МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Технология программирования
(направленность (профиль) / специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка алгоритма восстановления расфокусированных и смазанных изображений»

Студент А.В. Герасимов
(И.О. Фамилия) (личная подпись)

Руководитель к.т.н., доцент Э.В. Егорова
(ученая степень, звание, И.О. Фамилия)

Консультант О.А. Головач
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

## Аннотация

Темой выпускной квалификационной работы является «Разработка алгоритма восстановления расфокусированных и смазанных изображений».

Актуальность данной работы состоит в разработке нового алгоритма, позволяющего уменьшить затратность ресурсов и увеличить скорость восстановления изображений по сравнению с современными методами.

Объект исследования – процесс функционирования алгоритмов восстановления расфокусированных и смазанных изображений. Здесь рассматриваются задачи разработки алгоритма, и разработка программной реализации алгоритма с применением языка программирования Python.

Предмет бакалаврской работы – алгоритм восстановления расфокусированных и смазанных изображений.

Целью бакалаврской работы является создание алгоритма восстановления расфокусированных и смазанных изображений.

Для достижения поставленной цели были выполнены следующие задачи:

– Анализ существующих алгоритмов восстановления расфокусированных и размытых изображений;

– выявление недостатков в существующих алгоритмах;

– Формирование требований к алгоритму восстановления расфокусированных и размытых изображений;

– Тестирование алгоритма, на основе выявленных требований.

Структура бакалаврской работы содержит введение, три части, заключение, список литературы.

В данной выпускной квалификационной работе содержится 60 страниц, на которых расположены 2 таблицы, 47 рисунков и 25 используемых источников, 9 из которых являются иностранными, а также 2 приложения.

## Abstract

The title of the graduation work is "Development of an algorithm for recovering defocused and blurry images."

The structure of graduation work contains an introduction, three chapters, conclusion and list of references.

The graduation work consists of an explanatory note on 60 pages, introduction, including 47 figures, 2 tables, the list of 25 references including 9 foreign sources and 2 appendices.

The key issue of graduation work is the process of functioning of algorithms for recovering defocused and blurry images.

In this graduation work, we consider the tasks of developing an algorithm, and developing a software implementation of an algorithm using the Python programming language.

The aim of the work is to create an algorithm for recovering defocused and blurry images.

The first part of the graduation work gives details about the theory of images deconvolution.

The second part of the graduation work gives details about the development and design of the algorithm for restoring defocused and blurry images, development tools are selected, effectiveness of the selected development technologies is substantiated, and their functionality is described. The development is done by using neural network technologies.

The final part of the graduation work gives detailed information about testing the developed product. The application of the developed algorithm to images of various sizes is described.

**Содержание**

## Введение

Современные средства создания фотографий несовершенны: люди, пользующиеся различными фотоаппаратами, телефонами, вынуждены покупать различные приспособления для получения более качественных фотографий. К ним можно отнести, например, стабилизаторы.

Возможность осуществления восстановления расфокусированных и смазанных изображений без посредников реализуется с помощью нейронных сетей, а также алгоритмов компьютерного зрения.

Принцип применения алгоритмов для восстановления изображений был назван деконволюцией. Различные методы деконволюции (обратной свертки) позволяют решить вопрос восстановления расфокусированных и смазанных изображений без применения знаний в области обработки фотографий со стороны обычного пользователя.

**Актуальность** данной работы состоит в разработке нового алгоритма, позволяющего уменьшить затратность ресурсов и увеличить скорость восстановления изображений по сравнению с современными методами.

**Новизна** бакалаврской работы заключается в новом алгоритме восстановления расфокусированных и смазанных изображений.

**Практическая ценность** состоит в разработке нового алгоритма для восстановления расфокусированных и смазанных изображений.

**Объект** исследования – процесс функционирования алгоритмов восстановления расфокусированных и смазанных изображений.

**Предмет** исследования – алгоритм восстановления расфокусированных и смазанных изображений.

**Цель** исследования – разработка авторского алгоритма восстановления расфокусированных и смазанных изображений.

Для достижения цели поставлены следующие задачи:

– Проанализировать существующие алгоритмы восстановления расфокусированных и смазанных изображений;

– Выявить недостатки существующих алгоритмов;

– Сформировать требования к алгоритму восстановления;

– Разработать алгоритм восстановления расфокусированных и смазанных изображений;

– Протестировать алгоритм на реальных примерах.

Бакалаврская работа состоит из введения, трёх частей, заключения, списка литературы.

В первой части затрагиваются вопросы процесса работы алгоритмов восстановления расфокусированных и смазанных изображений, способы и сферы их применения. На основании полученных данных формулируется задача и требования к её непосредственному решению.

Во второй части рассматривается архитектура, выбранная для решения задачи, средства реализации и описываются причины, по которым они были выбраны.

В третьей части предоставляются результаты тестирования разработанного алгоритма на основе выявленных требований.

В заключении подводятся итоги исследования, формируются окончательные выводы и описываются результаты проделанной работы.

# 1 Анализ области применения алгоритмов восстановления расфокусированных и смазанных изображений

## 1.1 Описание работы алгоритмов восстановления расфокусированных и смазанных изображений

Проблема восстановления нечетких изображений является одной из наиболее интересных и важных тем в задачах обработки, как с теоретической, так и с практической точек зрения. Частными случаями являются размытие из-за неправильного фокуса и смаза.

Во время движения часто может происходить смазывание фотографий, которое приводят к возникновению нечетких изображений. Данные изображения формируются с информационным повреждением [6]. Приведенный дефект очень сложен в корректировке. Такие деформации изображений как шум, неправильная экспозиция, дисторсия специалисты уже умеют исправлять. Средства для решения данной проблемы давно разработаны.

На сегодняшний день практически не существует инструментов для устранения расфокусировки. Однако существуют различные алгоритмы деконволюции (или методы обратной свертки) изображений. Возможность восстановления расфокусированных и смазанных изображений достигается при помощи применения к ним следующих алгоритмов: фильтр Винера [4], алгоритм Люси-Ричардсона, а также других методов деконволюции изображений.

Задачу по восстановлению искаженных изображений можно отнести к областям компьютерного зрения, нейронных сетей, обработке и анализу изображений. Эти направления тесно связанны друг с другом и, как правило, имеют общие решения.

Анализ изображений, их обработка и восстановление сосредоточены на обработке цифровых изображений, преобразовании одного изображения в другое.

Как правило, искаженное изображение g(x, y), шум которого не учитывается, описывается с помощью двумерной свертки исходного изображения [6]. Данная двумерная свертка представляется в виде функции f(x,y), с линейной, неизменяемой к сдвигу функцией импульсного отклика h(x,y), представленной в формуле 1.

$$g(x,y) = f(x,y) * h(x,y) = \sum_{(n,m)} f(n,m)h(x-n, y-m) \quad (1)$$

где * - оператор двумерной линейной свертки, переменные m, n, x, y – целые числа.

Для проведения операции деконволюции исходного изображения, представленного в виде функции f(x,y) необходимо применение операции деконволюции к h(x,y) – функции импульсного отклика искаженного изображения g(x,y).

Иногда функция импульсного отклика h(x,y) является известной до операции деконволюции. Тогда для восстановления качества изображений применяются различные классические методы. На сегодняшний день к ним можно отнести такие методы как:

- обратная фильтрация;
- фильтрация методом наименьших квадратов;
- фильтр Калмана;
- фильтрация Винера;
- алгоритм Люси-Ричардсона.

Однако в действительности функция смаза часто не определена, а количество информации о реальном изображении очень мало. В связи с этим исходное изображение $f(x,y)$ должно быть восстановлено из $g(x,y)$ с использованием неполной информации о процессе смаза и реальном изображении. Данные оценочные подходы, модель искажения которых является линейной, следует определять в качестве слепой деконволюции. При данном методе восстановления изображений по характеристикам искаженного изображения должны быть найдены исходное изображение, а также функция импульсного отклика.

Существует несколько причин для использования методов слепой деконволюции:

– получение подготовительных данных о снимаемой сцене бывает дорогим, опасным, а иногда физически невозможным. Ярким примером является астрономия, где смаз не может быть точно воспроизведён случайным процессом. Причиной этому является сложный процесс описи колебаний функции импульсного отклика;

– сложность обработки изображений в реальном времени. Примером могут послужить медицинские конференции. Здесь параметры функции импульсного отклика не могут быть заблаговременно определены для проведения операции моментального восстановления.

В дополнение, современные методы восстановления, которые используются для оценки смаза, приводят к ошибке. Если изображение обрабатывается таким образом, то в восстановленном изображении могут возникать артефакты.

Основной проблемой вопроса слепой деконволюции является разбиение двух сигналов f и h. Как правило, сигналы либо неизвестны, либо частично известны.

Во многих средствах слепой деконволюции объединение неполной информации происходит по наиболее благоприятным критериям. Данные критерии необходимо минимизировать или максимизировать. Приведенная операция позволит найти оценочные значения всех известных параметров.

Существуют некоторые значительные составляющие проблемы слепой деконволюции изображений. Основным условием является свойство несократимости у входного изображение и функции импульсного отклика.

Что же представляет из себя данный тип сигнала? Несократимый сигнал – это некий сигнал, не имеющий возможности быть подлинно выраженным при помощи свёртки нескольких компонентов сигнала. Следует учитывать, что двумерная дельта-функция не должна являться элементом сигнала [5]. В противном случае, если сам снимок или функция импульсного

отклика будут обладать свойством сократимости, то полученное решение станет приближенным.

Традиционный подход к деконволюции линейными способами заключает в себе получение более четкого приближения к исходному изображению. В совершенстве случаев приближенное изображение $\hat{f}(x, y)$, получаемое с помощью процедуры восстановления, должно быть идентично входному изображению $f(x, y)$. Главная задача методов слепой деконволюции заключается в получении увеличенной и сдвинутой копии оригинального изображения. После восстановления сдвиг, а также масштабирование могут быть воссозданы с использованием дополнительных операций.

В конечном итоге изображение представляется в виде выражения $g(x, y) = f(x, y) * h(x, y) + n(x, y)$, где $f(x, y)$ – исходное изображение, $h(x, y)$ – функция импульсного отклика, $n(x, y)$ – шум, а результатом является функция $g(x, y)$, которая представляет из себя испорченное изображение.

Основные типы шума:

– электронный шум. Данный вид шума возникает вследствие теплового перемещения электронов в электронных частях средств съёмки;

– фотоэлектрический шум. Возникает в результате статистической природы света;

– зернистый шум. Возникает из-за зернистости пленки;

– дискретный шум. Появляется при преобразовании непрерывного сигнала изображения в последовательность чисел (представление сигнала в виде базиса).

Слепая деконволюция – это метод восстановления четкой версии исходного смазанного изображения, в котором отсутствует информация о функции размытия [8]. Математически можно разложить смазанное изображение g на составные части, как показано в формуле 2.

$$g = h \otimes f + n \quad (2)$$

где f – визуально правдоподобное четкое изображение, а h – неизвестное неотрицательное ядро смаза, размер которого мал по сравнению с размером изображения, n – шум (для удобства аналитического описания предполагаем его приблизительно Гауссовым).

Данная задача является плохо обусловленной, то есть имеет ошибку в начальных данных, что приводит к существованию бесконечного набора пар (h, f), объясняющий любое полученное значение g. Например, одно нежелательное решение, которое полностью удовлетворяет уравнению – это решение с отсутствием смаза: h – дельта ядра (тождественно ядру), а $f = g$. Некорректная природа проблемы предполагает, что для g или h должны быть введены дополнительные предположения.

Целью слепой деконволюции является получение неизвестного ядра смаза $h$ и визуально правдоподобного четкого изображения $f$ из одиночного входного смазанного изображения $g$. В качестве дополнительного условия описывается неотрицательность ядра смаза $h$ и его малая величина по сравнению с размером изображения [8].

Существует два основных подхода к задаче слепой деконволюции:

– импульсный отклик определяется изолированно от восстанавливаемого изображения. В этом случае полученная информация используется позже, применяя один из известных классических методов восстановления. Оценка функции импульсного отклика и восстановление изображения являются отдельными процедурами для данного подхода. Применяемые алгоритмы для реализации данного метода являются вычислительно простыми;

– процедура идентификации функции импульсного отклика применяется в восстанавливающем алгоритме. Данный подход подразумевает, что функция импульсного отклика и восстанавливаемого изображения будут оцениваться одновременно. Он применим к более сложным алгоритмам вычисления.

Теперь проанализируем принципы действия существующих алгоритмов.

Фильтрация Винера – один из первых методов, который был разработан для сокращения шумов на изображениях, полученных случайным образом [15]. Метод базируется на ожидании, что аддитивный шум является случайным процессом, самостоятельным от расположения пикселя. В алгоритме происходит минимизация квадратичной ошибку между входным и восстановленным изображением. Фильтрация Винера представляет из себя фильтр низких частот, где используется нижняя частота среза в областях с малой детализацией, и верхняя частота среза для сохранения деталей в областях с элементами высоких отклонений. Частота среза определяется размером области: у нижних частот среза большее окно, что приводит к большему подавлению размытости и шума [4].

В фильтре Винера пиксель выходного изображения *y* наследуется от пикселя *x* во входном смазанном изображении посредством равенства, отображенного на формуле 3.

$$y = \mu_x + (x - \mu_x)\frac{\vartheta_x}{\vartheta_x + \vartheta_y} \quad (3)$$

где $\mu_x$ и $\vartheta_x$ — среднее значение и дисперсия *x* в окрестностях пикселя (размер окрестности задается аргументами функции *win_h* и *win_w*), $v_n$ — дисперсия аддитивного шума, полученного из входного изображения.

Выходной пиксель представляет из себя среднюю сумму значений соседних входных пикселей и локального значения контрастности $x - \mu_x$, в таком масштабе, что в областях с большой детализацией, в которых дисперсия шума $v_n$ намного меньше, чем дисперсия изображения $v_x$, коэффициент масштабирования приближен к 1, а выходной пиксель *y* приближен к входному пикселю *x* с небольшой фильтрацией. А в областях с низкой детализацией, в которых дисперсия изображения ниже, выходной пиксель представляется в виде локального среднего значения [21].

Границы изображения считаются нулевыми значениями, представленными оттенками серого. Это делает недействительными те пиксели, которые находятся рядом с границами изображения. Ещё одним свойством является меньшая область окрестности по сравнению с входным изображением.

Важным минусом фильтра Винера является наличие краевых эффектов. Они проявляются в виде колебательной помехи, которая маскирует восстановленное изображение [20].

На рисунке 1 показан результат работы фильтра Винера для восстановления расфокусированного изображения.

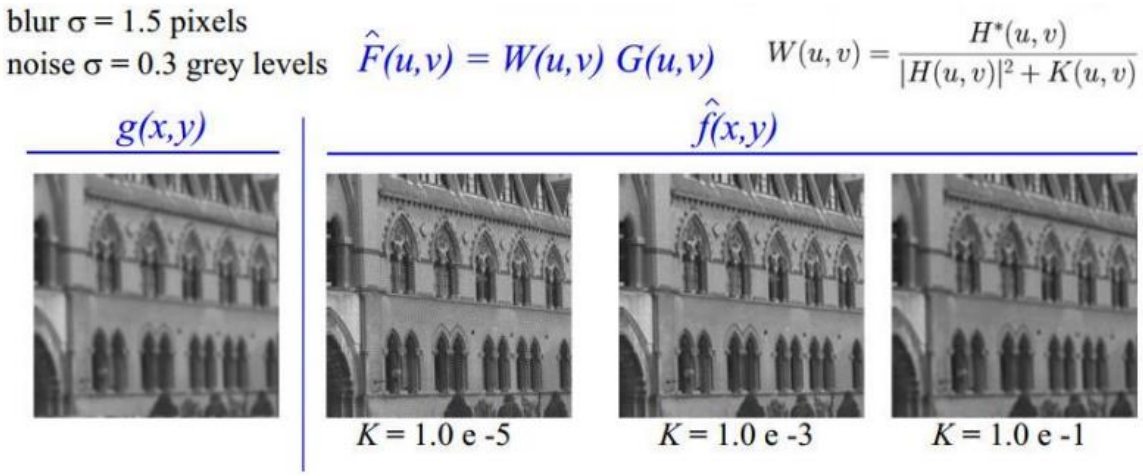



Рисунок 1 – Результат работы фильтра Винера

Алгоритм Люси-Ричардсона представляет собой реализацию, основанную на максимальном использовании правдоподобия. Изображение формируется при помощи статик Пуассона. Основным минусом данного алгоритма можно отметить, что после процесса обработки входного изображения происходит усиление шума в конечном восстановленном изображении [20].

Расчет выходного изображения производится по формуле 4.

$$u_j^{(t+1)} = u_j^{(t)} \sum_i \frac{d_i}{c_i} p_{ij} \quad (4)$$

где $p_{ij}$ – пиксель ядра, $u_j$ – яркость j-го пикселя скрытого изображения, $d_i$ – наблюдаемая яркость j-го пикселя, а $c_i$ – наблюдаемая яркость j-го пикселя в зависимости от времени t.

В свою очередь, наблюдаемая яркость j-пикселя $d_i$ вычисляется по формуле 5.

$$d_i = \sum_j p_{ij} u_j \quad (5)$$

А наблюдаемая яркость j-пикселя в зависимости от времени t вычисляется по формуле 6.

$$c_i = \sum_j p_{ij} u_j^{(t)} \quad (6)$$

где t – время, j – наблюдаемый пиксель.

Идея метода заключается в поиске наиболее вероятных значений яркости j-го пикселя скрытого изображения, зная значения наблюдаемой яркости j-го пикселя и пикселя ядра.

Фильтр Калмана представляет собой фильтр с импульсной характеристикой. Его задача заключается в оценке квантового состояния динамической системы. Для этого используется ряд шумных измерений. Основное применение фильтр находит в инженерных и эконометрических сферах. Ярким примером могут послужить системы технического зрения. Применение фильтра Калмана составляет важную часть теории эксплуатации. Данный алгоритм играет большую роль в создании систем управления. В составе с линейно-квадратичным регулятором возможно решение задачи линейно-квадратичного управления Гаусса. Потенциальным решением многих задач теории управления является именно Фильтр Калмана и линейно-квадратичный регулятор [15].

В большинстве случаев величина вектора состояния объекта превышает величину вектора, который был получен в результате слежения. Наряду с этим алгоритм позволяет оценивать полное внутреннее состояние объекта.

Основная цель данного фильтра состоит в рекурсивной оценке волновой функции для известной меняющейся системы. В связи с этим, для

вычисления фактического состояния системы нужно знать сиюминутное измерение и предыдущее состояние фильтра [15]. Реализация данного фильтра осуществлена во временном представлении, однако он обращается как к оценкам состояния, так и неопределённости вектора состояния, основываясь на выражении Байеса условной вероятности, отраженного в формуле 7.

$$P\ (A \mid B) = \frac{P(B \mid A) * P(A)}{P(B)} \quad (7)$$

где P(A) – априорная вероятность гипотезы A, P(A | B) – вероятность гипотезы A при наступлении события B, P(B | A) – вероятность наступления события B при истинности гипотезы A, P(B) – полная вероятность наступления события B.

Работа алгоритма происходит в несколько этапов. Этап прогнозирования предполагает предсказывание значений переменных состояния и их неопределённости. Далее, опираясь на измерения, погрешность которых известна, происходит подробное описание результатов. Основываясь на свойстве пошаговости, алгоритм в реальном времени отслеживает состояние объекта без предусмотрения. Используются только текущие замеры, а также данные о предыдущем состоянии и его неопределенности [15].

Недостатком фильтра является его расходимость. Она может проявляться из-за неточности задания процесса моделирующего сообщения, а также отсутствия достоверной информации о реальной задаче. В связи с этим коэффициент усиления фильтра K(t) неизменно стремится к нулю.

На рисунке 2 представлена последовательность вычислений при помощи данного фильтра.

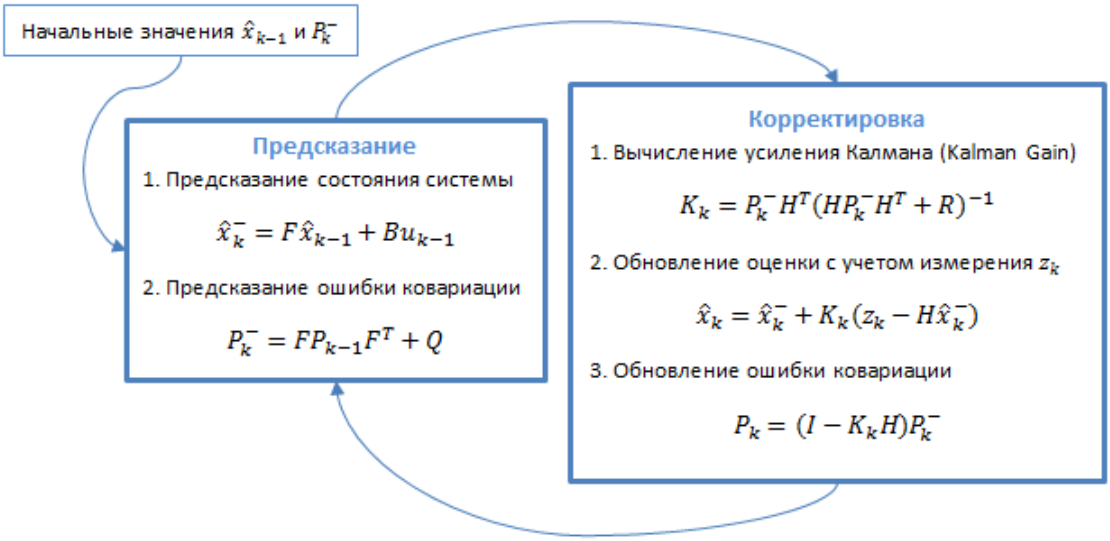



Рисунок 2 – Последовательность действий фильтра Калмана

На рисунке 3 показан результат работы фильтра Калмана.


*Повышение разрешения в 4 раза: (а) результат применения фильтра Калмана; (б) результат применения метода ИОП; (в) результат применения метода МП; (г) оригинальное изображение ВР*

Рисунок 3 – фильтр Калмана

В результате написания данного пункта были рассмотрены существующие алгоритмы и решения для деконволюции изображений, определены их сильные и слабые стороны. После этого перейдем к следующему пункту, где на основе описанных выше решений составим требования к разрабатываемому.

## 1.2 Формирование требований к разрабатываемому алгоритму восстановления расфокусированных и смазанных изображений

Опираясь на анализ существующих решений, представленный в предыдущем пункте, составим список требований к разрабатываемому алгоритму.

Разрабатываемый алгоритм восстановления расфокусированных и смазанных изображений должен обладать следующими характеристиками:

– уменьшение шума выходного изображения. Полученный критерий был определён с учетом неблагоприятных тенденций со стороны алгоритма Люси-Ричардсона;

– программная реализация алгоритма предполагает использование малого количества ресурсов оперативной памяти, графического процессора, процессорного времени. Из-за обилия различных компьютерных конфигураций необходимо разработать программный продукт с минимальным потреблением ресурсов. В таком случае программное обеспечение на основе составляемого алгоритма будет способно запускаться на компьютерах различной вычислительной мощности, что позволит охватить большой рынок устройств;

– исключение расходимости. Данный критерий определяет конечный момент работы алгоритма. Полученный аспект был определён с учетом предъявления неблагоприятной тенденции со стороны фильтра Калмана.

Полученный список требований был скомпонован в следующую модель информационной системы.

Система должна:

– импортировать изображение из файловой системы ОС;

– анализировать полученное изображение;

– применять алгоритм восстановления к импортированному изображению;

– уменьшать шум выходного изображения, а также сохранять его.

В результате рассмотрения существующих алгоритмов были составлены условия для разрабатываемого алгоритма, а также определены требования к системе.

Рассмотрев требования к разрабатываемому алгоритму и программному продукту, реализуемому по составленному алгоритму, сделаем выводы по первой части.

**Выводы по первой части:**

В процессе изучения теоретической части темы восстановления расфокусированных и смазанных изображений были проанализированы существующие алгоритмы деконволюции. На данный момент для восстановления расфокусированных изображений существуют такие алгоритмы как фильтр Винера, алгоритм Люси-Ричардсона и многие другие.

В каждой рассмотренной последовательности были определены принципы работы, расписаны математические аппараты, а также выявлены слабые и сильные стороны каждого существующего алгоритма.

Проведённый анализ существующих методов восстановления расфокусированных и смазанных изображений показал, что ни одно из рассмотренных программных средств полностью не удовлетворяет запросам разрабатываемого сервиса.

После этого в результате анализа существующих решений был сформулирован список требований к разрабатываемому алгоритму, а также программному продукту, создаваемому на его основе.

Перейдём ко второй части, где подробно рассмотрим процесс построения алгоритма, выбор инструментов разработки программного продукта на основе составляемого алгоритма, а также самого процесса создания программного обеспечения.

# 2 Проектирование алгоритма решения задачи восстановления расфокусированных и смазанных изображений

## 2.1 Составление структуры алгоритма для решения задачи восстановления расфокусированных и смазанных изображений

Основной задачей алгоритма является применение нейронных сетей и технологий машинного обучения для восстановления качества расфокусированных и смазанных изображений. В связи с этим рассмотрим процедуру восстановления изображений и все взаимодействия в системе, связанные с этим процессом.

Графическая запись (блок-схема) разрабатываемого алгоритма представлена в приложении А.

Для применения разрабатываемого алгоритма создадим консольное приложение.

В ходе обдумывания структуры проекта было решено добавить функционал получения данных из определённого каталога, задаваемого пользователем для загрузки существующих расфокусированных и смазанных изображений в память программы в качестве входных пикселей и построения внутреннего представления посредством преобразований, которые превращают необработанные пиксели изображения в сложные к пониманию функции, присутствующих в изображении.

Логическая модель разрабатываемого продукта показана на рисунке 4.

На этапе обсуждения взаимодействия с внешними системами определен и на этапе начальной реализации скорректирован состав переменных для создания программного обеспечения для решения задачи.

Для взаимодействия с классом загрузки расфокусированных и смазанных изображений используются следующие переменные:

– Img_paths – путь до входных изображений;
– Img_size – размер входных изображений в пикселях.

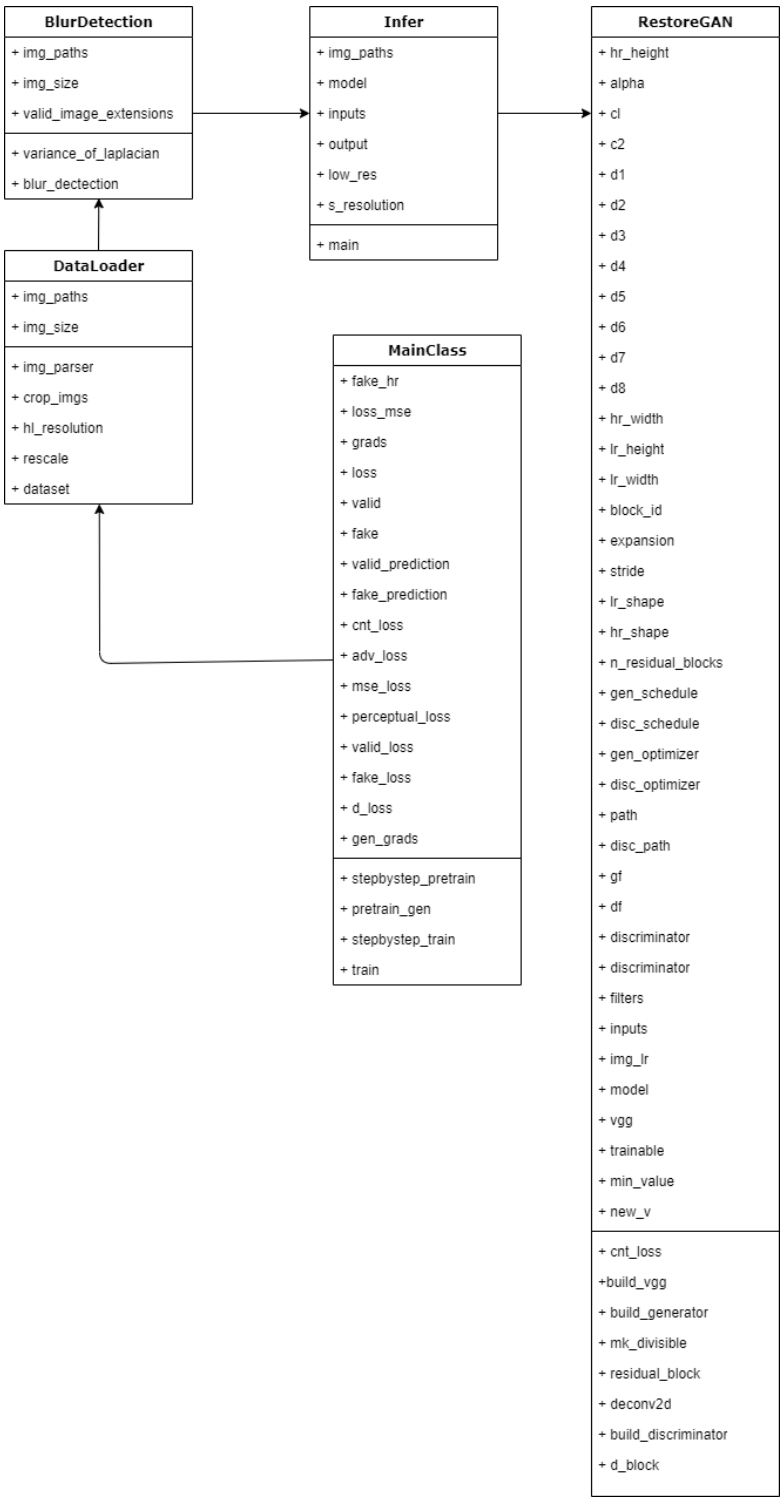



Рисунок 4 – Логическая модель данных разрабатываемого продукта

Для взаимодействия с классом определения смаза из-за функционального предназначения, реализуемого в рамках данной бакалаврской работы программного обеспечения, используются следующие переменные:

– Img_paths – путь до входных изображений, передаваемый из класса загрузки;

– Img_size – размер входных изображений в пикселях, передаваемый из класса загрузки;

– valid_image_extensions – список допустимых расширений входных изображений.

В результате описания данного пункта ВКР в пункте 2.1 была определена задача алгоритма, разработана блок-схема разрабатываемого алгоритма, определена логическая модель данных алгоритма, а также описаны некоторые переменные для взаимодействия с алгоритмом.

## 2.2 Выбор инструментов разработки программного продукта на основе алгоритма восстановления расфокусированных и смазанных изображений

На основе блок-схемы составленного алгоритма, вынесенной в приложение А, перейдем к выбору инструментов разработки программного продукта, использующего данный алгоритм.

Поскольку управление программным обеспечением может осуществляться со стороны внешней системы визуализации, то в качестве типа приложения была выбрана разработка консольного приложения.

Так как в качестве входных данных может использоваться не одно изображение, а сразу несколько, то было решено производить вычисления в многопоточном режиме, а в качестве инструмента разработки был выбран язык программирования Python.

Основными причинами выбора данного языка программирования для реализации дипломной работы являются:

– высокая скорость разработки. Если сравнивать Python с компилируемыми или строго типизированными языками, такими как C, C++ и Java, то данный язык программирования в несколько раз повышает производительность труда разработчика, а объем разрабатываемого

программного кода составляет треть эквивалентного программного кода на вышеупомянутых языках. Данный факт приводит к меньшему затрату времени на отладку и меньший объем затрат труда на сопровождение [1];

– отсутствие компиляции под каждую операционную систему. Программы, разрабатываемые на языке Python запускаются на любой операционной системе, пропуская этапы компиляции и линковки, которые необходимы в таких языках программирования, как C++, Java, C#. Этот критерий позволяет увеличить скорость разработки программного обеспечения;

– переносимость ПО. Наибольшая часть разработанных программ выполняется без изменений на всех основных современных платформах. Перенос программного кода из одной операционной системы в другую, как правило, заключается в копировании исходных файлов с одного носителя на другой;

– библиотеки. Хранилище пакетов Python содержит большое число собранных и переносимых функциональных возможностей, представляемых в виде различных библиотек. Стандартная библиотека содержит в себе огромный функционал, требуемый в прикладных программах. К ним можно отнести как самые простые математические подсчёты, так и библиотеки обработки изображений, а также библиотеки для разработки нейронных сетей. Так же допускается расширение функционала как своими собственными библиотеками и алгоритмами, так и библиотеками, созданными разработчиками по всему миру. Из числа сторонних разработок можно назвать инструменты компьютерного зрения, инструменты написания собственных нейронных сетей и многое другое. Например, библиотека OpenCV позиционируется как свободный и мощный инструмент для взаимодействия с изображениями [1];

– внедрение компонентов. Механизмы интеграции способствуют работоспособности части кода с другими частями приложения благодаря механизмам интеграции. Интеграция предусматривает использование языка

для расширения функциональных возможностей разрабатываемого программного обеспечения. Примерами интеграций является взаимодействие программного кода на языке Python с компонентами на языке Java, вызов функций из библиотек, разработанных на языках C/C++, обращение к программам, написанным на языках C/C++, взаимодействие с платформами COM и .NET.

К минусам данного языка можно отнести:

– скорость работы. Python – не самый быстрый среди языков программирования. Скорость выполнения программ может быть ниже относительно других инструментов разработки [1];

– потребление памяти. В связи с тем, что в Python отсутствуют строго-типизированные данные, то потребление памяти иногда оказывается не минимальным;

Однако для разрабатываемого программного обеспечения эти минусы теряют свою значимость на фоне положительных качеств Python.

На данный момент существует множество различных библиотек для работы с изображениями, которые имеют в своём функционале возможность загрузки изображений в память программы. Этими библиотеками являются:

– Scikit-image;

– OpenCV;

– Mahotas;

– SimpleITK;

– Scipy;

– PIL и его модификация Pillow;

– Matplotlib.

Рассмотрим возможности каждой из этих библиотек.

Scikit-Image представляет собой библиотеку обработки изображений с открытым исходным кодом для языка программирования Python. Она включает в себя алгоритмы сегментации, геометрических преобразований, управления цветовым пространством, анализа, фильтрации, морфологии,

обнаружения признаков и так далее. Предназначена для взаимодействия с числовыми и научными библиотеками Python – NumPy и SciPy. Scikit-Image использует массивы NumPy в качестве объектов изображений путём преобразования исходных изображений.

OpenCV представляет из себя библиотеку, содержащую набор модулей, каждый из которых связан с областями компьютерного зрения [23]. Библиотека включает в себя следующие наборы функций:

– работа с 2d и 3d изображениями;
– система распознавания лиц;
– распознавание жестов;
– распознавание движений;
– идентификация объектов;
– их сегментация;
– отслеживание движений.

Для поддержки некоторых из вышеперечисленных областей OpenCV включает в себя статическую библиотеку машинного обучения, которая содержит:

– бустинг. Он представляет из себя алгоритм машинного обучения. Основной целью является уменьшение смещения и дисперсии в обучении с учителем. Ещё он относится к семейству алгоритмов машинного обучения, которые преобразуют слабые обучающие алгоритмы к сильным [17]. Бустинг основан на вопросе, который был поставлен Кернсом и Вэлиантом. Суть вопроса заключается в возможности набора слабых обучающих алгоритмов создать сильный обучающий алгоритм. Слабый обучающий алгоритм определяется в качестве классификатора. Этот классификатор слабо ассоциируется с правильной классификацией. В связи с этим, в отличии от случайного угадывания, примеры могут помечаться лучше. В отличие от слабого алгоритма, сильный обучающий алгоритм – классификатор, который хорошо взаимодействует с верной классификацией. Большая часть алгоритмов бустинга состоит из обучения слабых классификаторов, которое

происходит при помощи итераций. Цель – сборка в сильный классификатор. Когда они добавляются, им обычно приписываются некоторым образом веса, которые, обычно, связаны с точностью обучения. После того, как слабый классификатор добавлен, веса пересчитываются («пересчёт весовых коэффициентов»). Входные данные с неверной классификацией получают больший вес, а правильно классифицированные экземпляры теряют вес. Таким образом, последующее слабое обучение фокусируется больше на примерах, где предыдущие слабые обучения дали ошибочную классификацию;

– деревья повышения градиента. Данный метод машинного обучения специализируется на задачах классификации, а также зависимости, создающей модель предсказания, которая представляется как множество моделей слабого прогнозирования. Построение модели происходит поэтапно, а также происходит их обобщение, что позволяет оптимизировать произвольную дифференцируемую функцию потерь [23];

– глубокие нейронные сети (DNN). Глубокая нейронная сеть представляет из себя нейронную сеть, где между входным и выходным слоями находятся ещё слои [18]. DNN находит правильную математическую манипуляцию с целью трансформации входных данных в выходные [7]. Сеть проходит по слоям, где на каждом слое рассчитывается вероятность каждого выхода. Например, DNN, который обучен распознавать породы котов, оценит импортированную фотографию и вычислит вероятность того, что кот на изображении относится к определенной породе. Пользователь отбирает результаты и останавливается на тех вероятностях, которые должна отображать сеть и возвращает предложенную метку. Каждая математическая манипуляция является слоем. В свою очередь сложные DNN имеют много слоев. DNN могут моделировать сложные нелинейные отношения. Архитектуры DNN генерируют композиционные модели, в которых объект выражается в виде многоуровневой композиции примитивов [25]. Дополнительные уровни позволяют составлять элементы из более низких

уровней, потенциально моделируя сложные данные с меньшим количеством единиц, чем мелкая сеть с аналогичным исполнением. DNN очень часто представляют собой сети с прямой связью, в которых данные передаются от входного уровня к выходному уровню без обратной связи. Первым делом сеть производит создание карты виртуальных нейронов, затем определяет случайные числовые значения или веса соединений между этими нейронами. Затем веса и входные данные перемножаются между собой и отдают выходной сигнал в диапазоне [0; 1]. В случае не точного распознавания шаблона, алгоритм будет поправлять весовые коэффициенты. В связи с этим некоторые параметры могут оказаться более влиятельными, пока он не произведет правильные математические операции с целью обработки входных данных.

Перейдём к дальнейшему рассмотрению существующих инструментов разработки, где так же опишем функциональные возможности каждой библиотеки.

Ещё одним инструментом обработки изображений является библиотека Mahotas. Она представляет из себя библиотеку для работы с 2D и 3D изображениями. Расширенная обработка изображений осуществляется путём извлечения информации из изображений. Библиотека обладает большими функциональными возможностями для компьютерного зрения, которые могут позволить выполнять такие процессы, как морфологическая обработка, свертка и многое другое [7].

Другим инструментом является библиотека SimpleITK.

SimpleITK - библиотека анализа изображений с открытым исходным кодом. Интерфейс предоставляет только наиболее часто изменяемые алгоритмические настройки компонентов ITK. Кроме того, библиотека предоставляет как объектно-ориентированный, так и процедурный интерфейс для большинства фильтров обработки изображений. Последний позволяет рабочие процессы анализа изображений с кратким синтаксисом [5].

Библиотека SciPy представляет из себя библиотеку с открытым исходным кодом, предназначенную для выполнения научных и инженерных расчётов [19].

Основными возможностями библиотеки являются:

– поиск минимумов и максимумов функций;

– вычисление интегралов функций;

– обработка сигналов;

– обработка изображений.

Пакет включает в себя функции для линейной и нелинейной фильтрации, бинарной морфологии, интерполяции B-сплайнами и измерений объектов.

PIL (Python Image Library) представляет из себя библиотеку языка Python, предназначенную для работы с растровой графикой.

Основными возможностями библиотеки являются:

– поддержка бинарных, полутоновых, индексированных, полноцветных и CMYK изображений;

– поддержка форматов BMP, EPS, GIF, JPEG, PDF, PNG, PNM, TIFF и некоторых других на чтение и запись;

– поддержка множества форматов (ICO, MPEG, PCX, PSD, WMF и др.) только для чтения;

– преобразование изображений из одного формата в другой;

– правка изображений (использование различных фильтров, масштабирование, рисование, матричные операции и т. д.).

Однако поддержка данной библиотеки завершена и на смену ей пришла новая – Pillow. Pillow является форком библиотеки PIL и имеет возможность работы с Python 3.

Matplotlib – библиотека на языке программирования Python для визуализации данных двумерной (2D) и трехмерной (3D) графикой. Matplotlib является гибким, легко конфигурируемым пакетом, который вместе с NumPy, SciPy и IPython предоставляет возможности, подобные

MATLAB. В настоящее время пакет работает с несколькими графическими библиотеками, включая wxWindows и PyGTK.

Библиотека Matplotlib поддерживает следующие форматы изображений:

– encapsulated postscript (eps);
– enhanced metafile (emf);
– jpeg;
– pdf;
– png;
– postscript;
– rgba;
– svg;
– svgz;
– tiff.

В свою очередь для реализации нейронных сетей существует множество различных библиотек.

К таким библиотекам можно отнести:

– Theano;
– Tensorflow;
– Keras;
– Lasagne;
– Neon;
– Kayak.

В качестве средства реализации нейронной сети была выбрана библиотека TensorFlow [3], её надстройка Keras, а также библиотека Theano.

Ознакомимся с каждым инструментом подробнее.

Theano представляет из себя библиотеку численных вычислений, выражаемых NumPy-подобным синтаксисом и компилируемых для

эффективных параллельных вычислений на CPU и на GPU. Данная библиотека представляет собой препроцессор для системы вычислений с тензорами (многомерными массивами данных), сочетающий в себе математические пакеты Mathematica и MATLAB [2].

Основными математическими методами, операциями и структурами, поддерживаемыми Theano являются:

– работа с тензорами, и поддержка множества тензорных операций;

– работа с разреженными матрицами, и поддержка ряда операций с ними;

– многочисленные методы линейной алгебры;

– возможность в режиме работы создавать новые операции с графами;

– операции по преобразованию графов;

– поддержка стандарта BLAS (Basic Linear Algebra Subprograms) для процедур линейной алгебры.

TensorFlow представляет из себя открытую программную библиотеку для машинного обучения, разработанную для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов [2]. Вычисления TensorFlow выражаются в виде потоков данных через граф состояний. Библиотека применяется в основном для аннотации изображений [3].

Keras – открытая нейросетевая библиотека. Данная библиотека является надстройкой над TensorFlow и Theano [22]. Основная цель данной библиотеки – оперативная работа с сетями глубинного обучения [19]. Она содержит реализации широко применяемых строительных блоков нейронных сетей (слои, целевые и передаточные функции, оптимизаторы, и множество инструментов для упрощения работы с изображениями и текстом).

После того, как рассмотрели все доступные библиотеки для работы с изображениями, следует выбрать тип разрабатываемого программного продукта на основе составленного алгоритма.

В качестве основы была выбрана технология генеративно-состязательных сетей.

Генеративно-состязательная сеть (GAN) представляет из себя алгоритм машинного обучения, который создан в результате слияния пары нейронных сетей: Сеть G (генератор) и сеть D (Дискриминатор) [9]. Алгоритм относится к категории машинного обучения без учителя.

Задачей генератора является создание образцов, использующих ряд переменных скрытого пространства [13].

Дискриминатор же в свою очередь находит расхождения сгенерированных образцов от образцов тренировочного набора и оценивает их [14].

Конечная цель этих сетей – генерация образца, в котором будут отличаться черты тренировочных образцов, но в то же время он будет выглядеть как их часть [13].

Основной целью GAN в дипломной работе является улучшение качества нечётких изображений.

Принцип работы генеративно-состязательной сети состоит в том, что сеть G (Generator) производит генерацию образцов, в свою очередь сеть D отличает правильные образцы от неправильных [24].

Генеративная сеть пытается создать новый образец путем использования набора переменных скрытого пространства. Для этого происходит смешивание нескольких исходных образцов. Сеть дискриминатора обучается отличать верные и поддельные образцы. В свою очередь результаты различия передаются на вход генеративной сети с целью подбора лучшего набора латентных параметров, чтобы дискриминативная сеть не смогла отличить подлинные образцы от поддельных [14]. Таким образом, цель генератора заключается в увеличении процента ошибок дискриминатора, а дискриминатор повышает точность распознавания [9].

Принцип работы генеративно-состязательной сети (GAN) показан на рисунке 5.

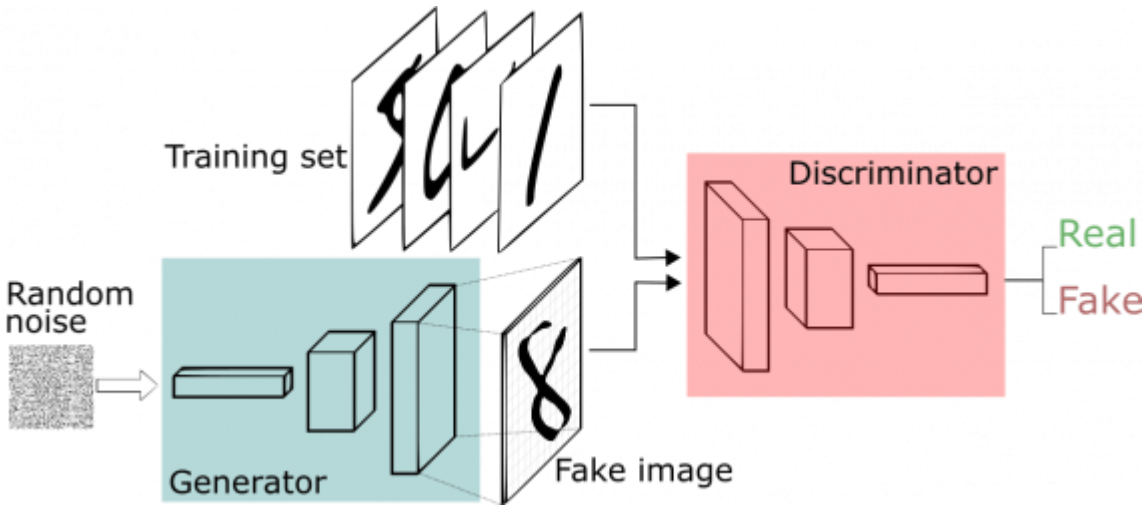



Рисунок 5 – Принцип работы GAN

В результате описания данного пункта были рассмотрены средства реализации разрабатываемого программного обеспечения на основе составленного алгоритма, а также определено понятие генеративно-состязательной сети. Перейдем к следующему пункту, где приступим к написанию программного обеспечения на основе составленного ранее алгоритма восстановления расфокусированных и смазанных изображений.

## 2.3 Реализация программного продукта на основе алгоритма восстановления расфокусированных и смазанных изображений

Для написания программного обеспечения восстановления расфокусированных и смазанных изображений понадобится использование генеративно-состязательной сети. Программное обеспечение разрабатывается с использованием языка программирования Python.

Для того, чтобы начать процесс восстановления изображений, загрузим их в память программы. Разработанное программное обеспечение способно воспринимать большинство популярных форматов статических изображений.

Опираясь на разрабатываемый алгоритм, рассмотрим процесс восстановления изображений и отобразим его на рисунке 6.


Рисунок 6 – Процесс восстановления расфокусированных и смазанных изображений

Теперь перейдем непосредственно к написанию кода программного обеспечения. Для начала следует загрузить набор входных изображений в программу.

Фрагмент кода массовой загрузки изображений представлен на рисунке 7.

```
def blur_dectection(self):
    """
        метод загрузки изображений из директории
    """
    img_paths = "data/"
    image_path_list = []
    valid_image_extensions = [".jpg", ".jpeg", ".png", ".tif", ".tiff", ".bmp", ".jp2"]
    for file in os.listdir(img_paths):
        extension = os.path.splitext(file)[1]
        if extension.lower() not in valid_image_extensions:
            continue
        image_path_list.append(os.path.join(img_paths, file))
```

Рисунок 7 – Фрагмент кода загрузки изображений из директории

Загрузив изображения, следует определить, являются ли они смазанными. Переведем наши изображения в серые тона и вычислим распределение Лапласа.

Распределение Лапласа – непрерывное распределение случайной величины, при котором плотность вероятности вычисляется по формуле 8.

$$f(x) = \frac{\alpha}{2} e^{-\alpha|x-\beta|}, -\infty < x < \infty, \quad (8)$$

где α > 0 – параметр масштаба, -∞ < β < +∞ – параметр сдвига.

Экспериментальным путем было выявлено, что если полученное значение дисперсии будет меньше 100, то изображение является смазанным, в остальных случаях дефектов на изображениях нет.

После просчета дисперсии, наносим на изображение её значение и состояние (Blurry, not blurry). Фрагмент кода представлен на рисунке 8.

```
for image_path in image_path_list:
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # перевод изображения в серые тона
    laplacian_var = ImageProcessing.variance_of_laplacian(gray)
    state = "Not blurry"

    if laplacian_var < 100:
        state = "Blurry"

    cv2.putText(image, "{}: {:.2f}".format(state, laplacian_var), (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 3)

    if image is not None:
        cv2.imshow(image_path, image)
```

Рисунок 8 – Фрагмент кода подсчета дисперсии и определения состояния изображения

Для тестирования фрагмента кода возьмем исходный рисунок и импортируем его в программу. Результат показан на рисунке 9.

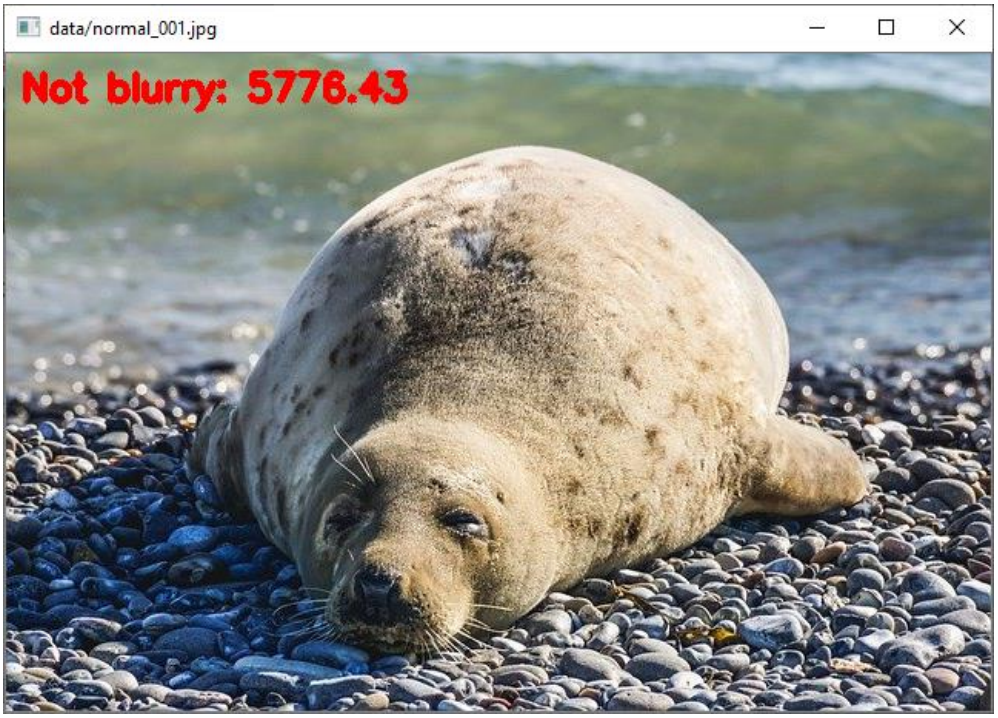



Рисунок 9 – Импортирование исходного изображения и подсчет функции

Из рисунка видно, что в результате работы программы был добавлен текст и определено распределение Лапласа.

Теперь намеренно смажем данное изображение. Для этого был написан отдельный код с использованием библиотеки компьютерного зрения OpenCV. Листинг кода для смаза изображения представлен на рисунке 10.

Переменная img отвечает за входное изображение, blur отвечает за смаз импортируемого изображения, при этом задается его размер, далее изображение показывается пользователю.

```
import cv2
import os
import numpy as np
from matplotlib import pyplot as plt

image_dir = "data/"
image_path_list = []
valid_image_extensions = [".jpg", ".jpeg", ".png", ".tif", ".tiff", ".bmp", ".jp2"]
for file in os.listdir(image_dir):
    extension = os.path.splitext(file)[1]
    if extension.lower() not in valid_image_extensions:
        continue
        image_path_list.append(os.path.join(image_dir, file))
    for image_path in image_path_list:
        image = cv2.imread(image_path)
        blur = cv2.blur(img, (5, 5))
        cv2.imwrite('data/image_path', image)
```

Рисунок 10 - Листинг кода для смаза изображения

Результат смаза представлен на рисунке 11.

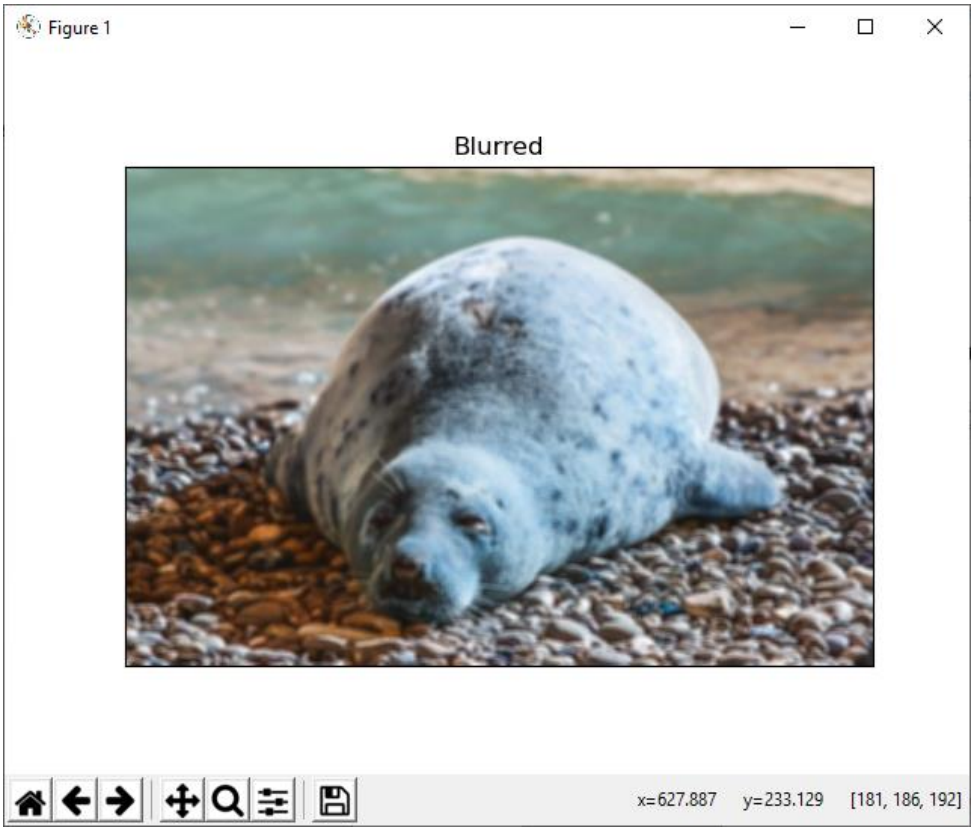



Рисунок 11 – Результат смаза изображения

Получив смазанное изображение, импортируем его в первоначальную программу определения смаза и запустим.

Результат показан на рисунке 12.

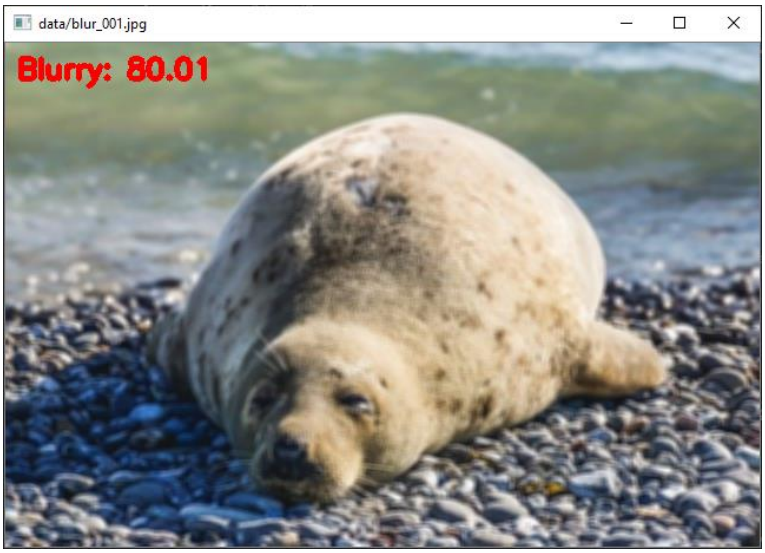



Рисунок 12 – результат определения смаза

В результате загрузки и обработки, результат определения смаза является положительным, алгоритм определил изображение как смазанное и вывел значение распределения Лапласа.

На следующем этапе приступим к обработке импортированного изображения и его деконволюции.

## 2.4 Разработка программного продукта на основе составленного алгоритма восстановления расфокусированных и смазанных изображений

В качестве основы для разрабатываемого алгоритма будем использовать свёрточную нейронную сеть VGG19, архитектура которой представлена на рисунке 13.

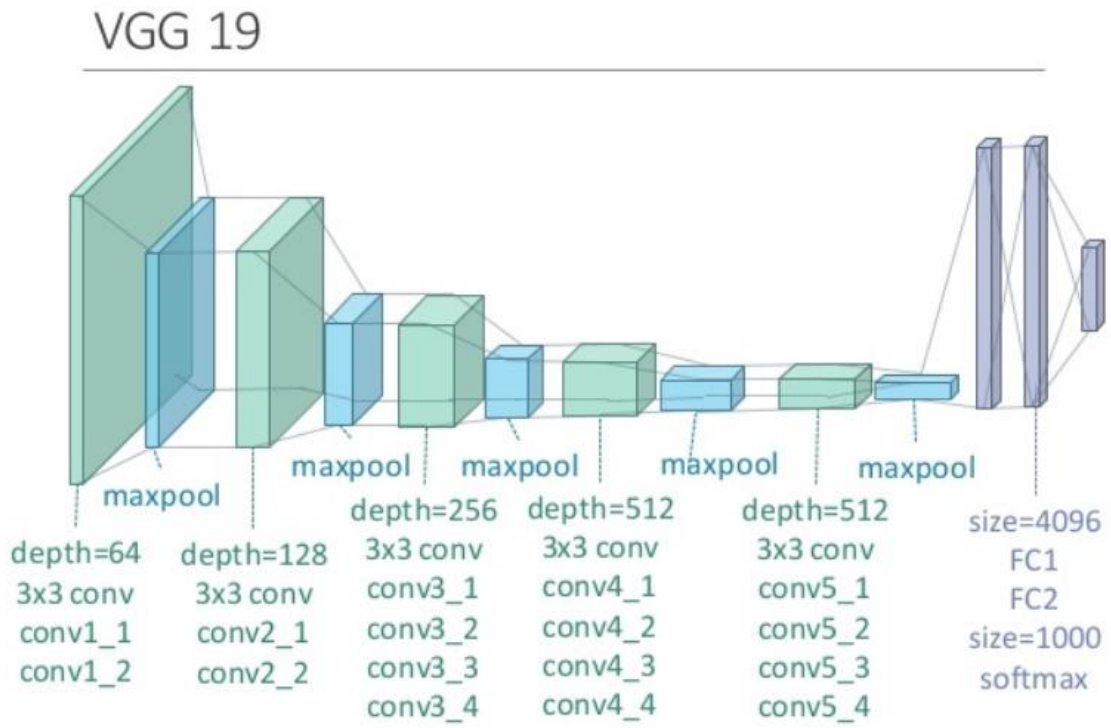



Рисунок 13 – Модель готовой свёрточной нейронной сети VGG19

VGG19 – модель свёрточной нейронной сети, которая была предложена К. Симоняном и А. Циссерманом из Оксфордского университета. Глубина сверточной нейронной сети составляет 19 слоёв, из которых 16 слоёв

являются свёрточными, а 3 оставшихся полносвязными. Основной идеей VGG-архитектур является использование большего числа слоёв с фильтрами меньшего размера [16].

VGG19 является улучшенной версией AlexNet. По сравнению с AlexNet, VGG19 представляет собой более глубокую свёрточную нейронную сеть с большим количеством слоёв [10].

Однако размер фильтров по сравнению с предыдущей версией не изменился. VGG19 так же содержит несколько фильтров размера 3x3, которые следуют один за другим, как это и было в VGG16. В AlexNet в первом и втором слоях располагались фильтры 11 и 5 [16].

Тем не менее, не смотря на то, что размер фильтра так и не поменялся, данная свёрточная нейронная сеть все же имеет свои преимущества. Основным преимуществом VGG19 является то, что для модели была достигнута средняя точность 95.0692 %. Данный показатель превышает среднюю точность модели VGG16 более чем на 2.3692% (точность VGG16 составляет 92.7%) [11].

Недостатком выбранной свёрточной нейронной сети является больший расход памяти в отличие от предшествующей VGG16.

На рисунке 14 показано сравнение VGG19 с сетями VGG16 и AlexNet.

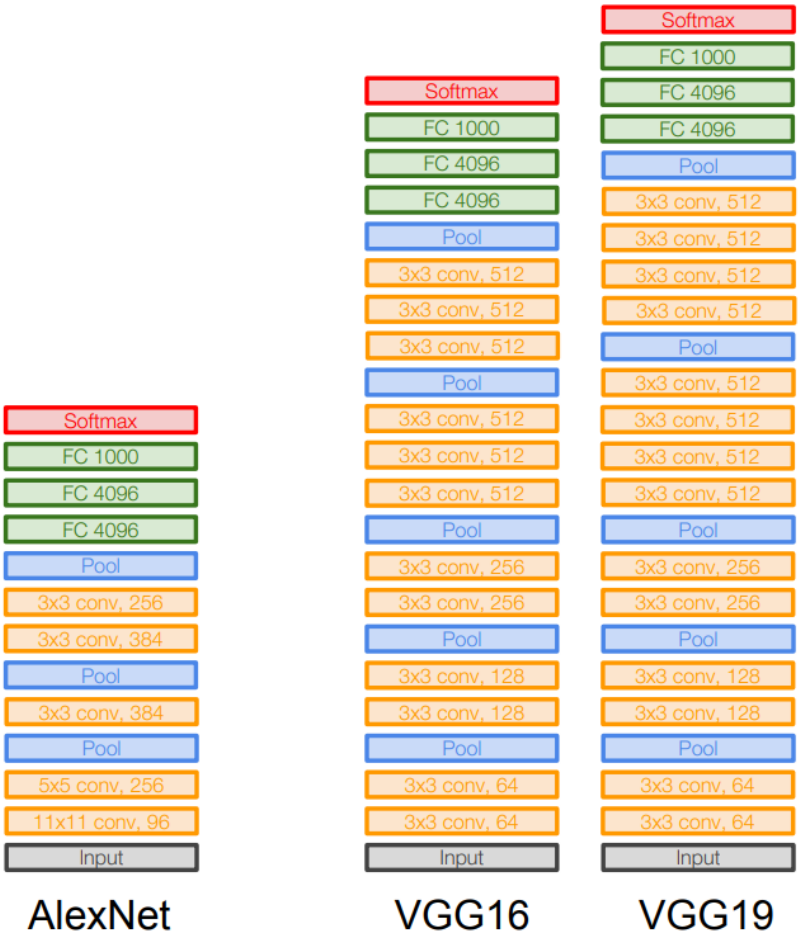



Рисунок 14 – Отличие VGG19 от VGG16 и AlexNet

Сеть VGG19 обучалась на протяжении нескольких недель с использованием видеокарт NVIDIA Titan Black. Модель обучалась с помощью оригинального эвристического алгоритма в режиме без учителя, а не классическим алгоритмом обратного распространения ошибки [11].

Архитектура свёрточной нейронной сети представляет из себя следующую структуру:

– Свёрточный слой [12];
– Слой подвыборки (пулинга);
– Слой активации;
– Полносвязный слой.

Свёрточный слой – самый главный слой сети. Основная задача данного слоя – выделение признаков на входном изображении и формирование карты

признаков [10]. Карта признаков – это тензор (массив матриц), в котором каждый канал отвечает за какой-нибудь выделенный признак. Чтобы слой мог выделять признаки, в нём имеются фильтры (ядра). Ядра — это набор тензоров, где каждый имеет один и тот же размер, а их количество определяет глубину выходного 3D массива. При этом глубина самих фильтров совпадает с количеством каналов входного изображения [12].

Чтобы сформировать карту признаков из входного изображения, производится операция свёртки входного тензора с каждым из фильтров. Свёртка – это операция вычисления нового значения выбранного пикселя, учитывающая значения окружающих его пикселей [25]. Алгоритм получения результата свёртки можно описать так: фильтр накладывается на левую верхнюю часть изображения и производится покомпонентное умножение значений фильтра и значений изображения, после чего фильтр перемещается дальше по изображению до тех пор, пока аналогичным образом не будут обработаны все его участки [24].

На рисунке 15 показано покомпонентное умножение значений фильтра и значений изображения.

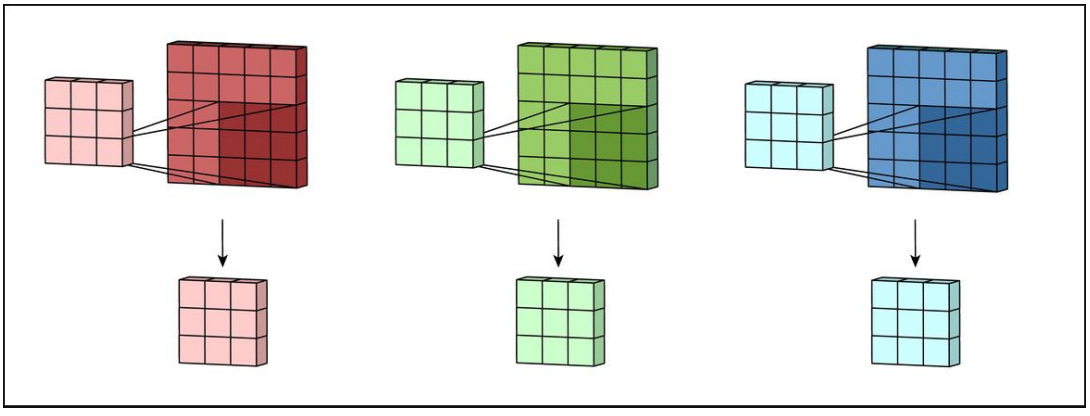


Рисунок 15 - Покомпонентное умножение значений фильтра и значений изображения

Числа полученных матриц суммируются в единую матрицу — результат применения фильтра. На рисунке 16 показано суммирование.

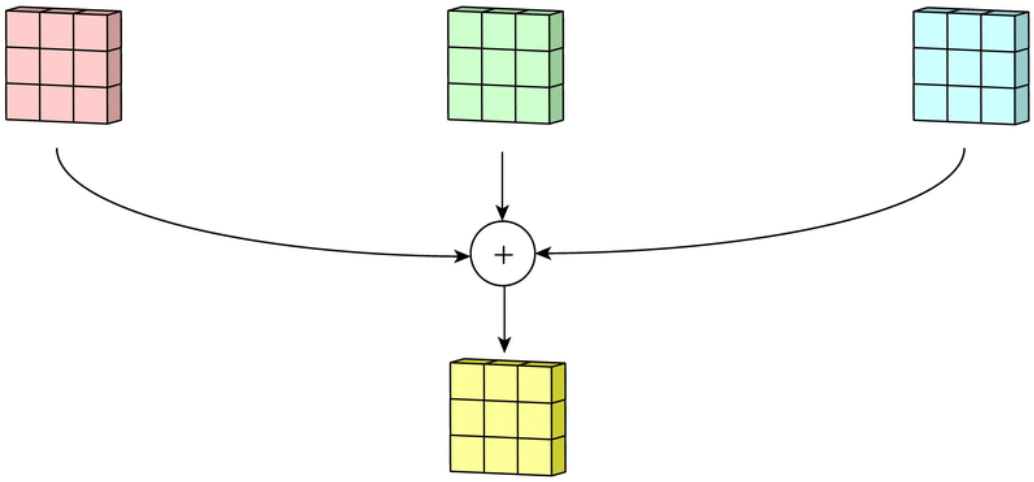


Рисунок 16 – Суммирование полученных матриц

После этого к каждому значению матрицы добавляется одинаковое число – значение смещения данного фильтра [22]. Полученная матрица составляет один канал выходной карты признаков. На рисунке 17 показано добавление числа.

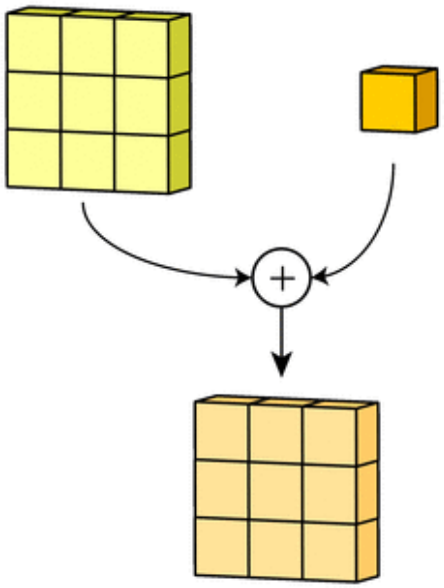


Рисунок 17 – Добавление числа

Затем, как будут получены каналы для каждого из фильтров, матрицы объединяются в единый тензор, благодаря чему на выходе снова получается изображение, с другим числом каналов, размером и количеством пикселей.

Далее рассмотрим слой пулинга.

Слой подвыборки (пулинга) является нелинейным уплотнением карты признаков, где группа пикселей размера 2x2 уплотняется до одного пикселя, при этом проходя нелинейное преобразование.

Преобразования происходят с непересекающимися прямоугольниками и квадратами, которые ужимаются в один пиксель. Далее выбирается пиксель, который имеет максимальное значение. Данная операция позволяет уменьшить пространственный объём изображения.

На рисунке 18 показан процесс пулинга.

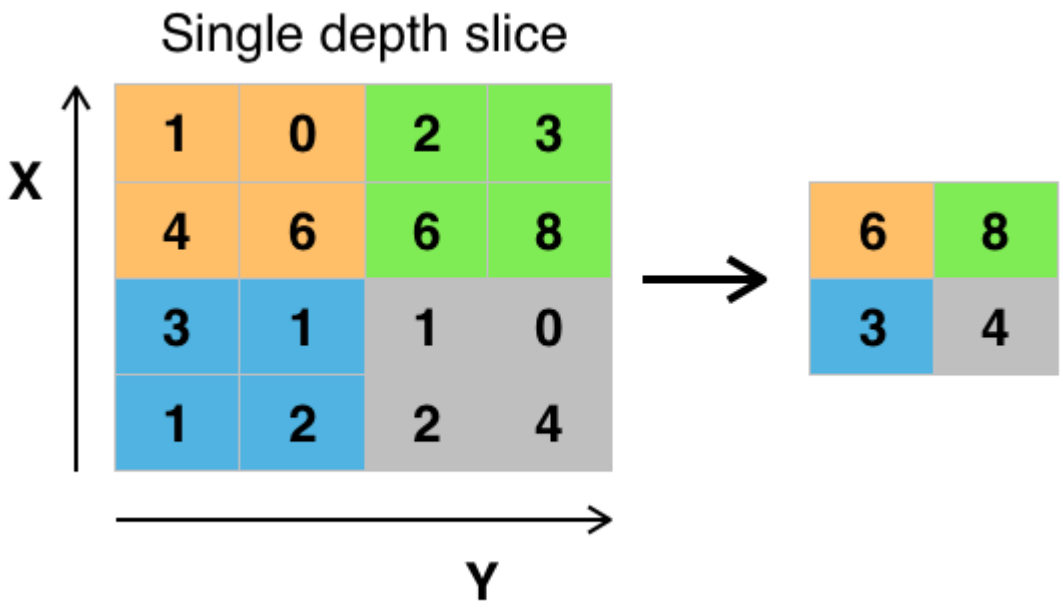



Рисунок 18 – Пример слоя подвыборки (пулинга)

Теперь перейдем к рассмотрению слоя активации.

Слой активации представляет из себя такой тип слоя, задачей которого является применение некоторой несложной функции к каждой точке входного тензора.

Он является некоторой функцией, которая применяется к каждому числу входного изображения.

Нелинейная функция может быть самостоятельно выбрана разработчиком.

Обычно функция активации может задаваться такими функциями как ReLU, Sigmoid, Tanh, LeakyReLU.

Наиболее часто используемой функцией активации является функция ReLU.

Данный слой может быть записан формулой 9.

$$x^l = f(a^l * subsample(x^{l-1}) + b^l), \quad (9)$$

где $x^l$ – выход слоя l, f() – функция активации, $a^l$, $b^l$ – коэффициенты сдвига слоя l, subsample() – операция выборки локальных максимальных значений.

Последним слоем нейронной сети является полносвязный слой. Принцип работы данного слоя заключается в следующем: после того как будут произведены несколько прохождений свёртки изображения и уплотнения с помощью слоя подвыборки, произойдёт перестроение от определенной сетки пикселей с высоким разрешением к наиболее абстрактным картам признаков, которые, зачастую с каждым слоем увеличивают число каналов и уменьшают размерность изображения.

Рассмотрев слои нейронной сети, перейдем к реализации самой программы на основе алгоритма восстановления. Начнем с написания фрагмента загрузки изображений.

Для начала пройдемся по каталогу с изображениями и загрузим их в память программы. Зададим алгоритму, что изображения загружены в цветовой палитре BGR и конвертируем их в RGB для обработки через библиотеку OpenCV.

Код загрузки и конвертации представлен на рисунке 19.

```
parser = ArgumentParser()
parser.add_argument('--image_dir', type=str, help='Каталог, где хранятся изображения')
parser.add_argument('--output_dir', type=str, help='Каталог выходных изображений.')

def main():
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
    args = parser.parse_args()

    # Получение путей к изображениям
    img_paths = [os.path.join(args.image_dir, x) for x in os.listdir(args.image_dir)]

    # Изменене формы ввода модели, чтобы принять все размеры ввода
    model = keras.models.load_model('models/generator.h5', compile=False)
    inputs = keras.Input((None, None, 3))
    output = model(inputs)
    model = keras.models.Model(inputs, output)

    # Проходим по всем изображениям
    for img_path in img_paths:
        low_res = cv2.imread(img_path, 1)
        low_res = cv2.cvtColor(low_res, cv2.COLOR_BGR2RGB) # Конвертирование в RGB
        low_res = low_res / 255.0 # Изменение масштаба изображений
        s_resolution = model.predict(np.expand_dims(low_res, axis=0))[0] # Получение изображения с высоким разрешением
        s_resolution = ((s_resolution + 1) / 2.) * 255 # Масштабирование значений в диапазоне 0-255
        s_resolution = cv2.cvtColor(s_resolution, cv2.COLOR_RGB2BGR) # Конвертирование обратно в BGR для OpenCV
        cv2.imwrite(os.path.join(args.output_dir, os.path.basename(img_path)), s_resolution) # Сохранение результатов
```

Рисунок 19 – Фрагмент кода алгоритма загрузки и конвертации изображений

Конвертируем входные изображения в RGB, изменим их масштаб, а также передадим в другой класс для проверки их размеров.

Код алгоритма загрузки изображений в другом классе и проверки размеров имеет вид, представленный на рисунке 20.

```
def img_parser(self, image_path):
    """
    Функция, которая загружает изображения по заданному пути.
    Args:
        image_path: путь к файлу изображения.
    Возвращает:
        image: тензор загруженного изображения.
    """

    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.convert_image_dtype(image, tf.float32)

    # Проверка, достаточно ли велико изображение
    if tf.keras.backend.image_data_format() == 'channels_last':
        shape = array_ops.shape(image)[:2]
    else:
        shape = array_ops.shape(image)[1:]
    cond = math_ops.reduce_all(shape >= tf.constant(self.img_size))

    image = tf.cond(cond, lambda: tf.identity(image),
                    lambda: tf.image.resize(image, [self.img_size, self.img_size]))

    return image
```

Рисунок 20 – Листинг кода алгоритма загрузки и проверки размеров изображений

В результате отработки методом получим тензор загруженного изображения – n-мерный массив базовых типов данных.

На следующем шаге создадим метод, целью которого является обрезка изображения в соответствии с заданной шириной и высотой (Рисунок 21).

```
def crop_imgs(self, image):
    """
    Функция, которая обрезает изображение в соответствии с определенной шириной
     и высотой.
     Args:
         изображение: тензор изображения.
     Возвращает:
         image: Tf тензор, содержащий обрезанное изображение.
    """

    image = tf.image.random_crop(image, [self.img_size, self.img_size, 3])

    return image
```

Рисунок 21 – Листинг кода метода, выполняющего обрезку изображений

После того как изображение обрезано, следует сгенерировать изображение низкого разрешения с учетом изображения с высоким разрешением. В качестве значения коэффициента понижающей дискретизации возьмем значение 4. То есть уменьшим разрешение входного изображения в 4 раза. Листинг кода показан на рисунке 22.

```
def hl_resolution(self, h_resolution):
    """
    Функция, которая генерирует изображение низкого разрешения с учетом
     изображения с высоким разрешением. Коэффициент понижающей дискретизации составляет 4x.
     Args:
         h_resolution: тензор высокого разрешения изображения.
     Возвращает:
         l_resolution: Tf-тензор изображения с низким разрешением.
         h_resolution: тензор высокого разрешения изображения.
    """

    l_resolution = tf.image.resize(h_resolution,
                                   [self.img_size // 4, self.img_size // 4],
                                   method='bicubic')

    return l_resolution, h_resolution
```

Рисунок 22 – Листинг кода метода, выполняющего уменьшение разрешения изображения

Теперь изменим значения пикселей в диапазоне [-1; 1]. Листинг кода показан на рисунке 23.

```
def rescale(self, l_resolution, h_resolution):
    """
    Функция, которая изменяет значения пикселей в диапазоне от -1 до 1.
     Для использования с выходной функцией генератора.
     Args:
         l_resolution: tf-тензор изображения с низким разрешением.
         h_resolution: tf-тензор изображения с высоким разрешением.
     Возвращает:
         l_resolution: tf-тензор изображения с низким разрешением, масштабированный.
         h_resolution: tf-тензор изображения с высоким разрешением, масштабированный.
    """
    h_resolution = h_resolution * 2.0 - 1.0

    return l_resolution, h_resolution
```

Рисунок 23 – Листинг кода метода изменения значений пикселей

Следом, как операции с изображениями проведены, перейдём к этапу обучения нейронной сети. В отдельный класс передадим необходимые параметры размеров для изображений, а также зададим количество инвертированных остаточных блоков в генераторе сети. Листинг кода представлен на рисунке 24.

```
class RestoreGAN(object):
    def __init__(self, args):
        """
        Инициализирует класс RestoreGAN.
         Args:
             args: аргументы CLI, которые определяют, как построить модель.
        """
        self.hr_height = args.hr_size
        self.hr_width = args.hr_size
        self.lr_height = self.hr_height // 4  # Высота низкого разрешения
        self.lr_width = self.hr_width // 4  # Ширина низкого разрешения
        self.lr_shape = (self.lr_height, self.lr_width, 3)
        self.hr_shape = (self.hr_height, self.hr_width, 3)
        self.iterations = 0

        # Количество инвертированных остаточных блоков в генераторе сети
        self.n_residual_blocks = 6
```

Рисунок 24 – Определение параметров нейронной сети

Далее, зададим график снижения скорости обучения. Листинг кода представлен на рисунке 25.

```
# Определить график снижения скорости обучения.
self.gen_schedule = keras.optimizers.schedules.ExponentialDecay(
    args.lr,
    decay_steps=100000,
    decay_rate=0.1,
    staircase=True
)

self.disc_schedule = keras.optimizers.schedules.ExponentialDecay(
    args.lr * 5,  # TTUR - Two Time Scale Updates
    decay_steps=100000,
    decay_rate=0.1,
    staircase=True
)

self.gen_optimizer = keras.optimizers.Adam(learning_rate=self.gen_schedule)
self.disc_optimizer = keras.optimizers.Adam(learning_rate=self.disc_schedule)
```

Рисунок 25 - Задание графика снижения скорости обучения

Затем, используем предварительно обученную модель VGG19 для извлечения характеристик изображения низкого разрешения и используем сгенерированные характеристики изображения с высоким разрешением, а также минимизируем MSE (среднюю квадратичную ошибку) между ними.

Листинг кода использования модели, генерации характеристик и минимизации MSE показан на рисунке 26.

```
# использование предварительно обученной модели VGG19
# для извлечения характеристик изображения низкого разрешения
# использование сгенерированных характеристик изображения с высоким разрешением
# минимизация MSE

self.vgg = self.build_vgg()
self.vgg.trainable = False
```

Рисунок 26 – Инициализация предварительно обученной модели VGG19

Среднеквадратическая ошибка высчитывается по формуле 10.

$$l^{SR}_{VGG/i.g} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{Wi,j} \sum_{y=1}^{Hi,j} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}\left(\phi_{\theta_G}(I^{LR})\right)_{x,y})^2 \quad (10)$$

где $\phi_{i,j}$ – карта характеристик для j-й свертки (после активации) перед i-м слоем максимального объединения.

Рассчитаем выходную форму D, а также зададим количество фильтров в первом слое G и D. Листинг кода представлен на рисунке 27.

```
# Рассчёт выходной формы D
patch = int(self.hr_height / 2 ** 4)
self.disc_patch = (patch, patch, 1)

# Количество фильтров в первом слое G и D
self.gf = 32  # Улучшение изображения в реальном времени.
self.df = 32
```

Рисунок 27 – Расчет выходной формы D и задание количества фильтров

После, соберём дискриминатор и генератор. Листинг кода представлен на рисунке 28.

```
# Сборка дискриминатора
self.discriminator = self.build_discriminator()

# Сборка генератора для предварительной подготовки.
self.generator = self.build_generator()
```

Рисунок 28 – Сборка дискриминатора и генератора

Теперь создадим модель VGG19, которая выводит функции изображения, извлеченные на третий блок модели. Фрагмент кода показан на рисунке 29.

```
def build_vgg(self):
    """
    Создает модель VGG19, которая выводит функции изображения, извлеченные на
     третий блок модели
    """
    # Получение сети VGG. Извлечение функций из блока 5, последняя свертка.
    vgg = keras.applications.VGG19(weights="imagenet", input_shape=self.hr_shape, include_top=False)
    vgg.trainable = False
    for layer in vgg.layers:
        layer.trainable = False

    # Создать модель и скомпилировать
    model = keras.models.Model(inputs=vgg.input, outputs=vgg.get_layer("block5_conv4").output)

    return model
```

Рисунок 29 – Создание модели VGG19

Построим генератор, который будет выполнять задачу построения моделей для улучшения качества изображения. Листинг кода показан на рисунке 30.

```
def build_generator(self):
    """Построение генератора, который будет выполнять задачу улучшения качества изображения."""

    def mk_divisible(v, divisor, min_value=None):
        if min_value is None:
            min_value = divisor
        new_v = max(min_value, int(v + divisor / 2) // divisor * divisor)
        # Проверка, что округление вниз не опускается более чем на 10%
        if new_v < 0.9 * v:
            new_v += divisor
        return new_v
```

Рисунок 30 – Построение генератора

Зададим инвертированный остаточный блок. Его задача заключается в использовании глубины сверток для проверки эффективности заданных параметров. Листинг кода показан на рисунках 31, 32.

```
def residual_block(inputs, filters, block_id, expansion=6, stride=1, alpha=1.0):
    """Инвертированный остаточный блок, который использует глубину сверток для проверки эффективности параметров.
    Args:
        inputs: карта входных объектов.
        filters: количество фильтров в каждой свертке в блоке.
        block_id: целочисленный спецификатор для идентификатора блока в графе.
        expansion: коэффициент расширения канала.
        stride: шаг свертки.
        alpha: коэффициент расширения глубины.
    Возвращает:
        x: выход инвертированного остаточного блока.
    """
    channel_axis = 1 if keras.backend.image_data_format() == 'channels_first' else -1

    in_channels = keras.backend.int_shape(inputs)[channel_axis]
    pointwise_conv_filters = int(filters * alpha)
    pointwise_filters = mk_divisible(pointwise_conv_filters, 8)
    x = inputs
    prefix = 'block_{}_'.format(block_id)

    if block_id:
        # Увеличение разрешения
        x = keras.layers.Conv2D(expansion * in_channels,
                                kernel_size=1,
                                padding='same',
                                use_bias=True,
                                activation=None,
                                name=prefix + 'expand')(x)
        x = keras.layers.BatchNormalization(axis=channel_axis,
                                            epsilon=1e-3,
                                            momentum=0.999,
                                            name=prefix + 'expand_BN')(x)
        x = keras.layers.Activation('relu', name=prefix + 'expand_relu')(x)
```

Рисунок 31 - Задание инвертированного остаточного блока

```
else:
    prefix = 'expanded_conv_'

# Увеличение глубины
x = keras.layers.DepthwiseConv2D(kernel_size=3,
                                 strides=stride,
                                 activation=None,
                                 use_bias=True,
                                 padding='same' if stride == 1 else 'valid',
                                 name=prefix + 'depthwise')(x)
x = keras.layers.BatchNormalization(axis=channel_axis,
                                    epsilon=1e-3,
                                    momentum=0.999,
                                    name=prefix + 'depthwise_BN')(x)

x = keras.layers.Activation('relu', name=prefix + 'depthwise_relu')(x)


x = keras.layers.Conv2D(pointwise_filters,
                        kernel_size=1,
                        padding='same',
                        use_bias=True,
                        activation=None,
                        name=prefix + 'project')(x)
x = keras.layers.BatchNormalization(axis=channel_axis,
                                    epsilon=1e-3,
                                    momentum=0.999,
                                    name=prefix + 'project_BN')(x)

if in_channels == pointwise_filters and stride == 1:
    return keras.layers.Add(name=prefix + 'add')([inputs, x])
return x
```

Рисунок 32 - Задание инвертированного остаточного блока (продолжение)

На данном этапе задается процесс увеличения разрешения слоя изображения, а также увеличения глубины.

Функция DepthwiseConv2d выполняет первый шаг глубокой пространственной свертки, применяя операцию свертки для каждого входного канала отдельно.

Функция BatchNormalization производит нормализацию и масштабирование входов или активаций.

Затем опишем слой, повышающий дискретизацию для увеличения высоты и ширины вводимого изображения.

Листинг кода повышения дискретизации представлен на рисунке 33.

```
    def deconv2d(layer_input):
        """Слой, повышающий дискретизацию для увеличения высоты и ширины ввода.
        Использует PixelShuffle для повышения частоты дискретизации.
        Args:
            layer_input: входной тензор для повышения.
        Возвращает:
            u: повышенный коэффициент ввода в 2 раза.
        """
        u = keras.layers.UpSampling2D(size=2, interpolation='bilinear')(layer_input)
        u = keras.layers.Conv2D(self.gf, kernel_size=3, strides=1, padding='same')(u)
        u = keras.layers.PReLU(shared_axes=[1, 2])(u)
        return u

    # Ввод изображения с низким разрешением
    img_lr = keras.Input(shape=self.lr_shape)

    # Предварительно-остаточный блок
    c1 = keras.layers.Conv2D(self.gf, kernel_size=3, strides=1, padding='same')(img_lr)
    c1 = keras.layers.BatchNormalization()(c1)
    c1 = keras.layers.PReLU(shared_axes=[1, 2])(c1)

    # Распространение через остаточные блоки
    r = residual_block(c1, self.gf, 0)
    for idx in range(1, self.n_residual_blocks):
        r = residual_block(r, self.gf, idx)

    # Пост-остаточный блок
    c2 = keras.layers.Conv2D(self.gf, kernel_size=3, strides=1, padding='same')(r)
    c2 = keras.layers.BatchNormalization()(c2)
    c2 = keras.layers.Add()([c2, c1])

    # повышающая дискретизация
    u1 = deconv2d(c2)
    u2 = deconv2d(u1)
    # Генерация вывода с высоким разрешением
    gen_hr = keras.layers.Conv2D(3, kernel_size=3, strides=1, padding='same', activation='tanh')(u2)
    return keras.models.Model(img_lr, gen_hr)
```

Рисунок 33 – Слой повышения дискретизации

На данном этапе производится импортирование изображения с низким разрешением, после задаётся предварительно-остаточный блок.

На следующем шаге производится описание распространения через остаточные блоки, задаётся пост-остаточный блок и задаётся процесс повышения дискретизации. После этого задаётся процесс вывода изображения с высоким разрешением.

А также создадим сеть дискриминатора. Листинг кода представлен на рисунке 34.

```
def build_discriminator(self):
    """Создает сеть дискриминатора"""

    def d_block(layer_input, filters, strides=1, bn=True):
        """Блок слоя дискриминатора.
        Args:
            layer_input: карта входных объектов для сверточного блока.
            filters: количество фильтров в свертке.
            steps: шаги свертки.
            bn: использовать ли пакетную норму или нет.
        """
        d = keras.layers.Conv2D(filters, kernel_size=3, strides=strides, padding='same')(layer_input)
        if bn:
            d = keras.layers.BatchNormalization(momentum=0.8)(d)
        d = keras.layers.LeakyReLU(alpha=0.2)(d)

        return d

    # Импорт изображения
    d0 = keras.layers.Input(shape=self.hr_shape)

    d1 = d_block(d0, self.df, bn=False)
    d2 = d_block(d1, self.df, strides=2)
    d3 = d_block(d2, self.df)
    d4 = d_block(d3, self.df, strides=2)
    d5 = d_block(d4, self.df * 2)
    d6 = d_block(d5, self.df * 2, strides=2)
    d7 = d_block(d6, self.df * 2)
    d8 = d_block(d7, self.df * 2, strides=2)

    validity = keras.layers.Conv2D(1, kernel_size=1, strides=1, activation='sigmoid', padding='same')(d8)

    return keras.models.Model(d0, validity)
```

Рисунок 34 – Создание сети дискриминатора

Для обучения дискриминатора функция потерь использует типичную потерю дискриминатора GAN, которая рассчитывается по формуле 11.

$$\min_{\theta_G} \max_{\theta_D} E_{I^{HR}\sim p_{train}(I^{HR})}[\log D_{\theta_D}(I^{HR})] + E_{I^{LR}\sim pG(I^{LR})}[\log(1 - D_\theta\left(G_{\theta_G}(I^{LR})\right)] \quad (11)$$

где LR – изображение с низким разрешением, HR – выходное изображение с высоким разрешением.

После того, как значения пикселей были изменены, вернём объект набора данных с указанными сопоставлениями. Листинг кода представлен на рисунке 35.

```python
def dataset(self, batch_size, threads=4):
    """
    Возвращает объект набора данных tf с указанными сопоставлениями.
     Args:
         batch_size: Int, Количество элементов в пакете, возвращаемых набором данных.
         потоки: Int, потоки процессора, используемые для многопоточной работы.
     Возвращает:
         набор данных: объект набора данных tf.
    """

    # Генерация набора данных tf из путей изображений с высоким разрешением.
    dataset = tf.data.Dataset.from_tensor_slices(self.img_paths)

    # Загрузка изображений
    dataset = dataset.map(self.img_parser, num_parallel_calls=tf.data.experimental.AUTOTUNE)

    # Обрезка куска для обучения
    dataset = dataset.map(self.crop_imgs, num_parallel_calls=tf.data.experimental.AUTOTUNE)

    # Имитация низкого разрешения путем снижения частоты дискретизации
    dataset = dataset.map(self._high_l_resolution_pairs, num_parallel_calls=tf.data.experimental.AUTOTUNE)

    # Масштабирование значения во входных данных
    dataset = dataset.map(self.rescale, num_parallel_calls=tf.data.experimental.AUTOTUNE)

    # Предварительная выборка данных для оптимального использования GPU.
    dataset = dataset.shuffle(30).batch(batch_size, drop_remainder=True).prefetch(tf.data.experimental.AUTOTUNE)

    return dataset
```

Рисунок 35 – Листинг кода метода возврата объекта набора данных

После построения алгоритма опишем тренировку генератора.

На первом шаге следует подготовить генератор, передав в него объект модели со скомпилированным генератором. Листинг кода представлен на рисунке 36.

```python
@tf.function
def stepbystep_pretrain(model, x, y):
    """
    Первый шаг предварительной подготовки генератора.
     Args:
         model: объект модели с скомпилированным генератором tf keras.
         x: тензор изображения низкого разрешения.
         y: тензор изображения с высоким разрешением.
    """
    with tf.GradientTape() as tape:
        fake_hr = model.generator(x)
        loss_mse = tf.keras.losses.MeanSquaredError()(y, fake_hr)

    grads = tape.gradient(loss_mse, model.generator.trainable_variables)
    model.gen_optimizer.apply_gradients(zip(grads, model.generator.trainable_variables))

    return loss_mse
```

Рисунок 36 – Первый шаг предварительной подготовки генератора

Затем зададим тренировку генератора для того, чтобы избежать локальных минимумов (Рисунок 37).

```python
def pretrain_gen(model, dataset, writer):
    """функция, которая тренирует генератор, чтобы избежать локальных минимумов.
     Args:
         model: модель Keras для тренировки.
         dataset: объект набора данных tf изображений с низким и высоким разрешением для предварительной подготовки.
         writer: объект записи.
    """
    with writer.as_default():
        iteration = 0
        for _ in range(1):
            for x, y in dataset:
                loss = stepbystep_pretrain(model, x, y)
                if iteration % 20 == 0:
                    tf.summary.scalar('MSE Loss', loss, step=tf.cast(iteration, tf.int64))
                    writer.flush()
                iteration += 1
```

Рисунок 37 – Тренировка генератора, для избежания локальных минимумов

Впоследствии запустим функцию пошаговой тренировки.

Произведем сглаживание меток изображения для лучшего градиентного потока. После при помощи генератора из изображения с низким разрешением запустим генерацию изображения с высоким разрешением, а также запустим тренировку дискриминатора. Листинг кода показан на рисунке 38.

```python
# Сглаживание меток для лучшего градиентного потока
valid = tf.ones((x.shape[0],) + model.disc_patch)
fake = tf.zeros((x.shape[0],) + model.disc_patch)

with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
    # Изображение с низким разрешением генерирует версию с высоким разрешением
    fake_hr = model.generator(x)

    # Тренировка дискриминаторов
    valid_prediction = model.discriminator(y)
    fake_prediction = model.discriminator(fake_hr)
```

Рисунок 38 – Листинг кода сглаживания меток и тренировки дискриминаторов

А также посчитаем потери генератора и дискриминатора. Листинг кода представлен на рисунке 39.

```
# Потери генератора
cnt_loss = model.cnt_loss(y, fake_hr)
adv_loss = 1e-3 * tf.keras.losses.BinaryCrossentropy()(valid, fake_prediction)
mse_loss = tf.keras.losses.MeanSquaredError()(y, fake_hr)
perceptual_loss = cnt_loss + adv_loss + mse_loss

# Потери дискриминатора
valid_loss = tf.keras.losses.BinaryCrossentropy()(valid, valid_prediction)
fake_loss = tf.keras.losses.BinaryCrossentropy()(fake, fake_prediction)
d_loss = tf.add(valid_loss, fake_loss)
```

Рисунок 39 – Подсчет потерь генератора и дискриминатора

После того как потери посчитаны, следует посчитать обратное распространение ошибки. Для начала введём для этого термина понятие.

Обратное распространение ошибки – метод вычисления градиента, который используется при обновлении весов в нейронной сети.

Посчитаем обратное распространение ошибки на генераторе и дискриминаторе. Листинг кода представлен на рисунке 40.

```
# Метод обратного распространения ошибки на генераторе
gen_grads = gen_tape.gradient(perceptual_loss, model.generator.trainable_variables)
model.gen_optimizer.apply_gradients(zip(gen_grads, model.generator.trainable_variables))

# Метод обратного распространения ошибки на дискриминаторе
disc_grads = disc_tape.gradient(d_loss, model.discriminator.trainable_variables)
model.disc_optimizer.apply_gradients(zip(disc_grads, model.discriminator.trainable_variables))
```

Рисунок 40 – Листинг кода подсчета обратного распространения ошибки на генераторе и дискриминаторе

После того, как все вышеописанные действия будут пройдены, изображение, полученное в результате обработки нейронной сетью, будет сохранено в выходной каталог.

Запустим разработанную нейронную сеть и посмотрим на результат восстановления после обучения.

Для проверки работы разработанного алгоритма было взято свободное изображение с открытого фотостока. После чего изображение было смазано. Применив к нему разработанный программный код, основанный на описанном алгоритме, было получено выходное изображение с более четкой детализацией. Выходное изображение показано на рисунке 41.

Рисунок 41 – Результат обработки изображения алгоритмом

Ядро смаза для исходного изображения выглядит следующим образом (Рисунок 42).

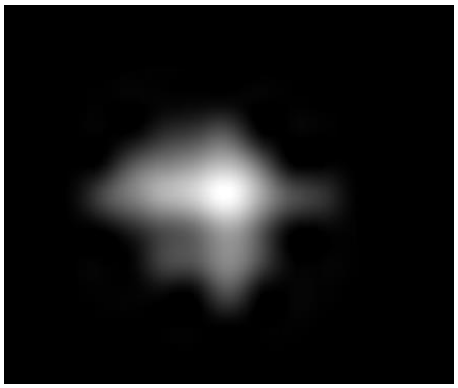


Рисунок 42 – Ядро смаза изображения

На приведенных выше рисунках показано выходное изображение, которое было сделано в результате обработки алгоритмом, а также определено ядро смаза изображения.

Отчет о восстановленном изображении показан на рисунке 43.

```
Input File: test.jpg
Blur Size: 100 px
Kernel Metric: 41 %
Defect Type: Auto-detect Blur
Analyzing Region: all image
Smoothness: 1% (Medium)
```

Рисунок 43 – Отчет о восстановленном изображении

Отчёт результата восстановления содержит в себе размер смаза.

В результате описания представленного пункта был продемонстрирован процесс разработки программного продукта, основанного на составленном алгоритме восстановления расфокусированных и смазанных изображений.

**Выводы по второй части:**

В результате описываемой в данной части работы было реализован программный продукт, который получает смазанные изображения от внешней системы и на их основе строит новое изображение при помощи алгоритма, составленного на основе свёрточной нейронной сети VGG19.

Разработанный программный продукт на основе составленного алгоритма выполняет следующие шаги:

– загружает набор изображений;

– определяет распределение Лапласа;

– переводит расфокусированное изображение в bgr палитру;

– производит описанные выше операции с изображениями для их восстановления;

– на основе свёрточной нейронной сети vgg19 строится новое восстановленное изображение.

Основным минусом разработанного алгоритма является долгое обрабатывание изображений на слабых процессорах.

Описав процесс разработки во второй части, перейдём к этапу тестирования разработанного программного обеспечения.

# 3 Тестирование алгоритма восстановления расфокусированных и смазанных изображений

## 3.1 Анализ результатов разработки программного продукта на основе созданного алгоритма восстановления расфокусированных и смазанных изображений

В результате данной бакалаврской работы было реализовано программное обеспечение на основе составленного алгоритма, которое принимает расфокусированные и смазанные изображения из внешних каталогов, определяет коэффициент смаза, а также восстанавливает полученные изображения, применяя к ним разработанный алгоритм с использованием свёрточной нейронной сети VGG19.

Тестирование разработанного алгоритма является важной частью реализации программного обеспечения. Для принятия решения о результате разработанного алгоритма необходимо провести ряд тестов, которые позволят проверить его работоспособность в реальных условиях.

В качестве оценки работы алгоритма, были проведены тестирования алгоритма восстановления. Ключевыми параметрами алгоритма восстановления являются: скорость и качество генерации выходного изображения.

Для проведения тестирования загрузим в разработанное программное обеспечение набор изображений, размеры которых отличаются друг от друга, замерим время восстановления каждого изображения, а также сравним их между собой.

В качестве тестируемого набора, был взят датасет из 800 изображений, разрешения которых составляют 525x750, 602x898, 750x1050, 898x1200, 1200x1800, 1800x2395 пикселей. Выбор описанных разрешений не случаен. Объясняется он тем, что приведенные размеры изображений соответствуют размерам для печати на принтере и в фотолаборатории при разрешающей способности принтера 150 dpi. В таблице 1 приведены результаты обработки и восстановления расфокусированных и смазанных изображений во времени.

Таблица 1 – Время обработки и восстановления расфокусированных и смазанных изображений.

| Размер изображения в пикселях | Время, затраченное на его обработку и восстановление в секундах |
|---|---|
| 525x750 | 32.40 |
| 602x898 | 33.80 |
| 750x1050 | 40 |
| 898x1200 | 49.35 |
| 1200x1800 | 52.70 |
| 1800x2395 | 62.15 |

Основываясь на данных таблицы, можно сделать вывод, что с увеличением размера изображения увеличивается и время, затрачиваемое на его восстановление.

Построим график зависимости времени, затраченного на восстановление изображений от разрешения на основе приведенной таблицы.

Данный график изображен на рисунке 44.

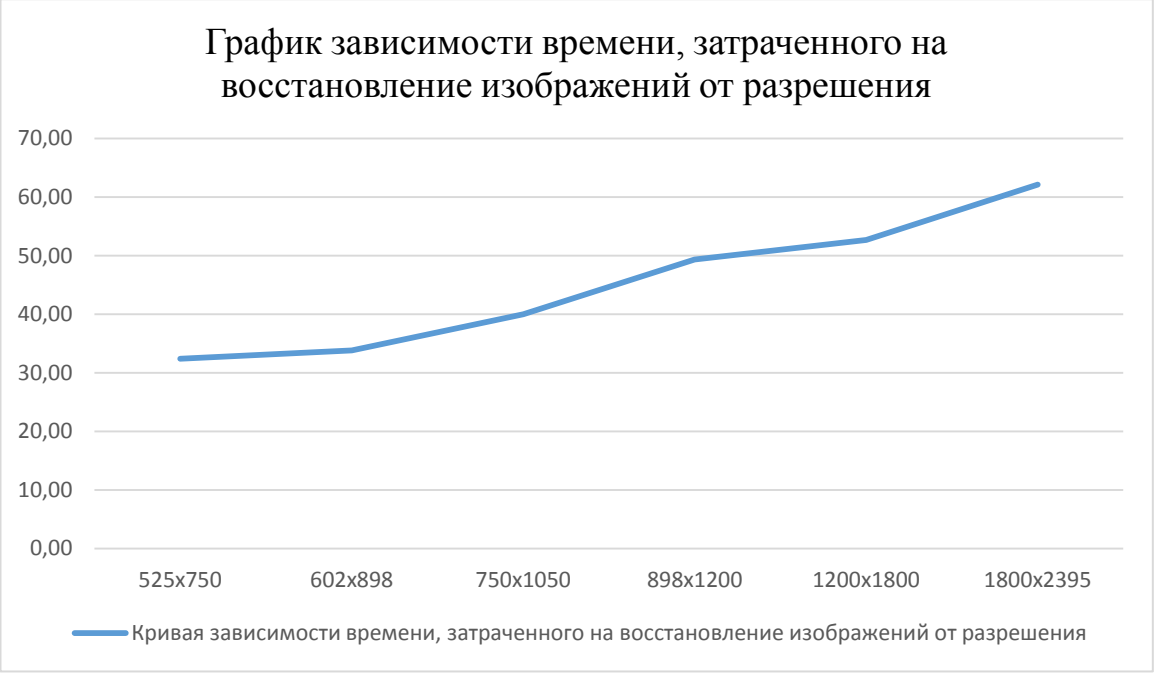



Рисунок 44 – График зависимости времени, затраченного на восстановление изображений от разрешения

В результате проведения тестирования, на основе кривой графика можно сделать вывод, что время, затраченное на восстановление

изображения, зависит от разрешения изображения, импортируемого в программный продукт. Чем больше разрешение изображения, тем больше времени требуется алгоритму на его восстановление.

## 3.2 Сравнительный анализ результатов восстановления изображений различными алгоритмами

После того, как результаты обработки изображений алгоритмом были получены, сравним его с уже существующими алгоритмами.

В таблице 2 приведено сравнение результатов скорости обработки изображений разработанным алгоритмом с существующей фильтрацией Винера и алгоритмом Люси-Ричардсона.

Замеры производились на изображениях с ранее описываемыми размерами.

Таблица 2 – Сравнение результатов скорости обработки изображений алгоритмами

| Алгоритм / Размер изображения | 525x750 | 602x898 | 750x1050 | 898x1200 | 1200x1800 | 1800x2395 |
|---|---|---|---|---|---|---|
| Фильтрация Винера | 33.60 с | 37.89 с | 44.21 с | 51 с | 59.35 с | 67.41 с |
| Алгоритм Люси-Ричардсона | 33.80 с | 38.01 с | 45.30 с | 52.70 с | 61.12 с | 69.13 с |
| Разработанный алгоритм | 32.40 с | 33.80 с | 40 с | 49.35 с | 52.70 с | 62.15 с |

По таблице 2 построим график зависимости времени, затрачиваемого на восстановление одного изображения в зависимости от его разрешения.

Данный график изображен на рисунке 45.

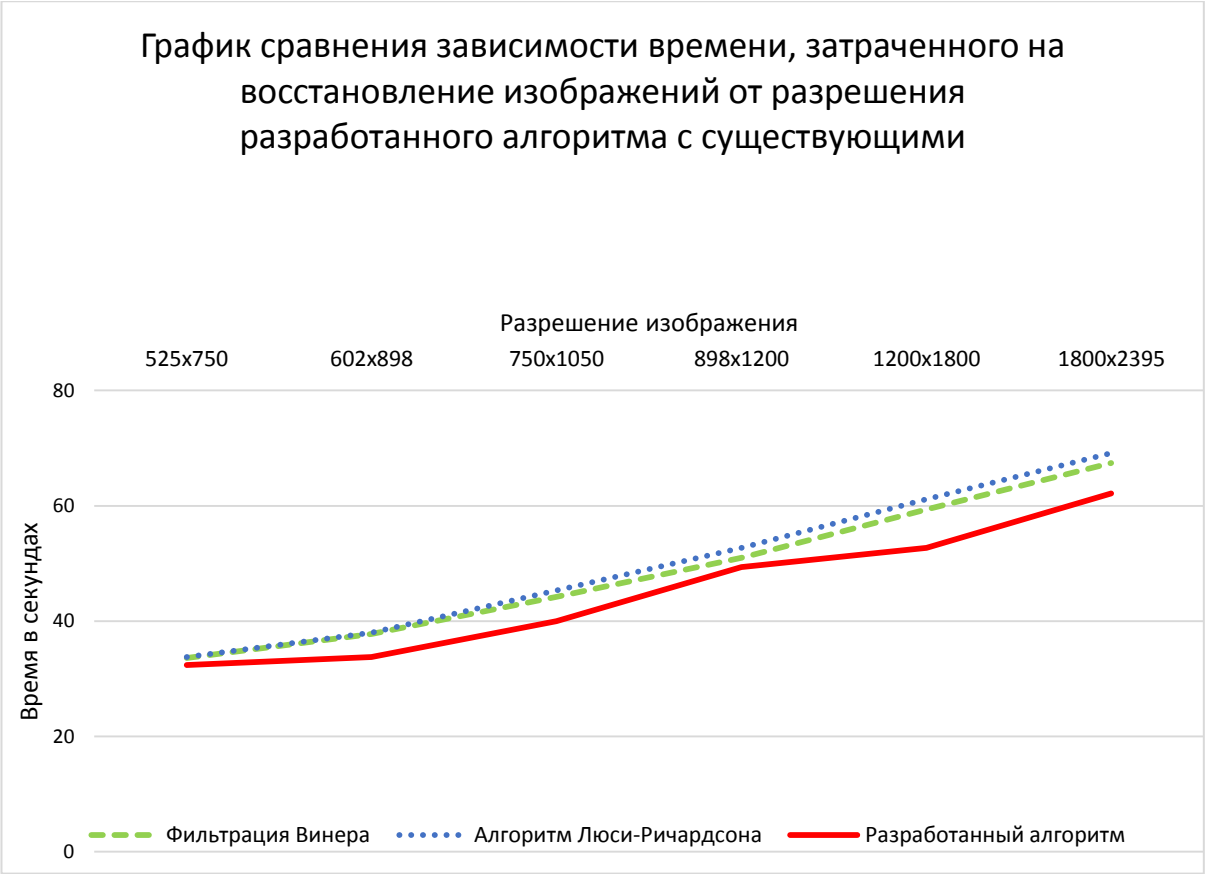



Рисунок 45 – График сравнения зависимости времени, затраченного на восстановление изображений от разрешения разработанного алгоритма с существующими

Опираясь на результаты таблицы и построенный график, можно сделать вывод, алгоритм на несколько секунд быстрее обрабатывает изображения, чем существующие, что делает его конкурентоспособным.

Перейдём к демонстрации результатов работы.

## 3.3 Демонстрация работы программного обеспечения на основе алгоритма восстановления расфокусированных и смазанных изображений

Для осуществления демонстрации результатов работы алгоритма были отобраны случайные смазанные фотографии с разной степенью смаза.

В связи с тем, что алгоритм выполняется итерационно, рассмотрим результаты восстановления одного изображения на примере нескольких шагов.

На рисунке 46 представлены результаты восстановления изображения на различных итерациях.

Рисунок 46 – Результат обработки изображения на различных итерациях

Уменьшение коэффициента смаза показано от крайнего левого изображения до крайнего правого. На первом изображении коэффициент смаза составляет 100%, данное изображение является исходным. На втором изображении коэффициент смаза составляет 70%, на третьем – 40%, на четвертом – 15%, на пятом – 1%. Достигается данный эффект за счет увеличения итераций применения алгоритма к входному изображению.

Конечный результат обработки изображения алгоритмом представлен на рисунке 47.

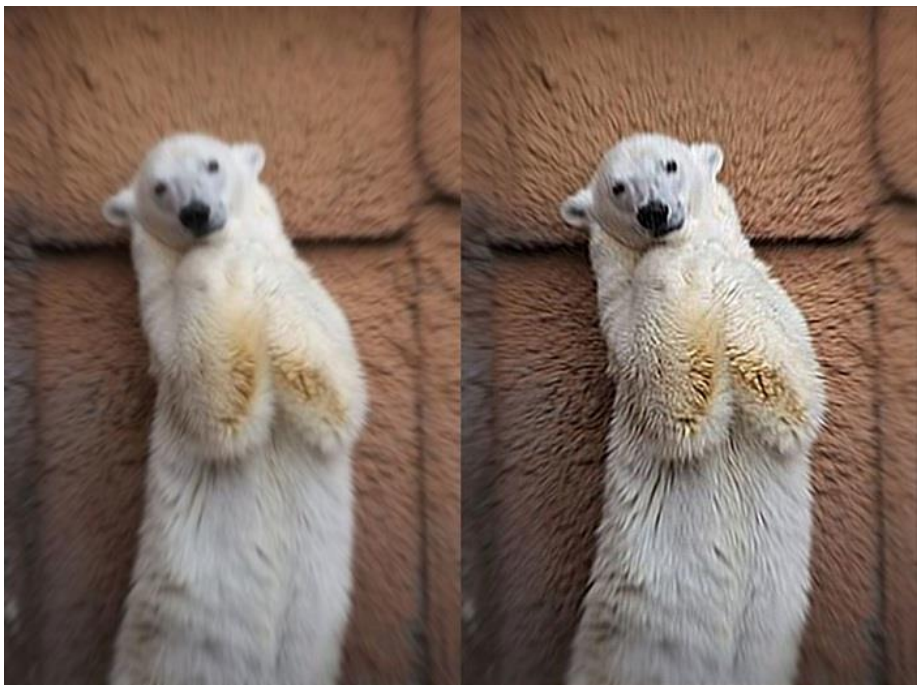


Рисунок 47 – Результат обработки изображений алгоритмом

В результате проведенных тестирований были получены результаты обработки изображений программным продуктом, основанном на разработанном алгоритме. Результаты обработки показывают, что изображение было восстановлено и более детализировано по сравнению с оригиналом.

**Выводы по третьей части:**

В результате описываемой в данной части работы было проведено тестирование разработанного программного обеспечения, которое получает смазанные изображения от внешней системы и на их основе строит новое изображение при помощи свёрточной нейронной сети VGG19.

Была проанализирована зависимость размера изображения от времени его восстановления: чем больше разрешение изображения, тем больше времени требуется на его восстановление, показаны результаты обработки изображений, а также проведено сравнение разработанного алгоритма с существующими.

## Заключение

Выпускная квалификационная работа посвящена разработке алгоритма восстановления расфокусированных и смазанных изображений, а также разработке программного продукта, использующего данный алгоритм. Цель работы определила ее основное направление – получение максимально возможной производительности одновременно выполняемых функций обработки изображений посредством разработки алгоритма восстановления их качества с использованием нейронной сети, а также разработки программного продукта, использующего данный алгоритм.

Для этого были разобраны существующие методы восстановления качества изображений. В ходе исследования были рассмотрены математические аппараты существующих алгоритмов, а также выявлены как сильные стороны этих алгоритмов, так и слабые.

В ходе анализа рассмотренных методов восстановления расфокусированных и смазанных изображений были сформированы задачи для разрабатываемого алгоритма и программного продукта на его основе, а также сформированы критерии.

В процессе разработки были реализованы:

– Алгоритм восстановления расфокусированных и смазанных изображений;

– Блок-схема для разработанного алгоритма;

– Программный продукт, использующий в качестве основы разработанный алгоритм.

Так же в качестве дополнительного компонента был разработан модуль, помогающий определить коэффициент смаза для входных изображений.

Данная разработка позволяет повысить качество исходных изображений при помощи применения к ним генеративно-состязательной сети.

В рамках выпускной квалификационной работы были выполнены все поставленные цели и задачи.

Разработанный алгоритм, а также программная реализация на основе этого алгоритма были протестированы на реальных примерах.

Были отражены результаты обработки изображений разработанным алгоритмом и программной оболочки к нему на различных итерациях.

Для тестирования программного продукта на основе разработанного алгоритма был взят набор смазанных изображений определенного размера. Все изображения были восстановлены через программные реализации существующих алгоритмов, а также через программную реализацию разработанного алгоритма. Разработанный программный продукт на основе составленного алгоритма показал лучшие результаты по сравнению с существующими инструментами, выиграв у них несколько секунд.

Полученные результаты разработки алгоритма и программного продукта на его основе можно считать успешными, поскольку при тестировании не было выявлено никаких ошибок, что могло бы привести процесс работы с приложением и алгоритмом к непригодности.

## Список используемых источников

1. Лутц, М. Python. Карманный справочник / М. Лутц ; [перевод с английского И. В. Берштейна]. – Москва : Диалектика, 2019. – 320 с. : ил. ; Библиогр.: с. 9-10. – 1500 экз. – ISBN 978-5-8459-1965-6. – Текст : непосредственный.

2. Шакла, Н. Машинное обучение и TensorFlow / Н. Шакла ; [перевод с английского ООО Издательство «Питер»]. – Санкт-Петербург : Питер, 2019. – 336 с. : ил. ; Библиогр.: с. 53-56. – 5000 экз. – ISBN 978-5-4461-0826-8. – Текст : непосредственный.

3. Хабр : Сверточная нейронная сеть и её интеграция в iOS (часть 1) : [сайт]. – Москва, 2020 – . – URL: https://habr.com/ru/post/500998 (дата обращения: 08.05.2020). – Текст : электронный.

4. Научный корреспондент : Субполосная фильтрация сигналов : [сайт]. – Москва, 2020 – . – URL: https://nauchkor.ru/uploads/documents/5a40323c7966e104c6a3e8bd.pdf (дата обращения: 16.03.2020). – Текст : электронный.

5. Научное издание МГТУ им. Н.Э. Баумана : Цель восстановления изображений : [сайт]. – Москва, 2019 – . – URL: http://engineering-science.ru/file/505163.html?__s=1 (дата обращения: 16.12.2019). – Текст : электронный.

6. КиберЛенинка : Метод ускоренного восстановления изображений, смазанных при движении : [сайт]. – Москва, 2019 – . – URL: https://cyberleninka.ru/article/n/metod-uskorennogo-vosstanovleniya-izobrazheniy-smazannyh-pri-dvizhenii (дата обращения: 26.12.2019). – Текст : электронный.

7. QWERTYU : Глубокое изучение – Deep Learning : [сайт]. – Гравлин, 2020 – . – URL: https://ru.qwe.wiki/wiki/Deep_learning (дата обращения: 25.02.2020). – Текст : электронный.

8. КиберЛенинка : Слепая коррекция изображений в векторном канале с неизвестной импульсной характеристикой : [сайт]. – Москва, 2019 – . – URL: https://cyberleninka.ru/article/n/slepaya-korrektsiya-izobrazheniy-v-vektornom-kanale-s-neizvestnoy-impulsnoy-harakteristikoy (дата обращения: 24.12.2019). – Текст : электронный.

9. Neurohive : Генеративно-состязательная сеть (GAN). Руководство для новичков : [сайт]. – Санкт-Петербург, 2020 – . – URL: https://neurohive.io/ru/osnovy-data-science/gan-rukovodstvo-dlja-novichkov (дата обращения: 11.02.2020). – Текст: электронный.

10. Хабр : Сверточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество : [сайт]. – Москва, 2020 – . – URL: https://habr.com/ru/post/348000/ (дата обращения: 13.01.2020). – Текст: электронный.

11. ProgramForYou : Свёрточная нейронная сеть с нуля. Часть 0. Введение : [сайт]. – Рязань, 2020 – . – URL: https://programforyou.ru/poleznoe/convolutional-network-from-scratch-part-zero-introduction (дата обращения: 30.01.2020). – Текст: электронный.

12. Neurohive : Как работает сверточная нейронная сеть: архитектура, примеры, особенности : [сайт]. – Санкт-Петербург, 2020 – . – URL: https://neurohive.io/ru/osnovy-data-science/glubokaya-svertochnaja-nejronnaja-set (дата обращения: 03.02.2020). – Текст: электронный.

13. Robocraft : GAN – генеративные состязательные сети : [сайт]. – Хельсинки, 2020 – . – URL: http://robocraft.ru/blog/machinelearning/3693.html (дата обращения: 09.03.2020). – Текст: электронный.

14. Хабр : Автоэнкодеры в Keras, Часть 5: GAN (Generative Adversarial Networks) и Tensorflow : [сайт]. – Москва, 2020 – . – URL: https://habr.com/ru/post/332000/ (дата обращения: 16.03.2020). – Текст: электронный.

15. КиберЛенинка : Блочные алгоритмы обработки изображений на основе фильтра Калмана в задаче построения сверхразрешения : [сайт]. – Москва, 2020 – . – URL: https://cyberleninka.ru/article/n/blochnye-algoritmy-obrabotki-izobrazheniy-na-osnove-filtra-kalmana-v-zadache-postroeniya-sverhrazresheniya (дата обращения: 18.03.2020). – Текст: электронный.

16. MathWorks : VGG-19 convolutional neural network : [сайт]. – Монреаль, 2020 – . – URL: https://www.mathworks.com/help/deeplearning/ref/vgg19.html (дата обращения: 19.03.2020). – Текст: электронный.

17. Howse J. Learning OpenCV 4 Computer Vision with Python 3: Get to grips, techniques, and algorithms for computer vision and machine learning 3rd Edition / J. Howse, J. Minichino. – USA : Packt Publishing, 2020. – 372 p.; p. 7 – 9. – ISBN: 978-1782163923.

18. Loy J. Neural Network Projects with Python: The ultimate guide to using Python to explore the true power of neural networks through six projects 1st Edition / J. Loy. – USA : Packt Publishing, 2019. – 308 p.; p. 30 – 40. – ISBN: 978-1789138900.

19. Geron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 2nd Edition / A. Geron. – USA : O'Reilly Media, 2019. – 849 p.; p. 120 – 140. ISBN: 978-1492032649.

20. Campisi P. Blind Image Deconvolution: Theory and Applications / P. Campisi, K. Egiazarian. – USA : CRC Press, 2007. – 472 p.; p. 206 – 210. – ISBN: 978-0849373671.

21. Parker J. Image Reconstruction in Radiology / J. Parker. – USA : CRC Press, 2018. – 531 p.; p. 153 – 155. – ISBN: 978-1315894263.

22. Thakar V. Deep Learning with Python and OpenCV: A begginer's guide to perform smart image processing techniques using TensorFlow and Keras / V. Thakar. – USA : Packt Publishing, 2019. – 363 p.; p. 53-54. – ISBN: 978-1788627320.

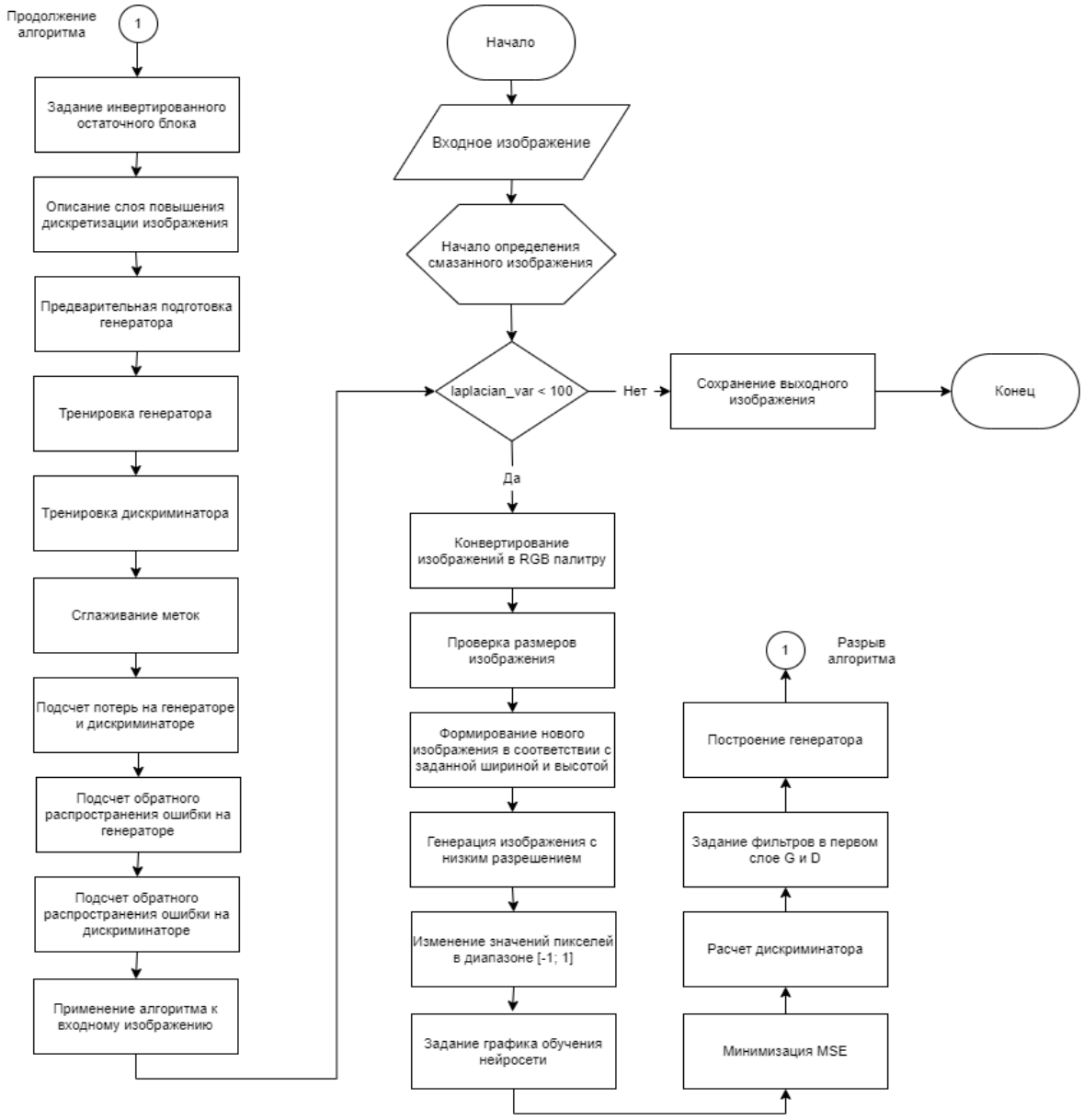
23. Millan Escriva D., Joshi P. Building Computer Vision Projects with OpenCV 4 and C++: Implement complex computer vision algorithms and explore deep learning and face detection / P. Joshi. – USA : Packt Publishing, 2019. – 538 p.; p. 124-130. – ISBN: 978-1838644673.

24. Langr J., Bok V. GANs in Action: Deep Learning with Generative Adversarial Networks / V. Bok. – USA : Manning Publications, 2019. – 276 p.; p. 234-240. – ISBN: 978-1617295560.

25. Aggarwal C. Neural Networks and Deep Learning: A Textbook / C. Aggarwal. – USA : Springer, 2018. – 497 p.; p. 340-370. – ISBN: 978-3319944623.

# Приложение А

## Блок-схема составленного алгоритма восстановления расфокусированных и смазанных изображений



## Приложение Б

## Режим доступа к программному обеспечению, разработанному на основе составленного алгоритма восстановления расфокусированных и смазанных изображений

Исходный код программы доступен для скачивания в электронном виде.

Режим доступа:

https://drive.google.com/open?id=17Sm1fPG3k68rBZwQz0pH78K9aNlh27nd