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 группы разработки
ООО «ЦЕНТР КИБЕРБЕЗОПАСНОСТИ»

_____ А. В. Горюшкин

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Автоматическое распознавание и удаление шума на видео с помощью рекуррентных нейронных сетей

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-1498.ВКР

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

Автор работы,
студент группы КЭ-229
_____ Г.А. Свизев

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 магистранта

студенту группы КЭ-229
Свизеву Геннадию Александровичу,
обучающемуся по направлению
09.04.04 «Программная инженерия»
(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Автоматическое распознавание и удаление шума на видео с помощью рекуррентных нейронных сетей.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Bazarevsky V., Grishchenko I., Raveendran K., Zhu T., Zhang F., Grundmann M. Unsupervised Coordinate-Based Video Denoising. // IEEE Neural Networks Council 3, 2020. – 87 p.

3.2. Bazarevsky V., Kartynnik Y., Vakunov A., Raveendran K. Real-time Controllable Denoising for Image and Video. // Complex Intelligent Systems. 4., 2019. – 25 p.

4. **Перечень подлежащих разработке вопросов**

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

4.2. Спроектировать архитектуру нейронной сети для задачи удаления шума на видео, используя рекуррентные слои.

4.3. Реализовать алгоритм удаления шума на видео с помощью обученной нейронной сети.

4.4. Провести тестирование полученной нейронной сети.

5. Дата выдачи задания: 29.01.2024 г.

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

Задание принял к исполнению Г.А. Свизев

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Шум на видео является общей проблемой, которая может снизить качество и понятность видео контента. В настоящее время массово производятся видеозаписи при помощи различных устройств, таких как мобильные телефоны и камеры низкого качества. Это приводит к появлению шума на видео [1], который может быть различного типа, например, гауссовский шум, «соль и перец», артефакты сжатия, мерцание и т.д.

Существующие методы удаления шума на видео, основанные на фильтрации и статистическом анализе, могут быть неэффективными в случае сложных шумовых паттернов или когда шум меняется со временем. Эти методы также часто требуют настройки параметров для каждого конкретного видео, что затрудняет автоматическое удаление шума.

Рекуррентные нейронные сети (RNN [2]) предоставляют мощный инструмент для моделирования последовательностей данных, таких как видео. Они способны учитывать контекст из прошлых кадров, что делает их особенно полезными для обработки видеоданных. RNN модели, такие как LSTM [3] (долгая краткосрочная память) и GRU [3] (обновление воротной единицы), имеют способность запоминать информацию из прошлых кадров и использовать ее для принятия решений о текущем кадре. Это позволяет им более эффективно обрабатывать шум на видео и улучшать его качество.

Также автоматическое распознавание и удаление шума на видео имеет практическое применение в различных областях. Например, в сфере визуальных коммуникаций, таких как видеоконференции и трансляции в реальном времени, качество видео играет важную роль для передачи информации и эмоций.

В области компьютерного зрения и машинного обучения, удаление шума может улучшить результаты других задач, таких как распознавание объектов, сегментация и отслеживание движущихся объектов.

В свете этих факторов, разработка и применение рекуррентных нейронных сетей для автоматического распознавания и удаления шума на видео обещает значительный прогресс в повышении качества видеозаписей и улучшении результатов других задач компьютерного зрения. Это делает данную тему актуальной и важной для дальнейших исследований и разработок. Также на рисунке 1 представлено уменьшение шума с помощью нейронных сетей.

Рисунок 1 – Пример уменьшения шума на изображении

Постановка задачи

Целью выпускной квалификационной работы является автоматическое распознавание и удаление шума на видео с помощью рекуррентных нейронных сетей.

Для достижения поставленной цели необходимо решить следующие задачи.

1. Провести обзор литературы и существующих аналогов по предметной области.
2. Спроектировать архитектуру нейронной сети для задачи удаления шума на видео, используя рекуррентные слои.
3. Реализовать алгоритм удаления шума на видео с помощью обученной нейронной сети.
4. Провести тестирование полученной нейронной сети.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения, списка литературы и приложения. Объем работы составляет 48 страниц, объем списка литературы – 21 источник.

В первой главе описывается анализ предметной области. В данной главе указывается какие методы будут использованы для реализации проекта. Производится сравнение методов и архитектур.

Вторая глава посвящена выбору архитектуры нейронной сети. В этой главе рассказывается о выборе конкретного метода для создания архитектуры нейронной чети. Описывается из каких слоев должна состоять нейронная сеть, какие данные подаются на вход, а каких будут на выходе. Также указывается, какие методы будет применены для реализации проекта.

В третьей главе описывается практическая часть работы. В данной части указывается что было выполнено для реализации нейронной сети. Какой набор данных был выбран или собран. Какие использовались ресурсы для обучения нейронной сети, какой был выбран язык программирования и фреймворк. Демонстрируются выходные результаты полученной нейронной сети, а также результаты обработки шума на видео.

В четвертой главе производится сравнение с аналогами, что позволяет оценить преимущества и недостатки разрабатываемой системы по сравнению с существующими решениями на рынке. Анализ метрик аналогов помогает выявить уникальные особенности разрабатываемой системы, ее конкурентные преимущества и потенциальные области улучшения.

В приложении содержится архитектура реализуемой модели.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЗОР ЛИТЕРАТУРЫ

1.1. Типы нейронных сетей

Автоматическое распознавание и удаление шума на видео является важной задачей в области компьютерного зрения и обработки видеоданных. Шум на видео может появляться в результате различных факторов, таких как низкое качество съемки, недостаточное освещение или проблемы с оборудованием. Этот шум может значительно повлиять на качество видео, усложняя восприятие и анализ содержимого. Он может влиять на задачи компьютерного зрения, такие как распознавание объектов, сегментация и отслеживание движущихся объектов.

Ранее для удаления шума на видео применялись методы, основанные на фильтрации и статистическом анализе. Однако такие методы могут быть неэффективными в случае сложных шумовых паттернов, а также требуют настройки параметров для каждого конкретного видео. В связи с этим, рекуррентные нейронные сети (Recurrent Neural Network, RNN) и сверточные нейронные сети (Convolutional Neural Networks, CNN) стали привлекать все больше внимания в данной области.

Сверточные нейронные сети – это разновидность многослойного перцептрона, которая использует операции свертки для обработки изображений и видео. Они нашли применение в таких областях, как распознавание изображений, автопилотирование, медицинское прототипирование и обработка естественного языка.

Структура CNN состоит из нескольких слоев, включая – сверточные, субдискретизирующие, полносвязные и выпрямляющие слои. Пример архитектуры представлен на рисунке 2. Сверточные слои позволяют изучать локальные шаблоны на ранних слоях и более сложные шаблоны на более поздних слоях. Субдискретизирующие слои уменьшают размерность входных данных, уменьшая количество обучаемых коэффициентов и выигрывая в вычислительных ресурсах. Полносвязные слои используются для вывода

конечных результатов. Выпрямляющий слой используется для нормализации выходных данных.

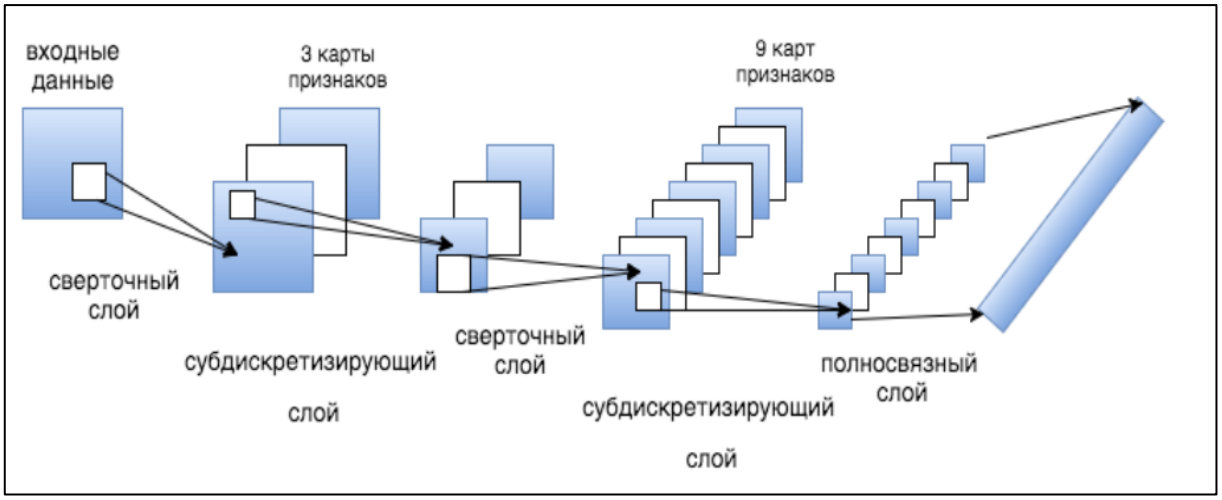

Рисунок 2 – Архитектура сверточной нейронной сети

Основные свойства CNN включают в себя получение представлений, которые являются инвариантными по отношению к переносу, пространственно-иерархическую модель, которая изучает локальные шаблоны, и операцию пулинга, которая уменьшает размерность выходных данных, сохраняя основные характеристики входных данных. Пример операции подвыборки указан на рисунке 3.

Основные элементы CNN включают – сверточный слой, пулинг, нормализацию по батчу и полносвязный слой. Сверточный слой использует ядра для изучения локальных шаблонов входных данных. Пулинг уменьшает размерность выходных данных, сохраняя основные характеристики входных данных. Нормализация по батчу используется для улучшения обучения нейросети. Полносвязный слой используется для вывода конечных результатов. Пример операции свертки представлен на 4 рисунке.

Обучение CNN включает в себя несколько этапов, включая инициализацию весов, вычисление выходных данных, вычисление ошибки и обновление весов. Обучение происходит путем обратного распространения ошибки, которая позволяет нейросети улучшать свою точность путем корректировки весов.

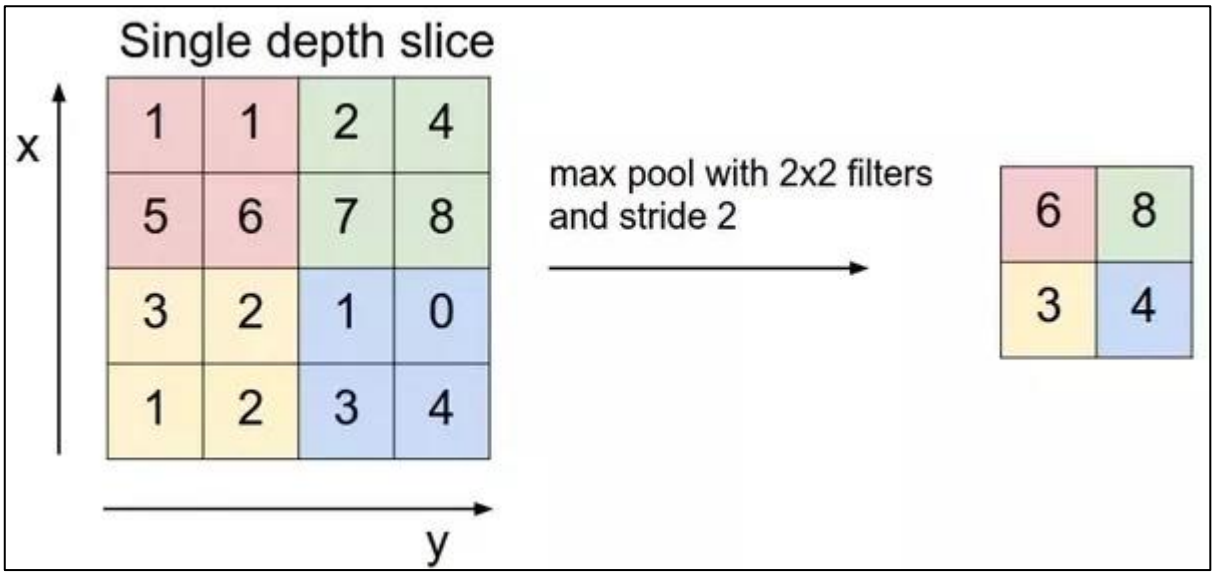

Рисунок 3 – Операция подвыборки с функцией максимума

Сверточные нейронные сети широко используются в компьютерном зрении для решения различных задач, таких как распознавание изображений, классификация, сегментация и детекция объектов.

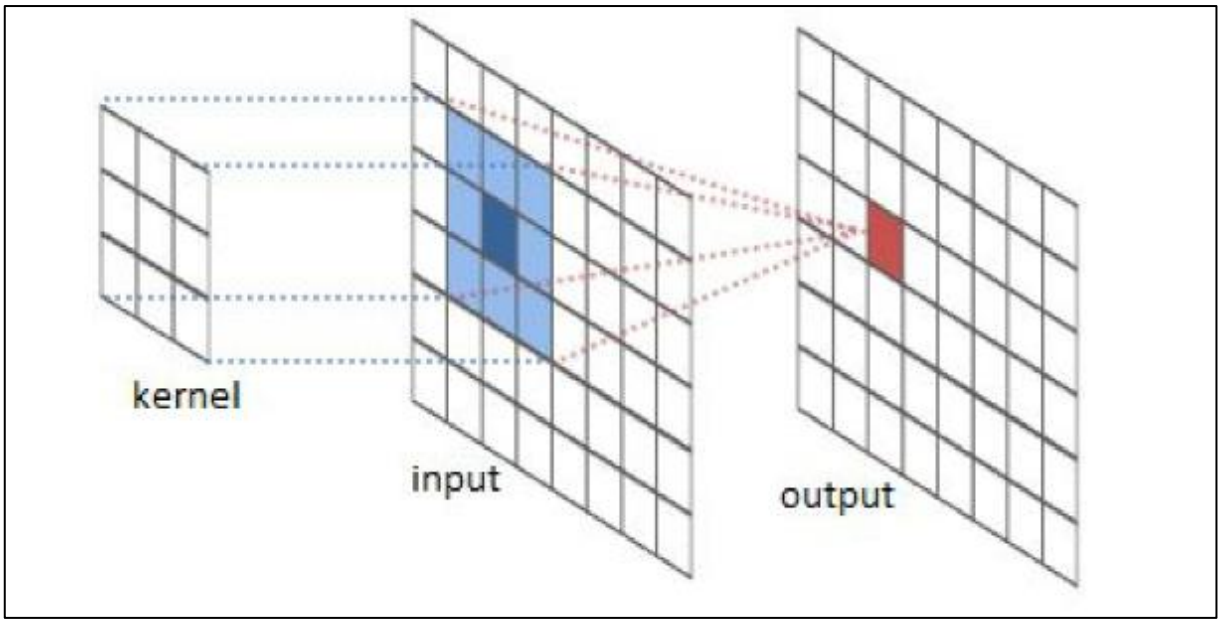

Рисунок 4 – Визуализация операции свертки

Основным преимуществом сверточных нейронных сетей является их способность изучать локальные шаблоны в изображениях, что позволяет им обнаруживать характерные особенности объектов, такие как уши, глаза и нос у кошки. Кроме того, сверточные нейронные сети способны выделять

более сложные шаблоны, такие как форма и ориентация объектов, что позволяет им распознавать объекты на изображениях с различными масштабами, углами зрения и освещением.

RNN – это класс нейронных сетей, которые способны обрабатывать и моделировать последовательности данных. Они имеют способность учитывать контекст из прошлых кадров, что делает их особенно полезными для обработки видеоданных. Основные типы RNN, используемые в задачах обработки видео, включают в себя LSTM (долгая краткосрочная память) и GRU (обновление воротной единицы). Эти модели имеют способность запоминать информацию из прошлых кадров и использовать ее для принятия решений о текущем кадре.

Краткое описание основных архитектур рекуррентных нейронных сетей представлено ниже.

1. RNN – простая архитектура с одним типом ячейки и ограниченной способностью к улавливанию долгосрочных зависимостей.
2. LSTM – Более сложная архитектура с четырьмя внутренними компонентами – ячейкой памяти, воротами забывания, входа и выхода.
3. GRU – Упрощенная версия LSTM с меньшим количеством внутренних компонентов, что делает его более эффективным с точки зрения ресурсов.

Применение рекуррентных нейронных сетей для автоматического распознавания и удаления шума на видео становится все более перспективным направлением исследований. Этот подход позволяет модели адаптироваться к различным шумовым паттернам и повышает качество видеозаписей, делая их более чистыми и понятными. Он также может улучшить результаты других задач компьютерного зрения, таких как распознавание объектов, сегментация и отслеживание движущихся объектов.

На рисунке 5 представлены архитектуры различных рекуррентных нейронных сетей.

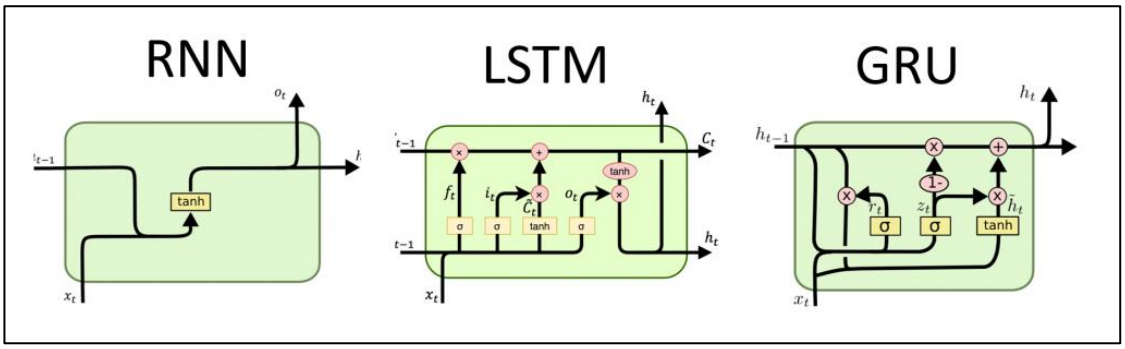

Рисунок 5 – Архитектуры различных рекуррентных нейронных сетей

Предполагается, что результаты исследования будут полезными для улучшения качества видеозаписей, а также для применения в различных областях, связанных с обработкой видеоданных.

1.2. Сравнение архитектур рекуррентных нейронных сетей

Перед выбором оптимальной архитектуры для данной работы необходимо провести сравнительный анализ основных архитектур.

В таблице 1 представлены преимущества и недостатки архитектур рекуррентных нейронных сетей.

Таблица 1 – Сравнительный анализ основных архитектур нейронных сетей

Архитектура нейронной сети	**Преимущества**	**Недостатки**
RNN (Recurrent Neural Networks)	1. RNN является более общим классом рекуррентных моделей, позволяющим обрабатывать последовательные данные. 2. Простота архитектуры RNN позволяет его эффективно применять в различных задачах обработки последовательных данных.	Проблема затухания градиента ограничивает способность RNN обрабатывать долгосрочные зависимости.

Окончание таблицы 1

Архитектура нейронной сети	Преимущества	Недостатки
GRU (Gated Recurrent Unit)	1. GRU, по сравнению с LSTM, имеет меньше параметров, что делает его более простым в использовании и менее затратным с точки зрения ресурсов. 2. GRU имеет механизмы обновления и забывания в одной ячейке, что может упростить обучение сети.	GRU может быть менее эффективным в задачах, где важно сохранение долгосрочных зависимостей
LSTM (Long Short–Term Memory)	1. LSTM обладает возможностью запоминать долгосрочные зависимости в данных благодаря механизму забывания и обновления информации в ячейках памяти. 2. Эффективно применяется в задачах, где важно сохранение контекста и избежание проблемы затухания градиента.	LSTM может быть более сложным и затратным по сравнению с другими моделями, что может привести к проблеме переобучения на небольших наборах данных.

Исходя из таблицы 1, можно заметить, что выбор архитектуры зависит от конкретной задачи и набора данных. В некоторых случаях GRU может быть более эффективным и быстрым в обучении, в то время как LSTM может быть лучше подходить для моделирования долгосрочных зависимостей. LSTM обычно используется при работе с задачами, где важны долгосрочные зависимости, GRU может быть предпочтителен при ограниченных ресурсах или для более простых задач, а RNN является базовым и универсальным типом рекуррентной нейронной сети. В данной задаче предпочтительно использовать GRU или LSTM.

Благодаря сравнению, было принято решение использовать слой LSTM. Так как будет использоваться небольшой набор данных, а также он отлично комбинируется со сверточными нейронными сетями.

1.3. Примеры готовых решений на основе нейронных сетей

Существуют несколько моделей распознавания и удаления шума на видео с помощью рекуррентных и сверточных нейронных сетей. Одним из таких методов является использование рекуррентных сверточных нейронных сетей (RCNN) для обработки видеоданных.

Область распознавания и удаления шума на видео с помощью рекуррентных нейронных сетей (RNN) активно исследуется в научной литературе. Ниже приведен обзор нескольких актуальных статей, исследующих эту тему.

Video Denoising Using Spatio-Temporal Recurrent Convolutional Networks [4] – в этой статье предложен метод удаления шума на видео с помощью специально-временных рекуррентных сверточных нейронных сетей (ST-RCNN).

Модель ST-RCNN варьирует свою архитектуру для учета пространственных и временных зависимостей в видеосигналах, и показывает хорошие результаты в сравнении с другими методами удаления шума на видео. Архитектура данной нейронной сети представлена на рисунке 6.

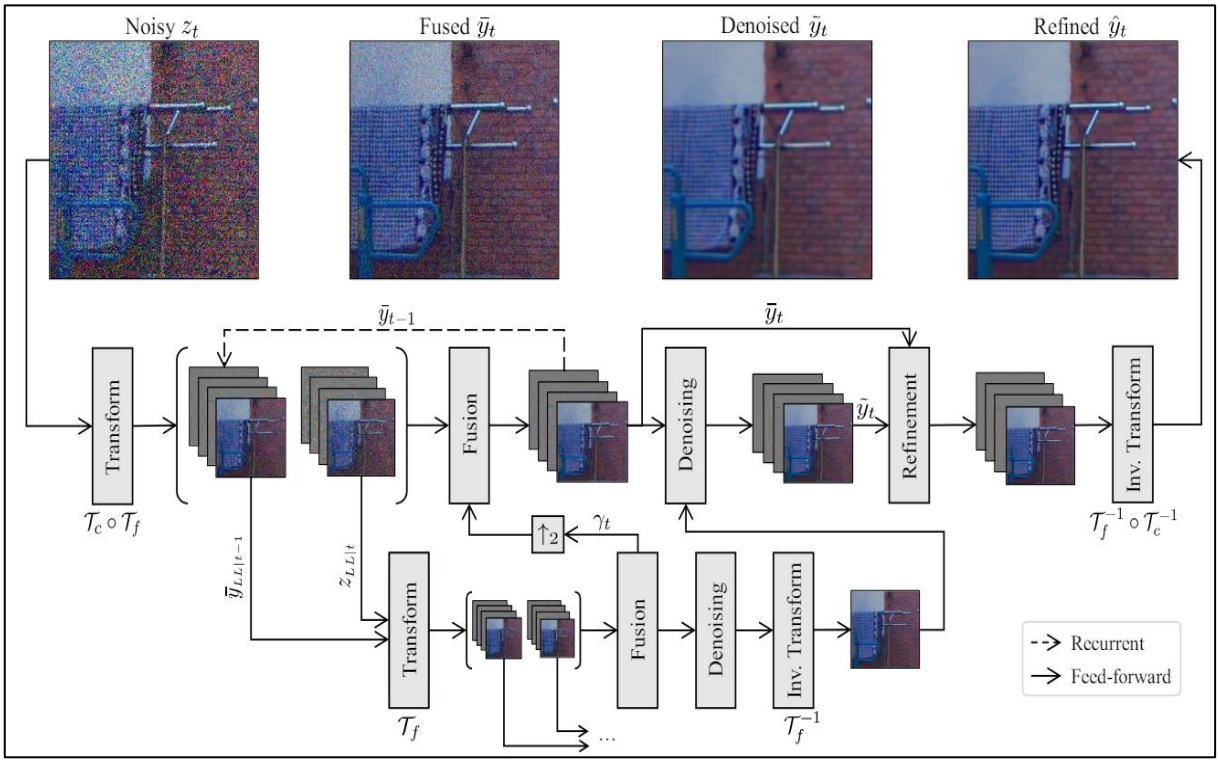

Рисунок 6 – Архитектура ST-RCNN

Denoising Videos with Recurrent Convolutional Networks [5] – в этой работе исследуется проблема удаления шума на видео с использованием рекуррентных сверточных нейронных сетей (RCNN). Авторы предлагают метод, который комбинирует RCNN с еще одной моделью, называемой двунаправленным модулем дополнения контекста (Bilateral Context Aggregation Module), для учета контекстуальных информаций при удалении шума.

Данный метод показывает хорошие результаты в сравнении с другими алгоритмами удаления шума на видео. DVRCN представлен на рисунке 7.

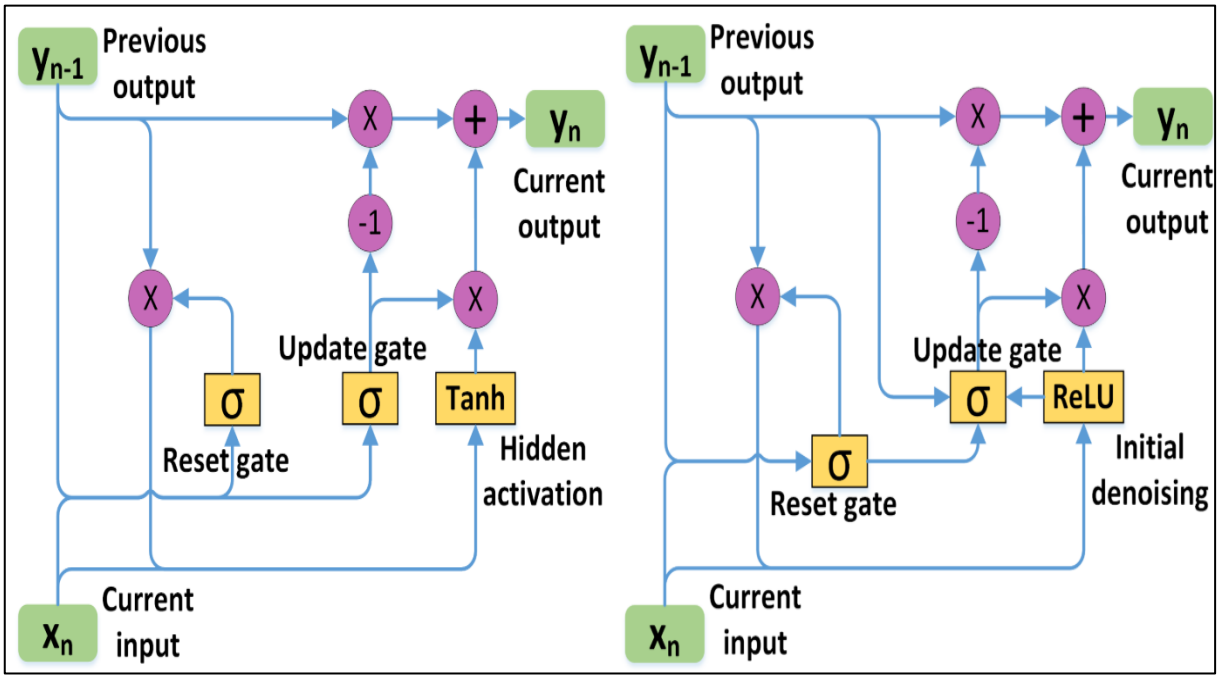

Рисунок 7 – Архитектура DVRCN

Deep Recurrent Neural Network for Video Denoising [6] – в данной работе представлен метод удаления шума на видео с использованием глубокой рекуррентной нейронной сети (DRNN). Авторы применяют DRNN для изучения временных зависимостей в видеосигналах и предлагают новый алгоритм обучения для улучшения удаления шума. Эксперименты показывают, что предложенный метод превосходит другие алгоритмы удаления шума на видео.

Video Denoising using Recurrent Neural Networks [7] – в этой статье рассматривается проблема удаления шума на видео с использованием рекуррентных нейронных сетей. Авторы предлагают метод, в котором RNN использует историю кадров для предсказания следующего кадра без шума. Кроме того, авторы также исследуют использование генеративно-состязательных сетей (GAN) для удаления шума на видео и показывают улучшение качества видеоконтента. Архитектура генеративно-состязательных сетей представлена на рисунке 8.

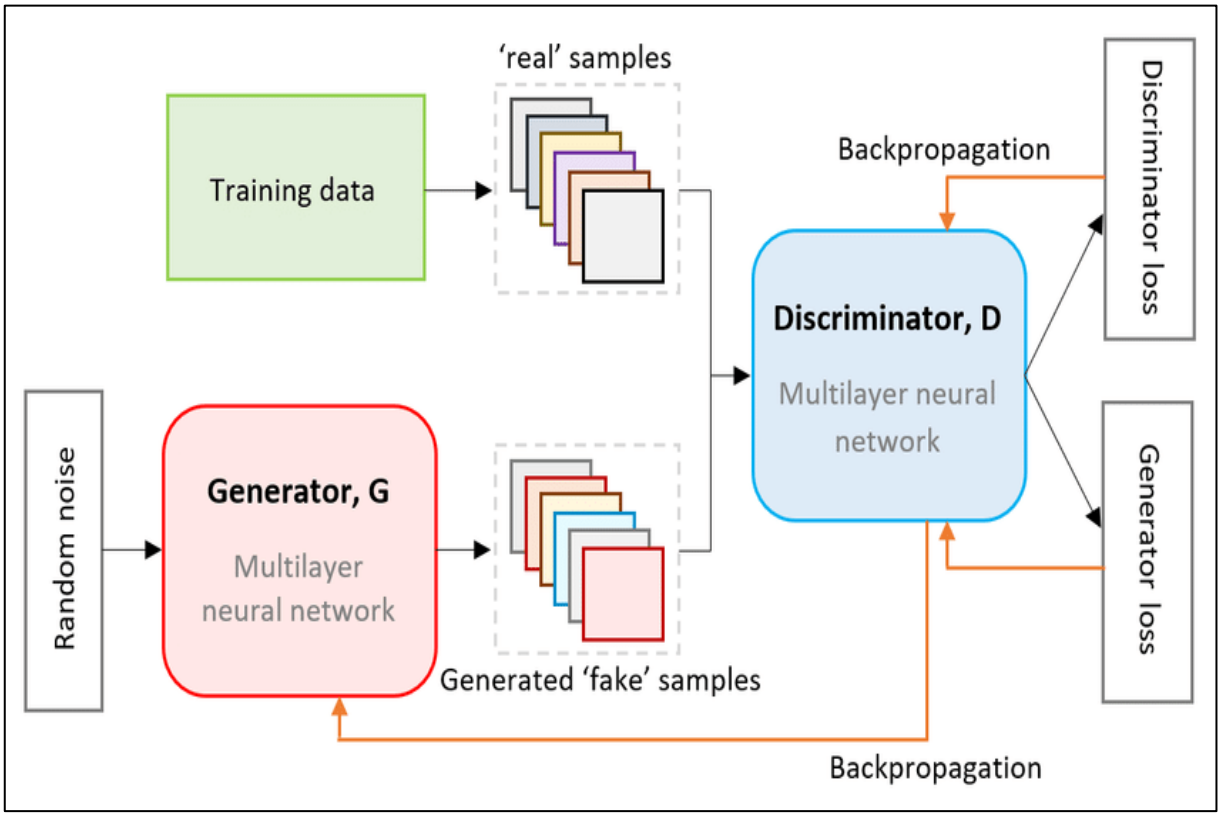

Рисунок 8 – Архитектура GAN

В настоящее время исследования в данной области активно продолжаются, и появляются все новые методы и подходы, направленные на улучшение качества видео контента.

Целью данной выпускной квалификационной работы является разработка и обучение рекуррентной нейронной сети, которая способна распознавать и удалять шум на видео с высокой точностью. Для достижения этой цели используется архитектуры RNN и CNN, а также методы оптимизации.

Оптимальная конфигурация модели будет найдена путем экспериментов с различными параметрами и архитектурами.

1.4. U-net сети

U-NET [8] находит широкое применение в различных областях компьютерного зрения. В медицине она используется для сегментации медицинских изображений, таких как МРТ, КТ и микроскопические снимки. В автономных транспортных средствах U-NET применяется для обнаружения дорожных разметок, знаков и свободных пространств на дороге. В спутниковом картографировании она позволяет автоматически распознавать различные типы земель, зданий и дорог. Кроме того, U-NET используется в точном сельском хозяйстве для различения посевов и сорняков.

Таким образом, U–NET является одной из ключевых архитектур сверточных нейронных сетей, которая зарекомендовала себя как эффективный инструмент для решения широкого спектра задач сегментации изображений в различных областях. Ее симметричная структура, использование skip-соединений и способность работать с ограниченными наборами данных делают ее одной из наиболее популярных и востребованных архитектур в современном компьютерном зрении.

В контексте шумоподавления на изображении, U-Net может быть адаптирована для обработки каждого пикселя изображения, учитывая его окружение и контекст. Архитектура U-Net для шумоподавления на изображении включает в себя сверточные слои для извлечения признаков, слои пулинга для субдискретизации, а также skip-соединения для передачи информации между различными уровнями сети. Это позволяет модели эффективно удалять шумы, сохраняя при этом детали и текстуры изображения.

Процесс шумоподавления на изображении с использованием U-Net включает в себя подачу зашумленного изображения на вход сети, обработку его сверточными слоями для извлечения признаков, удаление шумов и ар-

тефактов, а затем восстановление чистого изображения. Этот процесс позволяет значительно улучшить качество изображения, сделать его более четким и качественным. Использование U-Net для шумоподавления на изображении является эффективным подходом, особенно в областях, где высокое качество изображения играет важную роль, таких как медицинская диагностика, фотография, обработка изображений в реальном времени и другие области. U-Net позволяет улучшить визуальное восприятие изображений, убрав шумы и повысив их четкость и качество.

На рисунке 9 представлен пример удаление шума на наборе данных Fashion-MNIST [9], который содержит 70 тысяч изображений одежды и аксессуаров.

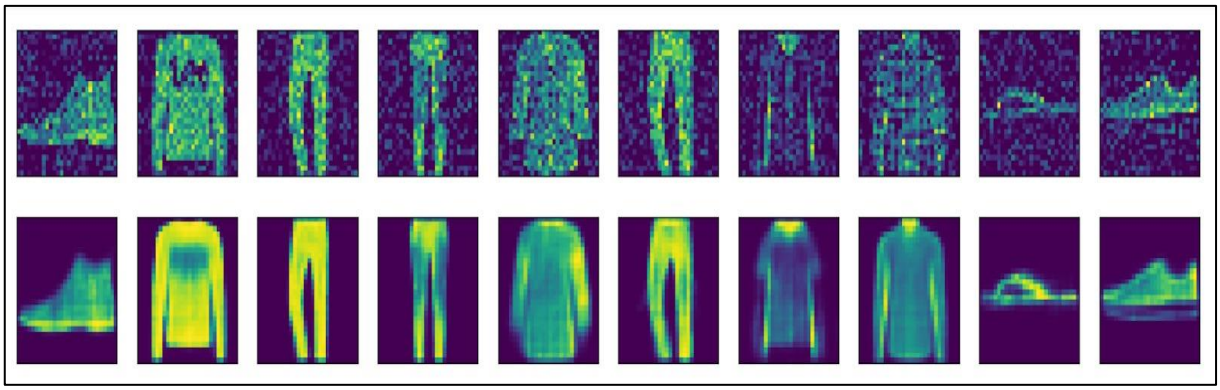

Рисунок 9 – Пример удаление шума на наборе данных Fashion-MNIST

В данной работе использование архитектуры U-Net будет ключевым, так как она обладает уникальными свойствами, позволяющими эффективно решать задачи шумоподавления на изображениях.

Адаптированная для шумоподавления, U-Net позволяет эффективно обрабатывать каждый пиксель изображения, учитывая его окружение, что приводит к улучшению качества изображения путем удаления шумов и артефактов. Таким образом, было принято решение использовать U-Net в данной работе, так как она обеспечивает высокую точность и качество обработки изображений.

2. ПРОЕКТИРОВАНИЕ

Функциональные требования к системе

Система должна соответствовать следующим функциональным требованиям.

1. Система должна распознавать шум на видео.
2. Система должна удалять шум из видео.
3. Система должна сохранять результат обработки видео в новом видеофайле.

Нефункциональные требования к системе

Система должна соответствовать следующим нефункциональным требованиям.

1. Система должна быть написана на языке программирования Python [10].
2. Система должна использовать нейронную сеть, содержащую рекуррентные слои.
3. Система должна иметь возможность обрабатывать видео различных разрешений и форматов.
4. Система должна обрабатывать видео покадрово.
5. Система должна использовать модель удаления шума на изображениях.
6. Система должна распознавать различные виды шумов на видео.

Проектирование архитектуры нейронной сети

Для реализации данной задачи была выбрана специальная архитектура искусственных нейронных сетей – U-net с добавлением рекуррентных слоев [11].

U-NET – это архитектура сверточной нейронной сети, которая была впервые предложена в 2015 году группой исследователей во главе с Олафом Роннебергером, Филиппом Фишером и Томасом Броком. Она была разрабо-

тана специально для задач сегментации медицинских изображений, но впоследствии нашла широкое применение и в других областях компьютерного зрения.

Основная особенность U-NET заключается в ее симметричной архитектуре, состоящей из двух частей – сжимающего пути (encoder) и расширяющего пути (decoder). Сжимающий путь отвечает за захват контекстной информации и уменьшение пространственного разрешения, в то время как расширяющий путь декодирует данные и использует информацию из сжимающего пути через специальные skip-соединения [12] для генерации финальной карты сегментации. Пример архитектуры U-net представлен на рисунке 10.

Также красным цветом обозначено «бутылочное горлышка», где в разработанной архитектуре находится рекуррентные слои.

Архитектура U-Net зарекомендовала себя как эффективный инструмент для решения задач сегментации изображений, продемонстрировав высокую точность на международных соревнованиях, таких как Kaggle. Впоследствии, она нашла широкое применение в медицинском программном обеспечении благодаря своим возможностям точного детектирования. Данная архитектура представляет собой двухэтапный процесс анализа изображений. Первый этап включает в себя процесс понижающей дискретизации, а второй – повышающей дискретизации. В качестве инструментов моделирования архитектуры U-Net используются открытые библиотеки OpenCV [13] и Keras [14], работающие на основе фреймворка TensorFlow и языка программирования Python.

Схема архитектуры U-Net относится к классу сверточных нейронных сетей. В отличие от классических полносвязных нейронных сетей, U-Net не содержит полносвязных слоев, а использует карты признаков между входным и выходным слоями, а также ядра свертки для извлечения и усиления характеристик искомых объектов. На первом этапе, этапе понижающей дис-

кретизации, U-Net получает на вход трехканальное изображение фиксированного размера, кратного степени двойки, чтобы сохранить каждый пиксель при применении методов субдискретизации [15].

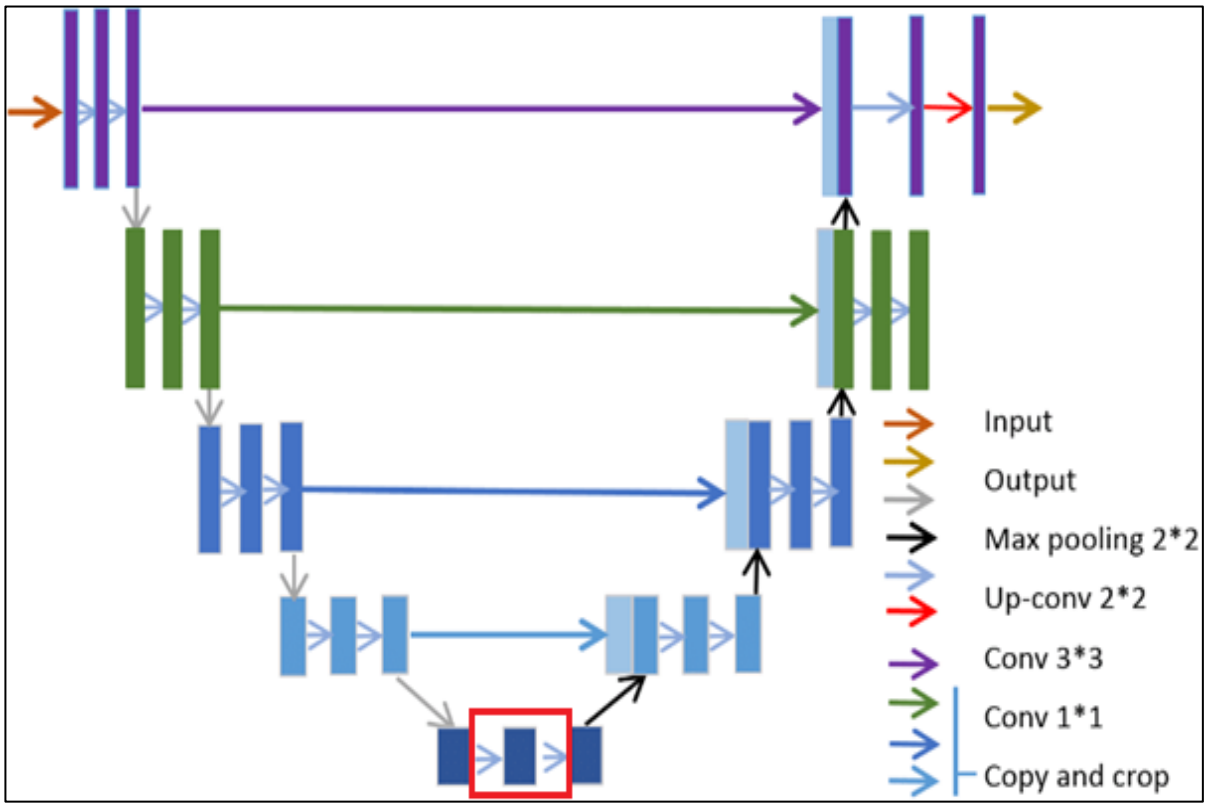

Рисунок 10 – Архитектура U–NET

Ядра свертки на этом этапе заполняются случайными значениями из диапазона [-10; 10]. Последовательное применение двух сверток и субдискретизации позволяет формировать карты признаков искомых объектов на разных уровнях.

На втором этапе, этапе повышающей дискретизации, происходит восстановление собранных на первом этапе признаков искомых объектов. Применяя методы транспонированной свертки, повышающей дискретизации и конкатенации карт признаков с первого этапа, архитектура U-Net собирает восстановленные характеристики объектов, сохраняя при этом высокоуровневые особенности.

После сегментации изображения U-Net выдает на выходе одноканальное бинарное изображение, которое сравнивается с соответствующей маской для определения ошибки модели. Для корректировки весов ядер свертки и улучшения точности сегментации применяется метод обратного распространения ошибки.

Одним из ключевых преимуществ U-NET является ее способность эффективно работать даже при ограниченном наборе данных для обучения. Это особенно важно в специализированных областях, таких как медицинская диагностика, где получение большого количества размеченных данных может быть затруднительно. Благодаря своей архитектуре, U-NET способна извлекать максимум информации даже из небольших наборов данных.

Архитектура реализуемой модели

Реализуемая модель основана на U-Net архитектуре, дополненной рекуррентными слоями в качестве «бутылочного горлышка», что позволяет ей также учитывать пространственные зависимости на изображении между энкодером и декодером.

Начиная с входного слоя, который принимает изображения размером 128×128 пикселей с тремя цветовыми каналами, модель последовательно применяет серию сверточных слоев. Эти слои постепенно увеличивают количество фильтров, извлекая все более сложные и абстрактные признаки из входных данных. Одновременно с этим, размер изображения уменьшается с помощью стратегии пулинга, реализованной через увеличение шага сверток. Для предотвращения переобучения модели, в архитектуру включены слои исключения (dropout), которые случайным образом отключают определенную долю нейронов на каждом этапе обучения. Это помогает модели обобщать извлекаемые признаки и повышает ее устойчивость к шуму и искажениям в данных. После прохождения через последовательность сверточных и исключающих слоев, результаты энкодера преобразуются в формат, подходящий для использования рекуррентных слоев.

В архитектуре модели, мной были спроектированы рекуррентные слои в качестве «бутылочного горлышка» между энкодером и декодером.

Эти рекуррентные слои позволяют модели учитывать пространственные зависимости на изображении, сохраняя важную информацию, полученную на ранних этапах обработки, и передавая ее для формирования окончательного выходного изображения.

Выходные данные рекуррентных слоев затем преобразуются обратно в формат изображений и подаются на вход декодера. Декодер состоит из слоев развертки, которые постепенно увеличивают размерность изображения, возвращая его к исходному размеру.

Важной особенностью декодера является использование слоев объединения (skip-соединения), которые соединяют результаты декодера с соответствующими результатами энкодера.

Это позволяет модели сохранять важную пространственную информацию, полученную на ранних этапах обработки, и использовать ее для формирования окончательного выходного изображения.

Завершающим этапом является выходной слой, который преобразует результаты декодера в изображение с тремя цветовыми каналами. Таким образом, вся архитектура модели представляет собой сложную и многоуровневую систему, способную эффективно обрабатывать и анализировать изображения, учитывая как пространственные, так и временные аспекты данных. Архитектура реализуемой модели представлена на рисунке 11.

Метод удаления шума на видео

Для удаления шума на видео был разработан метод, представленный в виде функции удаления шума на изображении, который использует предварительно обученную нейронную сеть для обработки изображений и функции разбиения видео на изображения, которая применяет этот метод к каждому кадру видео. В функции удаления шума на изображении, изображение

разбивается на фрагменты с установленным размером пикселей, нормализуется, обрабатывается нейронной сетью для удаления шума, а затем денормализуется и возвращается в исходное изображение.

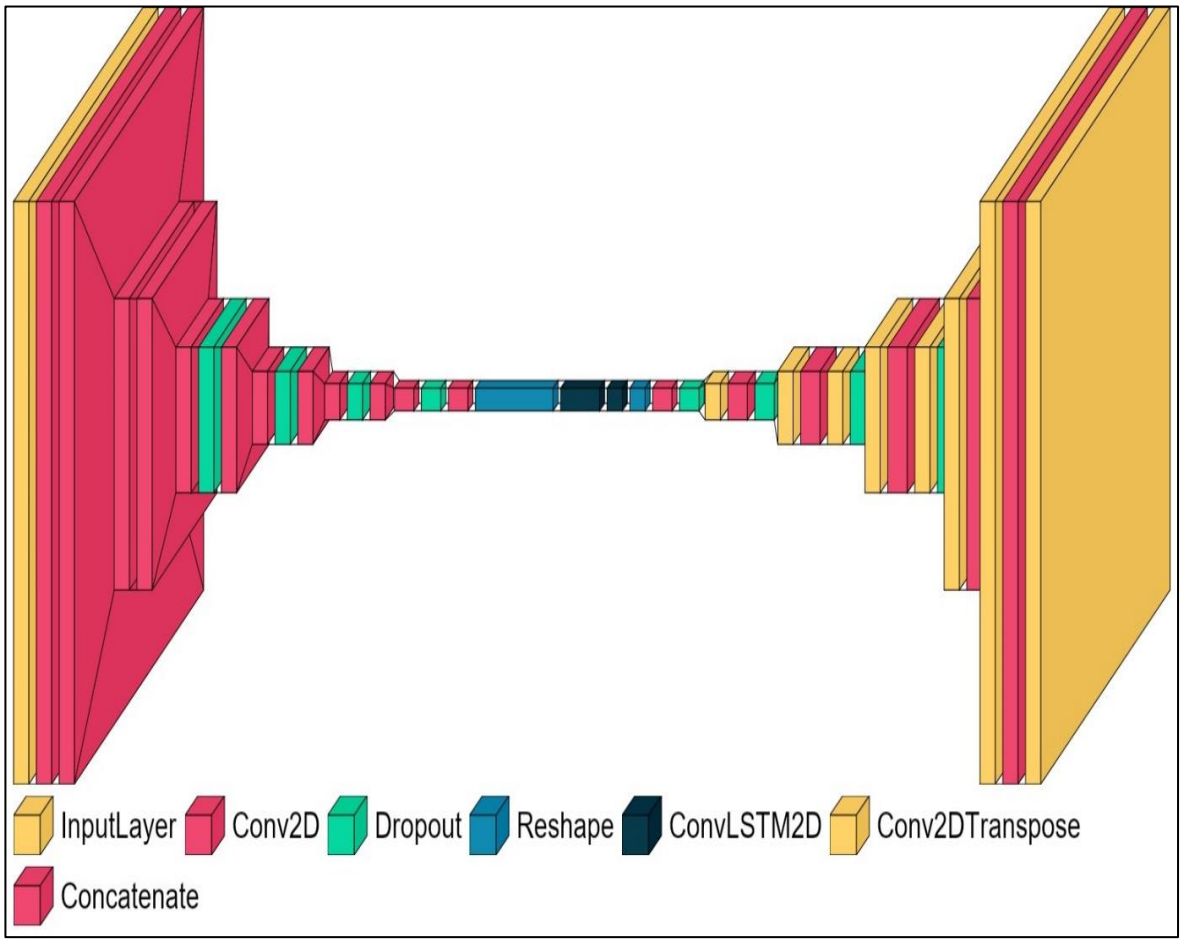

Рисунок 11 – Архитектура проектируемой сети

Функция разбиения видео на изображения начинает свою работу с открытия видеофайла, после чего последовательно обрабатывает каждый кадр видео. Для каждого кадра выполняется процесс удаления шума с использованием предварительно обученной модели. После обработки каждого кадра, функция объединяет оригинальный и обесшумленные кадры, создавая видеопоследовательность, в которой каждый кадр содержит как исходное изображение, так и его обесшумленную версию. На рисунке 12 представлена диаграмме деятельности, которая отражает работу метода удаления шума.

Этот процесс позволяет визуально сравнить оригинальное и обработанное видео, демонстрируя эффективность удаления шума. В результате работы функции создается новый видеофайл, который содержит улучшенную версию исходного видеоматериала, где шум значительно снижен или удален, что способствует повышению качества и четкости изображения.

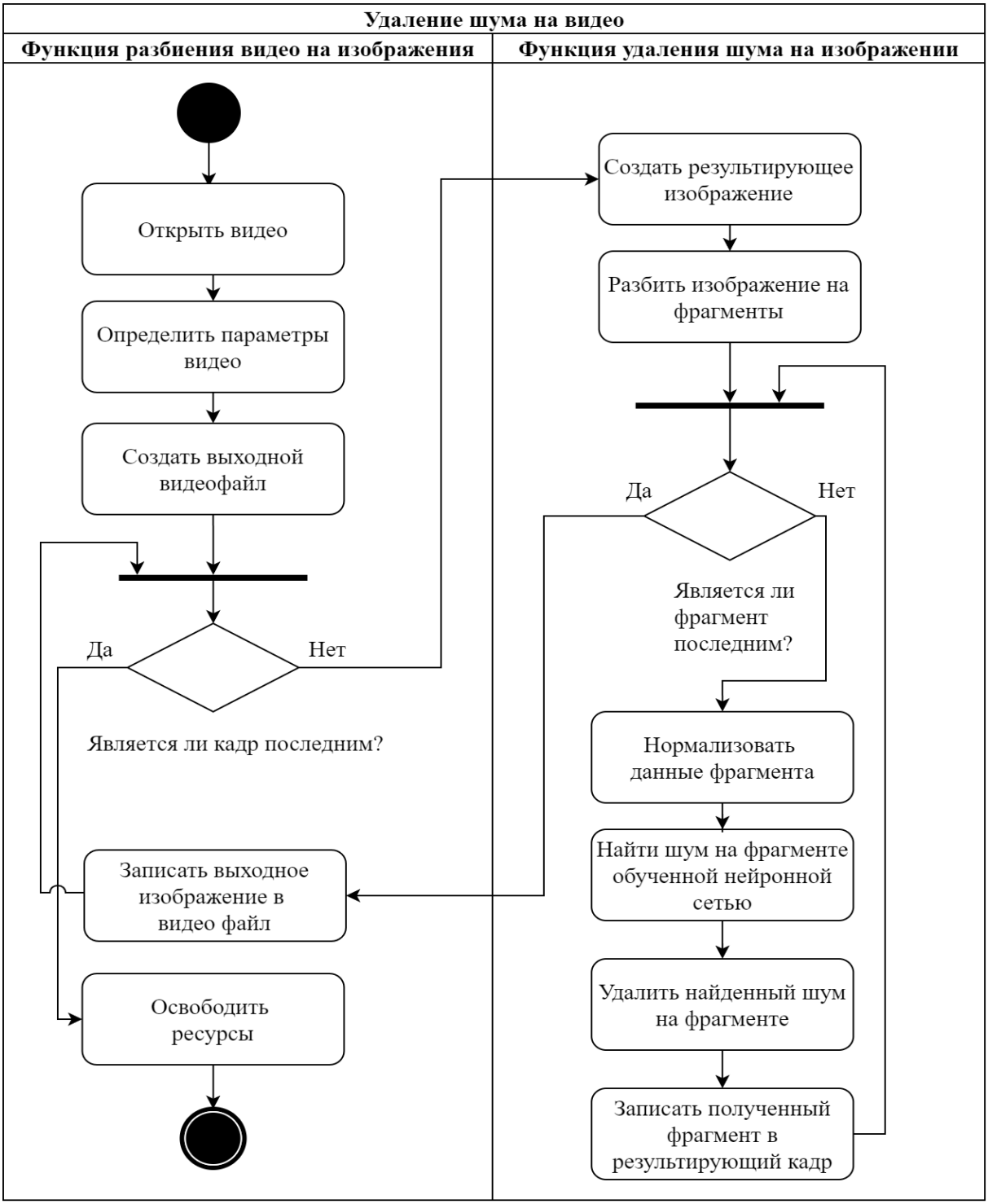

Рисунок 12 – Диаграмма деятельности

Данный подход позволяет эффективно удалять шум из видео, используя глубокое обучение и обработку кадров по частям.

Набор данных для обучения модели

Для обучения модели было решено использовать набор данных Flickr30k [16]. Данный датасет представляет собой набор изображений, который содержит 31783 изображения, каждое сопровождается пятью описаниями на естественном языке. Данный датасет был создан для обучения и оценки моделей компьютерного зрения и обработки естественного языка. Изображения в датасете охватывают различные сцены и объекты, что делает его разнообразным и подходящим для различных задач машинного обучения. В нашем случае, мы будем добавлять шум на изображения для создания подходящего под данную задачу набора данных. Пример изображения из набора данных Flickr30k представлен на рисунке 13.

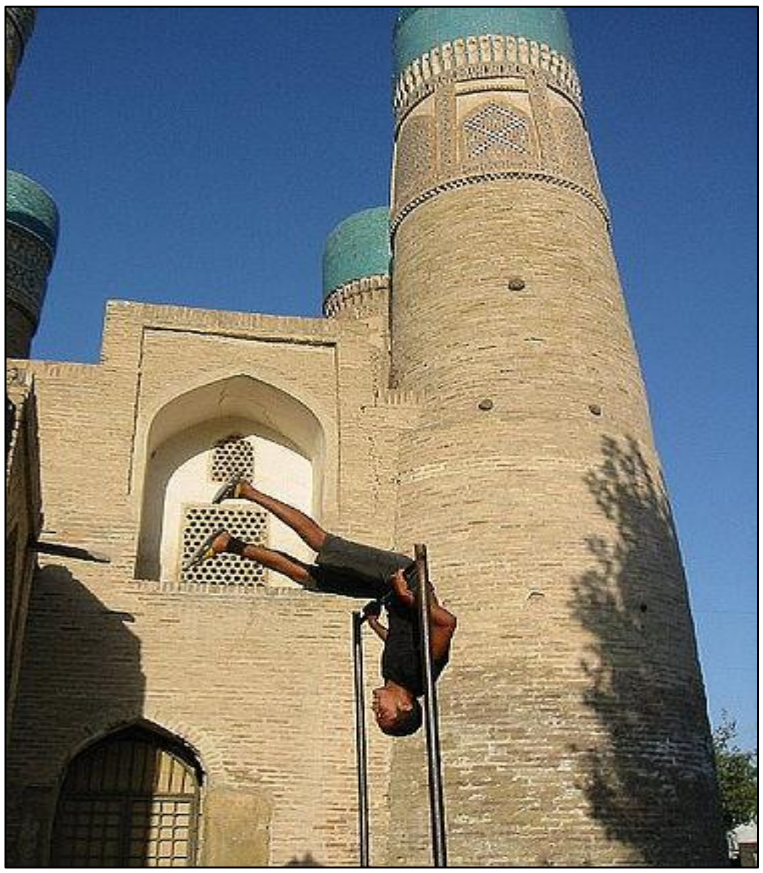

Рисунок 13 – Изображение из датасета Flickr30k

3. РЕАЛИЗАЦИЯ

3.1. Создание набора данных

Для создания набора данных (датасета), необходимо определить последовательность значимых процедур по предварительной обработке изображений с целью подготовки данных для обучения и проверки модели машинного обучения с применением фреймворка TensorFlow [17].

Изначально происходит загрузка данных изображений из корневой директории с помощью встроенной в TensorFlow функции. Данные разделяются на тренировочный и валидационный наборы в соотношении 90/10. Метки для изображений представлены в категориальном формате, что подходит для задач классификации. Размер пакета для обработки данных установлен на 64, а размер изображений изменен на 128×128 пикселей с использованием метода интерполяции «area». Доля данных для валидации составляет 10%, и данные перемешиваются с фиксированным случайным состоянием для воспроизводимости результатов.

Далее определяется функция `process(x, y)`, которая нормализует значения пикселей изображений. Каждое значение пикселя делится на 128 и затем вычитается 1, чтобы привести значения к диапазону [-1, 1]. Это стандартная практика для подготовки данных перед обучением модели глубокого обучения.

После нормализации данные тренировочного и валидационного наборов обрабатываются с помощью функции `map()`, чтобы применить нормализацию к каждому изображению. Затем создаются два набора данных `dataset` и `dataset_val`, которые кэшируются в памяти, предварительно загружают следующие пакеты данных для ускорения обучения и перемешиваются для обеспечения разнообразия данных при обучении модели.

Этот код является важным этапом в подготовке данных для обучения модели машинного обучения на изображениях. Он обеспечивает загрузку, нормализацию и оптимизацию данных, что помогает модели эффективно учиться на предоставленных данных и достигать лучших результатов при

обучении и валидации. Реализация обучающего и валидационного датасетов представлена в листинге 1.

Листинг 1 – Реализация обучающего и валидационного наборов данных

```
train_dataset,val_dataset = tf.keras.preprocessing.image_dataset_from_directory(
          'C:/Users/alexe/PycharmProjects/DIPLOM/flickr30k_images/',
          label_mode='categorical',
          batch_size=64,
          image_size=(128, 128),
          interpolation='area',
          validation_split=0.1,  # Процент данных для валидации
          subset='both',
          seed=123,
          shuffle=True
      )
      import numpy as np

def process(x, y):
        x /= 128.0
        x -= 1
        return x

normalized_ds = train_dataset.map(process)
normalized_val_ds = val_dataset.map(process)

AUTOTUNE = tf.data.AUTOTUNE

      dataset = normalized_ds.cache().prefetch(buffer_size=AUTOTUNE).shuffle(1000)
      dataset_val = normalized_val_ds.cache().prefetch(buffer_size=AUTOTUNE).shuffle(1000)
```

3.2. Реализация обучения нейронной сети

Как уже было указано, архитектура состоит из 3 составляющих – декодер, энкодер, скрытый слой «бутылочное горлышко».

Архитектура начинается с входного слоя, который принимает изображения размером 128×128 пикселей с 3 цветовыми каналами (RGB).

Размер принимаемого изображения был выбран с учетом аппаратных возможностей; доступные вычислительные ресурсы позволяли обучать сеть только на наборе данных размером 128×128 пикселей.

Декодер состоит из последовательности сверточных слоев с активацией «ReLU» и операцией `pooling`, которые постепенно уменьшают про-

странственные размеры изображения, увеличивая при этом количество признаков. Эти слои формируют энкодер, который извлекает важные признаки из входного изображения.

Результаты энкодера преобразуются в последовательность пространственных карт признаков, которые подаются на вход рекуррентным сверточным слоям ConvLSTM2D [18].

ConvLSTM2D – это двумерный сверточный слой с долгой краткосрочной памятью (LSTM), который сочетает в себе свойства сверточных нейронных сетей и рекуррентных нейронных сетей. ConvLSTM2D применяет операции свертки вместо полносвязных преобразований, что делает его эффективным для анализа пространственных шаблонов в последовательностях данных. Эти слои позволяют учитывать пространственные зависимости в данных.

После рекуррентных ConvLSTM2D слоев, данные к декодеру, который состоит из последовательности транспонированных сверточных слоев (Conv2DTranspose) с операцией `upsampling`. Эти слои постепенно восстанавливают пространственные размеры изображения, объединяя информацию из соответствующих слоев энкодера (с помощью Concatenate) для восстановления детальной информации.

Заключительный слой использует Conv2DTranspose для получения выходного изображения с 3 каналами (RGB). Код модели представлен в листинге 2.

Листинг 2 – Модель нейронной сети

```
inputs = Input((128, 128, 3))
x = Conv2D(32, 3, activation='relu', padding='same')(inputs)
res1 = Conv2D(32, 3, activation='relu', padding='same')(x)
x = Conv2D(64, 3, activation='relu', padding='same', strides=2)(res1)
res2 = Conv2D(64, 3, activation='relu', padding='same')(x)
x = Conv2D(128, 3, activation='relu', padding='same', strides=2)(res2)
x = Dropout(0.1)(x)
res3 = Conv2D(128, 3, activation='relu', padding='same')(x)
x = Conv2D(128, 3, activation='relu', padding='same', strides=2)(res3)
x = Dropout(0.1)(x)
res4 = Conv2D(128, 3, activation='relu', padding='same')(x)
x = Conv2D(128, 3, activation='relu', padding='same', strides=2)(res4)
x = Dropout(0.1)(x)
res5 = Conv2D(128, 3, activation='relu', padding='same')(x)
```

```
x = Conv2D(256, 3, activation='relu', padding='same', strides=2)(res5)
x = Dropout(0.1)(x)
res6 = Conv2D(256, 3, activation='relu', padding='same')(x)

# Конвертируем результаты энкодера в последовательности для ConvLSTM2D
convlstm_input = Reshape((-1, 4, 4, 256))(res6)
# Добавляем ConvLSTM2D слои
conv_lstm1 = ConvLSTM2D(128, 3, activation='relu', padding='same', return_se-
quences=True)(convlstm_input)
conv_lstm2 = ConvLSTM2D(64, 3, activation='relu', padding='same', return_se-
quences=False)(conv_lstm1)
# Преобразуем результаты ConvLSTM2D обратно в изображения
x = Reshape((4, 4, 64))(conv_lstm2) # Исправлено на 64, так как это количество
фильтров в ConvLSTM2D

# Декодер
x = Conv2D(256, 3, activation='relu', padding='same')(x)
x = Dropout(0.1)(x)
x = Conv2DTranspose(128, 3, activation='relu', padding='same', strides=2)(x)
x = Concatenate()([x, res5])
x = Dropout(0.1)(x)
x = Conv2DTranspose(128, 3, activation='relu', padding='same', strides=2)(x)
x = Concatenate()([x, res4])
x = Conv2DTranspose(128, 3, activation='relu', padding='same')(x)
x = Dropout(0.1)(x)
x = Conv2DTranspose(128, 3, activation='relu', padding='same', strides=2)(x)
x = Concatenate()([x, res3])
x = Conv2DTranspose(64, 3, activation='relu', padding='same')(x)
x = Dropout(0.1)(x)
x = Conv2DTranspose(64, 3, activation='relu', padding='same', strides=2)(x)
x = Concatenate()([x, res2])
x = Conv2DTranspose(32, 3, activation='relu', padding='same')(x)
x = Conv2DTranspose(32, 3, activation='relu', padding='same', strides=2)(x)
x = Concatenate()([x, res1])
# Выходной слой
x = Conv2DTranspose(3, 3, activation='linear', padding='same')(x)
outputs = x
unet = keras.Model(inputs, outputs)
```

В качестве оптимизатора был выбран «Adam», со скоростью обучения 0,0001.

Для реализации самописного обучения и валидации модели, был создан класс `Model`. В котором реализованы методы: добавление шума прелина в датасет – `apply_noise`, обучение модели – `training_step`, оценка качества модели на валидационном датасете – `validation_step`.

Метод `apply_noise` генерирует многомасштабный шум Перлина, который затем применяется к входным данным «x» в зависимости от параметра «t». Сначала, создается базовый гауссовский шум с помощью `tf.random.normal`. Затем, для различных масштабов (от 2 до 15), генерируются дополнительные шумы меньшего размера, которые масштабируются

до размера входных данных с помощью `tf.image.resize` и суммируются с базовым шумом. Это создает многомасштабный шум Перлина. Аналогичным образом, генерируется цветовой шум Перлина, который затем суммируется с серым шумом. Оба шума нормализуются путем деления на 8. Наконец, итоговый шум смешивается с входными данными x с использованием параметра «t». Если «t» равен 0, то на выходе будут только входные данные без шума. Если «t» равен 1, то на выходе будут только входные данные с полностью примененным шумом. Генерация шума представлена в листинге 3.

Листинг 3 – Генерация шума Перлина

```
def apply_noise(self, x, t):

        noise = tf.random.nor-
mal(shape=(tf.shape(x)[0],tf.shape(x)[1],tf.shape(x)[2],1))*5

        for tau in range(2,16):
            noise2 = tf.random.nor-
mal(shape=(tf.shape(x)[0],tf.shape(x)[1]//tau,tf.shape(x)[2]//tau,1))
            noise2 = tf.image.resize(noise2, (tf.shape(x)[1],
tf.shape(x)[2]))

            noise+=noise2
        noise = tf.concat([noise,noise,noise], axis = 3)

        color_noise = tf.random.nor-
mal(shape=(tf.shape(x)[0],tf.shape(x)[1],tf.shape(x)[2],3))*5
        for tau in range(2,16):
            color_noise2 = tf.random.nor-
mal(shape=(tf.shape(x)[0],tf.shape(x)[1]//tau,tf.shape(x)[2]//tau,3))
            color_noise2 = tf.image.resize(color_noise2, (tf.shape(x)[1],
tf.shape(x)[2]))
            color_noise+=color_noise2
        noise = noise + color_noise
        noise/=8
        b = 1. - t
        return x*b + noise*t, noise
```

Метод `training_step` используется для обучения модели. В нем применяется шум Перлина к входным данным, после чего с помощью градиентного спуска обновляются веса модели, чтобы минимизировать среднеквадратичную ошибку между предсказанным шумом и исходным шумом. Возвращается значение функции потерь в качестве метрики за контролем обучения. Метод обучения модели представлен в листинге 4.

Листинг 4 – Метод обучения модели

```
@tf.function
    def training_step(self, x):
        x_n, n = self.apply_noise(x, tf.random.uniform([],0.2,0.3))
        with tf.GradientTape() as tape_unet:
            predicted_noise = self.nn_unet(x_n, training=True)
            loss = tf.reduce_mean((predicted_noise - n) ** 2)

        grads_e = tape_unet.gradient(loss, self.nn_unet.trainable_variables)
        self.unet_optimizer.apply_gradients(zip(grads_e,
self.nn_unet.trainable_variables))
return loss
```

Метод `validation_step` используется для оценки модели на валидационных (проверочных) данных. Аналогично, к валидационным данным применяется шум Перлина, затем модель используется для предсказания шума.

Затем вычисляется среднеквадратичная ошибка между предсказанным шумом и исходным шумом на валидационных данных, и выходное значение возвращается. Полученное значение помогает отслеживать качество обучения модели и предотвращать переобучение. На листинге 5 представлен метод оценки модели.

Листинг 5 – Метод валидации модели

```
@tf.function
 def validation_step(self, x_val):
     x_n_val, n_val = self.apply_noise(x_val, tf.random.uniform([],0.2,0.3))
     predicted_noise_val = self.nn_unet(x_n_val, training=False)
     val_loss = tf.reduce_mean((predicted_noise_val - n_val) ** 2)
     return val_loss
```

Также была реализована функция `testing`, которая выводит результат удаления шума на нескольких картинках из валидационного набора данных.

Все методы и функции представленные выше используются в обобщающей функции. Данная функция является основной частью обучения нейронной сети. Она выполняет цикл обучения, который включает в себя шаги обучения и проверки модели на тестовых данных.

Вначале, функция инициализирует два списка для хранения историй потерь на обучающих и проверочных данных. Затем она задает количество эпох обучения (epochs).

В цикле по эпохам, функция выполняет следующие шаги.

1. Для каждого шага обучения, функция применяет метод обучение модели, к каждому батчу в обучающей выборке, суммируя потери каждого батча.

2. Для каждого шага проверки, функция применяет метод оценки модели к каждому батчу в проверочной выборке, суммируя потери.

3. Затем функция вычисляет средние потери на обучающих и проверочных данных за 1 эпоху, добавляет их в историю потерь и отображает график потерь на обучающих и проверочных данных. Пример графика представлен на рисунке 14.

4. В конце каждого шага, происходит вызов функции `testing` для визуальной оценки модели на тестовых данных. Пример визуального вывода представлен на рисунке 15.

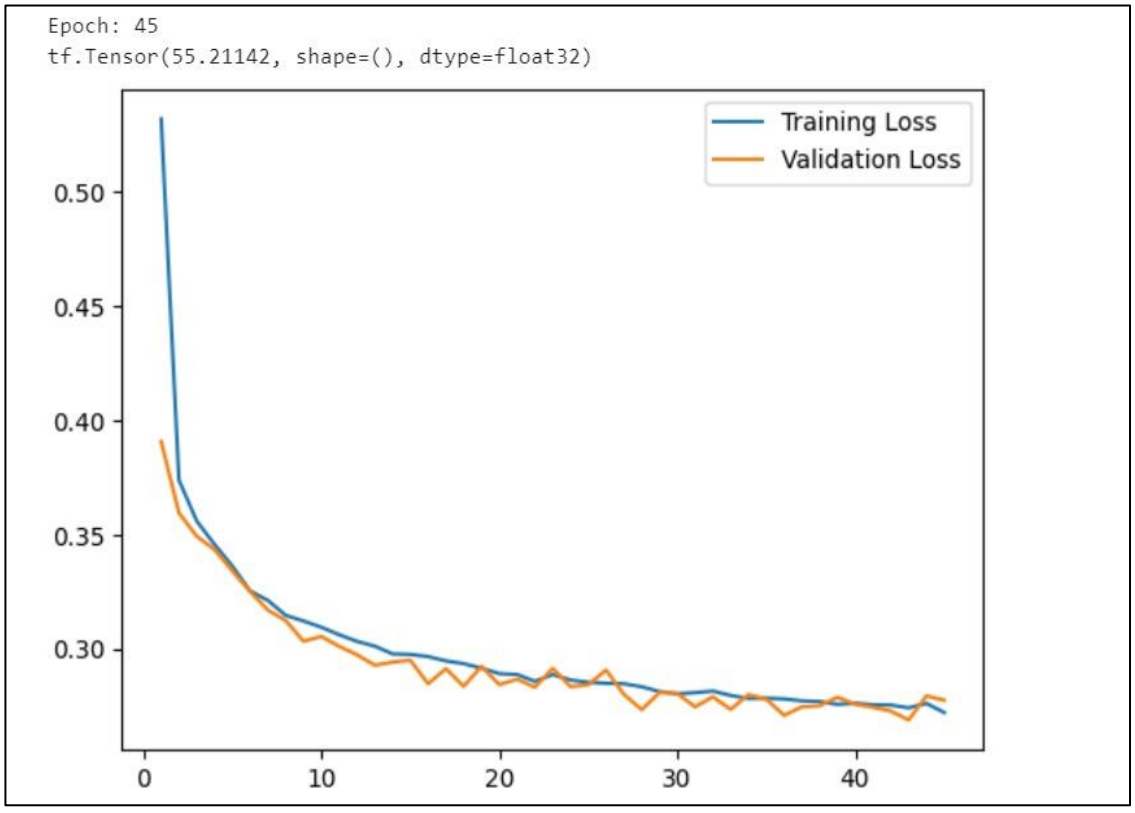

Рисунок 14 – График потерь на 45 эпохах

Таким образом, данная функция обеспечивает цикл обучения модели, отслеживает качество модели на обучающих и проверочных данных и отображает результаты обучения. На листинге 6 представлена обобщающая функция.

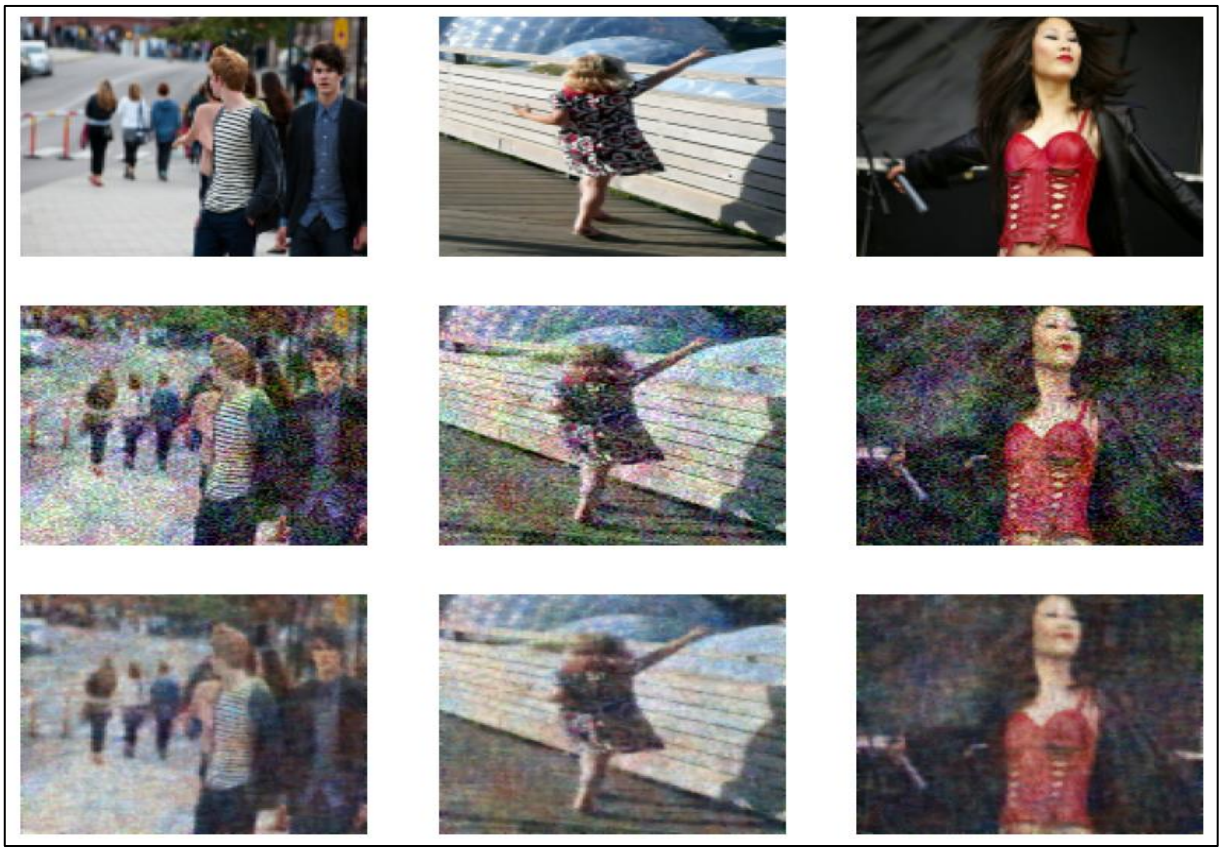

Рисунок 15 – Визуальный вывод обучения модели

Листинг 6 – Основная часть обучения нейронной сети

```
train_hist = []
val_hist = []
epochs = 100
for epoch in range(1, epochs + 1):
    train_loss = 0
    val_loss = 0
    # Training Step
    for step, x_train in enumerate(dataset):
        train_loss += model.training_step(x_train)

    # Validation Step
    for step, x_val in enumerate(dataset_val):
        val_loss += model.validation_step(x_val)
    clear_output(wait=True)
    print(f"Epoch: {epoch}")
    avg_train_loss = train_loss / len(train_dataset)
    train_hist.append(avg_train_loss)
    plt.plot(np.arange(1, epoch+1), train_hist, label='Training Loss')
    avg_val_loss = val_loss / len(val_dataset)
```

```
val_hist.append( avg_val_loss)
print(val_loss)
plt.plot(np.arange(1, epoch+1), val_hist, label='Validation Loss')
plt.legend()
plt.show()
testing()
```

Для обучения и тестирования нейронной сети использовалась видеокарта «GTX 1660 TI 6GB». Оптимальное количеством эпох – 80. Данное число эпох было выбрано так как – обучение сети на 45 эпохах было недостаточным, а на 100 эпохах скорость обучения сильно снижалась. Общее время обучения сети на 80 эпохах составило 7 часов. Графики потерь на 80 эпохах представлен на рисунке 16.

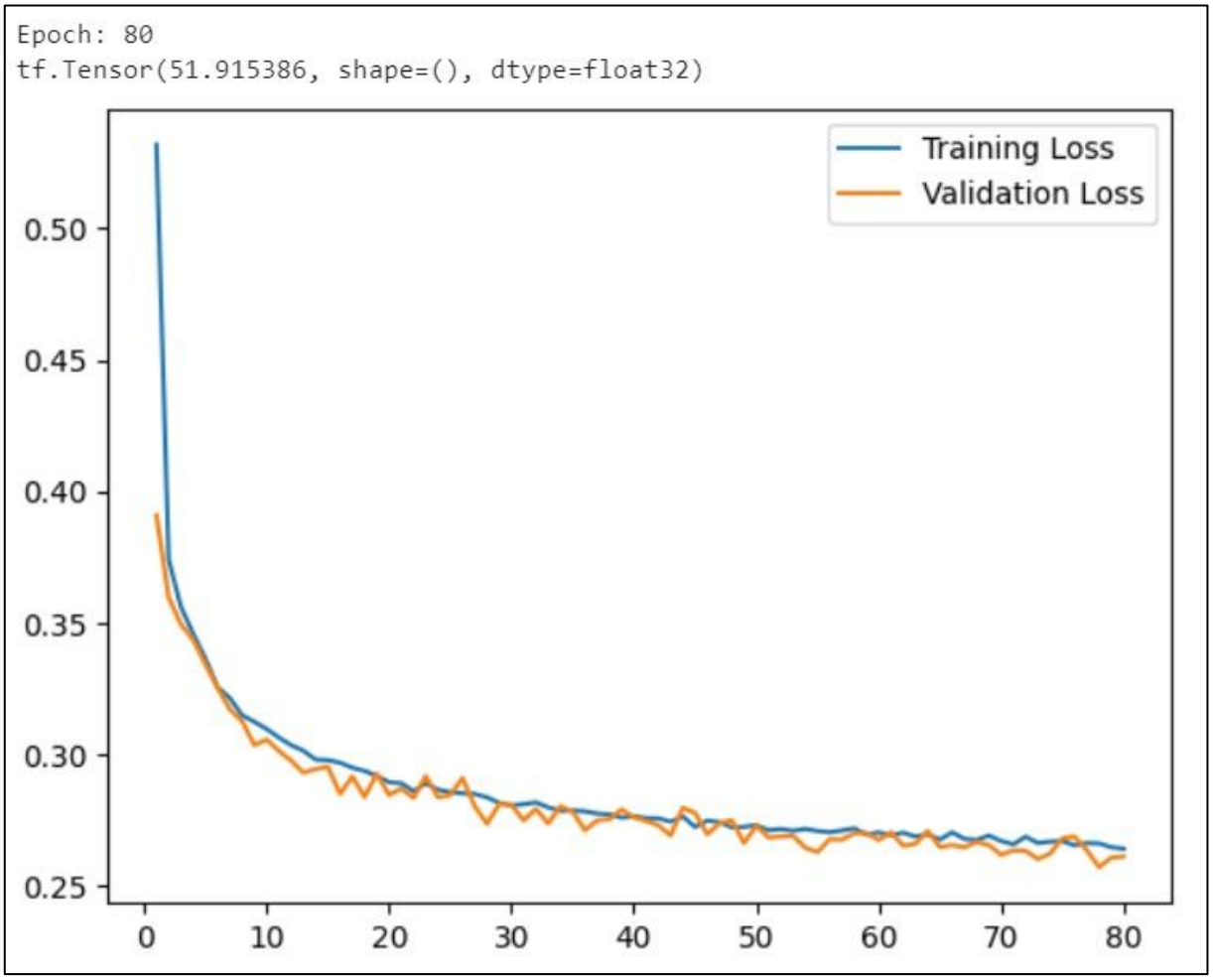

Рисунок 16 – График потерь на 80 эпохах

Из этого следует, что графики функции потерь при обучении и валидации примерно равны, но потери на валидационном наборе немного ниже,

можно сделать вывод, что модель успешно обучалась. Небольшая разница между ошибками на тренировочном и валидационном наборах говорит о том, что модель не переобучилась. Это означает, что модель достаточно хорошо обобщает данные и способна показывать хорошие результаты на новых примерах. Таким образом, можно считать, что обучение модели прошло успешно и она готова к использованию на практике. График функции потерь на 45 эпохах был представлен ранее. Архитектура моделируемой сети представлена в приложении.

3.3. Распознавание и удаление шума на видео

Распознавания и удаления шума на видео состоит из 2 функций.

Первая функция – `video_denoise`, предназначена для предобработки входного видеофайла и создания выходного видеофайла. Функция открывает видеофайл, считывает его кадры, применяет функцию удаления шума к каждому кадру, объединяет исходный кадр с очищенным кадром и записывает результат в новый видеофайл.

В начале функции определяется путь к папке с проектом и открывается видеофайл с помощью библиотеки OpenCV. Затем извлекаются размеры кадра и частота кадров.

Далее определяется кодек для записи видео в зависимости от расширения входного видеофайла. Если формат видео не поддерживается, генерируется исключение.

Создается встроенный в OpenCV объект `VideoWriter` для записи очищенного видео с удвоенной шириной, чтобы объединить исходный кадр с очищенным.

Затем функция начинает обработку кадров видео. Каждый кадр изменяется по размеру до ближайшего кратного 128, затем применяется функция `denoise_image` для удаления шума. Обработанный кадр затем объединяется с исходным кадром горизонтально, чтобы показать разницу между кадрами.

Итоговый кадр записывается в выходное видео, которое затем сохраняется. Пользователю также предоставляется возможность прервать процесс, нажав клавишу «q».

По завершении обработки видеофайла все ресурсы освобождаются, окна закрываются, и процесс завершается. На листинге 7 представлен функция предобработки видео для подачи в функцию удаления шума.

Листинг 7 – Функция video_denoise

```
def video_denoise(video_path):

    path = "C:/Users/alexe/PycharmProjects/DIPLOM/"
    # Открываем видео
    cap = cv2.VideoCapture(path + video_path)
    # Получаем размеры кадра
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)

    # Определяем кодек в зависимости от расширения файла
    if video_path.endswith('.mp4'):
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    elif video_path.endswith('.avi'):
        fourcc = cv2.VideoWriter_fourcc(*'MJPG')
    else:
        raise ValueError("Невозможно определить формат видео. Пожалуйста,
укажите формат явно.")
    # Создаем объект VideoWriter str(path + 'denoised_' + video_path)
    result = cv2.VideoWriter(str(path + 'denoised_' + video_path), fourcc,
fps, (width*2, height))
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        new_width = (width // 128 +1)*128
        new_height = (height // 128 +1)*128

        frame = tf.image.resize(frame, (new_width, new_height), meth-od='ar-
ea').numpy()
        a = denoise_image(frame)
        if a is not None:
            a = denoise_image(a * 255)
            a = cv2.resize(a, (width, height)).copy()
            frame = cv2.resize(frame, (width, height))
            b = np.concatenate((frame / 255, a), axis=1)

            b *= 255
            b = np.clip(b, 0, 255)
            ub = b.astype(np.uint8)
            cv2.imshow('frame', ub)
            result.write(ub)

        if cv2.waitKey(1) == ord('q'):
            break

    # Освобождение ресурсов
```

```
cap.release()
result.release()
cv2.destroyAllWindows()
```

Вторая функция – denoise_image, представляет собой алгоритм удаления шума на изображении, который разбивает входное изображение на фрагменты размером 128×128 пикселей, обрабатывает каждый фрагмент с использованием нейронной сети, и затем собирает обработанные фрагменты обратно в итоговое изображение. На рисунке 17 демонстрируется сравнение между кадром из видео с шумом и без него.

Рисунок 17 – Сравнение кадров на видео с шумом и без шума

Изначально функция получает высоту, ширину и число каналов входного изображения, а также вычисляет количество циклов по ширине и высоте, необходимых для разбиения изображения на фрагменты 128×128 пикселей. Затем создается пустой массив out_frame для хранения итогового изображения.

Далее функция проходит по всем фрагментам изображения с шагом 128×128 пикселей. Для каждого фрагмента она извлекает его из исходного изображения и подготавливает для обработки нейронной сетью. Если размер фрагмента 128×128, он преобразуется в тензор и нормализуется. Если размер меньше, фрагмент дополняется нулями до 128×128.

Затем нейронная сеть применяется к подготовленному фрагменту для получения тензора шума. Очищение фрагмента происходит путем вычитания шума, умноженного на коэффициент, который можно настраивать в зависимости от уровня шума в исходном фрагменте.

Основная идея заключается в том, что нейронная сеть способна выделить шум, присутствующий в изображении. Однако, если просто вычесть весь выделенный шум из исходного изображения, это может привести к чрезмерному подавлению деталей и артефактам.

Умножение тензора шума на коэффициент 0,1 позволяет сохранить большую часть исходного сигнала, но при этом использовать информацию о шуме, полученную от нейронной сети, для его частичного удаления. Таким образом, достигается баланс между сохранением деталей и удалением шума.

Если размер фрагмента не 128×128, происходит смешивание очищенного фрагмента с соседними фрагментами для уменьшения артефактов на границах.

Очищенный фрагмент сохраняется в итоговом изображении out_frame. Наконец, значения пикселей в out_frame нормализуются в диапазоне и возвращается итоговое изображение. Функция удаления шума на изображении представлена в листинге 8.

Листинг 8 – Функция удаления шума на изображении

```
def denoise_image(image):

    h, w, _ = image.shape

num_w_cycles = w // 128
    num_h_cycles = h // 128

    out_frame = np.zeros((h, w, 3), dtype=np.float32)

    for h_pos in range(num_h_cycles):
        for w_pos in range(num_w_cycles):
            start_h = h_pos * 128
            end_h = min((h_pos + 1) * 128, h)
            start_w = w_pos * 128
            end_w = min((w_pos + 1) * 128, w)
            fragment = image[start_h:end_h, start_w:end_w, :]

            if fragment.shape[0] == 128 and fragment.shape[1] == 128:
```

```
                tensor    =    tf.cast(np.expand_dims(fragment,    axis=0),
tf.float32) / 128 - 1

            else:

                tensor = tf.cast(np.expand_dims(np.pad(fragment, ((0, 128 -
fragment.shape[0]), (0, 128 - fragment.shape[1]), (0, 0)), mode='constant'),
axis=0), tf.float32) / 128 - 1

            noise_tensor = model.nn_unet(tensor)

            clear = tensor - noise_tensor * 0.1

            clear = clear.numpy()[0]

            if fragment.shape[0] == 128 and fragment.shape[1] == 128:
                out_frame[start_h:end_h, start_w:end_w, :] = clear

            else:
                overlap_h = min(16, fragment.shape[0])
                overlap_w = min(16, fragment.shape[1])

                blended = np.zeros_like(fragment)

                blended[:overlap_h,  :overlap_w,  :]  =  clear[:overlap_h,
:overlap_w, :]

                blended[overlap_h:end_h-start_h, overlap_w:end_w-start_w, :]
= clear[overlap_h:, overlap_w:, :]

                out_frame[start_h:end_h, start_w:end_w, :] = blended

    return np.clip((out_frame + 1) / 2, 0, 1)
```

Таким образом, функция `denoise_image` реализует алгоритм удаления шума на изображении с использованием нейронной сети и обработкой его фрагментов для улучшения качества изображения и удаления шума.

4. ТЕСТИРОВАНИЕ НЕЙРОННОЙ СЕТИ

Для тестирования единичного изображения была разработана функция, которая выводит зашумленное и обесшумленное изображение, а также метрики изображения. Работа функции представлена на рисунке 18.

Рисунок 18 – Удаление шума на изображении с метриками

Для комплексного тестирования и сравнения эффективности различных моделей по удалению шума из изображений, необходимо использовать набор данных (датасет), который содержит изображения с искусственно добавленным шумом различной интенсивности, например, гауссовским шумом.

Такой подход позволит оценить способность моделей справляться с шумом разного уровня и обеспечит более всестороннюю проверку их производительности. Использование датасета с контролируемым уровнем шума дает возможность объективно сравнивать результаты разных моделей в одинаковых условиях.

В качестве метрик используются PSNR (Peak Signal-to-Noise Ratio) и SSIM (Structural Similarity Index).

PSNR (Peak Signal-to-Noise Ratio) – это метод оценки качества изображения, который измеряет отношение между максимальной интенсивностью сигнала и уровнем шума в изображении. Более высокое значение PSNR указывает на более высокое качество изображения и меньшее искажение при сжатии или передаче данных.

SSIM (Structural Similarity Index) – это метод оценки сходства структуры между двумя изображениями. SSIM учитывает не только яркость и контраст изображений, но и их структурное сходство. Значение SSIM находится в диапазоне от -1 до 1, где 1 указывает на полное сходство между изображениями. SSIM считается более точным и сложным методом оценки качества изображений по сравнению с PSNR.

Для сравнения производительности были выбраны следующие модели: CBM3D, DnCNN, IrCNN, FFDNet, DHDN, RDUNet. Для более объективного сравнения эти модели были протестированы на датасете, зашумленном гауссовским шумом с различной интенсивностью, поскольку именно на таком шуме они были оценены. Для сравнения с моделями были выбраны наборы данных – Urban100 [19] и Kodak24 [20].

Итак, для получения метрик были реализованы две функции, которые описаны ниже.

1. Функция `calculate_metrics_for_dataset` – вычисляет PSNR и SSIM для набора данных с определенным уровнем шума.

2. Функция `apply_noise_2` – функция добавляет гауссовский шум различной интенсивности в набор данных и генерирует несколько наборов данных с разным уровнем шума.

Таким образом, первая функция рассчитывает метрики качества изображений, а вторая функция создает наборы данных с различным уровнем шума, которые затем используются для вычисления этих метрик.

Данные PSNR для вашей модели на тестовых данных Kodak24 и Urban100 следующие: Kodak24 – 10: 32,08; 30: 28,74; 50: 25,36;, Urban100 – 10: 32,80; 30: 27,56; 50: 25,89. В сравнении с CBM3D, ваша модель показала

близкие результаты на обоих наборах данных. По сравнению с DnCNN, ваша модель показала сопоставимые результаты на Urban100, но немного хуже на Kodak24. По сравнению с IrCNN, разработанная модель показала немного худшие результаты на обоих наборах данных. По сравнению с FFDNet, ваша модель показала сопоставимые результаты на Urban100, но немного хуже на Kodak24. По сравнению с DHDN, разработанная модель показала немного худшие результаты на обоих наборах данных. По сравнению с RDUNet, данная модель показала сопоставимые результаты на Urban100, но немного худшие на Kodak24. Данные представлены в таблице 2.

Таблица 2 – Сравнение моделей на метрике PSNR

	Датасет Kodak24			**Датасет Urban100**		
Модель / **Шум, %**	10	30	50	10	30	50
CBM3D	33,32	27,75	25,60	33,32	26,75	25,60
DnCNN	32,73	28,68	28,94	31,73	30,55	28,11
IrCNN	35,88	30,32	28,92	36,58	31,28	28,94
FFDNet	33,26	30,22	29,10	33,26	27,24	27,92
DHDN	34,96	31,98	29,72	33,73	31,39	29,10
RDUNet	31,86	30,72	27,72	37,86	29,87	27,61
Разработанная модель	32,08	28,74	25,36	32,80	27,56	25,89

Результаты оценки по метрике SSIM для нашей модели на наборах данных Kodak24 и Urban100 с различными уровнями шума показывают следующее. На наборе данных Kodak24, данная модель достигла отличных результатов по метрике SSIM при уровне шума 10, превосходя другие модели, такие как CBM3D, DnCNN и RDUNet. При уровне шума 30, SSIM составила 0,847, что является конкурентоспособным результатом, но немного уступает IrCNN и FFDNet. При уровне шума 50, SSIM у нашей модели составила 0,714, что является не самым большим показателем, среди представленных моделей, но данный результат не является отрицательным. На наборе данных Urban100, реализованная модель показала отличные результаты по SSIM при уровне шума 10, превзойдя все большинство моделей, включая

IrCNN и FFDNet. При уровне шума 30, SSIM составила 0,860, что также является хорошим результатом, но немного уступает IrCNN. Наконец, при уровне шума 50, SSIM для нашей модели составила 0,754, что немного уступает остальным моделям. Данные по метрике SSIM представлены в таблице 3.

Таблица 3 – Сравнение моделей на метрике SSIM

	Датасет Kodak24			**Датасет Urban100**		
Модель / **Шум, %**	10	30	50	10	30	50
CBM3D	0,913	0,773	0,686	0,951	0,843	0,763
DnCNN	0,939	0,845	0,774	0,948	0,849	0,771
IrCNN	0,945	0,858	0,792	0,951	0,861	0,788
FFDNet	0,945	0,858	0,794	0,953	0,861	0,789
DHDN	0,946	0,860	0,795	0,953	0,860	0,788
RDUNet	0,901	0,874	0,817	0,951	0,872	0,787
Разработанная модель	0,916	0,847	0,714	0,951	0,860	0,754

Таким образом, разработанная модель демонстрирует высокую конкурентоспособность на различных уровнях шума, что является результатом тщательной работы и эффективного выбора метрик для оценки результатов. Она показывает отличные результаты на обоих наборах данных, особенно при уровне шума 10, и даже может превзойти некоторые другие модели в некоторых случаях. Это свидетельствует о ее способности адаптироваться к различным условиям и обеспечивать высокую точность при обработке данных с разными уровнями шума.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана система автоматического распознавания и удаления шума на видео с помощью рекуррентных нейронных сетей. При этом были решены следующие задачи.

1. Проведен обзор литературы и существующих аналогов по предметной области.
2. Спроектирована архитектура нейронной сети для задачи удаления шума на видео, включающая в себя рекуррентные слои.
3. Реализован алгоритм удаления шума на видео с помощью обученной нейронной сети.
4. Проведено тестирование полученной нейронной сети.

В планах на будущее – увеличение размера входного изображения модели до 256×256 пикселей или даже больше, в зависимости от доступных ресурсов, с целью повышения точности модели. Это может помочь модели извлечь более сложные и абстрактные признаки из входных данных, что может улучшить ее способность к распознаванию шума на различных типах данных.

Также в планах внедрение модели в веб-сервис для обнаружения шума не только в видео, но и в аудио.

Полученные результаты, исходные коды, датасет расположены в репозитории GitHub [21].

ЛИТЕРАТУРА

1. Bengio Y., Patrice S., Frasconi P., Learning long–term dependencies with gradient descent is difficult. // IEEE Neural Networks Council 5, 1994.– 157–66 pp.

2. Jay A., Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention). // IEEE Neural Networks Council 5, 2018. – 239–256 pp.

3. Sepp H., Jürgen S. Long Short–term Memory. // Neural computation 9, 1997. – 1735–1780 pp.

4. Yang X., Wenhan F., Jiashi X., Guosen L., Jiaying Guo, Zongming Yan, Frasconi S. Video super–resolution based on spatial–temporal recurrent residual networks. // Computer Vision and Image Understanding., 2017. – 695 pp.

5. Enitan F., Ilesanmi E., Ilesanmi Q., Taiwo G. Methods for image denoising using convolutional neural network: a review. // Complex Intelligent Systems. 7., 2021. – 728–735 pp.

6. Chen, Xinyuan S., Li Y., Xiaokang. Deep RNNs for video denoising. Classification and Regression Trees. // Routledge., 2017. – 368–407 pp.

7. Faraji, Marjan N., Saeed G., Omid H., Saeed, Downey, Kay. An integrated 3D CNN–GRU deep learning method for short–term prediction of PM2.5 concentration in urban environment. // Science of The Total Environment., 2022. – 892–893 pp.

8. Ronneberger O., Brox T., Fischer P. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015. – 19–26 pp.

9. Han X., Kashif R., Roland V., Fashion–MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms., 2017. – 1–6 pp.

10. Welcome to Python. [Электронный ресурс]. URL: https://www.python.org/ (дата обращения: 12.05.2024 г.).

11. Vincent S., Pascal R.; Larochelle F., Hugo E. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local

Denoising Criterion. // Journal of Machine Learning Research. 11–12, 2010. – 3371–3408 pp.

12. Zhang K., Zuo W., and Zhang L., Ffdnet: Toward a fast and flexible solution for cnn–based image denoising. // IEEE Transactions on Image Pro–cessing, vol. 27, no. 9, 2018. – 4608–4622 pp.

13. OpenCV – OpenCV. [Электронный ресурс]. URL: https://opencv.org/ (дата обращения: 09.05.2024 г.).

14. Keras: Deep Learning for humans. [Электронный ресурс] URL: https://keras.io/api/ (дата обращения: 12.05.2024 г.).

15. Dabov K., Foi A., Katkovnik V., and Egiazarian K. Image denoising with block–matching and 3d filtering, in Image Processing: Algorithms and Systems, Neural Networks, and Machine Learning., // International Society for Optics and Photonics vol. 6064., 2006, – 606414 p.

16. Flickr30k. [Электронный ресурс] URL: https://paperswithcode.com/dataset/flickr30k (дата обращения: 12.05.2024 г.).

17. Tensorflow. [Электронный ресурс] URL: https://www.tensorflow.org/ (дата обращения: 12.05.2024 г.).

18. Gu S., Zhang L., Zuo W., and Feng X., Weighted nuclear norm minimization with application to image denoising, in Proceedings of the IEEE conference on computer vision and pattern recognition., 2014. – 2862–2869 pp.

19. Urban100. [Электронный ресурс] URL: https://www.kaggle.com/datasets/harshraone/urban100 (дата обращения: 12.05.2024 г.).

20. Kodak24. [Электронный ресурс] URL: https://www.kaggle.com/datasets/drxinchengzhu/kodak24 (дата обращения: 12.05.2024 г.).

21. Video-denosing. [Электронный ресурс] URL: https://github.com/Gena709/Video-denosing (дата обращения: 12.05.2024 г.).

ПРИЛОЖЕНИЕ. Архитектура модели

На рисунке 1 представлена архитектура разработанной модели

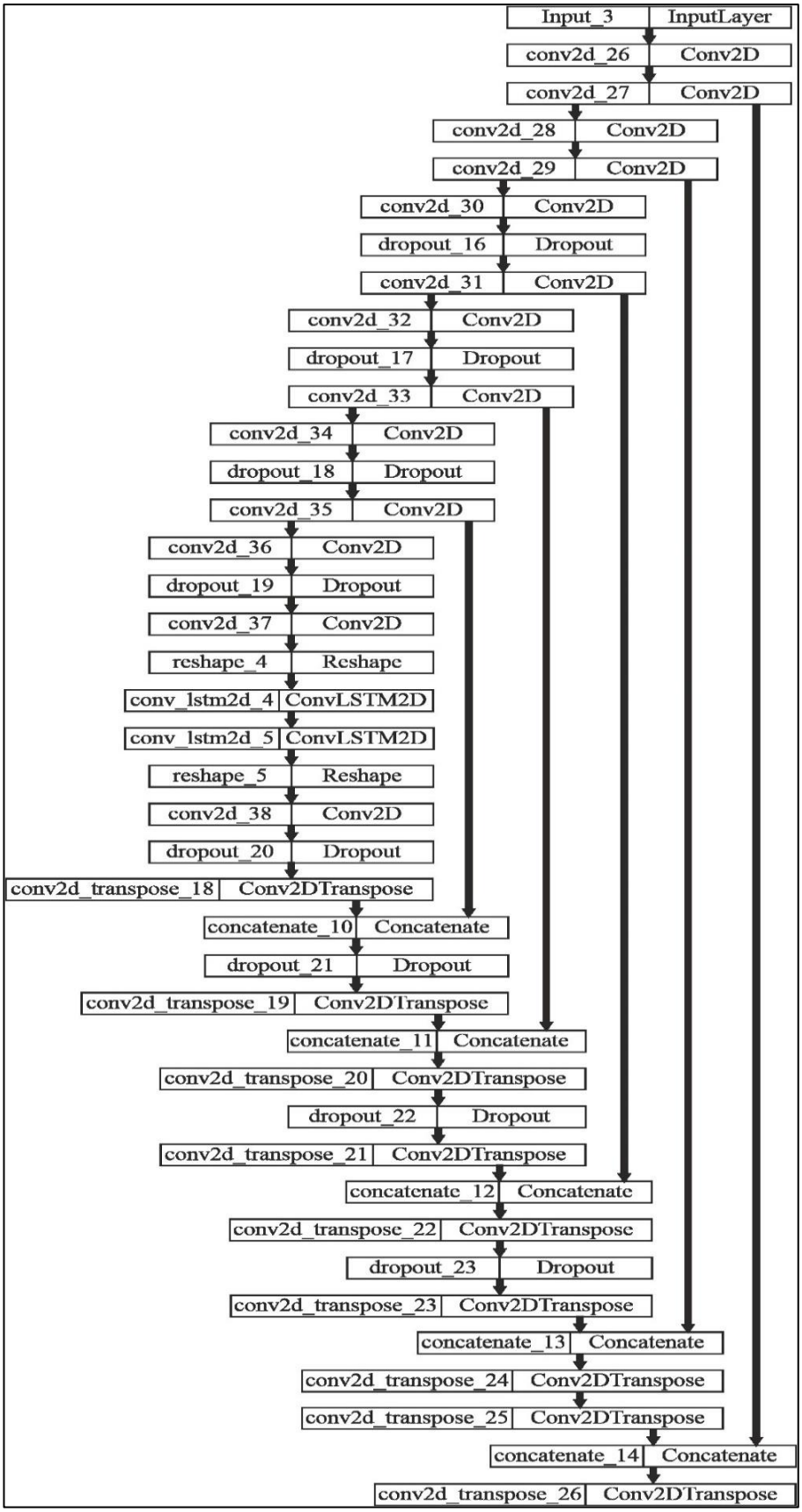

Рисунок 1 – Архитектура разработанной модели