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

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Генерация обликов персонажей видеоигры Minecraft с помощью нейросетей

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-364.ВКР

Научный руководитель,
доцент кафедры СП, к.т.н.
_____ М.В. Сухов

Автор работы,
студент группы КЭ-433
_____ Э.Э. Сагатдинов

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

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
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-433
Сагатдинову Эмилю Эдуардовичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Генерация обликов персонажей видеоигры Minecraft с помощью нейросетей.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Sohl-Dickstein J., Weiss E., Maheswaranathan N., Ganguli S. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. [Электронный ресурс] // arXiv.org. 2015. Дата обновления: 18.11.2015 г. URL: https://arxiv.org/abs/1503.03585 (дата обращения: 27.04.2024 г.).

3.2. Ho J., Jain A., Abbeel P. Denoising Diffusion Probabilistic Models. [Электронный ресурс] // arXiv.org. 2020. Дата обновления: 16.12.2024 г. URL: https://arxiv.org/abs/2006.11239 (дата обращения: 27.04.2024 г.).

3.3. AcademicTorrents. [Электронный ресурс] URL: https://academictorrents.com/details/14cf27fca7f26714d2a5193dc95348a4712cdcdf (дата обращения: 27.04.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области.

4.2. Осуществить сбор и предобработку данных для обучения нейронной сети.

4.3. Построить архитектуру нейронной сети.

4.4. Обучить нейронную сеть.
4.5. Оценить результаты работы нейронной сети.
4.6. Разработать веб-сайт для просмотра и оценки качества сгенерированных скинов.
5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.т.н. М.В. Сухов

Задание принял к исполнению Э.Э. Сагатдинов

ГЛОССАРИЙ

1. *GAN (Генеративно-состязательная нейронная сеть)* – подход машинного обучения, в котором обучаются две модели: генератор, который обучается подделывать распределение данных, и дискриминатор, который оценивает вероятность того, что образец пришел из тренировочных данных, а не из генератора [9].

2. *VAE (Вариационный автокодировщик)* – подход машинного обучения, в котором обучаются две модели: кодировщик, который обучается сжимать распределение данных в пространство меньшего размера, и декодировщик, который обучается восстанавливать исходные данные из сжатой формы [35].

3. *Диффузионная нейронная сеть* – модель, восстанавливающая исходные данные после зашумления [27].

4. *Эмбеддинг* – это вектор, получаемый в процессе преобразования объектов в векторное пространство фиксированной размерности, так что семантически похожие объекты находятся ближе друг к другу в этом пространстве [21].

5. *Нейронная сеть* – это математическая модель, которая имитирует работу биологических нейронов в мозге, используя связанные между собой элементы, называемые нейронами или узлами, для обработки и передачи информации [40].

6. *Механизм внимания (self-attention)* – это механизм, используемый в нейронных сетях, который позволяет модели сосредоточиться на наиболее важных частях входных данных и учитывать взаимосвязи между ними [32].

7. *Скин (skin)* – текстура, помещаемая на модель игрока в видеоигре Minecraft. Эта текстура определяет дизайн персонажа [36].

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

На сегодняшний день видеоигры являются очень популярным способом проведения досуга, и одной из самых популярных видеоигр является Minecraft [28]. В игре присутствует возможность каждому игроку создать свой уникальный дизайн персонажа или же скин. Создание скинов с помощью искусственных нейронный сетей может быть востребовано среди тех игроков, которые не обладают достаточными навыками для создания собственного скина.

Также в данный момент актуальной темой является генерации контента с помощью нейронных сетей, это может быть полезным в том числе для разработчиков модификаций для данной видеоигры, что упростило бы создание дизайна для модификаций и позволило бы ускорить их разработку.

Постановка задачи

Цель работы заключается в проектировании, реализации и обучении диффузионной нейронной сети, генерирующей скин (дизайн персонажа). Скины представляют собой четырехканальные изображения размера 64x64 в формате «.png».

Для достижения цели работы необходимо выполнить следующие задачи.

1. Провести анализ предметной области.
2. Осуществить сбор и предобработку данных для нейронной сети.
3. Построить архитектуру нейронной сети.
4. Обучить нейронную сеть.
5. Оценить результаты работы нейронной сети.
6. Улучшить работу нейронной сети.
7. Создать веб-сайт, который бы предоставлял пользователям возможность просмотра сгенерированных дизайнов.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 46 страниц, объем списка литературы – 40 источников.

В первой главе выполнен анализ предметной области и обзор существующих подходов к решению задачи выпускной квалификационной работы, также описаны сильные и слабые стороны каждого подхода и приведена аргументация касательно выбора диффузионных нейронных сетей.

Вторая глава посвящена проектированию системы, а именно, определению требований, выбору конкретных архитектур диффузионных нейронных сетей, проектированию топологий и определению необходимого функционала веб-сайта.

Третья глава описывает реализацию системы, а именно, содержит описание используемых для реализации средств, информацию о сборе и предобработке данных, а также сведения о реализации и обучении нейронных сетей.

В четвертой главе выпускной квалификационной работы проведено тестирование разработанной системы и проверка работоспособности созданного веб-сайта, а также результаты опросов среди целевой аудитории пользователей о качестве сгенерированных персонажей.

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 архитектур для реализации проекта. Были рассмотрены следующие архитектуры: GAN [9], VAE [35] и диффузионные нейронные сети [27]. Также были описаны достоинства и недостатки каждой из архитектур.

Анализ инструментов для реализации проекта

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

Анализ существующих архитектур для реализации проекта

В исследовании [39] автором были рассмотрены несколько видов GAN, а именно DCGAN, SAGAN и PGGAN. Наилучшие результаты были получены с помощью PGGAN.

При обучении GAN обучаются две нейросети: генератор и дискриминатор, при этом дискриминатор стремится различать образцы, пришедшие из обучающей выборки и образцы, сгенерированные своим конкурентом, генератор же стремится создавать такие образцы, которые дискриминатор не сможет отличить от настоящих [9].

PGGAN представляет следующий подход к обучению генеративных нейронных сетей, при котором нейросеть начинает обучение с малого разрешения изображений, размер которых растет по мере обучения сети. Такой подход обеспечивает наилучшее качество среди моделей данного семейства, однако требует большого количества вычислительных ресурсов, автор тратил 4 часа на эпоху. Изображения церквей, сгенерированные данной сетью представлены на рисунке 1. Как видно из изображений, они имеют приемлемое качество, однако их по-прежнему довольно просто отличить от настоящих.

Рисунок 1 – Результат генерации изображений PGGAN

Несмотря на приемлемое качество генерации, генеративно-состязательные сети имеют следующие недостатки: mode collapse (ситуация, при которой сеть не способна генерировать разнообразные изображения), исчезающие градиенты и низкое качество изображений [5]. Эти проблемы устраняются настройкой гиперпараметров модели, однако их подбор потребует большого количества вычислительных ресурсов.

Вариационные автокодировщики (VAE) работают следующим образом: имеются две нейросети, кодировщик и декодировщик. Кодировщик стремится сжать образец в пространство признаков меньшего размера, декодировщик обучается восстанавливать это сжатое пространство признаков в исходный образец [35]. С помощью кодировщика можно получить статистические параметры сжатого пространства признаков, а именно: среднее отклонение, дисперсию и матрицу ковариации между данными сжатого пространства, после чего случайным образом создать образец, соответствующий данным параметрам, используя эти сгенерированные признаки их

можно подать на вход декодировщику, в теории получая новое изображение [31].

В работе [35] автор применил вариационный автокодировщик для генерации рукописных цифр, используя набор данных MNIST [45]. Результаты этой генерации с прогрессом по эпохам представлены на рисунке 2.

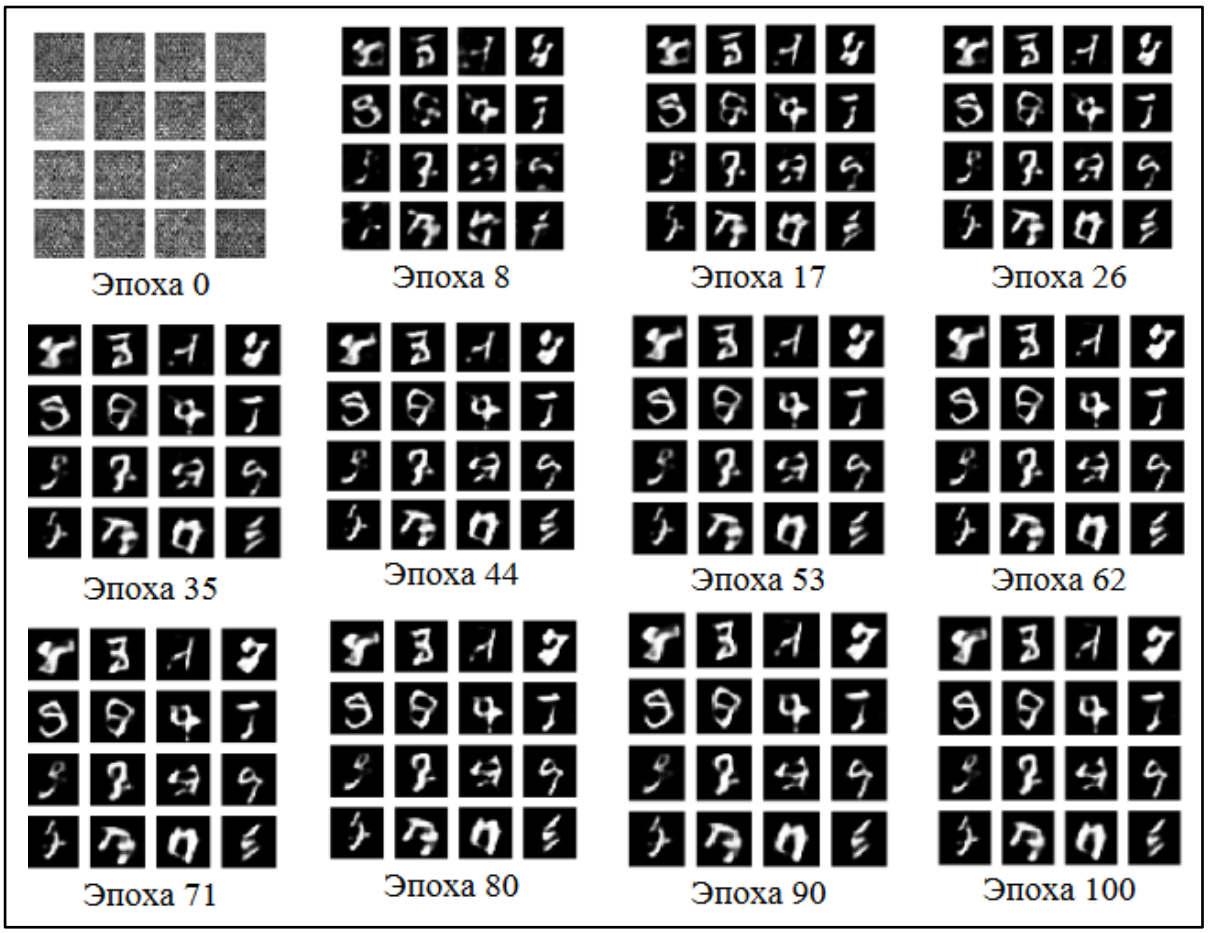

Рисунок 2 – Генерация рукописных цифр с использованием VAE

Как видно из рисунка, применение данной архитектуры не позволяет добиться приемлемого качества генерации, даже на изображениях разрешения 28x28.

Диффузионные нейронные сети имеют два этапа в своем обучении: прямая диффузия и обратная. Прямая диффузия является цепью Маркова, которая накладывает на образцы из обучающего набора случайный шум, до тех пор, пока не будет получено полностью зашумленное изображение. При

обратной диффузии нейронная сеть стремится из зашумленных изображений восстановить исходное, шаг за шагом удаляя шум, при этом нейросеть предсказывает наложенный шум [27]. Пример прямой и обратной диффузии представлен на рисунке 3.

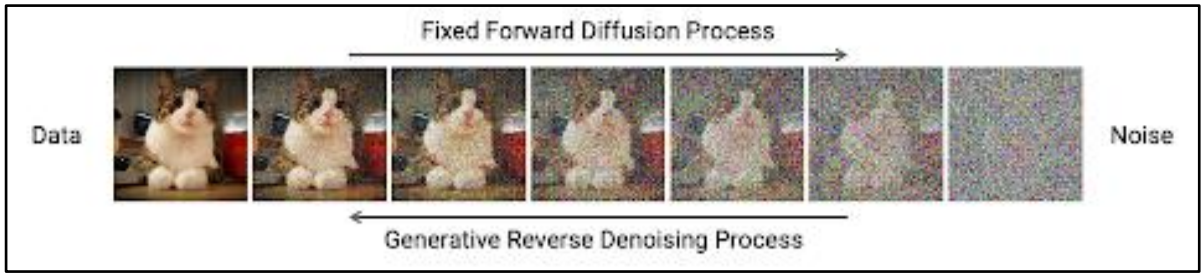

Рисунок 3 – Процессы прямой и обратной диффузии

В работе [27] авторы сгенерировали рукописные цифры, используя данный вид нейросетей и набор данных MNIST, результаты представлены на рисунке 4.

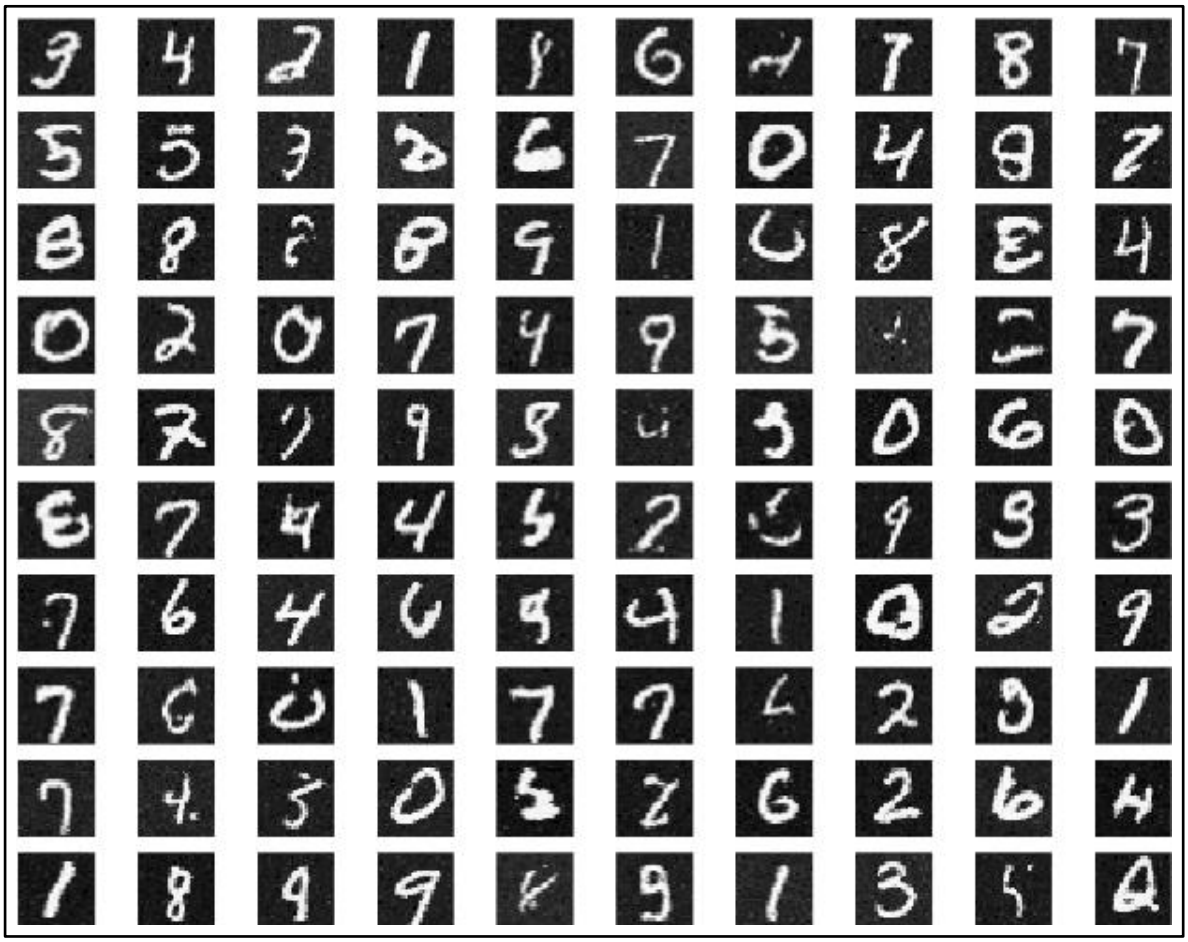

Рисунок 4 – Рукописные цифры, сгенерированные с помощью диффузии

Как видно из рисунка, диффузионная нейронная сеть лучше справилась с задачей, чем VAE, изображения получились гораздо более качественными и имеют меньше шума.

В отличие от GAN и VAE, для обучения диффузионной нейронной сети требуется обучать всего одну модель, т.е. все веса будут участвовать в генерации образцов, однако данный вид нейросетей работает медленнее, чем другие.

Выводы по первой главе

В результате анализа имеющихся работ можно сделать следующие выводы: GAN способны генерировать качественные образцы, однако их обучение достаточно сложное и требует тонкой настройки гиперпараметров, VAE обучаются довольно просто, однако не способны генерировать образцы приемлемого качества. И GAN, и VAE требуют обучения двух нейросетей, что означает, что половина обученных весов не будет участвовать непосредственно в генерации изображений. Диффузионные нейросети способны генерировать образцы высокого качества и обучаются проще чем модели, построенные на других архитектурах, однако диффузионные нейронные сети работают медленнее двух других архитектур, но при этом существуют способы ускорить их работу.

В связи со сложностью обучения GAN и низким качеством изображений, генерируемых VAE, было решено использовать диффузионные нейронные сети.

2. ПРОЕКТИРОВАНИЕ

2.1. Определение требований

В ходе проектирования системы были определены следующие требования.

Функциональные требования к проектируемой системе

Были выделены следующие функциональные требования.

1. Должно быть реализовано автоматическое сохранение контрольной точки обучения и модели каждые N эпох обучения.
2. Должна быть возможность продолжить обучение с сохраненной контрольной точки.
3. Во время обучения должно происходить сохранение истории изменения функции ошибок.
4. Веб-сайт должен предоставлять пользователям возможность визуальной оценки сгенерированных моделей и прохождения опроса о качестве созданных персонажей.

Нефункциональные требования к проектируемой системе

Были выделены следующие нефункциональные требования.

1. Для генерации дизайнов персонажей должна использоваться диффузионная нейронная сеть.
2. Нейросеть должна быть реализована с использованием языка программирования Python и фреймворка PyTorch.
3. Нейронная сеть должна генерировать четырехканальные изображения разрешения 64x64.

2.2. Проектирование нейросети на архитектуре U-Net

Диффузионная нейронная сеть обучается восстанавливать исходные данные после наложения на них шума, при этом очевидно, что размерность изображения после зашумления остается той же. А это означает, что вход и выход модели имеют одинаковую размерность (в данном случае на выходе модели находится четырехканальный тензор размерности 64x64).

Учитывая, что вход и выход имеют одинаковый размер, разумным будет использовать архитектуру U-net [23]. Изображение с примером такой архитектуры приведено на рисунке 5.

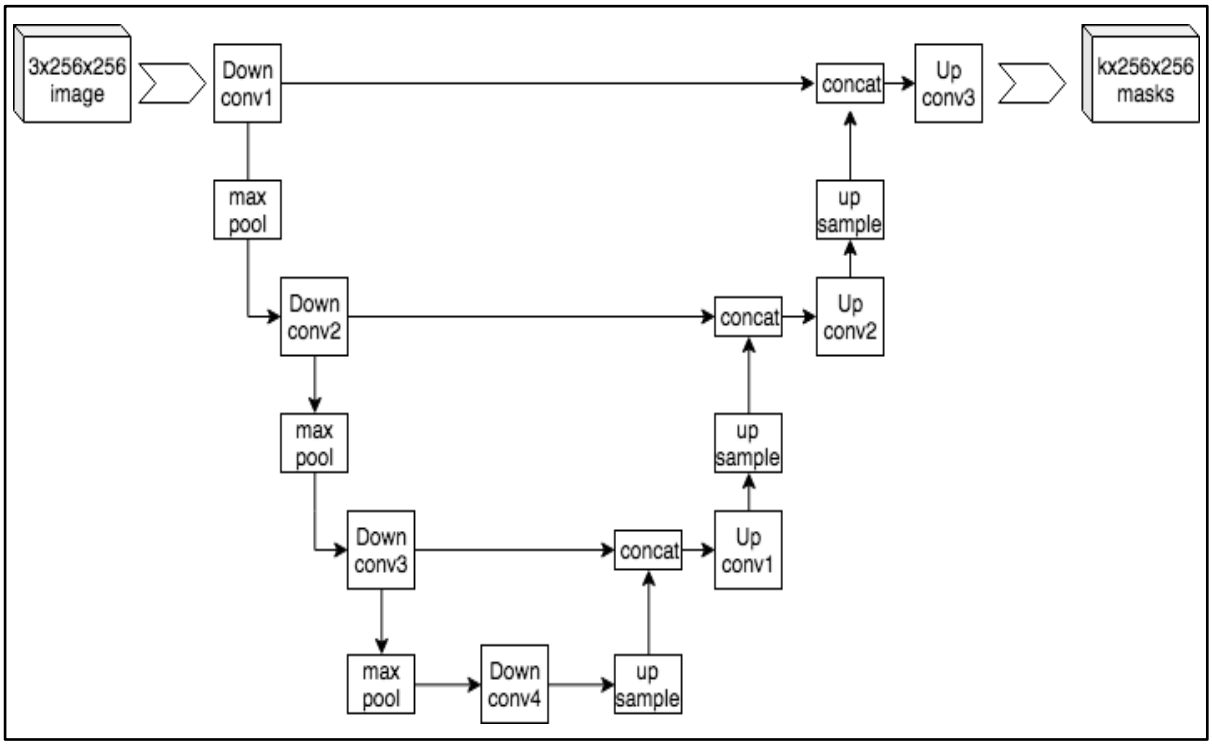

Рисунок 5 – Пример архитектуры U-Net

Как видно из рисунка, эта архитектура состоит трех частей: из «сверточной» части слева, которая несколько раз пропускает изображение через свертку и max pool, который вдвое уменьшает тензор по ширине и высоте, из центральной части, которая осуществляет только свертку и не меняет размерность, а также из «разверточной» части, которая пропускает данные через свертку и обратный пуллинг, перед сверткой складывая данные с данными соответствующей размерности из «сверточной» части.

Нейронная сеть будет состоять из следующих частей.

1. Эмбеддинг временного шага, который кодирует текущий шаг диффузии, что даст модели информацию о том, на каком этапе восстановления изображения она сейчас находится.

2. Residual block – основной блок нейронной сети, на котором модель с помощью сверток будет анализировать входные данные.

3. Модуль self-attention – данный модуль будет реализовывать механизм внимания, с помощью которого нейронная сеть будет воспринимать изображения целиком.

4. DownBlock – этот модуль является частью «сверточной» части U-Net с помощью которого она сжимает входящие данные.

5. UpBlock – модуль, реализующий «разверточную» часть U-Net.

6. MiddleBlock – блок, находящийся между «сверточной» и «разверточной» частями U-Net.

7. Downsample – блок, обеспечивающий сжатие данных вдвое с помощью механизма max pool.

8. Upsample – модуль, обеспечивающий увеличение входных данных вдвое.

Более подробное описание каждого модуля, а также реализация будут приведены в следующей главе.

Итоговая архитектура нейронной сети будет состоять из двух сверточных и разверточных блоков, между которыми будет находиться MiddleBlock. При этом в разверточной части будет использоваться SelfAttention для лучшей генерации образцов. Архитектура нейронной сети представлена на рисунке 6.

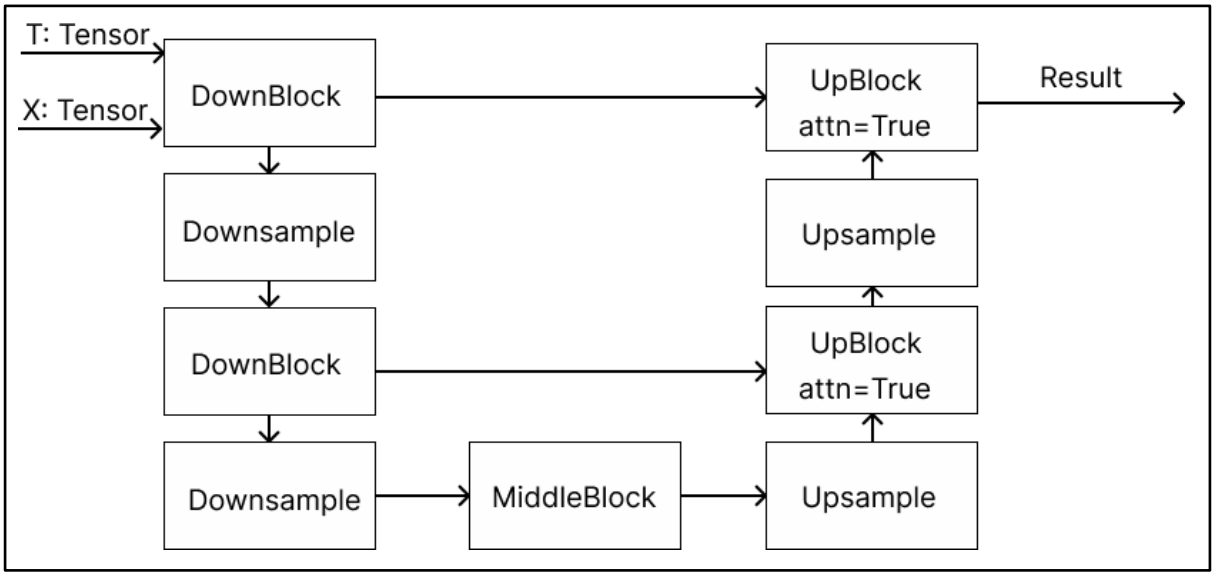

Рисунок 6 – Архитектура нейронной сети на базе U-Net

2.3. Проектирование нейросети на архитектуре «Трансформер»

Трансформер – это архитектура нейронных сетей, которая основана на механизме внимания (принцип работы которого описывался ранее) и предназначена для обработки различных последовательностей [32].

Трансформеры состоят из блоков, каждый блок содержит два основных элемента: многоголовый механизм внимания (его отличие от простого механизма внимания состоит в том, что каждая голова внимания применяет операцию self-attention к входному тензору параллельно и независимо друг от друга, после чего результаты всех голов складываются) и слоя прямой связи, который представляет собой полносвязную нейронную сеть из двух слоев.

Эта архитектура показала хорошие результаты в задачах, связанных с обработкой языка. Такие популярные большие языковые модели, как ChatGPT [3], Claude [4] и другие основаны на этой архитектуре.

Однако трансформеры применяют не только для работы с естественным языком, в работе [7] авторы показали, что данную архитектуру можно применять и для работы с изображениями, и что она превосходит используемые ранее архитектуры. Для работы с изображения как с последовательностями, перед обработкой трансформером изображения разбивают на патчи, патч – это небольшой фрагмент изображения, чем меньше размер патча, тем более качественно модели способны работать с изображением, ведь у них имеется больше информации о нем.

Раз трансформеры можно применять для анализа изображений, значит можно применить и для их генерации, сделав выход нейронной сети такой же размерности, как и вход. В статье [24] авторы применили трансформер для генерации изображений и, на момент публикации этой статьи, смогли добиться наилучшего качества генерации изображений на метрике FID [11]. Примеры изображений, сгенерированных данной нейронной сетью, представлены на рисунке 7.

Рисунок 7 – Примеры изображений, сгенерированных трансформером

В статье [24] авторы предложили несколько вариантов модели, отличающихся размером, всего было предложено 4 модели, параметры этих моделей приведены в списке ниже:

1) DiT-S – содержит 33 миллиона параметров, 12 слоев, размер скрытого представления 384, каждый attention block содержит 6 голов;

2) DiT-B – содержит 131 миллион параметров, 12 слоев, размер скрытого представления 768, каждый attention block содержит 12 голов;

3) DiT-L – содержит 459 миллионов параметров, 24 слоя, размер скрытого представления 1024, каждый attention block содержит 16 голов;

4) DiT-XL – содержит 676 миллионов параметров, 28 слоев, размер скрытого представления 1152, каждый attention block содержит 16 голов.

Также каждая модель может использовать патчи разного размера, авторы рассмотрели патчи размером 2, 4 и 8, примеры изображений, генерируемых разными моделями при разных размерах патча, приведены на рисунке 8.

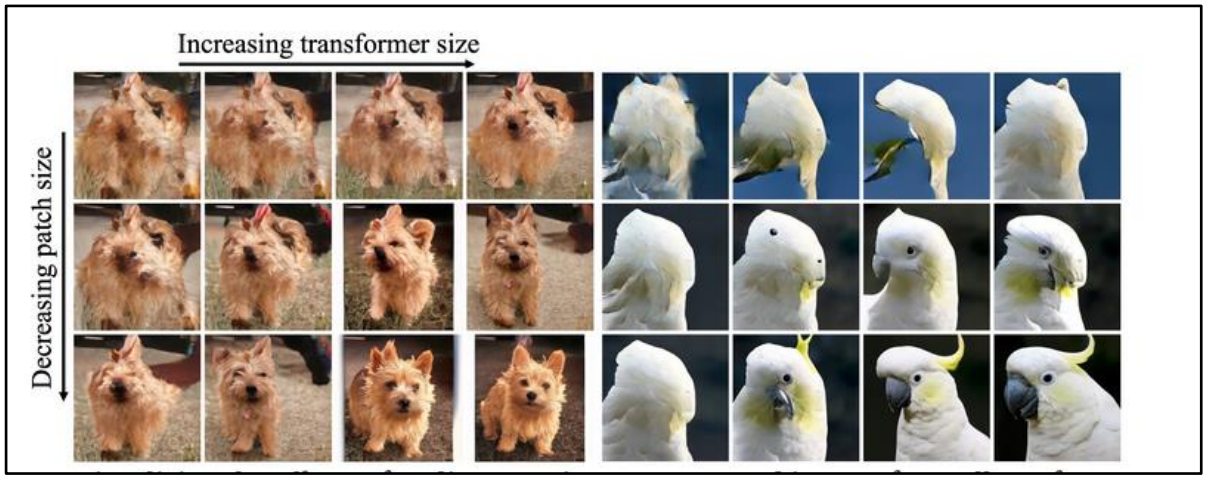

Рисунок 8 – Влияние размера патча и размера модели на изображения

Как видно из рисунка 8, даже самая большая модель при большом размере патча не способна генерировать качественные изображения. Однако, при самом маленьком патче, даже небольшие модели могут создавать образцы приемлемого качества.

В связи с ограниченностью вычислительных ресурсов, было решено использовать модель DiT-B с патчем размера 2.

2.4. Проектирование веб-сайта

Для демонстрации сгенерированных персонажей применялся веб-сайт. Он должен обеспечивать следующие сценарии взаимодействия.

1. Просмотр персонажа. На главной странице пользователь должен видеть случайного персонажа, который заранее был сгенерирован нейросетью (такое решение было принято в связи со сложностью генерации персонажа для каждого пользователя, генерация одного образца у диффузионной нейросети занимает несколько минут). Персонаж находится на фоне выбранной пользователем локации из видеоигры.

2. Прохождение опроса. Для оценки работы нейросети собирались оценки пользователей. Для того, чтобы предоставить им удобный способ прохождения опросов, была реализована веб-страница, на которой пользо-

ватели проходят опрос о качестве созданных дизайнов, во время прохождения опроса пользователи видят оцениваемого персонажа и могут изменить локацию, аналогично с главной страницей.

3. Скачивание персонажа. Пользователь имеет возможность скачать изображение с дизайном персонажа, которого он в данный момент видит, для этого ему нужно нажать на соответствующую кнопку.

Веб-сайт будет размещен на удаленном сервере по адресу https://emil.jipok.ru/.

Диаграмма прецедентов веб-сайта представлена на рисунке 9.

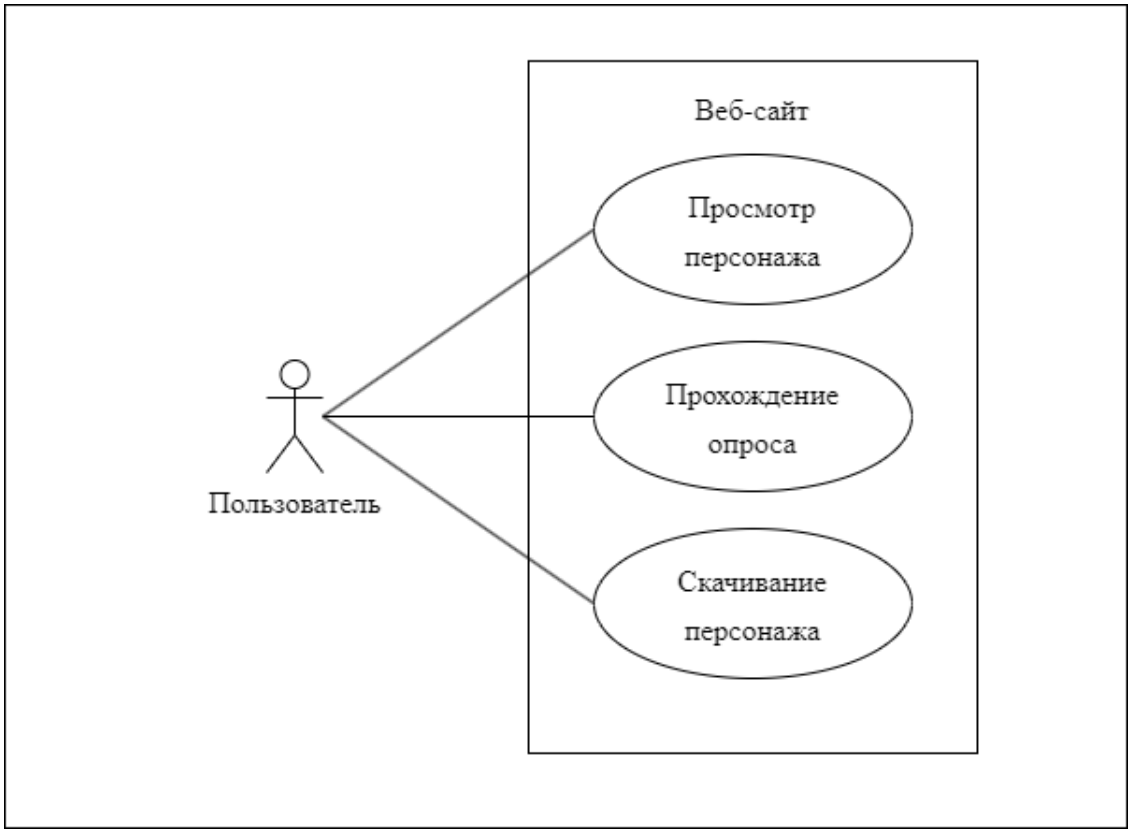

Рисунок 9 – Диаграмма прецедентов веб-сайта

Выводы по второй главе

В этой главе были описаны требования к проектируемой системе, проведено проектирование двух вариантов нейросетей, а также выбор конкретных архитектур, которые будут использованы для создания модели. Также были спроектированы и описаны сценарии взаимодействия пользователей с веб-сайтом.

3. РЕАЛИЗАЦИЯ

3.1. Средства реализации

Для реализации нейросети и веб-сайта был выбран язык программирования Python [25]. Для написания и отладки кода для реализации нейронной сети были использован Google Colaboratory [10] и Jupyter Notebook [17].

Для обучения нейронной сети изначально применялись вычислительные ресурсы, предоставляемые Google Colaboratory, однако позже обучение производилось на удаленном сервере с видеокартой AMD Radeon Instinct MI50 [2].

Для создания веб-сайта применялись: язык разметки HTML [14], язык стилей CSS [6], для написания скриптов веб-страниц использовался JavaScript [15], для написания бекэнда сайта Python, а также редактор кода Visual Studio Code [33].

В процессе разработки использовались следующие библиотеки.

PyTorch [26]

Библиотека с открытым исходными кодом для создания нейронных сетей для языка Python, созданная на базе проекта Torch. Содержит в себе множество инструментов, полезных при работе с нейросетями, в том числе и инструменты для работы с данными.

Torchvision [30]

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

Numpy [22]

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

Matplotlib [19]

Библиотека с открытым исходным кодом, предназначенная для визуализации данных.

MineRender [20]

Библиотека для языка JavaScript с открытым исходным кодом, которая предоставляет инструменты для рендера объектов из видеоигры Minecraft.

3.2. Сбор и предобработка данных

Для обучения нейронной сети применялся набор данных, содержащий более 900 000 изображений с дизайном персонажа [1]. Он содержит четырехканальные изображения в формате .png разрешения 64x64 пикселя. Пример таких изображений приведен на рисунке 10.

Рисунок 10 – Примеры корректных изображений

Однако данный набор данных содержит не только изображения с дизайном персонажа, но и некорректные изображения, примеры которых приведены на рисунке 11.

Рисунок 11 – Примеры некорректных изображений

Очистка набора данных от некорректных изображений

Очистка набора данных проводилась путем подсчета в изображениях количества непрозрачных пикселей. Как правило, в изображениях обликов 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

```
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 случайным образом выбраны 200 тысяч изображений для дальнейшего обучения нейронной сети.

3.3. Реализация и обучение нейросети на архитектуре U-Net

Нейронная сеть будет состоять из «блоков», описание каждого из них, а также реализация приведены ниже. Итоговая архитектура приведена в прошлой главе на рисунке 6.

Эмбеддинг временного шага

Учитывая, что диффузионная нейронная сеть восстанавливает данные «шаг за шагом» и на каждом из этих этапов поведение нейронной сети должно отличаться, чтобы обеспечить это будем передавать на вход нейронной сети эмбеддинг текущего шага диффузии. Для этого будем передавать тензор из одного числа (текущий шаг) на вход двухслойного перцептрона.

Код данного блока нейронной сети представлен в листинге 2.

Листинг 2 – Исходный код блока, кодирующего текущий шаг диффузии

```
class TimeEmbedding(nn.Module):
    def __init__(self, n_channels: int):
        super().__init__()
        self.n_channels = n_channels
        self.lin1 = nn.Linear(self.n_channels // 4, self.n_channels)
        self.act = nn.SiLU()
        self.lin2 = nn.Linear(self.n_channels, self.n_channels)


    def forward(self, t: torch.Tensor):
        half_dim = self.n_channels // 8

        emb = math.log(10_000) / (half_dim - 1)
        emb = torch.exp(torch.arange(half_dim, device=t.device) * -emb)
        emb = t[:, None] * emb[None, :]
        emb = torch.cat((emb.sin(), emb.cos()), dim=1)
        emb = self.act(self.lin1(emb))
        emb = self.lin2(emb)

        return emb
```

Residual block

Данный модуль состоит из двух блоков, каждый блок состоит из: слоя групповой нормализации, функции активации SiLU, слоя свертки 3x3, что позволяет модели находить зависимости между рядом стоящими пикселями, данные блоки следуют друг за другом последовательно и между ними находится DropOut слой, позволяющий уменьшить вероятность переобучения модели. На вход блок принимает тензор данных и тензор эмбеддинга временного шага. То, как этот слой взаимодействует с данными представлено на рисунке 12.

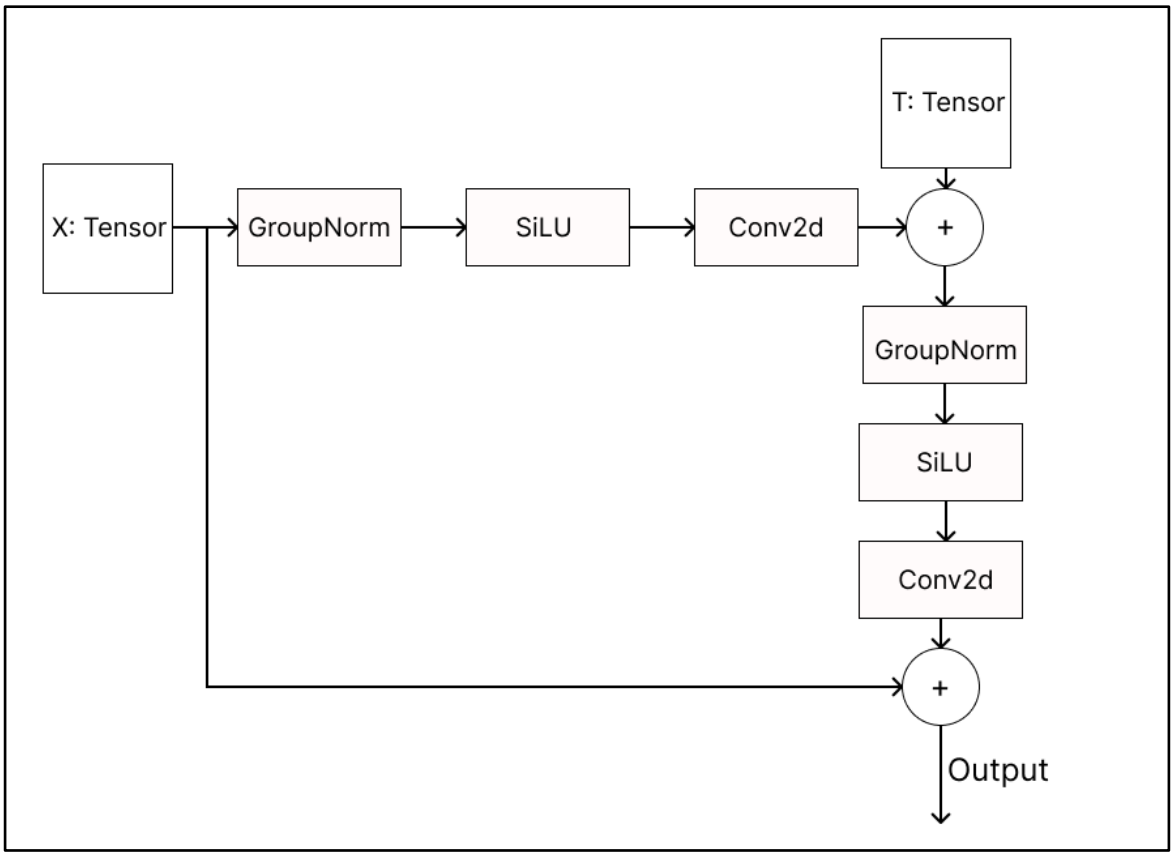

Рисунок 12 – Иллюстрация работы Residual блока

Модуль self-attention

Данный модуль реализует «механизм внимания», позволяющий искать зависимости в изображении за пределом фильтра свертки. В работе [13] авторы продемонстрировали эффективность данного модуля в диффузионных нейронных сетях для генерации изображений.

На вход механизму внимания поступает тензор признаков x, после чего, получаются три новых тензора f(x), g(x), h(x), это происходит путем умножения чисел тензора x на соответствующие обучаемые параметры). Далее, тензоры f(x) и g(x) перемножаются между собой, далее к результату применяется функция softmax, это позволяет получить attention map, который содержит информацию о «важности» каждого числа из исходного тензора, после чего, attention map умножается на тензор h(x), это позволяет выделить наиболее важные части тензора x.

Архитектура модуля self-attention и то, как он взаимодействует с входными данными, изображена на рисунке 13.

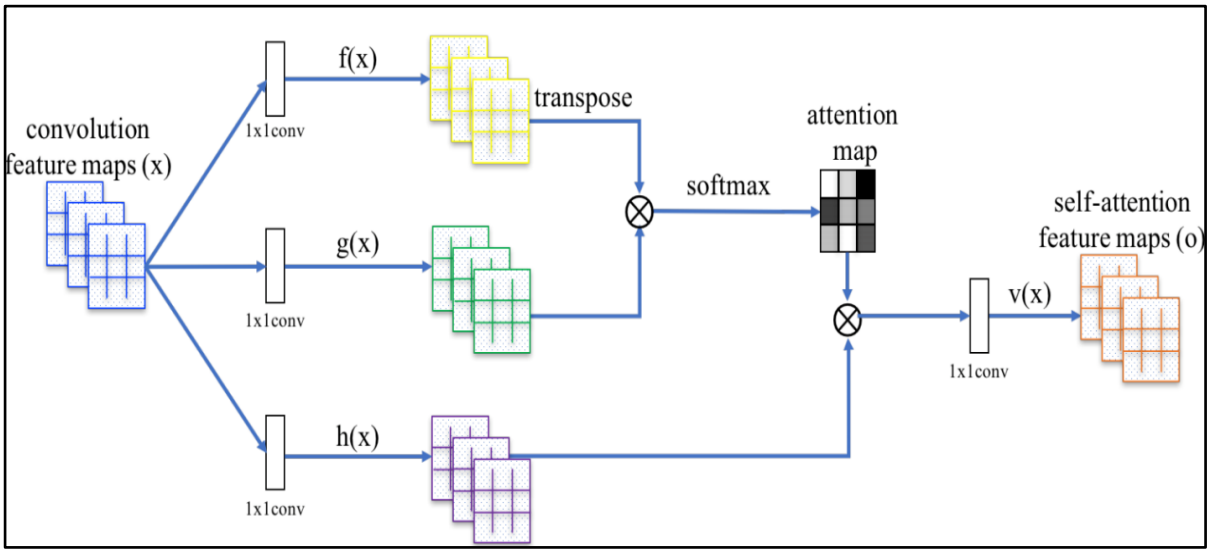

Рисунок 13 – Иллюстрация работы механизма внимания

DownBlock

Данный модуль нейронной сети реализует «сверточную» часть U-Net. На вход он принимает следующие параметры:

1) тензор входных данных x;
2) тензор эмбеддинга временного шага t.

Оба тензора являются четырехканальными с размерностью, которая соответствует текущему этапу свертки.

Данный блок состоит включает в себя `ResidualBlock`, а также может включать в себя `AttentionBlock`, если в конструктор передан `attn=True`.

Исходный код данного блока представлен в листинге 3.

Листинг 3 – Исходный модуля `DownBlock`

```
class DownBlock(nn.Module):
    def __init__(self, in_channels: int, out_channels: int, time_channels:
int, attn: bool):
        super().__init__()
        self.res = ResidualBlock(in_channels, out_channels, time_channels)
        if attn:
            self.attn = AttentionBlock(out_channels)
        else:
            self.attn = nn.Identity()
    def forward(self, x: torch.Tensor, t: torch.Tensor):
        x = self.res(x, t)
        x = self.attn(x)
        return x
```

UpBlock

Данный модуль нейронной сети реализует «разверточную» часть U-Net. На вход он принимает следующие параметры:

1) тензор входных данных x;
2) тензор эмбеддинга временного шага t.

В отличие от `DownBlock` тензор `x` является восьмиканальным, четыре канала в него приходят из предыдущего слоя, и еще четыре из соответствующего блока «сверточной» части. Тензор `t` является четырехканальным.

Данный блок включает в себя `ResidualBlock`, а также может включать в себя `AttentionBlock`, если в конструктор передан `attn=True`.

Исходный код данного блока представлен в листинге 4.

Листинг 4 – Исходный модуля `UpBlock`

```
class UpBlock(nn.Module):
    def __init__(self, in_channels: int, out_channels: int, time_channels:
int, has_attn: bool):
        super().__init__()
        self.res = ResidualBlock(in_channels + out_channels, out_channels,
time_channels)
        if has_attn:
            self.attn = AttentionBlock(out_channels)
        else:
            self.attn = nn.Identity()


    def forward(self, x: torch.Tensor, t: torch.Tensor):
        x = self.res(x, t)
        x = self.attn(x)
        return x
```

MiddleBlock

Данный модуль нейронной сети реализует часть U-Net, которая находится между сверточной и разверточной частями. На вход он принимает следующие параметры:

1) тензор входных данных `x`;

2) тензор эмбеддинга временного шага `t`.

Оба тензора являются четырехканальными с размерностью, которая соответствует текущему этапу свертки. Данный блок включает в себя `ResidualBlock`, `AttentionBlock` и еще один `ResidualBlock`.

Исходный код данного блока представлен в листинге 5.

Листинг 5 – Исходный модуля `UpBlock`

```
class MiddleBlock(nn.Module):
    def __init__(self, n_channels: int, time_channels: int):
        super().__init__()
        self.res1 = ResidualBlock(n_channels, n_channels, time_channels)
        self.attn = AttentionBlock(n_channels)
        self.res2 = ResidualBlock(n_channels, n_channels, time_channels)

    def forward(self, x: torch.Tensor, t: torch.Tensor):
        x = self.res1(x, t)
        x = self.attn(x)
        x = self.res2(x, t)
        return x
```

Downsample

Данный блок вдвое сжимает входящие данные по высоте и ширине, вместо пуллинга в нем используется свертка со следующими параметрами:

1) размер ядра равный 3;

2) шаг свертки равный 2 (для сжатия изображения вдвое);

3) padding равный 1, для того, чтобы свертка не изменяла размера изображения, перед тем, как сжать его.

Upsample

Данный блок вдвое увеличивает ширину и высоту входящих данных, вместо пуллинга в нем используется транспонированная свертка со следующими параметрами:

1) размер ядра равный 4;

2) шаг свертки равный 2 (для увеличения изображения вдвое);

3) padding равный 1, для того чтобы свертка не изменяла размер изображения перед тем, как сжать его.

Для дальнейшей работы с нейронной сетью были реализованы следующие классы.

GaussianDiffusionTrainer

Для обучения нейронной сети был реализован класс `GaussianDiffusionTrainer`. Его конструктор принимает следующие параметры:

1) `model`: объект модели, которую этот класс будет обучать;

2) `T = 1000`: количество шагов диффузии;

3) `beta1= 1e-4`: определяет количество шума, которое будет наложено на первом шаге диффузии;

4) `betaT = 0.02`: определяет количество шума, которое будет наложено на последнем шаге диффузии.

При этом количество шума, которое будет накладываться на предыдущих шагах распределено равномерно между `beta1` и `betaT`.

Обучение будет происходить следующим образом:

1) случайным образом выбирается шаг диффузии (между 1 и `T`) для текущей итерации;

2) на изображение накладывается шум, соответствующий шагу диффузии;

3) на изображение еще раз накладывается шум для одного шага диффузии;

4) на вход модели подается зашумленное изображение с шага 3 и получается предсказание наложенного на нее шума;

5) вычисляется функция потерь (в данном случае средняя квадратичная ошибка), между результатом работы на шаге 4 и наложенным на шаге 3 шумом.

Дальнейшее обучение происходило со следующими параметрами:

1) оптимизатор Adam;
2) скорость обучения равна 1e-4;
3) dropout = 0,1;
4) регуляризация не применялась;
5) размер пакета равнен 80.

Обучающая выборка была случайным образом поделена на подвыборки по десять тысяч изображений. Через каждые 10 эпох сохранялась контрольная точка обучения. В течение первых ста эпох использовались десять подвыборок, на каждой из которых модель обучалась по 10 эпох. Объясняется это тем, что изначально обучение велось в Google Colaboratory, в котором есть ограничение на используемое количество файлов из диска.

График функции потерь до 270 эпохи представлен на рисунке 14.

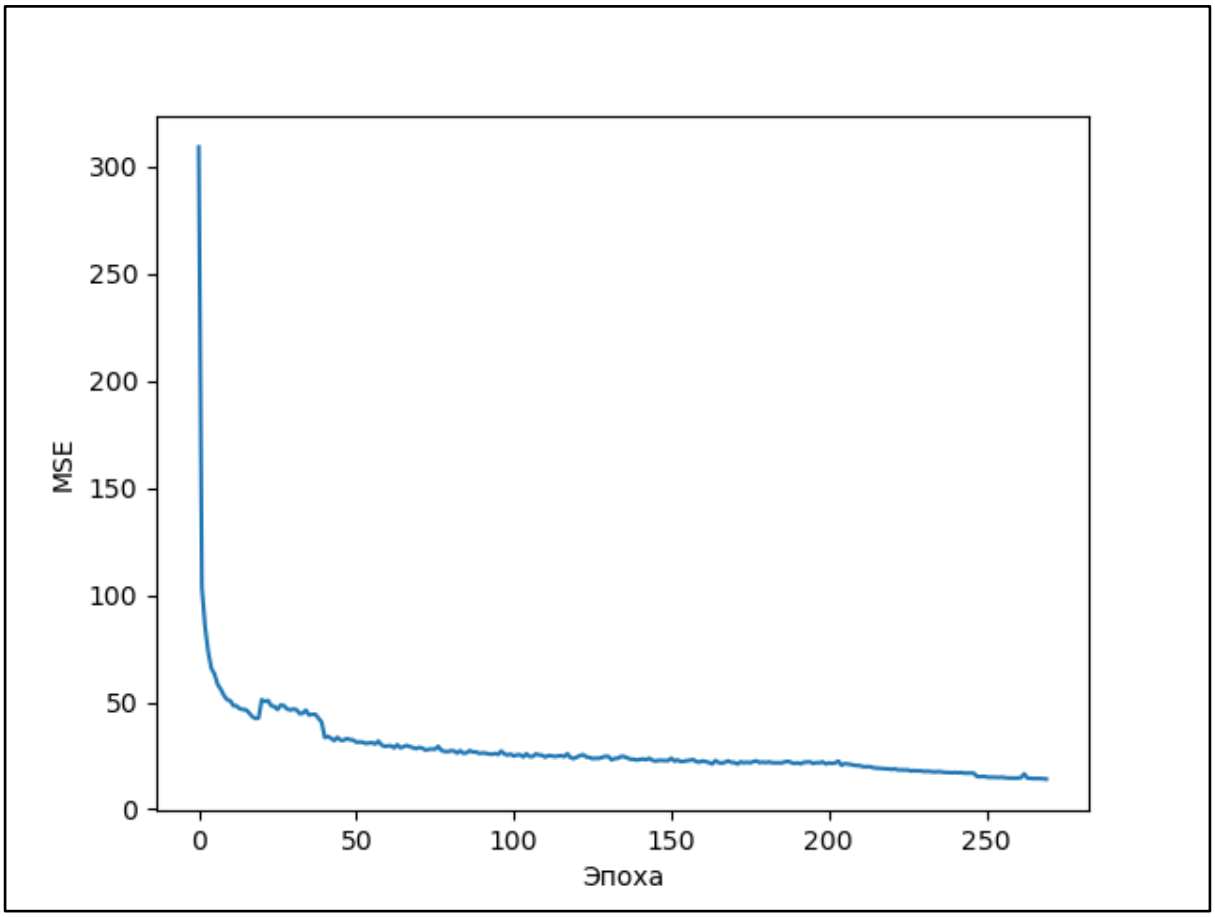

Рисунок 14 – График функции потерь

Генерируемые образцы на разных эпохах представлены на рисунке 15.

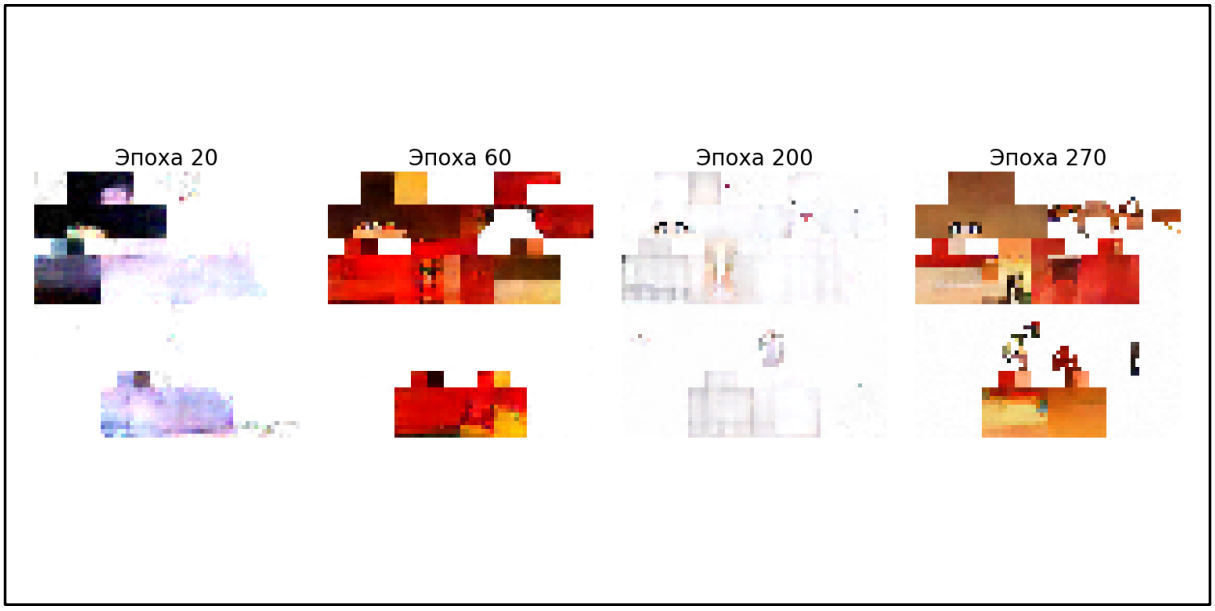

Рисунок 15 – Генерируемые нейросетью образцы

На 270 эпохе модель стала создавать достаточно качественны образцы, однако они очень похожи на те, что представлены в обучающей выборке, пример такого дизайна, непосредственно на персонаже представлен на рисунке 16. Голова сгенерированного персонажа в точности повторяет стандартный дизайн персонажа из видеоигры.

Рисунок 16 – Пример переобучения

Чтобы нейросеть генерировала более уникальные образцы были изменены некоторые параметры, а именно: добавлен коэффициент регуляризации 1e-4, dropout был повышен до 0,45, это должно предотвратить переобучение модели. Дальнейшее обучение продолжалось вплоть до 400 эпохи. График функции потерь представлен на рисунке 17.

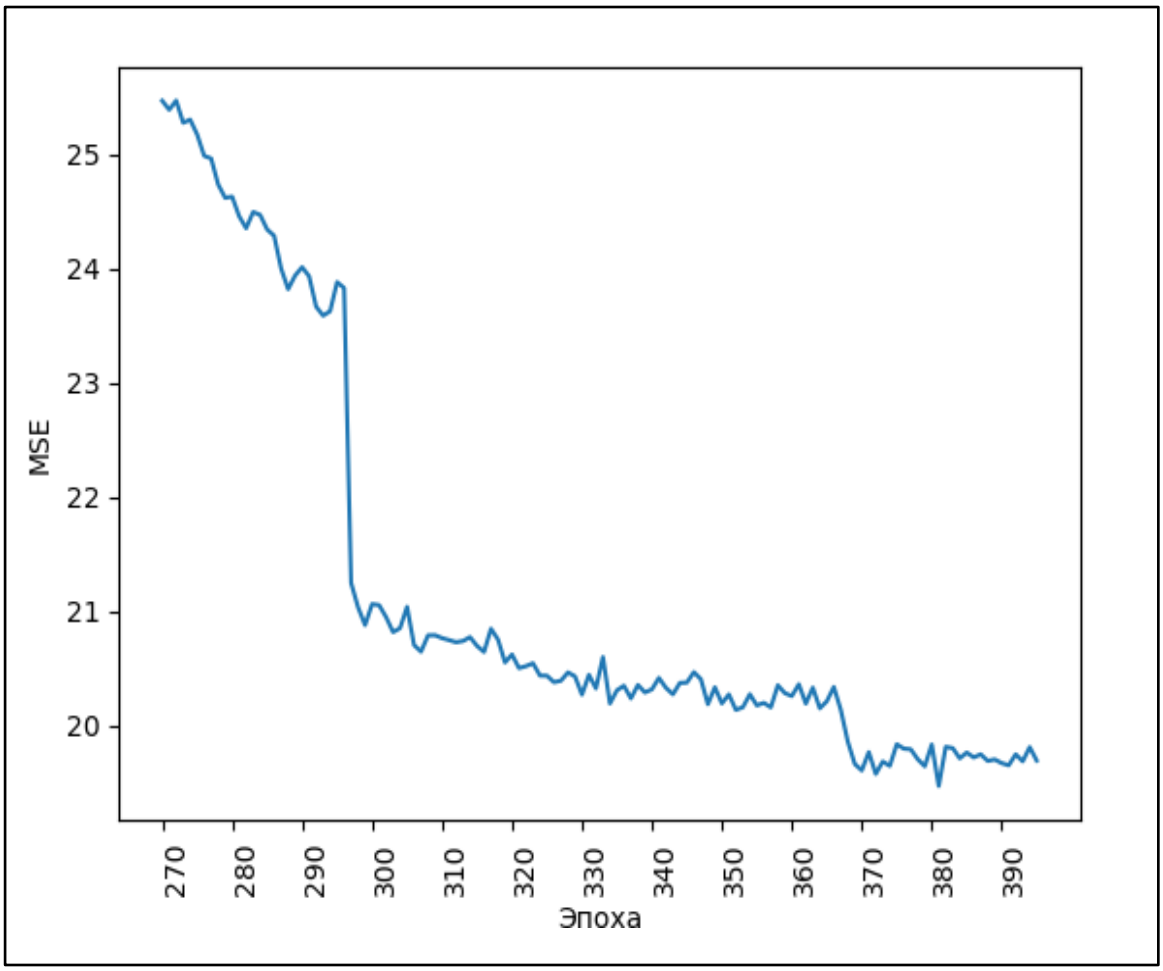

Рисунок 17 – График функции потерь с 270 эпохи

Как видно из графика, значение функции потерь продолжает уменьшаться, что может свидетельствовать о том, что модель способна обучаться и дальше, улучшая свои результаты.

В результате обучения нейросеть способна генерировать образцы приемлемого качества, они приведены на рисунке 18.

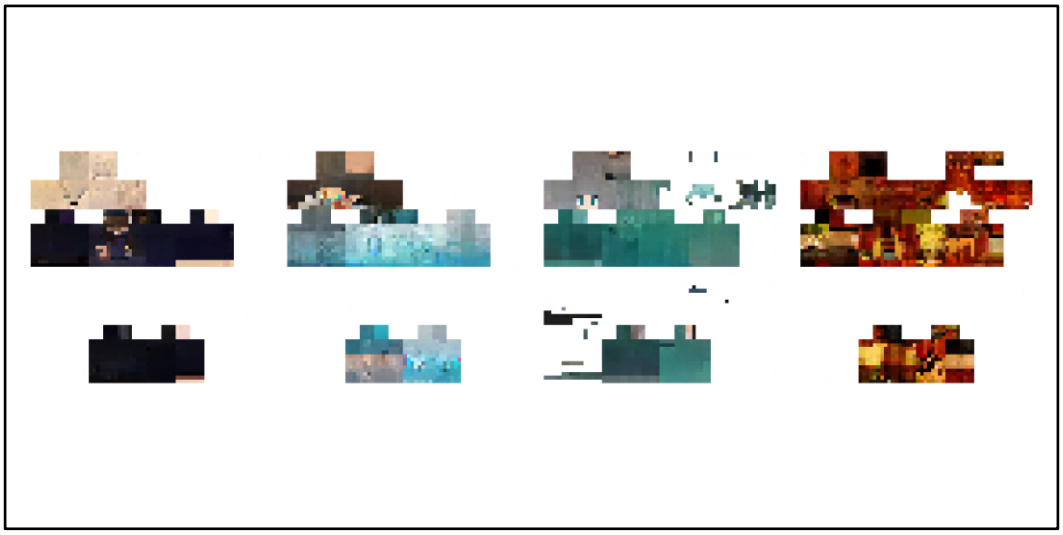

Рисунок 18 – Генерируемые нейросетью дизайны персонажей

Получаемые результаты из нейронной сети еще можно улучшить. Увеличим количество шагов диффузии до 1384, параметр beta1 уменьшим до 0,00008, а betaT до 0,015. Параметры beta задают скорость удаления шума с изображения, для более быстрой работы их можно увеличить, а для более точной и качественной генерации их следует уменьшить, в сочетании с увеличением количества шагов диффузии это должно сделать образцы более качественными.

После этого производилось дообучение модели в течение 10 эпох для ее адаптации к новым значениям параметров. Генерируемые после этого образцы представлены на рисунке 19.

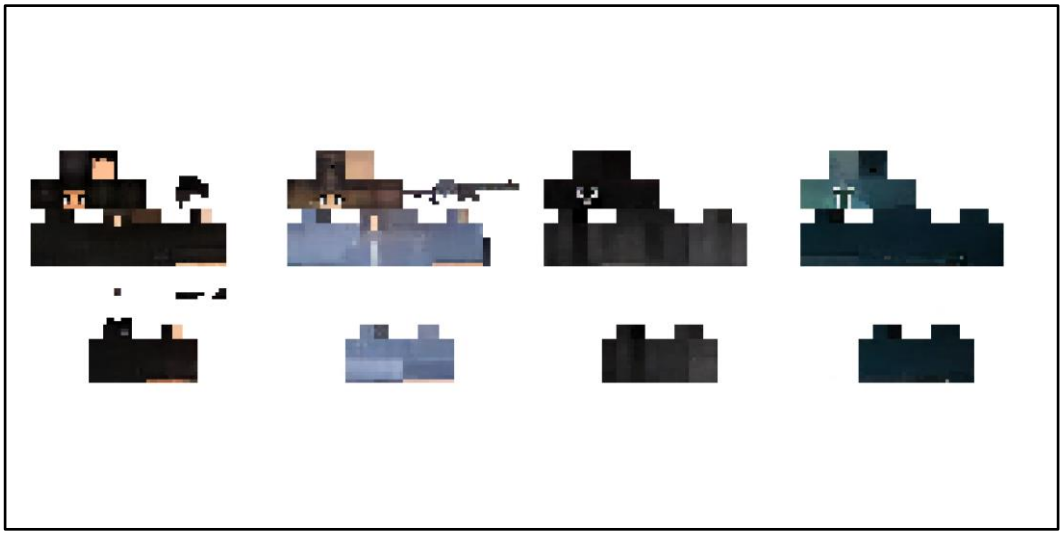

Рисунок 19 – Генерируемые после дообучения образцы

Как видно из сгенерированных образцов, их качество возросло, у персонажей появилось четко различимое лицо и их дизайн в целом выглядит более целостным.

3.4. Обучение нейросети на архитектуре «Трансформер»

В связи с большой сложностью реализации такой нейронной сети и процедуры обучения данной нейронной сети, использовалась готовая реализация из репозитория на GitHub [8].

Данную реализацию рекомендуют авторы работы [24], в связи с тем, что она эффективнее чем их собственная.

Созданные в данном репозитории модели, соответствуют моделям, реализованным в оригинальной статье. Для дальнейшего обучения была использована архитектура DiT-B с размером патча 2.

Дальнейшее обучение происходило со следующими параметрами:

1) оптимизатор Adam;
2) скорость обучения равна 1e-4;
4) регуляризация не применялась;
5) размер пакета равен 24;
6) количество шагов диффузии равно 1000.

Обучение модели происходило на удаленном сервере с GPU AMD Radeon Instinct MI50. Для обучения использовалось 200 тысяч изображений. Обучение проводилось в течение 10 эпох, каждая из который содержала 10 тысяч итераций, каждые 5 тысяч итераций происходило сохранение контрольной точки обучения, что обеспечивает возможность продолжить обучение нейронной сети, после завершения заданного количества эпох.

График функции потерь представлен на рисунке 20.

Как видно из графика, довольно быстро функция потерь стала близка к нулю, однако на некоторых пакетах происходили скачки функции потерь, со временем частота и значение таких всплесков уменьшились, что говорит

о том, что модель улучшала свою работу. Однако эти всплески все еще довольно высоки в сравнении с нулем, значит можно сделать вывод о недообученности этой модели.

Рисунок 20 – График функции потерь при обучении трансформера

Данная реализация, поддерживает изменение количества шагов диффузии уже после обучения, без необходимости дообучать нейронную сеть, однако, использовать больше шагов диффузии, чем при обучении, невозможно.

Поэтому дальнейшая генерация производилась в течение 250 шагов диффузии, это незначительно снижает качество, однако значительно ускоряет процесс.

Генерируемые нейросетью образцы представлены на рисунке 21.

Как видно из генерируемых изображений, в целом у них всех имеется четко различимое лицо, в рамках одного персонажа не наблюдается неуместных деталей и выбивающихся из общей картины частей, одежда каждого персонажа создана в одном стиле.

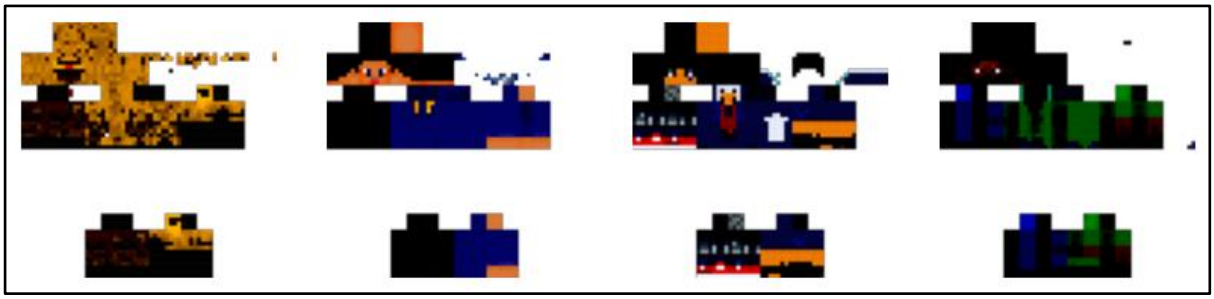

Рисунок 21 – Генерируемые трансформером образцы

На рисунке 22 представлен сгенерированный дизайн непосредственно на персонаже.

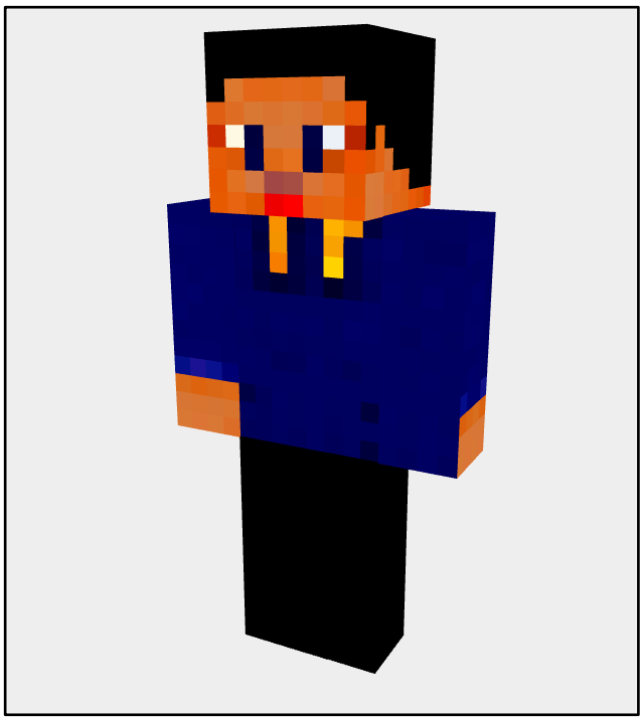

Рисунок 22 – Пример сгенерированного персонажа

3.5. Реализация веб-сайта

В прошлой главе были представлены сценарии взаимодействия пользователей с веб-сайтом, в этом разделе приведено более детальное описание имеющихся на сайтах страниц и их функционала, также приведены исходные коды некоторых функций.

Главная страница

При переходе на главную страницу пользователь видит одного из заранее созданных нейросетью персонажей, благодаря библиотеке MineRender пользователь может рассмотреть персонажа со всех сторон. Также у пользователя на данной странице есть возможность выбрать один из предложенных фонов, на котором будет находиться персонаж для этого пользователь должен выбрать нужный ему фон из списка.

Исходный код функции, которая обрабатывает посещение пользователем главной страницы представлен в листинге 6.

Листинг 6 – Исходный код обработчика посещения главной страницы

```
def _serve_index(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()

        with open('index.html', 'r', encoding='utf-8') as index_file:
            content = index_file.read()

        random_png = random.choice(os.listdir('skins'))
        content = content.replace('{%path%}', random_png)

        self.wfile.write(content.encode())
```

Как видно из кода, при посещении страницы сервер отправляет в ответ html-страницу, которая содержит изображение с персонажем, который случайным образом выбирается из папки `skins`.

Прохождение опроса

При попадании на эту страницу, пользователь видит персонажа, которого ему потребуется оценить и форму с критериями, по которым пользователь будет оценивать персонажей. Оценка производилась по 10-ти бальной шкале.

При оценивании использовались следующие критерии: «качество лица», «схожесть с созданными людьми персонажами», «четкость отрисовки отдельных частей» и «согласованность частей между собой».

Пользователи оценивали 16 созданных персонажей. Их оценки сохранялись на сервере в виде .csv файлов. Для предотвращения оценивания одного и того же персонажа пользователем дважды использовались cookie-файлы, содержащие информацию об оцененных дизайнах, по завершению опроса, пользователю выводится страница с благодарностью, которая позже перенаправляет его на главную. После прохождения опроса пользователи не могут повторно пройти опрос.

Исходный код функции, которая обрабатывает попадание пользователя на страницу с опросом представлен в листинге 7.

Листинг 7 – Исходный код обработчика посещения страницы с опросом

```
def _serve_survey(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        cookie_header = self.headers.get('Cookie')
        if cookie_header:
            cookies = http.cookies.SimpleCookie(cookie_header)
        else:
            cookies = http.cookies.SimpleCookie()
        if isinstance(cookies.get('current_skin', '0'), str):
            current_skin = int(cookies.get('current_skin', '0'))
        else:
            current_skin = int(cookies.get('current_skin', '0').value)

        if current_skin > 15:
            self._serve_thank_you_page()
            return

        cookies['current_skin'] = str(current_skin)
        cookies['current_skin']['path'] = "/"
        cookies['current_skin']['Max-Age'] = str(60*60*24*7)
        cookie_header = cookies.output(header='', sep=';')
        self.send_header("Set-Cookie", cookie_header)

        image_name = f"{current_skin}.png"
        # Serve the HTML page with the current image, no params mean regu-
lar GET request

        with open('survey.html', 'r', encoding='utf-8') as index_file:
            content = index_file.read()

        content = content.replace('{%path%}', urllib.parse.quote(im-
age_name))
        self.wfile.write(content.encode())
```

При оценке персонажей, пользователи заполняли форму с критериями оценки, данная форма обеспечивает невозможность ввода некорректных

значений. Результат оценки обрабатывается с помощью POST-запроса к серверу. Код обработчика данного запроса представлен в листинге 8.

Листинг 8 – Исходный код обработчика POST-запроса

```
def do_POST(self):
        if self.path.startswith('/submit-survey'):
            content_length = int(self.headers['Content-Length'])
            post_data = self.rfile.read(content_length).decode('utf-8')
            post_data = urllib.parse.parse_qs(post_data)

            cookies = http.cookies.SimpleCookie(self.headers.get('Cookie'))
            # Если 'current_skin' есть в куки, то обновляем его, иначе
начинаем с нуля
            current_skin = int(cookies.get('current_skin', '0').value) if
'current_skin' in cookies else 0

            image_name = f"{current_skin}.png"

            self.record_survey_data(image_name, post_data)
            if current_skin > 15:
                self._serve_thank_you_page()
                return
            cookies['current_skin'] = str(current_skin + 1)
            cookies['current_skin']['path'] = "/"
            cookies['current_skin']['Max-Age'] = str(60*60*24*7)
            cookie_header = cookies.output(header='', sep=';')
            self.send_response(303)
            self.send_header("Location", f"/survey")
            self.send_header("Set-Cookie", cookie_header)
            self.end_headers()
        else:
            self.send_error(404, "Not Found")
```

Скачивание персонажа

Для скачивания персонажа, пользователю достаточно нажать соответствующую кнопку на любой из двух страниц веб-сайта. После чего на компьютер пользователя будет скачано изображение формата .png с дизайном персонажа, данный файл пользователь может использовать для загрузки в игру для изменения дизайна своего персонажа.

Выводы по третьей главе

В ходе данной главы были описаны инструменты реализации, проведен сбор и предобработка данных, реализованы и обучены две нейронные сети, которые генерируют дизайн персонажа видеоигры Minecraft, дополнительно приведена реализация и описание модулей архитектуры U-Net. Также создан веб-сайт и приведено описание имеющегося на сайте функционала.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование веб-сайта

Для определения корректности работы веб-сайта проводилось тестирование, протокол тестирования представлен в таблице 1.

Таблица 1 – Протокол тестирования веб-сайта

№	Название теста	Входные данные	Ожидаемый результат	Тест пройден?
1.	Проверка главной страницы сайта	Пользователь заходит на главную страницу сайта	Пользователь видит случайно выбранного из числа заранее сгенерированных персонажа	Да
2.	Проверка выбора фона при просмотре персонажа	Пользователь при просмотре персонажа меняет фон, используя форму	Фон персонажа меняется в соответствии с выбранный пользователем	Да
3.	Проверка страницы с опросом о качестве сгенерированных персонажей	Пользователь заходит на страницу с опросом для оценки сгенерированных персонажей	Пользователь видит персонажа, которого ему предлагается оценить и форму с критериями	Да
4.	Проверка невозможности ввода некорректных результатов в форме	Пользователь пытается ввести в форму неверные данные	Ввод в форму оказывается неудачным, пользователь получает сообщение о верном формате данных	Да
5.	Проверка сохранения оценки пользователя в .csv файл	Пользователь вводит в форму все нужные данные, и нажимает кнопку «оценить»	Оценки пользователя сохраняются в .csv файл на сервере	Да
6.	Проверка завершения опроса	Пользователь оценивает последнего персонажа из списка	Пользователь перенаправляется на страницу с благодарностью	Да
7.	Проверка попытки повторно пройти опрос	Пользователь пытается попасть на страницу с опросом после его полного прохождения	Пользователь перенаправляется на страницу с благодарностью	Да

Окончание таблицы 1

№	Название теста	Входные данные	Ожидаемый результат	Тест пройден?
8.	Проверка продолжения опроса	Пользователь продолжает опрос с того персонажа, на котором он прервался	Пользователь продолжает опрос с того персонажа, на котором он прервался	Да
9.	Проверка скачивания файла с дизайном	Пользователь нажал на кнопку «Скачать скин!»	Файл с дизайном загружается на компьютер пользователя	Да

Как видно из протокола тестирования, все тесты пройдены успешно, что свидетельствует о корректной работе протестированного функционала.

4.2. Результаты опроса среди пользователей

Для оценки качества созданных персонажей был проведен опрос среди людей, знакомых с данной видеоигрой. В процессе опроса пользователи оценивали заранее сгенерированных персонажей, которые были одинаковыми у всех из них. Результаты опроса приведены ниже.

Результаты опроса о персонажах, созданных U-Net

Каждый пользователь имел возможность оценить 16 персонажей, разумеется, не все пользователи прошли опрос до конца и всего было получено 292 оценки.

Средние значение критериев по результатам опроса следующие:

1) «качество лица» – 5,28;
2) «схожесть с созданными людьми скинами» – 5,92;
3) «четкость отрисовки отдельных частей» – 5,22;
4) «согласованность частей между собой» – 5,95.

Как видно из результатов, нейронная сеть на архитектуре U-Net способна генерировать персонажей приемлемого, по мнению опрошенных качества, однако нейронная сеть на данной архитектуре довольно далека от идеала.

Результаты опроса о персонажах, созданных трансформером

Как и в случае с U-Net, пользователям предлагалось оценить 16 персонажей. Всего было получено 234 оценки, также как и в прошлом опросе, не все пользователи прошли его до конца.

Средние значение критериев по результатам опроса следующие:

1) «качество лица» – 6,35;
2) «схожесть с созданными людьми скинами» – 6,59;
3) «четкость отрисовки отдельных частей» – 6,32;
4) «согласованность частей между собой» – 7,04.

Как видно из результатов, нейронная сеть на архитектуре «Трансформер» способна генерировать персонажей довольно высокого, по мнению опрошенных качества, полученные результаты также указывают на то, что данная нейронная сеть, по мнению пользователей, превосходит нейросеть на архитектуре U-Net.

Выводы по четвертой главе

В этой главе было проведено тестирование работоспособности сайта, все тесты были успешно пройдены, что свидетельствует о работоспособности протестированного функционала сайта. Также был проведен опрос среди пользователей о качестве генерируемых персонажей, в ходе опроса выяснилось, что трансформер показывает лучшие результаты в сравнении с U-Net, в среднем на 1 балл. Всего в опросе было получено 526 оценок. Большее количество пользователей поучаствовало в опросе о U-Net.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была исследована возможность генерации дизайнов персонажей для видеоигры Minecraft с помощью диффузионных нейросетей.

В ходе работы были решены следующие задачи.

1. Проведен анализ предметной области.
2. Осуществлен сбор и предобработка данных для нейронной сети.
3. Построены две архитектуры нейронной сети.
4. Обучены две нейронные сети.
5. Оценены результаты работы нейронных сети.
6. Улучшена работа нейронной сети.
7. Создан веб-сайт, который предоставляет пользователям возможность просмотра сгенерированных персонажей.

В результате опроса пользователей, было определено, что нейросеть на архитектуре «трансформер» генерирует образцы более высокого качества, что можно объяснить способностью данной архитектуры лучше воспринимать части изображения как единое целое, ведь self-attention воспринимает все изображение целиком, в отличие от свертки, которая работает лишь в пределах ядра.

Также, нейросеть на трансформере училась куда меньше, чем нейросеть на U-Net, учитывая, что при этом ее результаты оказались лучше, можно сделать вывод о том, что трансформер более перспективная архитектура для генерации изображений, чем U-Net.

В дальнейшем можно улучшить качество работы полученной системы с помощью более тщательного отбора персонажей для обучающей выборки, использования модели с большим количеством параметров, также возможно использовать трансформеры в паре с вариационным автокодировщиком, который позволит модели генерировать изображения в сжатом виде, за счет чего можно достигнуть экономии вычислительных ресурсов.

ЛИТЕРАТУРА

1. AcademicTorrents. [Электронный ресурс] URL: https://academictorrents.com/details/14cf27fca7f26714d2a5193dc95348a4712cdcdf (дата обращения: 25.05.2024 г.).

2. AMD Radeon Instinct MI50. [Электронный ресурс] URL: https://www.amd.com/system/files/documents/radeon-instinct-mi50-datasheet.pdf (дата обращения: 25.05.2024 г.).

3. ChatGPT. [Электронный ресурс] URL: https://openai.com/chatgpt/ (дата обращения: 25.05.2024 г.).

4. Claude. [Электронный ресурс] URL: https://www.anthropic.com/claude (дата обращения: 25.05.2024 г.).

5. Cohen G. Generative Adversarial Networks. [Электронный ресурс] // arXiv.org. 2022. Дата обновления: 01.03.2022 г. URL: https://arxiv.org/abs/2203.00667 (дата обращения: 25.05.2024 г.).

6. CSS: Cascading Style Sheets. [Электронный ресурс] URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата обращения: 25.05.2024 г.).

7. Dosovitskiy A., Beyer L., Kolesnikov A., Weissenborn D., Zhai X., Unterthiner T., Dehghani M., Minderer M., Heigold G., Gelly S., Uszkoreit J., Houlsby N. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. [Электронный ресурс] // arXiv.org 2020. Дата обновления: 03.06.2021 г. URL: https://arxiv.org/abs/2010.11929 (дата обращения: 25.05.2024 г.).

8. Fast-DiT. [Электронный ресурс] URL: https://github.com/chuanyangjin/fast-DiT (дата обращения: 25.05.2024 г.).

9. Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y. Generative Adversarial Networks. [Электронный ресурс] // arXiv.org. 2014. Дата обновления: 10.06.2014 г. URL: https://arxiv.org/abs/1406.2661 (дата обращения: 25.05.2024 г.).

10. Google Colaboratory. [Электронный ресурс] URL: https://colab.research.google.com/ (дата обращения: 25.05.2024 г.).

11. Heusel M., Ramsauer H., Unterthiner T., Nessler B., Hochreiter S. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. [Электронный ресурс] // arXiv.org 2017. Дата обновления: 12.01.2018 г. URL: https://arxiv.org/abs/1706.08500 (дата обращения: 25.05.2024 г.).

12. Ho J., Jain A., Abbeel P. Denoising Diffusion Probabilistic Models. [Электронный ресурс] // arXiv.org. 2020. Дата обновления: 16.12.2020 г. URL: https://arxiv.org/abs/2006.11239 (дата обращения: 25.05.2024 г.).

13. Hong S., Lee G., Jang W., Kim S. Improving Sample Quality of Diffusion Models Using Self-Attention Guidance. [Электронный ресурс] // arXiv.org 2022. Дата обновления: 24.08.2023 г. URL: https://arxiv.org/abs/2210.00939 (дата обращения: 25.05.2024 г.).

14. HTML: HyperText Markup Language. [Электронный ресурс] URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата обращения: 25.05.2024 г.).

15. JavaScript. [Электронный ресурс] URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата обращения: 25.05.2024 г.).

16. Jupyter notebook Documentation. [Электронный ресурс] URL: https://docs.jupyter.org/en/latest/ (дата обращения: 25.05.2024 г.).

17. Jupyter. [Электронный ресурс] URL: https://jupyter.org/ (дата обращения: 25.05.2024 г.).

18. Ma. S., Zhang X., Jia C., Zhao Z., Wang S., Wang S. Image and Video Compression with Neural Networks: A Review. [Электронный ресурс] // arXiv.org 2019. Дата обновления: 10.04.2019 г. URL: https://arxiv.org/abs/1904.03567 (дата обращения: 25.05.2024 г.).

19. Matplotlib Documentation. [Электронный ресурс] URL: https://matplotlib.org/stable/index.html (дата обращения: 25.05.2024 г.).

20. MineRender. [Электронный ресурс] URL: https://github.com/InventivetalentDev/MineRender (дата обращения: 25.05.2024 г.).

21. Noraset T., Liang, C., Birnbaum L., Downey D. Definition Modeling: Learning to define word embeddings in natural language. [Электронный ресурс] // arXiv.org. 2016. Дата обновления: 01.12.2016 г. URL: https://arxiv.org/abs/1612.00394 (дата обращения: 25.05.2024 г.).

22. NumPy. [Электронный ресурс] URL: https://www.numpy.org/ (дата обращения: 25.05.2024 г.).

23. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Bio-medical Image Segmentation. [Электронный ресурс] // arXiv.org 2015. Дата обновления: 18.05.2015 г. URL: https://arxiv.org/abs/1505.04597 (дата обращения: 25.05.2024 г.).

24. Peebles W., Xie S. Scalable Diffusion Models with Transformers. [Электронный ресурс] // arXiv.org 2022 г. Дата обновления: 02.03.2023. URL: https://arxiv.org/abs/2212.09748 (дата обращения: 25.05.2024 г.).

25. Python Documentation. [Электронный ресурс] URL: https://docs.python.org/3.10/ (дата обращения: 25.05.2024 г.).

26. PyTorch Documentation. [Электронный ресурс] URL: https://pytorch.org/docs/stable/index.html (дата обращения: 25.05.2024 г.).

27. Sohl-Dickstein J., Weiss E., Maheswaranathan N., Ganguli S. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. [Электронный ресурс] // arXiv.org 2015. Дата обновления: 18.11.2015 г. URL: https://arxiv.org/abs/1503.03585 (дата обращения: 25.05.2024 г.).

28. Sportskeeda. [Электронный ресурс] URL: https://www.sportskeeda.com/minecraft/what-minecraft-s-current-player-count-2022 (дата обращения: 25.05.2024 г.).

29. The MNIST database. [Электронный ресурс] URL: http://yann.lecun.com/exdb/mnist/ (дата обращения: 25.05.2024 г.).

30. Torchvision Documentation. [Электронный ресурс] URL: https://pytorch.org/vision/stable/index.html (дата обращения: 25.05.2024 г.).

31. Understanding Variational Autoencoders (VAEs). [Электронный ресурс] URL: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 (дата обращения: 25.05.2024 г.).

32. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., Polosukhin I. Attention Is All You Need. [Электронный ресурс] // arXiv.org 2017. Дата обновления: 02.08.2023 г. URL: https://arxiv.org/abs/1706.03762 (дата обращения: 25.05.2024 г.).

33. Visual Studio Code. [Электронный ресурс] URL: https://code.visualstudio.com/ (дата обращения: 25.05.2024 г.).

34. Vivek Narayanaswamy, Jayaraman J. Thiagarajan, Rushil Anirudh, Andreas Spanias. Unsupervised Audio Source Separation using Generative Priors. [Электронный ресурс] // arXiv.org 2020. Дата обновления: 28.05.2020 г. URL: https://arxiv.org/abs/2005.13769 (дата обращения: 25.05.2024 г.).

35. Weiling M., Kingma D. Auto-Encoding Variational Bayes. [Электронный ресурс] // arXiv.org 2013. Дата обновления: 10.12.2022 г. URL: https://arxiv.org/abs/1312.6114 (дата обращения: 25.05.2024 г.).

36. What is Minecraft skin? [Электронный ресурс] URL: https://www.minecraft.net/en-us/article/what-is-minecraft-skin (дата обращения: 25.05.2024 г.).

37. Yang L.C., Chou S., Yang Y. MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation. [Электронный ресурс] // arXiv.org 2017. Дата обновления: 18.07.2017 г. URL: https://arxiv.org/abs/1703.10847 (дата обращения: 25.05.2024 г.).

38. Брынза А.А. Применение вариационного автокодировщика для имитации изображений. // Прикладная математика и информатика: современные исследования в области естественных и технических наук, Тольятти, 23–25 апреля 2020 – С. 199–203.

39. Бычков А.В. Алгоритмы синтеза изображений в больших разрешениях на основе генеративно-состязательных нейронных сетей: магистерская диссертация. – Минск, 2020. – 53 с.

40. Сапронов С.С. Нейронные сети: теория и практика. – М.: Издательство МГУ, 2005. – 320 с.