anime character's face. The work used the Torch library, and a deep convolutional adversarial network model to fake the character's face. The dataset represents the faces of an anime character. As a result of the work, the DCGAN model is able to generate an anime face character, and the model for image generation was also evaluated, as a result of the work, the model can get a new image combination of the generated anime character.
Keywords: GAN, DCGAN, Convolutional Neural Network, Torch.

1 Введение

1.1. Актуальность исследования

Актуальность исследования заключается в преимуществе DCGAN (Глубокая сверточная генеративно-состязательная сеть) того, что она может автоматизировать процесс создания новых дизайнов персонажей. Вместо того, чтобы вручную рисовать и уточнять каждый аспект внешности персонажа аниме, DCGAN может генерировать широкий спектр вариантов на основе входных параметров и данных обучения. Это позволяет художникам быстро и

эффективно исследовать множество различных вариантов, не жертвуя качеством или творчеством.

1.2. Цель исследования

Целью работы создания и обучение модели для подделки изображения лица персонажа аниме.

1.3. Обзор исследований

Щ. Фанг предоставляют экспериментальные результаты, демонстрирующие эффективность предложенного метода в повышении точности распознавания CNN на нескольких стандартных наборах данных изображений. Кроме того, они обсуждают потенциальные применения предложенного метода в различных областях, таких как автономное вождение, робототехника и биометрия. В целом, исследование дает представление о преимуществах объединения CNN и DCGAN для повышения эффективности распознавания изображений [1].

Я. Щу, Ы. Чен, Й. Менг, изучает использования методов увеличения данных для повышения точности идентификации болезней листьев томата. Исследователи использовали глубокую сверточную генеративно-состязательную сеть (DCGAN) для создания дополнительных изображений больных листьев помидоров, которые затем были добавлены в набор обучающих данных. Результаты показали, что этот подход значительно улучшил производительность модели классификации, достигнув уровня точности более 97%. Исследование предполагает, что дополнение данных на основе DCGAN может быть эффективным инструментом для повышения надежности систем идентификации болезней растений [2].

П. Л. Суарез, А. Д. Саппа, Б. Х. Винтимилла, предложили новую технику раскрашивания инфракрасных изображений. Предлагаемый метод основан на триплетной архитектуре dcgan, которая представляет собой тип алгоритма глубокого обучения, состоящего из трех генераторов и трех дискриминаторов. Генераторы обучены генерировать цветные изображения из инфракрасных изображений в градациях серого, а дискриминаторы обучены различать сгенерированные изображения и реальные цветные изображения. Исследователи оценили свой метод на наборе данных инфракрасных изображений и обнаружили, что он превосходит существующие методы как с точки зрения количественных, так и качественных показателей. Предлагаемый метод имеет потенциальные применения в различных областях, таких как дистанционное зондирование, наблюдение и медицинская визуализация [3].

С. Лу и др. предлагает методология диагностики последовательных дуговых замыканий на постоянном токе в фотоэлектрических системах с использованием подхода глубокой сверточной нейронной сети, называемого DA-DCGAN. Предлагаемый подход сочетает в себе как дополнение данных, так и генеративные состязательные сети для эффективного обучения и улучшения диагностических возможностей сети. Исследование показывает, что подход DA-DCGAN превосходит другие существующие методы обнаружения дугового замыкания в фотоэлектрических системах с точностью 94,1%. Результаты показали, что предложенная методология может

обеспечить эффективное решение для диагностики неисправностей в фотоэлектрических системах, что приведет к повышению безопасности и эффективности [4].

Я. Ли и др, предлагают разработка AF-DCGAN, глубокой сверточной генеративно-состязательной сети (GAN), способной генерировать отпечатки пальцев для внутренних систем локализации. Предлагаемая модель использует амплитудные характеристики карт индикатора мощности принятого сигнала (RSSI) и тестируется на трех наборах данных. Результаты показывают, что AF-DCGAN превосходит другие методы построения отпечатков пальцев, что указывает на его потенциал для повышения точности локализации внутри помещений [5].

Щ. Ханг, рассматривает модель глубокой условной генеративно-состязательной сети (DCGAN) для создания аниме-аватаров с определенными атрибутами. Предлагаемый метод включает условную информацию в сети генератора и дискриминатора, позволяя модели управлять сгенерированными образцами в соответствии с заданными атрибутами. Авторы также вводят новую метрику для измерения разнообразия сгенерированных образцов, которая может помочь улучшить качество сгенерированных аниме-аватаров. Предлагаемый метод оценивается на большом наборе данных аниме-персонажей и дает многообещающие результаты с точки зрения качества и разнообразия выборки. Авторы также проводят несколько экспериментов для анализа эффективности предложенного метода и сравнения его с другими современными подходами. В исследовании делается вывод, что предложенная условная модель DCGAN может быть полезным инструментом для создания аниме-аватаров с желаемыми атрибутами и может иметь потенциальное применение в индустрии развлечений [6].

2. Рабочий процесс

2.1. Набор данных

Исходные данные Anime. В исходных данных представлены изображения размера 512x512 лица персонажа аниме, количество 25 664, размер 9.4 Гб (рис 1) [7].

Количество обучающий пакетов изображений (batch_size) 128 (рис 1).

Этап обработки загруженных изображений (листинг 2.1):

1.Уменьшается до 128x128 пикселей.

2. Обрезается по центру с размером на 128x128 пикселей.

3. Преобразуем в тензор.

4. Нормализуем -1,0 до 1,0.

Рисунок 1. Набор данных изображений размера 512x512

Листинг 2.1. Набор действия для обработки набор данных.

```
self.transform = [
        Trans.Compose([
                transforms.Resize(128),
                transforms.CenterCrop(128),
                transforms.ToTensor(),
                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
        ])
]
```

Строка 2 — 7. Список действия для обработки набор данных.

2.2. Модель

GAN представляет собой состав из двух нейронных сетей: генератора и дискриминатора. Генератор создает новые данные, используя шум в качестве входных данных, в то время как дискриминатор пытается отличить сгенерированные данные от реальных данных. Две сети тренируются вместе. Задача генератора — проверить данные который соответствует реальных данных, а задача дискриминатора — правильно определить данные настоящие или поддельные.

Генератор пытается создать настолько реалистичные данные, чтобы дискриминатор не смог распознать подделку, в то время как дискриминатор пытается правильно определить, являются ли данные реальными или поддельными. Поскольку две сети продолжают соревноваться друг против друга, генератор улучшает свою способность генерировать реалистичные данные, а дискриминатор улучшает свою способность точно различать настоящий от подделки [8].

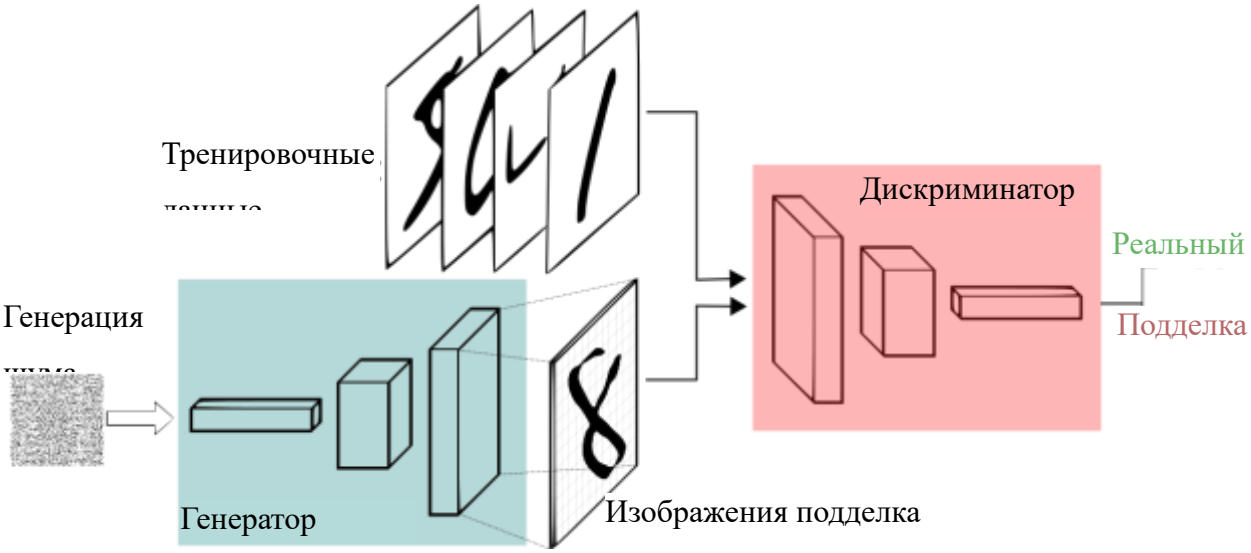

Рисунок 2. Принцип работы генеративно-состязательная сеть (GAN)

Модель дискриминант.

```
Discriminator(
  (conv1): Conv2d(3, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (relu1): LeakyReLU(negative_slope=0.2, inplace=True)
  (convs): Sequential(
    (0): Conv2d(128, 163, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(163, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Conv2d(163, 194, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(194, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.2, inplace=True)
    (6): Conv2d(194, 225, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(225, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Conv2d(225, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (conv_end): Conv2d(256, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (sig): Sigmoid()
 )
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1       [128, 128, 64, 64]           6,144
     LeakyReLU-2         [128, 128, 64, 64]               0
          Conv2d-3       [128, 163, 32, 32]         333,824
     BatchNorm2d-4       [128, 163, 32, 32]             326
     LeakyReLU-5         [128, 163, 32, 32]               0
          Conv2d-6       [128, 194, 16, 16]         505,952
     BatchNorm2d-7       [128, 194, 16, 16]             388
     LeakyReLU-8         [128, 194, 16, 16]               0
          Conv2d-9        [128, 225, 8, 8]          698,400
     BatchNorm2d-10       [128, 225, 8, 8]             450
     LeakyReLU-11         [128, 225, 8, 8]               0
          Conv2d-12       [128, 256, 4, 4]          921,600
     BatchNorm2d-13       [128, 256, 4, 4]             512
     LeakyReLU-14         [128, 256, 4, 4]               0
          Conv2d-15       [128, 1, 1, 1]             4,096
        Sigmoid-16        [128, 1, 1, 1]                 0
   Discriminator-17       [128, 1, 1, 1]                 0
```

```
================================================================
Total params: 2,471,692
Trainable params: 2,471,692
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 24.00
Forward/backward pass size (MB): 1712.69
Params size (MB): 9.43
Estimated Total Size (MB): 1746.12
```

Модель генератор.

```
Generator(
   (convt1): ConvTranspose2d(100, 128, kernel_size=(4, 4), stride=(1, 1), bias=False)
   (btnt1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (relut1): ReLU(inplace=True)
   (convs): Sequential(
     (0): ConvTranspose2d(128, 225, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
     (1): BatchNorm2d(225, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (2): ReLU(inplace=True)
     (3): ConvTranspose2d(225, 194, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
     (4): BatchNorm2d(194, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (5): ReLU(inplace=True)
     (6): ConvTranspose2d(194, 163, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
     (7): BatchNorm2d(163, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (8): ReLU(inplace=True)
     (9): ConvTranspose2d(163, 132, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
     (10): BatchNorm2d(132, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (11): ReLU(inplace=True)
   )
   (convt_end): ConvTranspose2d(132, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
   (tan): Tanh()
 )
----------------------------------------------------------------
----------------------------------------------------------------
       Layer (type)               Output Shape         Param #
================================================================
  ConvTranspose2d-1          [128, 128, 4, 4]         204,800
     BatchNorm2d-2          [128, 128, 4, 4]             256
            ReLU-3          [128, 128, 4, 4]               0
  ConvTranspose2d-4          [128, 225, 8, 8]         460,800
     BatchNorm2d-5          [128, 225, 8, 8]             450
            ReLU-6          [128, 225, 8, 8]               0
  ConvTranspose2d-7        [128, 194, 16, 16]         698,400
     BatchNorm2d-8        [128, 194, 16, 16]             388
            ReLU-9        [128, 194, 16, 16]               0
 ConvTranspose2d-10        [128, 163, 32, 32]         505,952
    BatchNorm2d-11        [128, 163, 32, 32]             326
           ReLU-12        [128, 163, 32, 32]               0
 ConvTranspose2d-13        [128, 132, 64, 64]         344,256
    BatchNorm2d-14        [128, 132, 64, 64]             264
           ReLU-15        [128, 132, 64, 64]               0
 ConvTranspose2d-16        [128, 3, 128, 128]           6,336
          Tanh-17        [128, 3, 128, 128]               0
       Generator-18        [128, 3, 128, 128]               0
================================================================
Total params: 2,222,228
Trainable params: 2,222,228
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 2410.69
```

```
Params size (MB): 8.48
Estimated Total Size (MB): 2419.21
----------------------------------------------------------------
```

Листинг 2.2. Параметры для обучения модели

```
1  self.criterion = nn.BCELoss().to(self.device)
2  self.real_label = 1.
3  self.fake_label = 0.
4  self.noise_z = 100
5  self.lr = 0.001
6  self.beta1 = 0.5
7  self.model = {
8          "dis": Discriminator().to(self.device),
9          "gen": Generator(noise_z=self.noise_z).to(self.device)
10 }
11 self.model["dis"].apply(self.weights_init)
12 self.model["gen"].apply(self.weights_init)
13 self.optimizer = {
14         "dis": optim.Adam(self.model["dis"].parameters(), lr=self.lr, betas=(self.beta1, 0.999)),
15         "gen": optim.Adam(self.model["gen"].parameters(), lr=self.lr, betas=(self.beta1, 0.999)),
16 }
```

Строка 1. Метрика BCELoss.

Строка 2 - 3. Метки, real_label — реальная изображения, fake_label - подделка.

Строка 4. Количество случайных шумов.

Строка 5 — 6. Скорость обучения и гиперпараметр для оптимизаторов Adam.

Строка 7 — 10. Модель дискриминатор и генератор.

Строка 11 — 12. Применяем изменение инициализируем весы из нормального распределения со средним значением = 0, стандартное отклонение = 0,02 (листинг 2.4).

Листинг 2.3. Инициализация весов.

```
1  def weights_init(self, m):
2          classname = m.__class__.__name__
3          if classname.find('Conv') != -1:
4                  nn.init.normal_(m.weight.data, 0.0, 0.02)
5          elif classname.find('BatchNorm') != -1:
6                  nn.init.normal_(m.weight.data, 1.0, 0.02)
7                  nn.init.constant_(m.bias.data, 0)
```

Строка 13 — 16, листинг 2.3. Оптимизация Adam.

2.3. Обучение

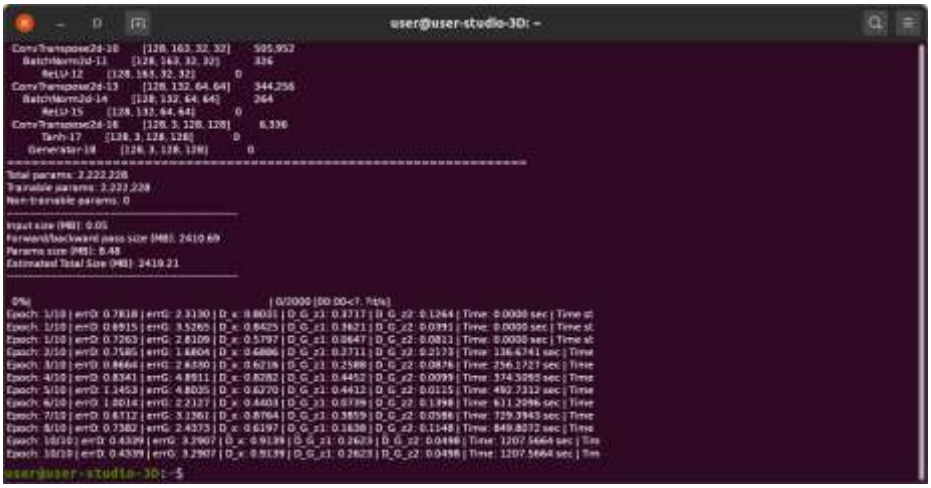

Рисунок 3. Обучение модели

Программа написано на языке Python использовалась библиотека Torch. Программа выполняет обучение модели (рис 2), задано эпохи 10, скорость обучение 0.001. Программа выполнялась с использования графический ускоритель GA106 (RTX 3060).

Для оценки модели использовали метрику BCELoss для обучение, для оценки генерация и дискриминанта модели.

Листинг 2.4. Функция для обучение модель

```
def step_train(self, sel: any, index :int) -> None:
        super().step_train(sel, index)
        image = sel["image"].to(self.device)

        real_cpu = image.to(self.device)
        b_size = real_cpu.size(0)
        label = torch.full((b_size,), self.real_label, dtype=torch.float, device=self.device)
        noise = torch.randn(b_size, self.noise_z, 1, 1, device=self.device)
        self.model["dis"].zero_grad()
        output = self.model["dis"](real_cpu).view(-1)
        errD_real = self.criterion(output, label)
        errD_real.backward()
        D_x = output.mean().item()

        fake = self.model["gen"](noise)
        label.fill_(self.fake_label)

        output = self.model["dis"](fake.detach()).view(-1)
        errD_fake = self.criterion(output, label)
        errD_fake.backward()
        D_G_z1 = output.mean().item()
        errD = errD_real + errD_fake
        self.optimizer["dis"].step()

        self.model["gen"].zero_grad()
        label.fill_(self.real_label)
        output = self.model["dis"](fake).view(-1)
        errG = self.criterion(output, label)
        errG.backward()
        D_G_z2 = output.mean().item()
```

```
31          self.optimizer["gen"].step()
32
33          self.metric.errD = errD.item()
34          self.metric.errG = errG.item()
35          self.metric.D_x = D_x
36          self.metric.D_G_z1 = D_G_z1
37          self.metric.D_G_z2 = D_G_z2
```

Строка 2, заглушка.

Строка 3, загрузка данных изображений размер пакета 128.

Строка 6 — размер пакета.

Строка 7 — заполняем метку для настоящий изображения.

Строка 8 — Генерируем шум количество 100.

Строка 9 — 13. Обучаем модель дискриминатора, изображения настоящая.

Строка 16 — заполняем метку для поддельные изображения.

Строка 18 — 23. Обучаем модель дискриминатора, изображения полученный модель генератора, метка для подделки.

Строка 26 - заполняем метку для настоящий изображения.

Строка 25 — 31. Обучаем модель генератора который стремиться «обмануть» дискриминатора.

Строка 33 — 37. сохраняем метрику в список.

2.4. Оценка модели

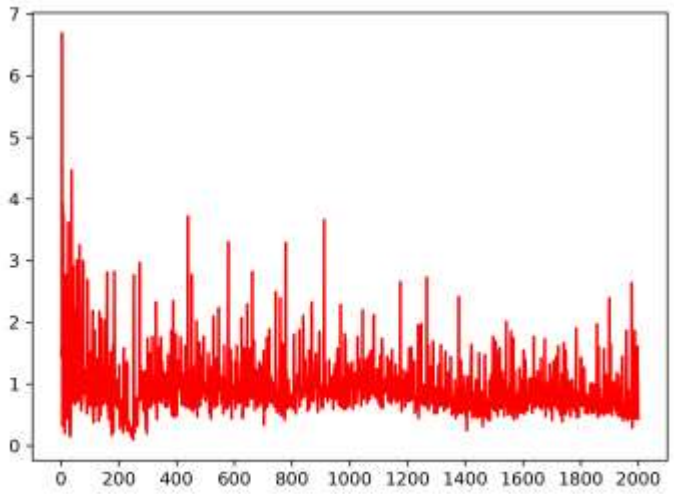

Рисунок 4. Функция потерь для дискриминатора (метрика BCELoss)

На графике представлено функция потерь по оси Y при 2000 шагов по оси X, метрика, показатель оценки способность модели дискриминатора оценивать изображения подлинность, устойчивость к подделке.

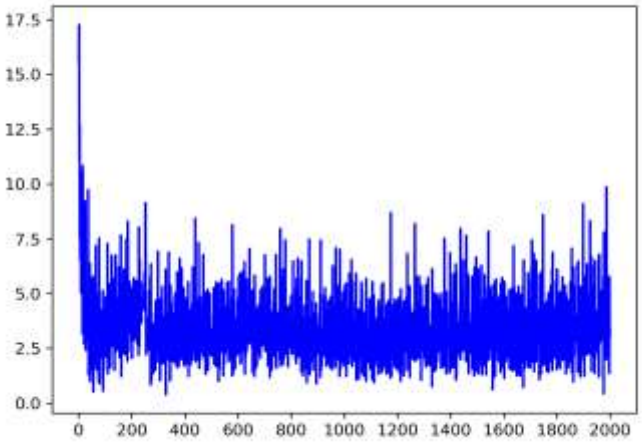

Рисунок 5. Функция потерь для генератора (метрика BCELoss)

Способность модели генератора обманывать модели дискриминатора.

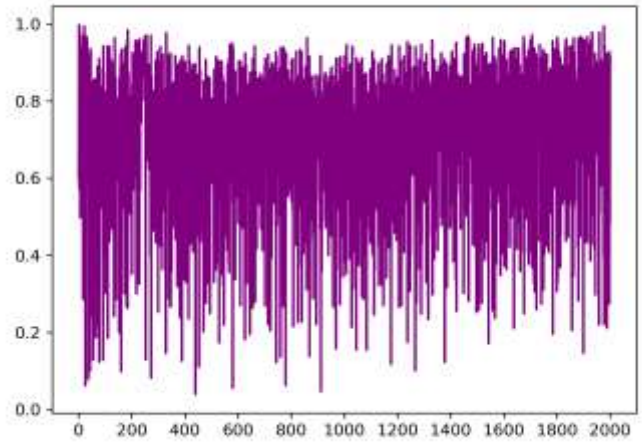

Рисунок 6. Функция потерь для дискриминатора D(x) (метрика BCELoss)

Оценка модели дискриминатора способность распознавать только подлинные (исходные) изображения.

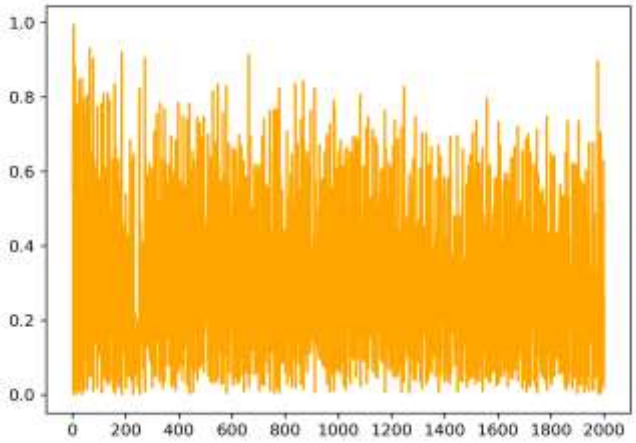

Рисунок 7. Функция потерь для дискриминатора $D(G(z_1))$ полученная генератором модели (метрика BCELoss).

Оценка модели дискриминатора полученный изображений от модели генератора, оценка для поделки изображений полученный изображений модели генератора. Приближенный значение к 0, модель стремиться распознать подделку.

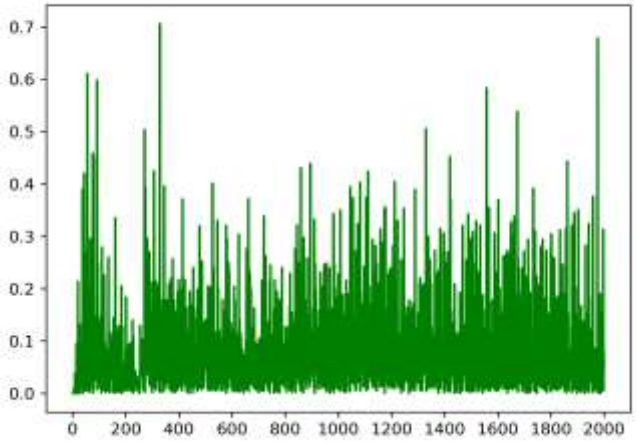

Рисунок 8. Функция потерь для дискриминатора $D(G(z_2))$ полученная генератором модели (метрика BCELoss).

Оценка модели дискриминатора полученный изображений от модели генератора, оценка для подлинных изображений от полученный изображений модели генератора.

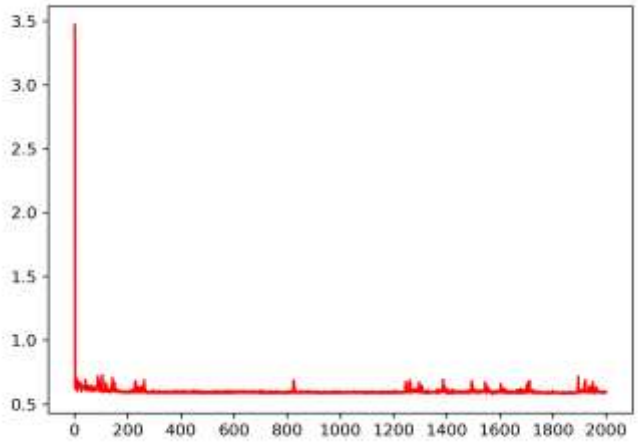

Рисунок 9. Время выполнения обучения в секундах.

На графике представлено время шага выполнения обучения в секундах.

Время всего цикла обучение составляет 1204 секунда (20 минута, 4 секунда).

2.5. Предсказания модели

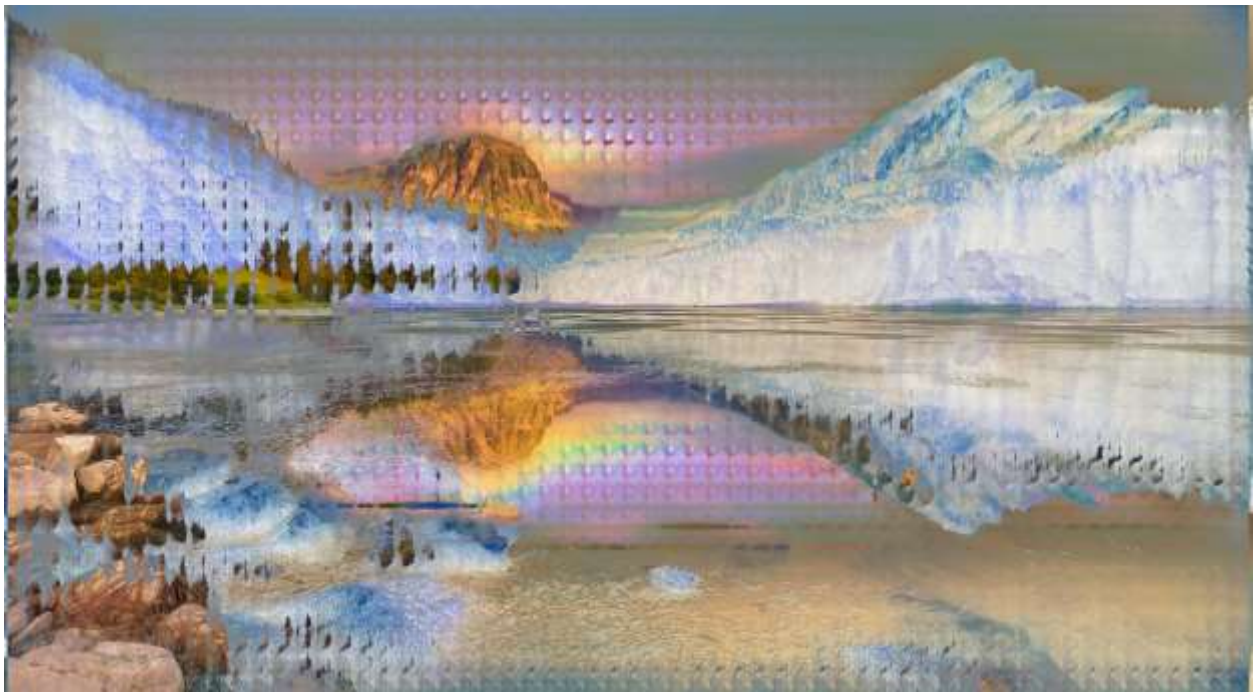

Рисунок 10. Пример обработки изображений с использовании модели GAN

Рисунок 11. Пример обработки изображений с использованием модели GAN

Рисунок 12. Слева обработка изображений с использованием модели GAN, справа - оригинал

Рисунок 13. Пример обработка изображений с использованием модели GAN.

Листинг 2.5. Пример использования модели GAN.

```
1   tr = Trans.Compose([
2           transforms.ToPILImage(),
3           Trans.ColorConvert("rgb", "ycrcb", "cv2"),
4           transforms.Grayscale(num_output_channels=1),
5           transforms.ToTensor(),
6           transforms.Normalize((0.5), (0.5))
7   ])
8
9   url_image = "./"
10
11  sel_image_org = cv2.imread(url_image + "/image.jpg")
12  sel_image = cv2.resize(sel_image_org, tuple(np.int32((np.array(sel_image_org.shape[1::-1])-128)/32+1)))
13  sel_image = tr(sel_image)[0]
14
15  data = torch.cat(torch.randn(100, sel_image.size(1), sel_image.size(2)) * sel_image[0])
16  ind_data = torch.randperm(len(data))
17  data = data[ind_data]
18  data = data.unsqueeze(0)
19  img = gen_model(data)
20  img_org = cv2.resize(sel_image_org, img.shape[2:][1::-1])
21  img_org = cv2.cvtColor(img_org, cv2.COLOR_BGR2RGB)
22  img_org = torch.tensor(np.moveaxis(img_org / 255.0 / 0.5 - 1.0, -1, 0), dtype=torch.float32).unsqueeze(0)
23  d = 0
24  #save_image(img.lerp(img_org, 0.1) * 0.5 + 0.5, url_image + "/Out/" + 'img_{0}.jpg'.format(d))
25  #save_image((img_org * img) * 0.5 + 0.5, url_image + "/Out/" + 'img_{0}.jpg'.format(d))
26  #img_org_mask = torch.sign((img_org + 1.0) * 255.)
27  img_org_mask = torch.clamp(img_org * 1.0 + 0.1, -1, 1)
28  save_image(img.lerp(img_org, img_org_mask) * 0.5 + 0.5, url_image + "/" + 'out_{0}.jpg'.format(d))
```

Строка 1 — 7, 13. обработка и извлечение канала Y в цвет YCrCb изображений.

Строка 11 чтение изображений

Строка 12 изменение размер изображений по расчету модели, а на выходе — исходный размер изображений.

Строка 15 генерация шума и умножения исходный изображений.

Строка 16 — 17 перемешивания случайная сортировка набора.

Строка 19 предсказания модели.

Строка 20 изменение размера исходный изображения

Строка 22 преобразования исходный изображений в тензор и нормировка.

Строка 23 — 28 смешивания исходный изображений и обработанный изображений и вывод в файл.

Рисунок 14. Пример произвольно сгенерированный изображений модель GAN

Листинг 2.6. Пример использования модели GAN для привольно генерация изображения.

```
1  for i in range(100):
2          data = torch.clamp(torch.randn(100, 1, 1), -0.8, 1.0).unsqueeze(0)
3          img = gen_model(data)
4          d = "i_" + str(i)
5          save_image(img, url_image + "/Out/o/" + 'img_{0}.jpg'.format(d))
```

Строка 2 заданно произвольный ограниченный диапазон значение -0.8 до 1.0.

Строка 3 генерирует изображения.

Строка 4 — 5 выводит в файл.

3 Выводы

В данной статье была использована модель глубокой сверточной генеративно-состязательной сети (DCGAN) для генерация лица персонажа аниме, в результате получили оценку работы модели, а так же использовали возможность модели генерировать изображений и смешивать изображения поверх оригинала.

Библиографический список

1. Fang W. et al. A Method for Improving CNN-Based Image Recognition Using DCGAN //Computers, Materials & Continua. – 2018. – Т. 57. – №. 1.
2. Wu Q., Chen Y., Meng J. DCGAN-based data augmentation for tomato leaf disease identification //IEEE Access. – 2020. – Т. 8. – С. 98716-98728.
3. Suárez P. L., Sappa A. D., Vintimilla B. X. Infrared image colorization based on a triplet dcgan architecture //Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. – 2017. – С. 18-23.
4. Lu S. et al. DA-DCGAN: An effective methodology for DC series arc fault diagnosis in photovoltaic systems //IEEE Access. – 2019. – Т. 7. – С. 45831-45840.
5. Li Q. et al. AF-DCGAN: Amplitude feature deep convolutional GAN for fingerprint construction in indoor localization systems //IEEE Transactions on Emerging Topics in Computational Intelligence. – 2019. – Т. 5. – №. 3. – С. 468-480.
6. Hang W. Conditional DCGAN For Anime Avatar Generation.
7. Anime GAN Lite | Kaggle // Kaggle URL: https://www.kaggle.com/datasets/prasoonkottarathil/gananime-lite (дата обращения: 2023-05-16).
8. Generative adversarial network - Wikipedia // Wikipedia URL: https://en.wikipedia.org/wiki/Generative_adversarial_network (дата обращения: 2023-05-16)] [Goodfellow I. J. et al. Generative adversarial networks. arXiv 2014 //arXiv preprint arXiv:1406.2661. – 2014.

4. Приложения

Листинг 4.1. Исходный код программы

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import cv2
from torchsummary import summary
from torchvision import transforms
import torchvision.transforms.functional as F
from typing import Optional, Callable, TypeVar, Type, Union
from PIL import Image
from Lib.AppMain import *
import Lib.Transforms as Trans
import torch.nn.functional as nnf
import matplotlib.pyplot as plt
import os
os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "max_split_size_mb:256"

#https://www.kaggle.com/code/rezasemyari/anime-gan-by-pytorch
class Discriminator(nn.Module):
	def __init__(self, size = 128, deep = 5):
		super(Discriminator, self).__init__()
		self.size = size
		self.deep = deep
		self.conv1 = nn.Conv2d(3, self.size, kernel_size=4, stride=2, padding=1, bias=False)
		self.relu1 = nn.LeakyReLU(0.2, inplace=True)

		prev_i = self.size
		self.convs = deque()
		j = 0
		for i in np.linspace(4, self.size, self.deep):
			if j == 0:
				j += 1
				continue
			i = self.size + int(i)
			self.convs.append(nn.Conv2d(prev_i, i, kernel_size=4, stride=2, padding=1, bias=False))
			self.convs.append(nn.BatchNorm2d(i))
			self.convs.append(nn.LeakyReLU(0.2, inplace=True))
			prev_i = i
			j += 1
		self.convs = nn.Sequential(*self.convs)
		self.conv_end = nn.Conv2d(prev_i, 1, kernel_size=4, stride=1, padding=0, bias=False)
		self.sig = nn.Sigmoid()

	def forward(self, Input):
		output = self.conv1(Input)
		output = self.relu1(output)

		output = self.convs(output)

		output = self.conv_end(output)
		output = self.sig(output)
		return output

class Generator(nn.Module):
	def __init__(self, noise_z = 100, size = 128, deep = 5):
		super(Generator, self).__init__()
		self.noise_z = noise_z
		self.size = size
		self.deep = deep

		self.convt1 = nn.ConvTranspose2d(self.noise_z, self.size, kernel_size=4, stride=1, padding=0, bias=False)
		self.btnt1 = nn.BatchNorm2d(self.size)
		self.relut1 = nn.ReLU(True)

		prev_i = self.size
		self.convs = deque()
		j = 0
		for i in np.linspace(self.size, 4, self.deep):
			if j == 0:
```

```
                                        j += 1
                                        continue
                                i = self.size + int(i)
                                self.convs.append(nn.ConvTranspose2d(prev_i, i, kernel_size=4, stride=2, padding=1, bias=False))
                                self.convs.append(nn.BatchNorm2d(i))
                                self.convs.append(nn.ReLU(True))
                                prev_i = i
                                j += 1
                        self.convs = nn.Sequential(*self.convs)

                        self.convt_end = nn.ConvTranspose2d(prev_i, 3, kernel_size=4, stride=2, padding=1, bias=False)
                        self.tan=nn.Tanh()
                def forward(self, Input):
                        output = self.convt1(Input)
                        output = self.btnt1(output)
                        output = self.relut1(output)

                        output = self.convs(output)

                        output = self.convt_end(output)
                        output = self.tan(output)
                        return output

class App(AppMain):
                def weights_init(self, m):
                        classname = m.__class__.__name__
                        if classname.find('Conv') != -1:
                                nn.init.normal_(m.weight.data, 0.0, 0.02)
                        elif classname.find('BatchNorm') != -1:
                                nn.init.normal_(m.weight.data, 1.0, 0.02)
                                nn.init.constant_(m.bias.data, 0)
                def file_begin(self, fname :str, mode :str) -> any:
                        return open(self.file_name(fname), mode)
                def file_name(self, fname :str) -> str:
                        return fname
                def main(self):
                        self.dir_prefix = "/mnt/ram/Ram/Data"
                        self.dirs = {
                                "dataset": "../Anime",
                                "dataset_test": "Test",
                                "fig": "Fig",
                        }
                        self.profile_name = "default"
                        self.datasets = None
                        self.model = {}
                        self.optimizer = {}
                        self.auto_save_exit = False
                        self.init_dirs()

                        self.disp_metric = ["errD", "errG", "D_x", "D_G_z1", "D_G_z2"]
                        self.metric.errD  = None
                        self.metric.errG  = None
                        self.metric.D_x  = None
                        self.metric.D_G_z1  = None
                        self.metric.D_G_z2  = None
                        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
                        self.criterion = nn.BCELoss().to(self.device)

                        self.real_label = 1.
                        self.fake_label = 0.
                        self.noise_z = 100
                        self.lr = 0.001
                        self.beta1 = 0.5
                        self.model = {
                                "dis": Discriminator().to(self.device),
                                "gen": Generator(noise_z=self.noise_z).to(self.device)
                        }
                        self.model["dis"].apply(self.weights_init)
                        self.model["gen"].apply(self.weights_init)
                        if (self.device.type == 'cuda'):
                                self.model["dis"] = nn.DataParallel(self.model["dis"])
                                self.model["gen"] = nn.DataParallel(self.model["gen"])

                        self.optimizer = {
                                "dis": optim.Adam(self.model["dis"].parameters(), lr=self.lr, betas=(self.beta1, 0.999)),
                                "gen": optim.Adam(self.model["gen"].parameters(), lr=self.lr, betas=(self.beta1, 0.999)),
                        }
                        self.transform = [
```

```
                                        Trans.Compose([
                                                transforms.Resize(128),
                                                transforms.CenterCrop(128),
                                                transforms.ToTensor(),
                                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                                        ])
                        ]
                        self.add_datasets(transform=self.transform, count_files=0, batch_size=128, count=1, header=["image"])
                        self.init_datasets()
                        self.cache_data = True
        def start_train(self) -> None:
                        self.model["dis"].train()
                        self.model["gen"].train()
        def step_train(self, sel: any, index :int) -> None:
                        super().step_train(sel, index)
                        image = sel["image"].to(self.device)

                        real_cpu = image.to(self.device)
                        b_size = real_cpu.size(0)
                        label = torch.full((b_size,), self.real_label, dtype=torch.float, device=self.device)
                        noise = torch.randn(b_size, self.noise_z, 1, 1, device=self.device)
                        self.model["dis"].zero_grad()
                        output = self.model["dis"](real_cpu).view(-1)
                        errD_real = self.criterion(output, label)
                        errD_real.backward()
                        D_x = output.mean().item()

                        fake = self.model["gen"](noise)
                        label.fill_(self.fake_label)

                        output = self.model["dis"](fake.detach()).view(-1)
                        errD_fake = self.criterion(output, label)
                        errD_fake.backward()
                        D_G_z1 = output.mean().item()
                        errD = errD_real + errD_fake
                        self.optimizer["dis"].step()

                        self.model["gen"].zero_grad()
                        label.fill_(self.real_label)
                        output = self.model["dis"](fake).view(-1)
                        errG = self.criterion(output, label)
                        errG.backward()
                        D_G_z2 = output.mean().item()
                        self.optimizer["gen"].step()

                        self.metric.errD = errD.item()
                        self.metric.errG = errG.item()
                        self.metric.D_x = D_x
                        self.metric.D_G_z1 = D_G_z1
                        self.metric.D_G_z2 = D_G_z2

                        sel["image"].to("cpu")
                        torch.cuda.empty_cache()
        def load_image(self, path :str) -> Image.Image:
                        with open(path, "rb") as f:
                                        img = Image.open(f)
                                        return img.convert("RGB")
        def save_image(self, path :str, img :Union[np.ndarray, Image.Image]) -> None:
                        pic = None
                        with open(path, "wb") as f:
                                        if type(img) != Image.Image:
                                                        pic = Image.fromarray(img, "RGB")
                                        else:
                                                        pic = img
                                        pic.save(f)
app = App()
app.main()

summary(app.model["dis"], input_size=(3, 128, 128), batch_size=128)
summary(app.model["gen"], input_size=(100, 1, 1), batch_size=128)

app.fit(10)
app.save_model("gan_anime")

#----------------------------
app.load_model("gan_anime")

sel = app.load_image("/home/ram/Ram/wallhaven/wallhaven-zxky9v.png")
```

```
229
230 gen_model = app.model["gen"].module.to("cpu").eval()
231 d = torch.randn(1, 100, 10, 10)
232 img_d = gen_model(d) * 0.5 + 0.5
233
234 img_d = F.to_pil_image(img_d[0])
235 app.save_image("/home/ram/Ram/out_0.jpg", img_d)
236
237 grid = Trans.Grid((128, 128))
238 sp_sel = grid(sel)
239
240 sel_img = np.array(sel)
241 sel_n = cv2.resize(sel_img, tuple(np.int32((np.array(sel_img.shape[1::-1])-128)/16+1)))
242 app.save_image("/home/ram/Ram/out_0_sel_n.jpg", sel_n)
243
244 #----------------------------
245
246 tr = Trans.Compose([
247             transforms.ToPILImage(),
248             Trans.ColorConvert("rgb", "ycrcb", "cv2"),
249             transforms.Grayscale(num_output_channels=1),
250             transforms.ToTensor(),
251             transforms.Normalize((0.5), (0.5))
252 ])
253
254 url_image = "/home/ram/Ram"
255
256 #sel_image_org = cv2.imread(url_image + "/wallhaven/wallhaven-0jy9qm.jpg")
257 #sel_image_org = cv2.imread(url_image + "/wallhaven-0jy9qm.jpg")
258 #sel_image_org = cv2.imread(url_image + "/wallhaven/wallhaven-0jvw5w.jpg")
259 #sel_image_org = cv2.imread(url_image + "/wallhaven/wallhaven-0pydzm.jpg")
260 #sel_image_org = cv2.imread(url_image + "/lena.png")
261 #sel_image_org = cv2.imread(url_image + "/wallhaven/wallhaven-ymrz37.jpg")
262 sel_image_org = cv2.imread(url_image + "/wallhaven/wallhaven-j8edwm.jpg")
263
264 sel_image = cv2.resize(sel_image_org, tuple(np.int32((np.array(sel_image_org.shape[1::-1])-128)/32+1)))
265
266 sel_image = tr(sel_image)[0]
267 #sel_image = cv2.resize(sel_image_org, tuple(np.int32((np.array(sel_image_org.shape[1::-1])-128)/32+1)))
268 #sel_image = torch.tensor(np.moveaxis(sel_image / 255.0 / 0.5 - 1.0, -1, 0), dtype=torch.float32)
269
270 img_cat = []
271 n = 0
272 #n = 30
273 #n max 33
274 #for i in range(n):
275 #             img_cat.append(sel_image)
276 #img_cat.append(torch.full((100-n*3, sel_image.size(1), sel_image.size(2)), 0.1, dtype=torch.float))
277 #img_cat.append(torch.clamp(torch.randn(100-n*3, sel_image.size(1), sel_image.size(2)), -1, 1))
278 img_cat.append(torch.randn(100-n*3, sel_image.size(1), sel_image.size(2)) * sel_image[0])
279
280 data = torch.cat(img_cat)
281 ind_data = torch.randperm(len(data))
282 data = data[ind_data]
283
284 data = data.unsqueeze(0)
285
286 img = gen_model(data)
287
288 img_org = cv2.resize(sel_image_org, img.shape[2:][1::-1])
289 img_org = cv2.cvtColor(img_org, cv2.COLOR_BGR2RGB)
290 img_org = torch.tensor(np.moveaxis(img_org / 255.0 / 0.5 - 1.0, -1, 0), dtype=torch.float32).unsqueeze(0)
291
292 d = 7
293 #save_image(img.lerp(img_org, 0.1) * 0.5 + 0.5, url_image + "/Out/" + 'img_{0}.jpg'.format(d))
294 #save_image((img_org * img) * 0.5 + 0.5, url_image + "/Out/" + 'img_{0}.jpg'.format(d))
295
296 #img_org_mask = torch.sign((img_org + 1.0) * 255.)
297 img_org_mask = torch.clamp(img_org * 1.0 + 0.1, -1, 1)
298 save_image(img.lerp(img_org, img_org_mask) * 0.5 + 0.5, url_image + "/Out/" + 'img_{0}.jpg'.format(d))
299 #save_image(torch.max(img_org, img) * 0.5 + 0.5, url_image + "/Out/" + 'img_{0}.jpg'.format(d))
300
301 #----------------------------
302
303 data = torch.clamp(torch.randn(100, 1, 1), -1, 1).unsqueeze(0)
304 img = gen_model(data)
305
306 d = 11
```

```
307 save_image(img, url_image + "/Out/" + 'img_{0}.jpg'.format(d))
308
309 for i in range(100):
310             data = torch.clamp(torch.randn(100, 1, 1), -0.8, 1.0).unsqueeze(0)
311             img = gen_model(data)
312             d = "i_" + str(i)
313             save_image(img, url_image + "/Out/o/" + 'img_{0}.jpg'.format(d))
314 #----------------------------
315
316 df = pd.DataFrame(app.metric._data)
317 df_m = df.groupby("Epoch").mean()
318 x_step = 200
319
320 fig, ax = plt.subplots()
321 ax.plot(df["Step"], df["Time"], label ='Time', color="red")
322 ax.set_xticks(np.arange(0, len(df)+1, x_step))
323 fig.savefig(app.get_path("fig", "{0}.png".format("Time")), dpi = 300)
324
325 fig, ax = plt.subplots()
326 ax.plot(df["Step"], df["errD"], label ='Error Discriminator', color="red")
327 ax.set_xticks(np.arange(0, len(df)+1, x_step))
328 fig.savefig(app.get_path("fig", "{0}.png".format("errD")), dpi = 300)
329
330 fig, ax = plt.subplots()
331 ax.plot(df["Step"], df["errG"], label ='Error Generator', color="blue")
332 ax.set_xticks(np.arange(0, len(df)+1, x_step))
333 fig.savefig(app.get_path("fig", "{0}.png".format("errG")), dpi = 300)
334
335
336 fig, ax = plt.subplots()
337 ax.plot(df["Step"], df["D_x"], label ='D(x)', color="purple")
338 ax.set_xticks(np.arange(0, len(df)+1, x_step))
339 fig.savefig(app.get_path("fig", "{0}.png".format("D_x")), dpi = 300)
340
341 fig, ax = plt.subplots()
342 ax.plot(df["Step"], df["D_G_z1"], label ='D(G(z1))', color="orange")
343 ax.set_xticks(np.arange(0, len(df)+1, x_step))
344 fig.savefig(app.get_path("fig", "{0}.png".format("D_G_z1")), dpi = 300)
345
346 fig, ax = plt.subplots()
347 ax.plot(df["Step"], df["D_G_z2"], label ='D(G(z2))', color="green")
348 ax.set_xticks(np.arange(0, len(df)+1, x_step))
349 fig.savefig(app.get_path("fig", "{0}.png".format("D_G_z2")), dpi = 300)
```