МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное   
учреждение высшего образования

**«Южно-Уральский государственный университет**

**(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

**Генерация скинов персонажей с помощью генеративно-состязательных нейросетей для видеоигры Minecraft**

КУРСОВАЯ РАБОТА

по дисциплине «Программная инженерия»

ЮУрГУ – 09.03.04.20231.308-3642.КР

|  |  |
| --- | --- |
| Нормоконтролер3,  Доцент кафедры СП  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.В. Сухов “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2023 г. | Научный руководитель:  Доцент кафедры СП  \_\_\_\_\_\_\_\_\_\_ М.В. Сухов  Автор работы:  студент группы КЭ-333  \_\_\_\_\_\_\_\_\_\_ Э.Э. Сагатдинов  Работа защищена  с оценкой: \_\_\_\_\_\_\_\_\_\_\_  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2023 г. |

Челябинск 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Южно-Уральский государственный университет**

**(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

УТВЕРЖДАЮ

Зав. кафедрой СП

\_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский

10.02.2023

**ЗАДАНИЕ**

**на выполнение курсовой работы**

по дисциплине «Программная инженерия»

студенту группы КЭ-333 Сагатдинову Эмилю Эдуардовичу,

обучающемуся по направлению 09.03.04 «Программная инженерия»

1. **Тема работы**Генерация скинов персонажей с помощью генеративно-состязательных нейросетей для видеоигры Minecraft.
2. **Срок сдачи студентом законченной работы:** 31.05.2023 г.
3. **Исходные данные к работе**
4. Айрапетов А.Э., Коваленко А.А. ИССЛЕДОВАНИЕ ГЕНЕРАТИВНО-СОСТЯЗАТЕЛЬНОЙ СЕТИ // Политехнический молодежный журнал. 2018 г. №10. URL: http://ptsj.ru/articles/380/eng/380.pdf
5. Салахутдинов Э.Р., Сиякина В.В. ПРИМЕНЕНИЕ ГЕНЕРАТИВНО-СОСТЯЗАТЕЛЬНЫХ НЕЙРОННЫХ СЕТЕЙ // MODERN SCIENCE 2019 г. №6-1. URL: https://www.elibrary.ru/item.asp?id=38236537
6. Шкиря А.С. РАЗРАБОТКА ГЕНЕРАТИВНО-СОСТЯЗАТЕЛЬНОЙ СЕТИ
7. ДЛЯ СОЗДАНИЯ ИЗОБРАЖЕНИЙ // Электронный сборник статей по материалам XCII студенческой международной научно-практической конференции 2020 г. №9 URL: https://sibac.info/archive/meghdis/9(92).pdf#page=38
8. **Перечень подлежащих разработке вопросов**
9. Рассмотреть и описать существующие методы создания изображений с помощью искусственных нейросетей;
10. Проанализировать и рассмотреть архитектуру генеративно-состязательных нейросетей;
11. Спроектировать и реализовать нейронную сеть генерирующую внешний вид персонажей.
12. **Дата выдачи задания:** 9 февраля 2023 г.

Научный руководитель М.В. Сухов

Доцент кафедры СП, к.т.н.

Задание принял к исполнению Э.Э. Сагатдинов

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 5](#_Toc135417068)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc135417069)

[1.1. Анализ аналогичных проектов 6](#_Toc135417070)

[1.2. Анализ существующих решений для реализации проекта 6](#_Toc135417071)

[1.3. Выводы из результатов анализа предметной области 9](#_Toc135417072)

[2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 10](#_Toc135417073)

[2.1. Генеративно-состязательные нейросети 10](#_Toc135417074)

[2.1. Описание принципа работы GAN 10](#_Toc135417075)

[2.2. Описание свёрточной нейронной сети 11](#_Toc135417076)

[2.3. Описание механизма внимания 11](#_Toc135417077)

[3. ПРОЕКТИРОВАНИЕ 13](#_Toc135417078)

[3.1. Определение требований 13](#_Toc135417079)

[3.2. Варианты использования системы 13](#_Toc135417080)

[4. АРХИТЕКТУРА СИСТЕМЫ 18](#_Toc135417081)

[4.1. Общее описание архитектуры системы 18](#_Toc135417082)

[4.2. Описание компонентов, составляющих систему 18](#_Toc135417083)

[4.2.1. Описание класса Trainer 19](#_Toc135417084)

[4.2.2. Описание класса Discriminator 20](#_Toc135417085)

[4.2.3. Описание класса Generator 20](#_Toc135417086)

[4.3. Топология нейронной сети 21](#_Toc135417087)

[4.3.1. Топология дискриминатора 21](#_Toc135417088)

[4.3.2. Топология генератора 22](#_Toc135417089)

[ЛИТЕРАТУРА 25](#_Toc135417090)

# ВВЕДЕНИЕ

**Актуальность**

TODO

**Постановка задачи**

Целью курсовой работы заключается в проектировании и реализации нейронной сети, генерирующей внешний вид игровых персонажей или скин. Скины представляют собой четырёхканальные изображения в формате .png.

Для выполнения данной задачи необходимо выполнить следующие подзадачи:

1. Требуется найти набор данных для обучения нейронной сети;
2. Требуется подготовить датасет для обучения нейронной сети;
3. Спроектировать нейронную сеть и обучить её.

**Структура и содержание работы**

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 50 страниц, объем списка литературы – 21 источник.

В первой главе описывается …

Вторая глава посвящена …

В третьей главе …

В приложении А содержится …

Глоссарий является необязательной частью текста работы.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В данной главе представлен анализ предметной области, который включает в себя, анализ аналогичных проектов, используемых в них технологий, а также анализ существующих решений для реализации проекта, где были рассмотрены следующие архитектуры: DCGAN, SAGAN и PGGAN. Также были описаны достоинства и недостатки каждой из архитектур.

1.1. Анализ аналогичных проектов

Задачей создания изображений с помощью генеративно-состязательных нейросетей занимались многие исследователи, применяя ее к различным областям: генерация изображений [8], генерация аудио [1] и выделение аудиодорожек [6].

Концепция, архитектура и результаты сравнения разных алгоритмов наиболее полно рассмотрены в работе [7].

Как правило, в работах применяется следующий стек технологий: язык Python и фреймворки для работы с нейронными сетями (PyTorch или TensorFlow).

1.2. Анализ существующих решений для реализации проекта

В работе [9] была разработана GAN для генерации рукописных чисел. В работе применялся датасет MNIST, содержащий одноканальные изображения размера 28х28. В этой работе применялась достаточно примитивная модель (7 слоев генератора и 5 слоев дискриминатора). В результате получена средняя точность генерации цифры 65.5%. Изображения сгенерированных цифр представлены на рисунке 1.

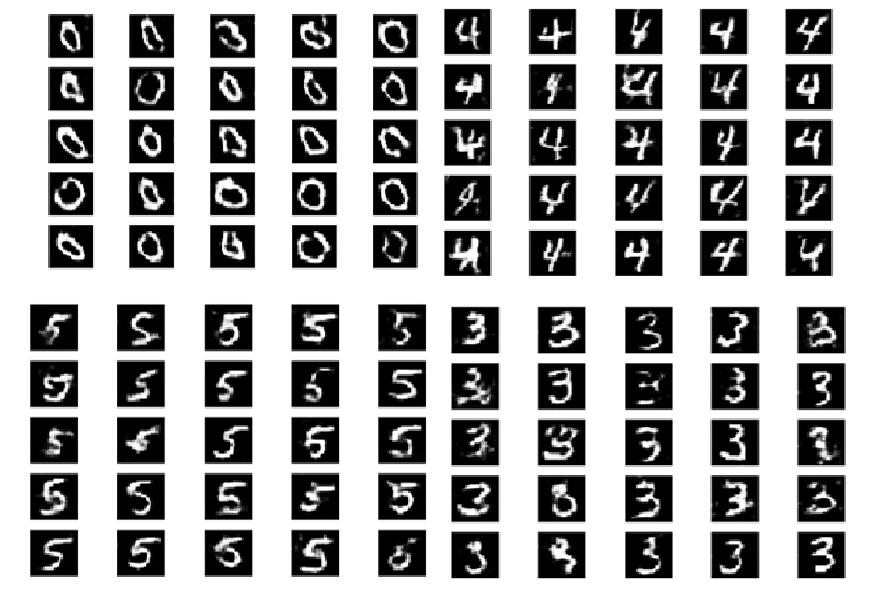


Рисунок 1 – Результат генерации рукописных цифр

Как видно из рисунка, у автора получилось добиться генерация моделью различимых цифр в большинстве случаев, что говорит о возможности приемлемой генерации изображений данного размера.

В исследовании [7] автором были рассмотрены несколько видов GAN, а именно DCGAN, SAGAN и PGGAN. DCGAN представляет собой глубокую сверточную нейронную сеть, автор в своих исследованиях показал, что уже в на разрешении 32х32 нейросеть не способна генерировать приемлемые изображения, что говорит о невозможности ее использования в текущей задаче. Примеры, приведённые автором представлены на рисунке 2.

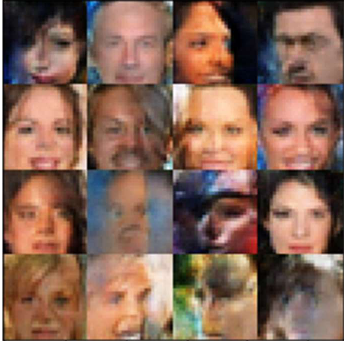


Рисунок 2 –Результат генерации изображений DCGAN

SAGAN представляет собой DCGAN, с модулем self-attention, который позволяет сети искать зависимости за пределом фильтра свёртки. Данный модуль описан в теоретической главе. Примеры сгенерированных данной сетью изображений размера 64х64 представлены в рисунке 3.

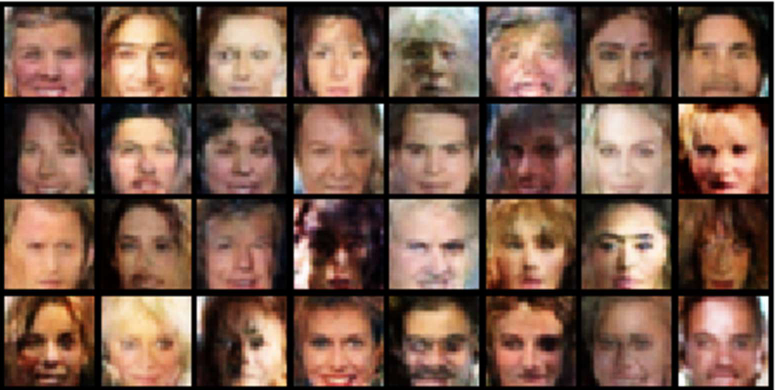


Рисунок 3 –Результат генерации изображений SAGAN

PGGAN представляет собой иной подход к обучению нейросети, при котором она начинает обучение с изображений малого разрешения, размер которых растёт по мере обучения сети. Изображения церквей, сгенерированные данной сетью представлены в рисунке 4.



Рисунок 4 – Результат генерации изображений PGGAN

PGGAN хоть и создаёт наиболее реалистичные изображения, однако он требует колоссальных вычислительных ресурсов, на 300 тысяч итераций у автора ушло 4 часа на производительной видеокарте NVI.

1.3. Выводы из результатов анализа предметной области

В результате анализа имеющихся работ можно сделать следующие выводы: DCGAN слабо подходит для решения данной задачи в связи с большим размером изображения, PGGAN в данной работе был бы наилучшим решением, однако в связи с ограниченностью вычислительных ресурсов использовать данную архитектуру не представляется возможным, поэтому в работе решено использовать архитектуру SAGAN.

# 2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. Генеративно-состязательные нейросети

В текущем разделе приведено описание используемой нейросети (SAGAN) и её составных элементов, а именно: свёрточная нейронная сеть (для неё приведено описание используемых в работе слоёв) и self-attention или «механизм внимания».

2.1. Описание принципа работы GAN

Генеративно-состязательные нейросети (GANs) – вид нейросетей, который позволяет генерировать новые данные, имитирующие распределение обучающей выборки. Данные нейросети состоят из двух подмоделей: дискриминатора и генератора, обучение GAN при этом является обучение двух этих моделей вместе.

Дискриминатор получает на вход либо настоящие данные, либо то, что создал генератор и старается отличить их. Генератор же на основе случайного шума пытается создать данные, которые бы были похожи на образцы из обучающей выборки. На рисунке 5 приведена иллюстрация, показывающая как работает данная модель.

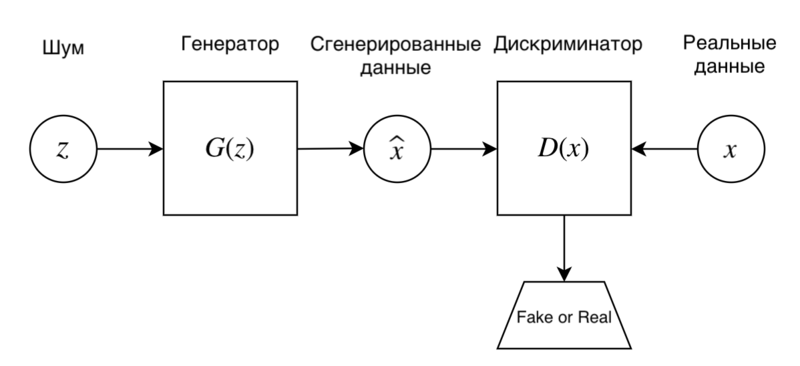


Рисунок 5 – Схема работы GAN

Таким образом, обучение GAN сводится к тому, что дискриминатор в процессе обучения всё лучше учится отличать обучающие образцы от того, что создал генератор, а генератор учится создавать такие образцы, которые были бы неотличимы для дискриминатора от реальных данных.

2.2. Описание свёрточной нейронной сети

Свёрточные нейронные сети – вид нейросетей, используемый преимущественно при анализе изображений. В данной работе используются следующие слои:

* Свёрточный слой: основной слой, который выполняет операцию свёртки над изображением. Он состоит из набора фильтров, которые применяются к изображению и позволяют выделить некоторые признаки;
* Слой активации: слой, применяющий некоторую функцию активации к поступившим на вход данным.

Свёрточные нейронные сети работают по принципу выделения признаков из изображений. Они начинают с применения сверточных фильтров к изображению, которые выделяют определенные признаки, такие как границы, текстуры и формы. Затем эти признаки уплотняются и передаются на следующий слой, где они соединяются с другими признаками и выделяют более сложные формы и объекты.

2.3. Описание механизма внимания

Механизм внимания или self-attention – модуль нейросети позволяющий выделять отдалённые друг от друга признаки, в отличие от свёртки, которая работает только в пределах своего фильтра, self-attention анализирует всё изображение целиком, позволяя выделить наиболее важные части, помимо этого в задачах генерации изображений этот модуль позволяет уменьшить размытость изображения, а также обеспечивает лучшую сходимость модели. Архитектура данного модуля и то, как он взаимодействует с входными данными изображено на рисунке 6.

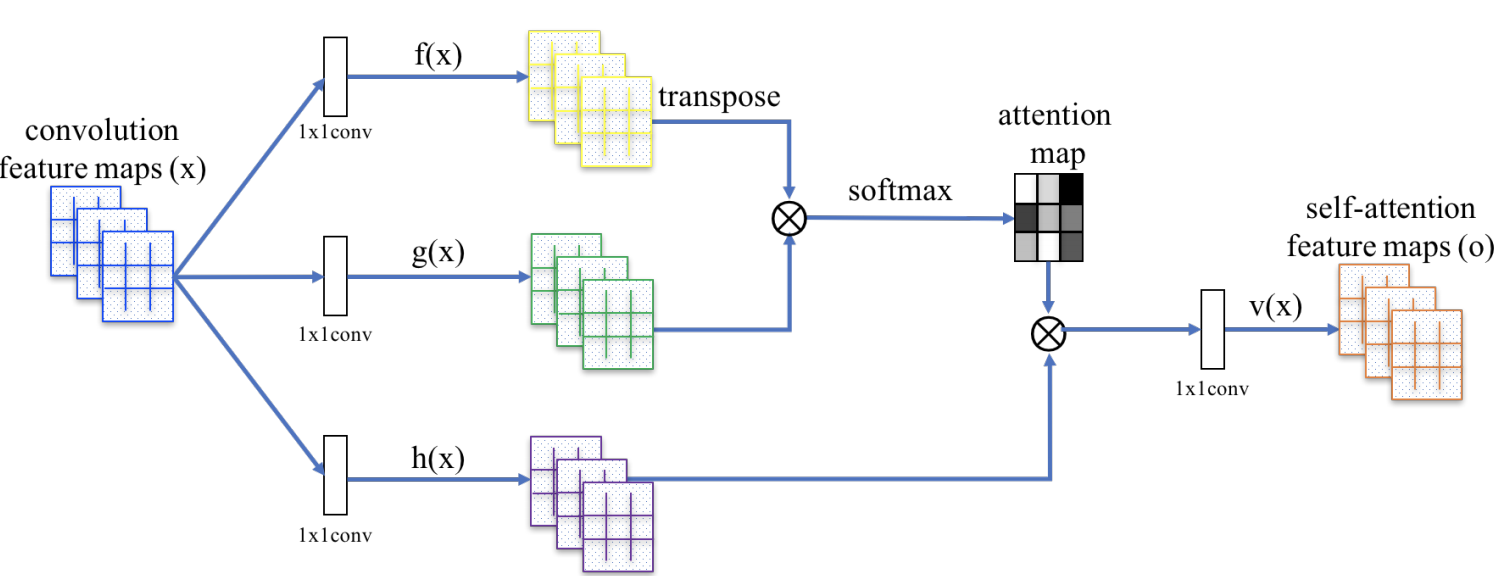


Рисунок 6 – Иллюстрация работы механизма внимания

# 3. ПРОЕКТИРОВАНИЕ

3.1. Определение требований

Результатом курсовой будет являться

**Функциональные требования к проектируемой системе**

1. Система должна автоматически сохранять контрольную точку обучения и модель каждые N (задаваемых пользователем) эпох обучения.
2. Система должна сохранять примеры сгенерированных изображений каждые N (задаваемых пользователем) эпох.
3. Должна быть возможность продолжить обучение с сохранённой контрольной точки.
4. Система должна сохранять историю изменения функции ошибок для генератора и дискриминатора в каждую эпоху.

**Нефункциональные требования к проектируемой системе**

1. Система должна представлять собой генеративно-состязательную нейросеть.
2. Система должна быть написана с использованием языка программирования Python и фреймворка PyTorch.
3. Система должна генерировать четырёхканальные изображения разрешения 64х64.

3.2. Варианты использования системы

Для проектирования системы был использован язык графического описания UML [15]. По результатам проектирования была построена диаграмма вариантов использования, п редставленная на рисунке 7.

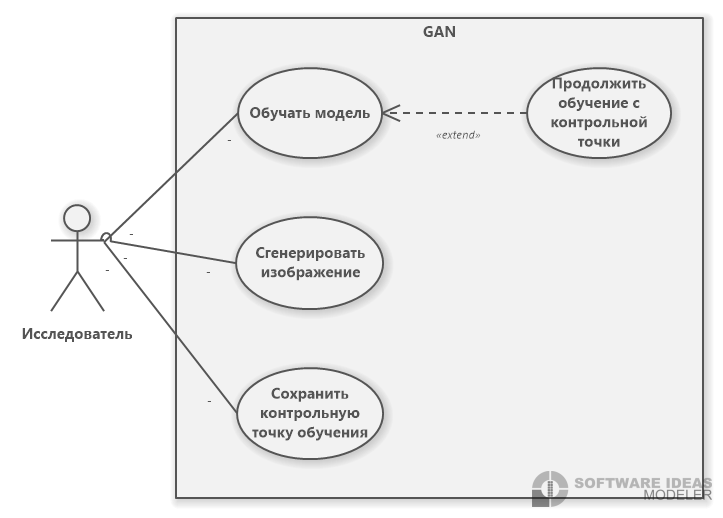
****

Рисунок 7 – Диаграмма вариантов использования

С системой взаимодействует лишь один пользователь – исследователь, он осуществляет обучение модели, задаёт различные параметры, такие как, количество эпох обучения, регулирует как часто модель будет сохранять контрольные точки обучения и примеры сгенерированных изображений.

При этом исследователю доступны следующие прецеденты.

1. Обучать модель – этот прецедент нужен для того, чтобы исследователь мог обучать модель, в том числе обучать модель на своих данных.
2. Сгенерировать изображение – данный прецедент применяется для того, чтобы получить изображение из генератора.
3. Сохранить контрольную точку обучения – этот прецедент используется для того, чтобы сохранить контрольную точку обучения и модель для последующей работы.
4. Продолжить обучение с контрольной точки – прецедент, используемый для того, чтобы продолжить обучение модели с заранее созданной контрольной точки.

Спецификации вариантов использования приведены в таблицах 1–4.

Таблица 1 – Спецификация варианта использования «Обучать модель»

|  |
| --- |
| *Прецедент:* Обучать модель |
| *ID:* 1 |
| *Аннотация:* Обучение нейросети на заданном наборе данных |
| *Главные актеры:* Исследователь |
| *Предусловия:* Имеется инициализированный объект модели. |
| *Основной поток:*  Исследователь запускает код обучения модели, задавая N эпох, в течение которых происходит обучение модели.  Модель обучается на заданном датасете в течение N эпох. |
| *Постусловия:* Веса нейронной сети изменены в процессе обучения. |

Таблица 2 – Спецификация варианта использования «Сгенерировать изображение»

|  |
| --- |
| *Прецедент:* Сгенерировать изображение |
| *ID:* 2 |
| *Аннотация:* Система генерирует случайное изображение |
| *Главные актеры:* Исследователь |
| *Предусловия:* Имеется инициализированный объект модели |

Окончание таблицы 2

|  |
| --- |
| *Основной поток:*  Исследователь запускает код генерации изображения, указывая название файла, в который будет сохранено изображение.  Система создаёт изображение и сохраняет его под заданным именем. |
| *Постусловия:* Под заданным именем сохранено сгенерированное изображение. |

Таблица 3 – Спецификация варианта использования «Сохранить контр. точку обучения»

|  |
| --- |
| *Прецедент:* Сохранить контрольную точку обучения |
| *ID:* 3 |
| *Аннотация:* Система создаёт контрольную точку обучения |
| *Главные актеры:* Исследователь |
| *Предусловия:* Имеется инициализированный объект модели. |
| *Основной поток:*  Исследователь запускает код сохранения контрольной точки, указывая название файла, в который будет сохранена контрольная точка.  Система создаёт контрольную точку обучения. |
| *Постусловия:* Создана контрольная точка обучения. |

Табл. 4. Спецификация UseCase «Продолжить обучение с контрольной точки»

|  |
| --- |
| *Прецедент:* Продолжить обучение с контрольной точки |
| *ID:* 4 |
| *Аннотация:* Нейросеть генерирует случайное изображение |
| *Главные актеры:* Исследователь |

Окончание таблицы 4

|  |
| --- |
| *Предусловия:* Имеется инициализированный объект модели и сохранённая контрольная точка обучения. |
| *Основной поток:*  Исследователь запускает код обучения, указав, что нужно продолжить обучение с контрольной точки и указав имя файла, в котором она хранится.  Нейросеть продолжает своё обучение с данной контрольной точки. |
| *Постусловия:* Веса нейросети изменены в процессе обучения. |

# 4. АРХИТЕКТУРА СИСТЕМЫ

4.1. Общее описание архитектуры системы

На рисунке 8 представлена диаграмма классов системы «Skins Generator». Система состоит из следующих основных компонентов: Trainer, Generator, Discriminator. Trainer применяется для обучения модели генератора и дискриминатора и использует данные, которые будет переданы в его функцию train. Generator используется для непосредственной генерации «скина». Discriminator для определения того, является ли переданное в него, в виде тензора изображение «скином».

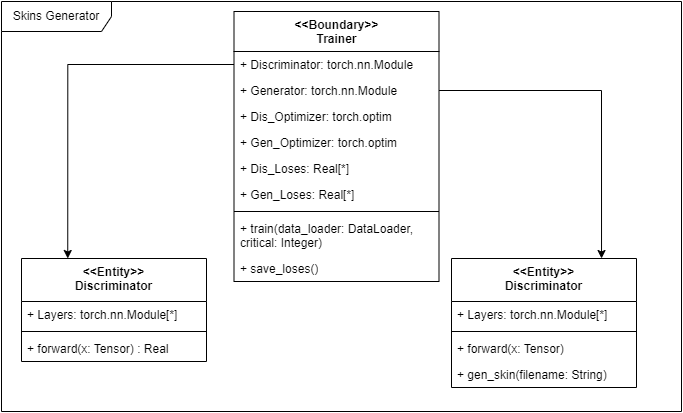


Рисунок 8 – Диаграмма классов системы Skins Generator

4.2. Описание компонентов, составляющих систему

В данном разделе предоставлено более подробное описание всех компонентов системы, включая описание полей и методов классов. При этом в текущем разделе были опущены некоторые подробности касающиеся непосредственно процесса обучения, они более подробно описаны в разделе реализации.

4.2.1. Описание класса Trainer

Данный класс применяется для тренировки генеративно-состязательной нейросети. Он содержит следующие поля:

* *Discriminator*: является дискриминатором в генеративно-состязательной нейросети, применяется для того, чтобы отличить созданное генератором изображение от сгенерированного;
* *Generator*: является генератором в генеративно-состязательной нейросети, применяется для генерации «скинов» из случайного шума;
* *Dis\_optimizer*: оптимизатор дискриминатора, отвечает за то как именно будут изменяться параметры дискриминатора в процессе тренировки;
* *Gen\_optimizer*: оптимизатор генератора, отвечает за то как именно будут изменяться параметры генератора в процессе тренировки;
* *Dis\_loses*: список значений функции потерь дискриминатора, применяется для того, чтобы проследить как изменяются значения функции потерь при обучении;
* *Gen\_loses*: список значений функции потерь генератора, применяется для того, чтобы проследить как изменяются значения функции потерь при обучении.

Помимо перечисленных полей у Trainer имеются следующие методы:

* *train*: метод осуществляющий непосредственно обучение генератора и дискриминатора. В аргумент data\_loader передается объект класса DataLoader из PyTorch, который содержит обучающую выборку, разделенную на пакеты или «батчи». В аргумент critical передаётся натуральное число, изменение весов дискриминатора происходит каждые critical батчей, это делается, во-первых, чтобы избежать переобучения дискриминатор, во-вторых, чтобы генератор успевал адаптироваться к текущей модели дискриминатор, чтобы стабилизировать обучение;
* *save\_losses*: метод, сохраняющий текущие значение функций потерь в соответствующие списки.

4.2.2. Описание класса Discriminator

Этот класс представляет собой дискриминатор в генеративно-состязательной нейросети. Он имеет всего одно поле: Layers, которое хранит в себе слои нейросети, более подробно это поле будет описано в этапе реализации. Также данный класс имеет один метод: forward, принимающий на вход изображение в виде тензора (многомерную матрицу) с четырьмя каналами, и определяющий, является ли поданное изображение «скином».

4.2.3. Описание класса Generator

Этот класс представляет собой генератор в генеративно-состязательной нейросети. Он имеет одно поле: Layers, которое хранит в себе слои нейросети, так же как и для дискриминатора подробно это поле будет описано в этапе реализации. Данный класс имеет два метода: forward, принимающий на вход шум в виде одноканального тензора размера 64х64 и генерирующий на основе этого шума «скин» в виде четырехканального тензора размера 64х64. Второй метод: gen\_skin(filename), создаёт случайный одноканальный тензор размера 64х64, после чего отправляет этот шум методу forward(), сохраняя созданный этим методом скин в файл с именем filename.

4.3. Топология нейронной сети

В данном разделе описана топология нейронной сети, а именно используемые слои, функции активации. При этом в нейронной сети имеются повторяющиеся комбинации элементов, которые, с целью упрощения архитектуры и кода программы, были объединены в блоки. Так для дискриминатора был выделен *DisConvBlock*, а для генератора *GenDeconvBlock*. Описание архитектуры этих блоков, и то, в каком количестве и порядке они представлены в нейронной сети, приведено в разделах с описанием топологии нейронных сетей.

В генераторе и дискриминаторе используется механизм внимания или *self-attention*, более подробное описание и архитектура которого приведены в теоретической главе.

4.3.1. Топология дискриминатора

В архитектуре дискриминатора можно выделить повторяющиеся элементы, а именно Conv2d, батч-нормализация, функция активации и Dropout, которые были объединены в блок дискриминатора. Архитектура этого блока представлена на рисунке 9.

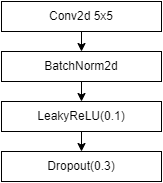


Рисунок 9 – Архитектура блока дискриминатора

По сути дискриминатор «собран» из блоков, с добавлением некоторым элементов. Также в дискриминаторе применяется механизм внимания, который на рисунке 10 назван SelfAttention. Архитектура дискриминатора приведена в рисунке 10. Рядом со стрелками изображены типы данных, которые имеются на входе или выходе соответствующего элемента.

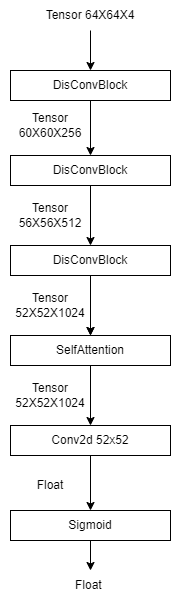


Рисунок 10 – Архитектура дискриминатора

4.3.2. Топология генератора

В архитектуре генератора, по аналогии с дискриминатором можно выделить повторяющиеся элементы, а именно, ConvTranspose2d, батч-нормализация, функция активация и Dropout, которые были объединены в блок генератора. Архитектура этого блока представлена на рисунке 11.

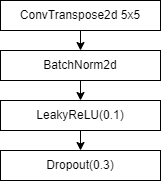


Рисунок 11 – Архитектура блока генератора

Генератор, так же, как и дискриминатор, «собран» из блоков, с добавлением некоторым элементов, и в нем также применяется механизм внимания, названный на рисунке 12 SelfAttention. его архитектура приведена на рисунке 12. Рядом со стрелками изображены типы данных, которые имеются на входе или выходе соответствующего слоя нейронной сети.

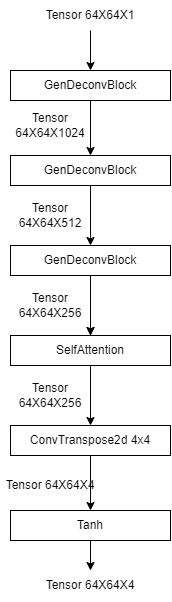


Рисунок 12 – Архитектура генератора

# 5. реализация

5.1 Программные средства реализации

Для реализации нейросети для генерации скинов персонажей видеоигры *Minecraft* был выбран язык программирования *Python 3.10* [3]. Для написания, отладки кода и обучения нейронной сети был использован *Google Colab*.

В процессе разработки использовались следующие библиотеки.

**PyTorch** [4]

Библиотека для создания нейронных сетей для языка Python, созданная на базе проекта Torch.

**Torchvision** [5]

Данная библиотека является частью проекта PyTorch. Она включает в себя инструменты для работы с изображениями и видео, а также популярные архитектуры нейронных и наборы данных для обучения.

**Numpy** [2]

Эта библиотека с открытым исходным кодом предоставляет инструменты для работы с многомерными массивами и высокоуровневые математические функции.

**Matplotlib** [11]

Библиотека с открытым исходным кодом, предназначенная для визуализации данных.

5.2. Сбор и предобработка данных

Для обучения генеративно-состязательной нейронной сети применялся набор данных, содержащих более 900 000 тысяч скинов [10]. Он содержит четырёхканальные изображения в формате .png разрешения 64x64 пикселя. Пример таких изображений приведен на рисунке 13.



Рисунок 13 – Примеры корректных изображений

Однако данный набор данных содержит не только изображения с дизайном персонажа, но и некорректные изображения, примеры которых приведены в рисунке 14.



Рисунок 14 – Примеры некорректных изображений

**Очистка набора данных от некорректных изображений**

Очистка набора данных проводилась путём подсчета в изображениях количества непрозрачных пикселей, как правило, в изображениях скинов количество непрозрачных пикселей варьируется от 2100 до 2500. Исходные изображения содержались в папке *Skins.* Изображения, в которых количество непрозрачных пикселей попадало в этот диапазон переносились в новую папку *SkinsCleaned.* Исходный код программы, выполняющей операцию подсчета пикселей и перенос корректных изображений в соответствующую папку, представлен в листинге 1.

Листинг 1 – Исходный код программы, выполняющей очистку набора данных

import os

import matplotlib.image as img

import shutil

from multiprocessing import Process

def remove\_not\_skin(a, b):

print(f"Deleting not skin in range({a}, {b}) is started")

for filename in names[a:b]:

image = img.imread(f"Skins/{filename}")

count\_of\_transparent\_pixels = 0

for x in image:

for y in x:

if y[3] == 0:

count\_of\_transparent\_pixels += 1

if 2100 < count\_of\_transparent\_pixels < 2500:

shutil.copy(f"Skins/{filename}", f"SkinsCleaned/{filename}")

names = os.listdir("SkinsCleaned")

if \_\_name\_\_ == "\_\_main\_\_":

p1 = Process(target=remove\_not\_skin, args=(0, len(names)//4))

p2 = Process(target=remove\_not\_skin, args=(len(names)//4, len(names)//2))

p3 = Process(target=remove\_not\_skin, args=(len(names)//2, len(names)//2 + len(names)//4

p4 = Process(target=remove\_not\_skin, args=(len(names)//2 + len(names)//4, len(names)))

p1.start()

p2.start()

p3.start()

p4.start()

p1.join()

p2.join()

p3.join()

p4.join()

В результате работы данной программы было отобрано более 700 000 подходящих изображений. Однако, в связи с ограниченностью вычислительных ресурсов, из очищенного набора данных было случайным образом отобрано около 10 000 изображений для дальнейшего обучения нейронной сети.

5.3. Реализация и обучение нейронной сети

При реализации генеративно-состязательной нейросети использовался фреймворк PyTorch. Для обучения генератора и дискриминатора применялся оптимизатор Adam со скоростью обучения 0.0001 и параметрами betas = (0.5, 0.999). Согласно исследованию [12] данные параметры дают наилучшие результаты при обучении.

Также в качестве функции потерь была использована бинарная кросс энтропия. Выводы, сделанные в исследовании [14] позволяют утверждать данная функция потерь позволяет эффективно обучать GAN.

В дискриминаторе и генераторе используется механизм внимания, исходный код реализации которого приведен в листинге 2.

Листинг 2 – Исходный код модуля SelfAttention

class SelfAttention(nn.Module):

def \_\_init\_\_(self, in\_channels):

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

self.f = torch.nn.Conv2d(in\_channels, in\_channels//8, 1)

self.g = torch.nn.Conv2d(in\_channels, in\_channels//8, 1)

self.h = torch.nn.Conv2d(in\_channels, in\_channels, 1)

self.gamma = nn.Parameter(torch.zeros(1))

self.softmax = nn.Softmax(dim=-1)

def forward(self, x):

batch\_size, C, width, height = x.size()

proj\_query=self.f(x).view(batch\_size,-1,width\*height).permute(0, 2,1)

proj\_key = self.g(x).view(batch\_size,-1,width\*height)

energy = torch.bmm(proj\_query,proj\_key)

attention = self.softmax(energy)

proj\_value = self.h(x).view(batch\_size,-1,width\*height)

out = torch.bmm(proj\_value, attention)

out = out.view(batch\_size,C,width,height)

out = self.gamma\*out + x

return out

5.3.1. Реализация дискриминатора

Для оптимизации процесса обучения дискриминатора применяется односторонне сглаживание меток, т.е. целевые значение дискриминатора для изображений из набора данных выбирались случайным образом из диапазона [0.8, 1.0).

В дискриминаторе был выделен DisConvBlock, являющийся его основным элементом, исходный код, реализующий этот модуль представлен в листинге 3.

Листинг 3 – Исходный код модуля DisConvBlock

class DisConvBlock(nn.Module):

def \_\_init\_\_(self, in\_channels, out\_channels, kernel\_size, padding):

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

self.conv = spectral\_norm(torch.nn.Conv2d(in\_channels=in\_channels,

out\_channels=out\_channels,

kernel\_size=kernel\_size, padding=padding,

bias=False))

self.norm = nn.BatchNorm2d(out\_channels)

self.act = nn.LeakyReLU(0.1)

self.drop = nn.Dropout(0.3)

def forward(self, x):

x = self.conv(x)

x = self.act(x)

x = self.drop(x)

return x

Исходный код дискриминатора представлен в листинге 4.

Листинг 4 – Исходный код дискриминатора

class discriminator(nn.Module):

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

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

self.conv\_block1 = DisConvBlock(4, 256, 5, 0)

self.conv\_block2 = DisConvBlock(256, 512, 5, 0)

self.conv\_block3 = DisConvBlock(512, 1024, 5, 0)

self.attn = SelfAttention(1024)

self.conv1 = spectral\_norm(nn.Conv2d(in\_channels=1024, out\_channels=1,

kernel\_size=52))

self.act = nn.Sigmoid()

def forward(self, x):

x = self.conv\_block1(x)

x = self.conv\_block2(x)

x = self.conv\_block3(x)

x = self.attn(x)

x = self.conv1(x)

x = x.view(-1, 1)

x = self.act(x)

return x

5.3.2. Реализация генератора

В генераторе был выделен GenDeconvBlock, являющийся его основным элементом, исходный код, реализующий этот модуль представлен в листинге 5.

Листинг 5 – Исходный код блока генератора

class GenConvBlock(nn.Module):

def \_\_init\_\_(self, in\_channels, out\_channels, kernel\_size, padding):

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

self.conv = torch.nn.ConvTranspose2d(in\_channels=in\_channels,

out\_channels=out\_channels,

kernel\_size=kernel\_size, padding = padding,

bias=True)

self.norm = nn.BatchNorm2d(out\_channels)

self.act = nn.ReLU(0.1)

self.drop = nn.Dropout(0.3)

def forward(self, x):

x = self.conv(x)

x = self.norm(x)

x = self.act(x)

x = self.drop(x)

return x

Исходный код генератора представлен в листинге 6.

Листинг 6 – Исходный код генератора

class generator(nn.Module):

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

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

self.deconv\_block1 = GenConvBlock(1, 1024, 5, 2)

self.deconv\_block2 = GenConvBlock(1024, 512, 5, 2)

self.deconv\_block3 = GenConvBlock(512, 256, 5, 2)

self.attn = SelfAttention(256)

self.deconv1 = nn.ConvTranspose2d(256, 4, 3, padding=1)

self.act4 = nn.Tanh()

def forward(self, x):

x = self.deconv\_block1(x)

x = self.deconv\_block2(x)

x = self.deconv\_block3(x)

x = self.attn(x)

x = self.deconv1(x)

x = self.act4(x)

return (x)

def gen\_skin(filename):

noise = (torch.rand(1, 1, 64, 64))

noise = noise.to(device)

save\_image(forward(noise), filename)

# ЛИТЕРАТУРА

1. Li-Chia Yang, Szu-Yu Chou, Yi-Hsuan Yang MIDINET: A CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORK FOR SYMBOLIC-DOMAIN MUSIC GENERATION // Research Center for IT innovation, Academia Sinica. 2017 г.
2. NumPy [Электронный ресурс]. URL: <https://www.numpy.org/> (дата обращения: 20.05.2023 г.).
3. Python Documentation [Электронный ресурс]. URL: <https://docs.python.org/3.10/> (дата обращения 20.05.2023 г.).
4. PyTorch Documentation [Электронный ресурс]. URL: https://pytorch.org/docs/stable/index.html (дата обращения: 20.05.2023 г.).
5. Torchvision Documentation [Электронный ресурс]. URL: https://pytorch.org/vision/stable/index.html (дата обращения: 20.05.2023 г.).
6. Vivek Narayanaswamy, Jayaraman J. Thiagarajan, Rushil Anirudh, Andreas Spanias Unsupervised Audio Source Separation using Generative Priors // SenSIP Center, School of ECEE, Arizona State University. 2020 г.
7. Бычков А. В. Алгоритмы синтеза изображений в больших разрешениях на основе генеративно-состязательных нейронных сетей: магистерская диссертация. Минск:, 2020. 53 с.
8. Ковалев В. А. Генерация искусственных рентгеновских изображений грудной клетки с использованием генеративно-состязательных нейронных сетей / В. А. Ковалев, С. А. Козловский, А. А. Калиновский // Информатика. – 2018. − Т. 15, No 2. – С. 7–16.
9. Шкиря А.С. РАЗРАБОТКА ГЕНЕРАТИВНО-СОСТЯЗАТЕЛЬНОЙ СЕТИ ДЛЯ СОЗДАНИЯ ИЗОБРАЖЕНИЙ // Электронный сборник статей по материалам XCII студенческой международной научно-практической конференции. Новосибирск. – 2020 г. 38 с.
10. AcademicTorrents [Электронный ресурс]. URL: https://academictorrents.com/details/14cf27fca7f26714d2a5193dc95348a4712cdcdf (дата обращения: 20.05.2023 г.).
11. Matplotlib Documentation [Электронный ресурс]. URL: <https://matplotlib.org/stable/index.html> (дата обращения: 20.05.2023 г.).
12. Radford A., Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks [Электронный ресурс]. URL https://arxiv.org/pdf/1511.06434.pdf (дата обращения: 20.05.2023 г.).
13. Goodfellow I.J., NIPS 2016 Tutorial: Generative Adversarial Networks [Электронный ресурс]. URL https://arxiv.org/pdf/1701.00160.pdf (дата обращения: 20.05.2023 г.).
14. Salimans T. [и др.]. Improved Techniques for Training GANs [Электронный ресурс]. URL: <https://arxiv.org/pdf/1606.03498.pdf> (дата обращения: 20.05.2023 г.).
15. Фаулер М. Основы UML. Краткое руководство по стандартному языку объектного моделирования, 3-е изд. ‒ 2004. ‒ 192 с.