

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

НЕЙРОСЕТЕВЫЕ АЛГОРИТМЫ
УЛУЧШЕНИЯ КАЧЕСТВА ИЗОБРАЖЕНИЙ

Студент

 17.06.24 Кривоzubов Антон Константинович
(подпись, дата)

Руководитель

 17.06.24 Слепцов Николай Владимирович
(подпись, дата)

Нормоконтролёр

 17.06.24 Бычков Андрей Станиславович
(подпись, дата)

Рецензент

наг. упр. Пензенского гос. ун-та
17.06.24  Афонин А.Ю.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 № 11)

Заведующий кафедрой

 М. А. Митрохин
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

18.06.2024 № 31)

Секретарь ГЭК

 Е.И. Гурин
(подпись)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

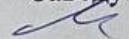
вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

 М.А. Митрохин

«13» сентября 2022 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Кривозубов Антон Константинович гр. 22ВВИм1 факультета вычислительной техники, направления 09.04.01.

2. Тема работы «Нейросетевые алгоритмы улучшения качества изображений»

Тема утверждена приказом ПГУ №765/о от "13" сентября 2022 г.

3. Руководитель магистерской диссертации к.т.н., доцент кафедры ВТ ПГУ Слепцов Н. В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

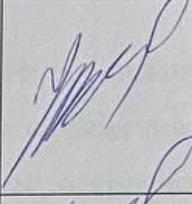
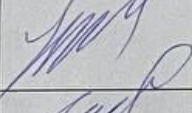
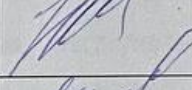
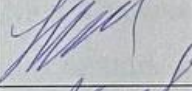
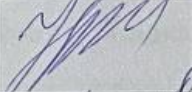
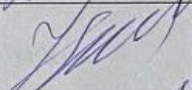
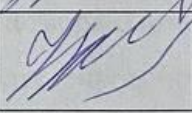
Назначение разработки – иллюстрация нейросетевого подхода к решению задачи обработки и улучшения качества изображений, а также создание нейросетевой модели направленной на увеличение разрешения изображений.

Исследование существующих нейросетевых моделей и алгоритмов, предназначенных для работы с изображениями и улучшение их качества, а также разработка нейросетевой модели и приложения, направленных на улучшение качества изображений. Показать применимость разработанных методов и средств на конкретных примерах.

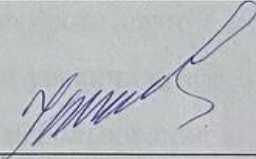
5. Перечень подлежащих разработке вопросов

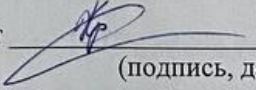
Требуется произвести анализ и исследование существующих нейросетевых алгоритмов и моделей, применяемых для обработки изображений и направленных на улучшение их качества, обзор существующих и общепринятых критериев качества изображений, изучить классические алгоритмы увеличения разрешения изображений, выделить различия с нейросетевыми методами, выбрать технологии, применяемые при создании и обучении нейросетевых моделей, разработать нейросетевую модель для увеличения разрешения изображения, обучить модель на доступных данных, протестировать и сравнить полученную модель с существующими, разработать и протестировать приложение, включающее в себя нейросетевую модель.

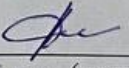
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ и исследование предметной области	15 %	13.09.2022-16.12.2022	
2.	Анализ и сравнение классических и нейросетевых алгоритмов, направленных на улучшение качества изображений	10 %	17.12.2022-18.02.2023	
3.	Разработка нейросетевой модели для увеличения разрешения изображения	20 %	19.02.2023-31.05.2023	
4.	Обучение модели для увеличения разрешения изображения	15 %	1.06.2023-27.08.2023	
5.	Тестирование и сравнение полученной модели с аналогами	15 %	28.08.2023-31.11.2023	
6.	Разработка функционала приложения	10 %	01.12.2023-31.01.2024	
7.	Тестирование и отладка разработанного приложения	5 %	01.02.2024-04.04.2024	
8.	Оформление пояснительной записки	10 %	05.04.24 - 31.05.24	

Дата выдачи задания "13" сентября 2022 г.

Руководитель магистерской диссертации  Слепцов Н.В.
(подпись, дата)

Задание к исполнению принял студент  14.09.2024 Кривоzubов А. К.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
12.06.2024 (подпись, дата)

РЕФЕРАТ

Отчет о магистерской диссертации содержит 109 листов, 42 рисунка, 29 таблиц, 28 источников

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, НЕЙРОННЫЕ СЕТИ, АЛГОРИТМЫ, КАЧЕСТВО ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ, GAN, ESRGAN, REAL-ESRGAN, SUPER-RESOLUTION

Цель работы сравнительный анализ и изучение нейросетевых алгоритмов для увеличения разрешения и качества изображений, исследование и выбор архитектуры в качестве основы для нейросетевого блока разрабатываемого приложения для обработки изображений.

В ходе выполнения магистерской работы были изучены критерии оценки качества обработанных изображений, сравнены и выделены преимущества и недостатки классических и нейросетевых подходов к задаче увеличения разрешения изображений, произведен сравнительный анализ и выбор архитектуры в качестве основы для нейросетевого блока разрабатываемого приложения. Разработана, обучена и протестирована нейросетевая модель, способная увеличивать разрешение изображений.

Результатом разработки является приложение, удовлетворяющий поставленной задаче. А именно, разработано, реализовано и протестировано приложение, включающее в себя нейросетевую модель, способную обрабатывать графические изображения и увеличивать его разрешение.

Разработка производилась с использованием широкого стека технологий, к которым относятся: язык программирования Python, интегрированная среда разработки PyCharm для работы со скриптами и фреймворк машинного обучения PyTorch для разработки нейросетевой модели.

					ПГУ 1.090401.12.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Кривоzubов А.К.		18.06.24	Нейросетевые алгоритмы улучшения качества изображений Пояснительная записка	Лит.	Лист	Листов
Провер.		Слепцов Н.В.		18.06.24			2	109
Н. Контр.		Бычков А.С.		18.06.24		ФВТ, каф. ВТ гр. 22ВВИМ1		
Утверд.		Митрохин М.А.		18.06.24				

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	3
ВВЕДЕНИЕ	4
1 Обзор и анализ существующих алгоритмов	7
1.1 Критерии качества изображений	7
1.2 Классические методы увеличения разрешения изображений	11
1.3 Нейросетевые методы увеличения разрешения изображений	17
1.4 Анализ инструментов для разработки	23
2 Модель увеличения разрешения изображений	29
2.1 Описание модели	29
2.2 Архитектура сети	30
2.3 Дискриминатор	34
2.4 Интерполяция	37
3 Реализация алгоритма	39
4 Обучение и тестирование модели	46
4.1 Детали обучения	46
4.2 Данные для обучения	47
4.3 Анализ результатов	49
4.4 Сравнение модели	52
4.5 Сравнение интерполяции	56
5 Проектирование и разработка	57
5.1 Структура приложения	57
5.2 Реализация интерфейса пользователя	57
5.3 Реализация функций предобработки	71
ЗАКЛЮЧЕНИЕ	73
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	75
Приложение А – Листинг программы	78
Приложение Б – Презентация	100

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

ИИ – искусственный интеллект

НС – нейронные сети

ЦОС – цифровая обработка изображений

CNN (Convolutional Neural Network) – свёрточная нейронная сеть

GAN (Generative Adversarial Network) – генеративно-сопоставительная сеть

HR (High Resolution) – изображение высокого разрешения

LR (Low Resolution) – изображение низкого разрешения

MOS (Mean Opinion Score) - средняя экспертная оценка

RB (Residual Block) – остаточный блок

RRDB (Residual in Residual Dense Block) – блок "остаток в остатке"

SR (Super Resolution) – суперразрешение

ВВЕДЕНИЕ

Актуальность темы. Методы повышения качества цифровых изображений направлены на улучшение восприятия изображения технической или художественной картины человеком. При этом понятие «повышение качества» связывается как с характером передаваемой зрителю посредством этого изображения информации, так и со спецификой человеческого зрения. В связи с этим обработка цифрового изображения, необходимая для улучшения его качества, в зависимости от указанных специфик может значительно варьироваться, чтобы соответствовать вытекающим из них весьма различным целям [1].

Одной из важных задач обработки изображений является улучшение качества изображения, которое включает в себя изменение его пространственного разрешения для улучшения возможности различения определенных деталей. Под пространственным разрешением обычно понимают количество пикселей, которыми изображение формируется. В некоторых случаях, например, для визуальной передачи простой информации, достаточно использовать изображения очень низкого разрешения, что обеспечит и высокую скорость ее восприятия. В других случаях, например, для восприятия большого количества мелких деталей картинки или схемы, требуется высокое пространственное разрешение цифрового изображения. Может оказаться, что пространственное разрешение, которое может обеспечить детектирующая матрица, недостаточно для получения требуемой детализации изображения. Это означает, что нужное цифровое изображение не может быть получено в течение одной экспозиции. В этом случае возникает задача его синтеза на основе группы изображений одной и той же картинки, имеющих более низкое разрешение, то есть так называемая задача «сверхразрешения».

Проблема получения изображения высокого разрешения исследуется в течение последних двадцати лет. Для решения этой задачи предложены различные классические подходы, которые можно классифицировать по

разным критериям. Но они не являются универсальными и, в большинстве случаев, необходимо человеческое вмешательство для определения перечня применяемых алгоритмов и их параметров [2].

В связи с этим в настоящий момент всё большую популярность набирает нейросетевой подход в обработке изображений. На текущий момент широко используются разнообразные алгоритмы на основе нейронных сетей, которые умеют принимать для обработки не только наборы чисел, но и объекты, имеющие некоторые дополнительные, главным образом геометрические, свойства, такие как изображения (такие алгоритмы учитывают не только значения пикселей, но и их взаимное расположение), аудио, видео и тексты. Они были обучены определять, что находится на картинке и, в случае возникновения необходимости, «дорисовывать» недостающие фрагменты. Нейронные сети используют различные методы обработки изображений, преследуя цель увеличить разрешение изображения. Обычно это проходит в несколько этапов и в конце представляется уже улучшенная картинка в качестве результата. Сфера применения данных технологий довольно обширна и программных продуктов, предоставляющих такие услуги, со временем становится только больше [3].

С учетом вышесказанного, тема исследования, связанная с разработкой нейросетевой модели направленной на увеличение разрешения изображения, является актуальной.

Цели и задачи исследования. Целью диссертационной работы является исследование нейросетевых алгоритмов и методов увеличения разрешения изображения и разработка приложения, включающего нейросетевую модель для задачи повышения качества изображения.

Для достижения поставленной цели необходимо решить следующие основные задачи:

- обзор и анализ существующих исследований и алгоритмов, применяемых и направленных на увеличение разрешения изображения, основанных на нейронных сетях,

- разработка нейросетевой модели для увеличения разрешения изображения,
- обучение модели для увеличения разрешения изображения,
- тестирование и сравнение полученной модели с аналогами,
- разработка функционала приложения,
- тестирование и отладка разработанного приложения.

Объектом исследования являются нейросетевые алгоритмы методы, увеличивающие разрешение изображений.

Предметом исследования является разработка, совершенствование и оптимизация нейросетевых алгоритмов, направленных на решение задачи увеличения разрешения изображений.

Методами исследования для решения задач в диссертационной работе являются методы интеллектуального анализа данных, методы обработки изображений, методы машинного обучения, классические и нейросетевые методы увеличения разрешения изображения.

Практическая ценность работы. Практические результаты, полученные в результате исследований:

- разработана, обучена нейросетевая модель на основе генеративно-сопоставительной сети, направленная на решение проблемы увеличения разрешения изображения,
- произведено тестирование и сравнение полученной нейросетевой модели с классическими методами и нейросетевыми аналогами, выявлены преимущества, недостатки и дальнейшие направления для улучшения сети,
- разработано приложение, включающее в себя нейросетевую модель, направленное на обработку поступающего на него изображения и увеличивающее его разрешение,
- показана применимость разработанной нейросетевой модели и приложения с использованием различных входных данных.

1 Обзор и анализ существующих алгоритмов

1.1 Критерии качества изображений

В настоящее время различные формы цифровой обработки визуальной и графической информации охватывают различные виды инфокоммуникационных приложений и расширяют их спектр использования. Сюда относятся как традиционные приложения (вещательное, промышленное, охранное телевидение, системы охраны), так и сравнительно новые приложения (видеоконференцсвязь, техническое зрение, цифровое кино, телевидение высокой четкости, 3D-телевидение) [4].

Алгоритмы обработки, сжатия и кодирования визуальной информации неизбежно вносят искажения различного типа. Качество цифрового изображения зависит от условий освещения, а также от характеристик регистрирующей системы: соотношения сигнал/шум, статистических характеристик шума, градационных характеристик, интервалов дискретизации спектральных (цветовых) характеристик и т.д. Оценка качества изображения является достаточно сложной комплексной задачей. Но невозможно говорить об улучшении качества изображений без определения критериев этого качества.

Под критериями качества изображения будем понимать специальные метрики, используемые для оценки качества работы системы, метода или алгоритма. В контексте обработки изображений критерии качества используются для оценки снимка до и после его обработки. Таким образом можно сравнивать различные методы обработки и выбирать наиболее эффективный подход. Критерии качества изображений могут включать в себя такие параметры, как резкость, контрастность, яркость, насыщенность цветов, шум, разрешение, цветовая гамма и др. Каждый из этих параметров может быть измерен определенным способом и использован для оценки качества обработки фото.

Очевидно, что для различных задач обработки могут использоваться разные критерии качества. Например, для задачи распознавания лиц на

изображении критерием качества может быть точность распознавания, а для задачи восстановления поврежденных изображений - сохранение деталей и минимизация искажений [5].

На текущий момент все методы оценки качества можно классифицировать на две большие категории:

- субъективные критерии оценки качества изображений. Такая оценка в данном случае осуществляется человеком,
- объективные критерии оценки качества изображений. Здесь оценка производится алгоритмически. Такой алгоритм может использовать оригинал изображения для оценки качества, то есть быть эталонным, или может определять качество без наличия неискаженной копии изображения, то есть быть неэталонными [6].

На рисунке 1 представлена классификация критериев оценки качества изображения:



Рисунок 1 – Критерии оценки качества изображения

Ниже перечислены и рассмотрены эталонные критерии качества, которые непосредственно используются и применяются в разработке и сравнении изображений.

Пиковое отношение сигнала к шуму (*PSNR*) - это один из наиболее распространенных критериев качества, используемых для оценки уровня

шума в изображении. $PSNR$ вычисляется путем сравнения оригинального изображения с улучшенным изображением и измерения разницы между ними.

Формула вычисления пикового отношения сигнала к шуму (1):

$$PSNR = 10\log\left(\frac{I_{max}^2}{MSE}\right), \quad (1)$$

где I – максимальное значение яркости, принимаемое пикселем изображения.

MSE – среднеквадратическая ошибка между двумя изображениями (2):

$$MSE = \frac{1}{n} \sum_{i=1}^n (I_{1,i} - I_{2,i})^2, \quad (2)$$

где n – общее количество пикселей в изображении,

I_1, I_2 – яркости пикселей зашумленного и оригинального изображения соответственно.

Основными практическими преимуществами использования данной эталонной оценки являются:

- отсутствие параметров и, как следствие, – простота расчета,
- физическая интерпретация в виде энергии сигнала ошибки,
- возможность использования приведенной оценки в оптимизационных задачах благодаря свойствам симметричности и дифференцируемости.

Этот критерий также устойчив к изменениям масштаба и поворотам изображений, что делает его более универсальным для различных задач [7].

Коэффициент структурного подобия ($SSIM$) – это критерий качества, который используется для измерения структурного сходства между оригинальным и улучшенным изображением, то есть метод проводит измерение качества на основе исходного изображения (не сжатого или без искажений). Он учитывает не только яркость и контрастность, но и текстуру и форму объектов на изображении. $SSIM$ -индекс является развитием традиционных методов, таких как $PSNR$ и метод среднеквадратичной ошибки MSE , которые оказались несовместимы с физиологией человеческого

восприятия. Отличительной особенностью метода, является то, что метод учитывает «восприятие ошибки» благодаря учёту структурного изменения информации. Формула вычисления коэффициента структурного подобия (3):

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (3)$$

где μ_x – среднее значение выборки пикселей x ,

μ_y – среднее значение выборки пикселей y ,

σ_x – дисперсия x ,

σ_y – дисперсия y ,

σ_{xy} – ковариация x и y ,

$c_1 = (k_1L)^2$, $c_2 = (k_2L)^2$ – две переменных для стабилизации деления,

L – динамический диапазон пикселей (обычно $2^{(\text{bits per pixel})} - 1$),

$k_1 = 0,01$ и $k_2 = 0,03$ – константы.

Коэффициент верности визуальной информации (*VIF*) - это критерий качества, который измеряет количество информации, передаваемой изображением. Он учитывает, как основные характеристики изображения, так и восприятие человеком, что делает его более точным при оценке качества изображения [8].

VIF получается из количественной оценки двух величин взаимной информации: взаимной информации между входом и выходом канала HVS, когда канал искажений отсутствует (называемой информацией опорного изображения), и взаимной информации между входом канала искажения, и выход канала HVS для тестового изображения [3]. Результат оценки представляется значением от 0 до 1. Формула вычисления коэффициента верности визуальной информации (4):

$$VIF = \frac{\sum_{j \in \text{subbands}} I(C^{N,j}; F^{N,j} | S^{N,j} = S^{N,j})}{\sum_{j \in \text{subbands}} I(C^{N,j}; E^{N,j} | S^{N,j} = S^{N,j})} \quad (4)$$

Индекс восприятия (PI) - это критерий качества изображения, основанный на оценке визуального восприятия изображения человеком. Он учитывает не только физические характеристики изображения, но и способность человека воспринимать и оценивать его [9].

PI принято считать более «человеческим» критерием, так как учитывает не только математические параметры изображения, но и человеческое восприятие его качества. Это позволяет получать более точные результаты при оценке качества изображений, особенно в случаях, когда PSNR может не справиться, например, с оценкой качества в условиях шума или искажений [10].

1.2 Классические методы увеличения разрешения изображений

Разрешение – это количество пикселей (точек) на единицу площади изображения. Он измеряется в мегапикселях или отображается в виде двух величин – высоты и ширины изображения. Интерполяция означает изменение размера и количества пикселей изображения, это может быть, как добавление, так и удаление общего количества пикселей. Интерполяция всегда приводит к ухудшению качества изображения. При уменьшающем виде интерполяции уменьшается размер файла и повышается резкость внешнего представления. При увеличивающем виде интерполяции увеличивается размер файла и размывается внешнее представление.

Интерполяция изображений происходит во всех цифровых фотографиях на определённом этапе, в независимости от того какая обработка происходит, дематризация или масштабирование. Она будет происходить всякий раз, когда пользователю необходимо изменить размер или развёртку изображения из одной сетки пикселей в другую. Изменение размера изображения необходимо, когда пользователю нужно увеличить или уменьшить число пикселей, тогда как изменение положения может происходить в самых различных случаях: исправление искажений объектива, смена перспективы или поворот изображения.

Даже если изменению размера или развёртки подвергается одно и то же изображение, результаты могут значительно отличаться в зависимости от алгоритма интерполяции. Поскольку любая интерполяция является всего лишь приближением, изображение будет несколько терять в качестве всякий раз, когда подвергается интерполяции [11].

Интерполяция изображений работает в двух измерениях и пытается достичь наилучшего приближения в цвете и яркости пикселя, основываясь на значениях окружающих пикселей [12]. На рисунке 2 представлен пример, иллюстрирующий работу интерполяции. На рисунке 2а можно увидеть исходное изображение с условным количеством пикселей, которое оно включает, рисунок 2b – является представлением изображения, которое должно получиться при увеличении разрешения на 180%, рисунок 2с – является ожидаемым результатом интерполяции, рисунок 2d – является результатом увеличения пикселей без интерполяции.

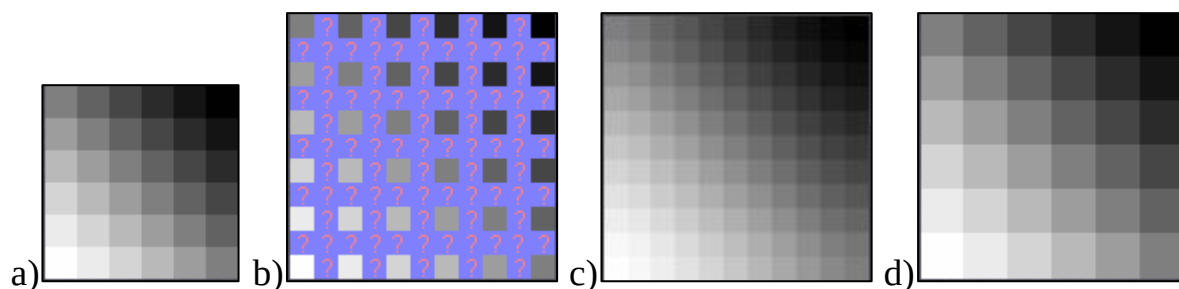


Рисунок 2 – Пример интерполяции

В отличие от других критериев и параметров, значения пикселей могут меняться намного более резко от точки к точке. Интерполяция придерживается правилу, что чем больше известно об окружающих пикселях, тем лучше работает интерполяция. Вот почему результаты быстро ухудшаются по мере растягивания изображения, а кроме того, интерполяция никогда не сможет добавить изображению детальности, которой в нём нет [13].

Общеизвестные и принятые алгоритмы интерполяции можно разделить на две крупные категории: адаптивные и неадаптивные.

Адаптивные методы подразумевают под собой изменения в зависимости от предмета интерполяции – это может быть гладкая текстура или же резкие границы исходного изображения. Такие методы в первую очередь разрабатываются для максимизации бездефектной детальности увеличенных изображений. Адаптивные методы разрешения включают в себя многие коммерческие алгоритмы в лицензированных программах. Многие из них применяют различные версии своих алгоритмов, когда обнаруживают наличие границы – с целью минимизировать неприглядные дефекты интерполяции в местах, где они наиболее видны [14].

Неадаптивные алгоритмы обрабатывают все пиксели одинаково, где в зависимости от сложности, они используют от 0 до 256 (или более) смежных пикселей для интерполяции. Чем больше смежных пикселей учитывается, тем более точными могут оказаться результаты, но это приводит и достигается за счёт значительного увеличения времени обработки. Такие алгоритмы могут применяться как для увеличения разрешения изображения, так и для масштабирования изображения. Неадаптивные алгоритмы включают в себя: метод ближайшего соседа, билинейная интерполяция, бикубическая интерполяция, сплайны, функция кардинального синуса (sinc), метод Ланцоша и другие.

Метод ближайшего соседа – интерполяция методом ближайшего соседа является самым простым методом интерполяции, он требует наименьшего времени обработки, поскольку учитывает только один пиксель – ближайший известный к точке интерполяции. В результате каждый пиксель просто становится больше.

Билинейная интерполяция – данная интерполяция рассматривает квадрат размером 2×2 известных пикселя, окружающих неизвестный. В качестве интерполированного значения используется взвешенное усреднение этих четырёх пикселей. В результате изображения выглядят значительно более гладко, чем результат работы метода ближайшего соседа. Главным недостатком метода при масштабировании изображения – для пикселей по

границам изображения (в каждой строке и столбце) исходного изображения не находится пары, с которой можно было бы провести интерполирование. Из-за этого требуется значительная предобработка изображения с помощью экстраполяции или замощения [15].

Бикубическая интерполяция – эта интерполяция идёт на один шаг дальше билинейной, рассматривая массив размером 4x4 окружающих пикселей – всего 16. Поскольку они находятся на разных расстояниях от неизвестного пикселя, ближайшие пиксели получают при расчёте больший вес. Бикубическая интерполяция производит значительно более резкие изображения, чем предыдущие два метода, и возможно, является оптимальной по соотношению времени обработки и качества на выходе. По этой причине она стала стандартной для многих программ редактирования изображений как, например, в Adobe Photoshop, драйверах принтеров и встроенной интерполяции камер.

Есть много других интерполяторов, которые принимают во внимание больше окружающих пикселей и таким образом требуют более интенсивных вычислений. Например, интерполяция высшего порядка: сплайны и функция кардинального синуса. Эти алгоритмы сохраняют большинство информации об изображении после интерполяции. Как следствие, они являются исключительно полезными, когда изображение требует нескольких поворотов или изменений перспективы за отдельные шаги. Однако, для однократных увеличений или поворотов такие алгоритмы высшего порядка дают незначительное визуальное улучшение при существенном увеличении времени обработки. Более того, в некоторых случаях алгоритм кардинального синуса на гладком участке отрабатывает хуже, чем бикубическая интерполяция [16].

Классические методы интерполяции включают в себя следующие некоторые недостатки, которые влияют на процесс увеличения разрешения изображения и сам результат. Все неадаптивные интерполяторы пытаются подобрать оптимальный баланс между тремя нежелательными дефектами:

граничный гало эффект, размытие и ступенчатость. На рисунке 3 представлен пример, на котором можно наблюдать дефекты при интерполяции изображения. На рисунке 3а можно увидеть исходное изображение, которое будет увеличено на 400%, рисунок 3б – является изображением, на котором представлен дефект ступенчатости, рисунок 3с – является изображением, на котором представлен дефект размытия, рисунок 3д – является изображением, на котором представлен дефект граничного гало эффекта.

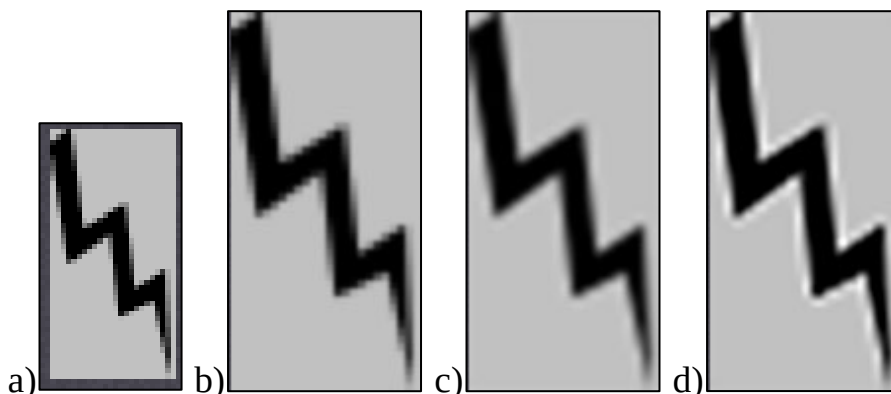


Рисунок 3 – Возможные дефекты интерполяции

Даже наиболее развитые неадаптивные интерполяторы всегда вынуждены увеличивать или уменьшать один из вышеприведенных дефектов за счёт двух других – как следствие, как минимум один из них будет замечен. Можно увидеть, насколько граничное гало похоже на дефект, порождаемый повышением резкости с помощью нерезкой маски, и как оно повышает кажущуюся резкость посредством усиления чёткости.

Адаптивные интерполяторы могут создавать или не создавать вышеописанные дефекты, но они тоже могут породить несвойственные исходному изображению текстуры или одиночные пиксели на крупных масштабах. На рисунке 4 представлено исходное изображение с малоразмерной текстурой и участок при увеличении в 220%.



Рисунок 4 – Эффект размытия при увеличении изображения

С другой стороны, некоторые «дефекты» адаптивных интерполяторов тоже могут рассматриваться как преимущества. Поскольку глаз ожидает увидеть в областях с мелкой текстурой, таких как листва, детали вплоть до мельчайших подробностей, подобные рисунки могут обмануть глаз на расстоянии [17].

Сглаживание является процессом, который пытается минимизировать появление ступенчатых или зубчатых диагональных границ, которые придают тексту или изображениям грубый цифровой вид. На рисунке 5 представлен результат сглаживания изображения, после его увеличения и появления дефекта ступенчатости.



Рисунок 5 – Эффект сглаживания

Важным соображением при увеличении изображений является предотвращение чрезмерной ступенчатости в результате интерполяции. Сглаживание удаляет эти ступеньки и создаёт впечатление более мягких границ и высокого разрешения. Оно принимает во внимание, насколько идеальная граница перекрывает смежные пиксели. Ступенчатая граница просто округлена вверх или вниз без промежуточного значения, тогда как сглаженная граница выдаёт значение, пропорциональное тому, насколько много от границы попало в каждый пиксель. На рисунке 6 представлен пример, на котором можно наблюдать варианты границы изображения. На рисунке 6а можно увидеть идеальную границу в мелком масштабе, рисунок 6б

– пример того, как выглядит ступенчатая граница, рисунок 6б – пример того, как выглядит сглаженная граница.

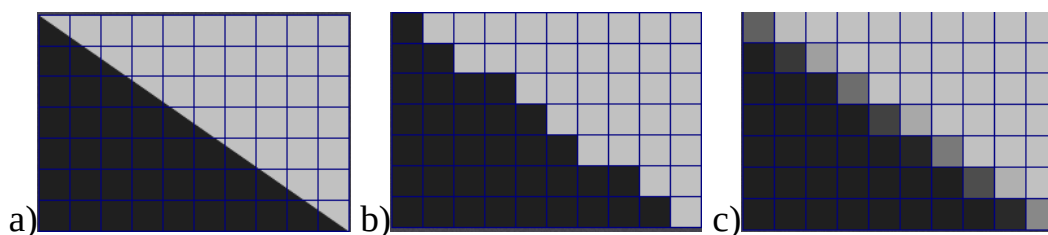


Рисунок 6 – Виды границ изображения

Многие интерполяторы определяют наличие границ и корректируются с целью минимизировать ступенчатость, сохранив при этом резкость границы. Поскольку сглаженная граница содержит информацию о своём положении при более высоком разрешении, именно поэтому классические алгоритмы увеличения разрешения не могут даже частично реконструировать границу при увеличении.

1.3 Нейросетевые методы увеличения разрешения изображений

Суперразрешение (Super-Resolution) – это результат восстановления и обработки изображения, которое является низкого разрешения (LR) в изображение с высоким разрешением (HR).

Существуют разнообразные типы сверток и архитектуры нейронных сетей, которые используются для обработки изображений. Кроме традиционных *2D*-сверток, для работы с масштабированием изображения можно использовать еще несколько интересных вариантов для получения необходимого результата. Например, развернутые (злокачественные) свертки, которые могут предложить широкое поле зрения на изображении. Такие приемы, как пропуск соединений, пространственный пирамидальный пуллинг и полносвязные блоки объединяют признаки как низкого, так и высокого уровня для повышения производительности. На рисунке 7 представлены основные виды сверток и блоков, которые используются при задаче увеличения разрешения изображения.

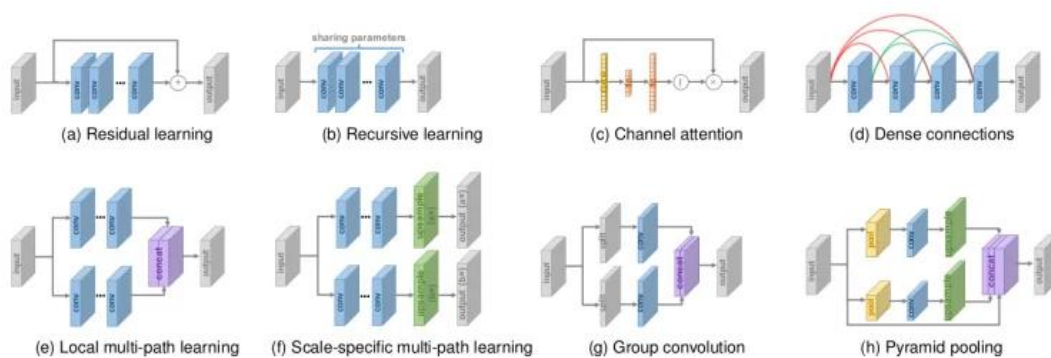


Рисунок 7 – Виды сверток и блоков

Все подходы связанные с решением проблемы увеличения разрешения при помощи нейросетевых методов можно разделить на три основные группы: предварительное увеличение разрешения (Pre-Upsampling), пост-увеличение разрешения (Post-Upsampling), прогрессивное увеличение разрешения (Progressive Upsampling).

Предварительное увеличение разрешения – в этой группе методов сначала происходит интерполяция изображения с низким разрешением для получения «грубых» изображений с более высоким разрешением. Затем нейронная сеть используется для обучения end-to-end отображения от интерполированных изображений с низким качеством до изображений с высоким качеством. Идея этой группы методов заключается в том, что может быть проще сначала увеличить разрешение изображений, с помощью традиционных методов (таких как билинейная или бикубическая интерполяция), а затем обработать изображение нейронными сетями чтобы получить необходимый результат, чем изучать ИИ прямому отображению из низкоразмерного пространства в многомерное пространство.

Основным преимуществом данного алгоритма является то, что поскольку Upsampling выполняется классическими методами, нейронную сеть необходимо только научить уточнять грубое изображение, что намного проще. Однако недостатком данной группы методов заключается в возможном усилении шумов и размытия изображения из-за предварительного повышения разрешения. На рисунке 8 представлен пример схематичной Pre-Upsampling сети.

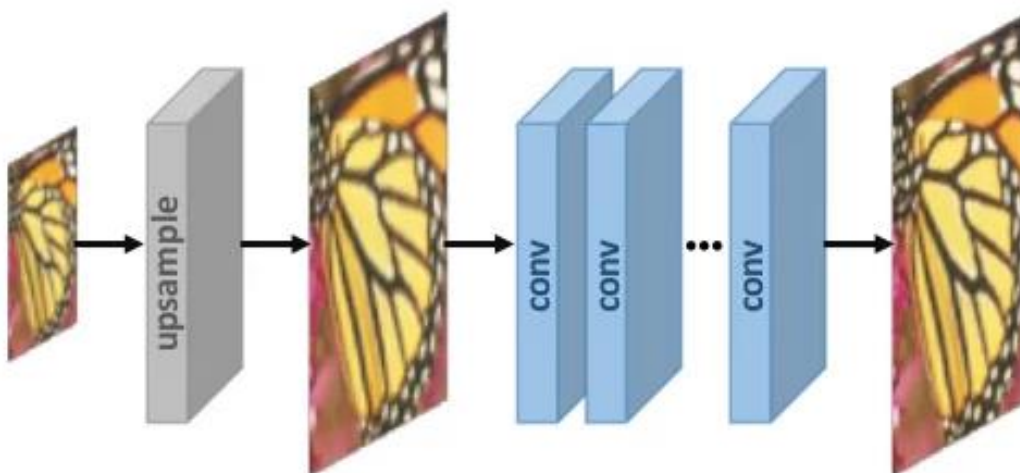


Рисунок 8 – Схематичный пример Pre-Upsampling сети

Пост-увеличение разрешения – в этом случае на сеть подаются оригинальные изображения с низким разрешением. Повышение разрешения выполняется на последнем обучаемом слое. Преимущество этого метода состоит в том, что извлечение признаков выполняется в пространстве меньшего размера (перед повышением разрешения), и, следовательно, вычислительная сложность уменьшается. Кроме того, используя обучаемый слой upsampling, модель можно обучать end-to-end. На рисунке 9 представлен пример схематичной Post-Upsampling сети.

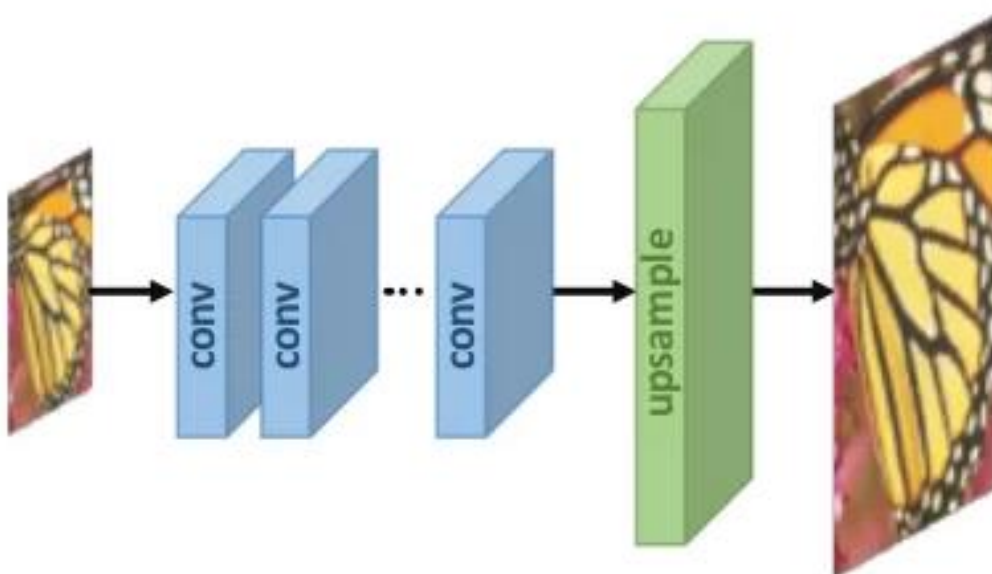


Рисунок 9 – Схематичный пример Post-Upsampling сети

Прогрессивное увеличение разрешения – в данной группе методов модели используют каскад слоев нейронной сети для постепенного восстановления изображений с высоким разрешением при меньших коэффициентах масштабирования на каждом этапе. На рисунке 10 представлен пример схематичной Progressive Upsampling сети.

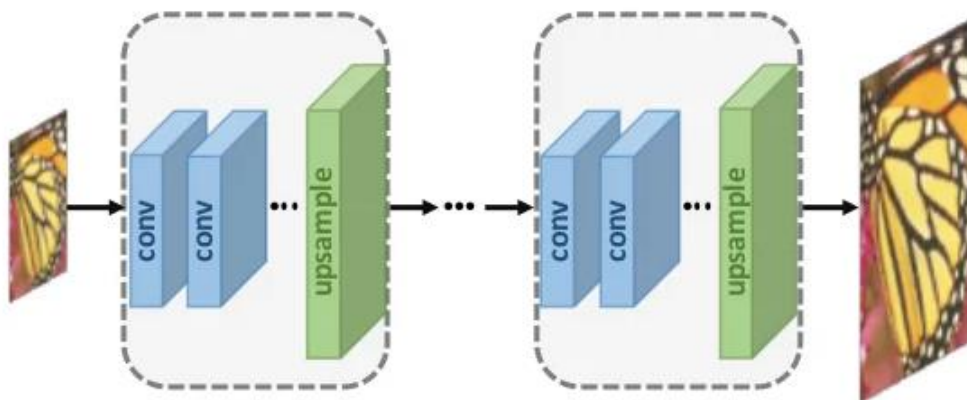


Рисунок 10 – Схематичный пример Progressive Upsampling сети

Нейронные сети, в отличие от классических методов, не требуют извлечения признаков из изображений вручную, а обучаются на больших объемах данных с известными метками. Существует большое количество различных нейросетевых моделей, которые имеют дополнительную возможность или специально разработаны для задачи увеличения разрешения изображения. Основные типы нейронных сетей, которые применяют для этого – сверточные нейронные сети (CNN) и генеративно-состязательные нейронные сети (GAN).

Сверточная нейронная сеть с суперразрешением (SRCNN) – CNN обычно используются для классификации изображений. В SRCNN он используется для суперразрешения одиночного изображения (SR). В данной сети есть только 3 части: извлечение и представление патчей, нелинейное отображение и реконструкция. На рисунке 11 показана схематичная структура SRCNN сети.

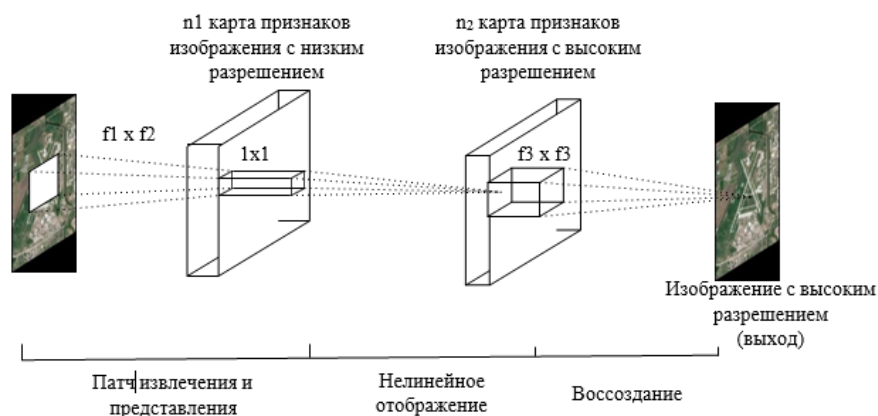


Рисунок 11 – Структура сети CNN

Сеть SRCNN имеет следующие особенности [27]:

- SRCNN полностью сверточная, что обеспечивает скорость ее работы,
- обучение проводится по критерию качества работы фильтров, а не достигнутой точности. Такой подход позволяет увеличивать масштаб изображения,
- SRCNN не требует дообучения при работе. После достижения необходимых параметров фильтров при обучении, сеть может выполнять простой прямой переход для получения изображения с высоким разрешением. Для получения результата не требуется оптимизация функции потерь для каждого изображения,
- SRCNN имеет сквозную (end-to-end) архитектуру. На вход сети подается изображение и на выходе получается изображение с высоким разрешением, без каких-либо промежуточных шагов.

EnhanceNet – нейросеть, специализирующаяся на обработке и улучшении изображения, а также реализует функционал увеличения разрешения. EnhanceNet использует глубокое обучение и мощные алгоритмы для повышения разрешения фотографий, делая их более четкими и детализированными. Данная нейронная сеть способна восстановить недостающие детали и сегменты изображений, увеличить резкость, контрастность, а также сделать более насыщенной цветовую гамму исходного изображения. Данная модель позволяет пользователю реализовать

высококачественные изображения с удивительно высокой степенью детализации и реалистичности.

В реализации сети лежит алгоритм масштабирования изображения при помощи бикубической интерполяции, предварительно перед работой модели все поступающие изображения из-за необходимости уменьшаются в четыре раза. Для совместимости и устранения возможных неприятностей используется версия TensorFlow для ЦП. Из возможных недостатков можно выделить то, что график вычислений модели перестраивается для каждого изображения, поэтому данная эталонная реализация не отражает производительность модели во время выполнения и не подходит для тестов во время выполнения.

SRGAN – это генеративно-сопоставительная нейронная сеть (Generative Adversarial Network) направленная на повышение и масштабирование разрешения изображений (Super-Resolution). На текущий момент это одно из нескольких решений из семейства GAN, которое способно восстанавливать и обрабатывать исходные текстуры из изображений, уменьшенных в четыре раза. Использование генеративно-сопоставительных нейронных сетей для решения задачи увеличения разрешения изначально реализовывалась, как модель способная и призванная решить ранее описанную проблему сглаживания пикселей. При использовании классических методов интерполяции в некоторых случаях результат оказывался непригодным, эти методы при скорости своей работы, выдавали решения с чрезмерно гладкими текстурами. В свою очередь GAN обеспечивают мощную основу для создания правдоподобных SR-изображений с высоким качеством восприятия.

Основные особенности SRGAN:

- увеличение разрешения изображения с высокими коэффициентами масштабирования (4x),
- использование функции потерь восприятия (perceptual loss function), рассчитанной с помощью карт признаков VGG сети, которые более инвариантны к изменениям пикселей,

– высокие результаты средней экспертной оценки MOS на общедоступных наборах данных.

1.4 Анализ инструментов для разработки

На текущий момент есть широкий выбор языков программирования для задания и использования нейронных сетей. В настоящее время наиболее часто применяется язык Python благодаря наличию множества мощных библиотек с набором встроенных математических функций, позволяющих определять, тренировать и запускать нейронные сети различных архитектур.

Для нейронных сетей, работающих с изображениями, необходимо подключение другого модуля.

TensorFlow – это библиотека для машинного обучения от компании Google с открытым исходным кодом. Применяется для построения и тренировки нейронных сетей любой сложности, решающих задачи нахождения и классификации образов, распознавание естественного языка, изображений и рукописных текстов, разнообразные задачи классификации или кластеризации, возможности обработки больших данных. Библиотека построена на парадигме программирования потоков данных, позволяющей оптимизировать математические вычисления.

Основные особенности TensorFlow:

– модели искусственного интеллекта представлены с помощью графов потоков данных – математических абстракций, которые состоят из вершин, которое отображают операции и путей между ними, являющимися потоками данных между узлами, то есть информация «движется» по графу, передается по путям от вершины к вершине,

– TensorFlow работает с уникальным типом данных – тензорами. Это многомерные структуры данных в векторном, то есть направленном пространстве. Они используются в линейной алгебре и физике. Отсюда пошло и название библиотеки. С помощью тензоров описываются пути графа, где вершинами являются математические операции,

– библиотека имеет возможность использовать ресурсы обычного центрального процессора (CPU) или задействовать мощности графического процессора (GPU), переключение режима устанавливается в программном коде. Существует специальный процессор тензорного уровня TPU, разработанный авторами библиотеки, – его можно использовать через облачные сервисы Google.

Преимущества TensorFlow:

– высокий уровень абстракции представленный в библиотеке. TensorFlow написана так, что разработчику не нужно беспокоиться о технической реализации абстрактных понятий. Можно сосредоточиться на описании логики программы и на математике, а способ реализации вычислений – задача TensorFlow, а не программиста. Это облегчает разработку и позволяет сконцентрироваться на важных задачах,

– одной из особенностей библиотеки является интерактивная разработка. TensorFlow позволяет работать с компонентами модели по отдельности и создавать ее «на ходу», при этом отдельно проверять каждый элемент. Это удобнее, чем описывать граф как единую монолитную структуру. Подход делает разработку более интерактивной – структуру можно гибко настраивать и менять,

– гибкий арсенал инструментов TensorFlow позволяет без проблем разработчику использовать библиотеку для создания нейронных сетей, для глубокого обучения и других направлений ML. Его функции разнообразны для решения широкого спектра задач. Благодаря графам и тензорам в TensorFlow можно легко изобразить сложную математическую структуру. Гибкость касается не только функций, но и технической стороны вопроса: библиотека работает и с центральным, и с графическим процессором, ее легко использовать с другими инструментами для машинного обучения. Например, TensorFlow применяют с API Keras,

– TensorFlow является кроссплатформенным инструментом, как и сам Python, работает в популярных операционных системах, локально или в

облаке. У нее есть расширения для мобильных устройств, IoT и браузерных приложений. Для мобильных устройств и интернета вещей можно воспользоваться средой TensorFlow Lite. А если модель машинного обучения должна работать в браузере, подойдет TensorFlow.js – версия для использования с JavaScript и Node.js,

- TensorFlow, являясь популярной библиотекой, обладает большим сообществом, поэтому на многие вопросы легко можно найти ответ в сообществе. Оно развивает технологию, создает новые продукты и дополнения, связанные с TensorFlow, пишет документацию и обучающие материалы. Все это облегчает старт и позволяет извлечь из библиотеки максимум пользы.

Недостатки TensorFlow:

- являясь продуктом Google TensorFlow обладает собственными стандартами. Компания известна собственными стандартами для технологий. Первая версия библиотеки предназначалась для внутреннего использования. До сих пор в TensorFlow встречается неочевидное поведение, из-за которого код может быть сложнее отлаживать. Сложности можно компенсировать, если внимательно изучать документацию и пользоваться дополнительными инструментами для отладки,

- разработка требует высокое потребление памяти. Если TensorFlow используется с графическим процессором, она забирает всю его память. Это снижает производительность. Например, если моделей несколько, и они написаны на разных библиотеках, TensorFlow отнимет видеопамять у других – возникнет ошибка. Чтобы такого не было, потребление памяти библиотекой надо ограничивать вручную,

- библиотека является сложной в изучении, это касается всего машинного обучения в целом. С TensorFlow сложности могут возникнуть из-за ее специфичных стандартов – это не самая дружелюбная к новичкам библиотека.

PyTorch – современная библиотека глубокого обучения, разрабатывается преимущественно группой искусственного интеллекта Facebook. Это библиотека для языка Python с открытым исходным кодом, созданный на базе своего предшественника Torch. Используется для решения таких различных задач, как компьютерное зрение, обработка естественного языка, исправление и преобразование изображений.

Библиотека PyTorch включает в себя следующие модули:

- модуль Autograd представляет собой метод автоматической дифференциации. Производится запись вычислений, произведенных в прямом направлении, затем производится воспроизведение в обратном порядке для вычисления градиентов (backpropagation). Этот метод особенно полезен при построении нейронных сетей, так как позволяет рассчитывать дифференциальные поправки параметров одновременно с прямым проходом,
- модуль Optim реализует несколько алгоритмов оптимизации, используемых при построении нейронных сетей. Реализовано большинство наиболее часто используемых методов,
- модуль nn позволяет легко определять вычислительные графы и работать с градиентами, однако может быть слишком низким уровнем для определения сложных нейронных сетей. Более высокоуровневой абстракцией для таких применений является модуль nn.

Преимущества PyTorch:

- гибкость библиотеки позволяет создавать большие и сложные модели и решать разнообразные задачи. Это возможно благодаря прямому доступу к математическим функциям и возможности писать собственные. Есть решения, которые не позволяют построить что-то нестандартное – PyTorch к ним не относится. В свою очередь он превосходно подходит именно для сложных и нестандартных моделей. Дополнительные плюсы – поддержка CUDA и инструменты для оптимизации. Также вокруг этой библиотеки выстроена экосистема, состоящая из различных библиотек, разрабатываемых сторонними командами: PyTorch Lightning и Fast.ai, упрощающие процесс

обучения моделей, Pygo – модуль для вероятностного программирования, от Uber, Flair, для обработки естественного языка и Catalyst, для обучения DL и RL моделей,

- благодаря динамическому построению графа вычислений PyTorch лишен недостатков, которые есть у некоторых альтернативных вариантов. Вносить изменения в граф можно «на лету», это упрощает отладку и переписывание нейронной сети при необходимости,

- некоторые функции и возможности PyTorch напоминают таковые у NumPy, хотя PyTorch мощнее и функциональнее. Тем не менее проекты на NumPy легко конвертировать в формат PyTorch – и наоборот. Тензоры и другие сущности в этих двух инструментах похожи структурой: достаточно немного их изменить для полноценной конвертации. Специальная функция перевода тензоров из формата NumPy в формат PyTorch и обратно позволяет с легкостью адаптировать код друг с другом,

- подробная, обширная и понятная документация, прекрасно помогает тем, кто впервые встречается с библиотекой, в которой расписаны все важные моменты. Все нужное задокументировано: если возникнут проблемы, всегда можно обратиться к актуальным документам,

- обширное сообщество библиотеки позволяет легко решить проблему, возникающую у разработчика. Ещё недавно PyTorch был менее популярен, чем некоторые другие решения, но в настоящий момент по частоте упоминаний догнал TensorFlow. У библиотеки большая экосистема, широкое и активное сообщество: это и помощь новичкам, и постоянное появление новых удобных решений для разработки.

Недостатки PyTorch:

- в библиотеке отсутствуют инструменты визуализации. Встроенных инструментов для построения графиков, диаграмм и других способов визуализации в PyTorch нет. Приходится пользоваться сторонними решениями, такими как Tensorboard, хотя изначально он предназначался не для PyTorch,

– в библиотеке отсутствуют инструменты развертывания. В PyTorch нет встроенных инструментов для легкого запуска на серверах, поэтому приходится использовать сторонние стандарты, протоколы и инструменты,

– в PyTorch присутствует потребность в дополнительных библиотеках, из-за всего перечисленного выше при профессиональной работе с PyTorch понадобится пользоваться дополнительными библиотеками. Это значит, что разработчик должен разобраться не с одним инструментом, а с несколькими, чтобы полноценно выполнять задачи.

Учитывая все преимущества и недостатки представленных средств разработки и потребности, которые могут возникнуть при работе и обучении нейронной сети, была выбрана библиотека PyTorch и соответствующий ей язык Python, в качестве среды разработки для основных задач было принято использовать IDE PyCharm.

2 Модель увеличения разрешения изображений

2.1 Описание модели

В основе работе применяется особый тип генеративно-сопоставительной сети: Real-ESRGAN (Enhanced Super-Resolution Generative Adversarial Network или расширенная генеративно-сопоставительная сеть сверхвысокого разрешения). Real-ESRGAN подразумевает под собой улучшенную версию ESRGAN, которая была разработана с использованием модифицированных архитектур и функций потерь для улучшения качества изображения.

Алгоритм-генератор Real-ESRGAN работает благодаря нейронной сети, использующей сверточные слои для сбора информации об исходном изображении с низким разрешением. При передаче это изображение захватывает так называемое «пространство признаков». Этот набор свойств используется для описания визуальных паттернов, например, изображен ли на картинке мех, кирпич или какой-то другой известный материал. Сверточные слои нужны для обработки и хранения этой информации, чтобы при создании изображения со сверхвысоким разрешением оно сохраняло то же пространство признаков, что и исходник. Получив эту информацию, генератор начинает масштабировать изображение. На рисунке 12 представлена архитектура нейросетевой модели Real-ESRGAN.

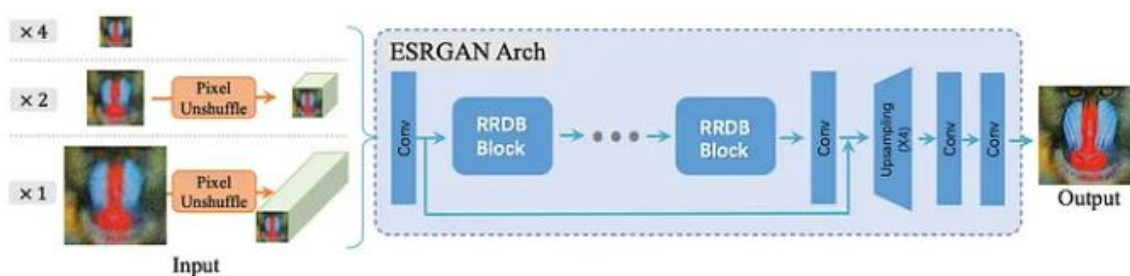


Рисунок 12 – Схема нейросетевой модели Real-ESRGAN

Алгоритм-генератор ESRGAN реализован на структуре RDB блоков, которые отвечают за сбор информации с изображения и на её основе генерируют новое изображение.

2.2 Архитектура сети

Пример стандартной ситуации работы генеративно-состязательной сети для задачи увеличения разрешения. Входные данные – изображения с низким разрешением - I^{LR} . Результат работы нейронной сети - I^{SR} . Изображение высокого разрешения (без обработки) - I^{HR} . I^{LR} будем получать уменьшением I^{HR} в n раз. В результате обучения нейронной сети получается генерирующая функция G . Генератор будет обучаться как сверточная нейронная сеть прямого распространения с параметрами θ_G (5):

$$\theta_G = \{W_{1:L}; b_{1:L}\}, \quad (5)$$

где $W_{1:L}$, $b_{1:L}$ вес и смещение нейронной сети глубиной в L слоев.

Этот параметр вычисляется с помощью оптимизации составной функции потерь I_{SR} . Для обучения нейронной сети нужно значение, вычисляемое по формуле (6):

$$\hat{\theta}_G = \min \frac{1}{N} I_{SR}(G(I_n^{LR}), I_n^{HR}), \quad (6)$$

Функция потерь I_{SR} (7):

$$I_{SR} = I_{MSE} + 10^{-3} I_{GEN}, \quad (7)$$

где I_{MSE} - средняя квадратическая ошибка при сравнении сгенерированного изображения I^{SR} с его оригиналом I^{HR} .

Соревновательная функция потерь I_{GEN} рассчитывается по формуле (8):

$$L_{GEN} = \sum_{n=1}^N \log (D(G(I^{LR}))), \quad (8)$$

где $D(G(I^{LR}))$ - вероятность распознавания дискриминатором того, что созданное генератором изображение $G(I^{LR})$ является изображением высокого разрешения I^{HR} .

Сеть генератора состоит из 5 остаточных блоков, которые манипулируют изображением в нижнем масштабе, и метода для восстановления изображения сверхвысокого разрешения без необходимости вручную заполнять промежуточные значения пикселей. Каждый остаточный блок генератора имеет два набора обычных блоков (Co-BN-ReLU) с постоянным номером канала 64 и размером ядра 3 для всех блоков. На рисунке 13 представлена модельная структура генератора, где Conv – сверточный слой с n картами признаков и шагом равным s , Input I^{LR} – изображение низкого разрешения, поступающее на вход генератору, ReLU – активационная функция Parametric ReLU, PixelShuffler x2 – субпиксельные сверточные слои, Elementwise sum – поэлементная сумма, I^{SR} – результат работы нейронной сети.

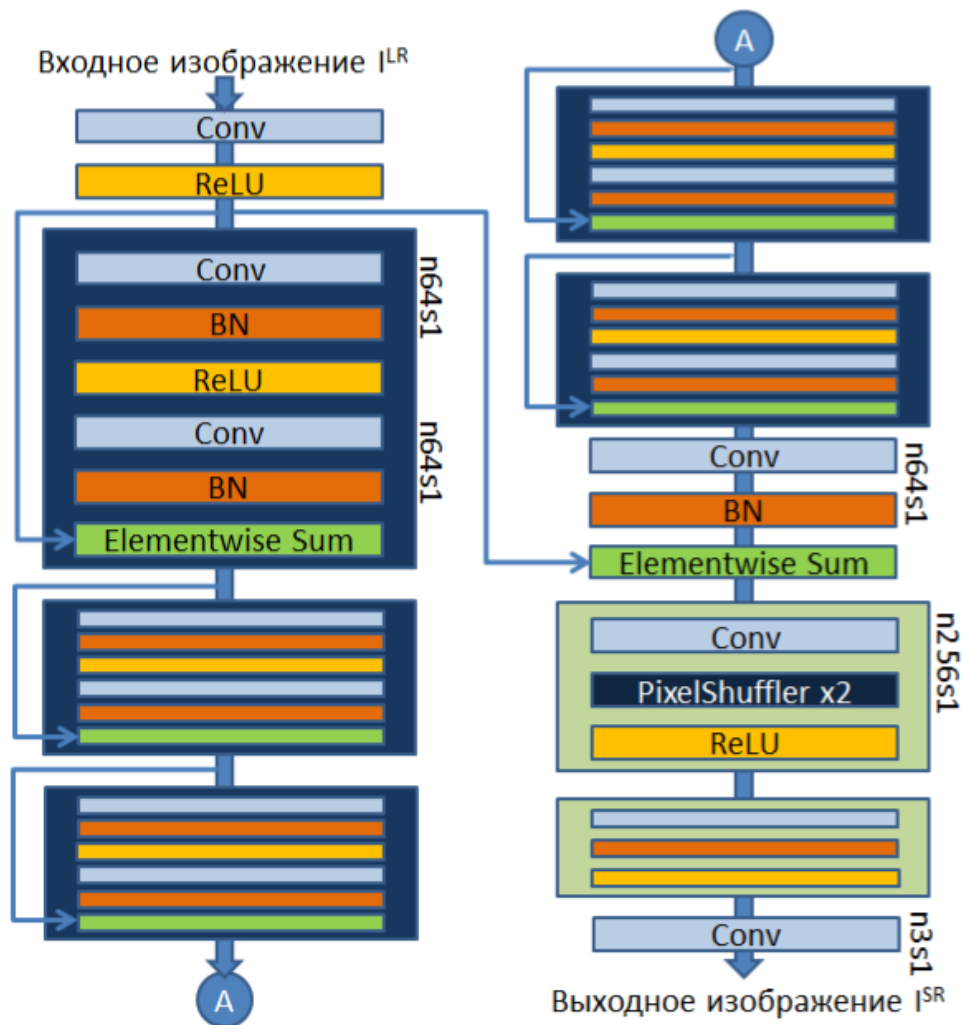


Рисунок 13 – Модельная структура генератора

В работах [17, 18] было подробно описано, что удаление слоев BN повышает производительность и снижает вычислительную сложность в различных задачах, ориентированных на PSNR, включая SR и устранение размытия. Слои BN нормализуют характеристики, используя среднее значение и дисперсию в пакете во время обучения, и используют оцененные среднее значение и дисперсию всего набора обучающих данных во время тестирования. Когда статистика наборов, обучающих и тестовых данных сильно отличается, слои BN, как правило, вносят неприятные артефакты и ограничивают способность к обобщению.

Поэтому удаление слоев BN необходимо для стабильного обучения и постоянной производительности. Кроме того, удаление слоев BN помогает снизить сложность вычислений и использование памяти как при обучении, так и при непосредственном выполнении увеличения разрешения входного изображения.

В структуре блока RRDB используется более глубокая и сложная структура, чем в первоначальном остаточном блоке в SRGAN. Основываясь на наблюдении [19], решено, что большее количество уровней и соединений всегда может повысить производительность. Остаточные блоки содержат "обходную связь идентичности" (identity shortcut connection), обходящую один или большее количество слоев. Такой блок добавляет возможность при обратном распространении ошибки значениям градиента в слоях быть более чувствительным к градиенту в слоях, с которыми связаны с помощью shortcut, то есть расположенными несколько дальше следующего слоя. На рисунке 14 представлена схематичная работа Residual Block.

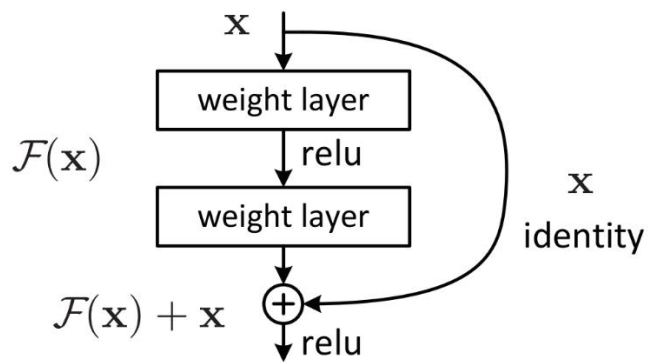


Рисунок 14 – Residual Block

В частности, предлагаемая база данных RRDB имеет структуру "остаток в остатке", где остаточное обучение используется в различных уровнях слоев. Входной слой настраивается для ввода изображений с несколькими разрешениями путем применения пиксельного перемешивания (pixels-shuffled) (обратного перетасовке пикселей (pixels-unshuffled)) для уменьшения пространственного размера и увеличения количества каналов для повышения эффективности. На рисунке 15 представлена схематичная работа субпиксельных слоев модели.

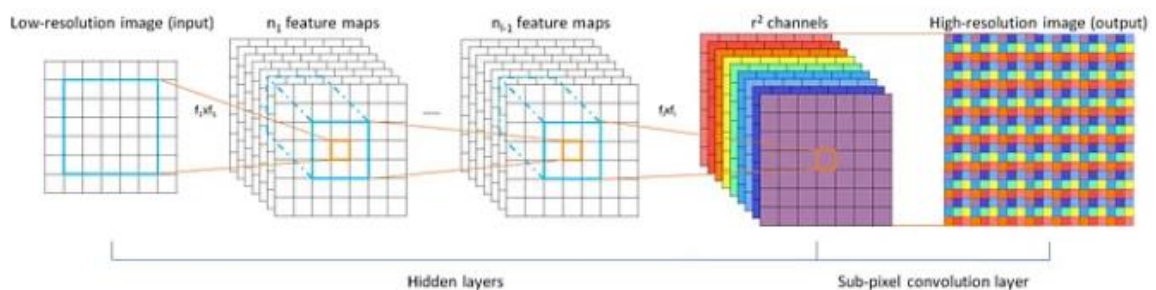


Рисунок 15 – Схематичная работа субпиксельных слоев модели

Блок RRDB отличается от [20] тем, что используемый остаточный блок применяется в основном потоке, где пропускная способность сети увеличивается за счет плотных соединений. На рисунке 16 представлена архитектура блока RRDB.

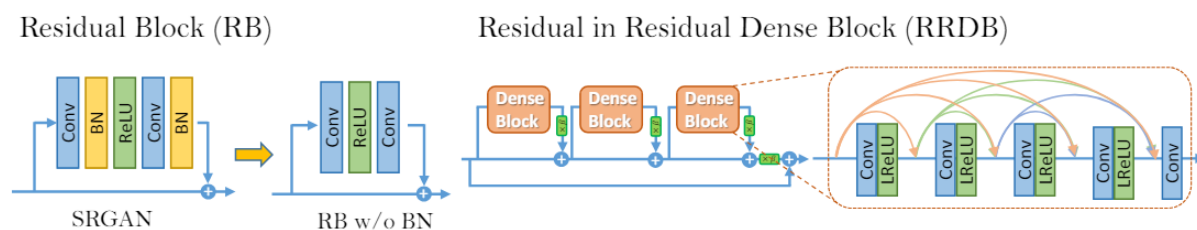


Рисунок 16 – Блок RRDB используемый в модели.

В дополнение к улучшенной архитектуре, также используется несколько методов для облегчения обучения очень глубокой сети:

- масштабирование по остаточному принципу [21, 22], т.е. уменьшение остаточных значений путем умножения константы между 0 и 1 перед добавлением их в основное изображение для предотвращения неустойчивости работы и дефектов в результате,
- уменьшение дисперсии между инициализируемыми параметрами, поскольку эмпирически установлено [23], что нейросетевую модель легче обучать, когда начальная дисперсия параметров меньше.

2.3 Дискриминатор

Дискриминатор – это обычная CNN, которая вводит изображение и стремится определить, является ли это реальным изображением или ложным. На рисунке 17 представлена модельная структура дискриминатор, где Input – входные данные изображения, созданные генератором и изображения высокого разрешения, Conv – сверточный слой с n картами признаков и шагом равным s , Leaky ReLU – активационная функция Leaky ReLU, Sigmoid – активационная функция сигмоида, I^{HR} , I^{SR} – изображение высокого и изображение низкого разрешения, а также метка класса изображения, т.е. настоящее изображение высокого разрешения или созданное генератором.

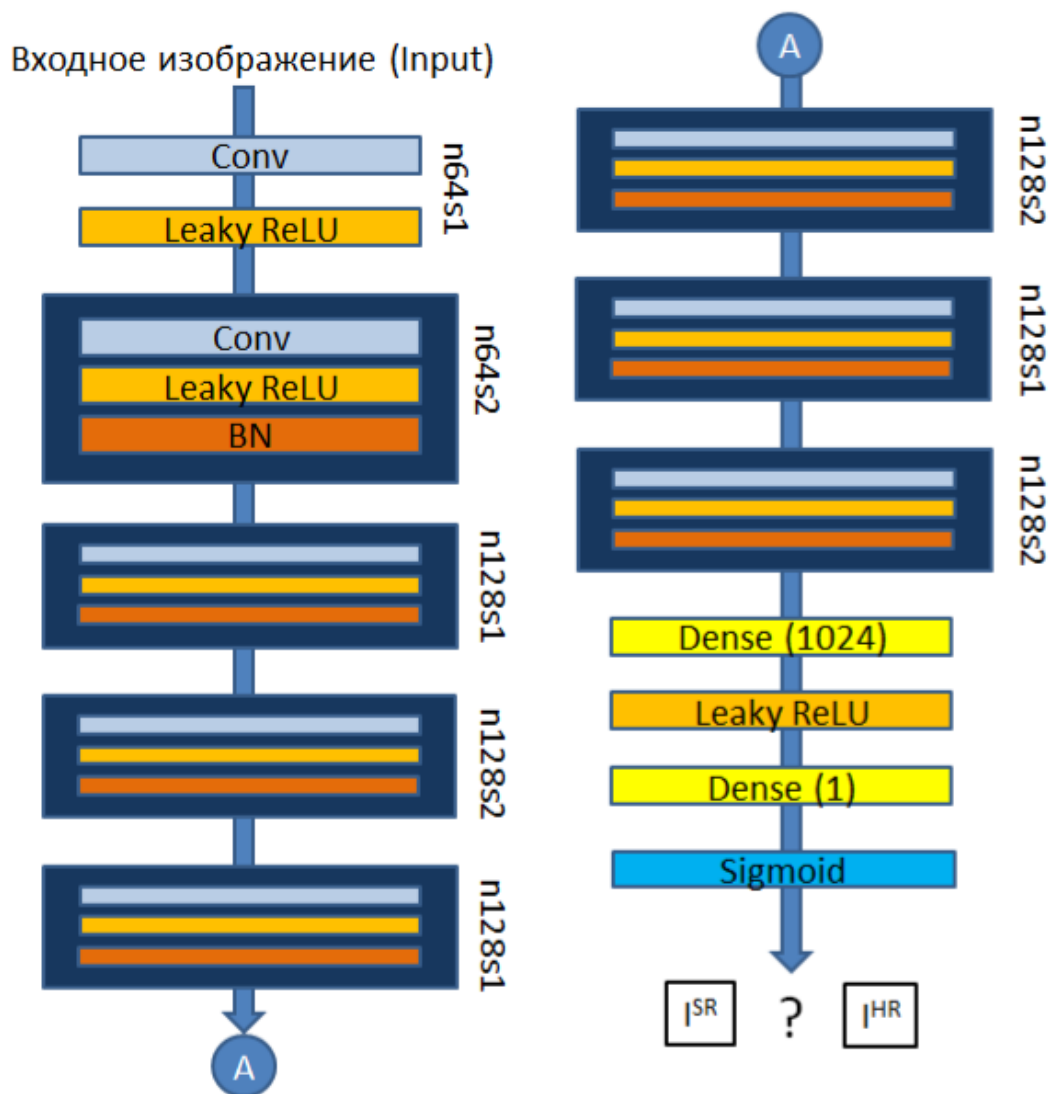


Рисунок 17 – Модельная структура дискриминатора

Помимо улучшенной структуры генератора, также усовершенствован дискриминатор, основанный на релятивистском GAN. В отличие от стандартного дискриминатора D в GAN подобных моделях и методах, который оценивает вероятность того, что одно входное изображение x является реальным и натуральным, релятивистский дискриминатор пытается предсказать вероятность того, что реальное изображение x_r относительно более реалистично, чем поддельное изображение x_f [24]. Данный алгоритм представлен на рисунке 18.

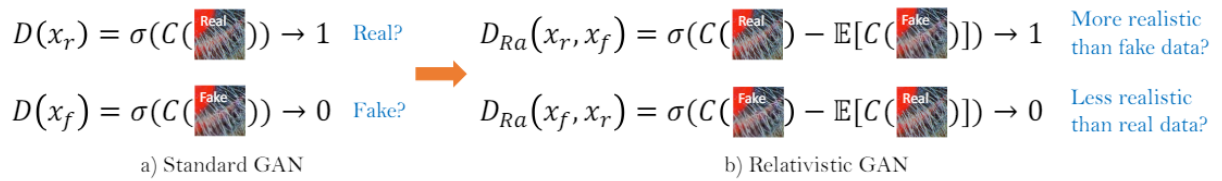


Рисунок 18 – Разница между стандартным дискриминатором и релятивистским дискриминатором

В частности, заменен стандартный дискриминатор на релятивистский средний дискриминатор RaD, обозначаемый как D_{Ra} . Стандартный дискриминатор в GAN подобных моделях и методах может быть выражен по формуле (9):

$$D(x) = \sigma(C(x)), \quad (9)$$

где σ – сигмовидная функция,

$C(x)$ – неинформированный выходной сигнал дискриминатора.

Затем RaD приводится к формуле (10):

$$D_{Ra}(x_r, x_f) = \sigma(C(x_r) - E_{x_f} [C(x_f)]), \quad (10)$$

где E_{x_f} – представляет собой операцию вычисления среднего значения для всех поддельных данных в мини-пакете (batch).

Затем потеря дискриминатора определяется по формуле (11) как:

$$L_D^{Ra} = -E_{x_r} [\log(D_{Ra}(x_r, x_f))] - E_{x_f} [\log(1 - D_{Ra}(x_r, x_f))], \quad (11)$$

Конкурентные потери для генератора выражаются в симметричной форме по формуле (12):

$$L_G^{Ra} = -E_{x_r} [\log(1 - D_{Ra}(x_r, x_f))] - E_{x_f} [\log(D_{Ra}(x_r, x_f))], \quad (12)$$

где $x_f = G(x_i)$, а x_i обозначает входное изображение LR.

Было протестировано, что потери при столкновении для генератора содержат как x_r , так и x_f . Таким образом, генератор извлекает выгоду из градиентов как сгенерированных, так и реальных данных при состязательном

обучении, в то время как в GAN подобные модели и методы в действие вступает только сгенерированная часть [25].

2.4 Интерполяция

Чтобы устранить неприятные шумы на изображении после обработки методами, применяемыми в GAN, и при этом не потерять хорошее качество восприятия, был выбран гибкий и эффективный алгоритм – сетевая интерполяция (Network Interpolation). Интерполяция – процесс заполнения неизвестных данных между двумя известными значениями. В частности, сначала обучается сеть G , ориентированная на PSNR, а затем получается сеть G_{GAN} , основанная на GAN, путем тонкой настройки. Необходимо интерполировать все соответствующие параметры этих двух сетей, чтобы получить интерполированную модель G , параметрами которой по формуле (13) являются:

$$\theta^{INTERP} = (1 - \alpha)\theta^{PSNR} + \theta^{GAN}, \quad (13)$$

где θ^{INTERP} , θ^{PSNR} и θ^{GAN} – являются параметрами G_{INTERP} , G_{PSNR} и G_{GAN} соответственно,

$\alpha \in [0, 1]$ – это параметр интерполяции.

Такая сетевая интерполяция обладает двумя преимуществами. Во-первых, интерполированная модель способна выдавать значимые результаты для любого допустимого значения α без значительной потери качества. Во-вторых, имеется возможность настраивать и поддерживать баланс между качеством восприятия и точностью без переобучения модели [26].

В качестве альтернативного варианта были рассмотрены методы, позволяющие сбалансировать эффекты методов, ориентированных на PSNR, и методов, основанных на GAN. Например, метод, когда можно напрямую интерполировать выходные изображения (пиксель за пикселем), а не параметры сети. Однако такой подход не всегда эффективен. При таком подходе не удастся достичь оптимального соотношения между шумом и

размытием, т.е. интерполированное изображение получается либо слишком размытым, либо зашумленным артефактами.

Другой метод заключается в настройке весовых коэффициентов потери графической информации и потерь из-за конкуренции, т.е. параметров λ и η в уравнении. Но этот подход требует настройки коэффициента потерь и тонкой настройки сети, и, таким образом, достижение постоянного контроля качества изображения обходится слишком дорого.

3 Реализация алгоритма

В классе `discriminator_arch.py` содержится реализация дискриминатора GAN со спектральной нормализацией SN. Поля класса `discriminator_arch.py` представлены в таблице 1.

Таблица 1 – Описание полей класса `discriminator_arch.py`

Тип данных	Название	Описание
int	num_in_ch	Количество входных каналов. По умолчанию: 3.
int	num_feat	Количество каналов для базовых промежуточных функций. По умолчанию: 64.
bool	skip_connection	Переменная следит за тем, необходимо ли использовать функцию пропуска соединений. По умолчанию: True.

Класс `discriminator_arch.py` включает в себя 1 метод. Далее рассмотрим описание метода данного класса.

`def forward(self, x)` – метод, принимает на вход контекст вызова класса и параметр типа `Torch`, который хранит в себе информацию об входном изображении. Внутри метода происходит понижение или повышение выборки, с помощью функций `leaky_relu` и `interpolate`, в зависимости от параметров.

В файле `arch_utils.py` реализованы несколько классов и функций необходимых для модели.

Класс `ResidualBlockNoBN` реализует логику остаточного блока без пакетной нормализации – `Batch Normalization`. Поля класса `ResidualBlockNoBN` представлены в таблице 2.

Таблица 2 – Описание полей класса ResidualBlockNoBN

Тип данных	Название	Описание
int	num_feat	Количество каналов для промежуточных функций. По умолчанию: 64.
float	res_scale	Размер разрешения. Значение по умолчанию: 1.
bool	pytorch_init	Если установлено значение True, использует pytorch init по умолчанию, в противном случае использует default_init_weights. По умолчанию: False.

Класс Upsample реализует модуль повышающей выборки. Поля класса Upsample представлены в таблице 3.

Таблица 3 – Описание полей класса Upsample

Тип данных	Название	Описание
int	scale	Коэффициент масштабирования. Поддерживаемые масштабы: 2^n и 3.
int	num_feat	Количество промежуточных функций канала.

Файл arch_utils.py включает в себя 5 метод. Далее рассмотрим описание методов данного файла.

`def default_init_weights(module_list, scale=1, bias_fill=0, **kwargs)` – метод, принимает на вход `module_list` – список модулей, подлежащих инициализации, `scale` – размер масштабирования для инициализированных весов, `bias_fill` – значение для заполнения смещения и массив остальных передаваемых аргументов. Функция инициализирует входные веса для нейронной сети.

`def make_layer(basic_block, num_basic_block, **kwarg)` – метод, принимает на вход контекст вызова класса и параметр типа Torch, который хранит в себе информацию об входном изображении. Функция создает слои

нейронной сети, возвращает контейнер `nn.Sequential`. Модули будут добавлены в него в том порядке, в котором они были переданы в конструкторе. Затем он последовательно “связывает” выходные данные с входными данными для каждого последующего модуля, в конечном итоге возвращая выходные данные последнего модуля.

`def flow_warp(x, flow, interp_mode='bilinear', padding_mode='zeros', align_corners=True)` – метод, принимает на вход параметр типа `Torch`, который хранит в себе информацию об входном изображении, `interp_mode` – вариант работы 'nearest' или 'bilinear', по умолчанию стоит 'bilinear'. `padding_mode` – вариант работы 'zeros', 'border' или 'reflection', по умолчанию стоит 'zeros'. Функция заворачивает изображение или объект с помощью оптического потока.

`def resize_flow(flow, size_type, sizes, interp_mode='bilinear', align_corners=False)` – метод, принимает на вход `flow` - предварительно рассчитанный поток, `size_type` - 'ratio' или 'shape', `sizes` - коэффициент для изменения размера или окончательной формы на выходе. Порядок отношения должен быть `[ratio_h, ratio_w]`. Для понижающей выборки отношение должно быть меньше 1,0 (т.е. отношение $< 1,0$). Для повышения дискретизации коэффициент должен быть больше 1,0 (т.е. коэффициент $> 1,0$). `interp_mode` – вариант работы 'nearest' или 'bilinear', по умолчанию стоит 'bilinear'. `padding_mode` – вариант работы 'zeros', 'border' или 'reflection', по умолчанию стоит 'zeros'. Функция изменяет размер потока в соответствии с соотношением изображения или формой.

`def pixel_unshuffle(x, scale)` – метод, принимает на параметр типа `Torch`, который хранит в себе информацию об входном изображении и коэффициент понижающей выборки. Внутри метода реализуется алгоритм перетасовки пикселей. Возвращает массив типа `Tensor`.

В файле `model.py` реализован класс основной модели.

Класс `RealESRGAN` инициализирует входные параметры, устанавливает параметр `device` со значениями `'cuda'` или `'cpu'`, а также масштаб увеличения разрешения изображения. Поля класса `RealESRGAN` представлены в таблице 4.

Таблица 4 – Описание полей класса `ResidualBlockNoBN`

Тип данных	Название	Описание
string	device	Устройство для выбора, на основе чего будет выполняться работа.
int	scale	Размер разрешения. Значение по умолчанию: 4.

Файл `arch_utils.py` включает в себя 2 метод. Далее рассмотрим описание методов данного файла.

`def load_weights(self, model_path)` – метод, принимает на вход контекст вызова класса и путь к модели с помощью которой реализуется увеличение разрешения. Внутри метода загружаются и устанавливаются веса, устанавливается устройство для работы.

`def predict(self, lr_image, batch_size=4, patches_size=192, padding=24, pad_size=15)` – Функция прогнозирования, принимает на вход `lr_image` – изображение с низким разрешением, возвращает `sr_image` – изображение с увеличенным разрешением.

В файле `rrdbnet_arch.py` реализованы несколько архитектурных классов необходимых для модели.

Класс `ResidualDenseBlock` инициализирует входные параметры, веса и значения для сети и блока. Поля класса `ResidualBlockNoBN` представлены в таблице 5.

Таблица 5 – Описание полей класса ResidualBlockNoBN

Тип данных	Название	Описание
int	num_feat	Количество каналов для промежуточных функций. По умолчанию: 64.
int	num_grow_ch	Количество каналов для каждого этапа обработки. По умолчанию установлено 32.

Класс RRDB инициализирует входные параметры, веса и значения для сети и блока. Поля класса RRDB представлены в таблице 6.

Таблица 6 – Описание полей класса RRDB

Тип данных	Название	Описание
int	num_feat	Количество каналов для промежуточных функций. По умолчанию: 64.
float	res_scale	Размер разрешения. Значение по умолчанию: 1.

Класс RRDBNet – класс сети, состоящий из блоков Residual in Residual Dense Block. Внутри применяется алгоритм pixel-unshuffle, чтобы уменьшить пространственный размер и увеличить размер канала, прежде чем вводить входные данные в основную архитектуру. Поля класса ResidualBlockNoBN представлены в таблице 7.

Таблица 7 – Описание полей класса ResidualBlockNoBN

Тип данных	Название	Описание
int	num_in_ch	Количество входных каналов.
int	num_out_ch	Количество выходных каналов.
int	num_feat	Количество каналов для промежуточных функций. По умолчанию: 64.
int	num_block	Количество блоков в сети. По умолчанию установлено 23.
int	num_grow_ch	Количество каналов для каждого этапа обработки. По умолчанию установлено 32.

В файле `utils.py` реализованы утилит функций необходимых для работы модели при обработке изображения.

Файл `utils.py` включает в себя 8 функций. Ниже представлены описания функций данного файла.

`def pad_reflect(image, pad_size)` – функция, принимает на вход объект входного изображения и размерность новых границ. Функция возвращает новое изображение с новыми границами.

`def unpad_image(image, pad_size)` – функция, принимает на вход объект входного изображения и размерность новых границ. Функция возвращает новое изображение с новыми границами.

`def process_array(image_array, expand=True)` – функция, принимает на вход массив `image_array`, который хранит в себе пиксели входного изображения и параметр `expand`, который является параметром по умолчанию и определяет, необходимо ли увеличить размерность выходного массива. Внутри функция преобразует трехмерный массив в масштабированный четырехмерный пакет размером равном одному.

`def process_output(output_tensor)` – функция, принимает на вход массив типа `Tensor`. Внутри функция преобразует четырехмерный массив тензоров в подходящий формат изображения и возвращает `sr_image`.

`def pad_patch(image_patch, padding_size, channel_last=True)` – функция, принимает на вход пакет значений изображения, значения пикселей для новых границ изображения. Функция в зависимости от условия дополняет входной пакет значениями краев для увеличенного разрешения изображения

`def unpad_patches(image_patches, padding_size)` – функция, принимает на вход пакет значений изображения, значения пикселей для новых границ изображения. Функция возвращает новый пакет в зависимости от значения параметра `padding_size`.

`def split_image_into_overlapping_patches(image_array, patch_size, padding_size=2)` – функция, принимает на вход . Функция разбивает изображение на частично перекрывающиеся участки. Участки накладываются друг на друга на величину пикселей равную `padding_size`. Функция дважды проходит по изображению, в первый, чтобы установить размер кратному размеру разбиваемого пакета, во второй, чтобы по краям изображения был одинаковый отступ.

`def stitch_together(patches, padded_image_shape, target_shape, padding_size=4)` – функция, принимает на вход `patches` – новые фрагменты, полученные от функции `split_image_into_overlapping_patches`, `padded_image_shape` – новая размерность дополненного изображения полученная от функции `split_image_into_overlapping_patches`, `target_shape` – размерность, которая должна получиться у конечного изображения, `padding_size` – размер области которой не достает. Функция восстанавливает изображение из перекрывающихся фрагментов.

4 Обучение и тестирование модели

4.1 Детали обучения

В соответствии с принятой практикой по исследованию нейросетевых алгоритмов, увеличивающих разрешение, все эксперименты проводятся с коэффициентом масштабирования $\times 4$ между изображениями Low Resolution (LR) и High Resolution (HR). Мы получаем изображения LR путем уменьшения выборки изображений HR с использованием функции бикубического ядра. Размер мини-пакета устанавливается равным 16. Пространственный размер обрезанного участка HR составляет 128×128 . Мы видим, что для обучения более глубокой сети более выгоден больший размер участка, поскольку увеличенное поле восприятия помогает улавливать больше семантической информации на изображении. Однако это требует больше времени на обучение и потребляет больше вычислительных ресурсов. Это явление также наблюдается в методах, ориентированных на PSNR [27].

Процесс обучения разделен на два этапа. Сначала мы обучаем модель, ориентированную на PSNR, с потерей L_1 . Скорость обучения инициализируется как $2 \cdot 10^{-4}$ и уменьшается в 2 раза каждые $2 \cdot 10^5$ мини-пакетных (batch) обновлений. Затем мы используем обученную модель, ориентированную на PSNR, в качестве инициализации генератора. Генератор обучается с использованием функции потерь в уравнении (14) с $\lambda = 5 \cdot 10^{-3}$ и $\eta = 1 \cdot 10^{-2}$:

$$L_G = L_{percep} + \lambda L_G^{Ra} + \eta L_1, \quad (14)$$

где $L_1 = E_{xi} \|G_{(xi)} - y\|_1$ - это потери содержимого, которые оценивают расстояние в 1 единицу между восстановленным изображением $G_{(xi)}$ и исходным y ,

λ, η - коэффициенты, уравнивающие различные потери.

Скорость обучения устанавливается равной $1 \cdot 10^{-4}$ и уменьшается вдвое при [50 тыс., 100 тыс., 200 тыс., 300 тыс.] итераций. Предварительное обучение

с потерей пикселей помогает методам на основе GAN получить более визуально приятные результаты. Причины заключаются в том, что

- это позволяет избежать нежелательных локальных оптимумов для генератора,

- после предварительной подготовки дискриминатор получает относительно хорошие изображения с высоким разрешением, вместо фальшивых (черных или зашумленных изображений) в самом начале, что помогает ему больше сосредоточиться на распознавании текстур и отдельных элементов изображения.

Для оптимизации использовался функция LeakyReLU с входными коэффициентами $\beta_1 = 0,9$, $\beta_2 = 0,999$. Необходимо поочередно обновлять сеть генератора и дискриминатора до тех пор, пока модель не будет сходиться. Использовались две настройки для нашего генератора – одна из них содержит 16 остаточных блоков, а другая - более глубокая модель с 23 блока RRDB. Реализация модели проводилась с помощью фреймворка PyTorch, на видеокарте – NVIDIA 3060TI 8G, процессор – Intel Core i5-12400f.

4.2 Данные для обучения

Для обучения использовался набор данных DIV2K. DIV2K – это набор данных изображений RGB (изображения высокого качества с разрешением 2K) с большим разнообразием содержимого для задач восстановления изображений.

Набор данных DIV2K разделен на:

- данные для обучения: 1600 изображений высокого разрешения и 1600 соответствующих изображений с низким разрешением,

- данные для тестирования: 400 изображений высокого разрешения и 400 соответствующих изображений с низким разрешением.

Все изображения были обработаны с использованием одних и тех же инструментов. Для простоты, поскольку наиболее распространенными коэффициентами увеличения SR являются $\times 2$, $\times 3$ и $\times 4$, изображения обрезаны до кратности 12 пикселей по обеим осям. Большинство просканированных

изображений изначально имели размер более 20 млн пикселей. Изображения имеют высокое качество как с эстетической точки зрения, так и с точки зрения небольшого количества шума и других искажений (таких как размытие и сдвиг цвета). На рисунке 19 представлены примеры изображений из набора данных DIV2K для обучающей модели.

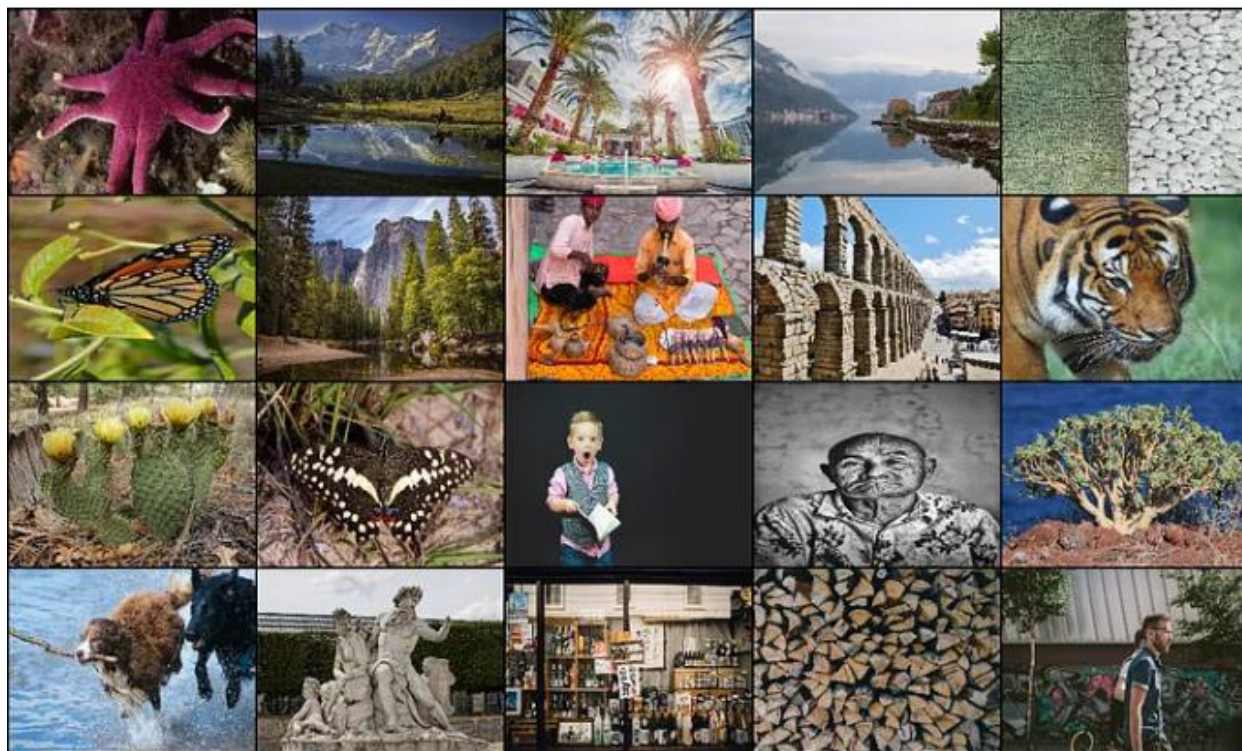


Рисунок 19 – Примеры изображений из DIV2K

Помимо обучающего набора DIV2K, содержащего 1600 изображений, необходимы другие наборы данных с богатыми и разнообразными текстурами для обучения, чтобы максимально исключить сценарии, при которых модель может сработать некорректно или вообще не выполнить необходимое. С этой целью был выбран набор данных Flickr2K, состоящий из 2650 изображений с высоким разрешением 2K, собранных на веб-сайте Flickr2K, и набора данных Outdoor Scene Training (OST) для обогащения деталями обучающего набора. На рисунках 20 и 21 представлены примеры изображений из наборов данных Flickr2K и Outdoor Scene Training (OST) для обучающей модели.



Рисунок 20 – Примеры изображений из Flickr2K



Рисунок 21 – Примеры изображений из OST

4.3 Анализ результатов

Результаты работы, обученной нейросетевой модели на изображениях с низким разрешением. В качестве примера были взяты изображения со следующим разрешением: изображение лица с разрешением 256x256, изображение острова с разрешением 510x339, изображение льва с разрешением 510x339. На рисунках 22-24 представлены данные изображения и их характеристики.

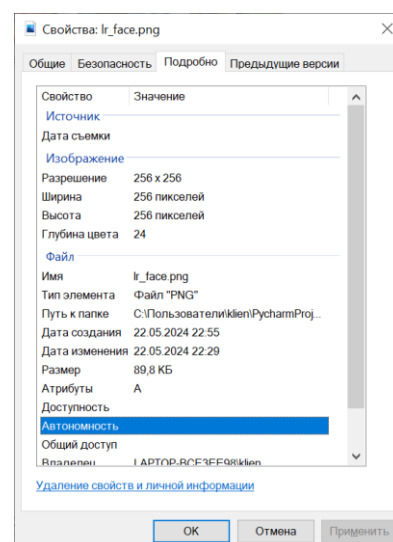


Рисунок 22 – Изображение лица низкого качества

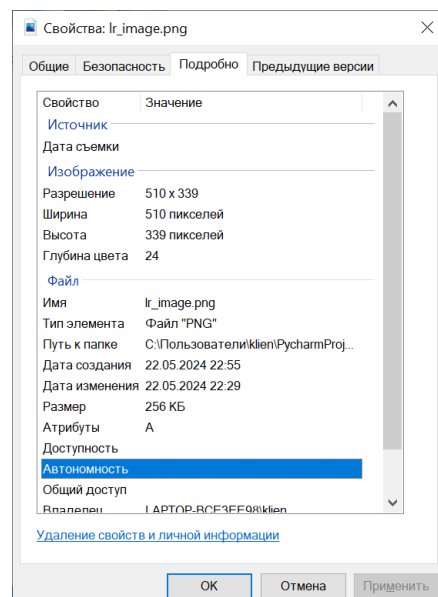


Рисунок 23 – Изображение острова низкого качества

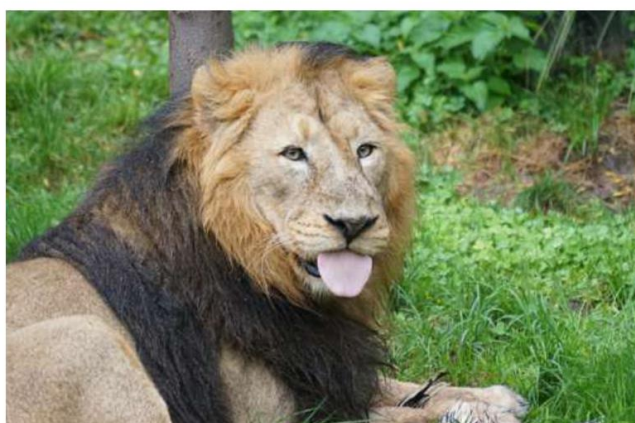
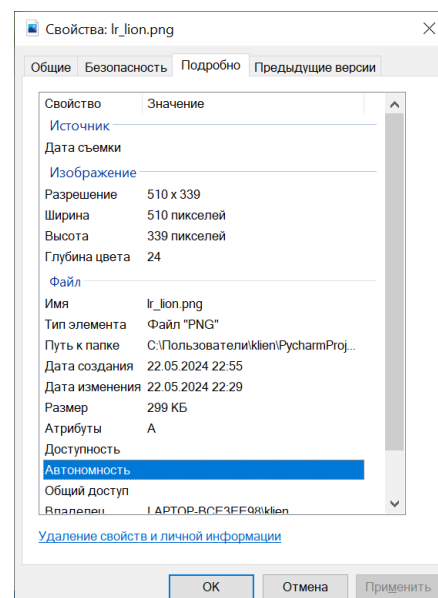


Рисунок 24 – Изображение льва низкого качества

Данные изображения были обработаны через нейронную сеть в разработанном приложении. Можно увидеть, что разрешение изображений было увеличено в четырехкратном размере. Разрешение изображения лица стало 1024x1024, разрешение изображения острова стало 2040x1356, разрешение изображения льва стало 2040x1356. Результаты работы представлены на рисунках 25 – 27.

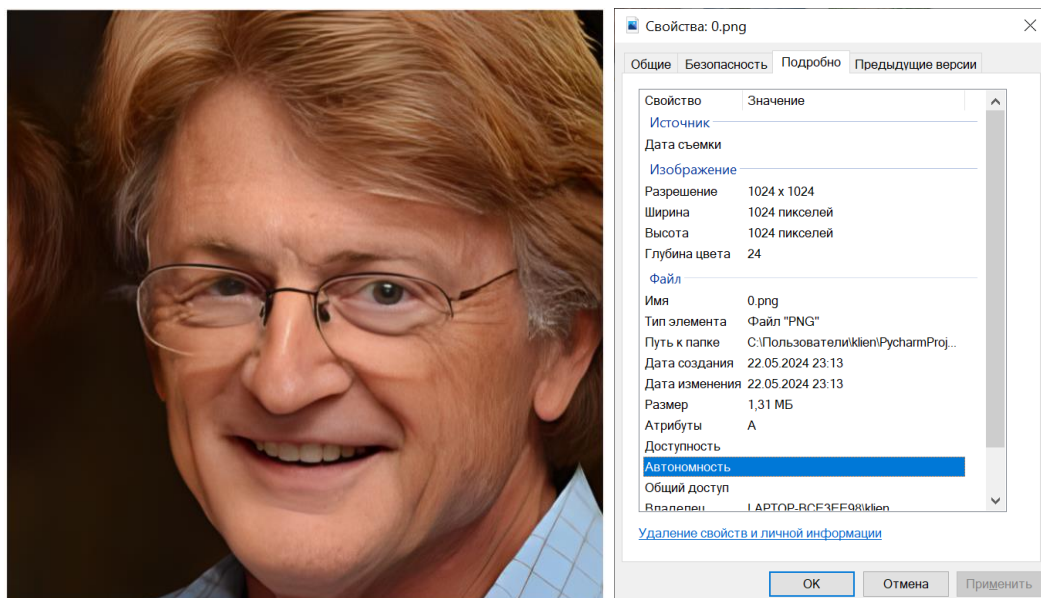


Рисунок 25 – Изображение лица высокого качества

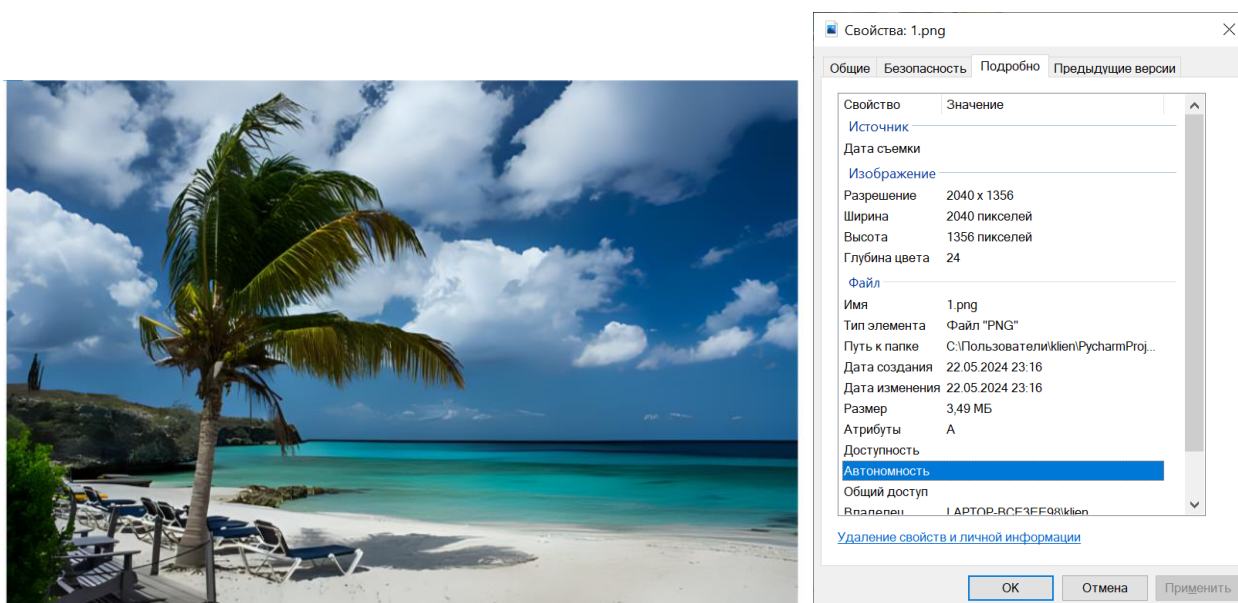


Рисунок 26 – Изображение острова высокого качества

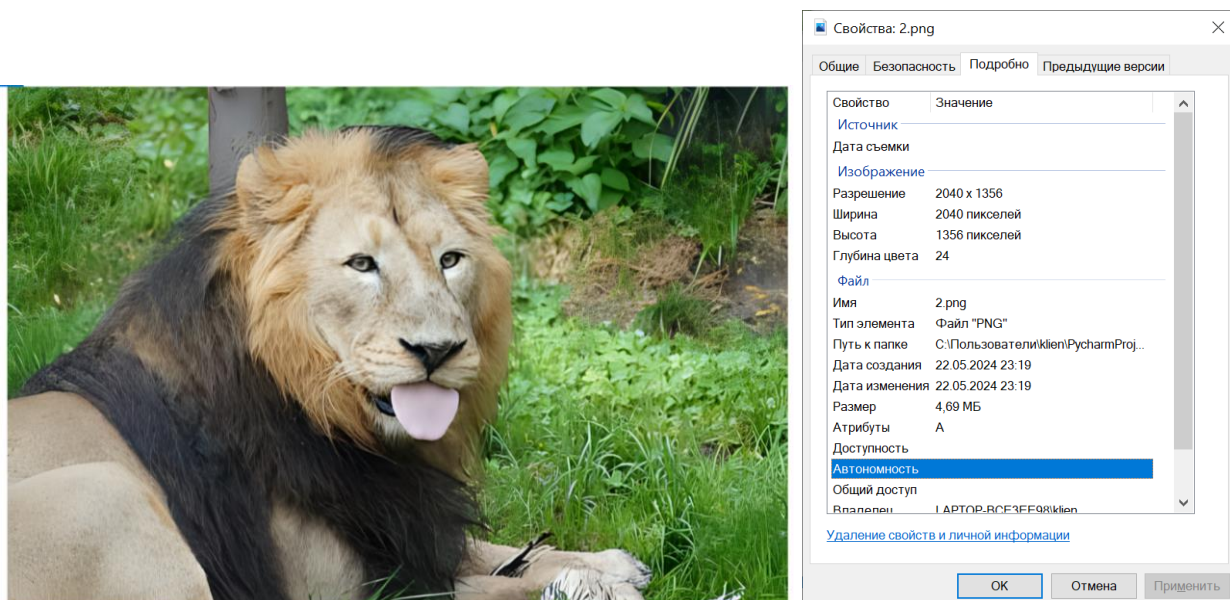


Рисунок 27 – Изображение льва высокого качества

4.4 Сравнение модели

На рисунках показаны качественные результаты детализации и качества изображений на нескольких общедоступных моделях. В качестве моделей были выбраны:

- HR – исходное изображение в высоком качестве,
- Bicubic – бикубическая интерполяция, метод, используемый в обработке изображений и компьютерной графике для изменения размера изображений. Он работает, используя окрестности пикселей 4x4 для оценки значения пикселя в новом месте,
- SRCNN – сверточная нейронная сеть суперразрешения, используется для суперразрешения одиночного изображения,
- EDSR – глубокая остаточная сеть для суперразрешения одиночного приложения, архитектура EDSR основана на архитектуре SRResNet и состоит из нескольких остаточных блоков,
- RCAN – сеть глубоких остаточных каналов, используемая для увеличения разрешения,
- EnhanceNet – модель суперразрешения одиночного изображения посредством автоматического синтеза текста,

– SRGAN – генеративно-состязательная нейронная сеть сверхразрешения.

Сравнение происходит по двум параметрам: PSNR (оценивается по каналу яркости в цветовом пространстве YCbCr) и индекс восприятия (PI).

На рисунках 28-31 представлены результаты изображений обученной модели и моделей, выбранных для сравнения.

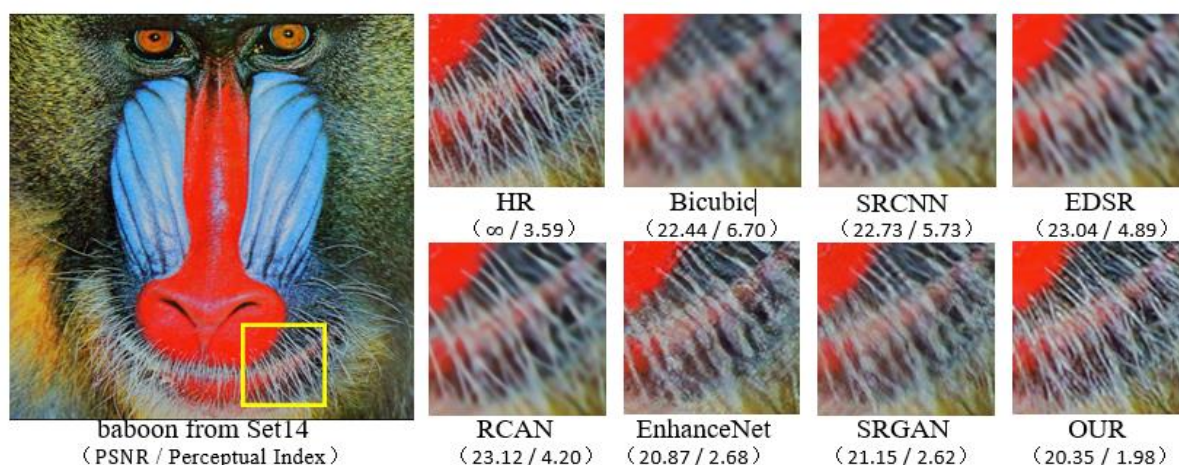


Рисунок 28 – Сравнение полученных результатов на изображении с лицом обезьяны

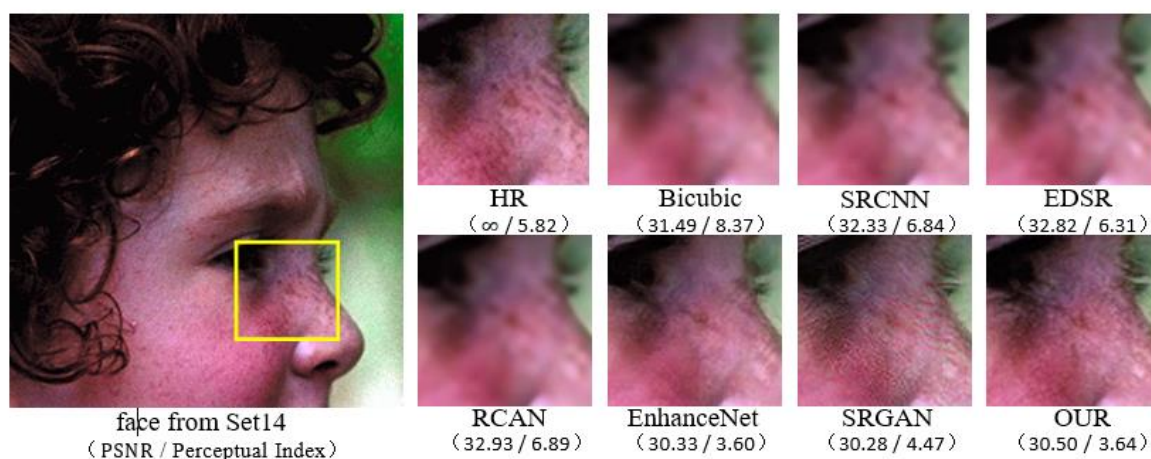


Рисунок 29 – Сравнение полученных результатов на изображении с лицом

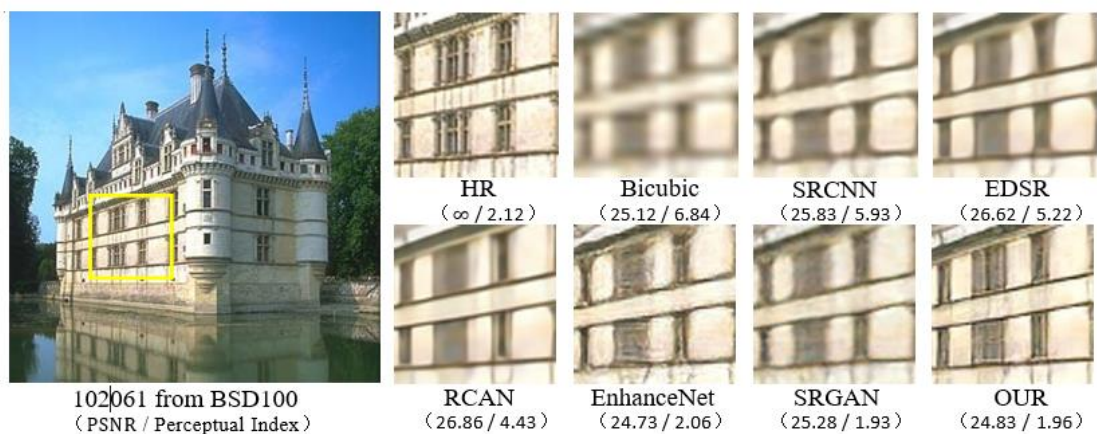


Рисунок 30 – Сравнение полученных результатов на изображении замка

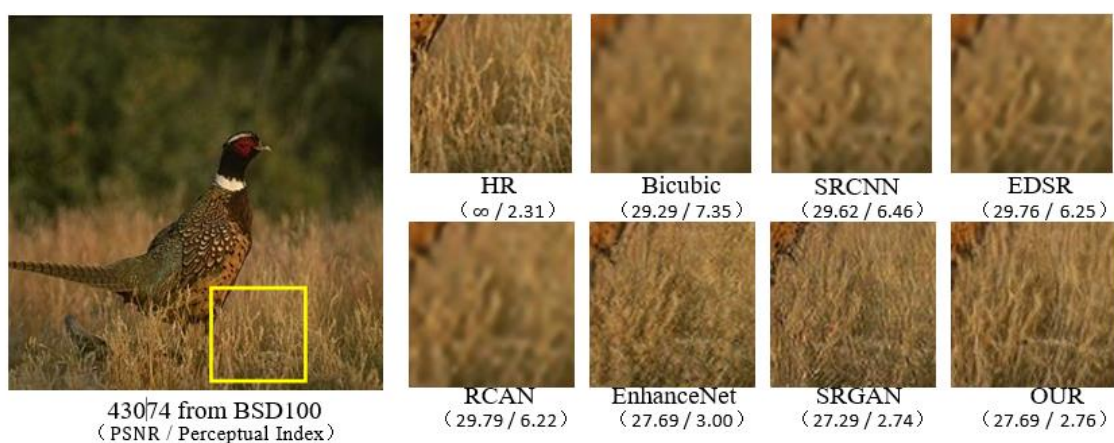


Рисунок 31 – Сравнение полученных результатов на изображении птицы в поле

Таблица 8 – Сравнение результатов по основным критериям PSNR и PI для изображения с лицом обезьяны

	HR	Bicubic	SRCNN	EDSR	RCAN	EnhanceNet	SRGAN	OUR
PSNR	∞	22.44	22.73	23.04	23.12	20.87	21.15	20.35
PI	3.59	6.70	5.73	4.89	4.20	2.68	2.62	1.98

Таблица 9 – Сравнение результатов по основным критериям PSNR и PI для изображения с лицом

	HR	Bicubic	SRCNN	EDSR	RCAN	EnhanceNet	SRGAN	OUR
PSNR	∞	31.49	32.33	32.82	32.93	30.33	30.28	30.50
PI	5.82	8.37	6.84	6.31	6.89	3.60	4.47	3.64

Таблица 10 – Сравнение результатов по основным критериям PSNR и PI для изображения замка

	HR	Bicubic	SRCNN	EDSR	RCAN	EnhanceNet	SRGAN	OUR
PSNR	∞	25.12	25.83	26.62	26.86	24.73	25.28	24.83
PI	2.12	6.84	5.93	5.22	4.43	2.06	1.93	1.96

Таблица 11 – Сравнение результатов по основным критериям PSNR и PI для изображения птицы в поле

	HR	Bicubic	SRCNN	EDSR	RCAN	EnhanceNet	SRGAN	OUR
PSNR	∞	29.29	29.62	29.76	29.79	27.69	27.29	27.69
PI	2.31	7.35	6.46	6.25	6.22	3.00	2.74	2.76

Полученная модель превосходит представленные выше нейросетевые и традиционные методы как по четкости, так и по детализации объектов на изображениях. Например, непосредственно полученная модель на изображениях может формировать более отчетливые и естественные текстуры волосинок обезьяны, веснушек на лице, оконный фасад замка или травы в поле, чем методы, ориентированные на PSNR, которые, как правило, дают размытые результаты, и чем предыдущие методы, основанные на GAN, текстуры которых неестественны и содержат неприятный шум, некоторые из которых даже не могут быть разобраны, различены и рассмотрены на детали. Исходя из полученных результатов видно, что изображения, восстановленные с помощью нейронных сетей, по качеству и близости к оригинальным изображениям превосходят аналогичные, полученные с использованием классических методов для увеличения разрешения. Полученная модель способна создавать более детализированные структуры в зданиях, в то время как другие методы либо не позволяют создать более детализированные структуры в зданиях (SRGAN) или добавляют нежелательные текстуры (EnhanceNet). Более того, предыдущие методы, основанные на GAN, иногда приводили к появлению неприятных артефактов, например, SRGAN добавлял

морщин на лице. За текущей моделью таких дефектов и артефактов не замечено, модель предоставляет естественный результат.

4.5 Сравнение интерполяции

Сравним влияние эффекта использования сетевой интерполяции, использованной в модели, на алгоритме, основанном на GAN и интерполяции изображений ориентированной на PSNR. Для этого применяем простую линейную интерполяцию в обеих схемах [28]. Параметр интерполяции α выбирается в диапазоне от 0 до 1 с интервалом 0,2.

Как показано на рисунке 32, метод, основанный на чистом GAN, создает четкие края и более насыщенные текстуры, но с некоторыми неприятными артефактами, в то время как метод, основанный на чистом PSNR, создает размытые изображения. Использование сетевой интерполяции позволяет уменьшить количество неприятных артефактов при сохранении текстур. Напротив, интерполяция изображений не позволяет эффективно устранить эти артефакты. На рисунке 32 также можно заметить, что стратегия сетевой интерполяции обеспечивает плавный контроль баланса качества восприятия и точности воспроизведения.

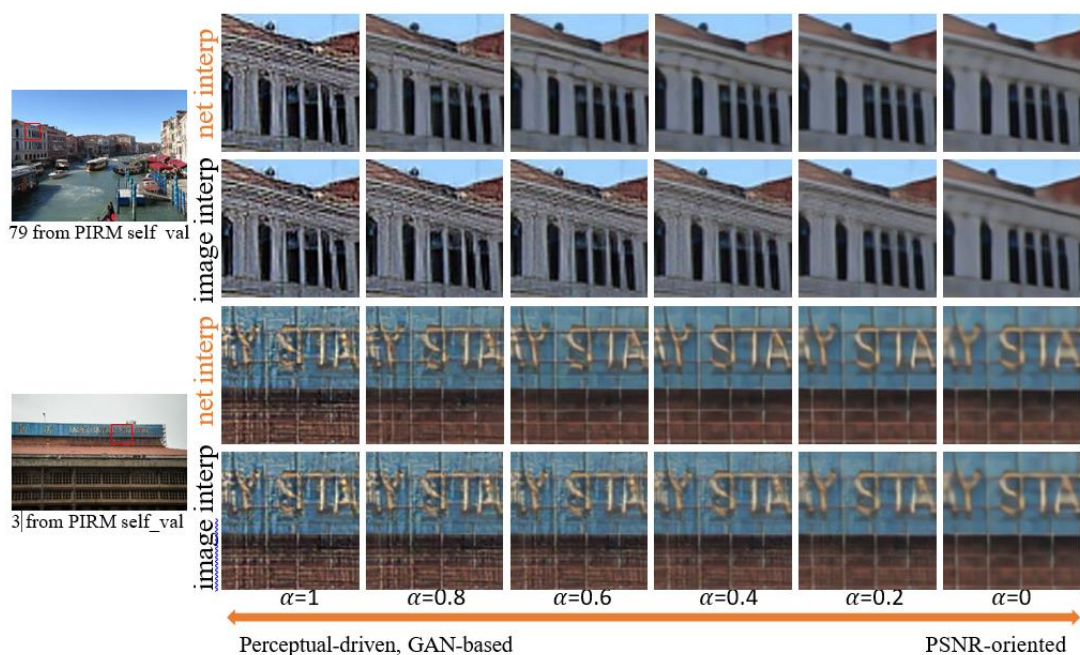


Рисунок 32 – Сравнение результатов интерполяции

5 Проектирование и разработка

5.1 Структура приложения

Разработанный проект содержит несколько каталогов, в которых хранятся: тестовые изображения, изображения после обработки, нейросетевая модель и файлы архитектуры, отвечающие за реализацию и работу нейронной сети. Также в общем каталоге приложения хранятся несколько скриптов, которые реализуют графический интерфейс приложения и функции предобработки. Файловая структура представлена на рисунке 33.

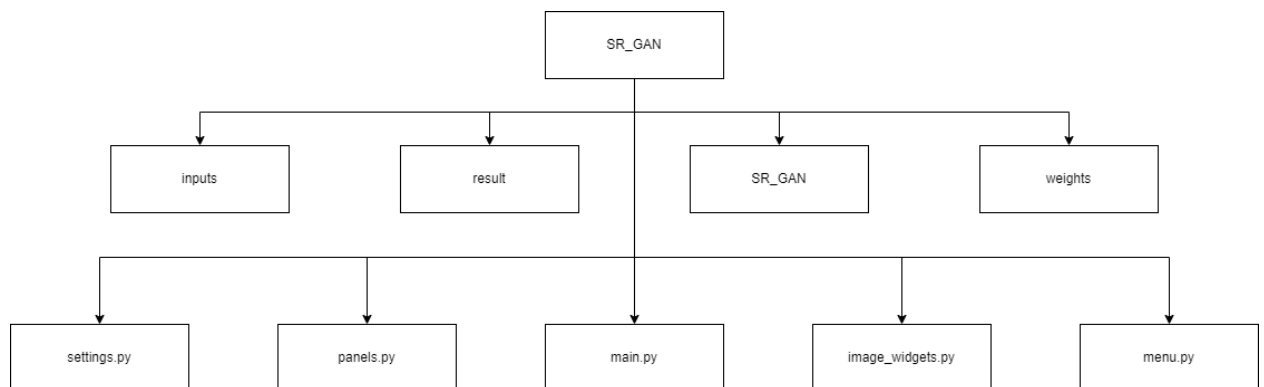


Рисунок 33 – Файловая структура приложения

Каталог `inputs` хранит в себе тестовые изображения, которые можно загрузить с помощью программы, как один из вариантов работы приложения.

В папке `result` хранятся обработанные изображения после увеличения разрешения, данная папка может использоваться как один из вариантов работы приложения.

Директория `SR_GAN` содержит в себе реализацию алгоритма сверхразрешения для изображений, основную архитектуру модели, блоки реализации и функции обработки изображения.

В папке `weights` хранится нейросетевая модель `SR_GAN_x4.pth`.

5.2 Реализация интерфейса пользователя

Графический пользовательский интерфейс реализован на языке Python с помощью библиотеки `CustomTkinter`. Данная библиотека является расширением `tkinter` и предоставляет разработчику дополнительные элементы пользовательского интерфейса, а также позволяет более широко и масштабно

настроить элементы интерфейса, которые будут необходимы разработчику в проекте. Общий вид стартового окна приложения представлен на рисунке 34.

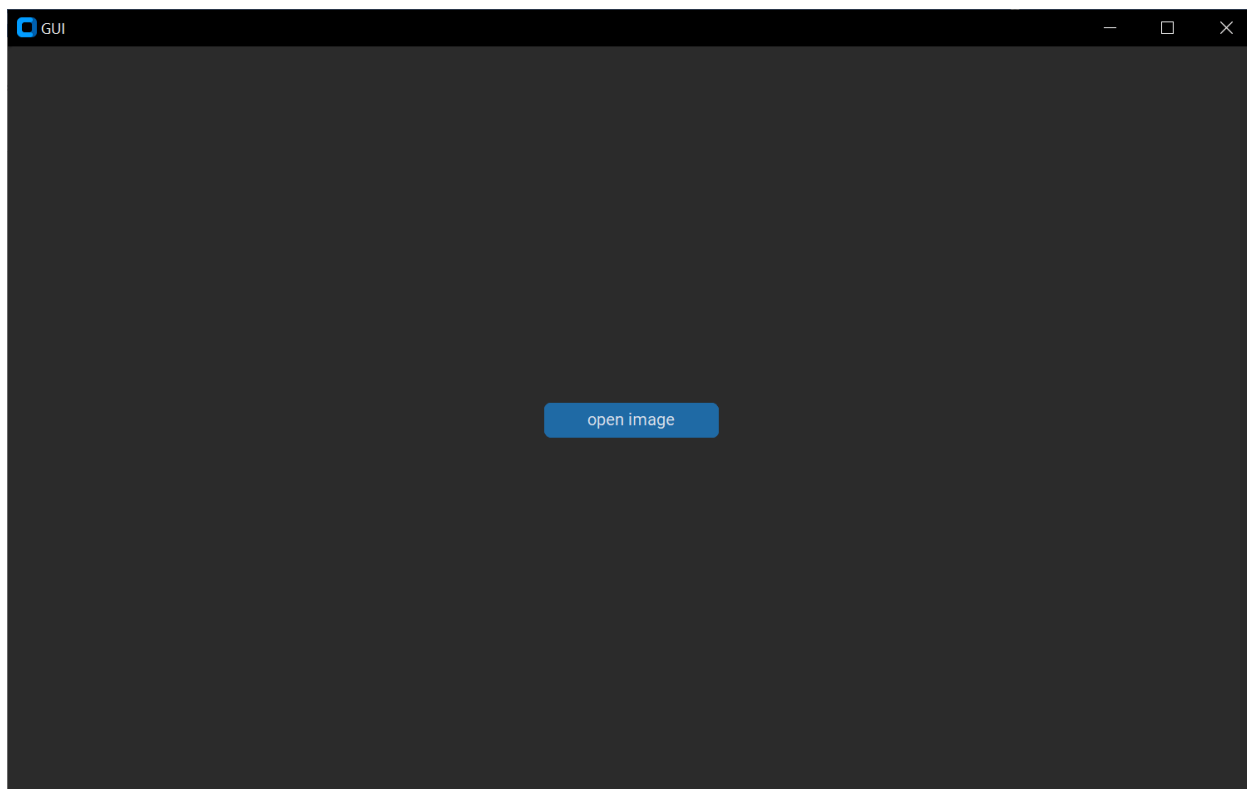


Рисунок 34 – Стартовое окно приложения

При запуске приложения появляется окно с единственной кнопкой `open image`, которое открывает диалоговое окно каталогов для выбора и загрузки изображения, которое необходимо обработать.

После выбора и загрузки изображения появляется основное окно приложения, которое делится на две части, боковое меню панелей, которое включает и реализует все функции приложения, методы предобработки и кнопку действия для запуска обработки изображения с помощью нейросетевой модели. Общий вид основного окна приложения представлен на рисунке 35.

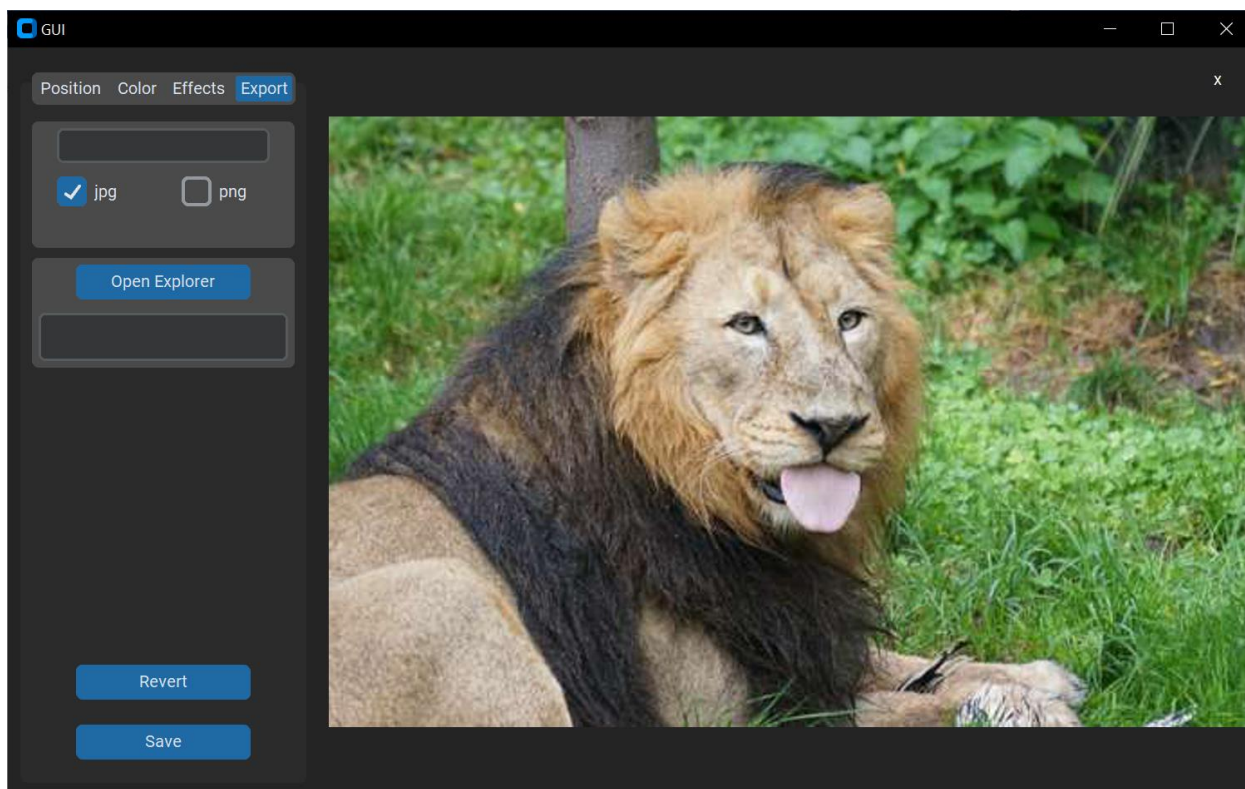


Рисунок 35 – Основное окно приложения

На рисунке 36 представлено меню приложения.

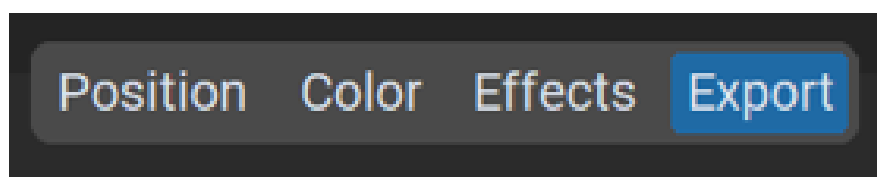


Рисунок 36 – Меню приложения

Меню приложения включает в себя четыре основных вкладки:

- Position – реализовывает функционал обработки изображения, поворот увеличение, а также функции инвертирования изображения, на рисунке 37 представлена данная панель,
- Color – реализовывает функционал работы с цветом изображения, возможность сделать черно-белое изображение, инвертировать цвет, изменить яркость и насыщенность, на рисунке 38 представлена данная панель,
- Effects – реализовывает функционал выпадающего меню для работы с границами изображения, размытием и контрастностью, на рисунке 39 представлена данная панель,

– Export – реализует функционал выбора имени файла для сохранения, тип файла который будет сохранен, а именно jpg или png и путь к каталогу в котором будет сохранено обработанное изображение, на рисунке 40 представлена данная панель.

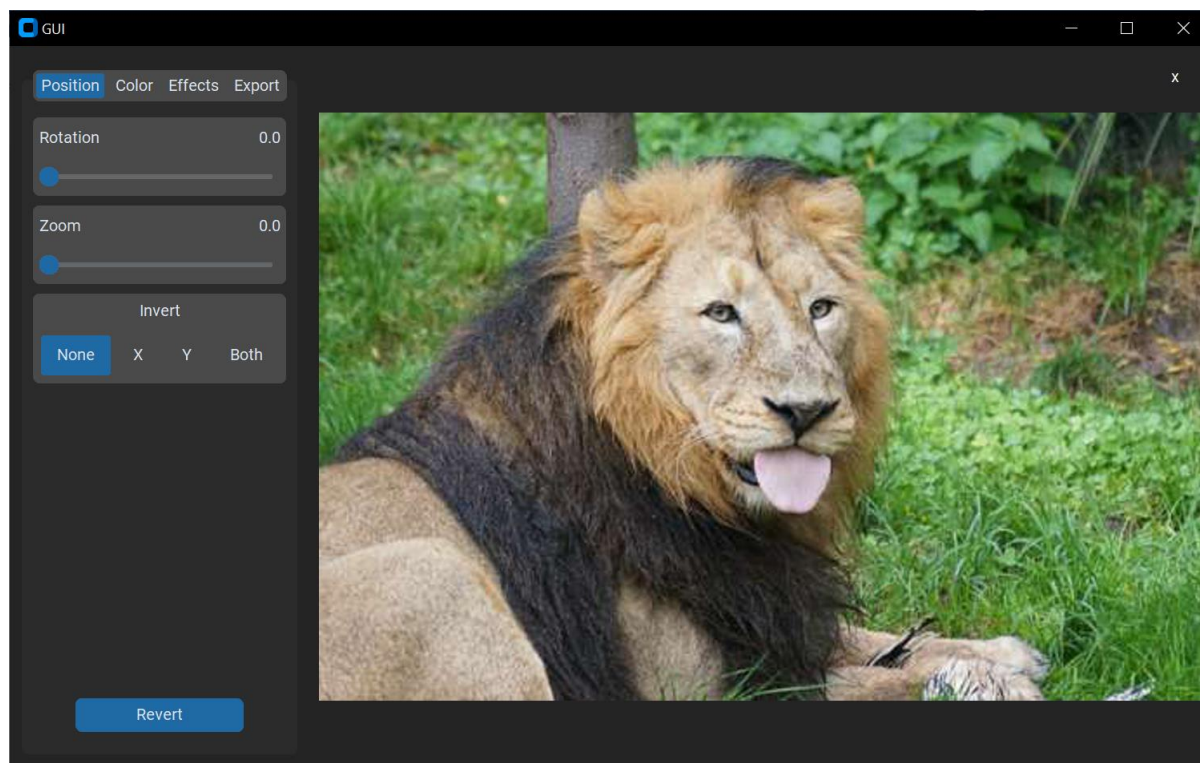


Рисунок 37 – Панель Position

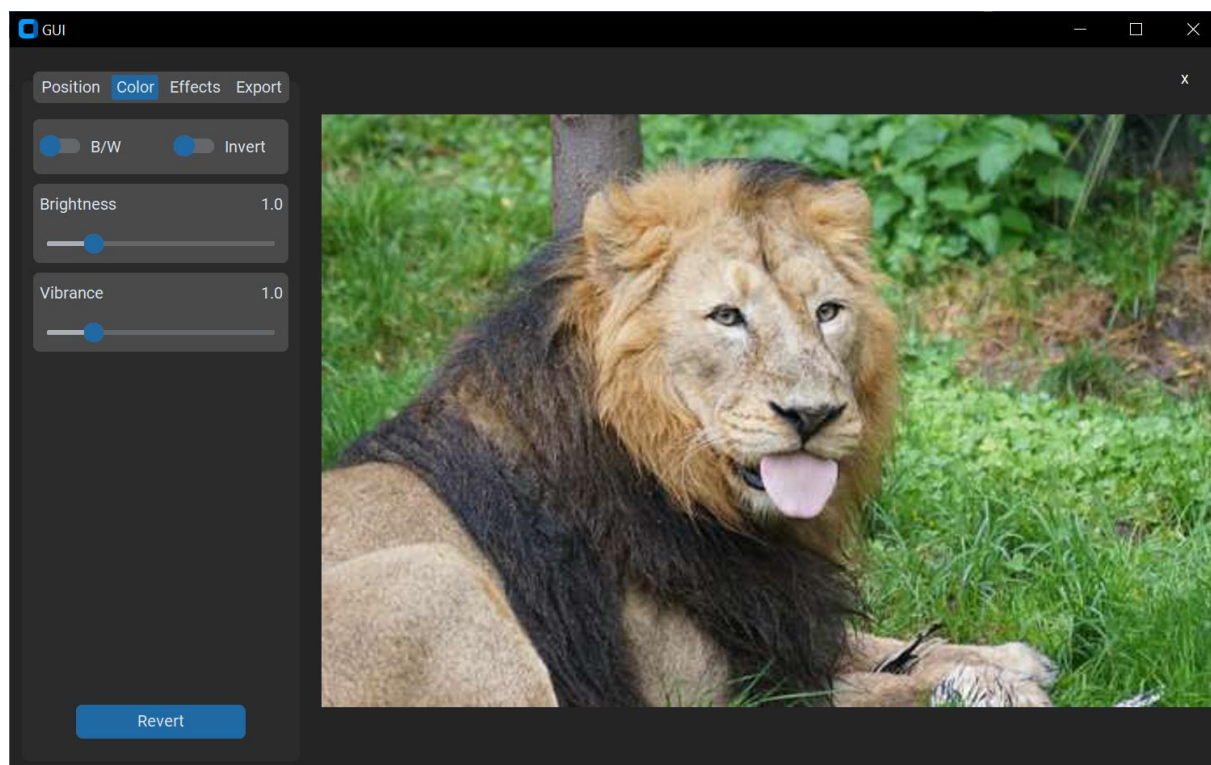


Рисунок 38 – Панель Color

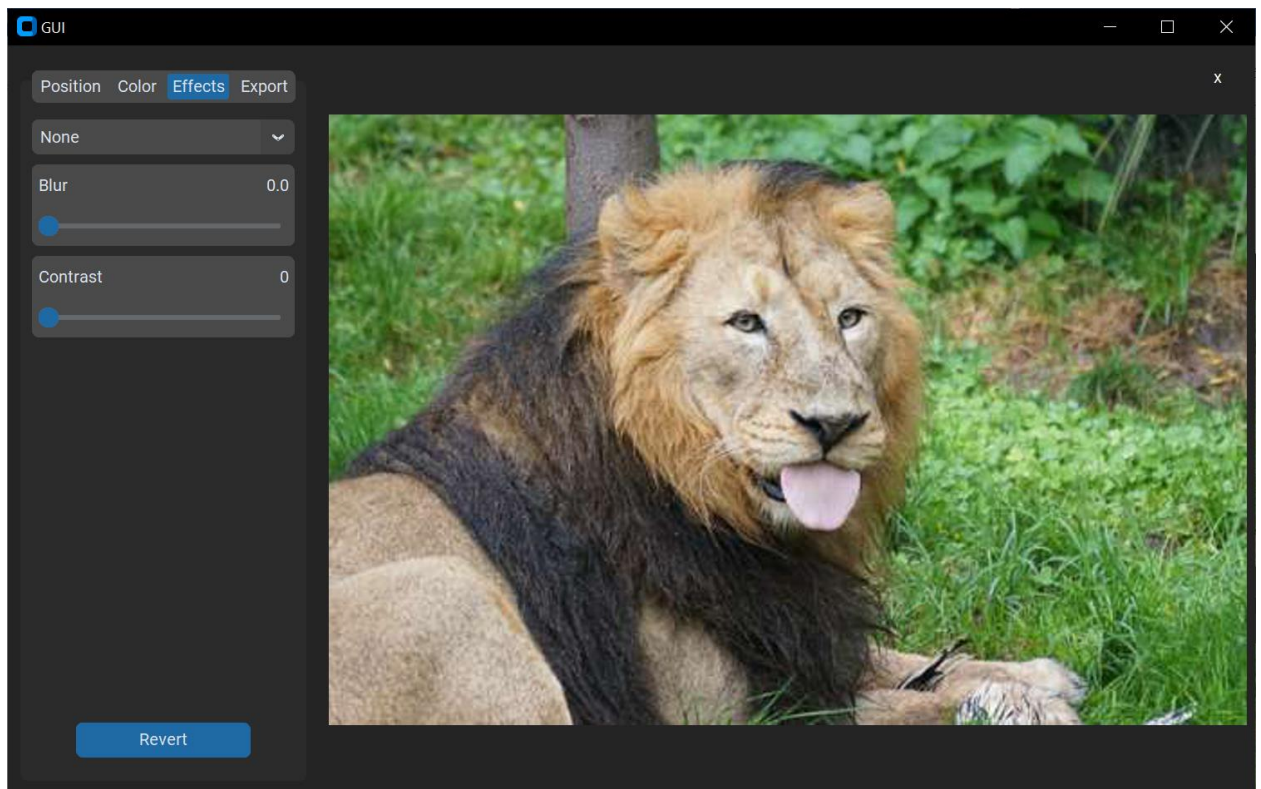


Рисунок 39 – Панель Effects

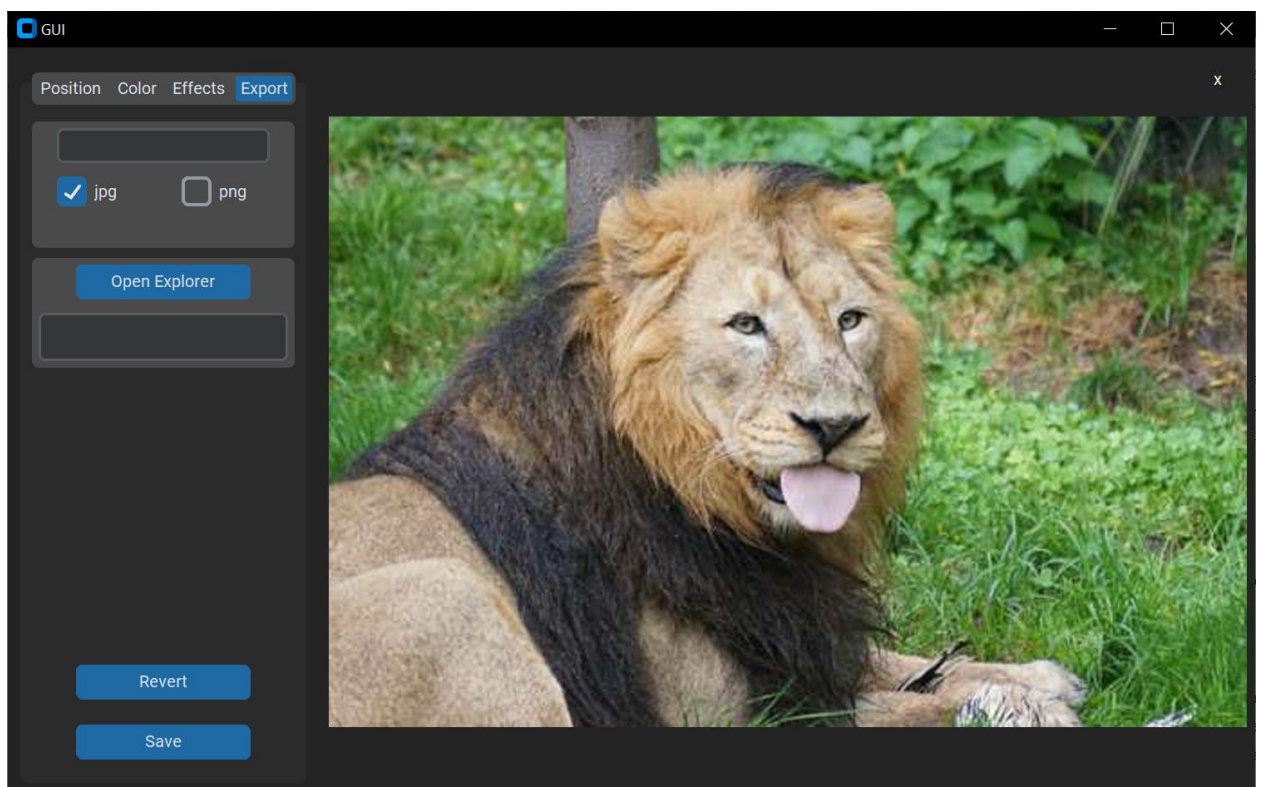


Рисунок 40 – Панель Export

В файле menu.py реализованы классы меню и панелей приложения.

Класс Menu инициализирует входные параметры, устанавливает вкладки меню, а также передает параметры в классы панелей. Поля класса Menu представлены в таблице 12.

Таблица 12 – Описание полей класса Menu

Тип данных	Название	Описание
Menu	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
int array	pos_vars	Массив данных элементов для панели Position
string array	color_vars	Массив данных элементов для панели Color
string array	effect_vars	Массив данных элементов для панели Effects
function	export_image	Экспорт функция
function	resolution_image	Функция для запуска нейросетевой модели.

Класс PositionFrame инициализирует входные параметры, устанавливает и передает параметры в классы для элементов обработчики текущей панели. Поля класса PositionFrame представлены в таблице 13.

Таблица 13 – Описание полей класса PositionFrame

Тип данных	Название	Описание
PositionFrame	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
int array	pos_vars	Массив данных элементов для панели Position

На рисунке 41 показаны пункты панели для вкладки меню Position.

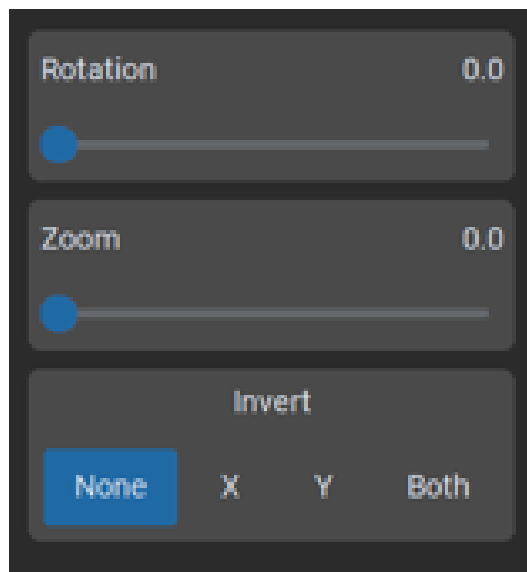


Рисунок 41 – Пункты меню Position

Функции инвертирования:

- None – отменить ранее примененное инвертирование изображения,
- X – инвертировать изображение по оси X,
- Y – инвертировать изображение по оси Y,
- Both – инвертировать изображение по оси X и Y вместе.

Класс ColorFrame инициализирует входные параметры, устанавливает и передает параметры в классы для элементов обработки текущей панели. Поля класса ColorFrame представлены в таблице 14.

Таблица 14 – Описание полей класса ColorFrame

Тип данных	Название	Описание
ColorFrame	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
string array	color_vars	Массив данных элементов для панели Position

Класс EffectFrame инициализирует входные параметры, устанавливает и передает параметры в классы для элементов обработки текущей панели. Поля класса EffectFrame представлены в таблице 15.

Таблица 15 – Описание полей класса EffectFrame

Тип данных	Название	Описание
EffectFrame	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
string array	effect_vars	Массив данных элементов для панели Position

На рисунке 42 показаны пункты панели для вкладки меню Effects.

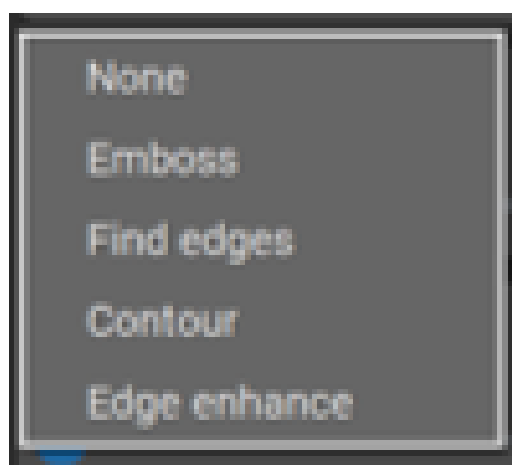


Рисунок 42 – Пункты меню панели Effects

Функции инвертирования:

- None – отменить ранее примененные эффекты к изображению,
- Emboss – выделить рельеф элементов изображения,
- Find edges – поиск краев изображения,
- Contour – выделить контур элементов изображения,
- Edge enhance – обработать края изображения.

Класс ExportFrame инициализирует входные параметры, устанавливает и передает параметры в классы для элементов обработчика текущей панели. Поля класса ExportFrame представлены в таблице 16.

Таблица 16 – Описание полей класса ExportFrame

Тип данных	Название	Описание
ExportFrame	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
function	export_image	Экспорт функция
function	resolution_image	Функция для запуска нейросетевой модели.

В файле panels.py реализованы классы панелей и элементов приложения.

Класс Panel инициализирует входные параметры для основного класса панели. Поля класса Panel представлены в таблице 17.

Таблица 17 – Описание полей класса Panel

Тип данных	Название	Описание
Panel	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса

Класс SliderPanel инициализирует входные параметры для элемента слайдера, устанавливает размер, начальное положение, возможные варианты.

Поля класса SliderPanel представлены в таблице 18.

Таблица 18 – Описание полей класса SliderPanel

Тип данных	Название	Описание
SliderPanel	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
string	text	Название функционального элемента
int array	data_var	Массив данных для панели
int	min_value	Минимальное значение слайдера

Тип данных	Название	Описание
int	max_value	Максимальное значение слайдера

Класс SegmentedPanel инициализирует входные параметры для элемента, устанавливает размер, начальное положение, возможные варианты. Поля класса SegmentedPanel представлены в таблице 19.

Таблица 19 – Описание полей класса SegmentedPanel

Тип данных	Название	Описание
SegmentedPanel	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
string	text	Название функционального элемента
int array	data_var	Массив данных для панели
string array	options	Варианты сегментной панели

Класс SwitchPanel инициализирует входные параметры для элемента переключателя, устанавливает размер, начальное положение, возможные варианты. Поля класса SwitchPanel представлены в таблице 20.

Таблица 20 – Описание полей класса SwitchPanel

Тип данных	Название	Описание
SwitchPanel	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
array	*args	Массив аргументов класса

Класс FileNamePanel инициализирует входные параметры, устанавливает размер, начальное положение, возможные варианты элемента, тип и поведение для каждого из возможных видов изображения, jpg или png. Поля класса FileNamePanel представлены в таблице 21.

Таблица 21 – Описание полей класса FileNamePanel

Тип данных	Название	Описание
FileNamePanel	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
string	name_string	Название файла изображения, которое будет экспортировано
string	file_string	Тип файла

Класс FilePathPanel инициализирует входные параметры для элемента, устанавливает размер, начальное положение, возможные варианты, открывает диалоговое окно для выбора каталога и файла. Поля класса FilePathPanel представлены в таблице 22.

Таблица 22 – Описание полей класса FilePathPanel

Тип данных	Название	Описание
FilePathPanel	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
string	path_string	Путь файла в каталоге

Класс DropDownPanel инициализирует входные параметры для элемента, выпадающего меню, устанавливает размер, начальное положение, возможные варианты. Также устанавливает элементы меню, порядок, исходный элемент. Поля класса DropDownPanel представлены в таблице 23.

Таблица 23 – Описание полей класса DropDownPanel

Тип данных	Название	Описание
DropDownPanel	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса

Тип данных	Название	Описание
int array	data_var	Массив данных для панели
string array	options	Варианты выпадающего меню

Класс RevertButton инициализирует входные параметры для элемента кнопки, устанавливает размер, начальное положение, возможные варианты. Данная кнопка появляется во всех видах панелей меню, отвечает за сброс и установку в начальное положение всех элементов на текущей панели. Поля класса RevertButton представлены в таблице 24.

Таблица 24 – Описание полей класса RevertButton

Тип данных	Название	Описание
RevertButton	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
array	*args	Массив аргументов класса

Класс SaveButton инициализирует входные параметры для элемента кнопки, устанавливает размер, начальное положение, возможные варианты. Данная кнопка отображается в панели Export, сохраняет изображение под выбранным форматом с установленным именем файла и выбранной директорией, после всей обработки изображения. Поля класса SaveButton представлены в таблице 25.

Таблица 25 – Описание полей класса SaveButton

Тип данных	Название	Описание
SaveButton	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
function	export_image	Экспорт функция

Тип данных	Название	Описание
string	name_string	Название файла изображения, которое будет экспортировано
string	file_string	Тип файла
string	path_string	Путь файла в каталоге

Класс ResolutionButton инициализирует входные параметры, устанавливает размер, начальное положение, возможные варианты. Отвечает за вызов функции resolution_image, которая применяет нейросетевую модель для увеличения разрешения изображения. Поля класса ResolutionButton представлены в таблице 26.

Таблица 26 – Описание полей класса ResolutionButton

Тип данных	Название	Описание
ResolutionButton	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
function	resolution_image	Функция для запуска нейросетевой модели обработки изображения

В файле settings.py описаны константные переменные, которые используются в других классах приложения. В основном тут описаны цвета элементов графического интерфейса, массивы переменных и вариантов элементов.

В файле image_widgets.py реализованы виджеты приложения.

Класс ImageImport инициализирует входные параметры для Элемента кнопки, которая открывает диалоговое окно. Поля класса ImageImport представлены в таблице 27.

Таблица 27 – Описание полей класса ImageImport

Тип данных	Название	Описание
Panel	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
function	import_fun	Принимает путь файла

Класс ImageOutput инициализирует входные параметры для элемента, устанавливает размер, начальное положение изображения, которое было выбрано и загружено из каталога. Поля класса ImageOutput представлены в таблице 28.

Таблица 28 – Описание полей класса ImageOutput

Тип данных	Название	Описание
SliderPanel	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса
function	resize_image	Изменение размеров изображения

Класс CloseOutput инициализирует входные параметры для элемента кнопки, устанавливает размер, начальное положение для кнопки, которая может закрыть текущее окно с изображением и вернуться в стартовое окно для выбора нового. Поля класса CloseOutput представлены в таблице 29.

Таблица 29 – Описание полей класса CloseOutput

Тип данных	Название	Описание
SegmentedPanel	self	Экземпляр текущего класса, необходимый для обращения к переменным и методам
widget object	parent	Родительский элемент класса

Тип данных	Название	Описание
function	close_func	Функция закрытия основного окна, переход в стартовое окно, очистка параметров

5.3 Реализация функций предобработки

В файле `main.py` реализован класс `App`, который содержит в себе функции предобработки и утилитарные функции, для обработки изображения.

Класс `App.py` включает в себя 9 функций. Ниже представлены описания функций данного класса.

`def __init__(self)` – базовая метод класса, принимает на вход контекст экземпляра класса, который может обращаться к переменным и методам класса. Инициализирует и задает стандарты и параметры основного окна пользовательского приложения, размеры меню и поле обработки изображения.

`def init_parameters(self)` – метод, принимает на вход контекст экземпляра класса, задает переменные для параметров и элементов панели в качестве объекта.

`def manipulate_image(self, *args)` – метод, принимающий на вход массив аргументов. Метод, в зависимости от принимаемого параметра, предоставляет набор инструкций для обработки изображения для каждого виджета или панели в меню.

`def import_image(self, path)` – метод, принимает на вход путь к исходному изображению, где с помощью функции `open`, получает объект `Image`, который хранит информацию и параметры изображения, вызывает и передает параметры в класс `Menu`.

`def resize_image(self, event)` – метод, принимает на вход `event`, который информирует текущий размер поле изображения и обновляет размеры под новое или изменённое изображение.

`def place_image(self)` – метод, устанавливает изображение в ранее выбранную область на графическом интерфейсе, вызывает метод `resize_image`.

`def close_edit(self)` – метод, функция закрывает текущее окно и переходит к стартовому окну для нового выбора изображения, очищает ранее установленные параметры для панелей и виджетов.

`def export_image(self, name, file, path)` – метод, принимает на вход имя экспортируемого файла, тип файла, который может быть видом jpg или png, и путь к каталогу, по которому будет сохранено обработанное изображение.

`def resolution_image(self)` – метод, отвечающий за обработку нейросетевой моделью входное изображение. Обращается к исходному изображению, устанавливает device для работы модели в один из вариантов, cuda или cpu, импортирует модель SR_GAN и запускает алгоритм для увеличения разрешения изображения, в зависимости от возможного варианта работы приложения может принимать в массив данных из папки inputs, для обработки и последующего сохранения в results.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы было исследовано и проанализировано использование искусственного интеллекта для обработки изображений и улучшения их качества посредством увеличения разрешения изображения, проанализированы существующие классические и нейросетевые методы решения данной проблемы, выявлены их преимущества и недостатки.

В результате выполнения данной выпускной квалификационной работы были достигнуты следующие результаты:

- проанализированы существующие нейросетевые модели, способные обрабатывать изображения и увеличивать их разрешение, сравнены и выявлены сильные и слабые стороны текущих решений,
- разработана нейросетевая модель для увеличения разрешения изображения,
- произведено обучение разработанной модели для увеличения разрешения изображения,
- реализовано тестирование и сравнение полученной модели с аналогами,
- разработан функционал приложения, направленного на обработку изображений и увеличение его качества,
- реализовано тестирование и отладка разработанного приложения,
- показана применимость разработанной нейросетевой модели и приложения с использованием различных входных данных.

Определена актуальность данной темы и поставлены цели выполнения для дальнейшего выполнения научно-исследовательской работы. Также была проанализирована архитектура нейросетевой модели из семейства ESRGAN. Была реализована, разработана и обучена нейросетевая модель для решения задачи увеличения разрешения и обработки качества изображений. Изучены и добавлены методы оптимизации алгоритма. Нейросетевая модель была протестирована на корректность обработки, было реализовано сравнение с

похожими нейросетевыми методами и моделями, сделаны выводы по уровню и качеству работы.

Дальнейшее развитие нейросетевых решений для проблемы улучшения качества изображений посредством увеличения разрешения изображения может включать в себя расширение возможностей нейросетевой модели и приложения следующим образом:

- исследованием новых архитектур нейронных сетей для улучшения качества изображений,
- развитие потенциала для увеличения скорости обработки графических изображений и оптимизации использования ресурсов устройств,
- расширение функционала системы, добавление возможности обработки видео и других типов графических и медиафайлов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Афанасьев, А. А. Цифровая обработка сигналов / А. А. Афанасьев, А. А. Рыболовлев, А. П. Рыжков. – М.: Горячая линия – Телеком, 2017.
2. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. – М.: Техносфера, 2012. – 1104 с.
3. Wang, Z., Image quality assessment: From error visibility to structural similarity / Z Wang [et. all] // IEEE Trans. Image Processing. – 2004. –V. 13, № 4, April. – P. 600–612.
4. Xu, L. Visual Quality Assessment by Machine Learning / L. Xu, W. Lin, C. Kuo. Springer, 2015. Wang, Z. Modern Image Quality Assessment / Z. Wang, A. Bovik. Morgan and Claypool Publishing Company, 2006.
5. Макаров А.О. Алгоритмы увеличения пространственного разрешения и обработки мультиспектральных спутниковых изображений: А.О. Макаров. – Минск, 2006. – 156 с.
6. Xintao Wang, Liangbin Xie, Chao Dong, Ying Shan. / Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data.
7. Wang, Xintao, et al. “Esrgan: Enhanced super-resolution generative adversarial networks.” Proceedings of the European Conference on Computer Vision (ECCV) Workshops. 2018.
8. Ledig, Christian, et al. “Photo-realistic single image super-resolution using a generative adversarial network.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
9. Jolicœur-Martineau, Alexia. “The relativistic discriminator: a key element missing from standard GAN.” *arXiv preprint arXiv: 1807.00734* (2018).
10. Lim, Bee, et al. “Enhanced deep residual networks for single image super-resolution.” Proceedings of the IEEE conference on computer vision and pattern recognition workshops. 2017.
11. CAMBRIDGE in COLOUR. A learning community for photographers. Интерполяция цифрового изображения.

12. Тихонов, В. И. Статический анализ и синтез радиотехнических устройств и систем связи / В. И. Тихонов, В. Н. Харисов. – М.: Радио и связь, 2004. – 608 с.
13. Ключко, В. К. Методы восстановления изображений и оценивания аппаратной функции по прореженной матрице наблюдений / В. К. Ключко, В. П. Кузнецов // Автометрия. – 2016. – Т. 52. – №6. – С. 12–20. – DOI: 10.15372/AUT20160602.
14. Zhang, J. Quantile analysis of image sensor noise distribution / J. Zhang, K. Hirakawa, X. Jin // ICASSP. – 2015. – DOI: 10.1109/ICASSP.2015.7178240.
15. Xiao-Jiao, M. Image Restoration Using Very Deep Convolutional Encoder-Decoder Networks with Symmetric Skip Connections / M. Xiao-Jiao, S. Chunhua, Y. Yubin // NIPS. – 2016. 8. Dey, B. SEM image denoising with Unsupervised Machine Learning for better defect inspection and metrology / B. Dey, S. Haldera, K. Khalil // SPIE Advanced Lithography. – 2021. – DOI: 10.1117/12.2584803.
16. Wang, Z. Defect simulation in SEM images using generative adversarial networks / Z. Wang, Y. Liangjiang, P. Lingling // SPIE Advanced Lithography. – 2021. – DOI: 10.1117/12.2581881.
17. Huang, X. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization / X. Huang, S. Belongie // arXiv: 1703.06868. – 2017.
18. Lim, B., Son, S., Kim, H., Nah, S., Lee, K.M.: Enhanced deep residual networks for single image super-resolution. In: CVPRW. (2017).
19. Nah, S., Kim, T.H., Lee, K.M.: Deep multi-scale convolutional neural network for dynamic scene deblurring. In: CVPR. (2017)
20. Mittal, A., Soundararajan, R., Bovik, A.C.: Making a completely blind image quality analyzer. IEEE Signal Process. Lett. 20(3) (2013) 209–212.
21. Zhang, K., Sun, M., Han, X., Yuan, X., Guo, L., Liu, T.: Residual networks of residual networks: Multilevel residual networks. IEEE Transactions on Circuits and Systems for Video Technology (2017).

22. Wang, X., Yu, K., Dong, C., Loy, C.C.: Recovering realistic texture in image super-resolution by deep spatial feature transform. In: CVPR. (2018).
23. Haris, M., Shakhnarovich, G., Ukita, N.: Deep backprojection networks for super- resolution. In: CVPR. (2018).
24. Timofte, R., Rothe, R., Van Gool, L.: Seven ways to improve example-based single image super resolution. In: CVPR. (2016).
25. Jolicoeur-Martineau, A.: The relativistic discriminator: a key element missing from standard gan. arXiv preprint arXiv:1807.00734 (2018).
26. Milukova, O. Image Restoration Spectral Techniques/ O. Milukova, V. Kober, I. A. Ovseevich // PRIP. – 2009. – 4 p.
27. . Huynh-Th u, Q. Scope of validity of psnr in image/video quality assessment / Q. Huynh-Th u and M. Ghanbari // Electronics letters. – 2013. – № 44 – P. 800–801. – DOI: 10.1049/el: 20080522.
28. Vincent Feng. An Overview of ResNet and its Variants.

Приложение А

(Обязательное)

Нейросетевые алгоритмы улучшения качества изображений

Листинг программы

Файл main.py

```
import os
import torch
from RealeSRGAN import RealeSRGAN
from image_widgets import *
from menu import Menu
from PIL import Image, ImageTk, ImageOps, ImageEnhance,
ImageFilter
import customtkinter as ctk

class App(ctk.CTk):
    def __init__(self):
        # setup
        super().__init__()
        ctk.set_appearance_mode('dark')
        self.geometry('1000x600')
        self.title('GUI')
        self.minsize(800, 500)
        self.init_parameters()

        # layout
        self.rowconfigure(0, weight=1)
        self.columnconfigure(0, weight=2, uniform='a')
        self.columnconfigure(1, weight=6, uniform='a')

        # canvas data
        self.image_width = 0
        self.image_height = 0
        self.canvas_width = 0
        self.canvas_height = 0

        # widgets
        self.image_import = ImageImport(self, self.import_image)

        # run
        self.mainloop()

    def init_parameters(self):
        self.pos_vars = {
            'rotate': ctk.DoubleVar(value=ROTATE_DEFAULT),
            'zoom': ctk.DoubleVar(value=ZOOM_DEFAULT),
            'flip': ctk.StringVar(value=FLIP_OPTIONS[0])
        }

        self.color_vars = {
            'brightness':
                ctk.DoubleVar(value=BRIGHTNESS_DEFAULT),
            'grayscale': ctk.DoubleVar(value=GRAYSCALE_DEFAULT),
            'invert': ctk.DoubleVar(value=INVERT_DEFAULT),
            'vibrance': ctk.DoubleVar(value=VIBRANCE_DEFAULT)
```

```

    }

    self.effect_vars = {
        'blur': ctk.DoubleVar(value=BLUR_DEFAULT),
        'contrast': ctk.IntVar(value=CONTRAST_DEFAULT),
        'effect': ctk.StringVar(value=EFFECT_OPTIONS[0])
    }

    combined_vars = list(self.pos_vars.values()) +
list(self.color_vars.values()) + list(self.effect_vars.values())

    # tracing
    for var in combined_vars:
        var.trace('w', self.manipulate_image)

    def manipulate_image(self, *args):
        self.image = self.original

        # rotate
        if self.pos_vars['rotate'].get() != ROTATE_DEFAULT:
            self.image =
self.image.rotate(self.pos_vars['rotate'].get())

        # zoom
        if self.pos_vars['zoom'].get():
            self.image = ImageOps.crop(image=self.image,
border=int(self.pos_vars['zoom'].get()))

        # flip
        if self.pos_vars['flip'].get() != FLIP_OPTIONS:
            if self.pos_vars['flip'].get() == 'X':
                self.image = ImageOps.mirror(self.image)
            if self.pos_vars['flip'].get() == 'Y':
                self.image = ImageOps.flip(self.image)
            if self.pos_vars['flip'].get() == 'Both':
                self.image = ImageOps.mirror(self.image)
                self.image = ImageOps.flip(self.image)

        # brightness
        if self.color_vars['brightness'].get() !=
BRIGHTNESS_DEFAULT:
            brightness_enhancer =
ImageEnhance.Brightness(self.image)
            self.image =
brightness_enhancer.enhance(self.color_vars['brightness'].get())

        # vibrance
        if self.color_vars['vibrance'].get() !=
VIBRANCE_DEFAULT:
            vibrance_enhancer = ImageEnhance.Color(self.image)
            self.image =
vibrance_enhancer.enhance(self.color_vars['vibrance'].get())

```

```

        # grayscale and invert of the colors
        if self.color_vars['grayscale'].get():
            self.image = ImageOps.grayscale(self.image)

        if self.color_vars['invert'].get():
            self.image = ImageOps.invert(self.image)

        # blur and contrast
        if self.effect_vars['blur'].get() != BLUR_DEFAULT:
            self.image =
self.image.filter(ImageFilter.GaussianBlur(self.effect_vars['blu
r'].get()))

        if self.effect_vars['contrast'].get() !=
CONTRAST_DEFAULT:
            self.image =
self.image.filter(ImageFilter.UnsharpMask(self.effect_vars['cont
rast'].get()))

        match self.effect_vars['effect'].get():
            case 'Emboss':
                self.image =
self.image.filter(ImageFilter.EMBOSS)
            case 'Find edges':
                self.image =
self.image.filter(ImageFilter.FIND_EDGES)
            case 'Contour':
                self.image =
self.image.filter(ImageFilter.CONTOUR)
            case 'Edge enhance':
                self.image =
self.image.filter(ImageFilter.EDGE_ENHANCE)

        self.place_image()

    def import_image(self, path):
        self.original = Image.open(path)
        self.image = self.original
        self.image_ratio = self.image.size[0] /
self.image.size[1]
        self.image_tk = ImageTk.PhotoImage(self.image)

        self.image_import.grid_forget()
        self.image_output = ImageOutput(self, self.resize_image)
        self.close_button = CloseOutput(self, self.close_edit)
        self.menu = Menu(
            self,
            self.pos_vars,
            self.color_vars,
            self.effect_vars,
            self.export_image,
            self.resolution_image
        )

```

```

def resize_image(self, event):
    # current canvas ratio
    canvas_ratio = event.width / event.height

    # update canvas attributes
    self.canvas_width = event.width
    self.canvas_height = event.height

    # resize
    if canvas_ratio > self.image_ratio: # canvas is wider
than the image
        self.image_height = int(event.height)
        self.image_width = int(self.image_height *
self.image_ratio)
    else: # canvas is taller than the image
        self.image_width = int(event.width)
        self.image_height = int(self.image_width /
self.image_ratio)

    self.place_image()

def place_image(self):
    self.image_output.delete('all')
    resized_image = self.image.resize((self.image_width,
self.image_height))
    self.image_tk = ImageTk.PhotoImage(resized_image)
    self.image_output.create_image(self.canvas_width / 2,
self.canvas_height / 2, image=self.image_tk)

def close_edit(self):
    # hide all superfluous
    self.image_output.grid_forget()
    self.close_button.place_forget()
    self.menu.grid_forget()
    # recreate the import button
    self.image_import = ImageImport(self, self.import_image)

def export_image(self, name, file, path):
    export_string = f'{path}/{name}.{file}'
    self.image.save(export_string)

def resolution_image(self):
    self.new_image = self.original
    print(self.new_image)

    device = torch.device('cuda' if
torch.cuda.is_available() else 'cpu')
    model = RealESRGAN(device, scale=4)
    model.load_weights('weights/RealESRGAN_x4.pth')

    image = self.new_image.convert('RGB')
    sr_image = model.predict(image)

```

```

sr_image.save(f'results/123.png')

# for i, image in enumerate(os.listdir("inputs")):
#     image =
Image.open(f"inputs/{image}").convert('RGB')
#     sr_image = model.predict(image)
#     sr_image.save(f'results/{i}.png')

App()

```

Файл меню.py

```

from panels import *
import customtkinter as ctk

class Menu(ctk.CTkTabview):
    def __init__(self, parent, pos_vars, color_vars,
effect_vars, export_image, resolution_image):
        super().__init__(master=parent)
        self.grid(row=0, column=0, sticky='nsew', pady=10,
padx=10)

        # tabs
        self.add('Position')
        self.add('Color')
        self.add('Effects')
        self.add('Export')
        # self.add('AI-Res')

        # widgets
        PositionFrame(self.tab('Position'), pos_vars)
        ColorFrame(self.tab('Color'), color_vars)
        EffectFrame(self.tab('Effects'), effect_vars)
        ExportFrame(self.tab('Export'), export_image,
resolution_image)
        # AIResFrame(self.tab('Export'))

class PositionFrame(ctk.CTkFrame):
    def __init__(self, parent, pos_vars):
        super().__init__(master=parent, fg_color='transparent')
        self.pack(expand=True, fill='both')

        SliderPanel(self, 'Rotation', pos_vars['rotate'], 0,
360)
        SliderPanel(self, 'Zoom', pos_vars['zoom'], 0, 200)
        SegmentedPanel(self, 'Invert', pos_vars['flip'],
FLIP_OPTIONS)
        RevertButton(
            self,
            (pos_vars['rotate'], ROTATE_DEFAULT),

```

```

        (pos_vars['zoom'], ZOOM_DEFAULT),
        (pos_vars['flip'], FLIP_OPTIONS[0]),
    )

class ColorFrame(ctk.CTkFrame):
    def __init__(self, parent, color_vars):
        super().__init__(master=parent, fg_color='transparent')
        self.pack(expand=True, fill='both')

        SwitchPanel(self, (color_vars['grayscale'], 'B/W'),
(color_vars['invert'], 'Invert'))
        SliderPanel(self, 'Brightness',
color_vars['brightness'], 0, 5)
        SliderPanel(self, 'Vibrance', color_vars['vibrance'], 0,
5)
        RevertButton(
            self,
            (color_vars['brightness'], BRIGHTNESS_DEFAULT),
            (color_vars['vibrance'], VIBRANCE_DEFAULT),
            (color_vars['grayscale'], GRAYSCALE_DEFAULT),
            (color_vars['invert'], INVERT_DEFAULT),
        )

class EffectFrame(ctk.CTkFrame):
    def __init__(self, parent, effect_vars):
        super().__init__(master=parent, fg_color='transparent')
        self.pack(expand=True, fill='both')

        DropDownPanel(self, effect_vars['effect'],
EFFECT_OPTIONS)
        SliderPanel(self, 'Blur', effect_vars['blur'], 0, 30)
        SliderPanel(self, 'Contrast', effect_vars['contrast'],
0, 10)
        RevertButton(
            self,
            (effect_vars['effect'], EFFECT_OPTIONS[0]),
            (effect_vars['blur'], BLUR_DEFAULT),
            (effect_vars['contrast'], CONTRAST_DEFAULT),
        )

class ExportFrame(ctk.CTkFrame):
    def __init__(self, parent, export_image, resolution_image):
        super().__init__(master=parent, fg_color='transparent')
        self.pack(expand=True, fill='both')

        # data
        self.name_string = ctk.StringVar()
        self.file_string = ctk.StringVar(value='jpg')
        self.path_string = ctk.StringVar()

```

```

        # widgets
        FileNamePanel(self, self.name_string, self.file_string)
        FilePathPanel(self, self.path_string)
        SaveButton(self, export_image, self.name_string,
self.file_string, self.path_string)
        ResolutionButton(self, resolution_image)

```

Файл panels.py

```

import customtkinter as ctk
from tkinter import filedialog
from settings import *

```

```

class Panel(ctk.CTkFrame):
    def __init__(self, parent):
        super().__init__(master=parent, fg_color=DARK_GREY)
        self.pack(fill='x', pady=4, ipady=8)

class SliderPanel(Panel):
    def __init__(self, parent, text, data_var, min_value,
max_value):
        super().__init__(parent=parent)

        # layout
        self.rowconfigure((0, 1), weight=1)
        self.columnconfigure((0, 1), weight=1)

        self.data_var = data_var
        self.data_var.trace('w', self.update_text)

        ctk.CTkLabel(self, text=text).grid(column=0, row=0,
sticky='W', padx=5)
        self.num_label = ctk.CTkLabel(self, text=data_var.get())
        self.num_label.grid(column=1, row=0, sticky='E', padx=5)

        ctk.CTkSlider(
            self,
            fg_color=SLIDER_BG,
            variable=self.data_var,
            from_=min_value,
            to=max_value,
        ).grid(
            row=1,
            column=0,
            columnspan=2,
            sticky='ew',
            padx=5,
            pady=5
        )

    def update_text(self, *args):

```

```
self.num_label.configure(text=f'{round(self.data_var.get(),
2)}')
```

```
class SegmentedPanel(Panel):
    def __init__(self, parent, text, data_var, options):
        super().__init__(parent=parent)

        ctk.CTkLabel(self, text=text).pack()
        ctk.CTkSegmentedButton(
            self,
            variable=data_var,
            values=options
        ).pack(
            expand=True,
            fill='both',
            padx=4,
            pady=4
        )
```

```
class SwitchPanel(Panel):
    def __init__(self, parent, *args):
        super().__init__(parent=parent)

        for var, text in args:
            switch = ctk.CTkSwitch(self, text=text,
variable=var, button_color=BLUE, fg_color=SLIDER_BG)
            switch.pack(side='left', expand=True, fill='both',
padx=5, pady=5)
```

```
class FileNamePanel(Panel):
    def __init__(self, parent, name_string, file_string):
        super().__init__(parent=parent)

        # data
        self.name_string = name_string
        self.name_string.trace('w', self.update_text)
        self.file_string = file_string

        # check boxes for file format
        ctk.CTkEntry(self,
textvariable=self.name_string).pack(fill='x', padx=20, pady=5)
        frame = ctk.CTkFrame(self, fg_color='transparent')

        jpg_check = ctk.CTkCheckBox(
            frame,
            text='jpg',
            variable=self.file_string,
            onvalue='jpg',
            offvalue='png',
```

```

        command=lambda: self.click('jpg')
    )
    jpg_check.pack(side='left', fill='x', expand=True)

    png_check = ctk.CTkCheckBox(
        frame,
        text='png',
        variable=self.file_string,
        onvalue='png',
        offvalue='jpg',
        command=lambda: self.click('png')
    )
    png_check.pack(side='left', fill='x', expand=True)

    frame.pack(expand=True, fill='x', padx=20)

    # preview text
    self.output = ctk.CTkLabel(self, text=' ')
    self.output.pack()

    def click(self, value):
        self.file_string.set(value)
        self.update_text()

    def update_text(self, *args):
        if self.name_string.get():
            text = self.name_string.get().replace(' ', '_') +
            '.' + self.file_string.get()
            self.output.configure(text=text)

class FilePathPanel(Panel):
    def __init__(self, parent, path_string):
        super().__init__(parent=parent)
        self.path_string = path_string

        ctk.CTkButton(self, text='Open Explorer',
            command=self.open_file_dialog).pack(pady=5)
        ctk.CTkEntry(self,
            textvariable=self.path_string).pack(expand=True, fill='both',
            padx=5, pady=5)

    def open_file_dialog(self):
        self.path_string.set(filedialog.askdirectory())

class DropDownPanel(ctk.CTkOptionMenu):
    def __init__(self, parent, data_var, options):
        super().__init__(
            master=parent,
            values=options,
            fg_color=DARK_GREY,
            button_color=DROPDOWN_MAIN_COLOR,

```

```

        button_hover_color=DROPDOWN_HOVER_COLOR,
        dropdown_fg_color=DROPDOWN_MENU_COLOR
    )
    self.pack(fill='x', pady=4)

class RevertButton(ctk.CTkButton):
    def __init__(self, parent, *args):
        super().__init__(master=parent, text='Revert',
command=self.revert)
        self.pack(side='bottom', pady=10)
        self.args = args

    def revert(self):
        for var, value in self.args:
            var.set(value)

class SaveButton(ctk.CTkButton):
    def __init__(self, parent, export_image, name_string,
file_string, path_string):
        super().__init__(master=parent, text='Save',
command=self.save)
        self.pack(side='bottom', pady=10)

        self.export_image = export_image
        self.name_string = name_string
        self.file_string = file_string
        self.path_string = path_string

    def save(self):
        self.export_image(
            self.name_string.get(),
            self.file_string.get(),
            self.path_string.get(),
        )

class ResolutionButton(ctk.CTkButton):
    def __init__(self, parent, resolution_image):
        super().__init__(master=parent, text='Resolution',
command=self.resolution_up)
        self.pack(side='bottom', pady=10)

        self.resolution_image = resolution_image

    def resolution_up(self):
        self.resolution_image()

```

Файл image_widgets.py

```
import customtkinter as ctk
from tkinter import filedialog, Canvas
from settings import *

class ImageImport(ctk.CTkFrame):
    def __init__(self, parent, import_func):
        super().__init__(master=parent)
        self.grid(column=0, columnspan=2, row=0, sticky='nsew')
        self.import_func = import_func

        ctk.CTkButton(self, text='open image',
            command=self.open_dialog).pack(expand=True)

    def open_dialog(self):
        path = filedialog.askopenfile().name
        self.import_func(path)

class ImageOutput(Canvas):
    def __init__(self, parent, resize_image):
        super().__init__(master=parent,
            background=BACKGROUND_COLOR, bd=0, highlightthickness=0,
            relief='ridge')
        self.grid(row=0, column=1, sticky='nsew', padx=10,
            pady=10)
        self.bind('<Configure>', resize_image)

class CloseOutput(ctk.CTkButton):
    def __init__(self, parent, close_func):
        super().__init__(
            master=parent,
            command=close_func,
            text='x',
            text_color=WHITE,
            fg_color='transparent',
            width=40,
            height=40,
            corner_radius=0,
            hover_color=CLOSE_RED
        )
        self.place(relx=0.99, rely=0.01, anchor='ne')
```

Файл settings.py

```
# default values
ROTATE_DEFAULT = 0
ZOOM_DEFAULT = 0
FLIP_OPTIONS = ['None', 'X', 'Y', 'Both']
```

```

BLUR_DEFAULT = 0
CONTRAST_DEFAULT = 0
EFFECT_OPTIONS = ['None', 'Emboss', 'Find edges', 'Contour',
'Edge enhance']
BRIGHTNESS_DEFAULT = 1
VIBRANCE_DEFAULT = 1
GRAYSCALE_DEFAULT = False
INVERT_DEFAULT = False

# Colors
BACKGROUND_COLOR = '#242424'
WHITE = '#FFF'
GREY = 'grey'
BLUE = '#1f6aa5'
DARK_GREY = '#4a4a4a'
CLOSE_RED = '#8a0606'
SLIDER_BG = '#64686b'

DROPDOWN_MAIN_COLOR = '#444'
DROPDOWN_HOVER_COLOR = '#333'
DROPDOWN_MENU_COLOR = '#666'

```

Файл arch_utils.py

```

import math
import torch
from torch import nn as nn
from torch.nn import functional as F
from torch.nn import init as init
from torch.nn.modules.batchnorm import _BatchNorm

@torch.no_grad()
def default_init_weights(module_list, scale=1, bias_fill=0,
**kwargs):
    if not isinstance(module_list, list):
        module_list = [module_list]
    for module in module_list:
        for m in module.modules():
            if isinstance(m, nn.Conv2d):
                init.kaiming_normal_(m.weight, **kwargs)
                m.weight.data *= scale
                if m.bias is not None:
                    m.bias.data.fill_(bias_fill)
            elif isinstance(m, nn.Linear):
                init.kaiming_normal_(m.weight, **kwargs)
                m.weight.data *= scale
                if m.bias is not None:
                    m.bias.data.fill_(bias_fill)
            elif isinstance(m, _BatchNorm):
                init.constant_(m.weight, 1)
                if m.bias is not None:
                    m.bias.data.fill_(bias_fill)

```

```

def make_layer(basic_block, num_basic_block, **kwarg):
    layers = []
    for _ in range(num_basic_block):
        layers.append(basic_block(**kwarg))
    return nn.Sequential(*layers)

class ResidualBlockNoBN(nn.Module):
    def __init__(self, num_feat=64, res_scale=1,
pytorch_init=False):
        super(ResidualBlockNoBN, self).__init__()
        self.res_scale = res_scale
        self.conv1 = nn.Conv2d(num_feat, num_feat, 3, 1, 1,
bias=True)
        self.conv2 = nn.Conv2d(num_feat, num_feat, 3, 1, 1,
bias=True)
        self.relu = nn.ReLU(inplace=True)

        if not pytorch_init:
            default_init_weights([self.conv1, self.conv2], 0.1)

    def forward(self, x):
        identity = x
        out = self.conv2(self.relu(self.conv1(x)))
        return identity + out * self.res_scale

class Upsample(nn.Sequential):
    def __init__(self, scale, num_feat):
        m = []
        if (scale & (scale - 1)) == 0: # scale = 2^n
            for _ in range(int(math.log(scale, 2))):
                m.append(nn.Conv2d(num_feat, 4 * num_feat, 3, 1,
1))
                m.append(nn.PixelShuffle(2))
        elif scale == 3:
            m.append(nn.Conv2d(num_feat, 9 * num_feat, 3, 1, 1))
            m.append(nn.PixelShuffle(3))
        else:
            raise ValueError(f'scale {scale} is not supported. '
'Supported scales: 2^n and 3.')
        super(Upsample, self).__init__(*m)

def flow_warp(x, flow, interp_mode='bilinear',
padding_mode='zeros', align_corners=True):
    assert x.size()[-2:] == flow.size()[1:3]
    _, _, h, w = x.size()
    # create mesh grid
    grid_y, grid_x = torch.meshgrid(torch.arange(0,
h).type_as(x), torch.arange(0, w).type_as(x))

```

```

        grid = torch.stack((grid_x, grid_y), 2).float() # W(x),
H(y), 2
        grid.requires_grad = False

        vgrid = grid + flow
        # scale grid to [-1,1]
        vgrid_x = 2.0 * vgrid[:, :, :, 0] / max(w - 1, 1) - 1.0
        vgrid_y = 2.0 * vgrid[:, :, :, 1] / max(h - 1, 1) - 1.0
        vgrid_scaled = torch.stack((vgrid_x, vgrid_y), dim=3)
        output = F.grid_sample(x, vgrid_scaled, mode=interp_mode,
padding_mode=padding_mode, align_corners=align_corners)

        return output

def resize_flow(flow, size_type, sizes, interp_mode='bilinear',
align_corners=False):
    _, _, flow_h, flow_w = flow.size()
    if size_type == 'ratio':
        output_h, output_w = int(flow_h * sizes[0]), int(flow_w
* sizes[1])
    elif size_type == 'shape':
        output_h, output_w = sizes[0], sizes[1]
    else:
        raise ValueError(f'Size type should be ratio or shape,
but got type {size_type}.')

    input_flow = flow.clone()
    ratio_h = output_h / flow_h
    ratio_w = output_w / flow_w
    input_flow[:, 0, :, :] *= ratio_w
    input_flow[:, 1, :, :] *= ratio_h
    resized_flow = F.interpolate(
        input=input_flow, size=(output_h, output_w),
mode=interp_mode, align_corners=align_corners)
    return resized_flow

def pixel_unshuffle(x, scale):
    b, c, hh, hw = x.size()
    out_channel = c * (scale ** 2)
    assert hh % scale == 0 and hw % scale == 0
    h = hh // scale
    w = hw // scale
    x_view = x.view(b, c, h, scale, w, scale)
    return x_view.permute(0, 1, 3, 5, 2, 4).reshape(b,
out_channel, h, w)

```

Файл model.py

```

import os
import torch
from torch.nn import functional as F

```

```

from PIL import Image
import numpy as np
import cv2
from huggingface_hub import hf_hub_url, cached_download

from .rrdbnet_arch import RRDBNet
from .utils import pad_reflect,
split_image_into_overlapping_patches, stich_together, \
    unpad_image

class RealESRGAN:
    def __init__(self, device, scale=4):
        self.device = device
        self.scale = scale
        self.model = RRDBNet(
            num_in_ch=3, num_out_ch=3, num_feat=64,
            num_block=23, num_grow_ch=32, scale=scale
        )

    def load_weights(self, model_path, download=True):
        if not os.path.exists(model_path) and download:
            assert self.scale in [2, 4, 8], 'You can download
models only with scales: 2, 4, 8'
            config = HF_MODELS[self.scale]
            cache_dir = os.path.dirname(model_path)
            local_filename = os.path.basename(model_path)
            config_file_url =
hf_hub_url(repo_id=config['repo_id'],
filename=config['filename'])
            cached_download(config_file_url,
cache_dir=cache_dir, force_filename=local_filename)
            print('Weights downloaded to:',
os.path.join(cache_dir, local_filename))

            loadnet = torch.load(model_path)
            if 'params' in loadnet:
                self.model.load_state_dict(loadnet['params'],
strict=True)
            elif 'params_ema' in loadnet:
                self.model.load_state_dict(loadnet['params_ema'],
strict=True)
            else:
                self.model.load_state_dict(loadnet, strict=True)
                self.model.eval()
                self.model.to(self.device)

    @torch.cuda.amp.autocast()
    def predict(self, lr_image, batch_size=4, patches_size=192,
                padding=24, pad_size=15):
        scale = self.scale
        device = self.device
        lr_image = np.array(lr_image)

```

```

        lr_image = pad_reflect(lr_image, pad_size)

        patches, p_shape = split_image_into_overlapping_patches(
            lr_image, patch_size=patches_size,
            padding_size=padding
        )
        img = torch.FloatTensor(patches / 255).permute((0, 3, 1,
2)).to(device).detach()

        with torch.no_grad():
            res = self.model(img[0:batch_size])
            for i in range(batch_size, img.shape[0],
batch_size):
                res = torch.cat((res, self.model(img[i:i +
batch_size])), 0)

            sr_image = res.permute((0, 2, 3, 1)).clamp_(0, 1).cpu()
            np_sr_image = sr_image.numpy()

            padded_size_scaled = tuple(np.multiply(p_shape[0:2],
scale)) + (3,)
            scaled_image_shape =
tuple(np.multiply(lr_image.shape[0:2], scale)) + (3,)
            np_sr_image = stich_together(
                np_sr_image, padded_image_shape=padded_size_scaled,
                target_shape=scaled_image_shape,
                padding_size=padding * scale
            )
            sr_img = (np_sr_image * 255).astype(np.uint8)
            sr_img = unpad_image(sr_img, pad_size * scale)
            sr_img = Image.fromarray(sr_img)

        return sr_img

```

Файл rrdnet_arch.py

```

import torch
from torch import nn as nn
from torch.nn import functional as F

from .arch_utils import default_init_weights, make_layer,
pixel_unshuffle

class ResidualDenseBlock(nn.Module):
    def __init__(self, num_feat=64, num_grow_ch=32):
        super(ResidualDenseBlock, self).__init__()
        self.conv1 = nn.Conv2d(num_feat, num_grow_ch, 3, 1, 1)
        self.conv2 = nn.Conv2d(num_feat + num_grow_ch,
num_grow_ch, 3, 1, 1)
        self.conv3 = nn.Conv2d(num_feat + 2 * num_grow_ch,
num_grow_ch, 3, 1, 1)

```

```

        self.conv4 = nn.Conv2d(num_feat + 3 * num_grow_ch,
num_grow_ch, 3, 1, 1)
        self.conv5 = nn.Conv2d(num_feat + 4 * num_grow_ch,
num_feat, 3, 1, 1)

        self.lrelu = nn.LeakyReLU(negative_slope=0.2,
inplace=True)

        # initialization
        default_init_weights([self.conv1, self.conv2,
self.conv3, self.conv4, self.conv5], 0.1)

    def forward(self, x):
        x1 = self.lrelu(self.conv1(x))
        x2 = self.lrelu(self.conv2(torch.cat((x, x1), 1)))
        x3 = self.lrelu(self.conv3(torch.cat((x, x1, x2), 1)))
        x4 = self.lrelu(self.conv4(torch.cat((x, x1, x2, x3),
1)))
        x5 = self.conv5(torch.cat((x, x1, x2, x3, x4), 1))
        # Emperically, we use 0.2 to scale the residual for
better performance
        return x5 * 0.2 + x

class RRDB(nn.Module):
    def __init__(self, num_feat, num_grow_ch=32):
        super(RRDB, self).__init__()
        self.rdb1 = ResidualDenseBlock(num_feat, num_grow_ch)
        self.rdb2 = ResidualDenseBlock(num_feat, num_grow_ch)
        self.rdb3 = ResidualDenseBlock(num_feat, num_grow_ch)

    def forward(self, x):
        out = self.rdb1(x)
        out = self.rdb2(out)
        out = self.rdb3(out)
        # Emperically, we use 0.2 to scale the residual for
better performance
        return out * 0.2 + x

class RRDBNet(nn.Module):
    def __init__(self, num_in_ch, num_out_ch, scale=4,
num_feat=64, num_block=23, num_grow_ch=32):
        super(RRDBNet, self).__init__()
        self.scale = scale
        if scale == 2:
            num_in_ch = num_in_ch * 4
        elif scale == 1:
            num_in_ch = num_in_ch * 16
        self.conv_first = nn.Conv2d(num_in_ch, num_feat, 3, 1,
1)
        self.body = make_layer(RRDB, num_block,
num_feat=num_feat, num_grow_ch=num_grow_ch)

```

```

        self.conv_body = nn.Conv2d(num_feat, num_feat, 3, 1, 1)
        # upsample
        self.conv_up1 = nn.Conv2d(num_feat, num_feat, 3, 1, 1)
        self.conv_up2 = nn.Conv2d(num_feat, num_feat, 3, 1, 1)
        if scale == 8:
            self.conv_up3 = nn.Conv2d(num_feat, num_feat, 3, 1,
1)

        self.conv_hr = nn.Conv2d(num_feat, num_feat, 3, 1, 1)
        self.conv_last = nn.Conv2d(num_feat, num_out_ch, 3, 1,
1)

        self.lrelu = nn.LeakyReLU(negative_slope=0.2,
inplace=True)

    def forward(self, x):
        if self.scale == 2:
            feat = pixel_unshuffle(x, scale=2)
        elif self.scale == 1:
            feat = pixel_unshuffle(x, scale=4)
        else:
            feat = x
        feat = self.conv_first(feat)
        body_feat = self.conv_body(self.body(feat))
        feat = feat + body_feat
        # upsample
        feat = self.lrelu(self.conv_up1(F.interpolate(feat,
scale_factor=2, mode='nearest'))))
        feat = self.lrelu(self.conv_up2(F.interpolate(feat,
scale_factor=2, mode='nearest'))))
        if self.scale == 8:
            feat = self.lrelu(self.conv_up3(F.interpolate(feat,
scale_factor=2, mode='nearest'))))
        out = self.conv_last(self.lrelu(self.conv_hr(feat)))
        return out

```

Файл utils.py

```

import numpy as np
import torch
from PIL import Image
import os
import io

def pad_reflect(image, pad_size):
    imsize = image.shape
    height, width = imsize[:2]
    new_img = np.zeros([height + pad_size * 2, width + pad_size
* 2, imsize[2]]).astype(np.uint8)
    new_img[pad_size:-pad_size, pad_size:-pad_size, :] = image

    new_img[0:pad_size, pad_size:-pad_size, :] =
np.flip(image[0:pad_size, :, :], axis=0) # top

```

```

        new_img[-pad_size:, pad_size:-pad_size, :] = np.flip(image[-
pad_size:, :, :], axis=0) # bottom
        new_img[:, 0:pad_size, :] = np.flip(new_img[:,
pad_size:pad_size * 2, :], axis=1) # left
        new_img[:, -pad_size:, :] = np.flip(new_img[:, -pad_size *
2:-pad_size, :], axis=1) # right

    return new_img

def unpad_image(image, pad_size):
    return image[pad_size:-pad_size, pad_size:-pad_size, :]

def process_array(image_array, expand=True):
    image_batch = image_array / 255.0
    if expand:
        image_batch = np.expand_dims(image_batch, axis=0)
    return image_batch

def process_output(output_tensor):
    sr_img = output_tensor.clip(0, 1) * 255
    sr_img = np.uint8(sr_img)
    return sr_img

def pad_patch(image_patch, padding_size, channel_last=True):
    if channel_last:
        return np.pad(
            image_patch,
            ((padding_size, padding_size), (padding_size,
padding_size), (0, 0)),
            'edge',
        )
    else:
        return np.pad(
            image_patch,
            ((0, 0), (padding_size, padding_size),
(padding_size, padding_size)),
            'edge',
        )

def unpad_patches(image_patches, padding_size):
    return image_patches[:, padding_size:-padding_size,
padding_size:-padding_size, :]

def split_image_into_overlapping_patches(image_array,
patch_size, padding_size=2):
    xmax, ymax, _ = image_array.shape
    x_remainder = xmax % patch_size

```

```

    y_remainder = ymax % patch_size

    # modulo here is to avoid extending of patch_size instead of
0    x_extend = (patch_size - x_remainder) % patch_size
    y_extend = (patch_size - y_remainder) % patch_size

    # make sure the image is divisible into regular patches
    extended_image = np.pad(image_array, ((0, x_extend), (0,
y_extend), (0, 0)), 'edge')

    # add padding around the image to simplify computations
    padded_image = pad_patch(extended_image, padding_size,
channel_last=True)

    xmax, ymax, _ = padded_image.shape
    patches = []

    x_lefts = range(padding_size, xmax - padding_size,
patch_size)
    y_tops = range(padding_size, ymax - padding_size,
patch_size)

    for x in x_lefts:
        for y in y_tops:
            x_left = x - padding_size
            y_top = y - padding_size
            x_right = x + patch_size + padding_size
            y_bottom = y + patch_size + padding_size
            patch = padded_image[x_left:x_right, y_top:y_bottom,
:]

            patches.append(patch)

    return np.array(patches), padded_image.shape

def stich_together(patches, padded_image_shape, target_shape,
padding_size=4):
    xmax, ymax, _ = padded_image_shape
    patches = unpad_patches(patches, padding_size)
    patch_size = patches.shape[1]
    n_patches_per_row = ymax // patch_size

    complete_image = np.zeros((xmax, ymax, 3))

    row = -1
    col = 0
    for i in range(len(patches)):
        if i % n_patches_per_row == 0:
            row += 1
            col = 0
        complete_image[

```

```
        row * patch_size: (row + 1) * patch_size, col *
patch_size: (col + 1) * patch_size, :
    ] = patches[i]
    col += 1
    return complete_image[0: target_shape[0], 0:
target_shape[1], :]
```

Приложение Б

(Обязательное)

Нейросетевые алгоритмы улучшения качества изображений

Презентация

Нейросетевые алгоритмы улучшения качества изображений

Автор работы:
студент группы 22ВВим1 Кривоzubов А. К.

Научный руководитель:
к.т.н., доцент кафедры ВТ Слепцов Н.В.

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Актуальность

Проблема получения изображения высокого разрешения исследуется в течение последних двадцати лет. Для решения этой задачи предложены различные классические подходы, которые можно классифицировать по разным критериям. Но они не являются универсальными и, в большинстве случаев, необходимо человеческое вмешательство для определения перечня применяемых алгоритмов и их параметров.

В связи с этим в настоящий момент всё большую популярность набирает нейросетевой подход в обработке изображений. Нейронные сети используют различные методы обработки изображений, преследуя цель увеличить разрешение изображения. Обычно это проходит в несколько этапов и в конце представляется уже улучшенная картинка в качестве результата. Сфера применения данных технологий довольно обширна и программных продуктов, предоставляющих такие услуги, со временем становится только больше.

С учетом вышесказанного, тема исследования, связанная с разработкой нейросетевой модели направленной на увеличение разрешения изображения, является актуальной.

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Цель работы

Цель работы - сравнительный анализ и изучение нейросетевых алгоритмов для увеличения разрешения и качества изображений, исследование и выбор архитектуры в качестве основы для нейросетевого блока разрабатываемого приложения для обработки изображений.

Задачи исследования

1. Обзор и анализ существующих исследований и алгоритмов, применяемых и направленных на увеличение разрешения изображения, основанных на нейронных сетях
2. Разработать нейросетевую модель для увеличения разрешения изображения
3. Обучить модель для решения поставленной задачи увеличения разрешения изображения
4. Тестирование и сравнение полученной модели с аналогами
5. Разработать функционал приложения
6. Тестирование и отладка разработанного приложения

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Классические алгоритмы

- Метод ближайшего соседа
- Билинейная интерполяция
- Бикубическая интерполяция
- Сплаины
- Функция кардинального синуса
- Метод Ланцоша

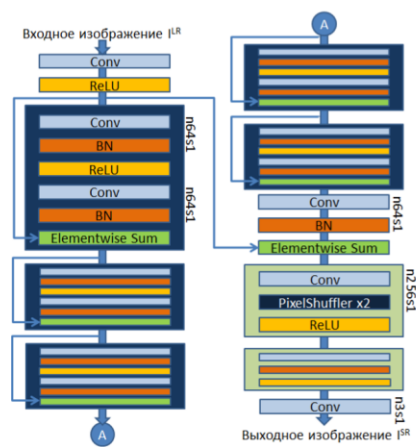
Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Генеративно-состязательные сети



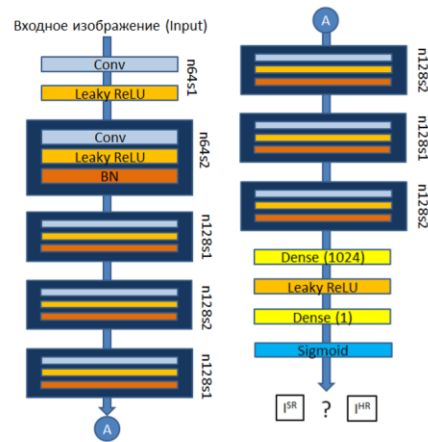
Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Структура генератора



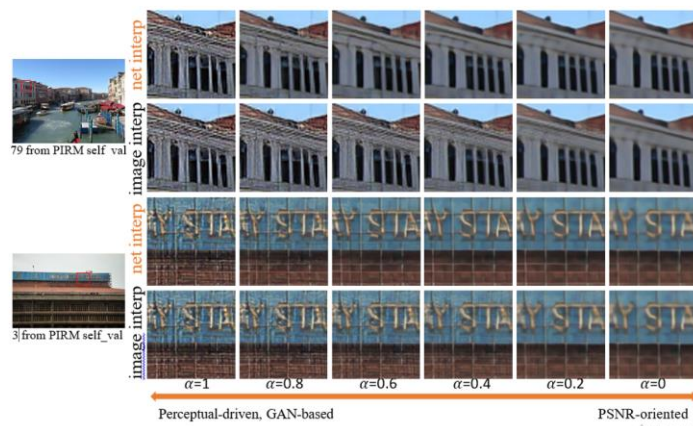
Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Структура дискриминатора



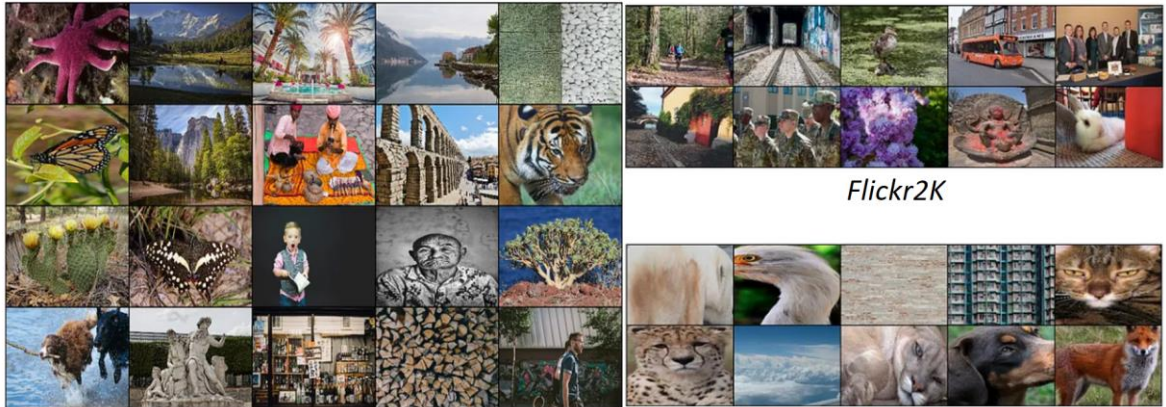
Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Сетевая интерполяция



Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Обучение и тестирование

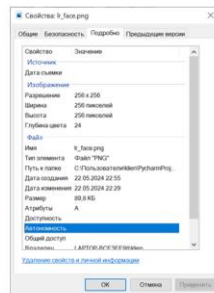
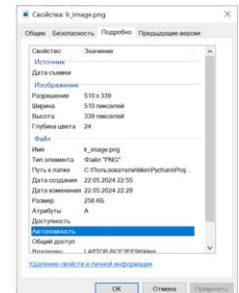
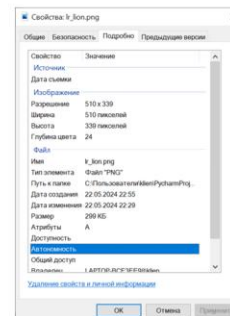


Flickr2K

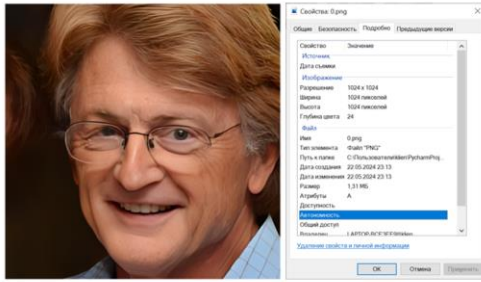
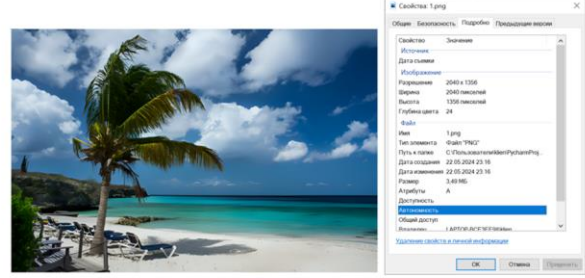
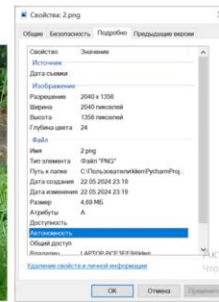
DIV2K

Outdoor Scene Training

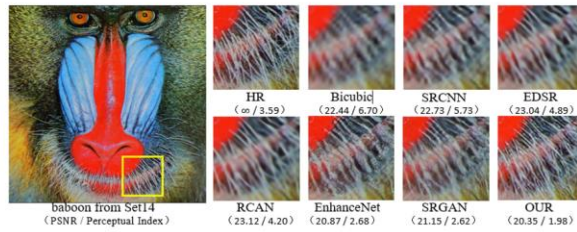
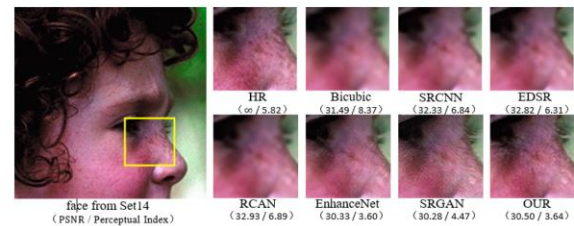
Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".



Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

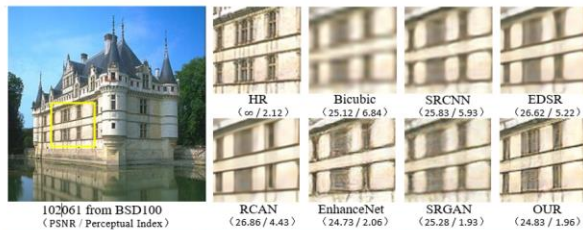
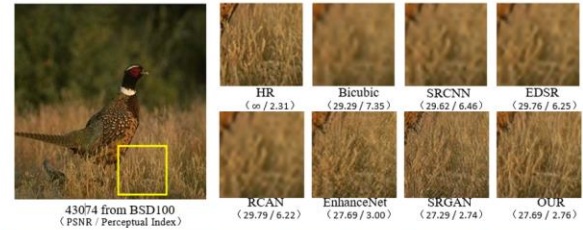


активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".



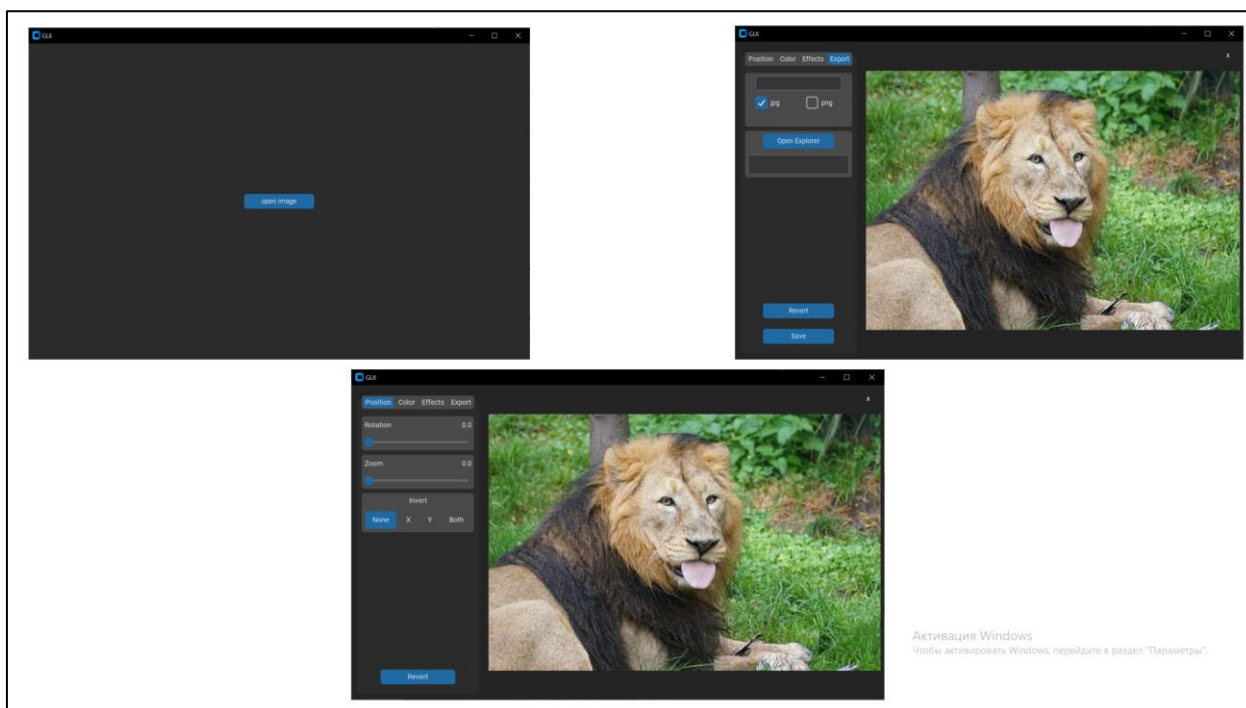
	HR	Bicubic	SRCNN	EDSR	RCAN	EnhanceNet	SRGAN	OUR
PSNR	∞	22.44	22.73	23.04	23.12	20.87	21.15	20.35
PI	3.59	6.70	5.73	4.89	4.20	2.68	2.62	1.98

	HR	Bicubic	SRCNN	EDSR	RCAN	EnhanceNet	SRGAN	OUR
PSNR	∞	31.49	32.33	32.82	32.93	30.33	30.28	30.50
PI	5.82	8.37	6.84	6.31	6.89	3.60	4.47	3.64



	HR	Bicubic	SRCNN	EDSR	RCAN	EnhanceNet	SRGAN	OUR
PSNR	∞	25.12	25.83	26.62	26.86	24.73	25.28	24.83
PI	2.12	6.84	5.93	5.22	4.43	2.06	1.93	1.96

	HR	Bicubic	SRCNN	EDSR	RCAN	EnhanceNet	SRGAN	OUR
PSNR	∞	29.29	29.62	29.76	29.79	27.69	27.29	27.69
PI	2.31	7.35	6.46	6.25	6.22	3.00	2.74	2.76



Результат работы

В ходе выполнения данной работы было исследовано и проанализировано использование искусственного интеллекта для обработки изображений и улучшения их качества посредством увеличения разрешения изображения, проанализированы существующие классические и нейросетевые методы решения данной проблемы, выявлены их преимущества и недостатки.

В результате выполнения данной выпускной квалификационной работы были достигнуты следующие результаты:

- проанализированы существующие нейросетевые модели, способные обрабатывать изображения и увеличивать их разрешение, сравнены и выявлены сильные и слабые стороны текущих решений,
- разработана нейросетевая модель для увеличения разрешения изображения,
- произведено обучение разработанной модели для увеличения разрешения изображения,
- реализовано тестирование и сравнение полученной модели с аналогами,
- разработан функционал приложения, направленного на обработку изображений и увеличение его качества,
- реализовано тестирование и отладка разработанного приложения,
- показана применимость разработанной нейросетевой модели и приложения с использованием различных входных данных.

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Спасибо за внимание!

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".