

Оглавление

[**Введение** 3](#_Toc168137208)

[**2. Анализ предметной области** 5](#_Toc168137209)

[**2.1. Выбор и развертывание среды разработки для языка программирования Python** 5](#_Toc168137210)

[**2.2. Анализ теоретических аспектов создания нейросетей** 5](#_Toc168137211)

[**2.3. Математическая модель нейронной сети** 5](#_Toc168137212)

[**2.4. Выводы по разделу** 5](#_Toc168137213)

[**3. Разработка нейронной сети** 5](#_Toc168137214)

[**3.1. Создание нейронов** 5](#_Toc168137215)

[**3.2. Создание нейронной сети** 5](#_Toc168137216)

[**3.2.1 Выбор топологии сети** 5](#_Toc168137217)

[**3.2.2. Группировка нейронных блоков в нейронную сеть** 5](#_Toc168137218)

[**3.2.3. Экспериментальный подбор характеристик сети** 5](#_Toc168137219)

[**3.3. Сбор и нормализация данных для обучения нейронной сети** 5](#_Toc168137220)

[**3.4. Обучение нейронной сети** 5](#_Toc168137221)

[**3.4.1. Выбор способа оценки адекватности результатов обучения** 5](#_Toc168137222)

[**3.4.2. Расчет количества потерь и их минимизация** 5](#_Toc168137223)

[**3.4.3. Анализ адекватности обучения** 5](#_Toc168137224)

[**3.5. Выводы по разделу** 5](#_Toc168137225)

[**Заключение** 5](#_Toc168137226)

[**Список используемых источников и литературы** 5](#_Toc168137227)

[**Приложение** 5](#_Toc168137228)

# **Введение**

Производственная практика проходила в Московском университете им. С.Ю. Витте на кафедре информационных систем. В ходе работы была разработан нейросеть-генератор 2D ассетов персонажей. Актуальность выражается популярностью нейросетей по генерации изображений. В рамках GameDEV наличие хороших и качественных генераторов в 2D пиксельной графике персонажей безумно мало, а рынок инди-разработчиков постепенно увеличивает потребность в художниках, которых ещё меньше. Из-за чего была выбрана именно эта тема. При выполнении данной задачи использовались современные эффективные методы и инструменты машинного обучения.

**Цель практики**

Целью практики было получение опыта в реальной среде разработки, а также получение и закрепление теоретических знаний в области генерации 2D, сбору датасета, созданию действующих моделей. В конкретном случае было важно разработать парсер ассетов с сайтов, с последующей их обработкой, обучением модели и автокодировщика, а также применением модели в создании новых ассетов.

**Задачи практики**

1. Анализ теоретических аспектов создания нейросетей:

- Изучение современной литературы и интернет-источников.

- Проведение математических расчетов для нейронной сети: выбор функции активации, вычисление весов нейронов, оптимизация скорости обучения, расчет процесса обратного распространения ошибки.

2. Сбор набора данных

- Поиск интернет-ресурсов

- Сбор ассетов парсингом страниц

3. Разработка и создание нейронных сетей:

- Создание нейронов и нейронных блоков на языке программирования Python.

- Выбор топологии сети и экспериментальный подбор характеристик сети (число слоев, блоков, функции передачи).

4. Сбор и подготовка данных:

- Сбор данных для обучения нейронной сети.

- Подготовка и нормализация данных.

5. Обучение нейронной сети и оценка результатов:

- Обучение нейронной сети на основе обучающей выборки.

- Выбор способа оценки качества работы сети.

- Минимизация потерь в процессе обучения и проверка адекватности обучения.

Источниками информации являлись различные интернет-источники, туториалы, видеоролики и официальная документация. Благодаря которым получилось создать программу по генерации.

# **2. Анализ предметной области**

# **2.1. Выбор и развертывание среды разработки для языка программирования Python**

В рамках поставленной задачи по генерации 2д персонажей средой разработки был выбран VSCODE, а также CMD для обработки результатов.  
VSCODE был выбран из-за его удобства в использовании, поддержку расширений, а также производительность. Не мало важен был и опыт, проведённый с данной средой разработки, поэтому можно было сразу начать разработку. Причина выбора командной строки для обработки результатов, также из субъективного удобства, т.к. длительный опыт позволил быстро и эффективно запускать отдельные программы в отдельных окнах и эффективнее отслеживать процесс обучения и сбора данных.

*Развёртывание среды разработки*

Установка и настройка VSCODE

* Установка VSCODE:
* Нужен установочный файл с официального сайт VSCODE
* Запуск файла и установка по инструкции

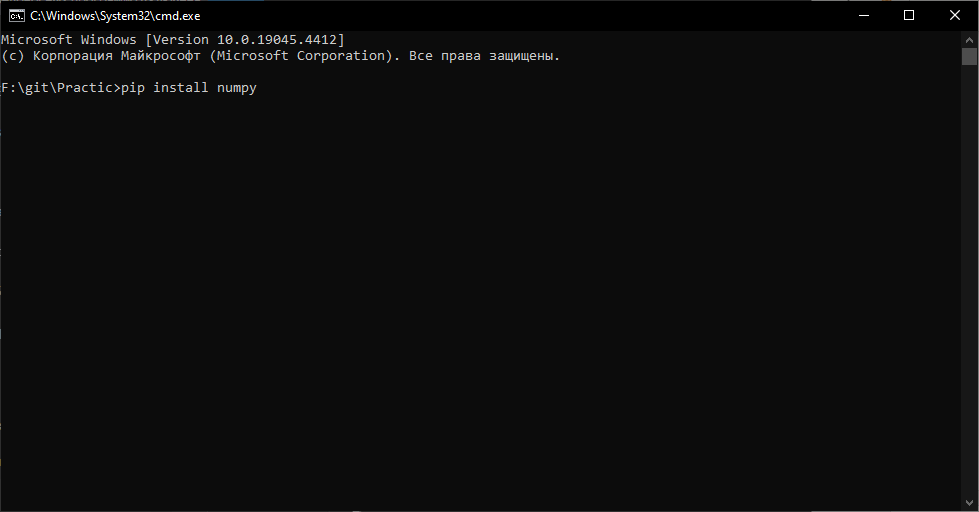
Установка расширений

* Открыть VSCODE
* Перейти в раздел Extensions (or CTRL+SHIFT+X)
* В работе требуются три расширения:  
  Python, Pylance, Jupyter.

Первые два обеспечивают поддержку языка и анализа кода, а последний включает себя работу с ноутбуком.  
В рамках моей работы разработка велась напрямую с .py , но .ipynb позволил некоторое время проводить тестирование и опыты в улучшении качества генерации.

Установка самого Python

Установка с официального сайта  
Добавление в системную переменную Path  
Установка необходимых библиотек с помощью pip

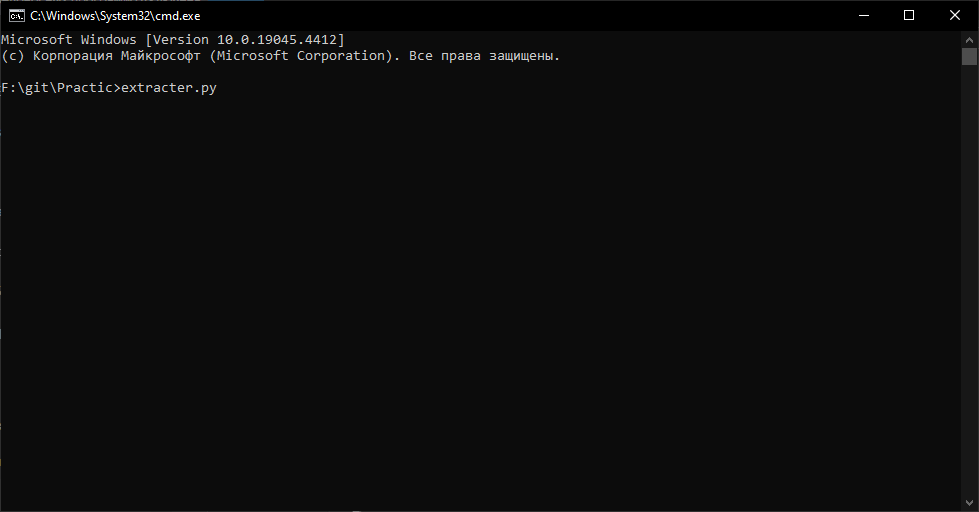


Использование командной строки CMD

Командная строка нужна была для обработки результатов.

Запуск скриптов был таким:  
Либо открывалась консоль, вводилось cd (путь), либо происходило нажатие на адресную строку проводника, в него вводилось cmd, что заменяло cd.

После уже в консоли вводилось наименование скрипта, например extracter.py , либо через python extracter.py.

В основном используемые библиотеки были:  
Pytorch, Torchvision, matplotlib, numpy.

Pytorch использовался в создании и обучении нейросети.

Torchvision для работы с компьютерным зрением в датасете и преобразовании.

Matplotlib визуализация и результаты нейросети.

Numpy работа с массивами и численными операциями.

Остальные библиотеки не представляли такую реализационную ценность, поэтому сюда не включены(или представляют расширенный функционал текущих указанных библиотек).

# **2.2. Анализ теоретических аспектов создания нейросетей**

Нейронная сеть – это вычислительная модель, работа которой аналогичная человеческому мозгу, а вернее его нейронным сетям. Компоненты такой сети являются нейроны, слои и связи. Эти нейроны принимают входные сигналы, после обрабатывают и передают выходные сигналы следующему слою.

Нейрон – это базовые элемент нейросети. Его задача в получении входных данных, их взвешивания, а после применения функции активации.  
Активация в свою очередь включать tanh(гиперболический тангенс) и ReLU(выпрямленную линейную единицу)

Слои нейросети имеют:  
Входной, Скрытый и Выходной слой.  
Входной принимает данные.  
Скрытый выполняет вычислительные действия. (Слоёв может быть несколько, а также состоят из множества нейронов)

Выходной слой даёт окончательный результат.

Весовые коэффициенты и смещения:  
Входные сигналы множатся на вес, а после складываются со смещением. Сами параметры будут обучатся в процессе обучения сети.

Архитектура нейросети:  
В данной работе было использовано две нейросети, это GAN и Autoencoders.

GAN

Состоит из генератора и дискриминатора.

Генератор создаёт изображение, стараясь обмануть дискриминатор о их реальности.

Дискриминатор в свою очередь пытается их отличить от сгенерированных.

Таким образом они борятся друг с другом постепенно улучшая свои навыки в создании реалистичных изображений.

Автокодировщик

Его используют в кодировании входных данных для компактного представления с целью восстановления исходных данных.

Он в свою очередь состоит из декодера и энкодера.

Энкодер преобразует входные в латентное пространство меньшего размера.

Декодер наоборот восстанавливает данные из такого пространства, стараясь их точно воспроизвести.

Обучение нейросети

Вся суть обучения сети в настройке весов и смещений для минимализации функции потерь. В данной работе использовались:  
  
Градиентный спуск.

Основной алгоритм которого минимизация потерь, а его вариации включают SGD и модификации ADAM, RMSprop.

Backpropagation

Это метод вычисления градиента функции потерь к весам сети. Градиенты в данном случае будут использованы для обновления весом с помощью алгоритма град. спуска.

Регуляризация

Предотвращает переобучение, включает в себя L2-регуляризацию и Dropout.

# **2.3. Математическая модель нейронной сети**

Генеративно-состязательная сеть (GAN)

1. Генератор

Генератор принимает на вход случайный вектор шума из латентного пространства и генерирует изображение .

Где:

- — распределение шума (обычно нормальное распределение).

- — сгенерированное изображение.

Архитектура генератора может быть описана как последовательность линейных преобразований (слоев) с нелинейными активациями:

Где:

- — веса и смещения -го слоя генератора.

2. Дискриминатор

Дискриминатор принимает на вход изображение (либо настоящее, либо сгенерированное) и выдаёт вероятность того, что это изображение настоящее.

Дискриминатор моделируется как последовательность линейных преобразований с нелинейными активациями, завершающаяся сигмоидной функцией:

Где:

- — сигмоидная функция.

- — веса и смещения -го слоя дискриминатора.

3. Функция потерь

Обучение GAN основано на минимаксной игре между генератором и дискриминатором. Функция потерь для дискриминатора:

Функция потерь для генератора:

Автокодировщик (Autoencoder)

1. Энкодер

Энкодер принимает на вход изображение и преобразует его в латентное представление .

Математически это можно описать как последовательность линейных преобразований с нелинейными активациями:

Где:

- — веса и смещения го слоя энкодера.

2. Декодер

Декодер принимает на вход латентное представление и восстанавливает изображение .

Декодер моделируется как последовательность линейных преобразований с нелинейными активациями:

Где:

- — веса и смещения -го слоя декодера.

3. Функция потерь

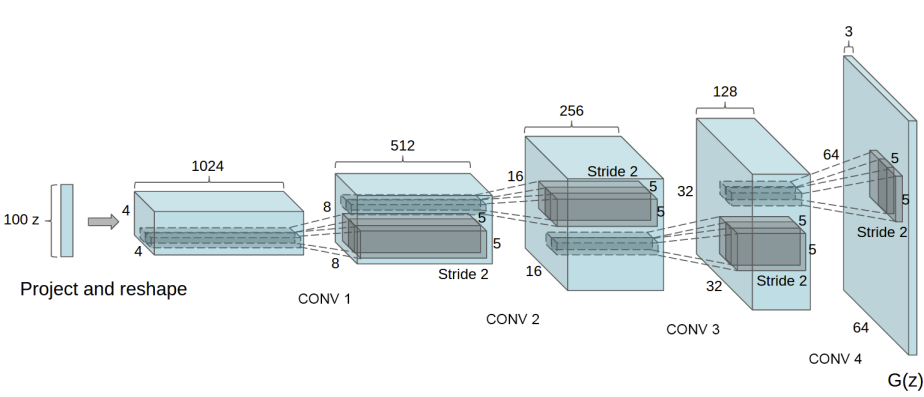
Функция потерь автокодировщика основана на разнице между входными и восстановленными изображениями. Обычно используется среднеквадратичная ошибка (MSE):

Где:

- — входное изображение.

- — восстановленное изображение.

- — количество образцов в обучающей выборке.



# **2.4. Выводы по разделу**

Процесс разработки нейросети для 2д ассетов персонажей представлял себя из 3-ёх этапов. Это выбор и развёртывание среды, в которые входили использование VSCODE в котором была написана программа, а также CMD для выполнения обработки результатов. Также были установлены библиотеки для комфортной и эффективной работы.

Далее происходил анализ теоретических аспектов нейросети. Мы изучили основные принципы и понятия, разобрались в определениях нейронов, слоёв, функций активации и построили архитектуру нашей программы. В неё вошли GAN и Autoencoder, которые и представляют из себя созданную нейросеть. Также в работу был включён градиентный спуск и обратное распространение ошибок с регуляризацией.

После этого была продемонстрирована математическая модель, в которой было описание применения GAN и Autoencoder. Были уравнения, которые описывали генератор, дискриминатор и функцию потерь. Они-же демонстрируют архитектуру и принципы работы данных моделей в нашей теме.

В целом мы приобрели понимание работы моделей, разобрались что именно обеспечивает эффективность и точность у нейросетей в задачах генерации 2д.

Выводы по разделу 1

|  |  |  |
| --- | --- | --- |
| Выводы | Сформированные компетенции | |
| Код компетенции | Содержание компетенции |
| Проведен выбор и развертывание среды разработки для языка программирования Python. | ПК-9 | Способность разрабатывать методы извлечения, анализа и обработки информации |
| Проведен анализ теоретических аспектов создания нейросетей |
| Разработана математическая модель нейронной сети | ПК-10 | Способность применять математические методы моделирования процессов обработки информации с использованием средств интеллектуального анализа данных и машинного обучения |

# **3. Разработка нейронной сети**

# **3.1. Создание нейронов**

Нейроны – это основа любой разработки нейронной сети.

Это буквально создание отдельных элементов, которые будут производить вычисления и преобразования входных данных.

Основным принципом является создание нейронных блоков из этих нейронов, которые и будут строить нейросеть.

Основные компоненты, как мы обсуждали ранее, это веса, смещения и функции активации. Конкретно в данной работе используются нейроны: ReLU (Rectified Linear Unit) и Leaky ReLU.

Для тренировки и создания нейрона использовались библиотеки torch, torch.nn, torch.optim.

Torch Создаёт нейросеть и работаем с тензером.

Torch.nn Создаёт слои нейросети.

Torch.optim Реализует оптимизацию.

Продемонстрируем примеры работы в коде:

Нейроны с ReLU

import torch

import torch.nn as nn

class SimpleNeurons(nn.Module):

def \_\_init\_\_(self):

super(SimpleNeurons, self).\_\_init\_\_()

self.layer = nn.Linear(128, 64)

self.activation = nn.ReLU()

def forward(self, x):

x = self.layer(x)

x = self.activation(x)

return x

Нейроны Leaky ReLU

class LeakyReLUNeurons(nn.Module):

def \_\_init\_\_(self):

super(LeakyReLUNeurons, self).\_\_init\_\_()

self.layer = nn.Linear(128, 64)

self.activation = nn.LeakyReLU(0.2)

def forward(self, x):

x = self.layer(x)

x = self.activation(x)

return x

Таким образом мы написали слои из нейронов для успешной работы нашего проекта по генерации 2д ассетов персонажей.

# **3.2. Создание нейронной сети**

Теперь рассмотрим конкретно создание нейросети. Объединим нейроны в единую архитектуру в виде нейронных блоков и слоёв.

Основными этапами станет:

Определение архитектуры сети. Для этого выберем типы слоёв и их последовательность.

Инициализация слоёв и параметров. Представляет из себя стартовые слои и начальные параметры.

Определение функции потерь и оптимизатора. Оценка сети и оптимизация.

Сборка и компиляция. Все слои объединены в модель.

Генератор для GAN

import torch

import torch.nn as nn

class Generator(nn.Module):

def \_\_init\_\_(self):

super(Generator, self).\_\_init\_\_()

self.main = nn.Sequential(

nn.ConvTranspose2d(100, 64 \* 8, 4, 1, 0, bias=False),

nn.BatchNorm2d(64 \* 8),

nn.ReLU(True),

nn.ConvTranspose2d(64 \* 8, 64 \* 4, 4, 2, 1, bias=False),

nn.BatchNorm2d(64 \* 4),

nn.ReLU(True),

nn.ConvTranspose2d(64 \* 4, 64 \* 2, 4, 2, 1, bias=False),

nn.BatchNorm2d(64 \* 2),

nn.ReLU(True),

nn.ConvTranspose2d(64 \* 2, 64, 4, 2, 1, bias=False),

nn.BatchNorm2d(64),

nn.ReLU(True),

nn.ConvTranspose2d(64, 3, 4, 2, 1, bias=False),

nn.Tanh()

)

def forward(self, input):

return self.main(input)

Дискриминатор для GAN

class Discriminator(nn.Module):

def \_\_init\_\_(self):

super(Discriminator, self).\_\_init\_\_()

self.main = nn.Sequential(

nn.Conv2d(3, 64, 4, 2, 1, bias=False),

nn.LeakyReLU(0.2, inplace=True),

nn.Conv2d(64, 128, 4, 2, 1, bias=False),

nn.BatchNorm2d(128),

nn.LeakyReLU(0.2, inplace=True),

nn.Conv2d(128, 256, 4, 2, 1, bias=False),

nn.BatchNorm2d(256),

nn.LeakyReLU(0.2, inplace=True),

nn.Conv2d(256, 512, 4, 2, 1, bias=False),

nn.BatchNorm2d(512),

nn.LeakyReLU(0.2, inplace=True),

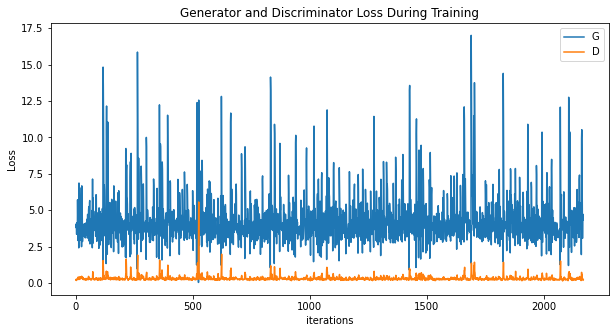
nn.Conv2d(512, 1, 4, 1, 0, bias=False),

nn.Sigmoid()

)

def forward(self, input):

return self.main(input)





Автокодировщик

class Autoencoder(nn.Module):

def \_\_init\_\_(self):

super(Autoencoder, self).\_\_init\_\_()

self.encoder = nn.Sequential(

nn.Linear(784, 256),

nn.ReLU(True),

nn.Linear(256, 64),

nn.ReLU(True)

)

self.decoder = nn.Sequential(

nn.Linear(64, 256),

nn.ReLU(True),

nn.Linear(256, 784),

nn.Sigmoid()

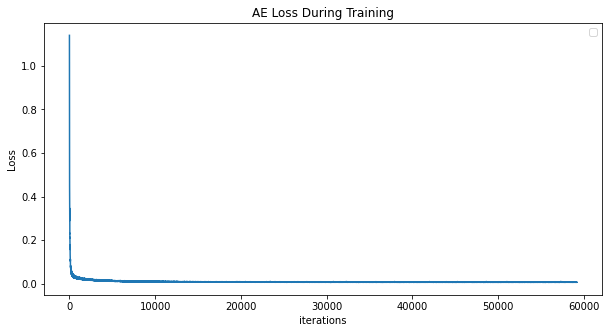
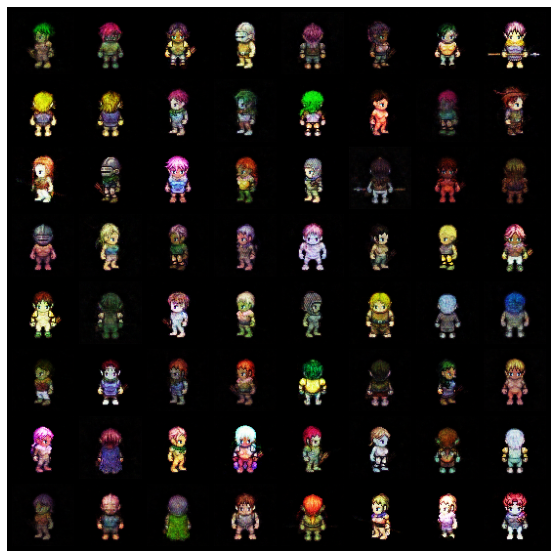
)

def forward(self, x):

x = self.encoder(x)

x = self.decoder(x)

return x

Само создание нейросети, это процесс многоэтапный, где присутствует выбор архитектуры, параметров, настроек функции потерь, оптимизаций и сборки. Здесь мы представили многое, включая действующие фрагменты кода по созданию нейросети. Уже с этим набором, можно эффективно и быстро начать решать поставленную задачу.

# **3.2.1 Выбор топологии сети**

Выбор топологии в данной работе являлся критически важным, ведь это напрямую влияло на взаимодействие слоёв, ошибка в которых могла нарушить эффективность обучения и создать проблему на выходе.

Из-за этого были использованы некоторые принципы в выборе топологии, а это:  
Корректные цели и задачи, в лице генерации 2д ассетов именно персонажей, т.к. при более широком применении на более широком датасете, стали бы возникать неоднозначные проблемы с генерацией и дефекты из-за шума.

Используемые слои выбирались определённо, свёрточные для обработки, полносвязные для классификации, а генерации были слои нормализации и активации.

Сама глубина сети определялась исходя из сложности задачи и объёма датасета. В нашем случае датасет был из 1000 ассетов, что уже было большим объёмом, но недостаточным для идеального обучения. Переобучения сеть не подверглась.

Архитектурой стали проверенные временем GAN и Autoencoder, т.к. не только содержали различные обучающие туториалы по своей применению, но и открытые исходные ресурсы для ознакомления в разных областях.

Топология генератора

В него входил случайный вектор шума, после чего преобразовался в изображение через серию транспонированных свёрт. слоёв, после идёт нормализация и активация. Благодаря чему качество было достаточно хорошим, даже при малом наборе данных.

Топология дискриминатора

На вход шли изображения, которые обрабатывались через свёрт.слой., после нормализацию и активацию. За счёт этого получалось различать реальные от сгенерированных изображений.

Топология автокодировщика

Он состоял из энкодера и декодера. Первый преобразовал вход в латентное представлен, а второй наоборот восстанавливал из него.

За счёт такого выбора топологии, удалось добиться приемлемых результатов генерации и оптимизировать выполнение программы.

# **3.2.2. Группировка нейронных блоков в нейронную сеть**

Нейроблоки – это модуль внутри нейросети. Они состоят из множества нейронов, которые производят операции по обработке данных. Группа таких блоков даёт возможность создать сложную архитектуру сети, которая сможет обучиться для работы на разных задачах.

1. Генератор GAN

В неё входят нейронные блоки, которые состоят из транспонированных слоёв свёртки, нормализации и активации. Они превращают шум в изображение.

Группировка блоков обеспечивает увеличение размерности данных от латентного до целевого изображения.

1. Энкодер и декодер

Состоит энкодер из свёрт.слоёв., слоёв утечки, нормализации. Нужен он для сжатия, а декодер, наоборот, для восстановления.

В практике это позволяет создавать сложные много или узко задачные сети из блоков, которые адаптируются и эффективно выполняет разнохарактерную работу.

# **3.2.3. Экспериментальный подбор характеристик сети**

Настройка гиперпараметров, который включает в себя определение оптимальных значений, которые влияют на производительность и результаты. Основными стоит выделить:  
Число слоёв и число нейронов, благодаря которым увеличивается улавливание сложные паттернов, но из минусов увеличивается и риск переобучения.

Размеры пакетов и эпох, что позволяет ускорить обучение, но снизить качество.

Скорость обучения, которая сильно позволяет экономить время, но легко может проскочить минимум, что приведёт к ошибкам в обучении на выходе.

Функция активации, для преобразования входных и выходных при разных задачах, могут лучше работать при разном подходе.

Оптимизаторы, которые также влияют на скорость и качество.

В методы экспериментального подбора стоит отнести:  
Ручная настройка, которая по сути трудоёмкая, но за то позволяет глубже понять суть работы и влияния на процесс обучения.

Сеточный поиск, представляющий себя из множества комбинаций параметров, но сильно требователен к ресурсам ПК.

Случайный поиск, который работает в заданных границах, а за счёт работы в больших пространствах, более эффективен, чем сеточный.

Байесовская оптимизация, используется для прогнозирования у параметров, для поиска лучших результатов.

В конкретной работе используется:

На нейросети:  
Число слоёв и размер фильтров, функции активации , оптимизаторы со скоростью обучения из коэффициента бета, также размер пакета и число эпох.

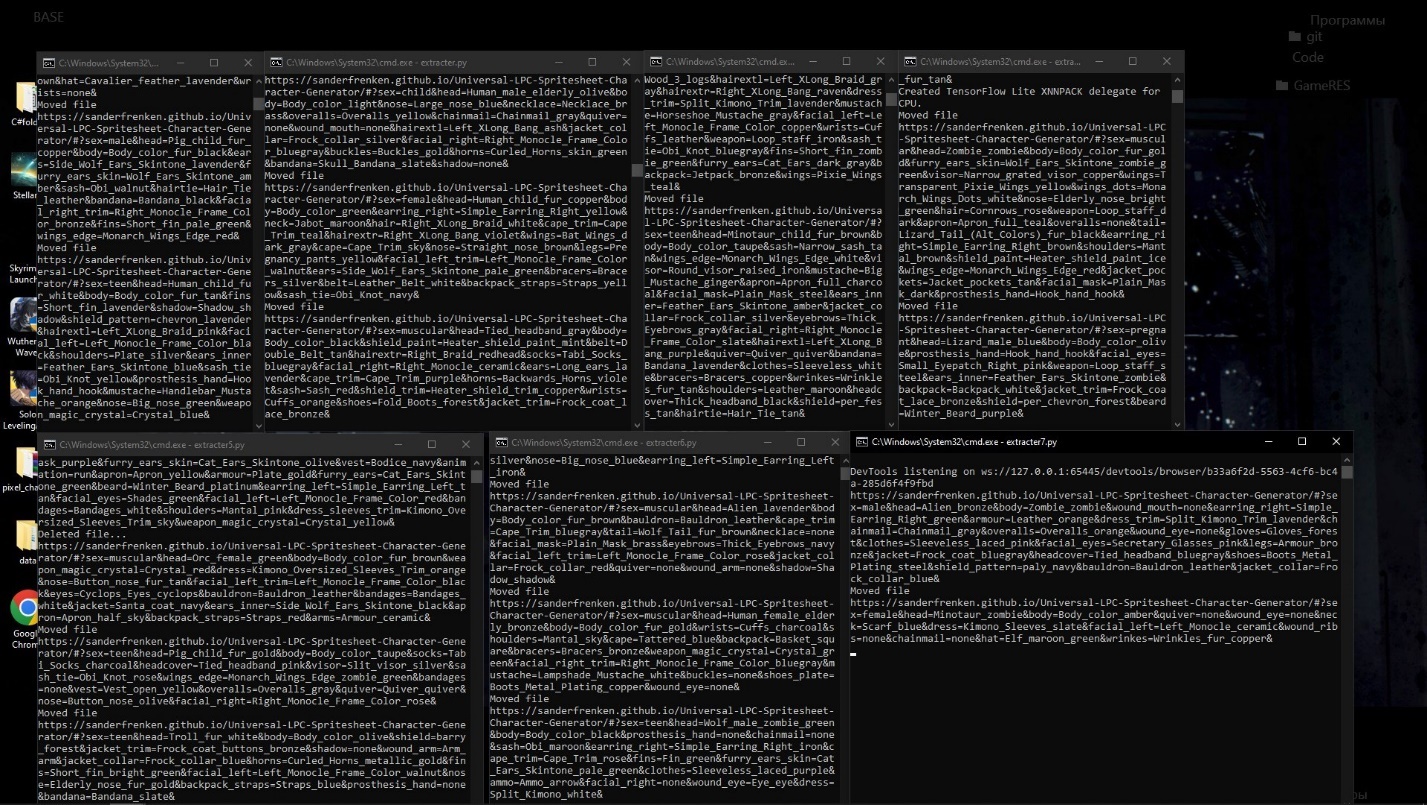
Автокодировщик:  
Архитектура сети, функции активации, оптимизаторы с бета, размер пакета и число эпох.

# **3.3. Сбор и нормализация данных для обучения нейронной сети**

Сбор и подготовка датасета – это основа всей работы с нейросетью. Ведь без него обучить будет не на чем. Поэтому для грамотной разработки нейросети следует правильно набрать данные и предварительно их обработать.

Сбор данных – процесс сбора материала, на котором нейросеть обучится. В нашей работе это были 2д ассеты, с разных интернет ресурсов.

Вообще датасеты делятся на два типа.  
Публичные, которые заранее сделаны для использования.  
Собственные, которые собираются самостоятельно из источников.



Наш датасет представляет собой самостоятельно сборный с сайта.  
<https://sanderfrenken.github.io/Universal-LPC-Spritesheet-Character-Generator/#?body=Body_color_light&head=Human_male_light>

Где мы выгружали ассеты на кастомных генерируемых нашей программой персонажей.

С помощью extracter.py мы автоматически собрали персонажа по заданным ссылкам, генерировали его ассет, а после выводили в пнг формате. Далее собирали 1000 ассетов в одной папке и запускали sprigan.py который вырезал по заданным координатам фрагменты персонажей и формировал датасет для дальнейшей работы в нейросети.

Нормализация данных представляла из себя приведение всех признаков к единому масштабу в заданном заранее диапазоне.

После проходилась стандартизация, где вычиталось среднее и делилось на отклонение, чтобы значение среднего было 0, а отклонения 1.

Подробнее о программах:  
extracter.py  
Парсил через Selenium и BeautifulSoup и извлекал файлы из архива.

Работа был представлена через генерацию ссылок с параметрами и запуска скачивания.

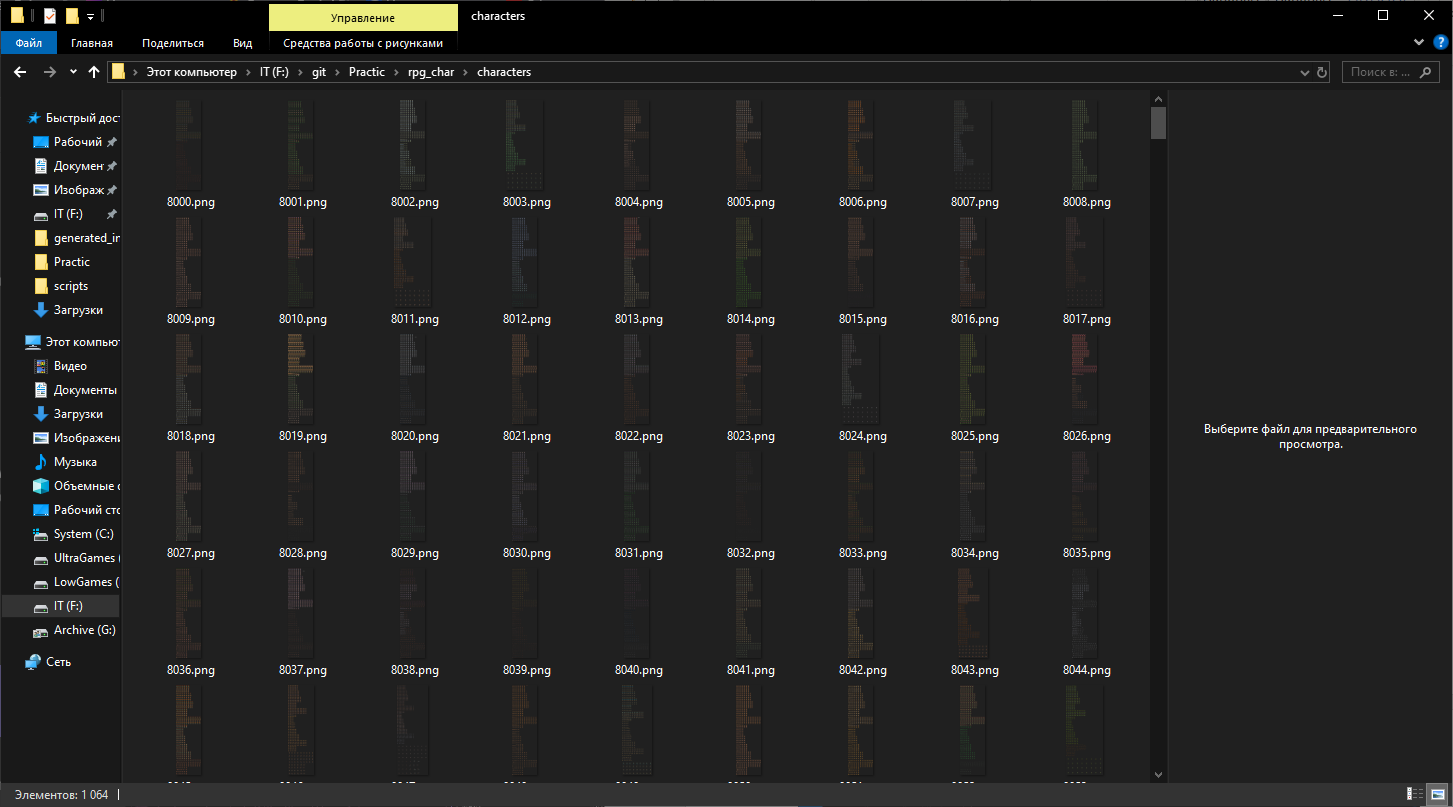
with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_path)

и

URL = generate\_random\_url(BASE\_URL, categories, all\_items, only\_male, only\_female)

driver.get(URL)



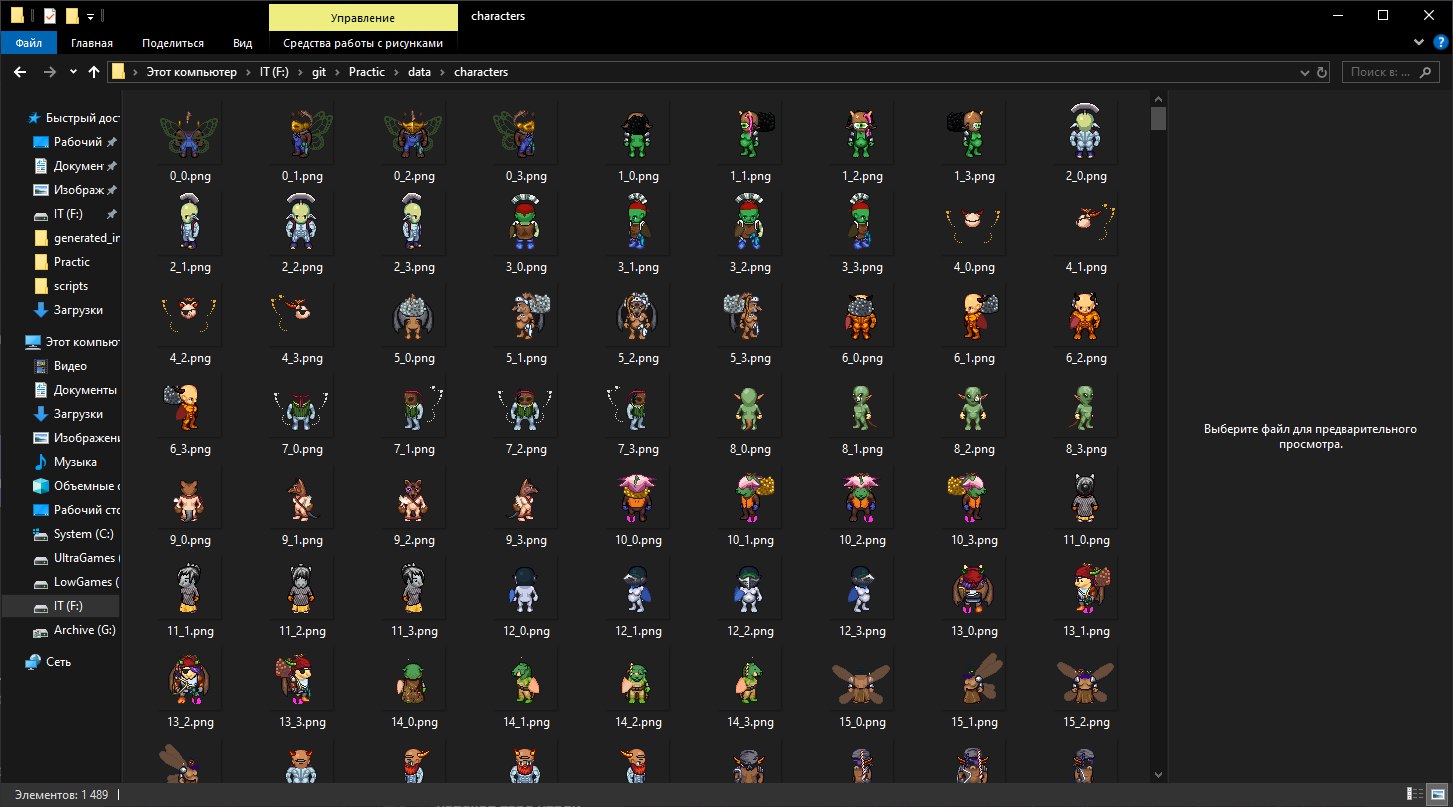
sprigan.py

Извлекал отдельные спрайты из листов по координатам и сохранял в директорию для нейросети.

image = mpimg.imread(spritesheet) # Чтение изображения

resized\_im = image[start\_h + char\_h \* nr : start\_h + char\_h \* (nr + 1), start\_w : start\_w + char\_w, :]

plt.imsave(path + str(i) + '\_' + str(nr) + '.png', resized\_im) # Сохранение изображения

****

# **3.4. Обучение нейронной сети**

Обучение нейросети – это возможность моделе распознавать закономерности в данных.

Для обучения следует собрать и предобработать набор данных, как было обсуждено ранее.  
После уже задать архитектуру, кол-во слоёв, функции и параметры.

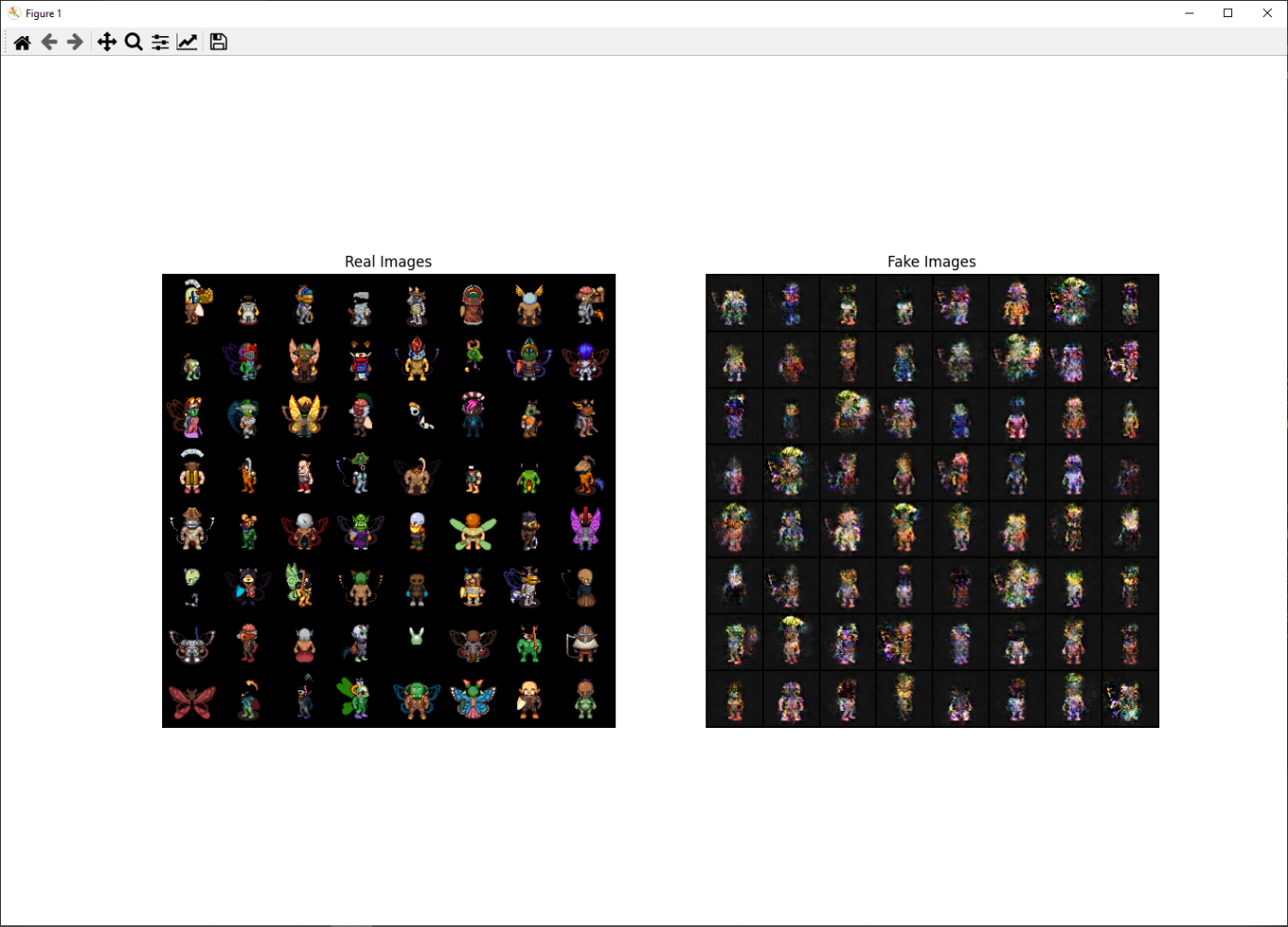
Далее поработать с функцией потерь и оптимизатором.

Функции потерь (например, кросс-энтропия, MSE) определяют, насколько хорошо модель справляется с задачей на этапе обучения.

Оптимизаторы (например, Adam, SGD) определяют, как модель будет обновлять свои веса на основе данных и функции потерь.

Обучение модели происходит в эпохах, каждая из которых включает проход модели по всему обучающему набору данных. На каждом шаге (batch) модель делает предсказания, вычисляет ошибку (потери) и обновляет свои веса (параметры).

После каждой эпохи модель оценивается на валидационном наборе. Это позволяет отслеживать, как модель справляется с новыми данными (не участвующими в обучении), и помогает предотвратить переобучение. Финальное тестирование проводится после завершения всех эпох обучения для оценки общей эффективности модели.





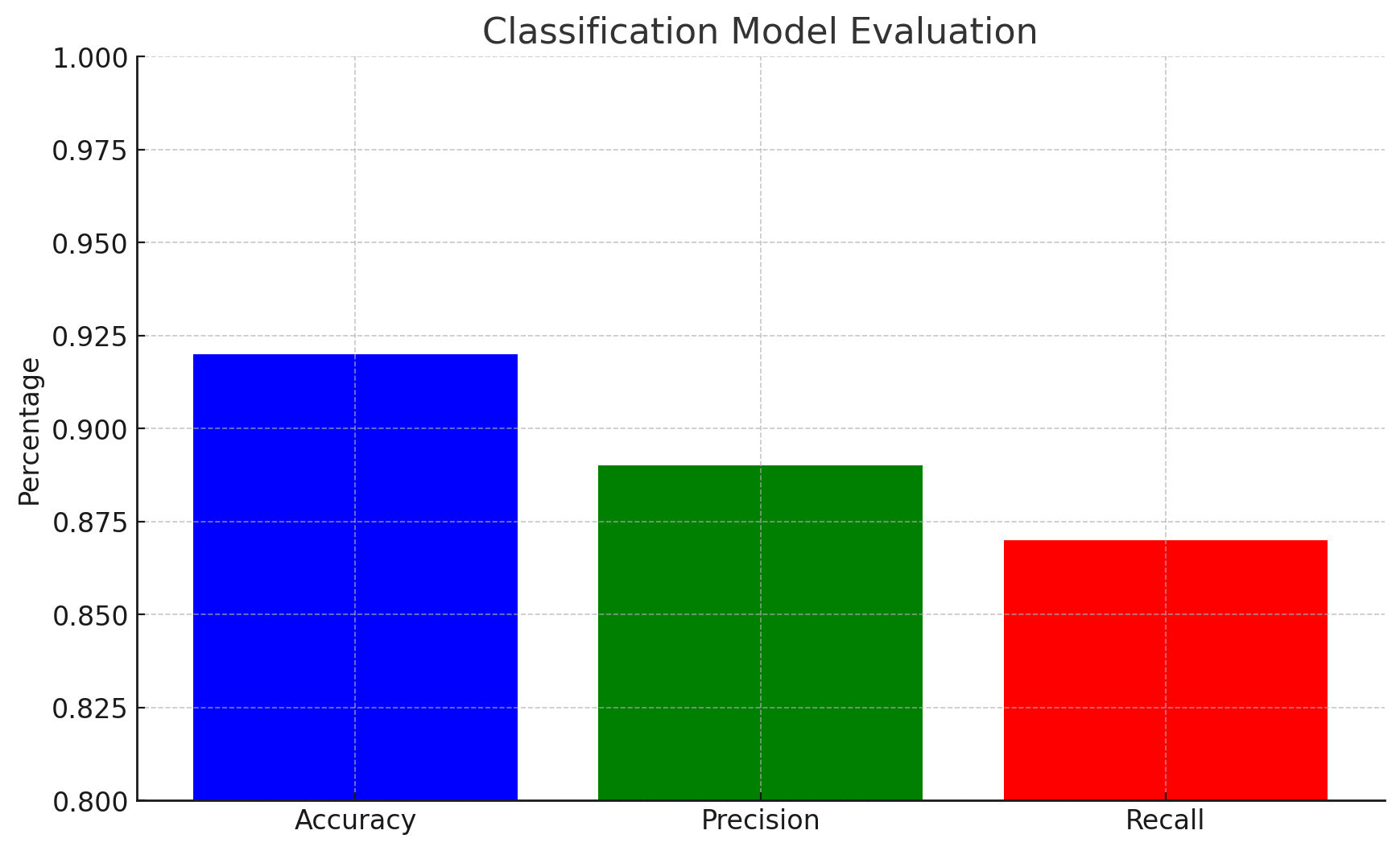
# **3.4.1. Выбор способа оценки адекватности результатов обучения**

Для задач классификации метрики:

Точность (Accuracy): Процент правильно классифицированных примеров из всего набора данных.

Точность (Precision) и Полнота (Recall): Точность измеряет долю правильных предсказаний среди всех предсказанных положительных, а полнота — долю найденных истинно положительных примеров среди всех истинных положительных.

F1-мера: Гармоническое среднее точности и полноты, используется для сбалансированной оценки между ними.

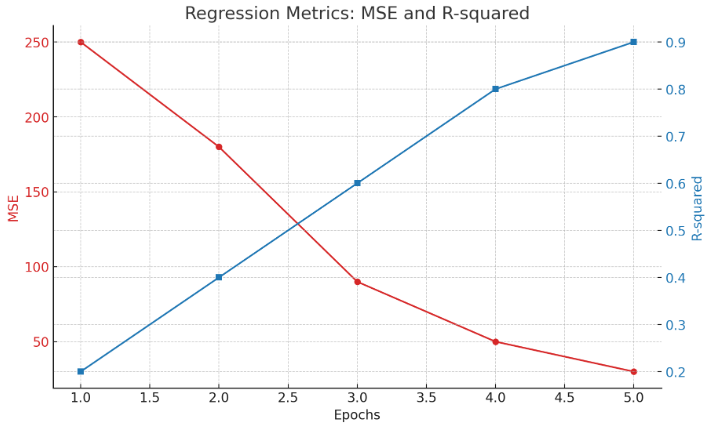


Для регрессии метрики:

Среднеквадратичная ошибка (MSE): Среднее квадратов разностей между предсказанными и истинными значениями, характеризует общую ошибку модели.

Средняя абсолютная ошибка (MAE): Среднее абсолютных значений разностей между предсказанными и истинными значениями, полезно для оценки величины ошибки без учёта направления.

Коэффициент детерминации (R²): Показывает, какая доля дисперсии зависимой переменной объясняется независимыми переменными в модели.

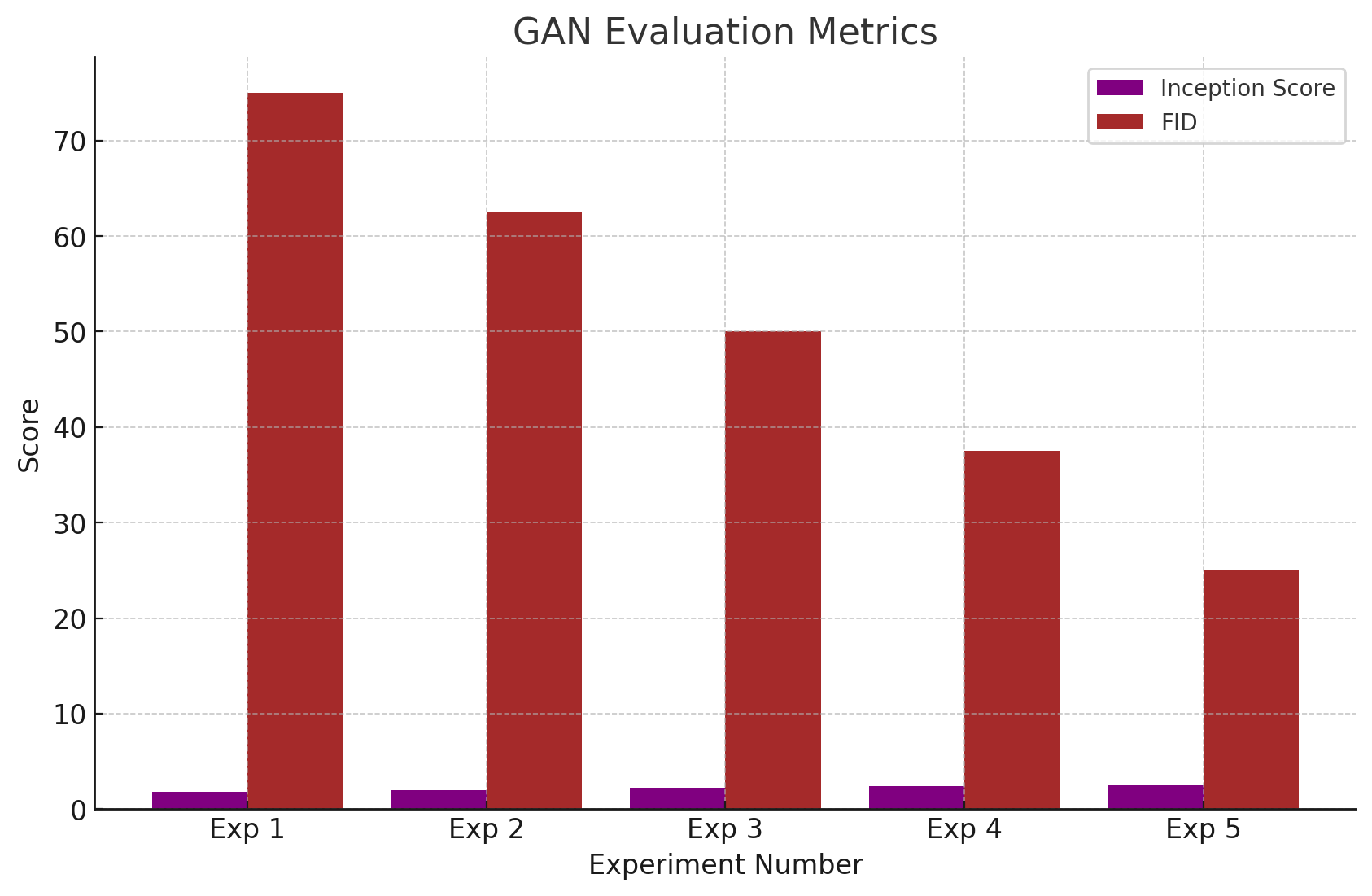


Для задач генерации GAN

Inception Score (IS): Использует классификатор Inception для оценки разнообразия и качества сгенерированных изображений.

Fréchet Inception Distance (FID): Измеряет сходство между распределениями реальных и сгенерированных изображений, чем ниже значение, тем лучше.

Перцептуальная потеря: Используется для оценки сходства между реальными и сгенерированными изображениями на основе признаков, извлечённых предобученными сетями.



Перекрёстная проверка (кросс-валидация): Эффективный метод оценки и улучшения устойчивости модели, при котором данные разбиваются на подмножества, и модель обучается и тестируется на разных комбинациях этих подмножеств.

Оценка на валидационном наборе: Регулярно проверяет производительность модели на данных, не участвовавших в обучении, чтобы избежать переобучения.

# **3.4.2. Расчет количества потерь и их минимизация**

Функция потерь (или функция ошибки) количественно оценивает, насколько предсказания модели отличаются от фактических значений. Цель обучения состоит в том, чтобы найти параметры модели, которые минимизируют эту функцию потерь на наборе тренировочных данных.

Функция потерь выбирается исходя из типа задачи:

Для задач классификации часто используется кросс-энтропия, которая измеряет разницу между двумя вероятностными распределениями: предсказанным и фактическим.

Для задач регрессии широко применяется среднеквадратичная ошибка (MSE), которая вычисляет средний квадрат разницы между предсказанными и истинными значениями.

Для задач генерации данных, таких как работа с GAN, могут использоваться сложные функции, такие как adversarial loss, которая включает компоненты для генератора и дискриминатора.

Методы минимизации потерь

Градиентный спуск: Это основной метод оптимизации, используемый для тренировки нейронных сетей. Суть метода заключается в итеративном обновлении параметров модели в направлении, противоположном градиенту функции потерь (направление наискорейшего увеличения функции).

Варианты градиентного спуска:

SGD (Stochastic Gradient Descent): Обновление параметров происходит на каждом шаге после обработки одного или нескольких примеров обучающего набора.

Momentum: Добавляет понятие инерции в обучение, помогая быстрее сходиться и избегать локальных минимумов.

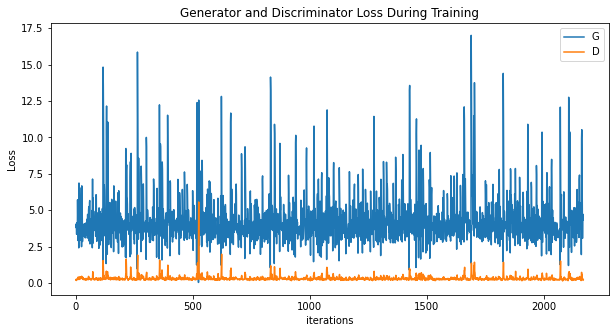
Adaptive learning rate methods: Алгоритмы, такие как Adam, Adagrad или RMSprop, которые адаптируют скорость обучения в зависимости от параметров, что часто приводит к лучшим результатам на практике.

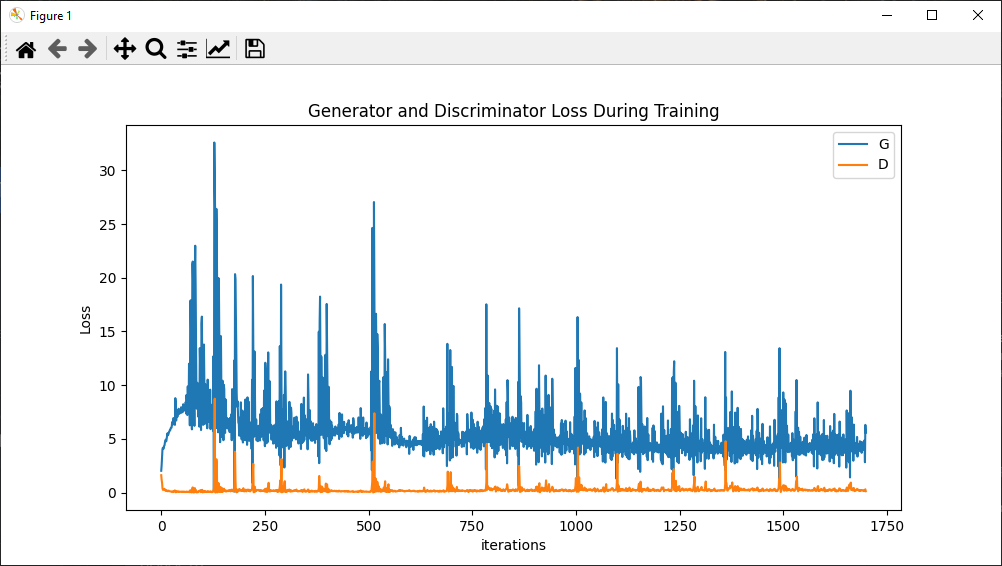
Регуляризация

Регуляризация помогает контролировать сложность модели и бороться с переобучением, добавляя штраф к функции потерь за слишком большие веса или слишком сложные модели:

L1 регуляризация (Lasso) добавляет штраф, равный абсолютной величине коэффициентов.

L2 регуляризация (Ridge) добавляет штраф, равный квадрату величины коэффициентов (чаще используется в глубоком обучении).





# **3.4.3. Анализ адекватности обучения**

Этот процесс включает оценку того, как модель справляется с новыми данными, и выявление возможных проблем, таких как переобучение или недообучение.

Ключевые аспекты анализа адекватности обучения:

Кривые обучения (Learning Curves):

Кривые потерь: Показывают изменение функции потерь на обучающем и валидационном наборах данных в течение обучения. Различие между этими кривыми может указывать на переобучение (если тренировочные потери низкие, а валидационные — значительно выше) или недообучение (если обе кривые высокие).

Кривые точности: Аналогично кривым потерь, но отражают точность модели.

Анализ ошибок:

Матрица ошибок (Confusion Matrix): Позволяет визуально оценить, какие классы чаще всего путаются с другими, что особенно полезно для задач классификации.

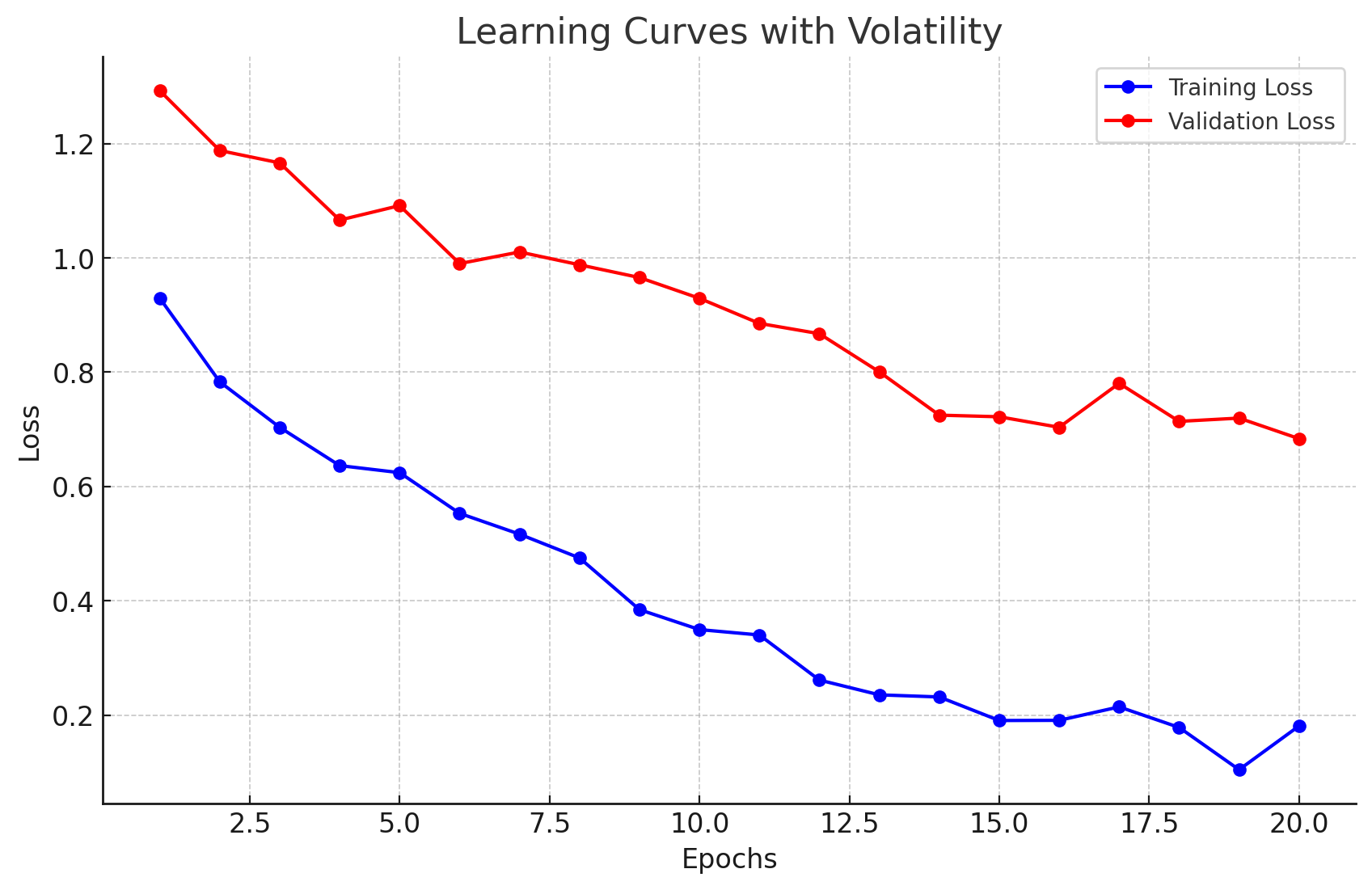
Отчет о классификации: Включает метрики точности, полноты, F1-меры для каждого класса, что помогает понять эффективность модели по различным категориям.

Валидация модели:

Перекрестная проверка (Cross-Validation): Оценивает устойчивость модели путем обучения и тестирования на разных подмножествах данных. Это помогает убедиться в том, что результаты не зависят от конкретного разбиения данных на обучающий и тестовый наборы.

Оценка влияния гиперпараметров:

Grid Search и Random Search: Помогают определить оптимальные гиперпараметры модели, что в свою очередь влияет на её способность к обобщению.



Синяя линия (Training Loss): Показывает потери на обучающем наборе. Значительные колебания отражают изменения в потерях между эпохами, что может указывать на высокую чувствительность модели к данным или настройкам обучения.

Красная линия (Validation Loss): Отображает потери на валидационном наборе. Большие прыжки в значениях могут сигнализировать о проблемах в обобщающей способности модели, потенциальном переобучении или неподходящем выборе гиперпараметров.

# **3.5. Выводы по разделу**

В разделе 3 мы рассмотрели ключевые аспекты обучения нейронных сетей, начиная от подготовки данных и заканчивая анализом адекватности обучения. Основные моменты, которые были охвачены, включают:

3.1 Создание нейронов

В ходе разработки нейронной сети первоначальным шагом было создание нейронов, которые являются фундаментальными строительными блоками любой нейронной сети. Применение различных типов функций активации, таких как ReLU и LeakyReLU, позволило улучшить процесс обучения, снижая вероятность возникновения проблем, связанных с исчезающими градиентами. Важным аспектом стала интеграция этих нейронов в большие функциональные блоки, что является ключом к построению эффективных архитектур для конкретных задач.

3.2 Создание нейронной сети

Разработка архитектуры нейронной сети была нацелена на создание эффективной модели для генерации 2D ассетов. Выбор топологии сети, включая количество и типы слоёв, был критически важен для достижения оптимальных результатов. Использование современных архитектур, таких как GAN (генеративно-состязательные сети) и автокодировщики, позволило решить задачу с высокой степенью детализации и качества генерируемых изображений. Экспериментальный подбор характеристик и настройка гиперпараметров способствовали максимальной адаптации модели под специфические требования и условия задачи.

3.2.1 Выбор топологии сети

Осознанный подход к выбору топологии сети позволил значительно увеличить эффективность обучения и качество генерации ассетов. Определение структуры слоёв, их взаимосвязей и последовательности активаций играли ключевую роль в достижении желаемой функциональности и производительности нейронной сети. Корректная организация нейронных блоков и слоёв обеспечила способность сети адекватно реагировать на сложные задачи генерации, обучения и обработки данных.

3.2.2 Группировка нейронных блоков

Организация и соединение нейронных блоков влияет на эффективность обучения и способность нейронной сети к обобщению. Примеры таких блоков включают свёрточные слои, полносвязные слои и блоки рекуррентных сетей. Примеры архитектур, такие как ResNet и Inception, показывают, как сложные соединения блоков могут улучшить производительность сети в задачах распознавания образов.

3.2.3 Экспериментальный подбор характеристик сети

Выбор и настройка гиперпараметров сети через методы, такие как сеточный поиск и случайный поиск, критически важны для оптимизации процесса обучения и достижения наилучших результатов.

3.3 Сбор и нормализация данных

Качественная подготовка данных является первым и важнейшим шагом в обучении нейронных сетей. Это включает в себя сбор данных, их очистку, трансформацию и нормализацию для обеспечения унифицированного входа для моделей.

3.4 Обучение нейронной сети

Процесс обучения охватывает настройку архитектуры сети, выбор функции потерь и оптимизатора, а также методы минимизации ошибок и переобучения через регуляризацию и кросс-валидацию.

3.4.1 Выбор способа оценки адекватности результатов обучения

Оценка производительности моделей с помощью различных метрик и методов, таких как кривые обучения, матрица ошибок и перекрёстная проверка, помогает понять, насколько хорошо модель работает на практике.

3.4.2 Расчёт количества потерь и их минимизация

Минимизация функции потерь через градиентный спуск и его варианты является ключевым элементом в обучении нейронных сетей, позволяющим улучшать их точность и обобщающую способность.

3.4.3 Анализ адекватности обучения

Анализ адекватности обучения включает диагностику и корректировку процесса обучения на основе данных о производительности на обучающих и валидационных наборах, что предотвращает переобучение и повышает обобщающую способность моделей.

Каждый из этих этапов критически важен для разработки эффективных нейронных сетей, способных решать сложные задачи в различных областях, от автоматического вождения до распознавания речи и обработки естественного языка.

Выводы по разделу 2

|  |  |  |
| --- | --- | --- |
| Выводы | Сформированные компетенции | |
| Код компетенции | Содержание компетенции |
| Разработаны нейроны и нейронные блоки на языке программирования Python.  Создана нейронная сеть. Экспериментальным способом подобраны характеристики нейронной сети: определено количество слоев, число блоков в скрытых слоях, наличие или отсутствие обходных соединений, передаточные функции нейронов.  Собраны данные для обучения нейронной сети. Параметры обучения подобраны экспериментальным способом.  Произведено обучение нейронной сети на собранных данных. Проверена адекватность результатов обучения сети. | ПК-9 | Способность разрабатывать методы извлечения, анализа и обработки информации |
| ПК-10 | Способность применять математические методы моделирования процессов обработки информации с использованием средств интеллектуального анализа данных и машинного обучения |

# **Заключение**

В рамках производственной практики были выполнены комплексные задачи по созданию и обучению нейросети-генератора 2D ассетов персонажей. Работа подтвердила актуальность и эффективность использования современных методов искусственного интеллекта в разработке игровой графики, что особенно важно в условиях растущего спроса инди-разработчиков на уникальные визуальные решения.

Основные этапы работы включали в себя анализ теоретических основ создания нейросетей, подготовку и нормализацию данных, разработку нейросетевой архитектуры, а также обучение и оценку моделей. Эксперименты по генерации 2D персонажей подтвердили, что даже при ограниченном наборе данных можно достичь значительного качества и разнообразия генерируемых изображений, благодаря грамотному выбору архитектуры и параметров нейросети.

Среди ключевых достижений практики стоит выделить:

Разработка и внедрение эффективных нейросетевых архитектур (GAN и Autoencoder), позволяющих создавать высококачественные изображения.

Оптимизация процесса обучения через экспериментальный подбор характеристик, использование современных методов минимизации потерь и анализ адекватности результатов обучения.

Практическое применение моделей для генерации ассетов, что демонстрирует реальную прикладную ценность разработанных решений.

Практика позволила не только углубить теоретические знания в области машинного обучения и нейросетей, но и приобрести ценный опыт их практического применения. Результаты работы могут быть использованы для дальнейшего развития проектов в области генерации игровой графики, а также для обучения специалистов в данной области.

В заключение, выполненная работа подтверждает, что нейросетевые технологии открывают новые возможности для индустрии разработки игр и могут значительно сократить время и усилия, необходимые для создания качественного визуального контента.

# **Список используемых источников и литературы**

1. Radford, A., Metz, L., & Chintala, S. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." Deep Convolutional Generative Adversarial Networks. Accessed June 1, 2024. <https://main--pytorch-tutorials-preview.netlify.app/beginner/dcgan_faces_tutorial.html.>
2. NIX\_Solutions. "Введение в генеративно-состязательные сети (GAN)." Habr. December 15, 2020. [https://habr.com/ru/companies/nix/articles/416129/.](https://habr.com/ru/companies/nix/articles/416129/)
3. Dutta, Indira. "Deep Convolutional Generative Adversarial Networks." GitHub. Last modified March 10, 2021. <https://github.com/indiradutta/DC_GAN.>
4. Makrohn. "Universal LPC Spritesheet." GitHub. Last modified March 5, 2021. <https://github.com/makrohn/Universal-LPC-spritesheet/tree/7040e2fe85d2cb1e8154ec5fce382589d369bdb8.>
5. Mikołajczyk, Aga. "Pixel Character Generator." GitHub. Accessed June 1, 2024. <https://github.com/AgaMiko/pixel_character_generator.>
6. Google Developers. "Generative Adversarial Networks (GANs) - Loss Functions." Accessed June 1, 2024. <https://developers.google.com/machine-learning/gan/loss?hl=ru.>
7. Brownlee, Jason. "A Gentle Introduction to GAN Failure Modes." Machine Learning Mastery. Accessed June 1, 2024. <https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/>
8. Brsoff. "Deep Convolutional Generative Adversarial Networks Tutorial." Accessed June 1, 2024. <https://brsoff.github.io/tutorials/beginner/dcgan_faces_tutorial.html>.

# **Приложение**

1\_extracter.py

import os

import zipfile

import requests

from bs4 import BeautifulSoup

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.common.by import By

import shutil

import glob

import random

import time

from datetime import datetime

# Установим seed для воспроизводимости

random.seed(datetime.now().timestamp())

# Параметры

base\_dir = "F:\\git\\Practic"

zip\_path = os.path.join(base\_dir, "data.zip")

extract\_path = os.path.join(base\_dir, "extracted")

download\_dir = os.path.join(base\_dir, "downloads")

save\_dir = os.path.join(base\_dir, "rpg\_char")

chrome\_driver\_path = 'F:\\git\\Practic\\chromedriver.exe'

chrome\_binary\_path = "C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe"

# Создание необходимых директорий

os.makedirs(extract\_path, exist\_ok=True)

os.makedirs(download\_dir, exist\_ok=True)

os.makedirs(save\_dir, exist\_ok=True)

# Извлечение содержимого data.zip

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_path)

# Проверка содержимого извлеченной директории

extracted\_files = os.listdir(extract\_path)

print("Extracted files:", extracted\_files)

# Альтернативный URL для скачивания спрайтов

BASE\_URL = "https://sanderfrenken.github.io/Universal-LPC-Spritesheet-Character-Generator/#?"

# Функция для получения всех опций с веб-страницы

def get\_all\_options(base\_url):

try:

page = requests.get(base\_url)

soup = BeautifulSoup(page.content, 'html.parser')

all\_items = []

names = []

labels = soup.find\_all('li')

for label in labels:

name = str(label).split('name="', 1)

if len(name) > 1:

name = name[1]

option = str(label.find\_all('label')).split('-', 1)

if len(option) > 1:

option = option[1]

name\_opt = str(name).split('"', 1)[0] + '#' + str(option).split('"', 1)[0]

names.append(str(name).split('"', 1)[0])

all\_items.append(name\_opt)

all\_items = list(set(all\_items))

names = list(set(names))

prefixes = ('hair-', 'hairs', '[', "'")

forbidden = '<'

for word in names[:]:

if word.startswith(prefixes):

names.remove(word)

for word in all\_items[:]:

if word.startswith(prefixes) or forbidden in word:

all\_items.remove(word)

categories = names

only\_male = []

only\_female = []

for label in labels:

if len(str(label).split('data-required="sex=', 1)) > 1:

sex = str(label).split('data-required="sex=', 1)[1].split('" id=', 1)[0]

equipment = str(label.find\_all('label')).split('-', 1)

if len(equipment) > 1:

equipment = equipment[1]

if sex == 'male':

only\_male.append(str(equipment).split('"', 1)[0])

elif sex == 'female':

only\_female.append(str(equipment).split('"', 1)[0])

only\_male = [i for i in only\_male if i not in only\_female]

only\_female = [i for i in only\_female if i not in only\_male]

return categories, all\_items, only\_male, only\_female

except requests.exceptions.RequestException as e:

print(f"Error fetching options from URL: {e}")

return [], [], [], []

# Функция для генерации URL с случайными параметрами

def generate\_random\_url(base\_url, categories, all\_items, only\_male, only\_female):

def rand\_item(item, options):

result = [i for i in options if i.startswith(item)]

if len(result) > 1:

selected\_option = random.randint(0, len(result) - 1)

else:

selected\_option = 0

equipment = result[selected\_option].split('#', 1)

if len(equipment) > 1:

equipment = equipment[1]

return equipment

def generate\_URL(base\_URL, item, option):

base\_URL += str(item + "=" + str(option) + "&")

return base\_URL

def rand\_URL(base\_URL, minimum\_equip=5, max\_equip=15, must\_equip=['body', 'head', 'sex'], categories=categories, all\_items=all\_items):

rand\_len = random.randint(minimum\_equip, max\_equip)

rand\_categories = random.sample(range(0, len(categories)), rand\_len)

for eq in must\_equip:

if categories.index(eq) not in rand\_categories:

rand\_categories.insert(0, categories.index(eq))

else:

rand\_categories.insert(0, rand\_categories.pop(rand\_categories.index(categories.index(eq))))

sex = rand\_item('sex', all\_items)

equip\_URL = str(base\_URL) + 'sex=' + str(sex) + '&'

for category in rand\_categories[1:]:

selected\_option = rand\_item(categories[category], all\_items)

equip\_URL = generate\_URL(equip\_URL, categories[category], selected\_option)

return equip\_URL

return rand\_URL(base\_url, minimum\_equip=7, max\_equip=20, must\_equip=['body', 'head', 'sex'], categories=categories, all\_items=all\_items)

# Получение всех опций

categories, all\_items, only\_male, only\_female = get\_all\_options(BASE\_URL)

# Запуск Selenium для скачивания спрайтов

chrome\_options = Options()

chrome\_options.binary\_location = chrome\_binary\_path

chrome\_options.add\_experimental\_option("prefs", {

"download.default\_directory": download\_dir,

"download.prompt\_for\_download": False,

"download.directory\_upgrade": True,

"safebrowsing.enabled": True

})

service = Service(chrome\_driver\_path)

driver = webdriver.Chrome(service=service, options=chrome\_options)

for i in range(8000, 10000):

URL = generate\_random\_url(BASE\_URL, categories, all\_items, only\_male, only\_female)

print(URL)

driver.get(URL)

time.sleep(10)

button = driver.find\_element(By.XPATH, '//\*[@id="saveAsPNG"]')

button.click()

time.sleep(10)

list\_of\_files = glob.glob(os.path.join(download\_dir, '\*'))

if list\_of\_files:

file = max(list\_of\_files, key=os.path.getctime)

if os.path.exists(file) and not os.path.isdir(file) and not os.path.islink(file):

if os.path.getsize(file) < 130 \* 1024:

os.remove(file)

print("Deleted file...")

else:

shutil.move(file, os.path.join(save\_dir, str(i) + '.png'))

print("Moved file")

else:

print("No files downloaded")

driver.quit()

2\_sprigan.py

import os

from datetime import datetime

import random

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

# Устанавливаем случайное семя для воспроизводимости

random.seed(datetime.now().timestamp())

# Путь к папке с исходными спрайтшитами

DIR = 'F:\\git\\Practic\\rpg\_char\\characters'

spritesheets = []

for r, d, f in os.walk(DIR):

for file in f:

if file.endswith(".png"):

spritesheets.append(os.path.join(r, file))

start\_h = 260

start\_w = 0

char\_h = 64

char\_w = 64

no\_chars = 4

# Извлечение персонажей

for i, spritesheet in enumerate(spritesheets):

for nr in range(no\_chars):

image = mpimg.imread(spritesheet)

resized\_im = image[start\_h + char\_h \* nr:start\_h + char\_h \* (nr + 1), start\_w:start\_w + char\_w, :]

path = 'F:\\git\\Practic\\data\\characters\\'

if not os.path.exists(path):

os.makedirs(path)

plt.imsave(path + str(i) + '\_' + str(nr) + '.png', resized\_im)

print("Извлечение персонажей завершено.")

3\_DCGANNER.py

import os

import random

import time

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision.transforms as transforms

import torchvision.utils as vutils

from PIL import Image

from torch.utils.data import Dataset, DataLoader

import numpy as np

import matplotlib.pyplot as plt

# Установим seed для воспроизводимости

manualSeed = 999

random.seed(manualSeed)

torch.manual\_seed(manualSeed)

# Параметры обучения

dataroot = "F:\\git\\Practic\\data\\characters"

workers = 2

batch\_size = 128

image\_size = 64

nc = 3 # Количество каналов в изображениях (RGB)

nz = 100 # Размер вектора шума (вход в генератор)

ngf = 64 # Размер признаковых карт в генераторе

ndf = 64 # Размер признаковых карт в дискриминаторе

num\_epochs = 50

lr = 0.0001

beta1 = 0.5

ngpu = 1 # Количество GPU (0 для использования CPU)

# Кастомный датасет для загрузки изображений из одной папки

class CustomDataset(Dataset):

def \_\_init\_\_(self, root\_dir, transform=None):

self.root\_dir = root\_dir

self.transform = transform

self.image\_files = [f for f in os.listdir(root\_dir) if os.path.isfile(os.path.join(root\_dir, f))]

def \_\_len\_\_(self):

return len(self.image\_files)

def \_\_getitem\_\_(self, idx):

img\_name = os.path.join(self.root\_dir, self.image\_files[idx])

image = Image.open(img\_name).convert('RGB') # Преобразование изображения в RGB

if self.transform:

image = self.transform(image)

return image

# Создание датасета и DataLoader

dataset = CustomDataset(root\_dir=dataroot,

transform=transforms.Compose([

transforms.Resize(image\_size),

transforms.CenterCrop(image\_size),

transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),

]))

dataloader = DataLoader(dataset, batch\_size=batch\_size, shuffle=True, num\_workers=workers)

# Устройство (GPU или CPU)

device = torch.device("cuda:0" if (torch.cuda.is\_available() and ngpu > 0) else "cpu")

# Создание генератора

class Generator(nn.Module):

def \_\_init\_\_(self, ngpu):

super(Generator, self).\_\_init\_\_()

self.ngpu = ngpu

self.main = nn.Sequential(

nn.ConvTranspose2d(nz, ngf \* 8, 4, 1, 0, bias=False),

nn.BatchNorm2d(ngf \* 8),

nn.ReLU(True),

nn.ConvTranspose2d(ngf \* 8, ngf \* 4, 4, 2, 1, bias=False),

nn.BatchNorm2d(ngf \* 4),

nn.ReLU(True),

nn.ConvTranspose2d(ngf \* 4, ngf \* 2, 4, 2, 1, bias=False),

nn.BatchNorm2d(ngf \* 2),

nn.ReLU(True),

nn.ConvTranspose2d(ngf \* 2, ngf, 4, 2, 1, bias=False),

nn.BatchNorm2d(ngf),

nn.ReLU(True),

nn.ConvTranspose2d(ngf, nc, 4, 2, 1, bias=False),

nn.Tanh()

)

def forward(self, input):

return self.main(input)

# Создание дискриминатора

class Discriminator(nn.Module):

def \_\_init\_\_(self, ngpu):

super(Discriminator, self).\_\_init\_\_()

self.ngpu = ngpu

self.main = nn.Sequential(

nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),

nn.LeakyReLU(0.2, inplace=True),

nn.Conv2d(ndf, ndf \* 2, 4, 2, 1, bias=False),

nn.BatchNorm2d(ndf \* 2),

nn.LeakyReLU(0.2, inplace=True),

nn.Conv2d(ndf \* 2, ndf \* 4, 4, 2, 1, bias=False),

nn.BatchNorm2d(ndf \* 4),

nn.LeakyReLU(0.2, inplace=True),

nn.Conv2d(ndf \* 4, ndf \* 8, 4, 2, 1, bias=False),

nn.BatchNorm2d(ndf \* 8),

nn.LeakyReLU(0.2, inplace=True),

nn.Conv2d(ndf \* 8, 1, 4, 1, 0, bias=False),

nn.Sigmoid()

)

def forward(self, input):

return self.main(input)

# Инициализация весов

def weights\_init(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)

# Создание экземпляров генератора и дискриминатора

netG = Generator(ngpu).to(device)

netG.apply(weights\_init)

print(netG)

netD = Discriminator(ngpu).to(device)

netD.apply(weights\_init)

print(netD)

# Определение функции потерь и оптимизаторов

criterion = nn.BCELoss()

fixed\_noise = torch.randn(64, nz, 1, 1, device=device)

real\_label = 1.

fake\_label = 0.

optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))

optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.999))

# Списки для отслеживания прогресса

img\_list = []

G\_losses = []

D\_losses = []

iters = 0

print("Starting Training Loop...")

if \_\_name\_\_ == '\_\_main\_\_':

start\_time = time.time()

for epoch in range(num\_epochs):

for i, data in enumerate(dataloader, 0):

# Обновление дискриминатора с реальными данными

netD.zero\_grad()

real\_cpu = data.to(device)

b\_size = real\_cpu.size(0)

label = torch.full((b\_size,), real\_label, dtype=torch.float, device=device)

output = netD(real\_cpu).view(-1)

errD\_real = criterion(output, label)

errD\_real.backward()

D\_x = output.mean().item()

# Обновление дискриминатора с фейковыми данными

noise = torch.randn(b\_size, nz, 1, 1, device=device)

fake = netG(noise)

label.fill\_(fake\_label)

output = netD(fake.detach()).view(-1)

errD\_fake = criterion(output, label)

errD\_fake.backward()

D\_G\_z1 = output.mean().item()

errD = errD\_real + errD\_fake

optimizerD.step()

# Обновление генератора

netG.zero\_grad()

label.fill\_(real\_label)

output = netD(fake).view(-1)

errG = criterion(output, label)

errG.backward()

D\_G\_z2 = output.mean().item()

optimizerG.step()

# Сохранение потерь для последующего графика

G\_losses.append(errG.item())

D\_losses.append(errD.item())

# Проверка прогресса генератора

if (iters % 500 == 0) or ((epoch == num\_epochs-1) and (i == len(dataloader)-1)):

with torch.no\_grad():

fake = netG(fixed\_noise).detach().cpu()

img\_list.append(vutils.make\_grid(fake, padding=2, normalize=True))

iters += 1

# Выводим прогресс каждые 2000 итераций

if iters % 2000 == 0:

elapsed\_time = time.time() - start\_time

print(f'Итерация {iters}, время: {elapsed\_time:.2f} сек')

elapsed\_time = time.time() - start\_time

print(f'Обучение завершено за {elapsed\_time:.2f} секунд.')

# Построение графиков потерь

plt.figure(figsize=(10,5))

plt.title("Generator and Discriminator Loss During Training")

plt.plot(G\_losses,label="G")

plt.plot(D\_losses,label="D")

plt.xlabel("iterations")

plt.ylabel("Loss")

plt.legend()

plt.show()

# Сохранение модели

torch.save(netG.state\_dict(), 'generator.pth')

torch.save(netD.state\_dict(), 'discriminator.pth')

# Генерация и визуализация результатов

real\_batch = next(iter(dataloader))

plt.figure(figsize=(15,15))

plt.subplot(1,2,1)

plt.axis("off")

plt.title("Real Images")

plt.imshow(np.transpose(vutils.make\_grid(real\_batch.to(device)[:64], padding=5, normalize=True).cpu(),(1,2,0)))

plt.subplot(1,2,2)

plt.axis("off")

plt.title("Fake Images")

plt.imshow(np.transpose(img\_list[-1],(1,2,0)))

plt.show()

4\_autocod.py

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision.transforms as transforms

import torchvision.utils as vutils

from torch.utils.data import DataLoader, Dataset

from PIL import Image

import matplotlib.pyplot as plt

import os

# Параметры обучения

dataroot = "F:\\git\\Practic\\data\\characters"

batch\_size = 128

image\_size = 64

nc = 3 # Количество каналов в изображениях (RGB)

latent\_size = 40 # Размер вектора латентного пространства

feature\_map\_size = 64

num\_epochs = 100

lr = 0.0001

beta1 = 0.5

ngpu = 1 # Количество GPU (0 для использования CPU)

# Устройство (GPU или CPU)

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

# Кастомный датасет для загрузки изображений из одной папки

class CustomDataset(Dataset):

def \_\_init\_\_(self, root\_dir, transform=None):

self.root\_dir = root\_dir

self.transform = transform

self.image\_files = [f for f in os.listdir(root\_dir) if os.path.isfile(os.path.join(root\_dir, f))]

def \_\_len\_\_(self):

return len(self.image\_files)

def \_\_getitem\_\_(self, idx):

img\_name = os.path.join(self.root\_dir, self.image\_files[idx])

image = Image.open(img\_name).convert('RGB') # Преобразование изображения в RGB

if self.transform:

image = self.transform(image)

return image

# Создание датасета и DataLoader

dataset = CustomDataset(root\_dir=dataroot,

transform=transforms.Compose([

transforms.Resize(image\_size),

transforms.CenterCrop(image\_size),

transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),

]))

dataloader = DataLoader(dataset, batch\_size=batch\_size, shuffle=True)

# Определение архитектуры автокодировщика

class Encoder(nn.Module):

def \_\_init\_\_(self, ngpu):

super(Encoder, self).\_\_init\_\_()

self.ngpu = ngpu

self.main = nn.Sequential(

nn.Conv2d(nc, feature\_map\_size, 4, 2, 1, bias=False),

nn.LeakyReLU(0.2),

nn.Conv2d(feature\_map\_size, feature\_map\_size \* 2, 4, 2, 1, bias=False),

nn.BatchNorm2d(feature\_map\_size \* 2),

nn.LeakyReLU(0.2),

nn.Conv2d(feature\_map\_size \* 2, feature\_map\_size \* 4, 4, 2, 1, bias=False),

nn.BatchNorm2d(feature\_map\_size \* 4),

nn.LeakyReLU(0.2),

nn.Conv2d(feature\_map\_size \* 4, feature\_map\_size \* 8, 4, 2, 1, bias=False),

nn.BatchNorm2d(feature\_map\_size \* 8),

nn.LeakyReLU(0.2),

nn.Conv2d(feature\_map\_size \* 8, feature\_map\_size, 4, 1, 0, bias=False),

nn.BatchNorm2d(feature\_map\_size),

nn.LeakyReLU(0.2),

)

self.fc = nn.Sequential(

nn.Linear(feature\_map\_size \* 1 \* 1, latent\_size),

nn.ReLU(True)

)

def forward(self, input):

x = self.main(input)

x = x.view(x.size(0), -1)

x = self.fc(x)

return x

class Decoder(nn.Module):

def \_\_init\_\_(self, ngpu):

super(Decoder, self).\_\_init\_\_()

self.ngpu = ngpu

self.main = nn.Sequential(

nn.ConvTranspose2d(latent\_size, feature\_map\_size \* 8, 4, 1, 0, bias=False),

nn.BatchNorm2d(feature\_map\_size \* 8),

nn.ReLU(True),

nn.ConvTranspose2d(feature\_map\_size \* 8, feature\_map\_size \* 4, 4, 2, 1, bias=False),

nn.BatchNorm2d(feature\_map\_size \* 4),

nn.ReLU(True),

nn.ConvTranspose2d(feature\_map\_size \* 4, feature\_map\_size \* 2, 4, 2, 1, bias=False),

nn.BatchNorm2d(feature\_map\_size \* 2),

nn.ReLU(True),

nn.ConvTranspose2d(feature\_map\_size \* 2, feature\_map\_size, 4, 2, 1, bias=False),

nn.BatchNorm2d(feature\_map\_size),

nn.ReLU(True),

nn.ConvTranspose2d(feature\_map\_size, nc, 4, 2, 1, bias=False),

nn.Tanh()

)

def forward(self, input):

input = input.view(input.size(0), input.size(1), 1, 1)

output = self.main(input)

return output

class Autoencoder(nn.Module):

def \_\_init\_\_(self, ngpu):

super(Autoencoder, self).\_\_init\_\_()

self.encoder = Encoder(ngpu)

self.decoder = Decoder(ngpu)

def forward(self, x):

x = self.encoder(x)

x = self.decoder(x)

return x

# Инициализация автокодировщика и установка функции потерь и оптимизатора

autoencoder = Autoencoder(ngpu).to(device)

criterion = nn.MSELoss()

optimizer = optim.Adam(autoencoder.parameters(), lr=lr, betas=(beta1, 0.999))

# Обучение автокодировщика

for epoch in range(num\_epochs):

for i, data in enumerate(dataloader, 0):

autoencoder.zero\_grad()

real\_images = data.to(device)

output = autoencoder(real\_images)

loss = criterion(output, real\_images)

loss.backward()

optimizer.step()

if i % 2000 == 0:

print(f"Epoch [{epoch}/{num\_epochs}] Step [{i}/{len(dataloader)}] Loss: {loss.item()}")

# Сохранение модели автокодировщика

torch.save(autoencoder.state\_dict(), 'autoencoder.pth')

5\_DCGAC.py

import torch

import torch.nn as nn

import torchvision.utils as vutils

import matplotlib.pyplot as plt

import numpy as np

# Определение архитектуры генератора

class Generator(nn.Module):

def \_\_init\_\_(self, ngpu):

super(Generator, self).\_\_init\_\_()

self.ngpu = ngpu

self.main = nn.Sequential(

nn.ConvTranspose2d(100, 64 \* 8, 4, 1, 0, bias=False),

nn.BatchNorm2d(64 \* 8),

nn.ReLU(True),

nn.ConvTranspose2d(64 \* 8, 64 \* 4, 4, 2, 1, bias=False),

nn.BatchNorm2d(64 \* 4),

nn.ReLU(True),

nn.ConvTranspose2d(64 \* 4, 64 \* 2, 4, 2, 1, bias=False),

nn.BatchNorm2d(64 \* 2),

nn.ReLU(True),

nn.ConvTranspose2d(64 \* 2, 64, 4, 2, 1, bias=False),

nn.BatchNorm2d(64),

nn.ReLU(True),

nn.ConvTranspose2d(64, 3, 4, 2, 1, bias=False),

nn.Tanh()

)

def forward(self, input):

return self.main(input)

# Устройство (GPU или CPU)

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

# Загрузка обученной модели генератора

netG = Generator(ngpu=1).to(device)

netG.load\_state\_dict(torch.load('generator.pth'))

netG.eval()

# Генерация изображений

nz = 100 # Размер вектора шума

batch\_size = 64 # Количество изображений для генерации

fixed\_noise = torch.randn(batch\_size, nz, 1, 1, device=device)

with torch.no\_grad():

fake\_images = netG(fixed\_noise).detach().cpu()

# Отображение сгенерированных изображений

plt.figure(figsize=(8, 8))

plt.axis("off")

plt.title("Generated Images")

plt.imshow(np.transpose(vutils.make\_grid(fake\_images, padding=2, normalize=True), (1, 2, 0)))

plt.show()

# Определение архитектуры автокодировщика

class Encoder(nn.Module):

def \_\_init\_\_(self, ngpu):

super(Encoder, self).\_\_init\_\_()

self.ngpu = ngpu

self.main = nn.Sequential(

nn.Conv2d(3, 64, 4, 2, 1, bias=False),

nn.LeakyReLU(0.2),

nn.Conv2d(64, 128, 4, 2, 1, bias=False),

nn.BatchNorm2d(128),

nn.LeakyReLU(0.2),

nn.Conv2d(128, 256, 4, 2, 1, bias=False),

nn.BatchNorm2d(256),

nn.LeakyReLU(0.2),

nn.Conv2d(256, 512, 4, 2, 1, bias=False),

nn.BatchNorm2d(512),

nn.LeakyReLU(0.2),

nn.Conv2d(512, 64, 4, 1, 0, bias=False),

nn.BatchNorm2d(64),

nn.LeakyReLU(0.2),

)

self.fc = nn.Sequential(

nn.Linear(64 \* 1 \* 1, 40),

nn.ReLU(True)

)

def forward(self, input):

x = self.main(input)

x = x.view(x.size(0), -1)

x = self.fc(x)

return x

class Decoder(nn.Module):

def \_\_init\_\_(self, ngpu):

super(Decoder, self).\_\_init\_\_()

self.ngpu = ngpu

self.main = nn.Sequential(

nn.ConvTranspose2d(40, 512, 4, 1, 0, bias=False),

nn.BatchNorm2d(512),

nn.ReLU(True),

nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),

nn.BatchNorm2d(256),

nn.ReLU(True),

nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),

nn.BatchNorm2d(128),

nn.ReLU(True),

nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),

nn.BatchNorm2d(64),

nn.ReLU(True),

nn.ConvTranspose2d(64, 3, 4, 2, 1, bias=False),

nn.Tanh()

)

def forward(self, input):

input = input.view(input.size(0), input.size(1), 1, 1)

output = self.main(input)

return output

class Autoencoder(nn.Module):

def \_\_init\_\_(self, ngpu):

super(Autoencoder, self).\_\_init\_\_()

self.encoder = Encoder(ngpu)

self.decoder = Decoder(ngpu)

def forward(self, x):

x = self.encoder(x)

x = self.decoder(x)

return x

# Загрузка обученной модели автокодировщика

autoencoder = Autoencoder(ngpu=1).to(device)

autoencoder.load\_state\_dict(torch.load('autoencoder.pth'))

autoencoder.eval()

# Восстановление изображений

with torch.no\_grad():

reconstructed\_images = autoencoder(fake\_images.to(device)).detach().cpu()

# Отображение восстановленных изображений

plt.figure(figsize=(8, 8))

plt.axis("off")

plt.title("Reconstructed Images")

plt.imshow(np.transpose(vutils.make\_grid(reconstructed\_images, padding=2, normalize=True), (1, 2, 0)))

plt.show()