ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

Институт управления и цифровых технологий

ДОПУСТИТЬ К ЗАЩИТЕ:

Организатор обучения

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Е. Нутович

"\_\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**по программе профессиональной переподготовки «ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И МАШИННОЕ ОБУЧЕНИЕ»**

**Сервис для обработки фото с применением технологий искусственного интеллекта**

Слушатели программы

Глушков А.А.

подпись ФИО

Талалова Н.С.

подпись ФИО

Костылев Е.А.

подпись ФИО

Швец М.С.

подпись ФИО

Руководитель работы

Заманов Е.А.

подпись ФИО

Москва 2024 г.

# **Оглавление**

[1 Оглавление 2](#_Toc168226248)

[2 Введение 4](#_Toc168226249)

[3 Постановка задачи 8](#_Toc168226250)

[4 Теоретическая часть 17](#_Toc168226251)

[4.1 Анализ существующих методов и технологий обработки изображений. 17](#_Toc168226252)

[4.2 Разработка алгоритмов обработки изображений. 18](#_Toc168226253)

[4.3 Создание датасета для обучения моделей машинного обучения. 23](#_Toc168226254)

[4.4 Разработка пользовательского интерфейса. 27](#_Toc168226255)

[4.5 Тестирование и оптимизация сервиса. 28](#_Toc168226256)

[4.6 Итоговый стек технологий и методологий. 29](#_Toc168226257)

[4.7 Итоговые методологии обучений нейронных сетей. 31](#_Toc168226258)

[5 Обучение модели 33](#_Toc168226259)

[5.1 Сбор и подготовка данных 33](#_Toc168226260)

[5.2 Обучение модели ИИ 37](#_Toc168226261)

[5.2.1 Задача автоулучшения качества. 39](#_Toc168226262)

[5.2.2 Модель раскрашивания фотографий 42](#_Toc168226263)

[5.2.3 Модель устранения бликов 45](#_Toc168226264)

[6 Имплементация модели 48](#_Toc168226265)

[7 Заключение 55](#_Toc168226266)

[8 Приложения 58](#_Toc168226267)

[8.1 Приложение А – программный код серверной части 58](#_Toc168226268)

[8.1.1 Серверная часть 58](#_Toc168226269)

[8.1.2 Модель раскрашивания изображений 59](#_Toc168226270)

[8.1.3 Модель улучшения качества 61](#_Toc168226271)

[8.1.4 Модель удаления бликов 62](#_Toc168226272)

[8.2 Приложение Б – программный код клиентской части 64](#_Toc168226273)

[8.2.1 Файл src/components/PageWrapper.vue 64](#_Toc168226274)

[8.2.2 Файл src/pages/ErrorNotFound.vue 64](#_Toc168226275)

[8.2.3 Файл src/pages/IndexPage.vue 65](#_Toc168226276)

[8.2.4 Файл src/pages/resultPage.vue 68](#_Toc168226277)

[8.2.5 Файл src/router/index.ts 70](#_Toc168226278)

[8.2.6 Файл src/router/routes.ts 70](#_Toc168226279)

[8.2.7 Файл src/store/index.ts 71](#_Toc168226280)

[8.2.8 Файл src/store/mutations.ts 71](#_Toc168226281)

[8.2.9 Файл src/store/state.ts 71](#_Toc168226282)

[8.2.10 Файл src/App.vue 72](#_Toc168226283)

[8.3 Приложение В – иконка сайта 72](#_Toc168226284)

[9 Список использованной литературы 73](#_Toc168226285)

# Введение

В современном мире визуальный контент приобретает все большее значение, становясь неотъемлемой частью различных сфер человеческой деятельности. От качества изображений зависит восприятие информации, продуктов и услуг потребителями. В эпоху цифрового прогресса обработка изображений выходит на новый уровень благодаря развитию технологий искусственного интеллекта (ИИ) и машинного обучения, которые открывают возможности для автоматизации и усовершенствования процессов, ранее требовавших значительных временных и человеческих ресурсов. ИИ и машинное обучение позволяют значительно ускорить и улучшить процессы обработки изображений, делая их более доступными и качественными для широкого круга пользователей.

Вопросы происхождения, развития и интеграции ИИ в различные сферы жизни активно изучались научным сообществом на протяжении второй половины XX века. В XXI веке исследования сосредоточились на практическом применении ИИ-технологий. Согласно исследованию аналитического центра TAdviser и провайдера «Ростелеком» «Проникновение решений на базе искусственного интеллекта в российских компаниях», на конец 2020 года свыше 85% крупных отечественных компаний, таких как ПАО «Сбербанк», ПАО «Банк ВТБ», X5 Retail Group, «Магнит», ПАО «Газпром нефть», уже внедрили или тестируют решения на базе ИИ. Однако некоторые ученые, в частности израильский мыслитель Юваль Ной Харари, видят в распространении ИИ-технологий угрозу для человеческой занятости и призывают к глобальному сотрудничеству и переходу от соперничества к симбиозу человека и ИИ. Харари подчеркивает, что необходимо выработать новые стратегии взаимодействия между людьми и ИИ, чтобы минимизировать риски и максимизировать выгоды.

Исследование Института статистических исследований и экономики знаний НИУ ВШЭ о мониторинге развития и распространения ИИ в России показывает, что 65% организаций применяют ИИ в тестовом режиме, изучая его потенциал для бизнеса. Большинство (75%) используют ИИ в связке с другими цифровыми технологиями, такими как промышленное ПО и системы автоматизированного проектирования. Среди наиболее востребованных технологий ИИ – компьютерное зрение и распознавание речи, которые применяются в 78,7% и 62% случаев соответственно. В данном контексте особенно актуальны разработки в области обработки фото с использованием ИИ.

Изучением природы искусственного интеллекта занимались многие выдающиеся умы, чьи работы заложили основу для современных исследований в этой области. Одним из таких пионеров был логик Джордж Буль, разработавший алгебру логики, которая стала основой для цифровых вычислений. Его идеи были революционны и до сих пор используются в компьютерных науках и ИИ. Философ Готфрид Лейбниц также внес значительный вклад, предложив двоичную систему исчисления, которая легла в основу цифровой электроники и компьютерной логики. Лейбниц был одним из первых, кто предложил концепцию универсального языка для выражения всех научных знаний, что предвосхитило современные идеи о машинном обучении и искусственном интеллекте.

Математик и изобретатель вычислительной машины Чарльз Бэббидж создал проект первой аналитической машины, которая была предшественницей современных компьютеров. Его работа дала толчок развитию вычислительных устройств, без которых невозможно представить сегодняшние достижения в области ИИ. Бэббидж не только разработал теоретические основы, но и построил прототипы, которые стали первыми шагами к созданию автоматических вычислительных систем.

Алан Тьюринг, еще один выдающийся математик, считается одним из основоположников искусственного интеллекта. Его работы по теории вычислимости, в частности, концепция машины Тьюринга, оказали огромное влияние на развитие компьютерных наук. Тьюринг предложил идею, что машина могла бы имитировать любую человеческую вычислительную задачу, если она могла быть четко определена. Это стало основой для создания универсальных компьютеров и дальнейшего развития ИИ. Тест Тьюринга, предложенный для оценки способности машины демонстрировать интеллект, остается актуальным и сегодня.

Эти ученые заложили фундамент для развития искусственного интеллекта, создав основные концепции и методы, которые используются до сих пор. Их идеи и исследования стали краеугольными камнями для всех последующих разработок в этой области. Однако, несмотря на богатую научную базу, вопросы практического применения ИИ в различных областях человеческой деятельности остаются недостаточно изученными.

Применение искусственного интеллекта в сфере фотографии вызывает оживленные дискуссии в научном и профессиональном сообществе. Как отмечает автор исследования Лаврова А.Г., использование ИИ-технологий открывает новые возможности для повышения эффективности и расширения творческих границ в фотографии.

С одной стороны, внедрение ИИ способствует автоматизации рутинных процессов обработки изображений, позволяя фотографам сосредоточиться на художественной составляющей своей работы. ИИ-алгоритмы могут корректировать экспозицию, контраст, насыщенность, улучшать качество за счет увеличения разрешения и шумоподавления, а также применять стилизацию и творческие эффекты, расширяя диапазон выразительных средств. Кроме того, ИИ вносит вклад в сохранение и оцифровку исторических фотографий, восстанавливая и улучшая старые снимки. Это особенно важно для сохранения культурного наследия и передачи его будущим поколениям.

С другой стороны, опасения вызывают этические аспекты использования ИИ, вопросы авторства и оригинальности произведений, созданных при его участии. Критики указывают на риски стандартизации и потери индивидуальности в творческом процессе, а также угрозу подрыва традиционных навыков фотографии из-за чрезмерной зависимости от технологий. Важно учитывать, что ИИ, хотя и является мощным инструментом, не должен заменять творческое мышление и индивидуальный подход, присущие человеку. Вместо этого ИИ должен стать помощником, который расширяет возможности фотографов и помогает им реализовывать свои идеи.

Таким образом, применение искусственного интеллекта в мире фотографии открывает новые горизонты, но также ставит вопросы, требующие глубокого осмысления и выработки сбалансированного подхода к интеграции ИИ-инструментов в творческие практики. Решение этих вопросов будет определять дальнейшее развитие фотографического искусства в эпоху цифровых технологий. Интеграция ИИ должна осуществляться таким образом, чтобы сохранять баланс между технологическими возможностями и человеческим творчеством, обеспечивая гармоничное развитие обоих аспектов.

# Постановка задачи

Предметная область, на которую направлена данная работа, охватывает широкий спектр отраслей, где качество визуального контента играет решающую роль. Это может быть сфера фотографии, архивного дела, издательского дела, рекламы, электронной коммерции, а также любая другая область, где важно представить продукт, услугу или идею в наиболее привлекательном виде. В современном мире, где визуальная информация стала неотъемлемой частью нашей повседневной жизни и бизнеса, важность качественных изображений трудно переоценить. Одной из ключевых задач в обработке изображений является устранение дефектов, возникающих в процессе съемки или при последующей обработке.

К числу распространенных дефектов съемки относятся блики и засветы, которые возникают из-за непростого освещения или отражения света от различных поверхностей. Они вызываются прямым попаданием источника света в объектив камеры и приводят к локальному пересвету и потере деталей на соответствующих участках изображения. Эти дефекты могут значительно снизить качество визуального контента, делая изображение менее привлекательным и профессиональным. Блики и засветы могут отвлекать внимание зрителя от основного объекта на фотографии, создавая нежелательные световые пятна и искажения. В результате общее впечатление от изображения ухудшается, что может негативно сказаться на восприятии продукта или услуги потенциальными клиентами.

Эти проблемы особенно актуальны для сфер, где визуальный контент играет ключевую роль. В фотографии и издательском деле качество изображений напрямую влияет на то, как зритель воспринимает материал. В рекламной индустрии каждое изображение должно быть безупречным, чтобы эффективно привлекать внимание и вызывать положительные эмоции. В электронной коммерции высококачественные фотографии продуктов могут существенно повысить их привлекательность и, соответственно, продажи. Следовательно, устранение дефектов съемки является важной задачей для обеспечения высокого уровня визуального контента.

В процессе обработки изображений важным аспектом является не только устранение дефектов, но и общее улучшение качества фотографий. Это включает в себя корректировку резкости и детализации изображения. Такие улучшения позволяют сделать изображение более ярким и выразительным, подчеркивая его сильные стороны и привлекая внимание зрителя. Кроме того, качественная обработка фотографий помогает создать единый стиль и гармонию, что особенно важно в рекламных и маркетинговых материалах.

Также одной из важных задач в обработке изображений является раскрашивание черно-белых фотографий. Этот процесс имеет большое значение в различных областях, начиная от архивирования исторических снимков и заканчивая созданием привлекательного визуального контента для современных медиа.

В сфере архивного дела и истории искусств раскрашивание старых фотографий играет ключевую роль в сохранении культурного наследия. Исторические фотографии, которые ранее существовали только в черно-белом формате, могут быть преобразованы в цветные изображения, что помогает лучше понять и визуализировать прошлое. Цветные фотографии делают исторические моменты более реалистичными и близкими, способствуя более глубокому погружению в контекст событий и эпохи.

В издательском деле и медиа раскрашивание фотографий позволяет создавать более привлекательный и разнообразный контент. Цветные изображения привлекают больше внимания и вызывают больший эмоциональный отклик у аудитории. Они могут быть использованы для иллюстрации статей, книг, документальных материалов, где важно подчеркнуть визуальную составляющую и сделать материал более интересным и доступным для широкого круга читателей.

В рекламной индустрии и маркетинге раскрашивание фотографий открывает новые возможности для креативного подхода к визуальному контенту. Цвет позволяет выделить основные элементы изображения, акцентировать внимание на важных деталях и создать более запоминающиеся образы. Это особенно важно для привлечения внимания к продуктам и услугам, а также для создания сильного визуального бренда. Цветные фотографии могут значительно повысить привлекательность рекламных материалов, делая их более эффективными в достижении маркетинговых целей.

Раскрашивание черно-белых фотографий также играет важную роль в семейных архивах и личных коллекциях. Оживление старых семейных фотографий помогает сохранить и передать семейные истории и воспоминания новым поколениям. Цветные изображения делают эти воспоминания более живыми и эмоциональными, способствуя укреплению семейных связей и сохранению культурного наследия.

Традиционные методы обработки изображений, основанные на ручной ретуши, требуют значительных временных и трудовых затрат. Это делает их дорогостоящими и неэффективными при работе с большими объемами данных. Кроме того, человеческий фактор может привести к неоднородности результатов и субъективности в оценке качества. В этом контексте разработка сервиса искусственного интеллекта для автоматизированной обработки фотографий открывает новые возможности для повышения эффективности и качества работы с визуальным контентом.

Используя передовые технологии машинного обучения, такой сервис способен раскрашивать изображения, анализировать изображения, выявлять дефекты и автоматически устранять их, обеспечивая высокое качество результата и значительную экономию времени и ресурсов. Это может оказать положительное влияние на широкий круг заинтересованных сторон. Профессиональные фотографы и дизайнеры смогут существенно сократить время, затрачиваемое на ретуширование, что позволит им уделить больше внимания творческим аспектам своей работы. Издательства и рекламные агентства смогут повысить качество визуального контента, используемого в их продуктах и кампаниях. Компании, занимающиеся электронной коммерцией, получат возможность представлять свои товары в наиболее привлекательном виде, что может положительно сказаться на продажах и удовлетворенности клиентов.

Таким образом, разработка сервиса ИИ для обработки фото является актуальной задачей, ввиду растущей потребности в качественном визуальном контенте и необходимости автоматизации процессов его обработки. Несмотря на наличие теоретической базы и первых практических шагов в этом направлении, данная область требует дальнейшего изучения и внедрения инновационных решений для полноценной интеграции ИИ-технологий в процессы работы с изображениями.

Цель данной работы заключается в разработке сервиса для автоматической обработки фотографий с использованием методов машинного обучения. Этот сервис должен обладать возможностью раскрашивать черно-белые фотографии, повышать резкость фотографии и устранять блики. Важным аспектом является удобство использования сервиса, чтобы он был доступен как профессионалам, так и любителям, при этом качество обработки изображений должно быть приближенно к ручной ретуши, выполненной профессионалами.

Основная задача заключается в создании системы, которая не только автоматизирует эти процессы, но и делает их доступными для широкого круга пользователей. Важным критерием успеха является интуитивно понятный интерфейс, который позволит легко загружать фотографии, применять необходимые улучшения и получать качественный результат без необходимости глубоких технических знаний.

Для достижения этой цели необходимо выполнить несколько задач:

1. Анализ существующих методов и технологий обработки изображений.

* Необходимо изучить массив научных статей, технических отчетов и другой литературы, связанной с методами автоматической обработки изображений. Это включает в себя исследование передовых разработок и последних достижений в области машинного обучения и компьютерного зрения.
* Следует провести анализ текущих коммерческих и открытых программных решений, которые предлагают функции, аналогичные разрабатываемому сервису. Это включает изучение различных инструментов для обработки изображений, используемых в профессиональной фотографии, издательском деле и рекламной индустрии. Необходимо понять, какие технологии и алгоритмы применяются в этих продуктах, как они реализованы и какие результаты демонстрируют на практике. Важно рассмотреть функциональные возможности этих инструментов, их производительность, удобство использования и доступность для различных категорий пользователей.
* Определить сильные и слабые стороны существующих решений. Для проведения сравнительного анализа различных методов и технологий нужно выделить ключевые критерии оценки, такие как точность, скорость обработки, качество результатов и простота использования. Сравнивая различные подходы по этим параметрам, можно выявить их преимущества и недостатки. Например, некоторые методы могут обеспечивать высокое качество изображения, но быть слишком медленными для практического использования. Другие могут быть быстрыми, но не давать достаточной точности или детализации. На основе полученных данных можно определить, какие аспекты могут быть улучшены или оптимизированы в разрабатываемом сервисе.

1. Разработка алгоритмов обработки изображений.

* Для создания алгоритмов на основе нейронных сетей для автоматического раскрашивания черно-белых фотографий следует:
  + Исследовать и выбрать подходящие архитектуры нейронных сетей. В рамках данного этапа необходимо провести детальное исследование и выбор оптимальных архитектур нейронных сетей для задачи автоматического раскрашивания черно-белых фотографий. Основное внимание следует уделить таким архитектурам, как Generative Adversarial Networks (GAN) и U-Net. GAN, состоящая из двух состязающихся сетей – генератора и дискриминатора, обладает высокой способностью к созданию реалистичных изображений. Генератор учится создавать изображения, которые дискриминатор не может отличить от реальных, что делает GAN идеальной для задач раскрашивания. U-Net, в свою очередь, известна своей эффективностью в задачах сегментации изображений благодаря использованию skip-connections, что позволяет сохранять контекстную информацию на различных уровнях абстракции.
  + После выбора архитектуры необходимо разработать и реализовать алгоритмы, способные эффективно распознавать объекты на черно-белых изображениях и предсказывать их цвет. Эти алгоритмы должны учитывать различные текстуры, формы и контекстные признаки, чтобы предсказания цвета были максимально точными. Разработка включает создание модели, обучение на большом наборе данных, состоящем из черно-белых и соответствующих цветных изображений, и последующее тестирование для оценки точности и качества раскрашивания. Особое внимание следует уделить тонким деталям и естественности цветов, чтобы результаты были неотличимы от реальных цветных фотографий.
* Разработать методы повышения четкости изображений
  + Исследовать методы восстановления и улучшения деталей изображений. Важным шагом является исследование методов, которые позволяют эффективно восстанавливать и улучшать четкость изображений. Эти методы, такие как архитектура U-Net и глубокие сверточные нейронные сети (CNN), используются для обработки изображений, сохраняя контекстную информацию на различных уровнях абстракции. Исследование должно охватить различные алгоритмы, включая SRCNN (Super-Resolution Convolutional Neural Network), EDSR (Enhanced Deep Super-Resolution), SRGAN (Super-Resolution Generative Adversarial Network), и их эффективность в различных сценариях.
  + На основе проведенного исследования необходимо разработать и внедрить алгоритмы, которые будут увеличивать разрешение изображений и улучшать их детализацию. Эти алгоритмы должны быть интегрированы в общий сервис и обеспечивать высокое качество обработки изображений. Реализация включает обучение моделей на обширных наборах данных и последующее тестирование, чтобы убедиться в том, что алгоритмы могут эффективно восстанавливать детали и улучшать четкость без появления артефактов.
* Реализовать алгоритмы для устранения бликов на фотографиях.
  + Разработать методы для обнаружения и устранения бликов, используя техники обработки изображений и машинного обучения. Алгоритмы должны быть способными распознавать области, где присутствуют блики, и корректировать их, заменяя блики на соответствующие текстуры и цвета, которые будут естественно интегрированы в изображение.
  + Обеспечить, чтобы алгоритмы корректно работали с фотографиями, снятыми при различных условиях освещения.

1. Создание и обучение моделей машинного обучения.

* Необходимо сформировать датасет, включающий черно-белые и цветные фотографии, а также изображения с разной четкостью и бликами. Первый шаг в создании высокоэффективных моделей машинного обучения заключается в сборе и подготовке обширного и разнообразного набора данных. Этот набор данных должен быть достаточно большим и репрезентативным, чтобы модели могли обучаться на множестве различных примеров и ситуаций. Изображения должны охватывать широкий спектр сцен и объектов, чтобы модели могли эффективно работать с любыми типами фотографий.
* Обучить модели на этом датасете, обеспечив высокое качество обработки изображений.
  + Для обучения модели необходимо использовать методы обучения с учителем, которые предполагают обучение моделей на размеченных данных, где истинные значения известны. Это позволяет моделям учиться на примерах и корректировать свои прогнозы на основе ошибок.
  + После начального этапа обучения необходимо провести тщательную оценку моделей на тестовых наборах данных. Эти наборы данных должны отличаться от тех, которые использовались для обучения, чтобы объективно оценить производительность моделей. На основе результатов оценки следует провести доработку моделей, корректируя их архитектуры, параметры и методы обучения. Это может включать добавление новых слоев, изменение гиперпараметров или применение более продвинутых техник оптимизации. Целью доработки является достижение высокого уровня точности и надежности моделей, чтобы они могли эффективно обрабатывать изображения в реальных условиях.

1. Разработка пользовательского интерфейса.

* Спроектировать и реализовать удобный интерфейс сервиса, позволяющий загружать фотографии, применять обработку и получать результаты. Создание удобного и интуитивно понятного интерфейса является критически важным аспектом для успешного функционирования сервиса. Интерфейс должен быть разработан таким образом, чтобы пользователи могли легко ориентироваться в нем и выполнять необходимые действия без каких-либо затруднений. Важно учитывать принципы пользовательского опыта и пользовательского интерфейса при проектировании, чтобы интерфейс был не только функциональным, но и эстетически приятным. Должны быть предусмотрены ясные и логичные шаги для загрузки фотографий, выбора параметров обработки и получения результатов. Визуальные элементы интерфейса, такие как кнопки, меню и панели инструментов, должны быть размещены удобно и понятно для пользователей всех уровней навыков.

Выполнение всех этих задач позволит достичь поставленной цели и создать эффективный сервис для автоматической обработки фотографий. Этот сервис будет полезен как профессиональным фотографам и дизайнерам, так и любителям, стремящимся улучшить качество своих снимков.

# Теоретическая часть

## Анализ существующих методов и технологий обработки изображений.

На рынке присутствует ряд инструментов и сервисов, предлагающих функции обработки изображений. Одним из наиболее известных является Adobe Photoshop, который предоставляет обширный набор инструментов для улучшения качества изображений, включая инструменты для раскрашивания, повышения резкости, удаления бликов и многие другие. Однако Photoshop требует значительных навыков и времени для освоения, что может быть сложно для пользователей, не имеющих соответствующей подготовки.

Также существуют специализированные сервисы, такие как DeepAI Image Colorization API, которые предлагают API для автоматического раскрашивания черно-белых изображений с использованием технологий искусственного интеллекта. Хотя такие решения легко интегрируются, они часто ограничены в функциональности и не предоставляют возможностей для дальнейшей настройки или расширения.

Еще один пример - сервис Let's Enhance, использующий искусственный интеллект для увеличения разрешения и улучшения качества изображений. Этот сервис демонстрирует хорошие результаты в повышении четкости и детализации изображений, но имеет высокую стоимость для коммерческого использования и ограниченную функциональность в бесплатной версии.

При проведении анализа существующих решений были выявлены следующие преимущества и недостатки:

* Adobe Photoshop обеспечивает высокую точность и гибкость в обработке изображений, но является сложным в использовании и требует значительных временных затрат на обучение.
* DeepAI Image Colorization API легко интегрируется, но имеет ограниченные возможности настройки и расширения функциональности.
* Let's Enhance демонстрирует хорошее качество улучшения изображений, но характеризуется высокой стоимостью для коммерческого использования и ограниченностью функций в бесплатной версии.

Таким образом, существующие решения имеют определенные ограничения в плане функциональности, гибкости, стоимости или удобства использования, что создает потребность в более универсальном и масштабируемом решении для обработки изображений с использованием современных технологий машинного обучения.

## Разработка алгоритмов обработки изображений.

Для задачи раскрашивания черно-белых фотографий будут исследованы архитектуры нейронных сетей GAN (Generative Adversarial Networks), U-Net и сверточная сеть (ConvNet). Эти архитектуры известны своей высокой эффективностью в задачах, связанных с генерацией и трансформацией изображений.

GAN состоит из двух основных компонентов: генератора и дискриминатора. Генератор создает цветные версии черно-белых изображений, пытаясь сделать их максимально реалистичными, в то время как дискриминатор пытается отличить сгенерированные изображения от реальных цветных фотографий. Этот состязательный процесс позволяет генератору постепенно улучшать качество своих предсказаний, что делает GAN идеальным выбором для задачи раскрашивания. Использование GAN для раскрашивания позволяет достигать высокого уровня реалистичности благодаря способности модели обучаться на различиях между реальными и сгенерированными изображениями, что в свою очередь стимулирует генератор создавать все более правдоподобные цветовые решения.

U-Net представляет собой архитектуру энкодер-декодерного типа, которая также хорошо подходит для раскрашивания фотографий. В этой архитектуре входное черно-белое изображение проходит через серию сверточных слоев, которые последовательно уменьшают его размер, извлекая при этом важные признаки. Затем эти признаки используются для восстановления цветного изображения на декодерном этапе. Ключевая особенность U-Net — использование skip-connections, которые позволяют соединять соответствующие уровни энкодера и декодера. Это помогает сохранять детали и контекстную информацию, что критически важно для точного раскрашивания изображений. С помощью U-Net можно добиться высокой точности и сохранения мелких деталей, что делает её подходящей для задач, где важно не только раскрашивание, но и сохранение текстур и особенностей исходного изображения.

Сверточная сеть (ConvNet) также будет рассмотрена для задачи раскрашивания черно-белых фотографий. ConvNet, благодаря своей архитектуре, включает несколько сверточных слоев, функций активации, нормализации и подвыборки. Она способна эффективно извлекать признаки из входных черно-белых изображений и предсказывать соответствующие цветовые компоненты. Применение ConvNet обеспечивает надежное и точное раскрашивание изображений, позволяя моделям восстанавливать цвета с высокой степенью реализма и детализации.

Помимо GAN, U-Net и ConvNet, существуют и другие методы колоризации, такие как методы на основе ручной разметки (Scribble-based Colorization), классические алгоритмы компьютерного зрения, такие как цветовой перенос (Color Transfer), и модели глубокого обучения с использованием автокодировщиков (Autoencoders). Например, метод колоризации на основе ручной разметки требует от пользователя минимальных аннотаций, что позволяет модели получать ориентиры для раскрашивания. Однако это требует значительных усилий от пользователя и может не подходить для массового использования. Классические методы цветового переноса, хотя и быстры, часто не способны захватывать сложные текстуры и детали, ограничивая их применимость. Автокодировщики, хотя и полезны, не достигают такой же высокой точности и реалистичности, как GAN, U-Net и ConvNet.

Для задачи повышения четкости изображений будет исследована архитектура U-Net. Эта архитектура зарекомендовала себя как высокоэффективная в задачах, требующих точного восстановления изображений, таких как сегментация, перенос стиля и цветокоррекция. U-Net позволит модели не только восстанавливать мелкие детали, но и улучшать общую структуру, резкость и цветопередачу изображений.

U-Net состоит из двух основных путей: сжимающего (энкодера) и расширяющего (декодера). Сжимающий путь включает несколько сверточных и подвыборочных слоев, которые постепенно уменьшают размер изображения и извлекают его признаки на разных масштабах и уровнях абстракции. Расширяющий путь включает операции апсэмплинга, которые восстанавливают исходное разрешение изображения на основе извлеченных признаков.

Ключевая особенность U-Net - использование skip-connections между соответствующими слоями сжимающего и расширяющего путей. Эти соединения позволяют модели объединять локальные признаки, такие как края, текстуры и мелкие детали из ранних слоев, с глобальной контекстной информацией о цвете, освещении и семантических признаках из более глубоких слоев. Благодаря этому модель лучше понимает структуру, содержание и цветовые характеристики изображения, что крайне важно для корректной обработки, восстановления деталей и выполнения задач вроде повышения четкости и цветокоррекции.

В задаче цветокоррекции U-Net может использоваться для восстановления и улучшения цветовых характеристик изображения. На сжимающем пути модель извлекает признаки, относящиеся к цвету, оттенкам, насыщенности, освещению и теням на разных уровнях абстракции. Затем, на расширяющем пути, эти цветовые признаки объединяются с локальной информацией о деталях и текстурах через skip-connections. Это позволяет модели восстанавливать исходные цвета, корректировать их с учетом глобальных и локальных контекстных сведений, улучшать цветопередачу и контрастность, делая изображения более реалистичными и привлекательными.

Применение U-Net в задачах повышения четкости и цветокоррекции позволит точно восстанавливать мелкие детали, текстуры, а также улучшать цветовые характеристики, контрастность и общее качество изображений. Благодаря объединению локальных и глобальных признаков модель способна учитывать как низкоуровневые особенности, так и семантическое содержание изображения, что критически важно для высококачественной обработки.

Хотя существуют и другие подходы, такие как классические методы суперрезолюции, цветокоррекции на основе интерполяции, а также модели сверточных (CNN) и рекуррентных (DRCN) нейронных сетей, многие из них уступают U-Net в плане способности восстанавливать мелкие детали, требуют больших вычислительных ресурсов или сложны в настройке. U-Net предлагает более сбалансированное решение, сочетающее высокую производительность с относительной простотой и эффективностью.

Таким образом, благодаря своей гибкой и мощной архитектуре, способности объединять локальные и глобальные признаки, а также продемонстрированной эффективности в смежных задачах обработки изображений, U-Net представляется одним из наиболее перспективных подходов для повышения четкости и цветокоррекции изображений.

Для удаления бликов будут разработаны методы, включающие сегментацию бликов с использованием глубоких нейронных сетей и их последующее устранение. Архитектура модели будет основана на энкодер-декодерной U-Net, которая хорошо зарекомендовала себя для задач семантической сегментации изображений.

Энкодер в U-Net будет отвечать за извлечение семантических признаков из входного изображения на разных уровнях абстракции. Эти признаки будут включать информацию о местоположении, форме и интенсивности бликов. Декодер будет использовать эти признаки для восстановления пиксельной сегментационной маски в исходном разрешении, которая будет указывать на области, содержащие блики.

В реализации энкодера и декодера U-Net мы планируем использовать эффективные сверточные блоки. Конкретно, наш код будет включать многослойные свёрточные сети, состоящие из нескольких последовательных сверточных слоёв (Conv2D) с применением функции активации LeakyReLU и InstanceNormalization, что позволит сети эффективно обучаться и извлекать значимые признаки из изображений. InstanceNormalization будет помогать стабилизировать обучение, особенно при использовании глубоких архитектур, уменьшая зависимость от размера мини-батча и улучшая переносимость модели на новые данные.

Декодер U-Net будет состоять из блоков транспонированных свёрток (Conv2DTranspose), которые будут выполнять операцию увеличения разрешения изображения (так называемая upsampling). Эти блоки также будут содержать InstanceNormalization и функцию активации ReLU, что будет способствовать стабильной генерации высококачественных изображений. В декодере также будут использоваться блоки конкатенации (Concatenate), которые будут объединять признаки из соответствующих уровней энкодера и декодера, что позволит сохранять детальную информацию и улучшать точность восстановления изображения.

Для обнаружения бликов в изображениях мы планируем использовать модель CNN. Эта модель будет состоять из нескольких свёрточных слоёв с увеличивающимся числом фильтров и операциями подвыборки (MaxPooling2D), что позволит постепенно уменьшать пространственное разрешение изображений и выделять более высокоуровневые признаки. В конечном счёте, свёрточные слои будут соединяться с полносвязными слоями (Dense) для выполнения классификации: определение наличия или отсутствия бликов.

Для восстановления областей изображений, повреждённых бликами, мы будем использовать генеративно-состязательную сеть (GAN). В GAN архитектуре, которую мы планируем использовать, будут две основные модели: генератор и дискриминатор. Генератор будет обучаться создавать реалистичные изображения без бликов, в то время как дискриминатор будет обучаться отличать реальные изображения от тех, которые были созданы генератором.

Генератор будет включать многослойные свёрточные и транспонированные свёрточные слои. Начальные слои генератора будут захватывать низкоуровневые признаки изображения, тогда как последующие слои будут рекурсивно восстанавливать и улучшать изображение, устраняя блики. Дискриминатор будет состоять из свёрточных слоёв, которые будут уменьшать пространственное разрешение изображения, выделяя ключевые признаки для определения подлинности изображения.

## Создание датасета для обучения моделей машинного обучения.

Для обучения модели, предназначенной для автоматического раскрашивания изображений, будет сформирован обширный и разнообразный датасет. Этот датасет будет включать в себя множество черно-белых и цветных фотографий. Черно-белые изображения будут использоваться в качестве входных данных, на которых модель будет обучаться. Соответствующие цветные версии этих изображений будут выступать в роли целевых выходных данных, к которым модель должна стремиться. Таким образом, модель будет учиться преобразовывать черно-белые фотографии в полноцветные изображения, максимально приближенные к оригиналу.

Для обеспечения эффективности и точности модели, в датасет войдут снимки различных сцен, объектов и условий освещения. Это будет включать фотографии пейзажей, городских улиц, а также изображений, сделанных при разном освещении — от яркого солнечного света до сумерек. Такое разнообразие обучающих примеров необходимо для того, чтобы модель могла успешно справляться с задачами раскрашивания в самых различных контекстах и ситуациях. Модель должна быть способна корректно распознавать и интерпретировать текстуры, формы и светотени, чтобы точно воссоздавать цвета, характерные для разных объектов и условий.

Кроме того, включение снимков с различными уровнями контраста и яркости поможет модели научиться адаптироваться к различным визуальным характеристикам изображений. Это обеспечит ее универсальность и способность работать с широким спектром фотографий, от старых архивных снимков до современных цифровых фотографий. Особое внимание будет уделено деталям и мелким элементам на изображениях, чтобы модель могла корректно раскрашивать как крупные объекты, так и мелкие детали, такие как лица людей или мелкие элементы архитектуры.

Для разработки модели, предназначенной для повышения четкости изображений, будет сформирован специальный датасет, состоящий из пар изображений. Каждая пара будет включать исходный снимок с низкой резкостью и его более качественную версию с высокой детализацией. Низкокачественные изображения будут использоваться в качестве входных данных, на которых модель будет обучаться, а их улучшенные аналоги будут выступать в роли целевых выходных данных, к которым модель должна стремиться. Этот подход позволит модели учиться восстанавливать и улучшать детализацию на основе примеров.

В датасет войдут фотографии, сделанные в различных условиях, с различными уровнями размытости и детализации. Это могут быть снимки, полученные при слабом освещении, при движении камеры или с использованием низкокачественной оптики. Такое разнообразие позволит модели обучаться на примерах с разными типами дефектов, что обеспечит её универсальность и способность эффективно повышать четкость изображений в различных сценариях. Включение фотографий с различными уровнями размытости поможет модели научиться различать тонкие детали и восстанавливать их в окончательных изображениях.

Особое внимание будет уделено изображениями, сделанным в условиях слабого освещения, где размытость и шумы наиболее заметны. Это позволит модели эффективно обрабатывать ночные снимки и фотографии, сделанные в помещении без достаточного освещения. Также в датасет войдут изображения с движущимися объектами, где размытость возникает из-за динамики сцены. Такие примеры помогут модели научиться корректировать размытие, вызванное движением, и восстанавливать чёткие контуры объектов.

Процесс формирования датасета требует тщательного подбора и подготовки пар изображений. Важно обеспечить высокое качество улучшенных версий, чтобы модель могла обучаться на максимально точных и детализированных примерах. Это включает в себя использование профессиональных инструментов и методов для создания высококачественных аналогов низкокачественных снимков. Таким образом, модель будет способна генерировать изображения с улучшенной четкостью, максимально приближенные к профессиональному уровню.

Для разработки модели, предназначенной для удаления бликов с фотографий, будет сформирован специализированный датасет. Этот датасет будет состоять из изображений, содержащих блики и засветки, а также их "исправленных" версий, где данные артефакты устранены. Изображения с бликами будут использоваться в качестве входных данных, а "чистые" снимки будут служить целевыми выходными данными. В набор данных войдут фотографии с различными типами бликов, возникающими из-за отражений, неправильной экспозиции, а также в различных условиях освещения и сценах.

Для эффективного обучения глубоких нейронных сетей потребуется большой объем размеченных данных. Поскольку ручная разметка бликов на реальных фотографиях является крайне трудоемкой задачей, создадим синтетический датасет с автоматической разметкой. Первым шагом станет сбор базового набора высококачественных изображений без бликов, включающего около 300 фотографий различных сцен.

Далее, с использованием библиотеки OpenCV, синтетически будут созданы блики. Для каждого изображения будет генерироваться уникальная маска бликов случайной формы, размера и интенсивности. Эти маски будут накладываться на исходные изображения путем пересвета соответствующих пикселей. Параметры, такие как угол, масштаб, яркость и форма бликов, будут варьироваться для имитации разнообразных условий съемки. Это позволит создать реалистичные блики, которые модель будет обучаться распознавать и устранять.

Для повышения обобщающей способности моделей будут применены различные методы аугментации данных. В аугментацию войдут геометрические трансформации, такие как повороты, отражения, сдвиги и масштабирование. Также будут использоваться изменения цветовых характеристик, включая регулировку яркости, контрастности, оттенков и температуры цвета. Дополнительно в изображения будут добавляться различные шумы, такие как гауссов шум, импульсный шум и периодический шум, а также искажения, такие как размытие, виньетирование и искривления.

Автоматическая разметка будет включать бинарную разметку пикселей (блик/не блик) с использованием сохраненных масок. Каждое изображение получит соответствующую маску-разметку, что позволит точно указать области с бликами. В результате будет сформирован синтетический датасет, включающий около 533 изображений с бликами и соответствующими бинарными масками, а также 680 изображений без бликов для тестирования и валидации.

Этап предобработки данных также будет включать нормализацию пикселей, обрезку лишних областей изображений и преобразование в требуемый формат для обучения сетей. Нормализация пикселей будет осуществляться для приведения всех изображений к единому стандарту, что обеспечит стабильность работы моделей. Обрезка лишних областей поможет сосредоточиться на значимых частях изображения, а преобразование формата данных позволит адаптировать их для оптимального использования в процессе обучения нейронных сетей.

Таким образом, создание и подготовка датасета для модели удаления бликов включает в себя несколько важных этапов, начиная от сбора и создания синтетических данных, до их аннотации и аугментации. Эти шаги обеспечат надежную основу для обучения модели, способной эффективно распознавать и устранять блики на фотографиях, обеспечивая высокое качество и универсальность сервиса.

Формирование специализированных наборов данных для каждой из этих задач обеспечит эффективное обучение отдельных моделей, оптимизированных для решения конкретной задачи обработки изображений с высокой точностью и производительностью. Будет проведена аннотация данных, если это необходимо, для улучшения качества обучения моделей.

Модели будут обучены на собранном датасете с использованием методов обучения с учителем. Будут применены техники регуляризации для повышения обобщающей способности моделей. Оценка моделей будет проводиться на тестовых наборах данных с последующей доработкой для повышения точности и надежности.

## Разработка пользовательского интерфейса.

Интерфейс сервиса будет разработан таким образом, чтобы быть интуитивно понятным и легким в использовании для всех категорий пользователей, независимо от их уровня технической подготовки. Особое внимание будет уделено созданию интерфейса, который обеспечит максимальное удобство и простоту взаимодействия.

Основная задача интерфейса – сделать процесс загрузки фотографий максимально простым и быстрым. Пользователи смогут загружать изображения различных размеров формата JPEG.

Еще одной важной функцией станет возможность сравнения результатов обработки с оригинальными изображениями. Для этого будет предусмотрена функция разделенного экрана, где одна половина будет показывать оригинальное изображение, а другая – обработанное.

Визуальные элементы интерфейса, такие как кнопки, меню и панели инструментов, будут расположены логично и удобно, чтобы минимизировать время, необходимое для освоения сервиса. Интерфейс будет тщательно протестирован с участием различных групп пользователей для обеспечения его удобства и функциональности.

Таким образом, интерфейс сервиса будет не только функциональным, но и приятным в использовании. Интуитивно понятный дизайн, функция сравнения результатов позволят пользователям легко и эффективно работать с сервисом. Это повысит их удовлетворенность и сделает сервис доступным и полезным как для профессионалов, так и для любителей, стремящихся улучшить качество своих фотографий.

## Тестирование и оптимизация сервиса.

Алгоритмы и модели, разработанные для сервиса автоматической обработки фотографий, будут подвергнуты тщательному тестированию на различных наборах данных для оценки их производительности и качества. Тестирование является критически важным этапом, который позволяет убедиться в том, что сервис работает корректно и эффективно в различных условиях и сценариях использования.

Для начала, тестирование будет проводиться на обширных и разнообразных наборах данных, включающих изображения с различными характеристиками и уровнями сложности. Это могут быть фотографии с различной резкостью, контрастом, освещением и наличием бликов. Использование разнообразных данных позволяет оценить, насколько хорошо модели справляются с задачами раскрашивания, повышения четкости и устранения бликов в различных ситуациях. Важно убедиться, что модели демонстрируют стабильные и качественные результаты на всех типах изображений, входящих в набор данных.

Кроме того, тестирование будет проведено на различных устройствах и платформах, чтобы обеспечить универсальность и доступность сервиса для широкой аудитории. Это включает проверку работы сервиса на настольных компьютерах, ноутбуках, планшетах и смартфонах, а также на различных операционных системах, таких как Windows, macOS, iOS и Android. Тестирование на различных устройствах позволит выявить возможные проблемы с производительностью и совместимостью, а также обеспечить, что интерфейс и функциональность сервиса остаются неизменными и удобными для всех пользователей, независимо от используемого устройства.

Особое внимание будет уделено проверке скорости работы алгоритмов и моделей. Важно, чтобы сервис предоставлял результаты обработки в разумные сроки, особенно при работе с большими объемами данных или изображениями высокого разрешения. Алгоритмы и модели будут подвергнуты тщательному тестированию на различных наборах данных и устройствах для обеспечения их производительности, качества и универсальности. Это позволит создать надежный и эффективный сервис, который будет удовлетворять потребности пользователей и работать корректно в любых условиях.

## Итоговый стек технологий и методологий.

Для реализации поставленных задач был выбран следующий стек технологий и методологий: Python, TensorFlow, Keras, OpenCV, Flask, Docker и PyTorch. Этот стек технологий был выбран для работы с машинным обучением и создания сервисов, поскольку он предлагал множество преимуществ на каждом этапе процесса.

Python стал основным языком программирования благодаря своей популярности, богатой экосистеме библиотек и простоте синтаксиса. Он обеспечивал поддержку научных вычислений и обработки данных через NumPy и pandas, а также интегрировался с ключевыми ML-библиотеками, такими как scikit-learn, TensorFlow и PyTorch, что делало его идеальным для быстрого прототипирования, тестирования гипотез и создания моделей.

TensorFlow, Keras и PyTorch предоставляли мощные инструменты для разработки и обучения сложных нейронных сетей. TensorFlow поддерживал распределенные вычисления, позволяя работать с большими объемами данных и масштабировать модели. Keras с высокоуровневым API упрощал процесс создания и обучения моделей, ускоряя разработку. Визуализационные и отладочные инструменты TensorFlow, такие как TensorBoard, помогали в мониторинге и оптимизации моделей.

OpenCV обеспечивал широкий спектр функций для обработки изображений и видео, что было важно для задач компьютерного зрения. С его помощью выполнялась предобработка данных, улучшение качества изображений и реализация различных алгоритмов анализа изображений, что было критично для подготовки данных и реализации моделей компьютерного зрения.

Flask использовался для разработки веб-приложений и создания интерфейсов взаимодействия с ML-моделями. Он позволял легко создавать REST API для моделей, упрощая их интеграцию в веб- и мобильные приложения. Flask был удобен для быстрого прототипирования сервисов и развертывания в производственной среде, обеспечивая обработку запросов и взаимодействие с пользователями.

Docker обеспечивал изоляцию среды выполнения, гарантируя воспроизводимость и стабильность работы моделей независимо от окружения. Он упрощал развертывание и обновление моделей, обеспечивая быструю доставку в производство. Docker также позволял эффективно управлять ресурсами и масштабировать ML-сервисы, что было важно для обработки большого количества запросов и работы с большими данными. Контейнеризация всех зависимостей приложения обеспечивала надежность и предсказуемость работы сервисов.

Этот стек технологий предоставлял универсальное и масштабируемое решение для создания сервиса на основе нейронных сетей, позволяя выполнять различные задачи обработки изображений, такие как раскрашивание, автоматическая коррекция и удаление бликов. Благодаря мощным инструментам для разработки, обучения и развертывания моделей машинного обучения, данный стек обеспечивал высокую производительность и возможность масштабирования сервиса в соответствии с потребностями. Он гибко адаптировался под различные задачи и сценарии использования, связанные с компьютерным зрением и обработкой визуальных данных.

## Итоговые методологии обучений нейронных сетей.

Для задачи раскрашивания черно-белых фотографий была выбрана сверточная сеть (ConvNet). Сверточные сети, благодаря своей структуре, могут создавать очень реалистичные цветовые переходы и обеспечивают точность и сохранение деталей. В ConvNet используется комбинация сверток, функций активации и операций подвыборки, что позволяет модели эффективно извлекать признаки из черно-белых изображений и предсказывать соответствующие цветовые компоненты. Использование ConvNet обеспечивает надежное и точное раскрашивание изображений, позволяя моделям восстанавливать цвета с высокой степенью реализма и детализации.

Архитектура U-Net была выбрана для задачи повышения четкости изображений благодаря своей способности сочетать сжимающий и расширяющий пути с операциями апсэмплинга и skip-connections. Свёрточные нейронные сети, подобные U-Net, имеют ряд преимуществ при работе с изображениями. Они используют свёртки для эффективного извлечения признаков на разных масштабах, что позволяет захватывать как глобальные, так и локальные особенности изображения. Кроме того, операции апсэмплинга и skip-connections обеспечивают сохранение мелких деталей и текстур, которые могут быть потеряны при сжатии. Такая архитектура позволяет модели точно восстанавливать высококачественные изображения с улучшенной четкостью и богатыми деталями, что критически важно для многих задач обработки изображений.

Для задачи удаления бликов также была выбрана U-Net благодаря её способности эффективно справляться с задачами сегментации изображений. Архитектура U-Net позволяет извлекать и использовать контекстную информацию на различных уровнях абстракции, что делает её идеальной для точного распознавания и удаления бликов. Использование эффективных сверточных блоков, таких как ResBlocks и MobileBlocks, дополнительно повышает производительность и обобщающую способность модели, что критически важно для работы в различных условиях освещения и с различными типами бликов.

В конечном итоге, выбор этих архитектур был обусловлен их способностью обеспечивать высокое качество и стабильность результатов, а также их универсальностью и адаптивностью к различным задачам и условиям. Такой подход позволяет создавать надежный и эффективный сервис автоматической обработки фотографий, который будет полезен как профессионалам, так и любителям, стремящимся улучшить качество своих изображений.

# Обучение модели

## Сбор и подготовка данных

Подготовка данных для нейронной сети, работающей с изображениями, начиналась с тщательной очистки данных. Этот процесс включал удаление различных видов шума и артефактов, которые могли негативно повлиять на качество обучения модели. В нашем случае, работая с фотографиями из личного архива, особое внимание мы уделили удалению цифровых артефактов, таких как следы компрессии и случайные помехи. Эти артефакты могли возникнуть при сохранении или передаче изображений, и их присутствие могло значительно затруднить обучение модели. Наличие подобных нежелательных элементов на изображениях могло помешать модели правильно выявлять ключевые признаки, что в свою очередь отрицательно сказалось бы на конечном результате обработки.

Процесс очистки данных включал в себя несколько этапов. На первом этапе мы вручную отобрали фотографии, которые показались нам чистыми и качественными. Это позволило избавиться от большинства очевидных дефектов.

Следующим этапом было обнаружение и удаление артефактов компрессии. Мы тщательно просмотрели отобранные изображения и удалили те, где следы компрессии были особенно заметны. Это шаг был важен для фотографий, которые проходили через множество циклов сохранения и передачи, что могло привести к значительным потерям качества. Особое внимание уделялось удалению случайных помех, таких как цифровые шумы и артефакты, вызванные неисправностями в камере или сбоями при передаче данных.

Кроме того, мы провели тщательную проверку и очистку данных для удаления любых поврежденных или искаженных изображений. Это включало выявление изображений с физическими дефектами, такими как царапины или пятна, а также с искажениями, вызванными неправильным экспонированием или ошибками при сканировании. В результате этих мероприятий удалось создать обучающий набор, состоящий только из высококачественных данных, что значительно повысило эффективность и точность обучения нейронной сети.

Таким образом, подготовка данных для нейронной сети включала комплексный процесс очистки, который обеспечивал удаление шума, артефактов и любых дефектных изображений. Этот процесс был необходим для того, чтобы гарантировать, что модель будет обучаться на данных высокого качества и сможет достигать высоких результатов в задаче обработки изображений.

После очистки данных следовал этап нормализации и выравнивания изображений. Нормализация включала приведение всех изображений к единому формату и масштабу, что позволяло модели обрабатывать их одинаково. В процессе нормализации мы выравнивали размеры изображений, приводя их к одному разрешению. Это позволило избежать ситуации, когда модель должна учитывать различия в размерах изображений, что могло бы усложнить процесс обучения. Кроме того, нормализация значений пикселей в диапазоне от 0 до 1 помогала модели лучше воспринимать различия в изображениях и ускоряла процесс обучения.

Для задачи раскрашивания изображений данные сначала преобразовывались в черно-белый формат. Этот процесс заключался в удалении всей цветовой информации из исходного изображения, оставляя только оттенки серого. Преобразование в черно-белый формат является важным этапом, поскольку модель должна была научиться восстанавливать утраченные цвета, исходя только из черно-белого изображения. Удаление цветовой информации позволило сосредоточиться на анализе яркостных значений пикселей, что помогало модели определять текстуры и градации серого, которые соответствуют различным цветам.

Преобразование включало в себя тщательный анализ яркостных значений каждого пикселя. Этот процесс позволял модели лучше понять, какие текстуры и градации серого соответствуют определенным цветам. Модель затем использовала эти данные для того, чтобы предсказать и воссоздать оригинальные цвета изображения. Она добавляла оттенки и насыщенность на основе изученных паттернов, что позволяло восстановить цветовую информацию с высокой степенью точности.

Для задачи улучшения качества изображений к изображениям применялась специальная маска размытости. Для внесения дефектов в изображения и последующего их исправления был задействован набор инструментов Python: OpenCV, NumPy и PIL (Python Imaging Library). Этот подход автоматизировал процесс добавления различных типов дефектов, таких как блики, искажения, шумы и артефакты, на фотографии, обеспечивая тем самым разнообразие и реалистичность обучающих данных для модели. OpenCV использовался для базовой обработки изображений, включая операции выделения областей, наложения фильтров и преобразований. NumPy обеспечивал эффективные вычисления для манипулирования массивами пикселей, в то время как PIL предоставлял дополнительные инструменты для загрузки, редактирования и сохранения изображений в различных форматах. Эта комбинация библиотек позволяла программно имитировать различные типы дефектов, характерных для реальных фотографий, создавая сложные сценарии для обучения модели. Полученные пары "исходное изображение - изображение с дефектами" использовались в качестве входных данных для модели, которая обучалась распознавать и устранять эти дефекты, восстанавливая оригинальное качество изображения.

Для генерации синтетических изображений с бликами был разработан Python-скрипт с использованием библиотеки OpenCV. Основная функция скрипта, которая добавляет блики, принимает путь к исходному изображению и путь для сохранения обработанного изображения. Внутри этой функции исходное изображение читается с помощью OpenCV. Затем создается пустая маска блика того же размера, что и изображение, с использованием NumPy.

Далее определены три функции для создания различных типов бликов на маске. Первая функция создает круглый блик со случайными координатами центра, радиусом, интенсивностью и степенью размытия. Координаты центра и радиус выбираются случайным образом с учетом размеров изображения. Интенсивность и размытие также задаются случайным образом. Вторая функция OpenCV рисует круглый блик на маске, затем может применяться размытие. Третья функция создает линейный блик со случайными начальной и конечной точками, интенсивностью, толщиной и степенью размытия. Эти параметры выбираются случайным образом в заданных диапазонах. Функция OpenCV рисует линейный блик на маске, после чего может применяться размытие.

Вспомогательная функция добавляет гауссовский шум на маску блика для реалистичности. Основная функция последовательно вызывает функции для создания бликов на маске с случайными параметрами. Затем на маску добавляется шум. Маска накладывается на исходное изображение с помощью OpenCV. Результат обрезается до диапазона [0, 255] с помощью NumPy и преобразуется в формат uint8. Обработанное изображение сохраняется по указанному пути.

Для классификации изображений был реализован классификатор на основе свёрточной нейронной сети (CNN) с использованием библиотеки Keras. Изображения загружались из двух папок: с бликами и без. Использовалась библиотека OpenCV, изображения изменялись до 128x128 пикселей и конвертировались в numpy-массивы. Метки классов (1 для изображений с бликами и 0 для без) объединялись с изображениями, затем данные разделялись на обучающую и тестовую выборки. Значения пикселей нормализовались путем деления на 255.

Для увеличения объема и разнообразия обучающего набора данных мы применяли аугментацию. Аугментация данных — это процесс искусственного увеличения обучающего набора путем применения различных преобразований к исходным изображениям. В нашем случае мы использовали несколько методов аугментации. Один из них — это повороты изображений на случайные углы, что позволило модели стать устойчивой к ориентации объектов на фотографии. Масштабирование, то есть изменение размера изображений, помогало модели обрабатывать объекты различных размеров, что особенно важно при работе с фотографиями, где объекты могли быть представлены в разном масштабе. Также мы применяли обрезку, случайно удаляя части изображения, что увеличивало вариативность данных и позволяло модели фокусироваться на различных частях изображения, выявляя ключевые признаки.

Эти методы позволили создать множество вариаций одного и того же изображения, что существенно увеличило объем и разнообразие обучающего набора данных. Все эти шаги в совокупности обеспечили подготовку данных, необходимую для эффективного обучения нейронной сети и достижения наилучших результатов в обработке изображений.

## Обучение модели ИИ

Для решения задачи автоматической обработки фотографий, включающей раскрашивание черно-белых изображений, улучшение их четкости и устранение бликов, было рассмотрено несколько подходов, каждый из которых имел свои преимущества и недостатки. Основной целью было создание эффективной модели машинного обучения, способной справляться с широким спектром задач по обработке изображений. Среди рассмотренных подходов были классические методы компьютерного зрения и современные методы глубокого обучения.

Классические методы, такие как использование различных фильтров и алгоритмов преобразования, предлагали определенные преимущества в плане вычислительной эффективности и простоты реализации. Однако они были ограничены в своих возможностях и не могли обеспечить необходимого уровня качества для сложных задач, таких как раскрашивание черно-белых фотографий. Глубокие нейронные сети, с другой стороны, предлагали гораздо более высокие возможности за счет своей способности обучаться на больших объемах данных и извлекать сложные, нелинейные зависимости. После тщательного анализа и сравнений было решено использовать подход, основанный на глубоких нейронных сетях, ввиду их превосходства в обработке изображений и адаптивности к различным условиям.

Для решения задачи автоматической обработки фотографий, включающей в себя раскрашивание черно-белых изображений, улучшение их четкости и устранение бликов, была выбрана архитектура модели, основанная на энкодер-декодерной U-Net. Этот тип сети отлично зарекомендовал себя для задач семантической сегментации изображений благодаря своей способности эффективно извлекать и восстанавливать информацию на различных уровнях абстракции. Энкодер в этой архитектуре извлекает семантические признаки из входного изображения, постепенно уменьшая его размерность, а декодер использует эти признаки для восстановления пиксельной сегментационной маски до исходного разрешения.

Энкодер модели состоит из нескольких сверточных блоков, каждый из которых включает операции свертки, нелинейной активации (ReLU) и объединения (максимального или среднего). Эти блоки предназначены для последовательного уменьшения размерности карт признаков, захватывая все более абстрактные семантические особенности изображения. Таким образом, модель может извлекать значимые признаки, необходимые для дальнейшего восстановления изображения. Эти признаки включают такие важные элементы, как границы объектов, текстуры и другие ключевые визуальные характеристики, которые помогают модели понимать структуру изображения и контекст.

Декодер, в свою очередь, содержит соответствующее количество блоков апсэмплинга (UpSampling), которые восстанавливают исходное разрешение изображения. Ключевой особенностью U-Net является использование skip-connections, которые объединяют карты признаков энкодера с соответствующими картами декодера. Это позволяет декодеру использовать как глобальные, так и локальные признаки, обеспечивая точную сегментацию и восстановление изображения. Skip-connections играют важную роль в сохранении деталей изображения, поскольку они передают информацию, которая могла бы быть утрачена при последовательных операциях свертки и подвыборки.

В реализации использовались эффективные сверточные блоки, такие как ресблоки (ResBlocks) и мобильные блоки (MobileBlocks), для построения энкодера и декодера. Эти блоки улучшают обобщающую способность модели и повышают эффективность вычислений, что особенно важно при работе с большими объемами данных и высокими требованиями к качеству обработки изображений. ResBlocks помогают справляться с проблемой исчезающих градиентов, обеспечивая стабильное и глубокое обучение, а MobileBlocks делают модель более легкой и быстрой, что особенно важно для обработки изображений в реальном времени.

Рассмотрим подробнее каждую из задач.

### Задача автоулучшения качества.

Для создания модели автокоррекции изображений была использована архитектура сверточной нейронной сети U-Net. Эта архитектура состоит из сжимающего пути, который извлекает признаки изображения на различных уровнях абстракции, и расширяющего пути, который восстанавливает изображение с учетом контекста из более глубоких слоев. Она позволяет модели эффективно извлекать и объединять признаки изображения на различных уровнях абстракции, сохраняя пространственную информацию и работая с изображениями разного размера, что важно для задач автокоррекции и улучшения качества изображений.

Данная модель представляет собой реализацию архитектуры сверточной нейронной сети U-Net с использованием библиотеки TensorFlow и фреймворка Keras для задачи автокоррекции и улучшения качества изображений. U-Net состоит из сжимающего пути (encoder), извлекающего признаки изображения на разных уровнях абстракции, и расширяющего пути (decoder), восстанавливающего изображение с учетом контекстной информации из более глубоких слоев.

Функция unet\_model принимает размер входного изображения (256, 256, 3), где 256 - ширина и высота, а 3 - количество цветовых каналов (RGB). Модель начинается с входного слоя Input, принимающего изображение указанного размера. Сжимающий путь состоит из последовательностей сверточных слоев Conv2D с ядром 3x3, активационной функцией ReLU и операций максимального пулинга MaxPooling2D с шагом 2. Эта последовательность повторяется четыре раза с удвоением количества фильтров на каждом уровне (64, 128, 256, 512).

Расширяющий путь начинается с операции апсемплинга UpSampling2D с шагом 2, затем результат конкатенируется (concatenate) с соответствующими признаками из сжимающего пути для сохранения деталей изображения. После конкатенации применяются два сверточных слоя Conv2D с ядром 3x3 и активационной функцией ReLU. Эта последовательность повторяется четыре раза с уменьшением количества фильтров на каждом уровне (512, 256, 128, 64).

Выходной слой Conv2D с ядром 1x1 и сигмоидальной активационной функцией генерирует изображение с тремя каналами (RGB), представляющее собой автокоррекцию входного изображения.

Для создания модели автокоррекции изображений была использована архитектура сверточной нейронной сети U-Net. Эта архитектура состоит из сжимающего пути, который извлекает признаки изображения на различных уровнях абстракции, и расширяющего пути, который восстанавливает изображение с учетом контекста из более глубоких слоев.

Обучение модели выполняется методом model.fit со следующими гиперпараметрами: 100 эпох для достаточного времени обучения, размер батча 16 для баланса между скоростью и использованием памяти, 10% данных для валидации и предотвращения переобучения.

График метрики Accuracy и функции потерь в зависимости от числа эпох см. на рисунке 1.

|  |
| --- |
| Изображение выглядит как текст, диаграмма, снимок экрана, График  Автоматически созданное описание |
| Рисунок 1. Графики корректности обученности модели |

Пример осуществления моделью оптимизации качества фотографии см. на рисунке 2.

|  |
| --- |
| Изображение выглядит как текст, дерево, снимок экрана, на открытом воздухе  Автоматически созданное описание |
| Рисунок 2. Сравнение оригинала с обработанным фото |

### Модель раскрашивания фотографий

Для решения задачи раскрашивания фотографий была выбрана сверточная сеть (ConvNet). Эта сеть включает несколько ключевых компонентов, таких как сверточные слои, функции активации, нормализации и подвыборки, которые совместно работают для достижения высокой точности и эффективности обработки изображений.

Первый слой сети принимает на вход черно-белое изображение, представленное одним каналом. Этот слой состоит из сверточного слоя, который включает 32 фильтра с размером ядра 5x5. Основная задача этого слоя — применение свертки к входному изображению для выделения первичных признаков и текстур. Для того чтобы сохранить размер входного изображения неизменным, используется padding. Это важно, так как позволяет сети сохранять пространственные характеристики изображения, что критично для задач раскрашивания. После свертки используется функция активации ReLU (Rectified Linear Unit), которая добавляет нелинейность модели, позволяя ей обучаться более сложным и разнообразным признакам. Далее следует слой пакетной нормализации (BatchNorm2d), который стабилизирует процесс обучения и улучшает производительность модели, позволяя ей быстрее сходиться и избегать переобучения. Первый слой завершается операцией подвыборки (MaxPool2d) с размером ядра 2x2, которая уменьшает пространственное разрешение изображения вдвое, сохраняя при этом важные признаки, что помогает уменьшить вычислительную сложность и улучшить эффективность обработки.

Второй слой сети принимает выход первого слоя и обрабатывает его. Он также состоит из сверточного слоя, но уже с 8 фильтрами и размером ядра 3x3. Этот слой также использует padding для сохранения размера входного изображения, что позволяет сети сохранять целостность пространственных признаков на всех уровнях обработки. Включение функции активации ReLU добавляет нелинейность и в этом слое, обеспечивая модели способность обучаться более сложным паттернам и признакам. Для стабилизации активаций и улучшения сходимости обучения используется слой BatchNorm2d. После этого, как и в первом слое, применяется операция подвыборки с размером ядра 2x2, которая снова уменьшает пространственное разрешение изображения вдвое. Это помогает модели фокусироваться на более крупных и значимых признаках, что особенно важно для восстановления и раскрашивания изображений.

Третий слой сети является завершающим сверточным слоем. Он принимает 8-канальный выход второго слоя и преобразует его в 2-канальный выход, представляющий цветовые компоненты изображения. Для этого используется сверточный слой с ядром 3x3 и padding, который сохраняет размер входного изображения. На этом этапе модель формирует окончательный вывод, состоящий из цветных компонентов, что позволяет раскрасить черно-белое изображение с учетом изученных признаков и текстур.

В процессе обучения сверточной сети используется функция потерь MSELoss (Mean Squared Error Loss), которая измеряет среднеквадратичное отклонение между предсказанными значениями и истинными значениями цветовых компонентов изображения. Эта функция потерь позволяет модели корректировать свои предсказания и минимизировать ошибки, что критично для получения высококачественных результатов. Для оптимизации параметров сети применяется оптимизатор RMSprop с фиксированной скоростью обучения. RMSprop является одним из наиболее эффективных оптимизаторов для задач с большими объемами данных, так как он адаптирует скорость обучения для каждого параметра, что позволяет модели быстрее сходиться и избегать переобучения.

График зависимости функции потерь от числа эпох см. на рисунке 3.

|  |
| --- |
| *Изображение выглядит как текст, снимок экрана, линия, График  Автоматически созданное описание* |
| Рисунок 3. График функции ошибки |

Результат работы модели см. на рисунке 4. Крайнее левое фото - оригинал, фото по центру - чёрно-белая версия, которую модель принимает на вход, справа - результат обработки.

|  |  |  |
| --- | --- | --- |
| Изображение выглядит как на открытом воздухе, небо, горизонт, забор  Автоматически созданное описание | Изображение выглядит как черно-белый, на открытом воздухе, монохромный, небо  Автоматически созданное описание | Изображение выглядит как на открытом воздухе, небо, пляж, вода  Автоматически созданное описание |
| Рисунок 4. Результат работы модели | | |

### Модель устранения бликов

Для обнаружения бликов на изображении использовалась архитектура U-Net. Архитектура U-Net является одним из наиболее популярных подходов для решения задач сегментации изображений, что делает ее подходящей и для задач восстановления изображений, таких как удаление бликов. В представленной модели используются сверточные слои, пулинг, а также транспонированные сверточные слои, что позволяет ей эффективно обрабатывать изображения и восстанавливать их из искаженных данных.

Модель UNet состоит из двух основных частей: контрактной (downsampling) и расширяющей (upsampling) путей. В коде, представленном выше, эти пути реализованы в виде сверточных блоков и транспонированных сверточных слоев.

Контрактная часть модели состоит из четырех последовательных сверточных блоков, каждый из которых включает в себя свертку (Conv2d), нормализацию (BatchNorm2d) и активацию (ReLU). Эти блоки постепенно уменьшают размер пространственных данных, увеличивая при этом количество каналов.

Каждый блок CBR (Conv2d-BatchNorm2d-ReLU) включает в себя следующие операции:

1. Conv2d: Сверточный слой с ядром 3x3 и паддингом 1, что позволяет сохранять размер входного изображения.
2. BatchNorm2d: Слой нормализации, который ускоряет обучение и стабилизирует его.
3. ReLU: Нелинейная функция активации, которая помогает модели учиться более сложным представлениям.

Расширяющая часть включает в себя транспонированные сверточные слои (ConvTranspose2d) и сверточные блоки CBR, которые восстанавливают пространственное разрешение данных, одновременно объединяя информацию из соответствующих слоев контрактной части через операции конкатенации. Транспонированные сверточные слои увеличивают размер пространственных данных, а сверточные блоки обрабатывают и восстанавливают детали изображения.

U-Net была выбрана для задачи удаления бликов по нескольким причинам:

1. Способность восстанавливать мелкие детали: Блики часто представляют собой мелкие, но важные детали на изображениях. Прямые соединения между соответствующими слоями контрактной и расширяющей частей позволяют модели эффективно восстанавливать такие детали.
2. Эффективное использование информации: UNet позволяет объединять локальную и глобальную информацию, что важно для удаления бликов, так как они могут проявляться как локально (мелкие блики), так и глобально (размытые области).
3. Простота адаптации: Архитектура UNet легко адаптируется к различным задачам обработки изображений, что позволяет использовать ее не только для удаления бликов, но и для других задач восстановления изображений без значительных изменений.
4. Высокая точность: UNet обеспечивает высокую точность восстановления изображений, что является критически важным для задач, связанных с улучшением качества изображений и удалением артефактов, таких как блики.

Архитектура UNet представляет собой мощный инструмент для задач сегментации и восстановления изображений. В контексте задачи удаления бликов с фотографий, ее преимущества в виде способности восстанавливать мелкие детали, эффективного использования вычислительных ресурсов и универсальности делают ее оптимальным выбором. Несмотря на некоторые недостатки, такие как большое количество параметров и сложность настройки, преимущества архитектуры значительно перевешивают, что делает ее востребованной в широком спектре приложений компьютерного зрения.

График зависимости функции потерь от числа эпох см. на рисунке 5.

|  |
| --- |
|  |
| Рисунок 5. График функции потерь |

Результат работы модели см. на рисунке 6.

|  |
| --- |
|  |
| Рисунок 6. Результат работы модели |

# Имплементация модели

В рамках данного проекта была поставлена задача создания веб-сервиса для обработки фотографий с помощью искусственного интеллекта. Для достижения этой цели потребовалось решить ряд задач, связанных с разработкой пользовательского интерфейса, серверной части и интеграцией обученной модели машинного обучения.

На первом этапе была спроектирована архитектура веб-приложения, которая включала в себя клиентскую и серверную части. Для создания клиентской части был выбран фреймворк Vue.js, предоставляющий гибкие возможности для построения интерактивных пользовательских интерфейсов. Разработка велась с использованием современных подходов, таких как компонентная архитектура и функциональное программирование.

Клиентская часть включала в себя следующие основные компоненты:

1. Компонент загрузки изображения, позволяющий пользователю выбрать фотографию из локального хранилища.
2. Компонент предварительного просмотра, отображающий выбранное изображение.
3. Компонент настройки параметров обработки, предоставляющий пользователю возможность выбора различных опций, таких как удаление дефектов, коррекция экспозиции и т.д.
4. Компонент отображения результата, демонстрирующий обработанное изображение после применения алгоритмов ИИ.
5. Компонент загрузки обработанного изображения, позволяющий пользователю сохранить результат на локальное устройство.

Для разработки серверной части нашего веб-приложения был выбран фреймворк Flask. Это решение было обусловлено несколькими ключевыми факторами, связанными с особенностями Flask, его преимуществами и удобством использования в процессе имплементации. В первую очередь это:

* Простота и лёгкость

Flask является микро-фреймворком, что означает его минимализм и лёгкость в освоении и использовании. Он предоставляет разработчику основу для создания веб-приложений, оставляя свободу выбора дополнительных инструментов и библиотек. Простота Flask позволила нам быстро развернуть базовый сервер и сосредоточиться на основной логике обработки изображений и взаимодействии с моделью искусственного интеллекта.

* Гибкость и расширяемость

Фреймворк предоставляет гибкость в выборе и интеграции различных компонентов, таких как базы данных, механизм авторизации и шаблоны. Благодаря своей модульной архитектуре, Flask позволяет легко расширять функциональность приложения, добавляя необходимые библиотеки и плагины. Это стало особенно полезным при интеграции модели машинного обучения и обработке изображений с использованием TensorFlow, OpenCV и NumPy.

* Лёгкость интеграции с другими технологиями

Flask легко интегрируется с различными технологиями и инструментами, необходимыми для разработки веб-приложений. В нашем случае это было критично для успешной интеграции обученной модели искусственного интеллекта и обработки изображений. Flask позволил без труда настроить маршрутизацию для загрузки и обработки изображений, а также взаимодействие с фронтендом приложения.

Для разработки клиентской части был использован фреймворк Vue.js. Его использование было обусловлено рядом причин:

* Модульность и переиспользуемость компонентов

Одной из ключевых особенностей Vue.js является компонентная архитектура, которая позволяет создавать переиспользуемые и изолированные компоненты. Это значительно упрощает разработку и поддержку кода, а также способствует лучшей организованности проекта. В нашем приложении мы создали отдельные компоненты для загрузки изображений, отображения результатов и выбора параметров обработки.

* Реактивность и производительность

Vue.js обеспечивает высокую производительность и реактивность, что позволяет создавать быстрые и отзывчивые интерфейсы. Механизм реактивности Vue.js автоматически отслеживает изменения в данных и обновляет пользовательский интерфейс, что значительно улучшает пользовательский опыт. Это особенно важно для нашего приложения, где пользователь должен видеть результаты обработки изображений в реальном времени. Использование Vuex позволило обеспечить решение в виде центрального хранилища данных. В этом хранилище хранятся общие данные, которые могут применяться в любом компоненте приложения. В случае сервиса для обработки фото – ссылки на исходное изображение, закодированные файлы картинок и параметры обработки, которые будут переданы на сервер.

* Лёгкость интеграции с серверной частью

Vue.js позволяет легко интегрировать клиентскую часть с сервером через API. В нашем случае, взаимодействие с серверной частью на Flask было реализовано с использованием Axios — популярной библиотеки для выполнения HTTP-запросов. Это обеспечило простое и эффективное общение между клиентом и сервером.

Полный программный код клиентской части и иконка сайта представлены в приложениях Б и В соответственно. Полный программный код серверной части представлен в приложении А.

Ключевым компонентом серверной части являлся модуль обработки изображений, который включал в себя следующие основные функции:

1. Функция предобработки изображения, выполняющая необходимые преобразования, такие как изменение размера и нормализация пикселей.

2. Функция обработки изображения с помощью обученной модели ИИ, которая принимала предобработанное изображение и применяла алгоритмы удаления дефектов, коррекции чёткости и другие необходимые преобразования.

3. Функция постобработки изображения, выполняющая финальные преобразования, такие как кодирование в необходимый формат.

Для успешной интеграции обученной модели машинного обучения потребовалось решить ряд задач, связанных с совместимостью форматов данных и оптимизацией производительности. Модель, обученная на языке Python с использованием библиотек PyTorch и Keras, была загружена в память сервера для обеспечения быстрого доступа.

Стандартный сценарий использования сервиса выглядит следующим образом:

1. Открытие веб-сайта: Пользователь открывает веб-сайт сервиса в своём браузере.
2. Загрузка фотографии: Пользователь нажимает на кнопку "Upload file" и выбирает фотографию, которую хочет обработать, с локального компьютера.

На рисунке 7 представлена основная страница сайта, которую пользователь видит после перехода по ссылке.

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер  Автоматически созданное описание |
| Рисунок 7. Главная страница |

1. Выбор параметров обработки: после загрузки фотографии пользователь имеет возможность выбрать параметры обработки изображения, такие как добавление цвета на чёрно-белое изображение, улучшение резкости или удаление бликов.

На рисунке 8 показано, как пользователь осуществляет выбор параметров.

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, программное обеспечение, Веб-сайт  Автоматически созданное описание |
| Рисунок 8. Выбор параметров обработки |

1. Обработка изображения: после выбора параметров обработки пользователь нажимает кнопку "Submit". Изображение отправляется на сервер, где происходит обработка с использованием обученной модели искусственного интеллекта.
2. Отображение результатов: Обработанное изображение отображается на экране пользователя в интерфейсе веб-сайта. Пользователь может просмотреть результаты обработки и оценить качество работы сервиса.

На рисунке 9 представлена страница, отображающая результат обработки фото.

|  |
| --- |
| Изображение выглядит как снимок экрана, текст, программное обеспечение, компьютер  Автоматически созданное описание |
| Рисунок 9. Демонстрация исходного и обработанного изображений |

1. Скачивание результатов: Пользователь имеет возможность скачать обработанное изображение на свой компьютер, нажав на кнопку "Download the result".
2. Завершение: после просмотра результатов обработки пользователь завершает сеанс использования, закрыв веб-сайт или перейдя на другую страницу.

Этот стандартный сценарий использования обеспечивает пользователю удобный и интуитивно понятный способ обработки фотографий с помощью нашего сервиса.

Таким образом, в результате проделанной работы был создан полнофункциональный веб-сервис для обработки фотографий с помощью искусственного интеллекта, который позволяет пользователям загружать изображения, настраивать параметры обработки и получать новые версии фотографий с добавленным цветом, улучшенным качеством и удаленными дефектами.

# Заключение

В ходе выполнения данной работы была достигнута главная цель - разработан сервис для автоматической обработки фотографий с использованием методов машинного обучения. Созданный сервис позволяет раскрашивать черно-белые изображения, повышать резкость и четкость фотографий, а также устранять нежелательные блики и засветы. Функционал сервиса открывает новые возможности в сфере обработки визуального контента для различных областей применения.

Для удобства использования был разработан интуитивно понятный пользовательский интерфейс, позволяющий легко загружать фотографии, применять необходимые улучшения и получать результаты обработки. Интерфейс был спроектирован с учетом принципов юзабилити, делая сервис доступным как для профессионалов, так и для любителей. Основой пользовательского интерфейса является фреймворк Vue.js. Компонентный подход, применяемый во Vue.js, позволил создать модульную и легко расширяемую структуру пользовательского интерфейса. Ключевыми компонентами являются: загрузка изображения, предварительный просмотр, выбор параметров обработки, отображение результатов и сохранение обработанного изображения. Благодаря реактивности Vue.js, пользователь может видеть результаты обработки изображений в режиме реального времени, что значительно повышает удобство использования сервиса.

Серверная часть была разработана с использованием фреймворка Flask, который обеспечил гибкость, расширяемость и легкую интеграцию с различными технологиями, такими как TensorFlow, OpenCV, NumPy и PyTorch. Ключевым компонентом серверной части является модуль обработки изображений, реализующий предобработку, применение алгоритмов ИИ и постобработку изображений. Успешная интеграция обученных моделей машинного обучения потребовала решения задач, связанных с совместимостью форматов данных и оптимизацией производительности.

Взаимодействие между клиентской и серверной частями было организовано через API с использованием библиотеки Axios, что обеспечило простое и эффективное общение между клиентом и сервером.

Для решения задачи раскрашивания черно-белых фотографий была использована сверточная нейронная сеть, обученная на большом наборе данных. Применение передовой архитектуры и современных методов оптимизации позволило добиться высокого качества раскрашивания, сохраняя детали и естественную цветопередачу.

Для обнаружения и устранения бликов была реализована двухэтапная модель. Первая часть, основанная на архитектуре U-Net, отвечала за точное обнаружение и локализацию бликов. Вторая часть, использующая генеративную состязательную сеть (GAN) DRCN, восстанавливала качество поврежденных областей, генерируя реалистичные результаты без бликов.

Задача повышения резкости и детализации изображений решалась с помощью методов суперрезолюции на основе глубоких сверточных нейронных сетей. Это позволило существенно увеличить качество фотографий, сохраняя при этом их естественность и избегая артефактов.

В ходе работы были исследованы существующие методы и технологии обработки изображений, проанализированы их сильные и слабые стороны. На основе полученных данных были разработаны оптимальные алгоритмы и архитектуры нейронных сетей для решения поставленных задач.

Результаты проделанной работы имеют широкие перспективы практического применения в различных областях, где качество визуального контента играет ключевую роль: фотография, издательское дело, реклама, электронная коммерция и другие. Сервис позволит повысить эффективность и производительность процессов обработки изображений, обеспечивая высокое качество результатов.

В дальнейшем рекомендуется продолжить исследования и развитие в данном направлении для расширения функциональных возможностей сервиса. Перспективными направлениями являются:

1. Интеграция дополнительных инструментов обработки изображений, таких как удаление нежелательных объектов, корректировка освещения, стилизация и применение художественных эффектов.
2. Развитие мультимодальных возможностей, позволяющих принимать на вход не только изображения, но и текстовые описания или инструкции для более гибкой и целенаправленной обработки.
3. Оптимизация производительности и масштабируемости сервиса для эффективной работы с большими объемами данных и обеспечения быстрого отклика.
4. Дальнейшее совершенствование интерфейса и удобства использования на основе обратной связи от пользователей для достижения максимальной доступности и удовлетворенности.
5. Исследование этических аспектов и разработка принципов ответственного использования ИИ-технологий в сфере обработки изображений для предотвращения злоупотреблений и негативных последствий.

Реализация предложенных рекомендаций позволит вывести сервис на новый уровень, расширив его возможности и область применения, а также обеспечив устойчивое развитие в долгосрочной перспективе в соответствии с современными тенденциями и потребностями рынка.

# Приложения

## Приложение А – программный код серверной части

### Серверная часть

import io  
import os  
  
from flask import Flask, render\_template, request, send\_file  
from flask\_cors import CORS  
import cv2 as cv  
import numpy as np  
import torch  
from PIL import Image  
  
from py.predict import sharpen  
from py.train import ConvNet  
from py.utils import cvt2Lab, upsample, cvt2rgb  
  
app = Flask(\_\_name\_\_, static\_folder="static", template\_folder="static", static\_url\_path="/")  
cors = CORS()  
cors.init\_app(app, resource={r"/api/\*": {"origins": "\*"}})  
  
MODEL\_PATH = "model/image\_colorization\_model-good.pt"  
  
  
@app.route("/", defaults={"path": ""})  
@app.route("/<path:path>")  
def index(path):  
 return render\_template("index.html")  
  
  
@app.route('/upload', methods=['POST'])  
def upload():  
 for fname in request.files:  
 photo = request.files.get(fname)  
 in\_memory\_file = io.BytesIO()  
 photo.save(in\_memory\_file)  
 data = np.frombuffer(in\_memory\_file.getvalue(), dtype=np.uint8)  
 color\_image\_flag = 1  
 img = cv.imdecode(data, color\_image\_flag)  
 result = img  
 colorized = request.form.get('colorize')  
 sharpened = request.form.get('sharpen')  
 remove\_glare = request.form.get('removeGlare')  
 if colorized.lower() == 'true':  
 result = np.array(colorize(result))  
  
 if sharpened.lower() == 'true':  
 result = np.array(sharpen(result))  
  
 if remove\_glare.lower() == 'true':  
 result = np.array(remove\_glare(result))  
  
 image\_bytes = cv.imencode('.jpg', result)[1].tobytes()  
 return send\_file(io.BytesIO(image\_bytes), mimetype="image/jpg")  
  
  
def get\_model():  
 model = ConvNet()  
 model.load\_state\_dict(torch.load(MODEL\_PATH, map\_location='cpu'))  
 return model  
  
  
def preprocess\_image(image):  
 img, real\_size = (image, image.shape[:2])  
 img\_light, \_ = cvt2Lab(img)  
 img\_input = np.expand\_dims(img\_light, axis=0)  
 img\_input = np.expand\_dims(img\_input, axis=0)  
 img\_input = torch.autograd.Variable(torch.from\_numpy(img\_input).float(), requires\_grad=False)  
 print(str(img\_light.shape) + 'preprocess')  
 return img\_light, img\_input, real\_size  
  
  
def process\_image(img, img\_light):  
 img = np.squeeze(img, axis=0)  
 img = np.transpose(img.astype(float), (1, 2, 0))  
 img = upsample(img)  
 print(str(img.shape) + 'after upscale')  
 img = np.insert(img, 0, img\_light, axis=2)  
 img = (cvt2rgb(img) \* 255.).astype(np.uint8)  
 return img  
  
  
def colorize(image):  
 model = get\_model()  
 img\_light, img\_input, real\_size = preprocess\_image(image)  
  
 img = model(img\_input).cpu().data.numpy()  
 img = process\_image(img, img\_light)  
 img\_result = Image.fromarray(img)  
 img\_result = img\_result.resize((real\_size[1], real\_size[0]))  
 return img\_result  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True, port=9000)

### Модель раскрашивания изображений

import torch  
import numpy as np  
from PIL import Image  
from skimage import io, color  
from skimage.transform import rescale, resize  
  
MODEL\_PATH = "../model/image\_colorization\_model-good.pt"  
  
  
def get\_model():  
 model = ConvNet()  
 model.load\_state\_dict(torch.load(MODEL\_PATH, map\_location='cpu'))  
 return model  
  
  
def preprocess\_image(inp\_imgpath):  
 img, real\_size = read\_image(inp\_imgpath)  
 img\_light, \_ = cvt2Lab(img)  
 img\_input = np.expand\_dims(img\_light, axis=0)  
 img\_input = np.expand\_dims(img\_input, axis=0)  
 img\_input = torch.autograd.Variable(torch.from\_numpy(img\_input).float(), requires\_grad=False)  
 return img\_light, img\_input, real\_size  
  
  
def process\_image(img, img\_light):  
 img = np.squeeze(img, axis=0)  
 img = np.transpose(img.astype(float), (1, 2, 0))  
 img = upsample(img)  
 print(str(img.shape) + 'after upscale')  
 img = np.insert(img, 0, img\_light, axis=2)  
 img = (cvt2rgb(img) \* 255.).astype(np.uint8)  
 return img  
  
  
class ConvNet(torch.nn.Module):  
 *"""Convolutional network that performs image colorization."""* def \_\_init\_\_(self, num\_classes=10):  
 super(ConvNet, self).\_\_init\_\_()  
 self.layer1 = torch.nn.Sequential(  
 torch.nn.Conv2d(1, 32, kernel\_size=5, padding=2),  
 torch.nn.ReLU(),  
 torch.nn.BatchNorm2d(32),  
 torch.nn.MaxPool2d(kernel\_size=2)  
 )  
  
 self.layer2 = torch.nn.Sequential(  
 torch.nn.Conv2d(32, 64, kernel\_size=3, padding=1),  
 torch.nn.ReLU(),  
 torch.nn.BatchNorm2d(64),  
 torch.nn.MaxPool2d(kernel\_size=2)  
 )  
  
 self.layer3 = torch.nn.Sequential(  
 torch.nn.Conv2d(64, 128, kernel\_size=3, padding=1),  
 torch.nn.ReLU(),  
 torch.nn.BatchNorm2d(128),  
 torch.nn.Conv2d(128, 64, kernel\_size=3, padding=1),  
 torch.nn.ReLU(),  
 torch.nn.BatchNorm2d(64)  
 )  
  
 self.layer4 = torch.nn.Sequential(  
 torch.nn.Conv2d(64, 32, kernel\_size=3, padding=1),  
 torch.nn.ReLU(),  
 torch.nn.BatchNorm2d(32),  
 torch.nn.Conv2d(32, 2, kernel\_size=3, padding=1)  
 )  
  
 self.upsample = torch.nn.Upsample(scale\_factor=4, mode='bilinear', align\_corners=True)  
  
 self.refine = torch.nn.Sequential(  
 torch.nn.Conv2d(2, 32, kernel\_size=3, padding=1),  
 torch.nn.ReLU(),  
 torch.nn.BatchNorm2d(32),  
 torch.nn.Conv2d(32, 2, kernel\_size=3, padding=1)  
 )  
  
 def forward(self, x):  
 x = self.layer1(x)  
 x = self.layer2(x)  
 x = self.layer3(x)  
 x = self.layer4(x)  
 x = self.upsample(x)  
 x = self.refine(x)  
 return x  
  
  
def read\_image(filename, size=(256, 256), training=False):  
 img = io.imread(filename)  
 real\_size = img.shape  
 if img.shape != size and not training:  
 img = resize(img, size, anti\_aliasing=False)  
 if len(img.shape) == 2:  
 img = np.stack([img, img, img], 2)  
 return img, real\_size[:2]  
  
  
def cvt2Lab(image):  
 Lab = color.rgb2lab(image)  
 return Lab[:, :, 0], Lab[:, :, 1:] *# L, ab*def cvt2rgb(image):  
 return color.lab2rgb(image)  
  
  
def upsample(image):  
 scale = (1, 1, 1)  
 return rescale(image, scale, mode='constant', order=3)

### Модель улучшения качества

import numpy as np  
from PIL import Image  
import concurrent.futures  
  
from keras.src.saving import load\_model  
  
*# Путь к модели относительно текущего файла*model\_path = 'model/photo\_sharpen\_model.h5'  
  
*# Параметр, определяющий насколько наложены одна на другую части изображения при их разделении на блоки*overlap = 10 *# Можно менять, до 200 тояно, дальше не знаю, на 255 упало)*def split\_image(image, size=(256, 256), overlap=0):  
 width, height = image.size  
 slices = []  
 positions = []  
 for x in range(0, width, size[0] - overlap):  
 for y in range(0, height, size[1] - overlap):  
 box = (x, y, min(x + size[0], width), min(y + size[1], height))  
 slices.append(image.crop(box))  
 positions.append((x, y))  
 return slices, positions  
  
  
def merge\_images(slices, positions, original\_size):  
 width, height = original\_size  
 result = Image.new('RGB', (width, height))  
 for slice\_img, (x, y) in zip(slices, positions):  
 result.paste(slice\_img, (x, y))  
 return result  
  
  
def predict\_slice(model, input\_slice, target\_size):  
 input\_slice\_resized = input\_slice.resize(target\_size)  
 input\_slice\_resized = np.array(input\_slice\_resized) / 255.0  
 predicted\_slice = model.predict(np.expand\_dims(input\_slice\_resized, axis=0))[0]  
 predicted\_img = Image.fromarray((predicted\_slice \* 255).astype(np.uint8)).resize(input\_slice.size)  
 return predicted\_img  
  
  
def load\_images(image):  
 images = []  
 original\_sizes = []  
 img = Image.fromarray(image)  
 original\_sizes.append(img.size)  
 img = img.convert('RGB') *# Convert image to RGB* images.append(np.array(img))  
 return images, original\_sizes  
  
  
def sharpen(image):  
 *# Загрузка обученной модели* model = load\_model(model\_path)  
  
 *# Загрузка изображений без изменения размера* x\_test, input\_sizes = load\_images(image)  
  
 *# Фиксированный размер для подачи в модель* target\_size = (256, 256)  
  
 *# Предсказание изображений по частям и их объединение* predicted\_images = []  
 for input\_img, original\_size in zip(x\_test, input\_sizes):  
 input\_pil\_img = Image.fromarray(input\_img)  
 input\_slices, positions = split\_image(input\_pil\_img, size=target\_size, overlap=overlap)  
  
 with concurrent.futures.ThreadPoolExecutor() as executor:  
 futures = []  
 for input\_slice in input\_slices:  
 futures.append(executor.submit(predict\_slice, model, input\_slice, target\_size))  
 predicted\_slices = [future.result() for future in futures]  
  
 predicted\_img = merge\_images(predicted\_slices, positions, input\_pil\_img.size)  
 predicted\_images.append(predicted\_img)  
  
 return predicted\_images[0]

### Модель удаления бликов

import torch  
from PIL import Image  
from torch import nn  
from torchvision import transforms  
import matplotlib.pyplot as plt  
  
  
MODEL\_PATH = "../model/unet\_glare\_removal.pth"  
  
  
class UNet(nn.Module):  
 def \_\_init\_\_(self, in\_channels=3, out\_channels=3):  
 super(UNet, self).\_\_init\_\_()  
  
 def CBR(in\_channels, out\_channels):  
 return nn.Sequential(  
 nn.Conv2d(in\_channels, out\_channels, kernel\_size=3, padding=1),  
 nn.BatchNorm2d(out\_channels),  
 nn.ReLU(inplace=True)  
 )  
  
 self.enc1 = CBR(in\_channels, 64)  
 self.enc2 = CBR(64, 128)  
 self.enc3 = CBR(128, 256)  
 self.enc4 = CBR(256, 512)  
 self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)  
  
 self.upconv4 = nn.ConvTranspose2d(512, 256, kernel\_size=2, stride=2)  
 self.dec4 = CBR(512, 256)  
 self.upconv3 = nn.ConvTranspose2d(256, 128, kernel\_size=2, stride=2)  
 self.dec3 = CBR(256, 128)  
 self.upconv2 = nn.ConvTranspose2d(128, 64, kernel\_size=2, stride=2)  
 self.dec2 = CBR(128, 64)  
 self.dec1 = nn.Conv2d(64, out\_channels, kernel\_size=1)  
  
 def forward(self, x):  
 enc1 = self.enc1(x)  
 enc2 = self.enc2(self.pool(enc1))  
 enc3 = self.enc3(self.pool(enc2))  
 enc4 = self.enc4(self.pool(enc3))  
  
 dec4 = self.upconv4(enc4)  
 dec4 = torch.cat((dec4, enc3), dim=1)  
 dec4 = self.dec4(dec4)  
 dec3 = self.upconv3(dec4)  
 dec3 = torch.cat((dec3, enc2), dim=1)  
 dec3 = self.dec3(dec3)  
 dec2 = self.upconv2(dec3)  
 dec2 = torch.cat((dec2, enc1), dim=1)  
 dec2 = self.dec2(dec2)  
 dec1 = self.dec1(dec2)  
  
 return torch.sigmoid(dec1)  
  
  
*# Load the trained model*model = UNet().cpu()  
model.load\_state\_dict(torch.load(MODEL\_PATH))  
model.eval()  
  
*# Transformations*transform = transforms.Compose([  
 transforms.Resize((256, 256)),  
 transforms.ToTensor()  
])  
  
  
def remove\_glare(image):  
 *# Load and transform the image*

image = Image.fromarray(image)image = image.convert('RGB')  
 input\_tensor = transform(image).unsqueeze(0).cpu() *# Add batch dimension  
  
 # Inference* with torch.no\_grad():  
 output\_tensor = model(input\_tensor)  
  
 *# Convert tensor to image* output\_image = transforms.ToPILImage()(output\_tensor.squeeze(0))  
  
 *# Save the output image* return output\_image

## Приложение Б – программный код клиентской части

### Файл src/components/PageWrapper.vue

<template>  
 <q-layout>  
 <q-page-container>  
 <slot />  
 </q-page-container>  
 </q-layout>  
</template>

### Файл src/pages/ErrorNotFound.vue

<template>  
 <div class="fullscreen bg-blue text-white text-center q-pa-md flex flex-center">  
 <div>  
 <div style="font-size: 30vh">  
 404  
 </div>  
  
 <div class="text-h2" style="opacity:.4">  
 Oops. Nothing here...  
 </div>  
  
 <q-btn  
 class="q-mt-xl"  
 color="white"  
 text-color="blue"  
 unelevated  
 to="/"  
 label="Go Home"  
 no-caps  
 />  
 </div>  
 </div>  
</template>  
  
<script setup lang="ts">  
*defineOptions*({  
 name: 'ErrorNotFound'  
});  
</script>

### Файл src/pages/IndexPage.vue

<template>  
 <PageWrapper>  
  
 <q-page class="main-page">  
 <div class="upload">  
 <h3 style="margin-bottom: 0">AI image enhancer</h3>  
 <h5 style="margin-top: 1em">Apply various effects to an image using AI</h5>  
 <q-file  
 v-model="uploadedImage"  
 standout  
 rounded  
 class="file-input"  
 bg-color="red"  
 label="Upload files"  
 label-color="white"  
 >  
 </q-file>  
  
 <span style="opacity: 0.6">Images must be exactly 256 by 256 pixels</span>  
  
 <div  
 v-if="uploadedImageObjectUrl"  
 style="display: flex; flex-direction: row; align-items: center; justify-content: center; gap: 80px ">  
 <q-img class="image-to-upload" :src="uploadedImageObjectUrl"></q-img>  
 <div style="display: flex; flex-direction: column; gap: 20px">  
 <h5 style="margin: 0">Enhancing params</h5>  
 <div>  
 <q-checkbox v-model="colorize">  
 Colorize  
 <q-tooltip anchor="center right" self="center left">  
 Colorize monochromatic image  
 </q-tooltip>  
 </q-checkbox>  
 </div>  
  
 <div>  
 <q-checkbox v-model="sharpen">  
 Sharpen  
 <q-tooltip anchor="center right" self="center left">  
 Sharpen image  
 </q-tooltip>  
 </q-checkbox>  
 </div>  
  
 <div>  
 <q-checkbox v-model="removeGlare">  
 Remove glare  
 <q-tooltip anchor="center right" self="center left">  
 Remove glare defects from your image  
 </q-tooltip>  
 </q-checkbox>  
 </div>  
 </div>  
 </div>  
  
 <q-btn  
 v-if="uploadedImageObjectUrl"  
 :disabled="!anyCheckboxSelected()"  
 color="red"  
 style="margin-top: 16px"  
 @click="submit"  
 >  
 Submit  
 </q-btn>  
 <span v-if="uploadedImageObjectUrl && !anyCheckboxSelected()" style="color:red;">Please choose at least one enhancement parameter</span>  
 </div>  
 </q-page>  
 </PageWrapper>  
</template>  
  
<script setup lang="ts">  
  
import { *ref*, *watch* } from 'vue';  
import { *useQuasar* } from 'quasar';  
import PageWrapper from 'components/PageWrapper.vue';  
import axios from 'axios';  
import { *useRouter* } from 'vue-router';  
import { *useStore* } from 'vuex';  
  
const allowedTypes = ['image/jpg', 'image/jpeg', 'image/png'];  
  
const uploadedImage = *ref*<File | null>(null);  
const uploadedImageObjectUrl = *ref*<string | undefined>(undefined);  
  
const colorize = *ref*(false);  
const sharpen = *ref*(false);  
const removeGlare = *ref*(false);  
  
const $q = *useQuasar*();  
const router = *useRouter*();  
const store = *useStore*();  
  
const anyCheckboxSelected = (): boolean => {  
 return colorize.value || sharpen.value || removeGlare.value;  
};  
  
*watch*(uploadedImage, (newValue) => {  
 if (newValue) {  
 if (allowedTypes.find((type) => type === newValue?.type) === undefined) {  
 $q.notify({  
 type: 'negative',  
 message: 'Uploaded file type is not supported'  
 });  
 uploadedImage.value = null;  
 throw new Error('Uploaded file type is not supported');  
  
 }  
  
 const objectUrl = URL.createObjectURL(newValue);  
  
 let img = new *Image*();  
 img.src = objectUrl;  
  
 img.onload = () => {  
 if (img.height > 256 || img.width > 256) {  
 $q.notify({  
 type: 'negative',  
 message: 'Image must be exactly 256 by 256 pixels',  
 caption: `Provided image is ${img.height} by ${img.width}`  
 });  
 uploadedImage.value = null;  
 throw new Error('Image must be exactly 256 by 256 pixels');  
 }  
 uploadedImageObjectUrl.value = objectUrl;  
  
 store.commit('setOldImage', newValue);  
 };  
 }  
});  
  
const submit = () => {  
 let data = new *FormData*();  
 if (!uploadedImage.value) {  
 return;  
 }  
  
 if (!anyCheckboxSelected()) {  
 $q.notify({  
 type: 'negative',  
 message: 'At least one parameter must be selected'  
 });  
 throw new Error('At least one parameter must be selected');  
 }  
 data.append('file', uploadedImage.value);  
 data.append('colorize', colorize.value.toString());  
 data.append('sharpen', sharpen.value.toString());  
 data.append('removeGlare', removeGlare.value.toString());  
  
  
 $q.loading.show();  
 axios.post('upload', data, {  
 headers: {  
 'accept': 'application/json',  
 'Accept-Language': 'en-US,en;q=0.8',  
 'Content-Type': 'multipart/form-data'  
 },  
 responseType: 'blob'  
 })  
 .then(async (response) => {  
 $q.loading.hide();  
 const newImage = new *File*([response.data], 'result.jpg', {  
 type: 'image/jpeg'  
 });  
 store.commit('setNewImage', newImage);  
 await router.push('/result');  
 }).catch((error) => {  
 $q.loading.hide();  
 $q.notify({  
 type: 'negative',  
 message: 'Internal error: ' + error.message  
 });  
 });  
};  
  
*defineOptions*({  
 name: 'IndexPage'  
});  
</script>  
  
<style scoped lang="scss">  
  
  
.main-page {  
 display: flex;  
 justify-content: center;  
 flex-direction: row;  
}  
  
.drawer-content {  
 padding: 8px;  
}  
  
.slider-container {  
 margin-top: 10px;  
}  
  
.file-input {  
 width: 400px  
}  
  
.upload {  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 position: relative;  
}  
  
.image-to-upload {  
 margin-top: 16px;  
 border: 1px solid black;  
 height: auto;  
 width: 18vw;  
}  
  
h5 {  
 font-size: max(0.8em, 1vw);  
}  
</style>

### Файл src/pages/resultPage.vue

<script setup lang="ts">  
import { *onMounted*, *ref* } from 'vue';  
import { *useRouter* } from 'vue-router';  
import store from 'src/store';  
import PageWrapper from 'components/PageWrapper.vue';  
  
const router = *useRouter*();  
const oldImageSrc = *ref*<string | undefined>(undefined);  
const newImageSrc = *ref*<string | undefined>(undefined);  
  
  
*onMounted*(() => {  
 if(store.state.oldImage && store.state.newImage) {  
 oldImageSrc.value = URL.createObjectURL(store.state.oldImage);  
 newImageSrc.value = URL.createObjectURL(store.state.newImage);  
 } else {  
 router.push('/');  
 }  
});  
  
const downloadClick = () => {  
 if(store.state.newImage) {  
 const blob = new *Blob*([store.state.newImage], { type: 'image/jpeg' });  
 const url = URL.createObjectURL(blob);  
 const a = document.createElement('a');  
 a.href = url;  
  
 a.download = getFileName(store.state.oldImage?.name) + '\_result.jpg';  
 document.body.appendChild(a);  
 a.click();  
 document.body.removeChild(a);  
 URL.revokeObjectURL(url);  
 }  
}  
  
const getFileName = (fileName: string | undefined) => {  
 if (fileName) {  
 return fileName.replace(/\.[^/.]+$/, '');  
 }  
 return ''  
}  
  
*defineOptions*({  
 name: 'ResultPage'  
});  
</script>  
  
<template>  
 <PageWrapper :drawer-open = false>  
 <q-page>  
 <div class="result">  
 <h5>See the result</h5>  
 <div style="display: flex; flex-direction: row; align-items: center;">  
 <div style="display: flex; flex-direction: column; align-items: center; margin-right: 1vw">  
 <q-img class="image" :src="oldImageSrc"></q-img>  
 <p>Before</p>  
 </div>  
 <div style="display: flex; flex-direction: column; align-items: center; margin-left: 1vw">  
 <q-img class="image" :src="newImageSrc"></q-img>  
 <p>After</p>  
 </div>  
 </div>  
 <div style="display: flex; flex-direction: row; align-items: center; gap: 20px">  
 <q-btn color="red" @click="() => {router.push('/')}">Go back</q-btn>  
 <q-btn color="red" @click="downloadClick">Download the result</q-btn>  
 </div>  
 </div>  
 </q-page>  
 </PageWrapper>  
</template>  
  
<style scoped lang="scss">  
.result {  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 position: relative;  
}  
  
.image {  
 margin-top: 16px;  
 border: 1px solid black;  
 height: auto;  
 width: 18vw;  
}  
  
p {  
 font-size: max(0.8em, 0.5vw);  
}  
</style>

### Файл src/router/index.ts

import { *route* } from 'quasar/wrappers';  
import {  
 *createMemoryHistory*,  
 *createRouter*,  
 *createWebHashHistory*,  
 *createWebHistory*,  
} from 'vue-router';  
  
import routes from './routes';  
export default *route*(function (*/\* { store, ssrContext } \*/*) {  
 const createHistory = process.env.SERVER  
 ? *createMemoryHistory* : (process.env.VUE\_ROUTER\_MODE === 'history' ? *createWebHistory* : *createWebHashHistory*);  
  
 const Router = *createRouter*({  
 scrollBehavior: () => ({ left: 0, top: 0 }),  
 routes,  
history: createHistory(process.env.VUE\_ROUTER\_BASE),  
 });  
  
 return Router;  
});

### Файл src/router/routes.ts

import { RouteRecordRaw } from 'vue-router';  
  
const routes: RouteRecordRaw[] = [  
 {  
 path: '/',  
 component: () => import('pages/IndexPage.vue'),  
 },  
 {  
 path: '/result',  
 component: () => import('pages/resultPage.vue'),  
 },  
 {  
 path: '/:catchAll(.\*)\*',  
 component: () => import('pages/ErrorNotFound.vue'),  
 },  
];  
  
export default routes;

### Файл src/store/index.ts

import Mutations from './mutations';  
import type { InjectionKey } from 'vue';  
import { State, StateInterface } from './state';  
import { *createStore*, *useStore* as baseUseStore, Store } from 'vuex';  
  
export type StoreType = Store<StateInterface>;  
  
export const key: InjectionKey<StoreType> = *Symbol*();  
  
const store = *createStore*<StateInterface>({  
 mutations: new Mutations(),  
 state: new State(),  
});  
  
export function *useStore*(): StoreType {  
 return *baseUseStore*(key);  
}  
  
export default store;

### Файл src/store/mutations.ts

import { Mutation, MutationTree } from 'vuex';  
import { StateInterface } from './state';  
  
interface MutationsInterface extends MutationTree<StateInterface> {  
 setOldImage: Mutation<StateInterface> &  
 ((state: StateInterface, payload: File) => void);  
  
 setNewImage: Mutation<StateInterface> &  
 ((state: StateInterface, payload: File) => void);  
}  
  
export default class Mutations implements MutationsInterface {  
 setNewImage = (state: StateInterface, payload: File) => {  
 state.newImage = payload;  
 };  
  
 setOldImage = (state: StateInterface, payload: File) => {  
 state.oldImage = payload;  
 };  
  
 [key: string]: Mutation<StateInterface>;  
  
};

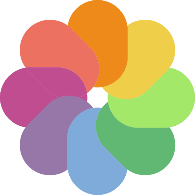
### Файл src/store/state.ts

export interface StateInterface {  
 newImage: File | null,  
 oldImage: File | null,  
}  
  
export class State implements StateInterface {  
 newImage: File | null = null;  
 oldImage: File | null = null;  
}  
  
export default State;

### Файл src/App.vue

<template>  
 <router-view />  
</template>  
  
<script setup lang="ts">  
*defineOptions*({  
 name: 'App'  
});  
</script>

## Приложение В – иконка сайта



# Список использованной литературы

1. Ж. Орельен Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow. - Москва: Вильямс, 2018. - 688 с.
2. Sebastian Raschka, Vahid Mirjalili Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. - 3-е изд. - United Kingdom: Packt Publishing, 2019. - 770 с.
3. Раскрашиваем чёрно-белую фотографию с помощью нейросети из 100 строк кода // Хабр URL: https://habr.com/ru/companies/nix/articles/342388/ (дата обращения: 20.05.2024).
4. Сверточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество // Habr URL: https://habr.com/ru/articles/348000/ (дата обращения: 05.04.2024).
5. OpenCV в Python. Часть 1 // Habr URL: https://habr.com/ru/articles/519454/ (дата обращения: 14.04.2024).
6. TensorFlow для начинающих. Часть 1: общие сведения, установка библиотеки // Habr URL: https://habr.com/ru/companies/ruvds/articles/450314/ (дата обращения: 15.04.2024).
7. Реставрация и колоризация старых фотографий с использованием нейронных сетей // CyberLeninka URL: https://cyberleninka.ru/article/n/restavratsiya-i-kolorizatsiya-staryh-fotografiy-s-ispolzovaniem-neyronnyh-setey/viewer (дата обращения: 31.05.2024).
8. Vladimir Iglovikov (ternaus), Alexey Shvets U-Net для семантической сегментации. //GitHub URL:
9. TernausNet // Github URL: https://github.com/ternaus/TernausNet/ (дата обращения: 04.05.2024).
10. Т. Ганегедара Обработка естественного языка с TensorFlow, - Москва: ДМК Пресс, 2020. - С. 132-154.