МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра дискретной математики и информационных технологий

# РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ АВТОМАТИЧЕСКОГО ОБНАРУЖЕНИЯ И КОРРЕКЦИИ ШУМА НА ИЗОБРАЖЕНИЯХ С ИСПОЛЬЗОВАНИЕМ МАШИННОГО ОБУЧЕНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студентки 4 курса 421 группы
направления 09.03.01 — Информатика и вычислительная техника
факультета КНиИТ
Тугушевой Айгуль Алиевны

Научный руководитель
д. э. н., профессор _____ Л. В. Кальянов

Заведующий кафедрой
доцент, к. ф.-м. н. _____ Л. Б. Тяпаев

Саратов 2025

# ВВЕДЕНИЕ

В условиях стремительного развития цифровых технологий и распространения визуальных данных задача обеспечения высокого качества изображений приобретает особую значимость. Шум, возникающий при съёмке, передаче или хранении изображений, значительно снижает их информативность и может привести к ошибкам в последующей обработке.

Современные методы обработки изображений всё чаще опираются на алгоритмы машинного обучения, в частности — на нейронные сети. Одной из таких моделей является DnCNN (Denoising Convolutional Neural Network), показавшая высокие результаты при восстановлении изображений, и адаптируемая под различные типы искажений.

Целью данной бакалаврской работы является разработка веб-приложения для автоматического удаления шума на изображениях с использованием модели DnCNN. В отличие от классических исследований, ориентированных только на архитектуру модели, в работе акцент сделан на практическую реализацию программного решения, доступного конечному пользователю.

Для достижения поставленной цели были сформулированы следующие задачи:

— провести обзор типов шумов;
— проанализировать архитектуру модели DnCNN;
— сформировать и подготовить обучающую выборку изображений;
— реализовать и обучить модель глубокого обучения с кастомным слоем;
— разработать веб-приложение для работы с интегрированной обученной моделью.

## КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

**В первом разделе** представлено описание аналитических процедур, связанных с моделированием шумов на изображениях и подготовкой обучающих данных для последующей нейросетевой обработки. Рассматриваются основные типы искажений, типичные для изображений, получаемых с помощью цифровых сенсоров, а также методы искусственной генерации шумов, обеспечивающих разнообразие обучающих примеров и реалистичность условий, приближённых к реальным.

Цифровые изображения в современных системах машинного зрения часто подвержены искажениям, возникающим в результате работы аппаратных компонентов или условий окружающей среды. Наличие шумов снижает визуальное качество изображений и может критически повлиять на точность выполнения задач распознавания и анализа. В связи с этим важной частью процесса обучения моделей подавления шума является генерация зашумлённых данных, отражающих реальные сценарии искажения. Это необходимо для повышения обобщающей способности нейросети и её устойчивости к различным видам артефактов.

В рассматриваемой работе применён подход комбинированного добавления шумов, включающий три различных типа: гауссовский, импульсный (*salt-and-pepper*) и локализованный градиентный шум. Такое сочетание обеспечивает наличие как непрерывных, так и дискретных искажений, воспроизводя широкий спектр возможных нарушений качества изображений. Каждый из типов шума был реализован с помощью программных библиотек `NumPy` и `scikit-image`, что позволило достичь высокой гибкости и воспроизводимости при генерации данных.

Гауссовский шум моделирует тепловые искажения и нестабильность освещения, добавляя к каждому пикселю значение, выбранное из нормального распределения с нулевым средним и случайной дисперсией. Импульсный шум, напротив, проявляется как отдельные пиксели, заменённые на белые или чёрные точки — характерная проблема, возникающая из-за ошибок передачи или неисправностей сенсора. Локализованный градиентный шум имитирует засветы и локальные повреждения, возникающие, например, при попадании постороннего объекта в объектив или при локальных перегревах.

Особенность используемого подхода заключается в вероятностном на-

ложении всех трёх типов шумов на одно изображение, с вариацией параметров, таких как интенсивность, радиус локального шума и плотность импульсного шума. Локализованный шум добавляется не всегда — только с вероятностью 50%, что повышает разнообразие набора и исключает переобучение на однотипных примерах. Благодаря такой методике создаются данные, способные повысить устойчивость нейросети к нештатным ситуациям.

На этапе подготовки данных формируется обучающая выборка в виде пар: зашумлённое и соответствующее ему чистое изображение. Перед добавлением шумов все изображения нормализуются в диапазон $[0, 1]$, что является стандартной практикой при работе с нейросетями и способствует стабильности обучения. После наложения шумов производится обрезка изображений на патчи размером $64 \times 64$ пикселя с перекрытием 50%. Такой приём увеличивает число обучающих примеров и позволяет нейросети обрабатывать как локальные, так и глобальные искажения.

Также внимание уделяется корректности значений пикселей после наложения шумов — применяется функция `np.clip`, ограничивающая диапазон значений, что предотвращает появление невалидных данных, способных привести к искажению результатов.

Разработанный подход обеспечивает формирование репрезентативной выборки, имитирующей реальное распределение шумов и позволяющей эффективно обучать модели восстановления изображений. Используемые техники генерации и сегментации изображений делают возможным последующее применение разработанных алгоритмов в условиях, близких к эксплуатационным, что особенно важно для задач в области компьютерного зрения, медицинской диагностики и промышленной инспекции.

**Во втором разделе** представлена архитектура и процесс обучения нейросетевой модели, предназначенной для устранения шума на изображениях. Основное внимание уделяется использованию модифицированной версии свёрточной нейронной сети DnCNN, которая показала высокую эффективность в задаче восстановления изображений. DnCNN отличается отсутствием слоёв пулинга, что позволяет сохранять пространственное разрешение, а также архитектурой, ориентированной на предсказание остаточного шума, что упрощает регрессионную задачу.

Ключевым отличием представленной реализации является добавление

пользовательского слоя `Negative`, реализующего поэлементное изменение знака значений, что позволило улучшить подавление остаточных искажений. Модель реализована с помощью API Keras и фреймворка TensorFlow, что обеспечивает удобство и гибкость разработки. Архитектура модели включает один входной слой, пятнадцать повторяющихся блоков (состоящих из слоёв `Conv2D`, `BatchNormalization` и `ReLU`), а также выходной и пользовательский слой.

Для обучения была использована синтетическая выборка, полученная путём искажения изображений из датасета BSDS500 комбинированным шумом (гауссовским, импульсным и локализованным градиентным). Это позволило реалистично имитировать условия, в которых возникает шум, например, при передаче по каналам связи или при съёмке в неблагоприятных условиях. В качестве функции потерь применялась среднеквадратичная ошибка (MSE), а оптимизация проводилась с использованием алгоритма Adam с начальной скоростью обучения $1 \cdot 10^{-3}$. Были также использованы обратные вызовы `ModelCheckpoint` и `ReduceLROnPlateau`, что способствовало стабильной и быстрой сходимости модели.

Обучение осуществлялось в среде Google Colab с использованием GPU и продолжалось 100 эпох, занимая около 12 часов. Прогресс обучения отслеживался с помощью логов и графиков функции потерь, которые показали стабильное снижение ошибок и отсутствие признаков переобучения.

Модель, изначально обученная на патчах $64 \times 64$, была адаптирована для обработки изображений произвольного размера с помощью стратегии скользящего окна с перекрытием. Такой подход предотвращал появление артефактов на границах и обеспечивал высокое качество восстановления. Реализация данного алгоритма приведена в виде Python-функции, обрабатывающей изображение фрагментами и усредняющей перекрывающиеся области.

После завершения обучения модель была сохранена и использована для инференса. Её эффективность была подтверждена с помощью метрик PSNR и SSIM, значения которых на валидационном наборе составили от 28 до 31 дБ и от 0,85 до 0,92 соответственно. Кроме того, были приведены примеры визуального сравнения оригинальных, зашумлённых и восстановленных изображений, иллюстрирующие эффективность работы модели.

Для демонстрации практической применимости разработанного мето-

да модель была протестирована на реальном астрономическом изображении Hubble Deep Field, известном своей научной ценностью. Несмотря на наличие значительных искажений на оригинале, модель успешно устранила шум, сохранив ключевые структурные элементы. Это подтверждает, что разработанная нейросеть способна обрабатывать как синтетические, так и реальные сложные изображения, эффективно восстанавливая их качество.

**В третьем разделе** рассматривается архитектура разработанной системы, предназначенной для интеграции модели DnCNN в виде полноценно функционирующего веб-приложения. Основное внимание уделено построению клиент-серверной архитектуры, обеспечивающей взаимодействие между пользовательским интерфейсом, серверной логикой и моделью машинного обучения. Переход от исследовательского прототипа к прикладному решению реализован с учетом требований гибкости, масштабируемости и удобства использования.

Система построена по принципу трёхкомпонентной архитектуры, включающей клиент (фронтенд), сервер (бэкенд) и модель машинного обучения. Клиентская часть реализована с использованием JavaScript-библиотеки `React` и отвечает за интерфейс взаимодействия с пользователем: загрузку изображений, отображение результатов, а также отправку запросов к серверу. Вся передача данных осуществляется с помощью REST API, что позволяет достичь независимости между клиентом и сервером и обеспечить их масштабируемость.

Серверная часть, реализованная на Python с использованием микрофреймворка `Flask`, обрабатывает HTTP-запросы, выполняет декодирование и предобработку изображений, взаимодействует с моделью и формирует ответы. Для повышения эффективности модель загружается один раз при инициализации и используется повторно при каждом запросе. Это позволяет существенно снизить время отклика системы.

Модель DnCNN, реализованная с использованием `Keras` и `TensorFlow`, принимает на вход изображения, очищает их от шума и возвращает восстановленный результат. В серверной части реализована поддержка кастомных слоёв модели, таких как `Negative`, что позволяет использовать гибко настроенные архитектуры нейросетей.

Описывается полный цикл взаимодействия между компонентами: от

инициализации веб-интерфейса, загрузки изображения, формирования и отправки HTTP POST-запроса, до обработки на сервере, инференса модели и возврата результата клиенту. Клиент получает обработанное изображение, которое декодируется и отображается в интерфейсе для сравнения с оригиналом.

В разделе также уделено внимание безопасности системы. Реализованы проверки формата и размера загружаемых изображений, ограничение количества запросов для предотвращения атак типа DoS, а также возможность расширения проверки содержимого изображений на наличие вредоносного кода.

Подробно описаны преимущества клиент-серверной архитектуры: масштабируемость, независимость компонентов, упрощённое тестирование и сопровождение, повторное использование кода, а также гибкость развёртывания — как на локальной машине, так и в облаке. Благодаря архитектуре REST API и соблюдению принципов stateless взаимодействия, система легко масштабируется горизонтально и адаптируется под различные условия эксплуатации.

Отдельный акцент сделан на гибкости и расширяемости системы. Возможность изолированной замены модели, адаптивность интерфейса, использование контейнеризации (например, Docker) и развёртывание в облачных сервисах демонстрируют готовность архитектуры к будущим доработкам. В перспективе возможна интеграция дополнительных моделей, новых алгоритмов предобработки и настройки параметров обработки изображений, а также расширение пользовательского интерфейса.

**В четвёртом разделе** описана разработка клиентской части веб-приложения для взаимодействия пользователя с моделью DnCNN. Здесь подробно изложены этапы создания фронтенда с использованием JavaScript-библиотеки React, описание структуры проекта и основных компонентов, механики загрузки, валидации и отображения изображений, а также особенности пользовательского интерфейса и его отладки.

Сначала показано, как инициализировался React-проект при помощи `create-react-app` и организована структура папок: `public/` для статических ресурсов и `src/` для исходного кода. В корневых файлах `App.js` и `index.js` реализуются ключевые для SPA-приложения функции. Затем подробно опи-

сан интерфейс: форма загрузки с элементом `<input type="file»`, кнопка обработки и блоки для отображения исходного и восстановленного изображения, что обеспечивает наглядность результата для пользователя.

Далее рассмотрена логика работы компонентов: загрузка файла, валидация типа и размера (до 5 МБ, только JPEG/PNG), отображение предварительного просмотра через `URL.createObjectURL()`, формирование `FormData`, отправка POST-запроса на `/api/denoise` при помощи axios и получение результата в виде `Blob`. При этом интерфейс реагирует на состояния: показывает индикатор загрузки, выводит сообщения об ошибках — будь то неподдерживаемый формат, слишком большой файл или сетевые проблемы.

Особое внимание уделено архитектуре компонентов: `ImageUploader.js` отвечает за выбор и проверку файла, `App.js` — за управление состоянием и связь с сервером, `ResultViewer.js` — за отображение сравнения «до и после». Предусмотрены адаптивная верстка на базе CSS-модулей и Flexbox, стилизованная разметка и плавные анимации при загрузке изображений и отображении результатов.

В разделе также описаны аспекты отладки и обеспечения совместимости: настройка proxy в `package.json` для обхода CORS, запуск фронтенда и бэкенда на разных портах в процессе разработки, использование инструментов разработчика для отслеживания запросов и ошибок. Представлены фрагменты кода, демонстрирующие ключевые механизмы: проверка MIME-типа, ограничение размера, отправка запроса, обработка ошибок и управление состоянием загрузки.

**В пятом разделе** представлена серверная часть разрабатываемого веб-приложения, реализованная с использованием микрофреймворка Flask. Этот фреймворк выбран за его лёгкость, гибкость и широкую поддержку внешних библиотек, что делает его особенно подходящим для задач быстрой интеграции моделей машинного обучения в веб-сервисы. Раздел подробно описывает архитектуру и функциональность backend-компонента системы, реализующей удаление шума с изображений с помощью модели DnCNN.

Главная задача серверной части заключается в приёме изображений от клиента, их обработке с использованием нейросети и возврате результата обратно пользователю. На первом этапе сервер получает изображение, отправленное пользователем через POST-запрос. Для корректной обработ-

ки используется формат `multipart/form-data`, который позволяет отправлять бинарные данные. Принятое изображение затем декодируется с помощью библиотеки Pillow, которая преобразует поток в объект изображения в формате RGB — стандарте, ожидаемом моделью DnCNN.

Далее изображение проходит предобработку, включающую масштабирование значений пикселей в диапазон $[0, 1]$, изменение размера до необходимого (например, 256x256 пикселей) и преобразование к нужной размерности входного тензора. После этого подготовленное изображение передаётся в модель, которая предсказывает шум, присутствующий на изображении. Результат работы модели вычитается из исходного изображения, и полученное очищенное изображение подвергается постобработке: нормализуется в диапазон $[0, 255]$, преобразуется обратно в изображение и сохраняется в формате PNG.

Маршрутизация запросов осуществляется через основной эндпоинт `/api/denoise`, который обрабатывает входящие изображения и выполняет полный цикл взаимодействия с моделью. В разделе представлена структура проекта, включающая основные файлы и директории, такие как `app.py`, `model/weights`, и модули с кастомными слоями и вспомогательными функциями. Важной особенностью является использование кастомного слоя `Negative`, необходимого для корректной работы модели, который регистрируется при загрузке модели с помощью механизма `get_custom_objects`.

Особое внимание уделено функции `denoise_image_fullsize`, реализующей предсказание на всём изображении путём его разбиения на патчи с последующим объединением результатов. Это позволяет обрабатывать изображения произвольного размера, что повышает универсальность решения. Также подробно описана реализация возврата обработанного изображения клиенту через `send_file`, используя буферизацию потока в `io.BytesIO`, что исключает необходимость сохранять файл на диск.

Кроме основной логики, раздел затрагивает аспекты обеспечения надёжности и совместимости: реализация базовой обработки ошибок (например, при отсутствии изображения в запросе), поддержка CORS для взаимодействия с клиентами, расположенными на других доменах или портах, а также запуск и логирование сервера с использованием встроенных средств Flask.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы решены следующие задачи:

— Разработана и обучена модель DnCNN с кастомным слоем `Negative`, демонстрирующая высокую эффективность в устранении комбинированного шума

— Создано веб-приложение с клиент-серверной архитектурой, обеспечивающее удобный интерфейс для обработки изображений

— Реализован механизм обработки изображений произвольного размера методом скользящего окна

— Проведена успешная проверка модели на реальных изображениях, включая астрономические снимки Hubble Deep Field

Разработанное решение может быть использовано в различных областях, требующих обработки изображений: медицине, системах видеонаблюдения, спутниковом мониторинге. Перспективы развития включают расширение типов обрабатываемых шумов и интеграцию дополнительных метрик оценки качества.

**Основные источники информации**:


1. He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition // Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. P. 770–778.
2. Simonyan, K., Zisserman, A. Very deep convolutional networks for large-scale image recognition // arXiv preprint arXiv:1409.1556. 2014.
3. Berkeley Segmentation Dataset 500 (BSDS500) // Kaggle. URL: https://www.kaggle.com/datasets/balraj98/berkeley-segmentation-dataset-500-bsds500
4. Flask documentation. URL: https://flask.palletsprojects.com/
5. React documentation. URL: https://react.dev/