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

РАБОТА ПРОВЕРЕНА

Рецензент
Руководитель отдела
операционной работы
ООО «Наполеон АйТи»

_____С.П. Башков

“___”_____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

“___”_____ 2020 г.

Разработка приложения для генерации изображения по наброску с применением нейросетевых технологий

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2020.114-133.ВКР

Научный руководитель,
к.ф.-м.н., доцент кафедры СП
_____ Н.Ю. Долганина

Автор работы,
студент группы КЭ-403
_____ Ю.Д. Пономарева

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
“___”_____ 2020 г.

Челябинск-2020

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
09.02.2020

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра

студенту группы КЭ-403
Пономаревой Юлии Дмитриевне,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 24.04.2020 № 627)

Разработка приложения для генерации изображения по наброску пользователя с применением нейросетевых технологий.

2. **Срок сдачи студентом законченной работы:** 05.06.2020.

3. **Исходные данные к работе**

3.1 Goodfellow I.J., Pouget-Abadie J., Mirza M., et al. Generative adversarial nets. //Advances in Neural Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc., 2014

3.2 Айрапетов А.Э. Исследование генеративно-состязательной сети / А.Э. Айрапетов, А.А. Коваленко // Политехнический молодежный журнал. – 2018. – No 10.

4. **Перечень подлежащих разработке вопросов**

4.1 Провести обзор существующих аналогов и научной литературы по заданной предметной области.

4.2 Подготовить обучающую и тестовую выборку изображений.

4.3 Программно реализовать и обучить нейронную сеть.

4.4 Провести тестирование реализованной нейронной сети.

4.5 Разработать приложение, генерирующее изображения и провести его тестирование.

5. **Дата выдачи задания:** 08.02.2020.

Научный руководитель
к.т.н., доцент кафедры СП Н.Ю. Долганина

Задание принял к исполнению Ю.Д. Пономарева

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Основные определения

Нейронные сети – математические модели, а также их программные или аппаратные реализации, построенные по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма [11].

Генеративно-состязательная нейронная сеть (Generative Adversarial Networks, GAN) – алгоритм машинного обучения, работа которого строится на основе двух «соперничающих» нейронных сетей. Принцип алгоритма заключается в том, что одна из этих сетей, называемая генератором, пытается сгенерировать определенные образцы (например, изображения, видео или любые другие данные, на генерацию которых она запрограммирована), а другая сеть, называемая дискриминатором, старается решить, является ли представленный ей образец настоящим или сгенерированным. Задачей генератора является производство таких образцов, которые дискриминатор сочтет настоящими, в то время как задача дискриминатора противоположна – он должен отбраковать сгенерированные образцы [3].

U-net – это сверточная нейронная сеть, которая была создана в 2015 году для сегментации изображений. Архитектура сети представляет собой полносвязную сверточную сеть, модифицированную так, чтобы она могла работать с меньшим количеством примеров и делала более точную сегментацию. Сеть содержит сжимающий путь (слева) и расширяющий путь (справа). [8]

Актуальность темы исследования

В последние несколько лет интерес к изучению нейронных сетей очень возрос. Они успешно применяются для решения многих задач: распознавание речи, обнаружение объектов, распознавание образов.

Нейронные сети зарекомендовали себя в таких областях, как меди-

цина, компьютерное зрение, производство, прогнозирование и моделирование различных процессов человеческой деятельности. Но в областях, которые связаны с творческой деятельностью, нейронные сети применяются мало. Наиболее развитой областью является различного рода работа с изображениями.

Генерация изображений с помощью генеративно-состязательных сетей очень перспективна и найдет применение во многих областях. Например, можно использовать данную технологию для увеличения количества изображений в обучающих выборках, также можно улучшать качество изображений и видеопотоков. Также изображения, которые полностью сгенерированы искусственным интеллектом, перспективны в современном искусстве. Поэтому на данный момент одной из наиболее актуальных задач искусственного интеллекта является машинное творчество.

Цель и задачи исследования

Целью данной работы является разработка приложения, для генерации изображений из набросков пользователя с применением нейросетевых технологий. Для достижения поставленной цели необходимо выполнить следующие задачи.

1) Провести обзор существующих аналогов и научной литературы по заданной предметной области.
2) Подготовить обучающую и тестовую выборку изображений.
3) Программно реализовать и обучить нейронную сеть.
4) Провести тестирование реализованной нейронной сети.
5) Разработать приложение, генерирующее изображения и провести его тестирование.

Структура и объем работы

Работа состоит из введения, 4 разделов, заключения, библиографии и приложения. Объем работы составляет 37 страниц, объем библиографии составляет 16 источников.

В первой главе приводится анализ предметной области, обзор аналогов, а также существующих решений поставленной задачи.

Во второй главе содержится описание архитектуры нейронной сети, приведены функциональные и нефункциональные требования к системе, диаграмма вариантов, а также спецификация.

В третьей главе содержится описание процесса разработки приложения.

В четвертой главе приведены результаты тестирования системы.

В заключении описываются результаты проделанной работы, а также направления дальнейших исследований.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Анализ аналогичных проектов

Progressive Growing of GANs [6]

Исследователи NVIDIA и Университета Аалто разработали инновационный подход к генерации изображений. Идея заключается в добавлении новых слоев при обучении генератора и дискриминатора. Это делается для того, чтобы нейросети смогли изучить сначала крупные структуры на изображении, а после перейти к более мелким. За счет этого достигается более стабильный процесс обучения и также улучшается качество финального изображения. В предлагаемой работе авторы используют зеркальные нейросети Генератора и Дискриминатора, начиная с очень малого разрешения (4x4 пикселя) и последовательно развивая нейросеть для работы с высоким разрешением (1024x1024).

Процесс добавления новых слоев показан на рисунке 1.

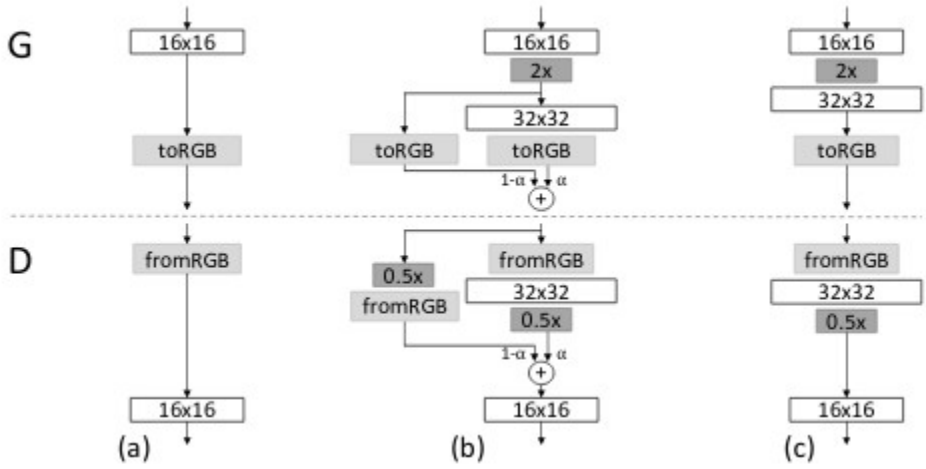

Рис. 1. Добавление новых слоев

Происходит постепенное удвоение разрешение генератора и дискриминатора. Изображения на вход дискриминатора поступают масштабированные, но с увеличением слоев, увеличивается и размер входа.

На выходе получаются очень реалистичные изображения людей.

Пример работы нейросети показан на рисунке 2.

Рис. 2. Сгенерированные изображения

A Style-Based Generator Architecture for Generative Adversarial Networks [5]

StyleGAN была представлена в 2018 году. Она основывается на стандартной архитектуре GAN, которая использовалась в ProGAN, но также берет плюсы из механизма передачи стиля. Топология сети представлена на рисунке 3.

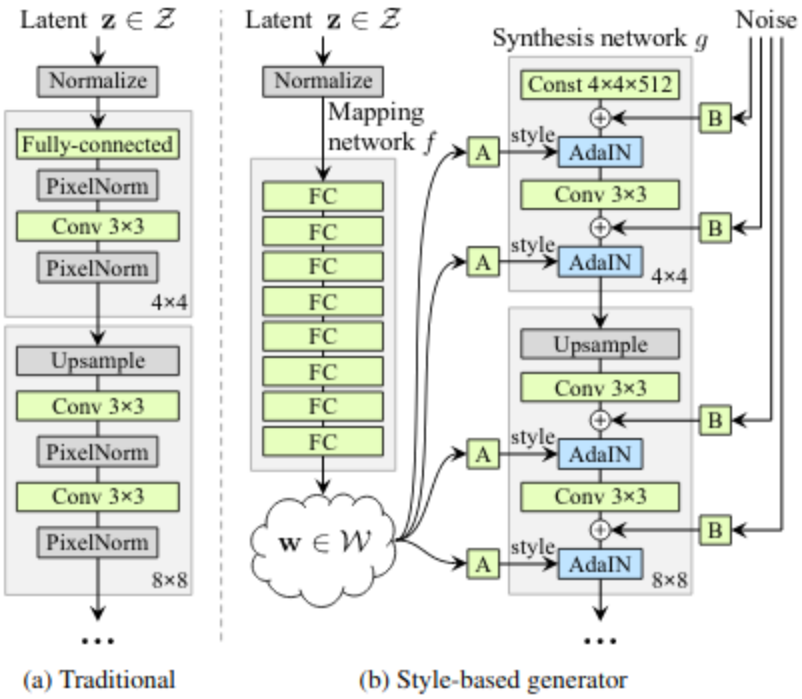

Рис. 3. Топология генератора StyleGAN

StyleGAN модифицирует генератор, который создает изображение путем многократного увеличения. При этом на каждом уровне используется комбинация случайных входных данных. Это помогает нейросети понять, как стилизовать изображения с определенным разрешением. Постоянно создавая такую случайность на каждом этапе процесса обучения, StyleGAN может эффективно выбирать более удачные варианты. Пример работы нейросети показан на рисунке 4.

Рис. 4. Пример сгенерированных изображений

1.2. Анализ существующих решений для реализации проекта

В качестве входных данных разрабатываемой системы используется изображение, но перед подачей этих данных на нейронную сеть необходимо обработать его, то есть извлечь только ту информацию, которая необходима для решения задачи. В нашем случае эта информация представляет собой контуры главного объекта на изображении в таком стиле, как будто его нарисовали карандашом. Для получения контуров объектов существуют соответствующие алгоритмы. В данном подразделе при-

водится анализ основных методов обработки изображений, а также существующих программных решений для обучения нейронных сетей.

1.2.1. Анализ алгоритмов выделения контуров

Все указанные методы основываются на одном из базовых свойств сигнала яркости – разрывности. Наиболее общим способом поиска разрывов является обработка изображения с помощью скользящей маски, которая представляет собой некую квадратную матрицу, соответствующую указанной группе пикселей исходного изображения.

Оператор Кэнни [1]

Был разработан в 1986 году Джоном Кэнни и использует многоступенчатый алгоритм для обнаружения широкого спектра границ в изображениях.

Сначала изображение сглаживается для удаления шума, затем ищутся градиенты. Границы отмечаются там, где градиент изображения приобретает максимальное значение. Они могут иметь различное направление, поэтому алгоритм Кэнни использует четыре фильтра для обнаружения горизонтальных, вертикальных и диагональных ребер в размытом изображении. Затем выбираются только локальные максимумы, которые и отмечаются как границы.

Пример работы оператора представлен на рисунке 5.

Рис. 5. Оператор Кэнни

Оператор Собеля [9]

Дискретный дифференциальный оператор, вычисляющий приближенное значение градиента яркости изображения. Результатом применения оператора Собеля в каждой точке изображения является вектор градиента яркости в этой точке или его норма.

Оператор основан на свертке изображения небольшими сепарабельными целочисленными фильтрами в вертикальном и горизонтальном направлениях, что не затрудняет вычисления. Но используемая им аппроксимация градиента очень грубая.

Оператор вычисляет градиент яркости изображения в каждой точке. Результат показывает, как резко меняется яркость изображения в каждой точке.

Пример работы оператора представлен на рисунке 6.

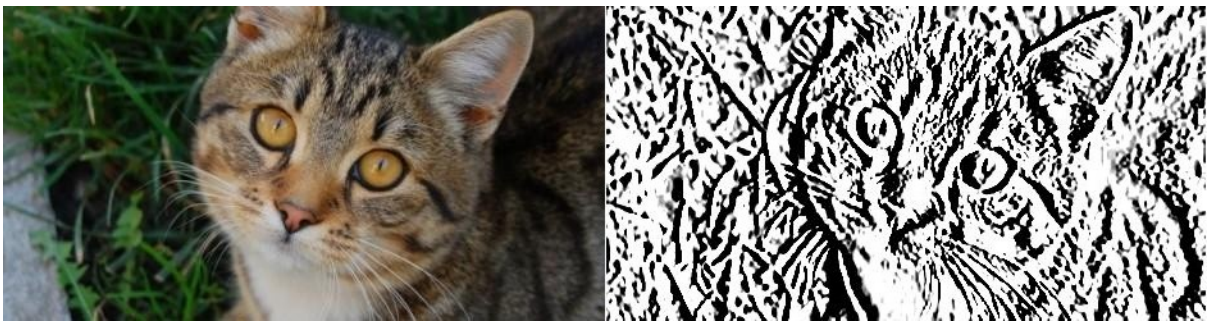

Рис. 6. Оператор Собеля

Оператор Лапласа [7]

Оператор обладает недостатком, заключающимся в раздваивании границы, если она не является достаточно резкой, т.е. если функция яркости содержит участки постоянного наклона. В этом случае необходимо применять специальную дополнительную обработку устранения раздваивания линий.

Пример работы оператора представлен на рисунке 7.

Рис. 7. Оператор Лапласа

1.2.2. Анализ программных решений

Существует большое количество программных решений, связанных с машинным обучением, для языка Python. Далее будут рассмотрены основные их них.

TensorFlow [13]

Открытая библиотека от Google для машинного обучения. Для хранения данных в данной библиотеке используются многомерные массивы – тензоры, а для представления нейронной сети используются графы. Можно применять аппаратный ускоритель – тензорный процессор (TPU) – специализированная интегральная схема, адаптированной под задачи для TensorFlow. Плюсами библиотеки являются хорошая документация, а также большое количество готовых примеров в реализации генеративно-состязательных нейронных сетей.

Keras [11]

Библиотека для создания нейронные сетей, которая представляет из себя надстройку над Tensorflow, что позволяет использовать возможности этой библиотеки для проведения вычислений. Отличительными особенностями Keras являются простота в использовании, модульность и легкость масштабирования.

PyTorch [12]

Библиотека для машинного обучения от Facebook, обеспечивающая тензорные вычисления с GPU-ускорением. Характеризуется

гибкостью в использовании за счет того, что в ней не используются статические расчетные графы.

В ходе обзора библиотек было решено использовать Tensorflow, как наиболее гибкую, интуитивно понятную. Еще одним важным фактором было наличие реализаций схожих проектов с помощью данной библиотеки.

Выводы по первому разделу

Обзор существующих решений показал, что задача генерации изображения с помощью нейросетевых технологий является актуальной.

Анализ работ по генерации изображений показал, что из существующих решений наибольшего качества изображения можно добиться, используя генеративно-состязательные нейронные сети.

Существует много алгоритмов выделения контуров на изображении, но их реализация в библиотеке OpenCV не дает тех изображений, которые бы были похожи на наброски карандашом, которые в дальнейшем будут использоваться для обучения нейросети, поэтому принято решение написать свою обработку изображений.

Также существует широкий ряд библиотек для создания и обучения нейронных сетей. Использование подобных библиотек может значительно облегчить задачу реализации. Принимая во внимание все описанные выше факторы, можно сделать вывод, что исследование в данной области является актуальным.

2. ПРОЕКТИРОВАНИЕ

2.1. Сверточные нейронные сети

Использование классических нейронных сетей для распознавания изображений затруднено из-за большой размерности входного вектора, большого количества нейронов в промежуточных слоях, вследствии этих факторов увеличиваются вычислительные ресурсы на обучении. Сверточные нейронные сети в отличие от классических нейронных сетей используют гораздо меньшее количество настраиваемых весов, так как одно ядро весов используется целиком для всего изображения, вместо того, чтобы делать для каждого пикселя входного изображения свои персональные весовые коэффициенты.

Сверточная нейронная сеть – архитектура искусственных нейронных сетей, предложенная Яном Лекуном и нацеленная на эффективное распознавание изображений, входит в состав технологий глубокого обучения. Технология построена по аналогии с принципами работы зрительной коры головного мозга, в которой были открыты простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определенного набора простых клеток [14-15].

В сверточной нейронной сети в операции свертки используется лишь ограниченная матрица весов небольшого размера, которую «двигают» по всему обрабатываемому слою, формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. То есть для различных нейронов выходного слоя используются одна и та же матрица весов, которую также называют ядром свертки. Один из этапов процесса свертки представлен на рисунке 8.

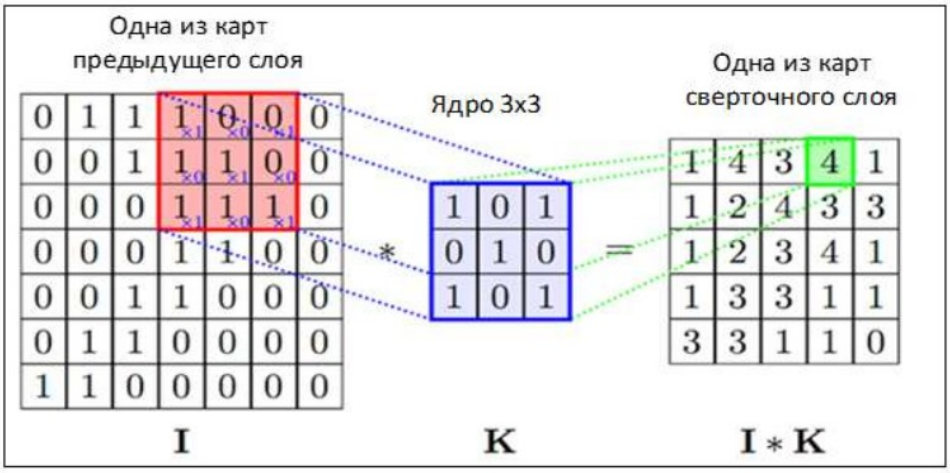

Рис. 8. Процесс свертки в сверточной нейронной сети

Нейроны, использующие одни и те же веса, объединяются в карты признаков, а каждый нейрон карты признаков связан с частью нейронов предыдущего слоя.

Помимо сверток в сверточной нейронной сети могут быть слои субдискретизации, их цель в уменьшении размерности карт признаков. Обычно каждая карта имеет ядро размером 2 на 2, что позволяет уменьшить предыдущие карты сверточного слоя в 2 раза. Вся полученная карта признаков разделяется на ячейки 2 на 2 элемента, из которых выбираются и максимальные по значению, и уже они формируют новую карту. Операция подвыборки изображена на рисунке 9.

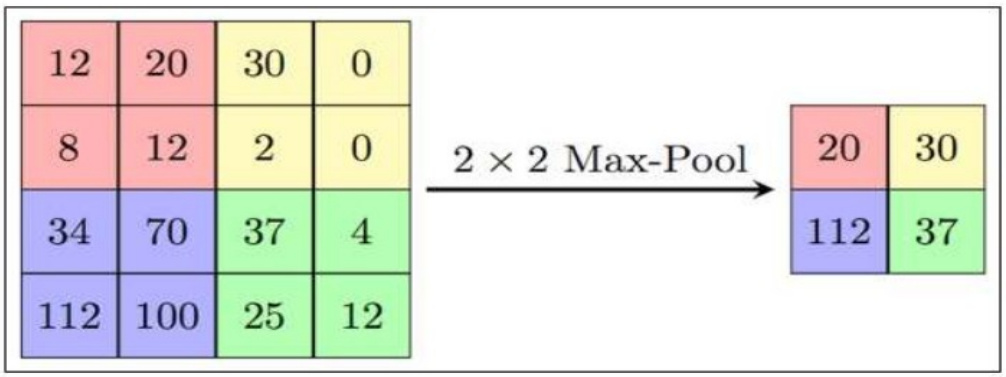

Рис. 9. Процесс подвыборки в сверточной сети

Также в сверточной нейронной сети присутствуют полносвязные слои. Все три вида слоев могут чередоваться в произвольном порядке, что позволяет составлять карты признаков из карт признаков, а это на практике означает способность распознавания сложных иерархий признаков [11].

2.2. Генеративно-состязательные нейронные сети

Данный тип архитектуры нейронных сетей был предложен в 2014 году Йеном Гудфеллоу (Ian Goodfellow) [2-3, 10]. Модель сети представлена на рисунке 10.

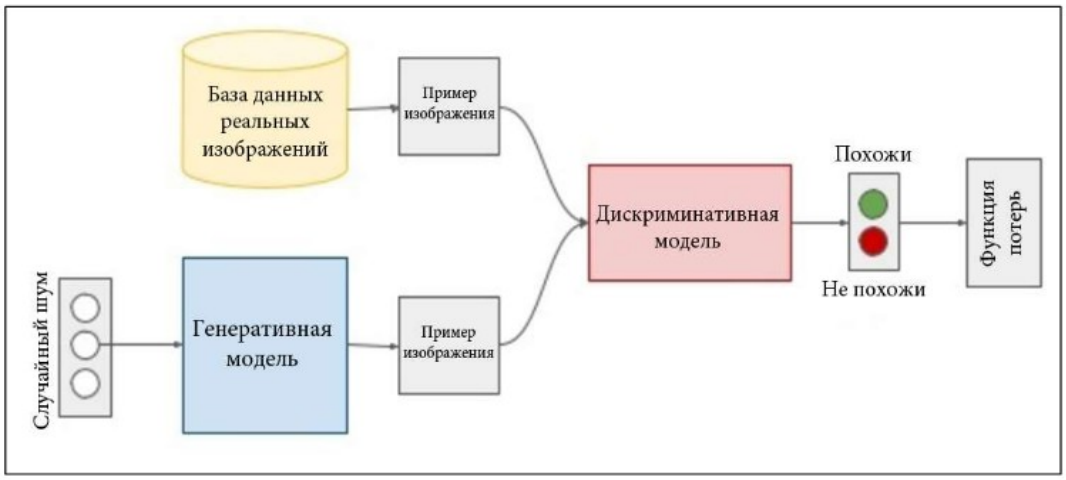

Рис.10. Генеративно-состязательная нейронная сеть

Суть идеи в комбинации двух нейросетей, при которой одновременно работает два алгоритма “генератор” и “дискриминатор”. Первая сеть на вход получает случайный шум, после чего ей надо сгенерировать образ заданной категории. После этого, данный объект получает дискриминатор, который пытаться распознать созданный образ и определить, является полученный объект реальным, или же сгенерированным.

Задача Генеративной модели – генерировать такие изображения, которые вторая сеть – Дискриминатор, будет не в состоянии отличить от

настоящих. Предполагается, что в таком соревновании, обе модели будут помогать друг другу в успешном обучении.

Процесс обучения таких сетей можно представить следующим образом.

1) Выбирается n примеров из датасета и m примеров от генератора.

2) Фиксируются веса генератора, обновляются параметры дискриминатора. Данный этап похож на задачу классификации, однако нет необходимости тренировать дискриминатор до сходимости.

3) Фиксируются веса дискриминатора и производится обновление весов генератора таким образом, чтобы максимизировать ошибку дискриминатора на новых примерах, созданных генератором.

4) Этапы 1-3 повторяются до тех пор, пока дискриминатор и генератор не придут к состоянию равновесия.

2.3. Функциональные и нефункциональные требования

Для решения задачи генерации рисунков из набросков пользователя должно быть создано приложение в виде чат-бота, обеспечивающее возможность автоматизированного генерирования одного изображения. Пользователь должен иметь возможность отправить изображение в приложение.

Функциональные требования

Функциональные требования – условия, накладывающиеся на поведение системы, а также действия, которые система имеет возможность выполнять. Разрабатываемая система должна удовлетворять следующим функциональным требованиям:

1) система должна принимать на вход пользовательское изображение;

2) система должна создавать файл с результатами генерации в доступной форме;

3) система должна быть реализована в виде чат-бота в мессенджере Telegram.

Нефункциональные требования

Нефункциональные требования представляют собой условия, не связанные с поведением системы и характеризующие условия окружающей среды, или же качества, которыми должна обладать система.

Для проектируемой системы были определены следующие нефункциональные требования:

1) система должна быть реализована на языке Python;

2) система должна использовать фреймворк для создания нейронных сетей Tensorflow;

3) система должна быть единой и не требовать от пользователя дополнительных действий для работы, кроме указания входных данных.

2.5. Варианты использования системы

Для проектирования приложения был использован язык графического описания для объектного моделирования UML. Была построена модель взаимодействия актера «Пользователь» с системой. Диаграмма вариантов использования представлена на рисунке 11.

В рамках разрабатываемой системы предусмотрен один актер – пользователь.

Пользователь может *задать входные данные* системы. Для этого он должен загрузить файл с рисунком.

Пользователь может *запустить генерацию* изображения. Для этого он должен отправить файл загруженным ранее рисунком.

Пользователь может *просмотреть результаты*. Для этого нужно открыть файл, который отправит приложение после успешной генерации изображения.

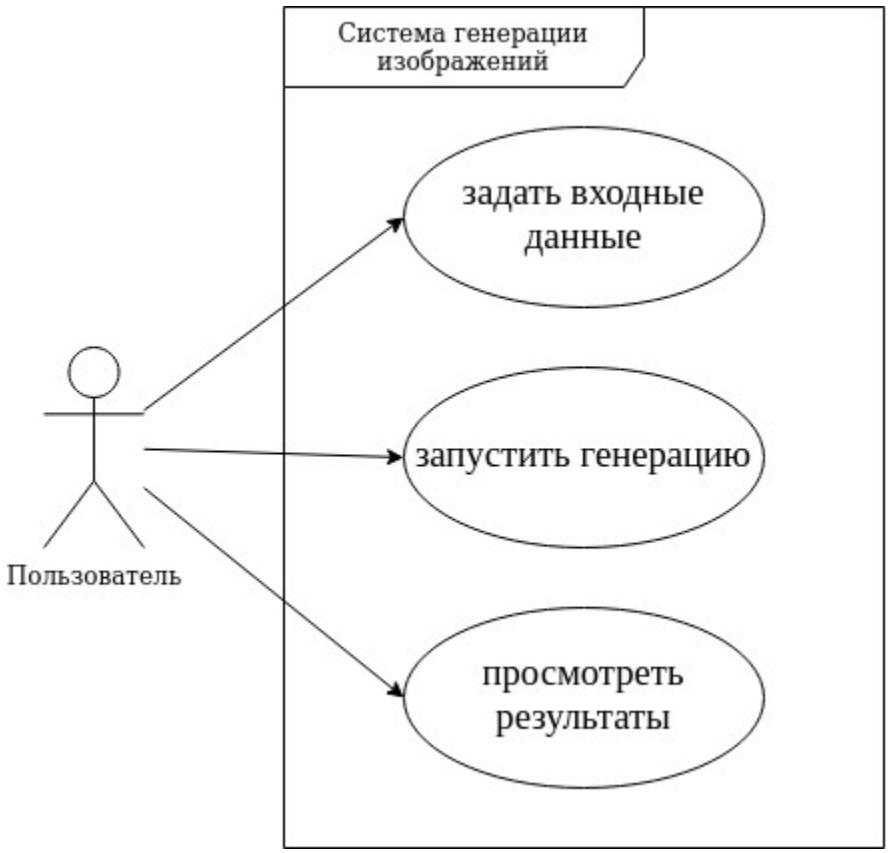

Рис. 11. Диаграмма вариантов использования

2.5 Общее описание архитектуры системы

На рисунке 12 представлена диаграмма классов системы. Система содержит 5 классов: Main, ImageProcessing, NeuralNetwork, Bot, TrainingNN которые подробно описаны далее.

1) Main – главный класс, обеспечивающий работу системы. Метод startBot() запускает работу чат-бота, обращаясь к классу Bot. Метод startNN() запускает работу нейросети, обращаясь к классу NeuralNetwork.

2) ImageProcessing – класс, отвечающий за обработку данных. Метод preprocess() предобрабытвает изображение пользователя, метод combine_input_output() сохраняет рисунок пользователя и сгенерирован-

ное изображение в один файл, метод transform_tf_tensor_to_bytes() конвертирует сгенерированный тензор в байты, для последующей отправки файла.

3) NeuralNetwork – класс, отвечающий за работу нейронной сети. Он содержит следующие атрибуты: generator – модель Генератора, discriminator – модель Дискриминатора. Метод Discriminator() позволяет создать модель Дискриминатора, метод Generator() создает модель Генератора, метод predict() генерирует изображение из входных данных.

4) Bot – класс, отвечающий за работу чат-бота. Он имеет атрибут token – уникальный идентификатор. Метод start_polling() запускает работу бота, метод get_image_bytes() скачивает файл пользователя, метод start() является обработчиком команды /start, метод handle_image() является обработчиком поступивших файлов от пользователя, метод unknown_command() является обработчиком неизвестных команд или сообщений от пользователя.

5) TrainingNN – класс, отвечающий за обучение нейронной сети. Он содержит следующие атрибуты: path – путь до обучающего датасета, epochs – количество эпох для обучения, batch_size – количество обучающих примеров, поступающих за одну итерацию, img_width – ширина изображения, img_height – высота изображения. Метод load() – загружает изображение, метод preprocess() предобрабатывает данные масштабируя и изменяя их размер, метод random_jitter() трансформирует изображение для увеличения обучающей выборки, методы load_image_train() и load_image_test() загружают тренировочную и обучающую выборки соответственно, метод generate_images() позволяет сгенерировать изображение, метод train_step() проходит одну итерацию обучения, оптимизируя веса модели, метод fit() – запускает обучение модели.

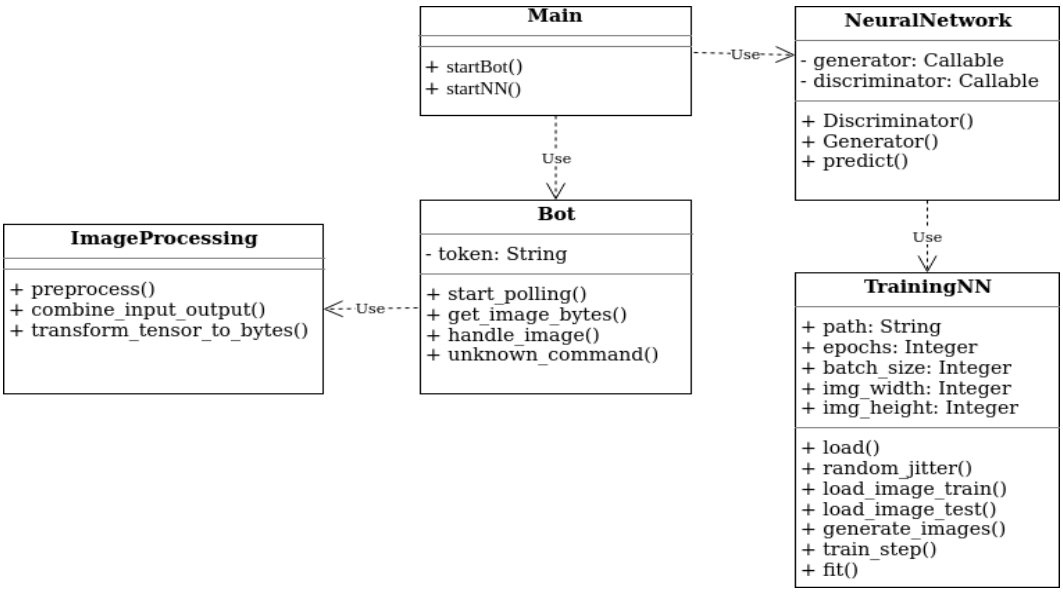

Рис. 12. Диаграмма классов

2.6. Описание компонентов, входящих в систему

Система состоит из следующих компонентов: MainProgram, Bot, NeuralNetwork, PreprocessingScript. Диаграмма компонентов системы изображена на рисунке 13, эти компоненты подробно описаны далее.

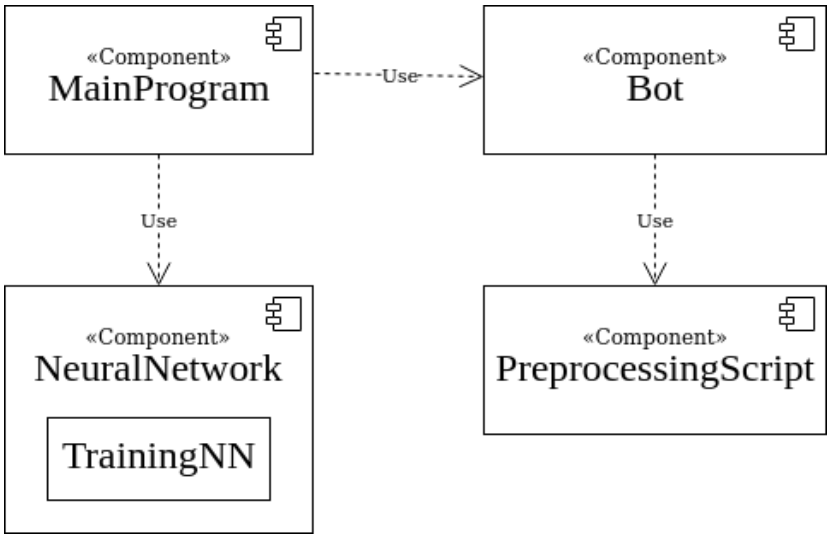

Рис. 13. Диаграмма компонентов

Компонент MainProgram является главным модулем программы, который осуществляет контроль над всеми действиями системы и запускает работу остальных компонентов, передавая им входные параметры и запуская их.

Bot – компонент, запускающий работу чат-бота.

NeuralNetwok – компонент, выполняющий генерацию изображения по файлу пользователя с помощью искусственной нейронной сети. Компонент NeuralNetwork содержит класс TrainingNN.

PreprocessingScript – компонент, обеспечивающий предобработку входных данных системы. Параметры предобработки, включая расположение входных данных, получены на вход от компонента Bot.

2.7 Реализация архитектуры системы

На рисунке 14 представлена диаграмма деятельности системы, описывающая процесс генерации изображения.

Рис. 14. Диаграмма деятельности

Выводы по главе 2

В данном разделе были сформулированы функциональные и нефункциональные требования к системе, созданы диаграмма вариантов использования, диаграммы классов, компонентов и деятельности. Спроектирована система генерации изображений, состоящая из 5-ти классов и 4-х программных компонентов.

3. РЕАЛИЗАЦИЯ

3.1. Средства разработки

Для разработки программной части был использован высокоуровневый язык программирования Python версии 3.6.4. Обучение нейронной сети проводилась на ОС Kubuntu 18.04 с характеристиками: Nvidia RTX 2080 Ti GPU, 6 CPU cores(Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz) и 11 GB GDDR6. Разработка велась в среде разработки PyCharm Community Edition 2018.2.3, обучение нейросети и подготовка данных для обучения велись в JupyterLab 2.1.2.

Для реализации использовались следующие технологии.

1) Для парсинга изображений с сайта использовалась библиотека selenium.

2) Для обучения нейронной сети была использована библиотека машинного обучения TensorFlow.

3) Для реализации чат-бота была библиотека python-telegram-bot.

4) Для работы с изображениями использовались библиотеки opencv-python, pillow, matplotlib.

3.2. Формирование обучающей выборки

Для обучающей и тестовой выборок датасет был собран с помощью парсинга сайта с картинками с помощью библиотеки selenium. Функция, выполняющая парсинг картинок представлена в листинге 1.

Листинг 1. Функция парсинга картинок

```
def parse_images(driver, path_to_data):
      images = driver.find_elements_by_class_name('serp-item__link')
      name = 1
      for i in tqdm_notebook(images):
      i.click()
      url = driver.find_element_by_class_name(
            'MMImage_origin'
      ).get_attribute('src')

      if not url.endswith('.jpg'):
            url += '.jpg'
```

```
      try:
            img = requests.get(url).content
            image = Image.open(io.BytesIO(img))
            image = np.array(image)
            image = image.astype(np.float32)/255.0
            plt.imsave(path_to_data / Path(f'{name}.jpg'), image)
            time.sleep(1.2)
            name += 1
      except Exception:
            pass

      driver.find_element_by_class_name(
            'MMViewerModal-Close'
      ).click()
```

Далее нужно было изображения перевести в контура. Для этого была написана функция, которая переводит изображение в серый цвет и подсчитывает градиент. Код функции представлен в листинге 2.

Листинг 2. Функция получения контуров изображения

```
def get_edges_2(path_to_source_img, path_to_dist_img, show=False):
      # Load image:
      input_image = Image.open(path_to_source_img)
      input_pixels = input_image.load()
      width, height = input_image.width, input_image.height

      # Create output image
      output_image = Image.new("RGB", input_image.size)
      draw = ImageDraw.Draw(output_image)

      # Convert to grayscale
      intensity = np.zeros((width, height))
      for x in range(width):
      for y in range(height):
            intensity[x, y] = sum(input_pixels[x, y]) / 10

      # Compute convolution between intensity and kernels
      for x in range(1, input_image.width - 1):
      for y in range(1, input_image.height - 1):
            magx = intensity[x + 1, y] - intensity[x - 1, y]
            magy = intensity[x, y + 1] - intensity[x, y - 1]

            # Draw in black and white the magnitude
            color = int(sqrt(magx**2 + magy**2))
            draw.point((x, y), (color, color, color))

      output_image = 255 - np.array(output_image)
      output_image = Image.fromarray(output_image).convert('L')
      if show:
            plt.imshow(output_image)
            plt.show()
      else:
            output_image.save(path_to_dist_img)
```

Для обучения на вход нейросети подаются контура и изначальное изображение, поэтому нужно было объединить все изображения. Результат объединения представлен на рисунке 15.

Рис. 15. Входное изображение на нейросеть

3.3. Топология нейронной сети

Для решения поставленной задачи была разработана соответствующая топология нейронных сетей.

Архитектура Генератора

Генератор представляет из себя модифицированный U-Net.

Каждый блок кодера (сжимающая часть) состоит из сверточных слоев, BatchNormalization [4] используется для ускорения обучения. В качестве активационной функции сверточных слоев используется LeakyReLu.

Каждый блок декодера (расширяющая часть) состоит из следующих слоев.

1) Разверточный слой (слой обратный сверточному).
2) BatchNormalization.
3) Dropout (случайный отброс нейронов на время итерации для избегания переобучения).

В качестве активационной функции сверточных слоев используется ReLU.

Топология Генератора представлена на рисунке 16.

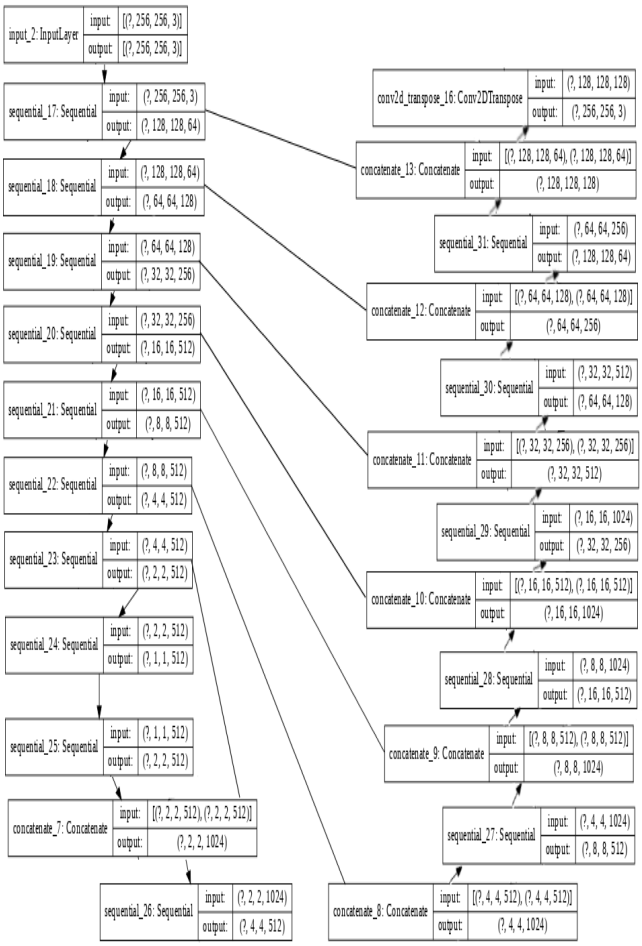

Рис. 16. Топология Генератора

Архитектура Дискриминатора

Нейросеть принимает 2 входа.

1) Входное изображение и целевое изображение, этот вход должен классифицироваться как настоящий.

2) Входное изображение и сгенерированное изображение, этот вход должен быть классифицирован как ненастоящий.

Каждый блок состоит из сверточного слоя, BatchNormalizarion и в качестве активационной функции используется Leaky ReLU.

Топология Дискриминатора представлена на рисунке 17.

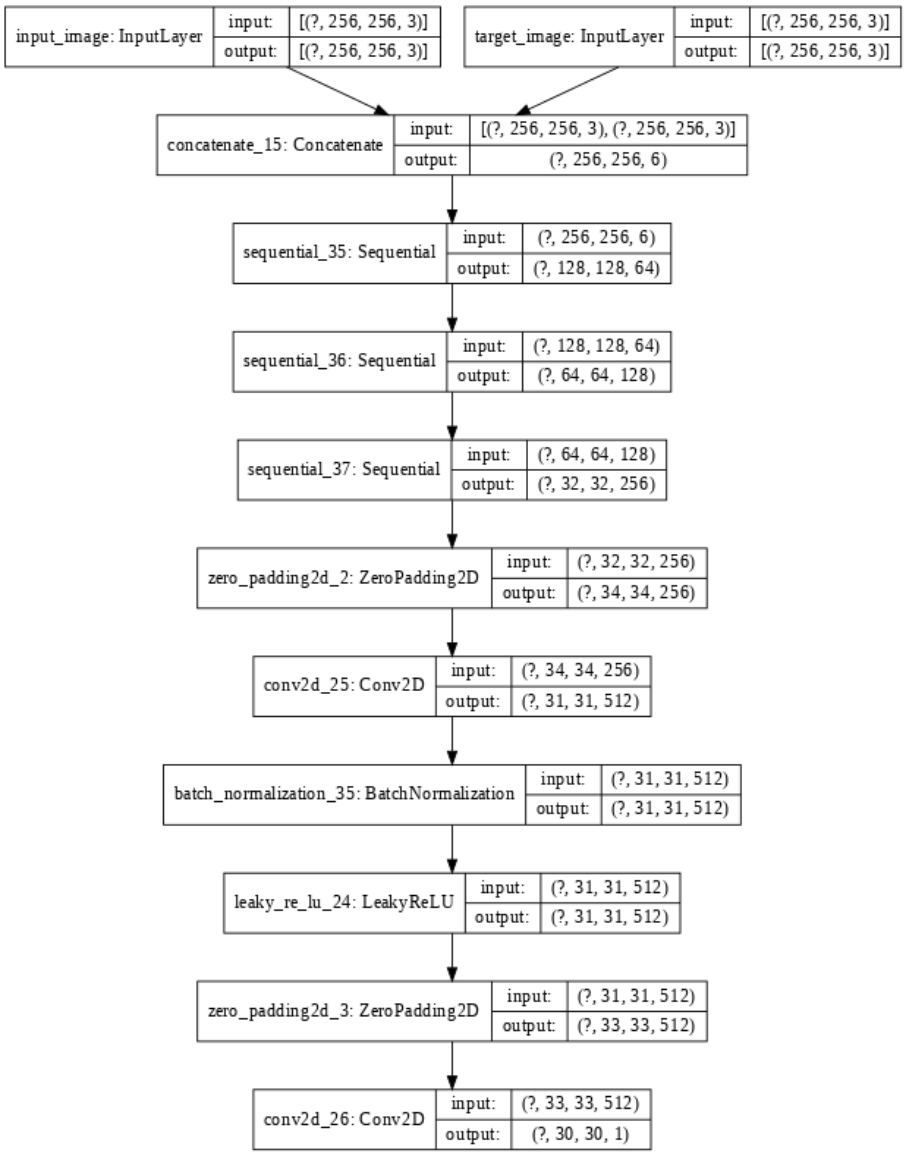

Рис. 17. Топология Дискриминатора

3.4. Реализация нейронной сети

На вход сети подается вектор признаков изображения, равный размерности каждого кадра на количество цветовых каналов в нем. На выходе получаем вектор такой же размерности.

В данной работе были применены следующие параметры: ширина и высота каждого кадра – 256 пикселей, количество цветовых каналов – 3 (RGB изображение).

Реализация построения моделей на языке Python с использованием фреймворка Tensorflow приведены в листинге 3 и листинге 4.

Листинг 3. Построение модели Генератора

```
def Generator():
 inputs = tf.keras.layers.Input(shape=[IMG_HEIGHT, IMG_WIDTH, 3])
 down_stack = [
   downsample(64, 4, apply_batchnorm=False), # (bs, 128, 128, 64)
   downsample(128, 4), # (bs, 64, 64, 128)
   downsample(256, 4), # (bs, 32, 32, 256)
   downsample(512, 4), # (bs, 16, 16, 512)
   downsample(512, 4), # (bs, 8, 8, 512)
   downsample(512, 4), # (bs, 4, 4, 512)
   downsample(512, 4), # (bs, 2, 2, 512)
   downsample(512, 4), # (bs, 1, 1, 512)
 ]

 up_stack = [
   upsample(512, 4, apply_dropout=True), # (bs, 2, 2, 1024)
   upsample(512, 4, apply_dropout=True), # (bs, 4, 4, 1024)
   upsample(512, 4, apply_dropout=True), # (bs, 8, 8, 1024)
   upsample(512, 4), # (bs, 16, 16, 1024)
   upsample(256, 4), # (bs, 32, 32, 512)
   upsample(128, 4), # (bs, 64, 64, 256)
   upsample(64, 4), # (bs, 128, 128, 128)
 ]

 initializer = tf.random_normal_initializer(0., 0.02)
 last = tf.keras.layers.Conv2DTranspose(
     OUTPUT_CHANNELS, 4, strides=2,
     padding='same', kernel_initializer=initializer,
     activation='tanh') # (bs, 256, 256, 3)
 x = inputs

 skips = []
 for down in down_stack:
   x = down(x)
   skips.append(x)
 skips = reversed(skips[:-1])

 for up, skip in zip(up_stack, skips):
   x = up(x)
   x = tf.keras.layers.Concatenate()([x, skip])
 x = last(x)
 return tf.keras.Model(inputs=inputs, outputs=x)
```

Листинг 4. Построение модели Дискриминатора

```
def Discriminator():
 initializer = tf.random_normal_initializer(0., 0.02)

 inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_image')
 tar = tf.keras.layers.Input(shape=[256, 256, 3], name='target_image')

 x = tf.keras.layers.concatenate([inp, tar]) # (bs, 256, 256, channels*2)

 down1 = downsample(64, 4, False)(x) # (bs, 128, 128, 64)
 down2 = downsample(128, 4)(down1) # (bs, 64, 64, 128)
 down3 = downsample(256, 4)(down2) # (bs, 32, 32, 256)

 zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3) # (bs, 34, 34, 256)
 conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                               kernel_initializer=initializer,
                                 use_bias=False)(zero_pad1) # (bs, 31, 31,
512)

 batchnorm1 = tf.keras.layers.BatchNormalization()(conv)

 leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)

 zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)

 last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                 kernel_initializer=initializer)(zero_pad2)
 return tf.keras.Model(inputs=[inp, tar], outputs=last)
```

3.5. Реализация чат-бота

Боты – специальные аккаунты в Telegram, созданные для того, чтобы автоматически обрабатывать и отправлять сообщения. Пользователи могут взаимодействовать с ботами при помощи сообщений, отправляемых через обычные или групповые чаты. Логика бота контролируется при помощи HTTPS запросов к нашему API для ботов. По сути, этот аккаунт играет роль интерфейса к системе [16].

Команды представляют собой более гибкий способ общения с роботом. Рекомендуется следующий синтаксис: /команда. Команда

должна начинаться с символа косой черты «/» и не может быть длиннее 32 символов.

Необходимо создать каркас чат-бота и получить токен, и уникальный адрес, с помощью которых будет происходить процесс взаимодействия с ботом.

Для этих целей, используется чат-бот BotFather разработанный Telegram (@BotFather). BotFather представляет собой раздел для администратора чат-ботов.

Далее для создания необходимо отправить команду /newbot, после этого нам необходимо ввести название и адрес нашего чат-бота. После выполненного действия BotFather пришлет нам уникальный токен, который мы будем использовать при написании программы.

Токен – это маркер, который содержит в зашифрованном виде всю минимально необходимую информацию для аутентификации и авторизации.

4. ТЕСТИРОВАНИЕ

Тестирование осуществлялось в системе со следующими данными: Nvidia RTX 2080 Ti GPU 11 GB GDDR6, 6 CPU cores(Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz). Производительность данного оборудования позволяет быстро генерировать изображения.

Тестирование нейронной сети

Обучение нейронной сети производилось на 1200 эпохах, с коэффициентом скорости обучения 0,0001. При обучении функция потери генератора была равна 12,545, а функция потери дискриминатора – 0,738.

В качестве функции потерь для генератора использовалась сумма бинарной кросс-энтропии (Binary (Sigmoid) Cross-Entropy) и среднеквадратичной ошибки (mean absolute error) между сгенерированными и целевыми изображениями. Это позволяет генерировать изображения структурно похожие на реальные. Итоговая формула функции потеря generator loss = gan_loss + LAMBDA * l1_loss, где LAMBDA = 100 (эмпирически выведенный коэффициент).

В качестве функции потерь для дискриминатора использовалась сумма бинарной кросс-энтропии реальных изображений и массивы единиц и бинарной кросс-энтропии сгенерированных изображений и массива нулей.

Для оптимизации использовался метод адаптивной инерции (Adam Optimizer).

Функциональное тестирование

Функциональное тестирование – это тестирование ПО для проверки реализуемости функциональных требований, то есть способность ПО в определенных условиях решать задачи, нужные пользователям.

Функциональные требования определяют, что именно делает

программное обеспечение, какие задачи оно решает. Результаты тестирования приведены в таблице 1.

Табл. 1. Тестирование системы

No	**Цель**	**Действие**	**Ожидаемый результат**	**Результат теста**
1	Проверить работу чат-бота	Пользователь отправил боту команду /start	Бот ответил на команду приветственным сообщением	Пройден
2	Проверить процесс генерации изображения	Пользователь отправляет файл боту	Система сгенерировала изображение	Пройден
3	Проверить процесс объединения начального и сгенерированного изображения	Пользователь отправляет файл боту	Начальное и сгенерированное изображения объединяются корректно	Пройден
4	Проверить процесс обработки неверных команд, отправленных боту	Пользователь отправил некорректную команду	Бот ответил соответствующим сообщением	Пройден

Вывод по главе 4

В ходе тестирования было проведено обучение нейронных сетей и их тестирование, функция потерь которых может быть уменьшена путем увеличения обучающей выборки, что позволит избежать запоминания сетью конкретных изображений.

Была протестирована работа системы генерации изображений. Система выполнила свою задачу и сгенерировала изображение из наброска пользователя, предоставив результаты в файле.

ЗАКЛЮЧЕНИЕ

Была разработана система генерации изображений из набросков пользователя. Были проанализированы существующие методы генерации изображений. Было принято решение генерировать изображения на контурах, полученных на основе градиента, а также использовать модифицированную U-net архитектуру нейронной сети.

На основе требований к системе были выделены основные классы и компоненты системы, установлены связи между ними. Была подробно рассмотрена реализация отдельных прецедентов.

В ходе разработки системы были получены следующие результаты:

1) проведен анализ аналогов и научной литературы в предметной области;
2) реализован алгоритм предобработки данных;
3) сформированы обучающая и тестовая выборки;
4) реализована, обучена и протестирована модель нейронной сети для генерации изображений;
5) реализована и протестирована система генерации изображений.

Также были получены навыки программирования на языке Python, разработки нейронных сетей с помощью фреймворка Tensorflow, работы с библиотекой алгоритмов компьютерного зрения OpenCV, работы с библиотекой python-telegram-bot для реализации чат-бота.

В будущем планируется провести работу по улучшению качества работы Генератора. Для этого необходимо расширить обучающую выборку и скорректировать топологию нейронной сети.

На данных момент система успешно генерирует только некоторых животный. Возможно увеличить пользу системы путем добавления генерации и других объектов.

ЛИТЕРАТУРА

1. Canny Edge Detection. [Электронный ресурс] URL: https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html (дата обращения: 11.03.2020).

2. Goodfellow I.J., Pouget-Abadie J., Mirza M., et al. Generative adversarial nets. //Advances in Neural Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc., 2014.

3. Goodfellow I. Generative Adversarial Nets / I. Goodfellow, J. Pouget-Abadie, M. Mirza, et al. // Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, (December 8–13 2014, Montreal, Quebec, Canada). – 2014. – P. 2672–2680.

4. Ioffe S. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift / S. Ioffe, C. Szegedy // Proceedings of the 32nd International Conference on International Conference on Machine Learning. – 2015. – Vol. 37. – P. 448–456.

5. Karrs T. A Style-Based Generator Architecture for Generative Adversarial Networks / T. Karras, S. Laine, T. Aila // ArXiv preprints, 2019. [Электронный ресурс] URL: https://arxiv.org/pdf/1812.04948.pdf (дата обращения: 06.05.2020).

6. Karras T. Progressive Growing of GANs for Improving Quility, Stability and Variation / T. Karras, T. Aila, S. Laine, et al. // Conference paper at ICLR, 2018. [Электронный ресурс] URL: https://research.nvidia.com/sites/default/files/pubs/2017-10_Progressive-Growing-of/karras2018iclr-paper.pdf (дата обращения: 05.05.2020).

7. Laplace Operator [Электронный ресурс] URL: https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html (дата обращения: 11.03.2020).

8. Ronneberger O. U-Net: Convolutional Networks for Biomedical Image Segmentation / O. Ronneberger, P. Fischer, T. Brox // ArXiv preprints,

2019. [Электронный ресурс] URL: https://arxiv.org/pdf/1505.04597.pdf (дата обращения: 01.05.2020).

9. Sobel Derivatives. [Электронный ресурс] URL: https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html (дата обращения: 11.03.2020).

10. Айрапетов А.Э. Исследование генеративно-состязательной сети / А.Э. Айрапетов, А.А. Коваленко // Политехнический молодежный журнал. – 2018. – No 10

11. Документация Keras. [Электронный ресурс] URL: https://keras.io/ (дата обращения: 13.03.2020).

12. Документация PyTorch. [Электронный ресурс] URL: https://pytorch.org/docs/stable/index.html (дата обращения: 13.03.2020).

13. Документация TensorFlow. [Электронный ресурс] URL: https://www.tensorflow.org/api_docs/python/ (дата обращения: 23.03.2020).

14. Петров С. Сверточная нейронная сеть для распознавания символов номерного знака автомобиля // Международный Университет природы, общества и человека «Дубна», 2013. –Вып. 3. –С. 56.

15. Солдатова О., Гаршин А. Применение сверточной нейронной сети для распознавания рукописных цифр. –Самара: Самарский государственный университет,2010. –С. 252-259.

16. Справочник по Bot API [Электронный ресурс] URL: https://tlgrm.ru/docs/bots/api (дата обращения: 03.05.2020)