

Генеративно-состязательные сети



Выполнили студенты
Павловский Никита группа 0392
Серебрякова Софья группа 0373

Цели работы



Изучить свойства, особенности и
принцип работы генеративно-
состязательных сетей

Создать генеративно-состязательную
сеть

Обучить созданную сеть

Сделать анализ результата
работы

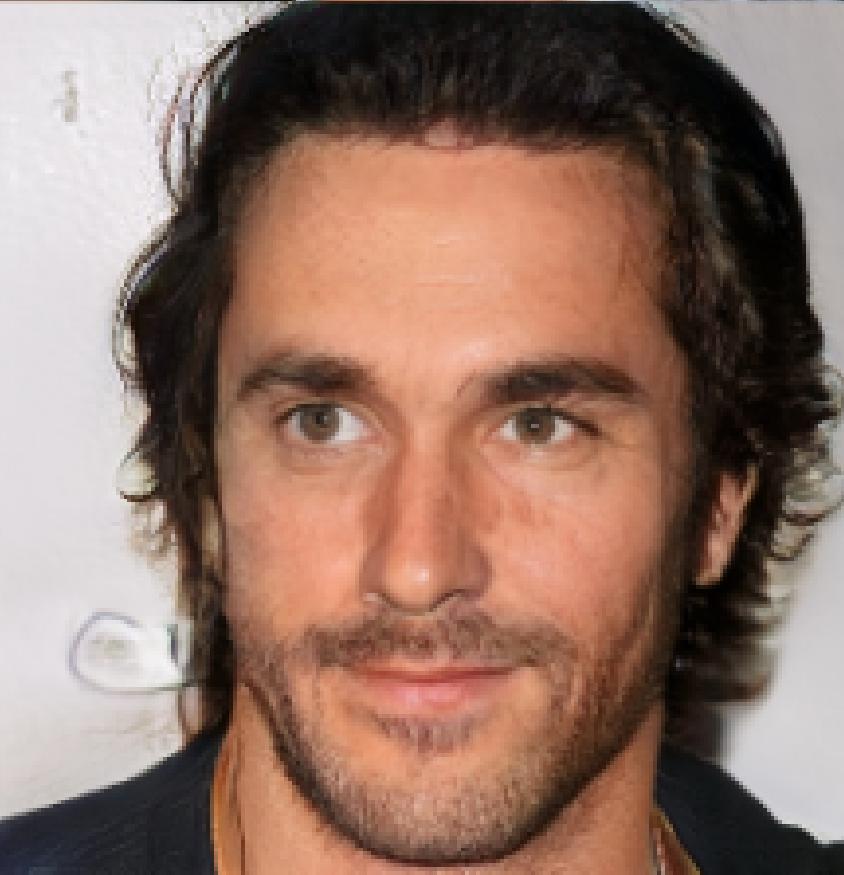
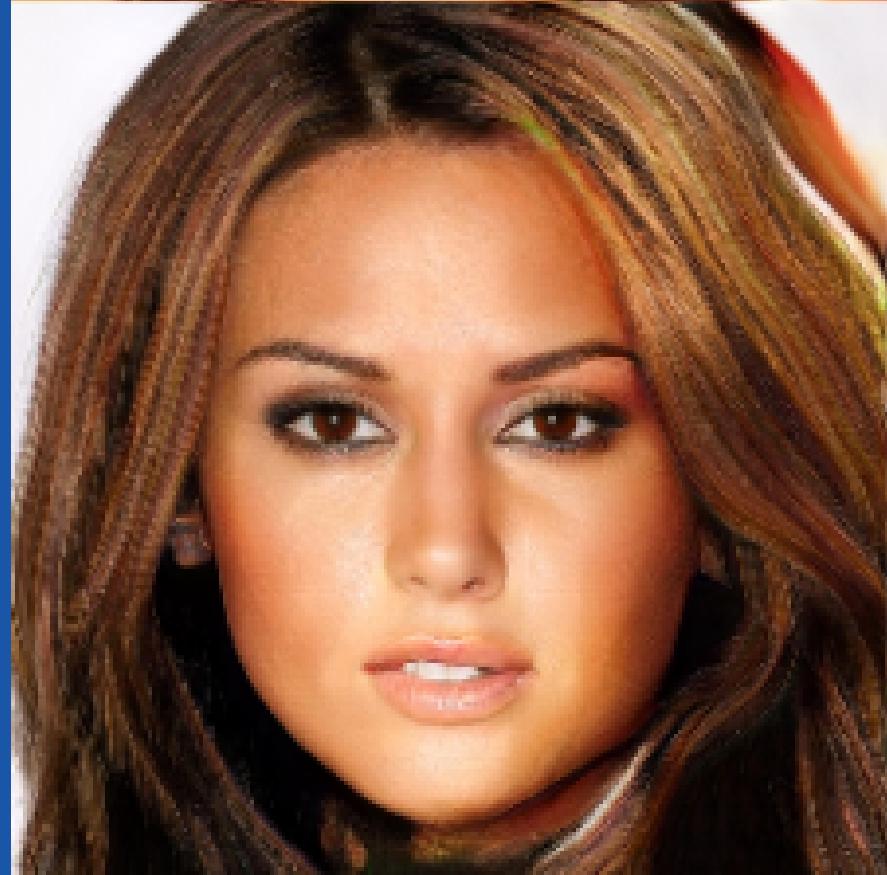
Что такое генеративно-состязательная сеть?

GAN являются модельной архитектурой для обучения генеративной модели, и в этой архитектуре чаще всего используются модели глубокого обучения.

Архитектура модели GAN включает две под-модели: модель генератора для создания новых примеров и модель дискриминатора для классификации того, являются ли сгенерированные примеры реальными, из области исходных данных или поддельными, сгенерированными генеративной моделью.



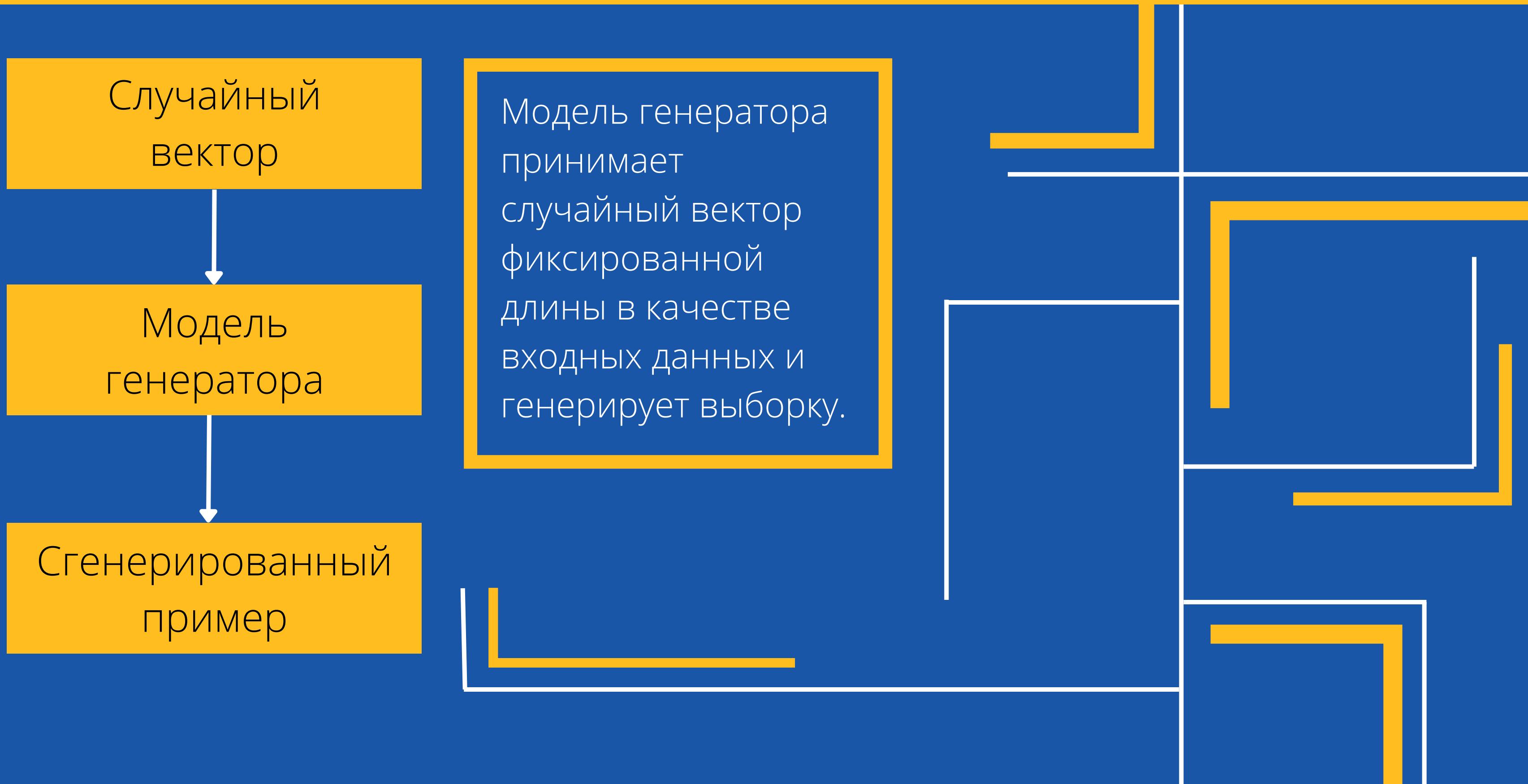
Для чего нужны GAN?



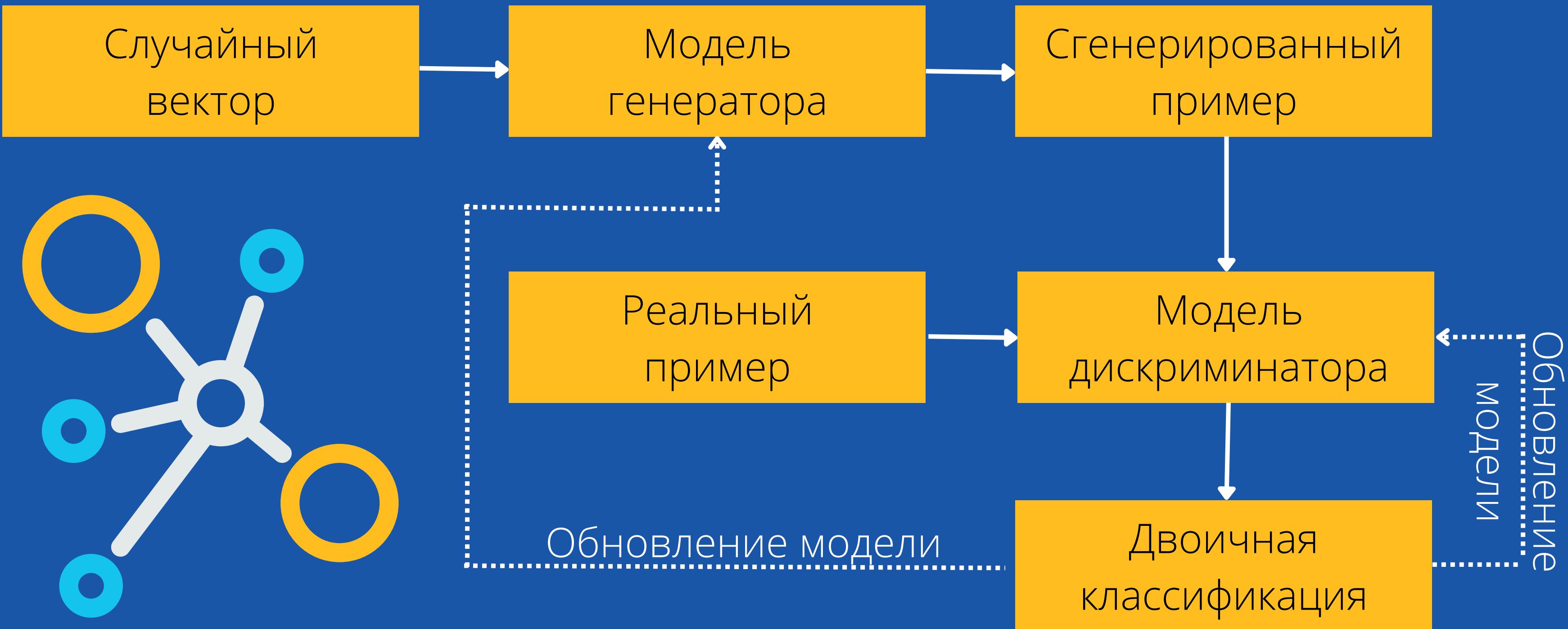
Модель дискриминатора



Модель генератора



Совместная работа двух моделей



Обучение дискриминатора



Бинарная кросс-энтропия:

$$\text{loss_diss_real} = -\text{tr} \cdot \log(\text{real}) - (1 - \text{tr}) \cdot \log(1 - \text{real})$$

$$\text{loss_diss_fake} = -\text{tf} \cdot \log(\text{fake}) - (1 - \text{tf}) \cdot \log(1 - \text{fake})$$

$$\text{tr} = 1, \text{tf} = 0$$

$$\text{loss_dis} = -\log(\text{real}) - \log(1 - \text{fake})$$

Обучение генератора



Бинарная кросс-энтропия:
 $\text{loss_gen} = -\text{tf} * \log(\text{fake}) - (1 - \text{tf}) * \log(1 - \text{fake})$
 $\text{tf} = 1$

$$\text{loss_gen} = -\log(\text{fake})$$

Процесс обучения GAN

Шаг 1. Выбираем несколько реальных изображений из тренировочного набора

Шаг 2. Генерируем несколько фейковых изображений.

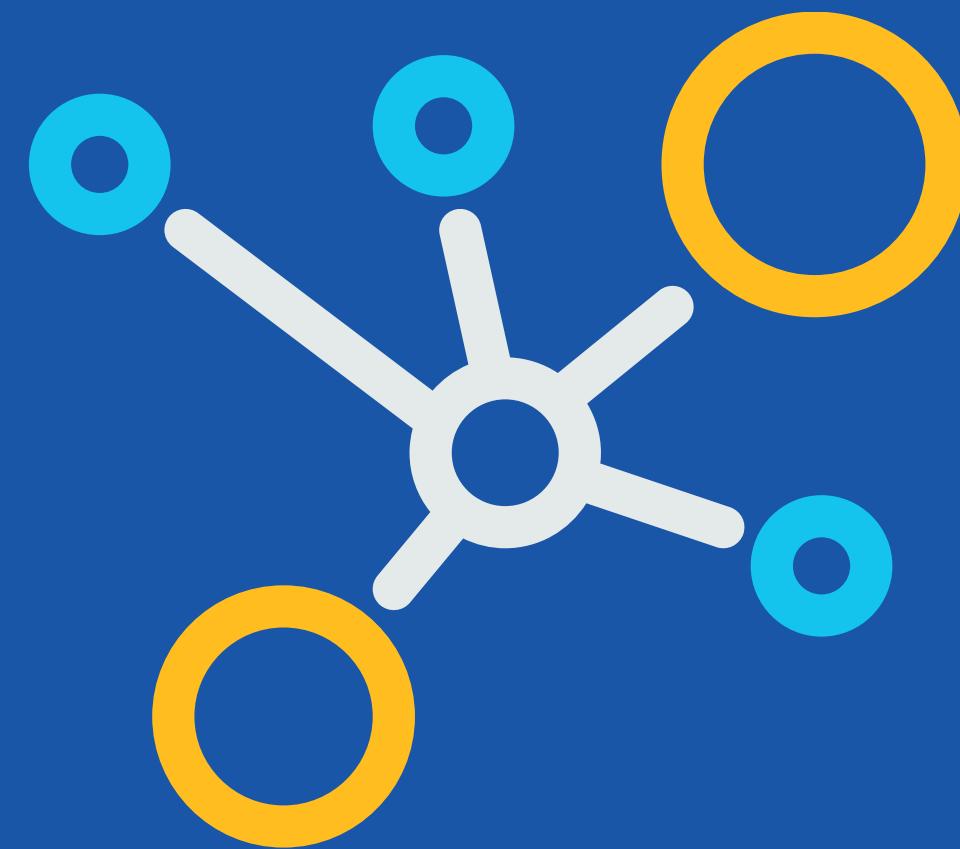
Шаг 3. Обучаем дискриминатор используя как реальные, так и фейковые изображения. При этом будут обновляться только веса дискриминатора, поскольку мы пометим все реальные изображения как 1, а фейковые как 0.

Шаг 4. Создаем еще несколько фейковых изображений.

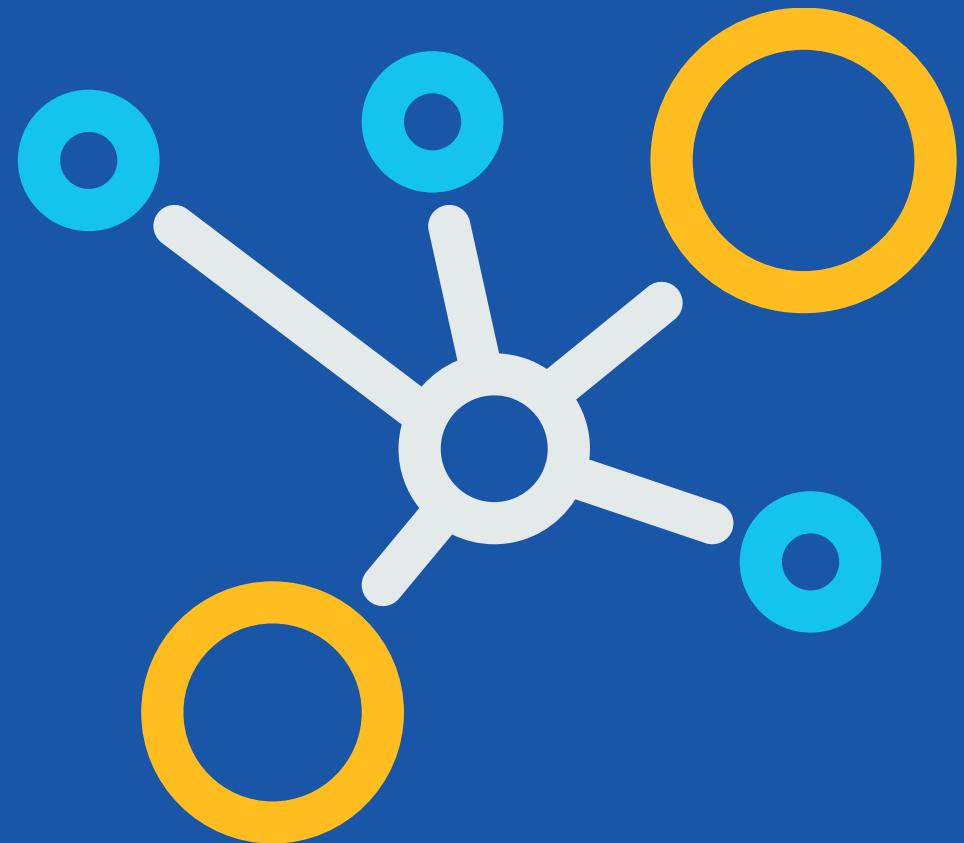
Шаг 5. Обучаем полную модель GAN используя только фейковые изображения. При этом будут обновляться только веса генератора, а всем фейковым изображениям будет назначена метка 1.



Реализация GAN



Инструменты



Python 3

Набор данных MNIST

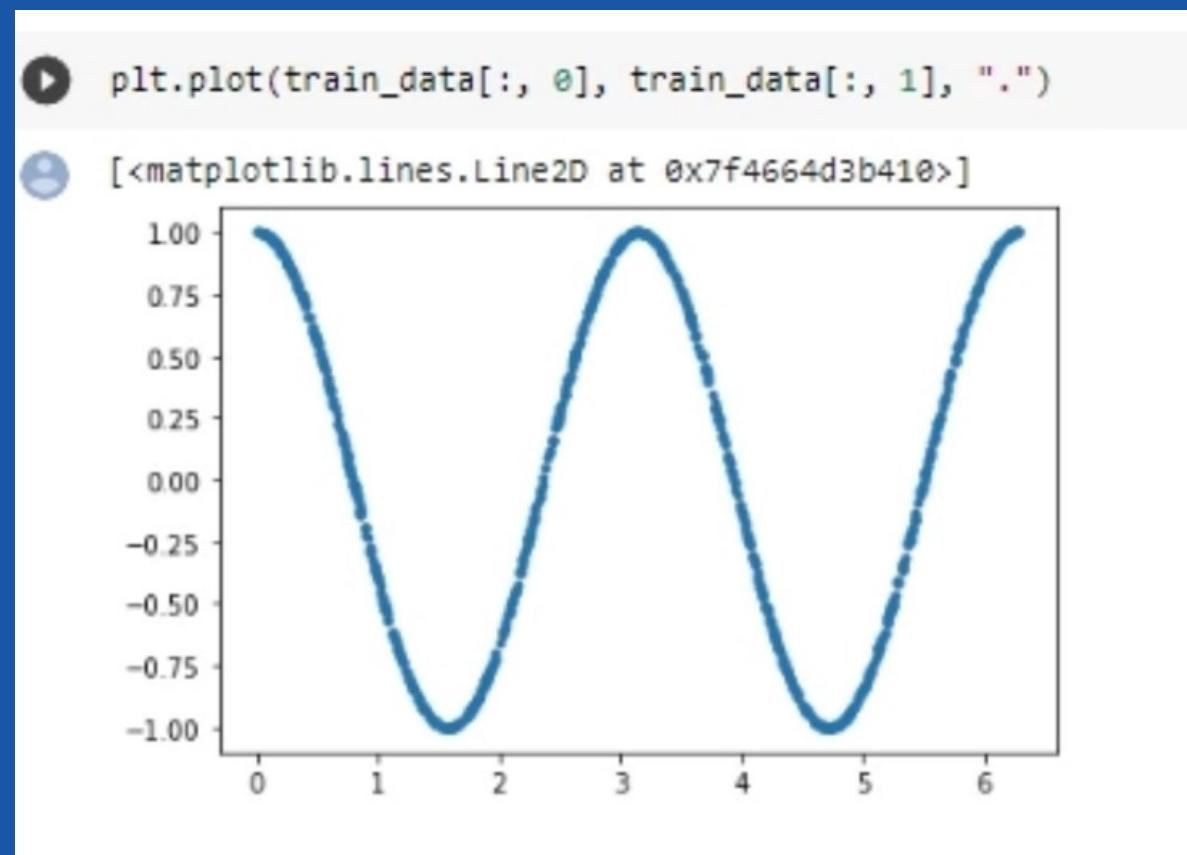
Набор данных с сайта Kaggle

GoogleColab

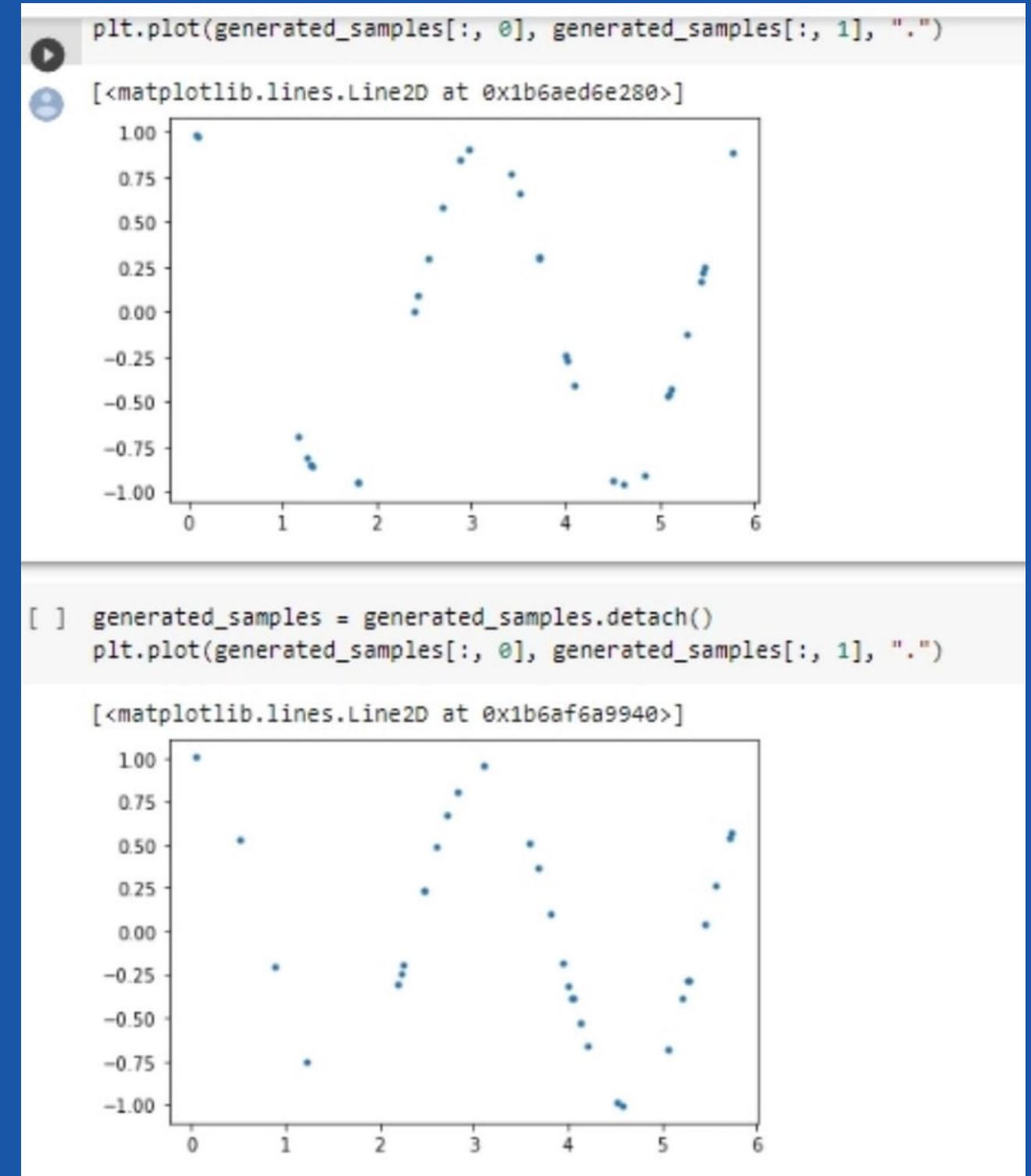
Начнем с простого

В качестве первого эксперимента с генеративно-состязательными сетями реализуем пример с гармонической функцией. Для работы с примером будем использовать популярную библиотеку PyTorch.

Тренировочные
данные



Что у нас получилось



Рассмотрим блоки кода

Дискриминатор

```
▶ class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(2, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 1),
            nn.Sigmoid(),
        )
    def forward(self, x):
        output = self.model(x)
        return output
[ ] discriminator = Discriminator()
```

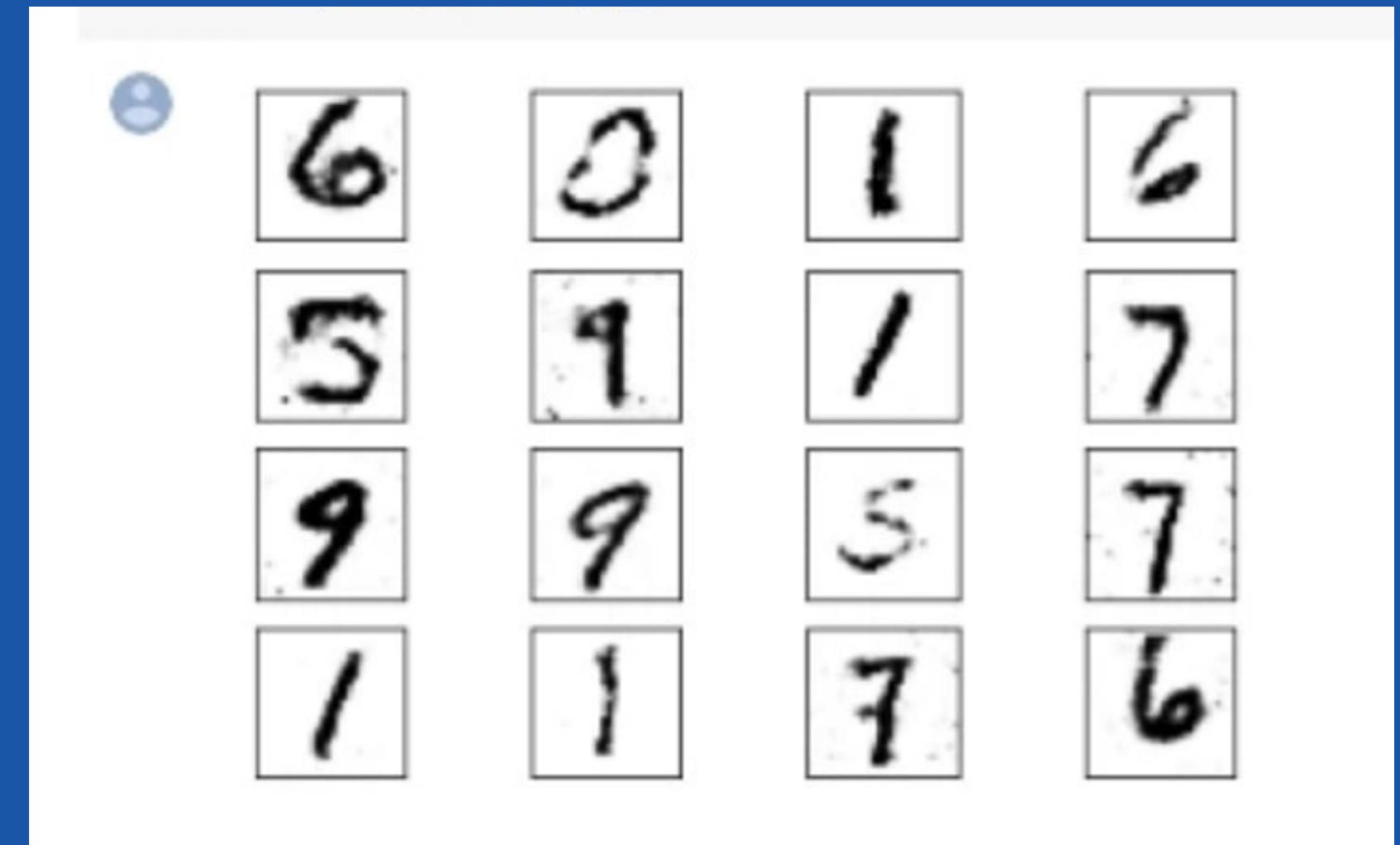
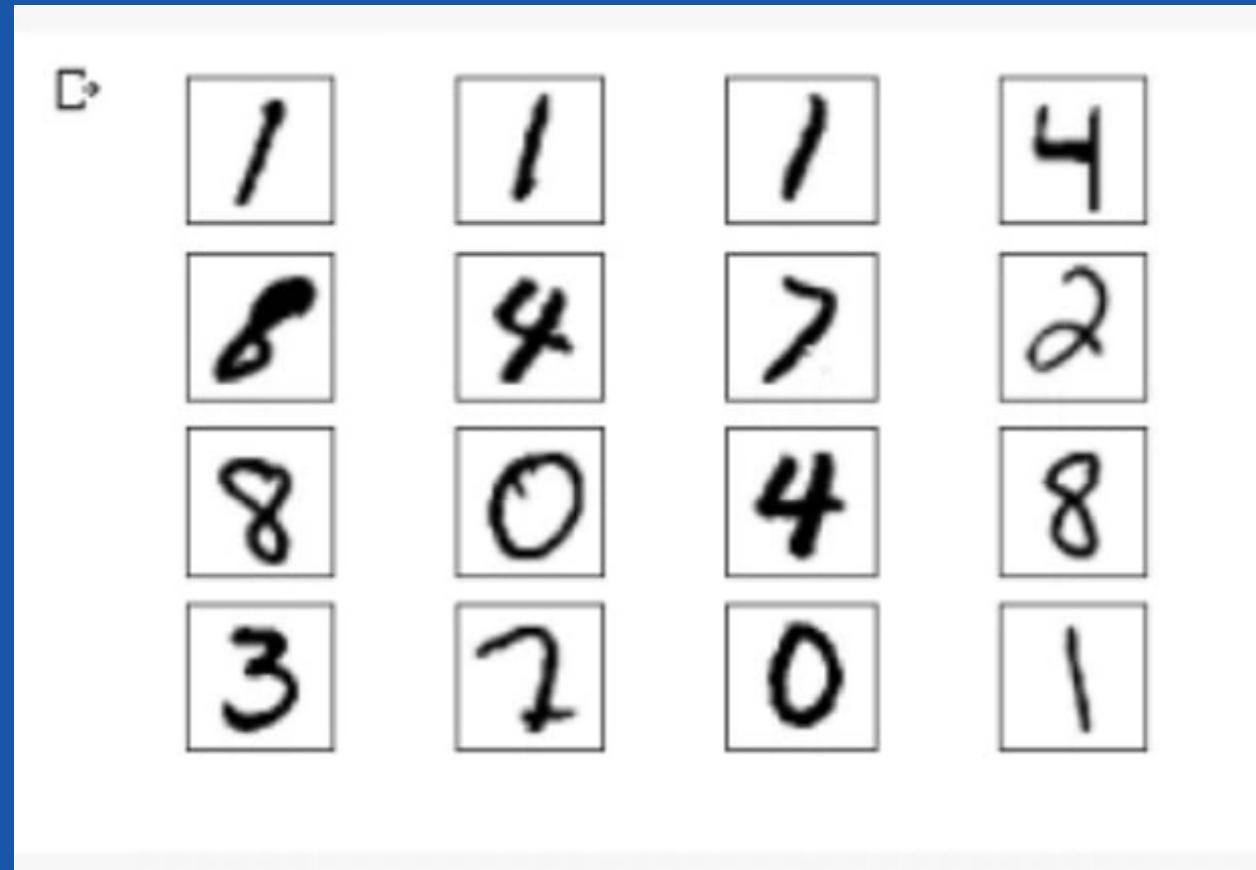
Генератор

```
▶ class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(2, 16),
            nn.ReLU(),
            nn.Linear(16, 32),
            nn.ReLU(),
            nn.Linear(32, 2),
        )
    def forward(self, x):
        output = self.model(x)
        return output
generator = Generator()
```

Генерация рукописных цифр

Загружаем набор данных MNIST

Создаём генератор и
дискриминатор. Обучаем их. И
через несколько эпох получаем:



Генерация рукописных цифр

Дискриминатор

```
▶ class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(784, 1024),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        x = x.view(x.size(0), 784)
        output = self.model(x)
        return output
```

```
[ ] discriminator = Discriminator().to(device=device)
```

Генератор

```
[ ] discriminator = Discriminator().to(device=device)

▶ class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(100, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, 784),
            nn.Tanh(),
        )

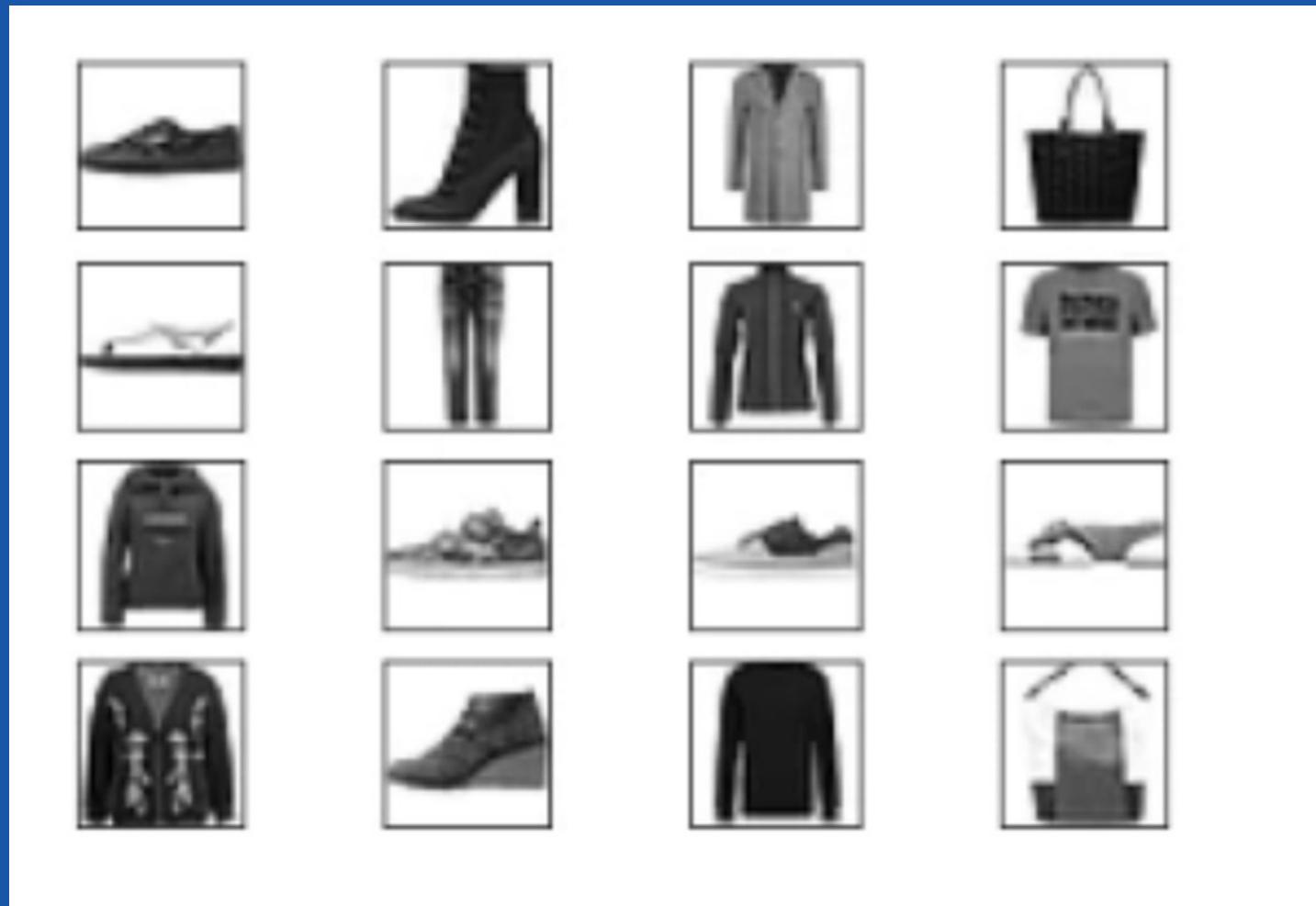
    def forward(self, x):
        output = self.model(x)
        output = output.view(x.size(0), 1, 28, 28)
        return output

generator = Generator().to(device=device)

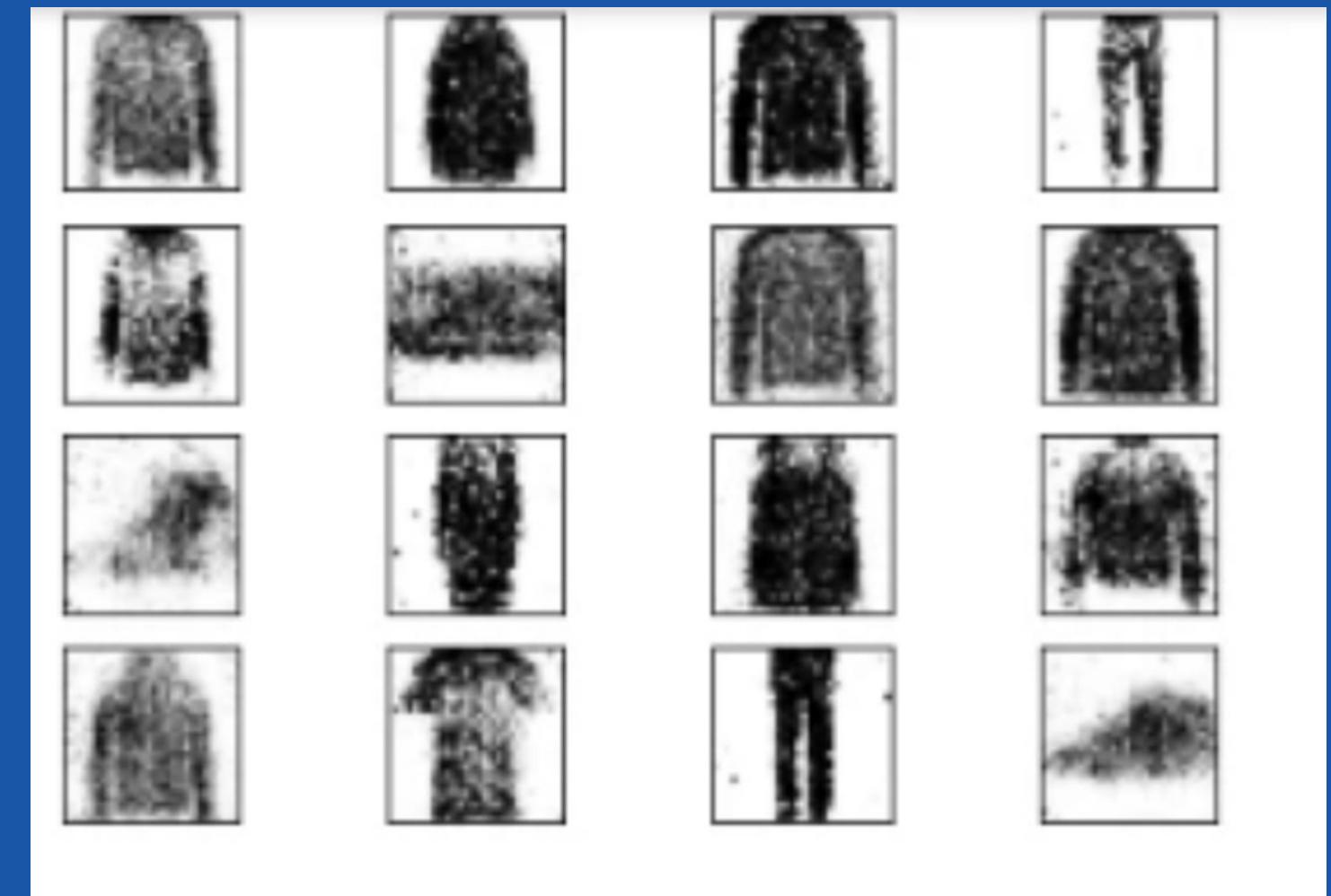
[ ] lr = 0.0001
```

Все зависит от входных данных!

Ради интереса загрузим в уже созданную GAN картинки с одеждой.



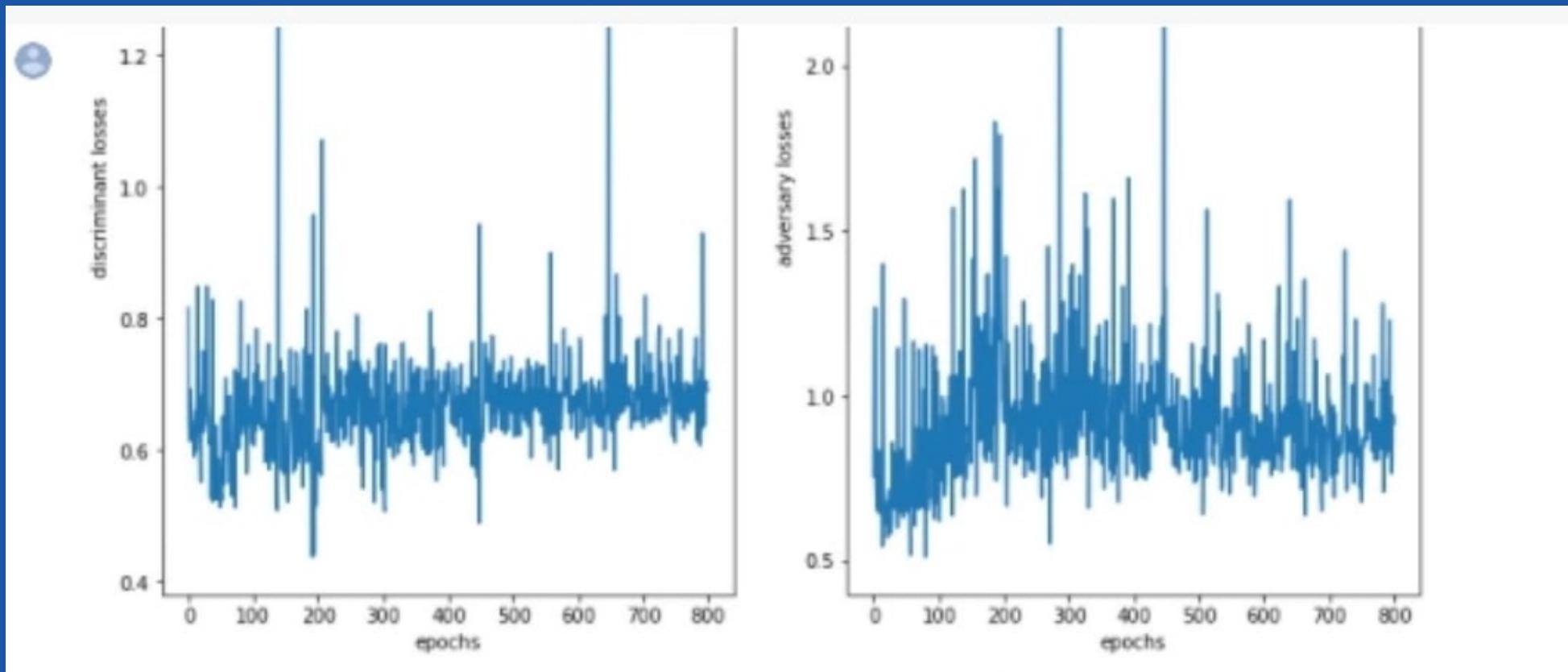
Результат довольно правдоподобный.



Генерация лиц

Еще более сложный процесс - сгенерировать лица людей. Такая работа требует много времени и вычислительных мощностей.

Загружаем в программу сет из 10000 лиц знаменитостей и смотрим результат 800 итераций.



Генерация лиц

Дискриминатор

```
def create_discriminator():
    disc_input = Input(shape=(HEIGHT, WIDTH, CHANNELS))

    x = Conv2D(256, 3)(disc_input)
    x = LeakyReLU()(x)

    x = Conv2D(256, 4, strides=2)(x)
    x = LeakyReLU()(x)

    x = Flatten()(x)
    x = Dropout(0.4)(x)

    x = Dense(1, activation='sigmoid')(x)
    discriminator = Model(disc_input, x)

    optimizer = RMSprop(
        lr=.0001,
        clipvalue=1.0,
        decay=1e-8
    )

    discriminator.compile()
```

Генератор

```
LATENT_DIM = 32
CHANNELS = 3
def create_generator():
    # вход - это вектор шума
    gen_input = Input(shape=(LATENT_DIM, ))
    # проекция вектора шума в тензор с такой же размерностью,
    # как последний сверточный слой дискриминатора
    x = Dense(128 * 16 * 16)(gen_input)
    x = LeakyReLU()(x)
    x = Reshape((16, 16, 128))(x)

    x = Conv2D(256, 5, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2D(512, 5, padding='same')(x)
    x = LeakyReLU()(x)
    x = Conv2D(512, 5, padding='same')(x)
    x = LeakyReLU()(x)
    x = Conv2D(CHANNELS, 7, activation='tanh', padding='same')(x)

    generator = Model(gen_input, x)
    return generator
```

Результаты



Были рассмотрены и изучены основные свойства и принципы работы GAN

Были выбраны и изучены средства для создания GAN

Были созданы и обучены несколько GAN

Были получены сгенерированные правдоподобные изображения

Спасибо за внимание!

