Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО–ПОЯСНИТЕЛЬНАЯ ЗАПИСКА *К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ НА ТЕМУ:*

«Метод удаления импульсных шумов из цветных изображений с помощью сверточных нейронных сетей»

Студент группы **ИУ7–85Б** _____ (Подпись, дата) **А.Н. Прянишников** (И.О. Фамилия)

Руководитель _____ (Подпись, дата) **В.П. Степанов** (И.О. Фамилия)

Нормоконтроллер _____ (Подпись, дата) _____ (И.О. Фамилия)

Москва — 2023г.

РЕФЕРАТ

Расчетно–пояснительная записка 63 с., 24 рис., 1 табл., 29 ист., 3 прил.

В работе представлена разработка метода удаления импульсных шумов из цветных изображений с помощью сверточных нейронных сетей.

Ключевые слова: шум, фильтрация, алгоритм, сверточная нейронная сеть, изображение.

Рассмотрена задача удаления импульсных шумов из цветных изображений. Рассмотрены существующие методы и технологии удаления импульсных шумов. Проведена формализация постановки задачи в виде IDEF0-диаграммы. Разработан метод, позволяющий удалить импульсные шумы из цветных изображений, с помощью сверточных нейронных сетей. Представлена реализация разработанного метода, приведены результаты исследования эффективности разработанного метода по сравнению с аналогами.

Содержание

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

1. NN (от англ. Neural network) — нейронная сеть.
2. CNN (от англ. Convolutional neural network) — сверточная нейронная сеть.
3. PSNR (от англ. Peak signal-to-noise ratio) — пиковое отношение сигнала к шуму.
4. SSIM (от англ. Structure similarity) — индекс структурного сходства.

ВВЕДЕНИЕ

Цифровой шум в изображениях появляется вследствие свойств фотона света или вмешательства человека [1]. Он оказывает огромное влияние на восприятие человеком качества картинки, а также ухудшает результаты работы алгоритмов, анализирующих изображения [2]. Нейросети не справляются с определением объектов на фотографии, если внести на неё шум, а человеческий глаз не всегда в состоянии распознать, что на картинке присутствуют помехи [3].

Актуальность работы состоит в том, что аппаратные способы борьбы с шумами в изображениях в текущий момент не реализованы, и компании стараются бороться с дефектами на фотографиях с помощью алгоритмических методов [2]. Технические средства Kodak обрабатывают изображения с помощью таких методов, как медианный фильтр [4].

Цель работы — разработать алгоритм для борьбы с импульсными шумами в цветных изображениях с помощью сверточных нейронных сетей.

Для достижения поставленной цели требуется решить **задачи**:

— описать термин шума, причины его появления и классифицировать его типы;

— описать существующие методы удаления шума в изображениях, провести сравнительный анализ;

— формализовать постановку задачи в виде IDEF0-диаграммы;

— разработать метод удаления импульсных шумов из цветных изображений;

— разработать программный комплекс, реализующий метод;

— исследовать разработанный метод на применимость.

1 Аналитический раздел

В этом разделе будет произведен анализ предметной области: описаны основные понятия в области удаления шумов из цветных изображений. Будут рассмотрены существующие методы удаления импульсных шумов из цветных изображений, приведены результаты сравнительного анализа. Будет определено понятия нейронной сети, разобран принцип работы сверточной нейронной сети. Будет сформирована цель работы, постановка задачи формализована в виде IDEF0-диаграммы.

1.1 Анализ предметной области

Шум – дефект изображения, в основе которого лежит эффект появления на фотографии пикселей случайного цвета на изображении [1].

Причины возникновения такого эффекта делятся на два типа: естественные и искусственные. Основных источником естественных помех на изображениях является фотосенсор [1]. Существует несколько физических объяснений появления шума на изображении [5]:

1. При дефектах потенциального барьера происходит утечка заряда. В этом случае шум на изображениях проявляется в виде темных точек на светлом фоне.

2. При подаче потенциала на электрод может возникнуть темновой ток, который отображается на картинке в виде светлых точек на темном фоне. Основная причина возникновения темнового тока — это примеси в кремниевой пластине или повреждение кристаллической решетки кремния.

Также шум на изображениях может быть вызван умышленным вмешательством человека или состязательной атакой [6]. Состязательная атака – это манипуляция обучающими данными, архитектурой модели или манипулирование тестовыми данными таким образом, что это приведет к неправильному выходу из модели машинного обучения [6].

Существует несколько основных типов естественных шумов, возникающих на фотографиях [7]. От точного определения характеристики шума зависит то,

какой метод требуется выбрать для автоматического определения дефектных пикселей на изображении и последующего его устранения. Для выявления причины требуется понять, на каком устройстве была сделана фотография, и через какие этапы обработки она прошла. При этом для реальных цветных фотографий результирующий шум является комбинацией всех возможных типов дефектов, соответственно, нельзя точно сказать о происхождении каждого дефекта на рассматриваемой фотографии [8].

1.1.1 Гауссов шум

Так как квантовым процессам свойственна случайность, то такие процессы можно отнести к Гауссовым, следовательно, они обладают следующим свойством: распределение суммы независимых случайных величин сходится к нормальному, вне зависимости от характера распределения слагаемых [9].

Пусть I – интенсивность изначального пикселя, а ν – интенсивность шума, распределенная по нормальному распределению. Тогда интенсивность загрязненного пикселя можно представить по формуле 1.1:

$$I_f = I + \nu, \nu \sim N(0, \sigma^2). \tag{1.1}$$

Вид гауссова шума представлен на рисунке 1.1:

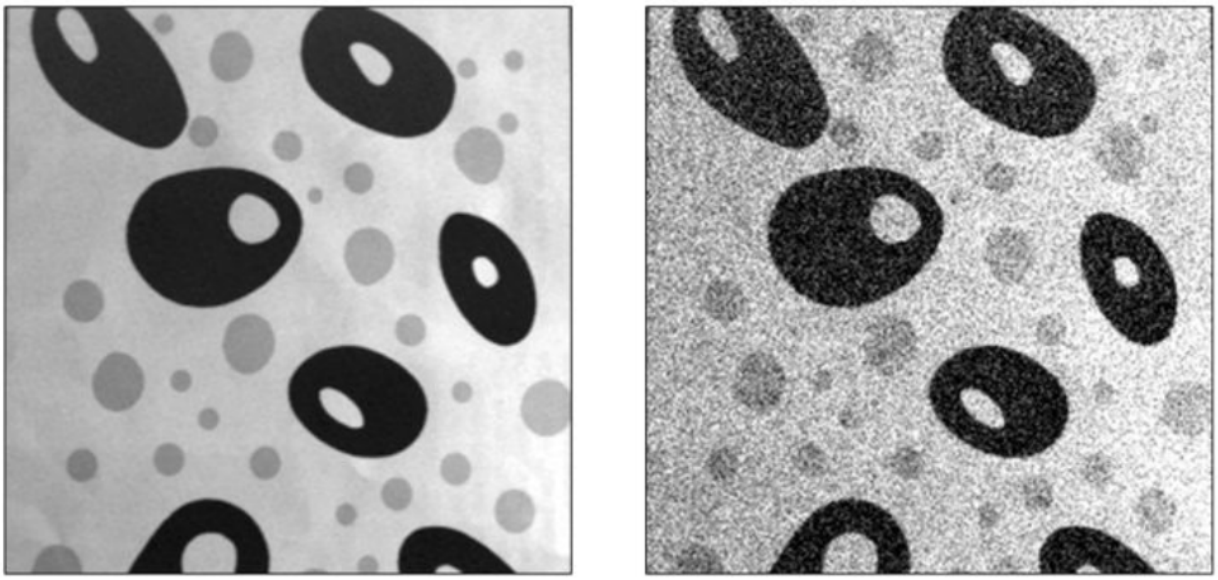

Рисунок 1.1 – Вид гауссова шума

Причиной возникновения таких шумов как правило является некорректная работа фотосенсора, что часто происходят в условиях ночного освещения или высокой температуры. Именно этот вид шумов на практике встречается чаще всего [7].

1.1.2 Импульсный шум

Импульсный шум проявляется в том, что на изображениях в случайных местах появляются черные и белые пиксели [10]. Основной причиной их возникновения является темновой ток и утечка заряда в фотосенсоре, а также наличие пикселей с дефектами [1].

Когда пиксель имеет белый цвет на темном фоне, то говорят, что здесь присутствует шум соли. Шум перца – обратная ситуация: на светлом изображении заметны темные точки [10].

Описать появление шума можно по формуле 1.2:

$$P(S_{i,j} = 1) = p, \tag{1.2}$$

где S – исходное изображение, i, j – координаты пикселя, p – вероятность появления шума.

Вероятность появления шума зависит от характеристик фотоаппарата, а также от внешних условий, поэтому точно рассчитать значение невозможно [10].

Вид шума соли и перца представлен на рисунке 1.2:

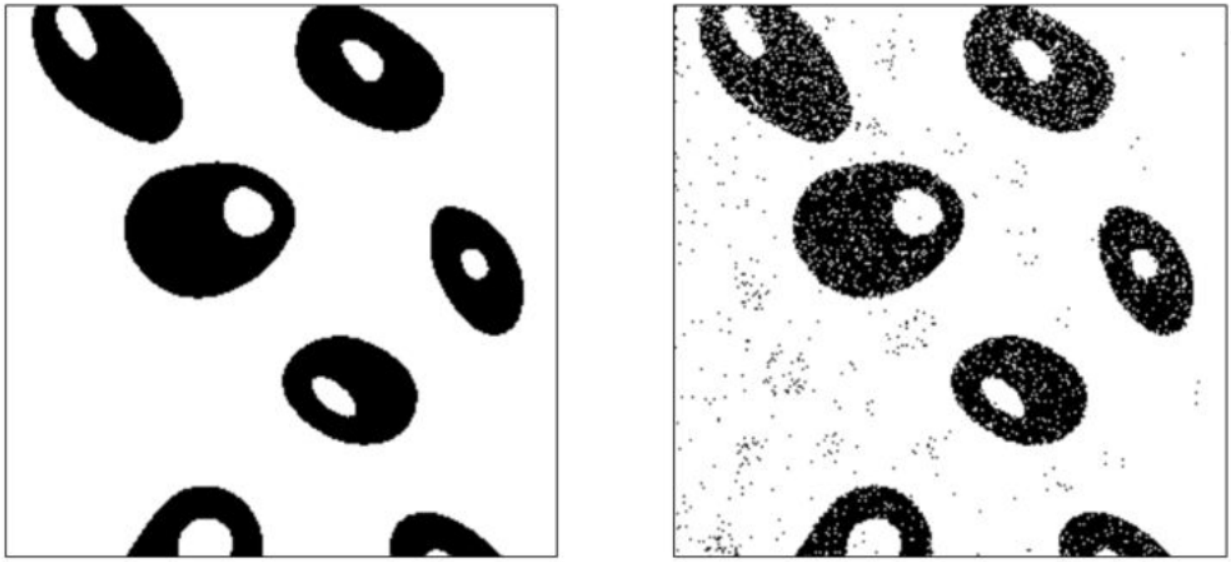

Рисунок 1.2 – Шум соли и перца

Основными методами борьбы с таким видом шумов на изображениях являются медианный фильтр и сверточные нейронные сети.

1.2 Нейронные сети

Нейронная сеть – математическая модель, основанная на принципах функционирования биологических нейронных сетей – например, нервных клеток человека [11]. Требуется ввести следующие основные определения в математическом контексте нейронной сети:

1. **Нейрон** – вычислительная единица, получающая на вход информацию, обрабатывает ее, и возвращает результат на выходе [11].

2. **Синапс** – связь между двумя нейронами, построенная по следующему принципу: выход одного нейрона является входом для другого нейрона. У каждого синапса есть один параметр – вес, который влияет на результат вычислений в нейроне.

3. **Слой** – множество нейронов, которые получают на вход информацию от всех нейронов из предыдущего слоя [11].

Все нейроны можно разделить на три больших типа:

1. **Входной** – он получает на вход исходный вектор, которую требуется обработать в нейронной сети, и без предварительной обработки передаёт на вход следующему слою [12]. Графически принято обозначать входной нейрон синим цветом.

2. **Выходной** – значение, полученное на выходе нейрона, считается итоговым результатом всех вычислений нейронной сети [12]. На схемах для обозначения выходного нейрона используется зеленый цвет.

3. **Скрытый** – нейрон, который не является ни входным, ни выходным, соответственно, ему на вход поступает информация от других нейронов, и его выход также переходит другим нейронам. Скрытые нейроны являются основным вычислительным центром всей сети. Выделяются красным цветом.

Математическая модель одного нейрона представляется формулой 1.3:

$$Y = f(\sum_{i=1}^{N} w_i * x_i + w_0 * x_0), \tag{1.3}$$

где Y – выход нейрона, x_i – выход i-го нейрона предыдущего слоя, w_i – вес синапса, связывающего x_i нейрон и текущий, x_0 и w_0 – дополнительный вход и его вес, которые используются для инициализации нейронной сети, f – передаточная функция.

Пусть x – значение из входного вектора, y – результат вычислений нейрона. На эти значения накладываются ограничения, указанные в формуле 1.4:

$$x \in [0, 1]; y \in [0, 1]. \tag{1.4}$$

Передаточная функция является основным параметром нейронной сети. Как правило, она обладает следующими ключевыми свойствами:

1. Передаточная функция f монотонно возрастает.

2. Область значений функции f лежит в интервале [0, 1].

Передаточная функция позволяет добавлять нелинейность нейронной сети. Существует несколько основных типов передаточных функций, используемых в нейронных сетях:

1. **Линейная передаточная функция**

   В этом случае информация на выходе линейно зависит от результата суммирования входных сигналов. В основном она используется для входного слоя нейронной сети, причём коэффициент равен 1.

2. **Шаговая передаточная функция** Алгоритм работы этой функции представлен на формуле 1.5:

$$f(x) = \begin{cases} 1, & \text{если } x \geq 1, \\ 0, & \text{если } x \leq 0, \\ x, & \text{иначе.} \end{cases} \tag{1.5}$$

   Самая распространенная вариация функции – исправленная линейная передаточная функция или **ReLu** [13]. Математически она описывается формулой 1.6:

$$f(x) = \begin{cases} x, & \text{если } x \geq 0, \\ 0, & \text{иначе.} \end{cases} \tag{1.6}$$

   Она получила широкое распространение в сверточных нейронных сетях из-за простоты использования [14].

3. **Сигмоидальная передаточная функция**

   Эта передаточная функция позволяет усиливать слабые сигналы значительно заметнее, чем сильные, поэтому область сильных сигналов представлена пологими участками [13].

   На формуле 1.7 представлен вид передаточной функции [14]:

$$f(x) = \frac{1}{1 + \exp(-tx)}, \tag{1.7}$$

   где t – параметр функции, регулирующий крутизну функции.

4. **Гиперболическая передаточная функция** Эта передаточная функция использует операцию гиперболического тангенса для расчёта итогового значения [13]. По свойствам она похожа на сигмоиду, но слабые сигналы усиливаются еще сильнее.

   Формула гиперболической передаточной функции 1.8:

$$f(x) = \frac{2}{1 + \exp(-2x)} + 1. \tag{1.8}$$

Перед тем, как нейросеть начнет вычислять результат, требуется провести ее обучение.

Для этого используется метод обратного распространения ошибки.

Предположим, для заданного сигнала заранее известен вектор результатов $Y0$, который ожидается на выходе нейронной сети. Нейронная сеть после подачи на вход сигнала выдала вектор $Y1$.

Функция ошибки для отдельного синапса, полученная методом наименьших квадратов, представлена на формуле 1.9:

$$Ew = \frac{1}{2} * \sum Ni = 0(Y0_i - Y1_i)^2, \tag{1.9}$$

Для проведения обучения нейронной сети используется стохастический градиентный спуск. Требуется изменять веса после вычислений результат для каждого входящего нейрона.

Добавленное значение к весу представлено на формуле 1.10:

$$\Delta w_{ij} = -\eta * \frac{\partial E}{\partial w_{ij}}, \tag{1.10}$$

где w_{ij} – вес синапса между нейронами i и j, η – коэффициент, влияющий на скорость обучения.

Суть обучения нейронной сети состоит в минимизации функции ошибки для отдельного входящего сигнала. С помощью корректировки коэффициентов синапсов сеть пытается достигнуть точки сходимости или локального минимума. Именно для этого используется градентный спуск.

1.2.1 Сверточная нейронная сеть

Сверточная нейронная сеть – один из подвидов нейронных сетей, получивший широкое распространение в области обработки изображений [15].

В контексте СНС нужно ввести следующие определения:

1. **Ядро свертки** – квадратная матрица весов, используемая для операции свертки [16]. Ее размер, как правило, определяется программистом при инициализации сети.

2. **Свертка** – математическая операция, в результате которой уменьшается размерность вектора исходного сигнала [17]. Ее можно описать следующей формулой 1.11:

$$x_i^l = f(\sum ix_i^{l-1} * k_j^l + b_j^l), \tag{1.11}$$

   где x_i^l – выходной сигнал слоя l, f – передаточная функция, x_i^{l-1} – входной сигнал для входа l, k_j^l – ядро свертки слоя l, b_j^l – коэффициент сдвига для слоя l.

3. **Карта признаков** – результат, полученный после операции свертки, который отражает наличие какого-либо признака во входном сигнале [17].

Процесс обучения сверточной нейронной сети осуществляется методом обратного распространения ошибки [17]. В сверточной нейронной сети несколько ядер свертки, которые отвечают за отдельные признаки, но такое разбиение возникает в результате обучения – программист не предусматривает это при проектировании архитектуры сети. Таким образом, каждый набор вес весов формирует собственную карту признаков, в результате возникает явление многоканальности в нейронной сети: на одном слое появляются независимые карты признаков [18]. Во время выполнения операции свертки ядро матрицы передвигают не на полный шаг, а на маленькое расстояние – несколько пикселей, в результате искомый признак отражается в нескольких весах выходного вектора.

Так как в результате свертки итоговая размерность векторов уменьшается, то величина сигнала стремится к размерам ядра свертки. Последовательность таких действий называется *субдискретизацией* [18].

В качестве передаточной функции чаще всего используют ReLu [18].

Архитектура сверточной сети состоит из нескольких компонентов:

1. **Начальный слой** – принимает на вход исходное изображение [15].
2. **Сверточные слои** – состоят из ядр свертки, каждый слой выполняют субдискретизацию полученных карт признаков [15].
3. После прохождения субдискретизации карты признаков вырождаются до векторов или скаляров. В дальнейшем они попадают на нейроны из полносвязной нейронной сети, которая рассчитывает итоговый результат.

При прохождении через сверточные слои размерность каждого слоя уменьшается, в то время как количество каналов – растет. Это позволяет использовать полносвязную нейронную сеть для определение результата [17].

Схематично архитектура сверточной нейронной сети представлена на рисунке 1.3:

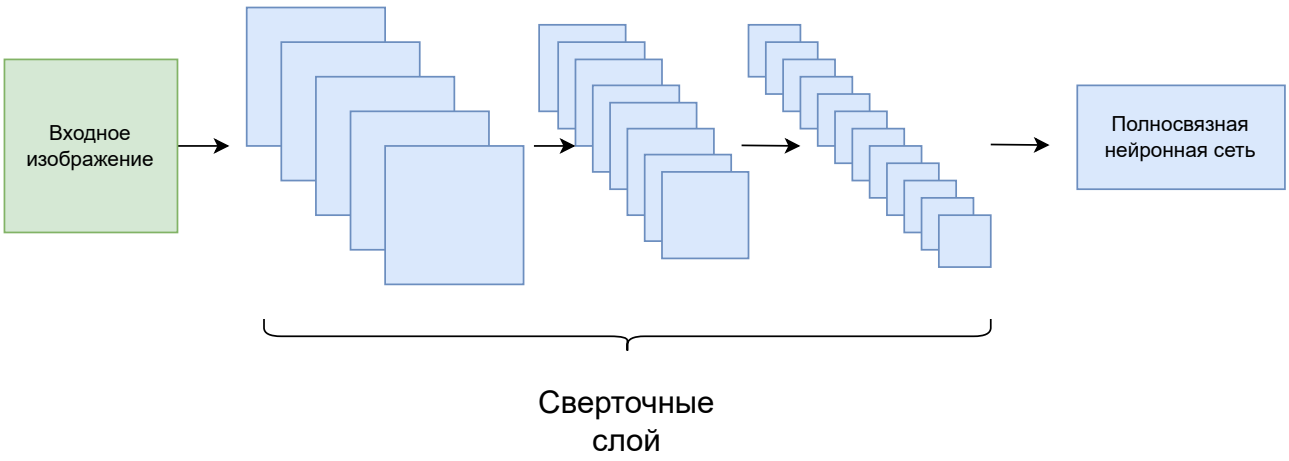

Рисунок 1.3 – Архитектура сверточной нейронной сети

Во время обучения принято на некоторых итерациях убирать из сети одиночные нейроны, что позволяет точнее настроить другие нейроны для более точного обучения [12].

Признакам, сформированные в результате работы нейронной сети, невозможно дать объяснение, поэтому в случае, если нейронная сеть перестает выполнять поставленную задачу, разработчики изменяют ее конфигурацию или выбирают другой этап применения в общем алгоритме [16].

На текущий момент не существует архитуктур нейросетей, которые бы в точности позволяли бороться с шумами в изображениях, поэтому подходящие настройки сети подбираются экспериментально [11].

Нейронные сети активно применяются в области удаления шумов из изображений, так как они позволяют распознать и удалить шумы, с которыми не справляются стандартные методы. Для трехмерных изображений ядра матрицы составляются для каждого из составляющих цветов пикселя, итоговые результаты по цветам суммируется [12]. Это позволяет менять вариативность нейронной сети, например, настраивать на более приоритетный цвет.

1.3 Обзор существующих методов

Шумы искажают исходную картинку и портят ее качество так, что это способен распознать человеческий глаз. Однако могут возникнуть трудности в обнаружении помех, поскольку они трудно различимы при совпадении цвета фона и цвета пикселя, например, светлые точки будут плохо заметны на ярком фоне.

Было разработано несколько алгоритмов, которые производят бинарную классификацию пикселей и определяют, какие из них можно идентифицировать как шумы и затем их устранить.

В качестве классификации алгоритмы можно разделить на два типа:

1. **Изотропная фильтрация** – такие методы устраняют помехи, но не учитывают детали пикселя и увеличивают размытость.

2. **Анизотропная фильтрация** – алгоритмы устраняют эффекты сглаживания, уменьшают размытость и сохраняют детали пикселя, устраняя при этом непосредственно шум из изображения.

1.4 Общий алгоритм работы фильтров

Алгоритмы, анализирующие наличие шумов в изображениях, имеют дело с различными характеристиками одного пикселя. Например, цвет пикселя можно разбить на три составляющие – синюю, красную и зеленую [19].

В таком случае метод работает с каждой из составляющих пикселя, вычисляя новое значение для каждой характеристики. Результат работы в этом случае является объединением подсчетов по всем характеристикам.

Общая схема работы алгоритмов представлена на рисунке 1.4:

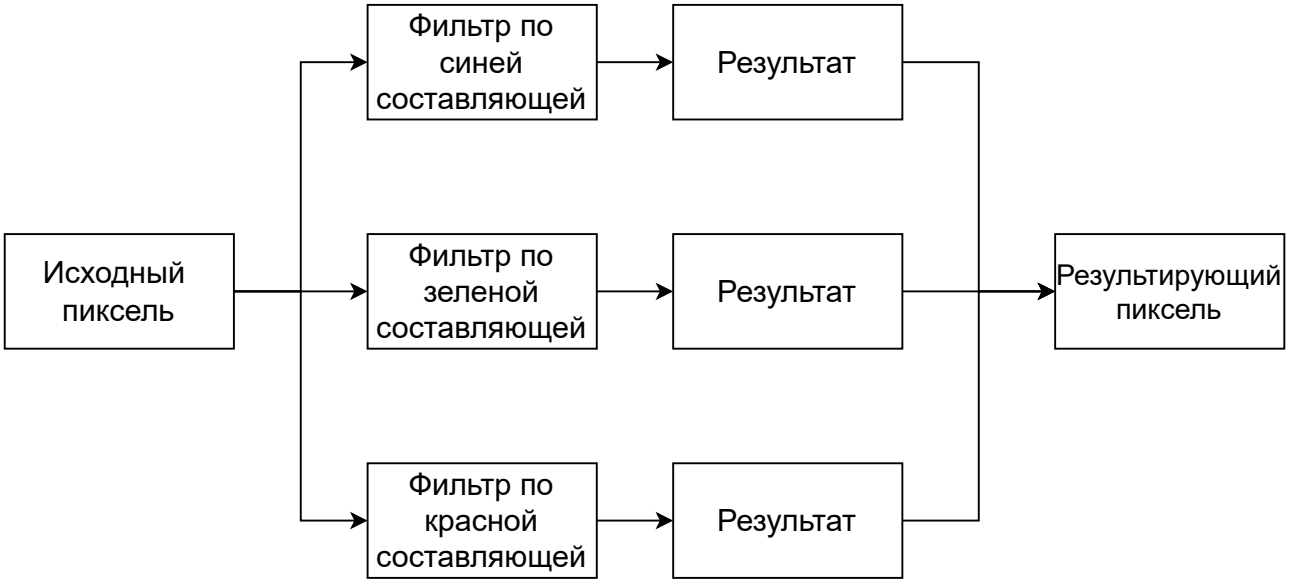

Рисунок 1.4 – Общая схема работы всех алгоритмов

Каждый из фильтров, перечисленный ниже, работает с каждым из параметров пикселя одинаково, поэтому для корректной работы алгоритмов требуется вычислить значение каждого свойства для результирующего пикселя [7].

1.4.1 Медианный фильтр

Под медианным фильтром понимается семейство однотипных алгоритмов, относящихся к классу нелинейных фильтров [20].

Метод работает в цикле с каждым пикселем изображения [7]. В окрестности каждого пикселя находится восемь соседних, каждый обладает собственными свойствами.

На рисунке 1.5 изображена сетка, с которой работает алгоритм:

Рисунок 1.5 – Рассматриваемая сетка пикселей при работе алгоритма

Пусть $C_{i,j}$ – один из параметров рассматриваемого пикселя, а Ω – все пиксели сетки. Алгоритм подсчитывает медиану от такого же параметра соседних клеток и заменяет параметр пикселя на значение этой медианы [20].

Схема работы алгоритма изображена на рисунке 1.6:

Рисунок 1.6 – Схема работы алгоритма медианного фильтра

К недостаткам данного метода можно отнести то, что алгоритм может убрать значительные детали из изображения, посчитав их за шум [21].

1.4.2 Гауссовский фильтр

Работа алгоритма гауссовского фильтра также зависит от значений цветовых свойств пикселей в сетке, рассмотренной на рисунке 1.5.

В этом случае для каждого соседнего рассчитывается вес, с которым он влияет на новое значение рассматриваемого пикселя [7]. Пусть d – расстояние до центрального пикселя сетки, σ – стандартное отклонение, подсчитанное для всех значений определенного параметра текущей сетки. Тогда вес w пикселя

рассчитывается по формуле 1.12:

$$w_{ij} = e^{\left(\frac{-d^2}{2\sigma^2}\right)}. \tag{1.12}$$

Подсчитав вес для каждого пикселя в сетки, можно рассчитать новое значение свойства рассматриваемого пикселя по формуле 1.13:

$$p_i = \frac{1}{\sum_{j \in \Omega} w_{ij}} * \sum_{j \in \Omega} w_{ij} * p_j. \tag{1.13}$$

Схема алгоритма гауссовского фильтра представлена на рисунке 1.7:

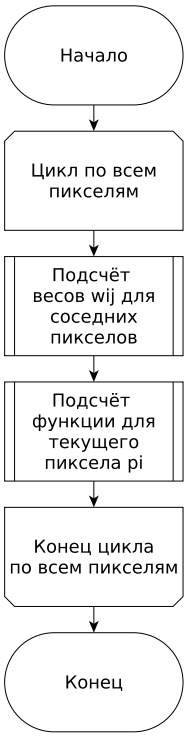

Рисунок 1.7 – Схема работы алгоритма гауссовского фильтра

1.4.3 Билатеральный фильтр

Алгоритм билатеральной фильтрации является улучшением метода Гауссовского фильтра [22].

Для каждого пикселя сетки соседних пикселей используется сразу два веса: один аналогичный параметру из исходного алгоритма, а второй отвечает за анизотропную составляющую.

Расчет веса w_s, отвечающего за изотропную составляющую, происходит по формуле 1.12. Коэффициент, регулирующий анизотропные свойства фильтрации, рассчитывается по формуле 1.14:

$$w_r = \exp\left(\frac{-|p_i - p_j|}{2\sigma^2}\right). \tag{1.14}$$

В таком случае результат работы некоторого пикселя можно посчитать по формуле 1.15:

$$p_i = \frac{1}{\sum_{j\in\Omega} w_s w_r} * \sum_{j\in\Omega} w_s w_r p_j. \tag{1.15}$$

Схема работы алгоритма представлена на рисунке 1.8:

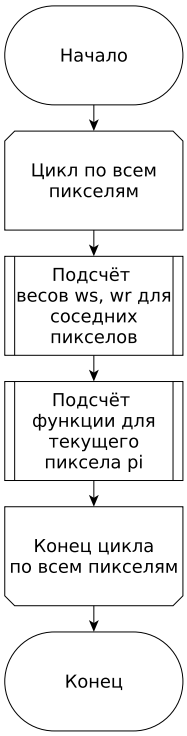

Рисунок 1.8 – Схема работы алгоритма билатеральной фильтрации

1.4.4 Алгоритм Цзяньвэй

Этот алгоритм был описан в 2014 году индонезийским ученым Ван Цзяньвэй [23]. Алгоритм позволяет эффективно убирать шумы соли и перца даже в случае сильной загрязненности изображения.

Процедура заключается в обходе всех пикселей фотографии в заданном порядке и определении того, соответствуют ли значения пикселей функции плот-

ности вероятности импульсного шума или нет. Если пиксель на первом классифицируется как шум, то подсчитывается количество импульсного шума в маске определенной формы. Результатом операции маски является замена значения пикселя. В противном случае это не рассматривается как шум, значение пикселя остается неизменным.

Схема алгоритма Цзяньвэй представлена на рисунке 1.9:

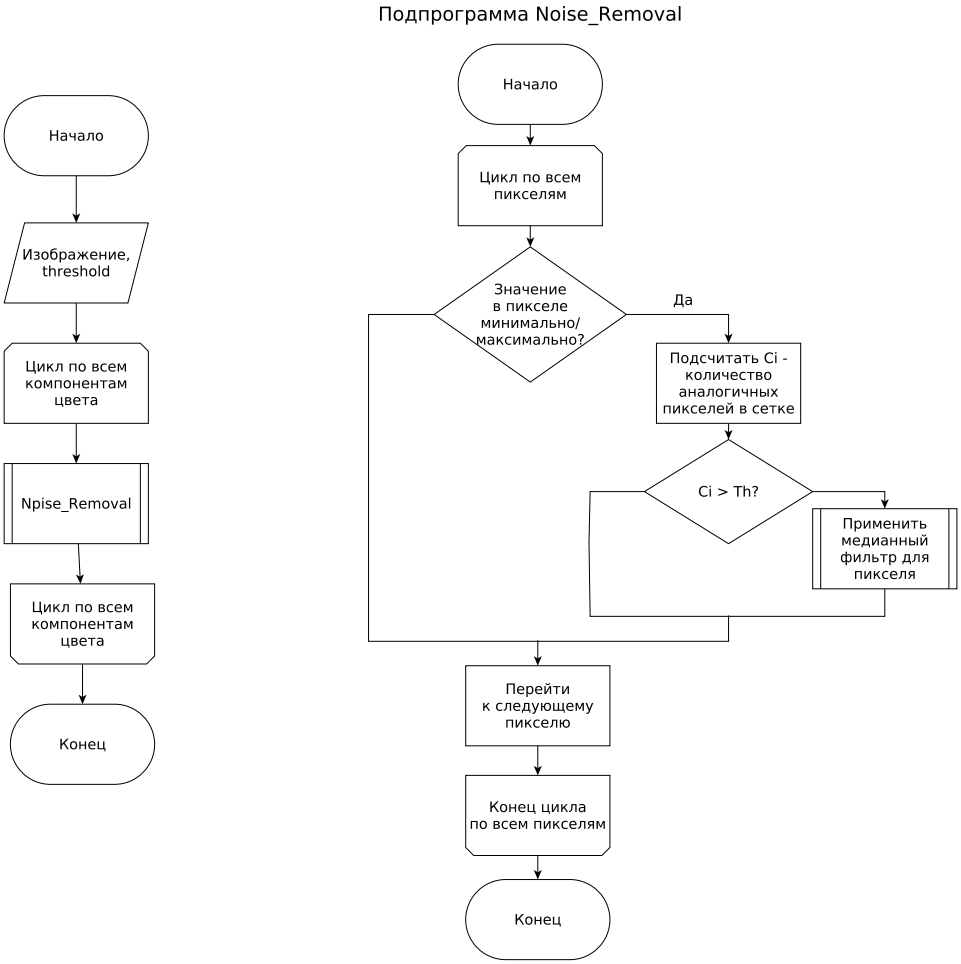

Рисунок 1.9 – Схема работы алгоритма Цзяньвэй

1.4.5 DnCNN

Алгоритмы, основанные на сверточных нейронных сетях, получили широкое распространение в области удаления шумов из изображений. Они применяются в ситуациях, требующих восстановления исходного изображения по имеющемуся загрязненному фото, например, при анализе снимков из телескопов или фотоаппаратов [11]. DnCNN относится к этому классу методов.

Перед запуском алгоритма требуется задать суммарную глубину слоев D, а также количество цветовых составляющих c – для цветных изображений $c = 3$, для серых $c = 1$ [24]. Всего в конфигурации нейронной сети используется три типа слоев [25]:

1. Первый слой состоит из 64 фильтров размером $3*3*c$, в качестве функции активации используется ReLU.

2. Следующие $2 * (D - 1)$ слоев состоят из фильтров размером $3 * 3 * 64$, между каждым слоем применяется пакетная нормализация.

3. Последний слой состоит из c фильтров размером $3 * 3 * 64$ каждый.

Во всех слоях, кроме последнего, используется ReLU в качестве функции активации нейрона [24].

Пусть x – входное значение нейрона.

Тогда выход считается по формуле 1.16:

$$f(x) = \max(0, x). \tag{1.16}$$

Конфигурация нейронной сети представлена на рисунке 1.10:

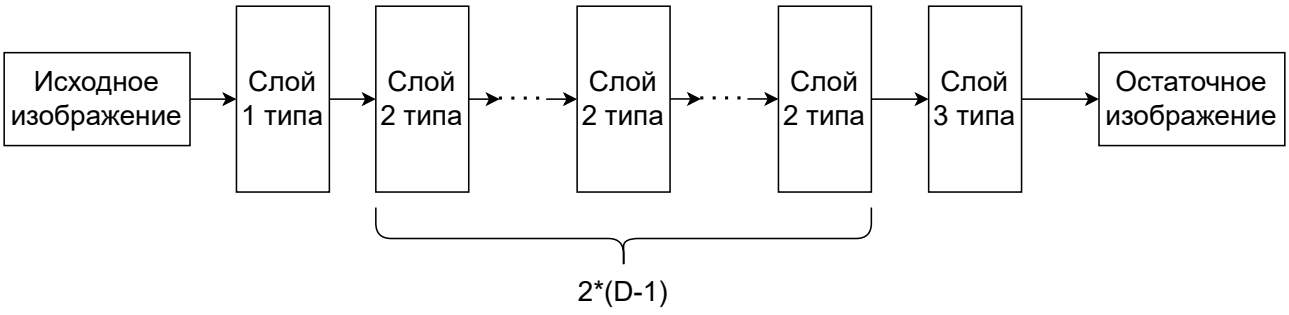

Рисунок 1.10 – Конфигурация нейронной сети в алгоритме DNCNN

На выходе алгоритма получается остаточное изображение R. Пусть Y – исходное изображение. Тогда очищенный снимок X можно посчитать по формуле 1.17:

$$X = Y - R. \tag{1.17}$$

1.4.6 RIDNet

Алгоритм RIDNet также использует сверточные нейронные сети, однако в его основе лежит другой подход [26]. Идея состоит в том, что не во всех ситуациях требуется, чтобы модель одинаково оценивала каждую цветовую составляющую пикселя, поэтому у каждой из составляющих будет свой вес [12].

Конфигурация сети состоит из трех основных модулей [12]:

1. **Модуль извлечения признаков** – он состоит из одного слоя, который производит свертку изображения и инициализирует признаки f_0.

2. **Модуль извлечения остаточных признаков из остаточного изображения** – выполняет основную работу в сети, состоит из четырех дополнительных модулей – EAM (Enhancing attention module).

3. **Модуль реконструкции изображения** – использует один слой, восстанавливает исходное изображение. ReLU используется в качестве передаточной функции.

При этом модули в алгоритме связаны между собой, например, после прохождения первого модуля полученный результат попадает на вход как второго модуля, так и третьего.

Общая конфигурация нейронной сети представлена на рисунке 1.11:

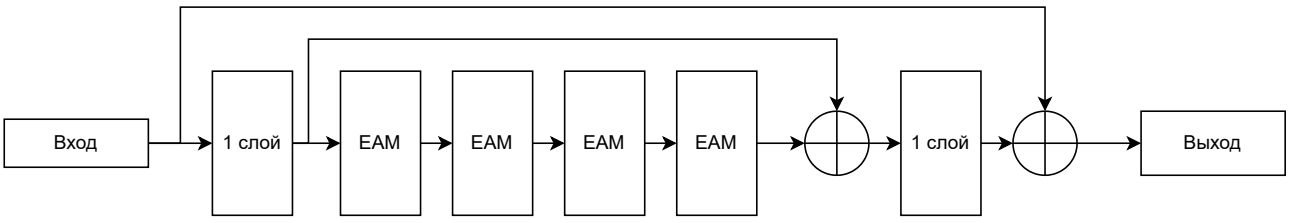

Рисунок 1.11 – Обшая конфигурация нейронной сети в алгоритме RIDNet

EAM внутри себя состоит из множества различных слоев, которые комбинируются между собой [12]. Каждый модуль внутри себя выполняет три действия: выделяет основные признаки из изображения, сжимает изображение для

большей производительности и увеличивает веса по наиболее значимым признакам.

Внутри модуля используются сетки размером $3x3$. В качестве функции активации на выходе последнего слоя EAM используется сигмоида. Пусть x – вход нейрона, тогда выход можно посчитать по формуле 1.18:

$$f(x) = \frac{1}{1 + e^x}. \tag{1.18}$$

При этом авторы алгоритма предложили использовать больше, чем четыре EAM-модуля, однако это не увеличило результаты работы алгоритма [26]. Также в алгоритме предусмотрено использование регуляризации для борьбы с переобучением системы.

1.5 Критерии сравнения алгоритмов

Для верификации работы алгоритмов используются открытые наборы реальных изображений. Алгоритмам предоставляют на вход загрязненные изображения, в результате получается очищенное изображение. У начальных изображений зафиксирован одинаковый размер, так как это позволяет зафиксировать размер входа для нейронной сети. Для полученного результата подсчитываются метрики, и на их основе устанавливается эффективность для метода.

Для сравнения результатов используется две основные метрики: PSNR и SSIM [27].

1.5.1 PSNR

PSNR – метрика, обозначающее пиковое отношение сигнала к шуму [28]. Она универсальна и используется не только для удаления шумов, а также для измерения уровня искажения при сжатии изображений. PSNR рассчитывается по логарифмической шкале.

Пусть есть два изображения I – исходное, и K – полученное в результате обработки. Размер каждого составляет MxN пикселей.

Разница между ними рассчитывается по формуле 1.19 [28]:

$$\text{MSE} = \frac{1}{MN} \sum_{0}^{M-1} \sum_{0}^{N-1} |I(i,j) - K(i,j)|. \tag{1.19}$$

Максимальное значение любого пикселя обозначим за MX. Чаще всего оно равно 255 [28]. Теперь искомую метрику PSNR можно рассчитать по формуле 1.20:

$$\text{PSNR} = 20 \log_{10} \left(\frac{\text{MX}}{\sqrt{\text{MSE}}} \right). \tag{1.20}$$

Метрика изначально была создана для монохромных изображений, поэтому для цветных изображений результат усредняется по каждой из цветовых составляющих пикселя.

1.5.2 SSIM

SSIM – метрика, обозначающая индекс структурного сходства изображений [29]. В отличие от PSNR, эта характеристика позволяет учесть не только разницу в фактических величинах между пикселями, но и взаимосвязь между ними, то есть структурное сходство.

Пусть есть два изображения I – исходное, и K – полученное в результате обработки. Размер каждого составляет MxN пикселей. Для каждого изображения были рассчитаны характеристики: μ_i – среднее по цветовой составляющей для изображения i, σ_i^2 – дисперсия цветовой составляющей для фотографии i, σ_{ij} – ковариация цветовой составляющей для изображений i и j.

Также вводится константа L, которая обозначает динамический диапазон. Как правило, она равняется 255 [29].

Исходя из этого, метрику SSIM можно рассчитать по формуле 1.21:

$$\text{SSIM} = \frac{(2\sigma_I^2\sigma_K^2 + 0.01L)(2\sigma_{IK}^2 + 0.03L)}{(\mu_I^2 + \mu_K^2 + 0.01L)(\sigma_I^2 + \sigma_K^2 + 0.03L)}. \tag{1.21}$$

1.6 Классификация существующих решений

Классификацию существующих алгоритмов можно привести к таблице. В качестве основных параметров классификации были использованы следующие признаки:

— учитывается ли анизотропная составляющая при очистке изображения от шумов;

— учитывают ли алгоритмы взаимосвязь пикселей между собой;

— используются ли алгоритмом нейронные сети;

— происходит ли сглаживание в ходе работы алгоритма;

— какой размер изображения требуется для его работы;

— для какого вида шумов используется алгоритм.

Классификация существующих алгоритмов представлена на таблице 1. Под заголовком Gauss приведены критерии для алгоритма гауссовского фильтра, под заголовком Median – для алгоритма медианного фильтра, Bilat – для билатерального фильтра, Jianwei – для алгоритма Цзяньвэй.

Таблица 1 – Таблица сравнения существующих алгоритмов

	Median	Gauss	Bilat	Jianwei	DnCNN	RIDNet
Учитывается ли анизотропия?	нет	нет	да	да	да	да
Учитывается ли взаимосвязь пикселей?	нет	да	да	да	нет	да
Происходит ли сглаживание?	нет	да	да	да	нет	нет
С каким шумом работает метод?	любой	гауссов	гауссов	любой	любой	гауссов
Используются нейронные сети?	нет	нет	нет	нет	да	да

Полученные результаты можно свести к следующим тезисам:

— шум на изображениях возникает вследствие несовершенства технических средств. На практике применяются алгоритмические средства борьбы;

— существующие алгоритмы, не использующие нейронные сети, справляются лишь с определенным видом шумов, имея проблемы с устранением остальных типов;

— в алгоритмах DnCNN и RIDNet детали реализации сети могут быть изменены: функция активации, количество слоев;

— новые решения в области удаления шумов являются лишь улучшением или комбинированием уже существующих алгоритмов.

1.7 Формулировка цели

В ходе выполнения работы требуется разработать метод, позволяющий удалить импульсные шумы из цветных изображений. Метод должен отвечать следующим требованиям:

1. Должны удаляться импульсные шумы для цветных изображений любого размера.
2. Гарантируется, что максимальное количество процентов шума – до 30% зашумленных пикселей.
3. Исходное количество шума неизвестно.
4. Для выполнения цели должны быть использованы нейронные сети.

1.8 Формализация задачи. IDEF0-диаграмма

Для формализации задачи была составлена IDEF0-диаграмма нулевого и первого уровня.

IDEF0-диаграмма нулевого уровня представлена на рисунке 1.12:

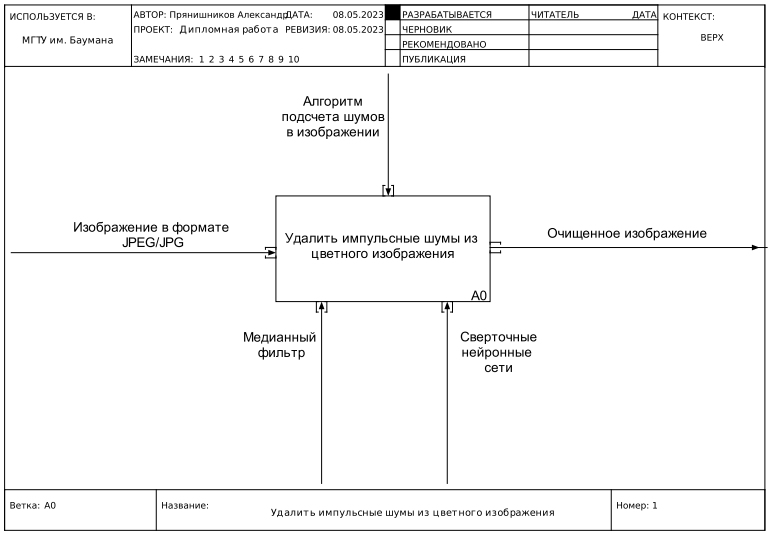

Рисунок 1.12 – IDEF0-диаграмма нулевого уровня

IDEF0-диаграмма первого уровня представлена на рисунке 1.13:

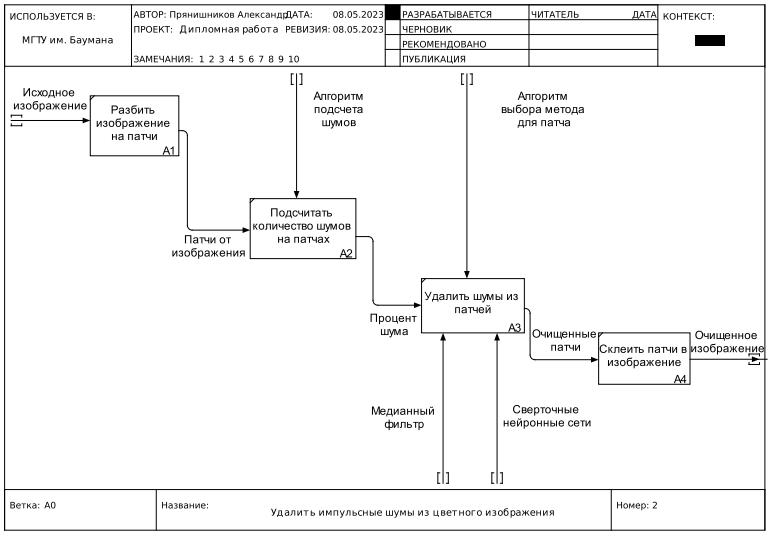

Рисунок 1.13 – IDEF0-диаграмма первого уровня

Выводы

Был проведен анализ предметной области, введены основные понятия. Описаны физические причины появления шумов на изображениях. Разобраны основные алгоритмы, которые используются в области удаления шумов, представлены схема их работы. Результаты сравнительного анализа представлены в табличном виде.

Была сформирована цель работы. Задача была формализована в виде IDEF0-диаграммы.

2 Конструкторский раздел

В этом разделе выделяются требования к разрабатываемому методу. Кратко описывается спроектированный метод и рассматриваются его основные особенности. Для основных алгоритмов, использующихся в методе, будут приведены графические иллюстрации. Описываются конфигурации нейронных сетей и датасет для их обучения. Составляется структура программного комплекса, описывается взаимодействия отдельных частей системы.

2.1 Требования к разрабатываемому методу

Метод очищения цветного изображения от импульсных шумов должен соответствовать следующим требованиям:

1. Принимать на вход изображение в форматах JPG или JPEG, а также вид шума – шум соли или перца. Гарантируется, что процент шума на изображении не превышает 30%.
2. Очистить изображение от импульсных шумов выбранного типа.
3. Вывести на экран результат работы алгоритма: загрязненное и очищенное изображения.
4. Использовать в своей работе сверточные нейронные сети.

При этом допускается использовать помимо сверточных нейронных сетей и другие алгоритмы, позволяющие реализовать комбинированный алгоритм.

2.2 Требования к программному комплексу

Програмнный комплекс, реализующий интерфейс к разработанному методу, должен позволять выполнять следующие действия:

— возможность загрузки изображений с помощью интерфейса;

— возможность выбора типа искомого шума: шум соли или шум перца;

— возможность просмотра результата работы метода и сравнения с исходным изображением.

Для загрузки изображений разрешается использовать относительный путь до файла в операционной системе. Также требуется добавить возможность сравнения полученного изображения с оригинальным.

2.3 Проектирование метода удаления шумов

Разработанный метод должен учитывать следующие особенности ранее известных алгоритмов:

1. Метод медианной фильтрации позволяет очистить изображения от большинства проблемных пикселей, но искажает цветовую составляющую исходного изображения и создает эффект сглаживания.

2. Нейронная сеть RIDNET плохо справляется с импульсными шумами. В случае обучения на картинках с импульсными шумами она подстраивается на определенный их процент.

3. Конфигурация нейронной сети RIDNET позволяет обучить ее для того, чтобы корректировать цвета пикселей, полученные в результате применения медианного фильтра.

2.3.1 Основной алгоритм работы

Учитывая особенности известных методов, перечисленные ранее, был разработан метод, позволяющий бороться с шумами на изображениях.

Картинку требуется разбить на изображения небольшого размера, называющиеся патчами. Это требуется для корректного обучения нейронной сети.

Затем нужно оценить количество шума на патче. В случае, если оно больше, чем некое пороговое значение, то предварительно применить медианный фильтр. Результат нужно обработать в нейронной сети, способной к коррекции изображения. По итогу предварительный результат попадает на нейронную сеть, предназначенную для очистки изображения от шумов.

Схема алгоритма представлена на рисунке 2.1:

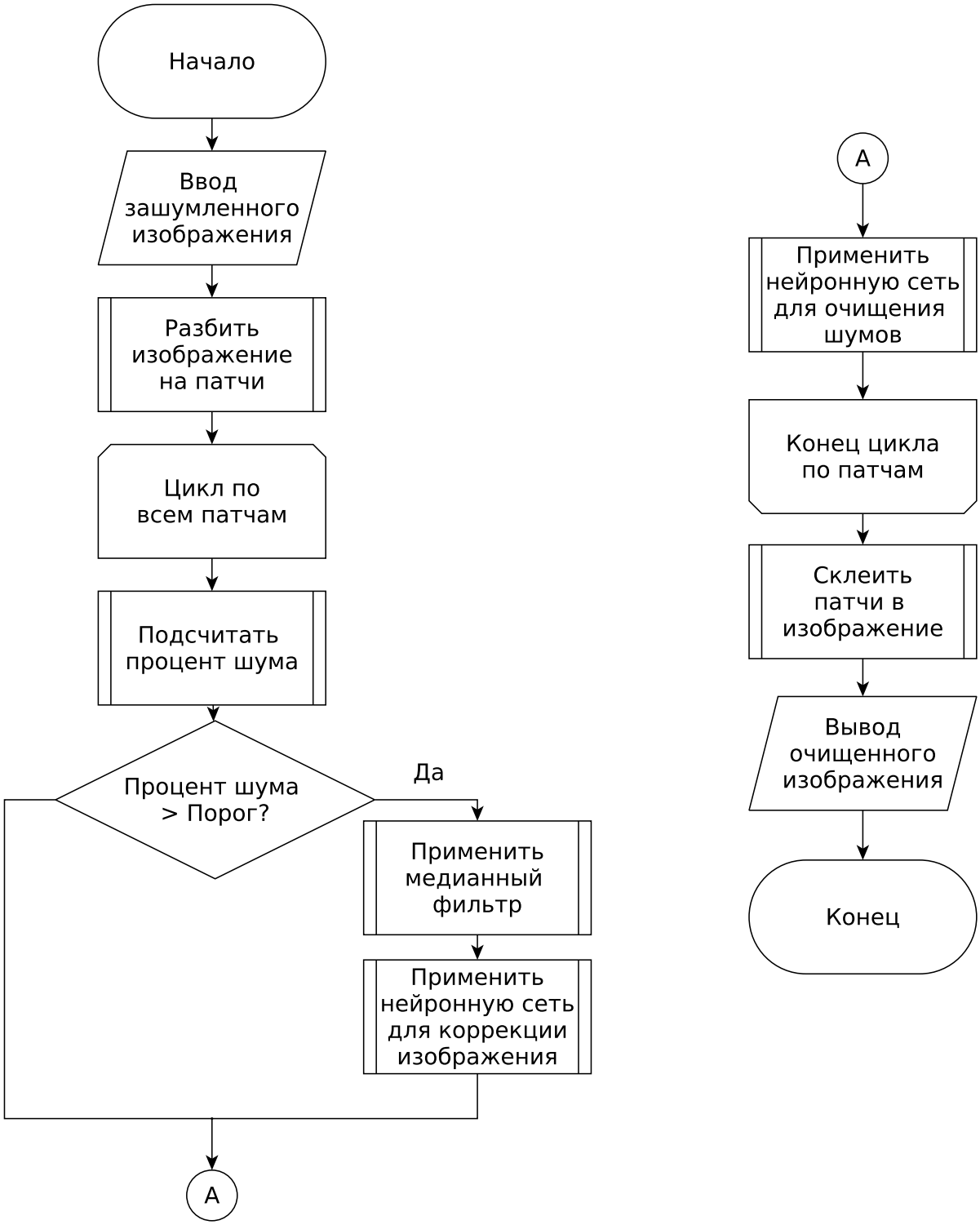

Рисунок 2.1 – Верхнеуровневая схема алгоритма

2.3.2 Подсчет количества шумов в изображении

Для принятия решения об использовании медианного фильтра требуется подсчитать, превышает ли количество импульсных шумов необходимый по-

рог.

Все цвета пикселей делятся на следующие три категории:

1. Цвета, принимающиеся за шум соли.
2. Цвета, принимающиеся за шум перца.
3. Цвета, не принимающиеся за шум.

Однако главная сложность состоит в том, что пиксель может быть ошибочно принят за шум, если он находятся на фоне, соответствующим шуму. Например, если рассматривать яркое небо, то почти все пиксели в этом случае можно интерпретировать как шум соли. Соответственно, нужно учитывать контекст для принятия решения.

Для этого требуется оценить цвет рассматриваемого пикселя по сравнению с соседями. Нужно подсчитать среднюю разницу цветов с соседними пикселями, в соответствии с сеткой, используемой в медианном фильтре. Если она получается больше, чем заданная разница, то пиксель определяется как шум, иначе – как исходный пиксель. Заданная разница рассчитывается экспертной оценкой.

В результате требуется посчитать количество пикселей, принятых за шум.

Схема алгоритма подсчета количества шумов представлена на рисунке 2.2:

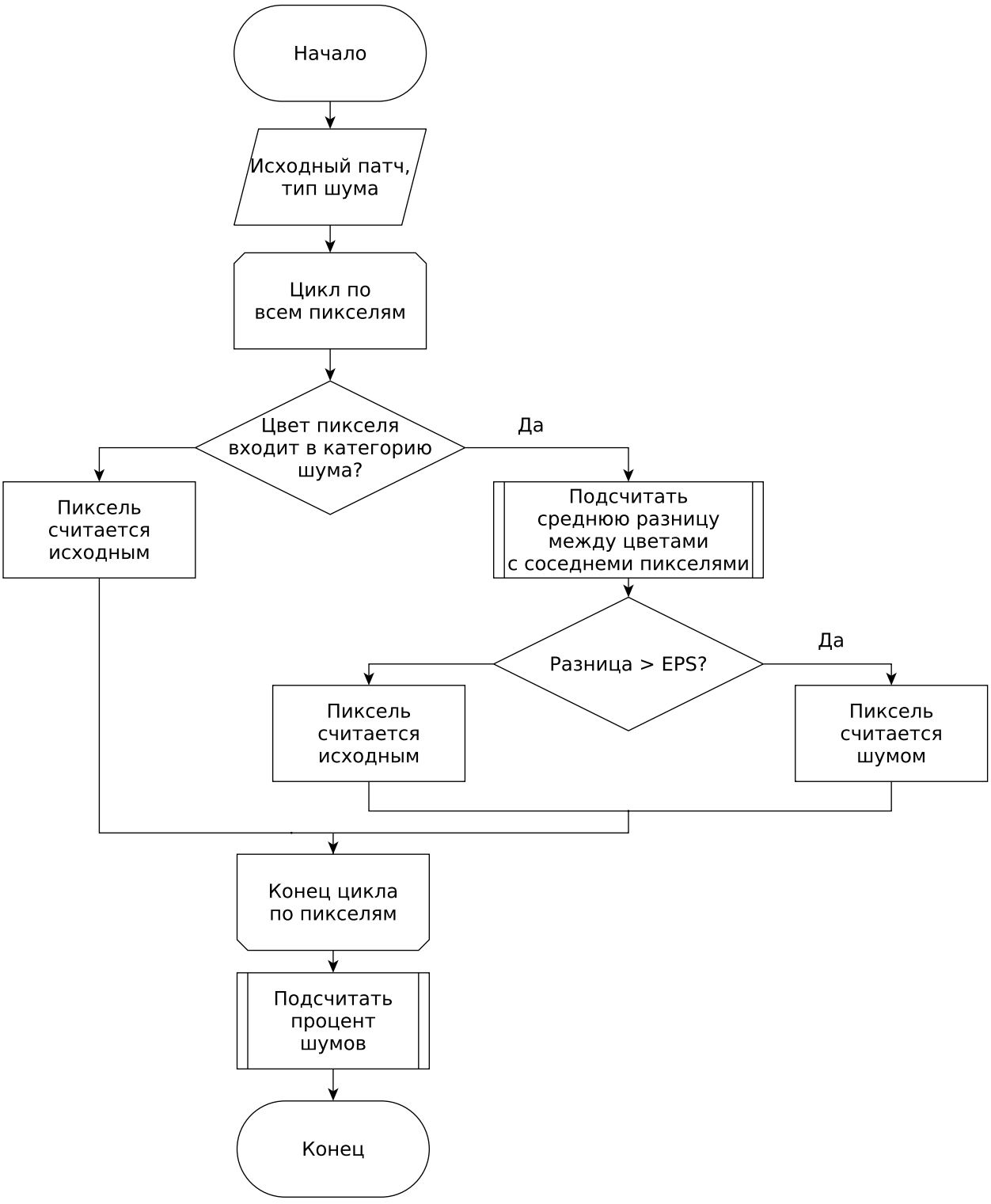

Рисунок 2.2 – Схема алгоритма подсчета количества шумов

Так как размер патча заранее известен, достаточно найти отношения импульсного шума к общему число пикселей.

2.3.3 Конфигурация нейронных сетей

В качестве основы для итоговой конфигурации нейронной сети была выбрана модель RIDNET.

В нее были добавлены соответствующие изменения:

1. Она состоит из трех EAM-модулей, а не из четырех, как в исходном алгоритме.

2. В качестве функции активации выбрана сигмоида на всех элементах EAM-модуля, а не ReLu.

3. Функция потерь – среднеквадратичное отклонение от результата, а не абсолютная ошибка, как в RIDNET.

Конфигурация нейронной сети для очищения изображения от шумов также основана на RIDNET, но в качестве функции активации внутри EAM-модуля использовалась сигмоида для частей, склеивающих результат с нескольких каналов в один выход.

В качестве функции оптимизации был использован Adam (адаптивная оценка момента). Вместо того чтобы адаптировать скорость обучения параметров на основе среднего первого момента (среднего значения), Adam также использует среднее значение вторых моментов градиентов.

2.3.4 Данные для обучения модели

В качестве данных, используемых для обучения моделей, был использован датасет BSDS500 [26], использующий для верификации алгоритмов, специализирующихся на очищении гауссовых шумов из изображений. Выбор обусловлен тем, что он уже использовался для оценки качества работы других нейронных сетей, перечисленных в аналитическом разделе, соответственно, не придется дополнительно обучать другие модели и применять фильтры для сравнения результата. Также множество изображений было составлено таким образом, что цветовая гамма изображений в них отличается, что позволит проверить метод в различных ситуациях.

Однако для моделирования импульсных шумов требуется разработать собственные функции.

2.4 Структура программного комплекса

Разрабатываемый программный комплекс состоит из трех модулей:

1. Модуль разработки и обучения нейронных сетей, в котором также происходит разработка датасета.
2. Модуль обработки изображений.
3. Модуль взаимодействия с пользователем.

На рисунке 2.3 представлена схема модуля разработки и обучения нейронных сетей:

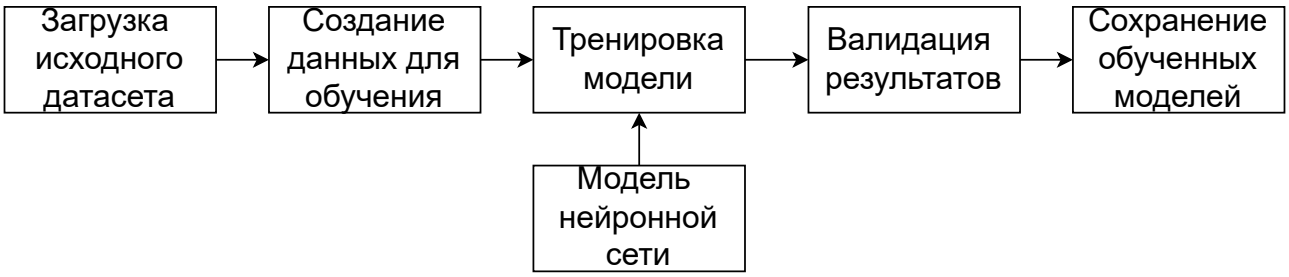

Рисунок 2.3 – Схема модуля разработки и обучения нейронных сетей

Валидация результата должна происходить по следующим правилам:

1. Значение PSNR для очищенного изображения превышает показатель для загрязненного изображения.
2. Значение PSNR больше, чем для реализации медианного фильтра.

Схема взаимодействия с пользователем представлена на рисунке 2.4:

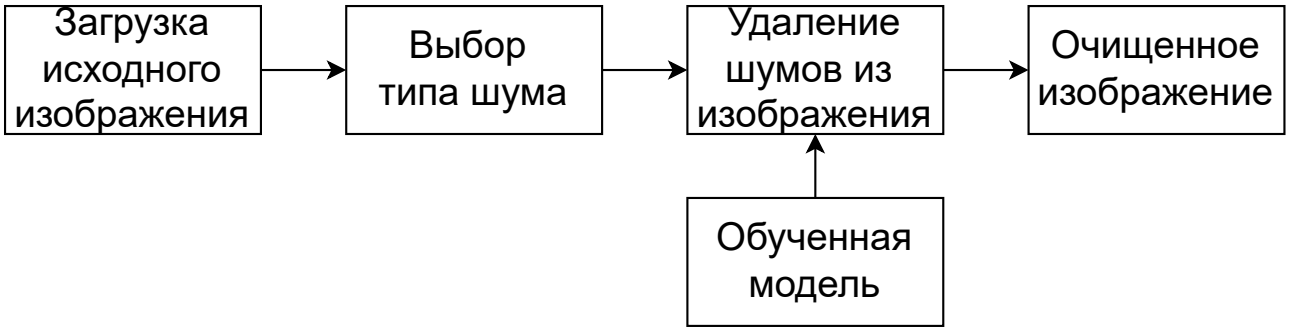

Рисунок 2.4 – Схема модуля взаимодействия с пользователем

Выводы

Были составлены требования к разрабатываемому методу. Был спроектирован метод с учетом особенностей уже существующих способов, описаны его особенности, составлены схемы основных алгоритмов. Была разработана конфигурация нейронных сетей, описан путь составления датасета для его обучения. Была составлена структура программного комплекса, описаны взаимодействия отдельных частей системы.

3 Технологический раздел

В этом разделе обосновывается выбор средств программной реализации метода. Будут приведены результаты разработки программного обеспечения, реализующее метод удаления импульсных шумов из цветных изображений. Будут описан формат входных и выходных данных. Будут показаны демонстрация работы программы и принципы, применяющиеся в тестировании. Будут приведены результаты предварительного тестирования разработанного программного комплекса.

3.1 Средства реализации программного комплекса

Перед тем, как программно реализовать метод, нужно перечислить требования к средствам реализации программного комплекса:

1. Язык программирования должен иметь библиотеки, позволяющие работать со сверточными нейронными сетями а также с большими массивами данных.

2. Среда разработки должна поддерживать выбранный язык программирования.

3. Требуется ориентироваться на облачные вычисления, так как они позволяют не использовать локальные вычислительные ресурсы.

В качестве среды разработки был использован проект Yandex DataSphere. Он позволяет на платной основе использовать облачные вычислительные ресурсы, в частности процессор NVIDIA Tesla V100.

Также в возможности DataSphere входит версионирование моделей и сохранение сеанса сессии, в отличие от аналогов.

3.1.1 Выбор языка программирования

Для написания программного обеспечения был использован язык Python. Это обусловлено следующими причинами:

— имеется опыт разработки на данном языке;

— для Python было реализовано множество библиотек, позволяющих работать со сверточными нейронными сетями.

Для работы с нейронными сетями была выбрана библиотека tensorflow версии 2.12.

Это обусловлено тем, что на текущий момент tensorflow показывает более высокие показатели производительности, чем другие аналоги, например, pytorch. Также у автора имеется опыт использования данной библиотеки для разработки сверточных нейронных сетей.

В качестве библиотеки для работы с изображениями был выбран OpenCV, так как в ней присутствуют модули, позволяющие взаимодействовать с функциями из tensorflow.

3.2 Формат входных и выходных данных

В качестве входного параметра выступают два обязательных параметра:

1. Путь до изображения в формате jpg или jpeg размера, большего чем 40 x 40 пикселей.

2. Тип импульсного шума, который нужно очистить: шум соли или шум перца.

Также пользователь имеет возможность указать путь до базового изображения для расчета метрик качества очистки изображения от шумов.

На выходе метод образует изображение такого же размера, что и исходное изображение, но очищенное от шумов. Если пользователь задал базовое изображение без шумов, то программа также рассчитывает и метрики качества очистки изображений.

3.3 Реализация программного комплекса

Реализация программного комплекса делится на три основные части: реализация и обучение нейронной сети, реализация модуля взаимодействия с поль-

зователем и реализация модуля обработки изображений. В комплекс действий также входит и создание собственного датасета для обучения модели, так как текущие датасеты содержат либо исходные изображения без шумов, либо искаженные гауссовым шумом. Также требуется реализовать алгоритмы, использующиеся в самом методе.

3.3.1 Нейронная сеть

Исходную базу изображений, требуемых для обучения модели, потребовалось разбить на небольшие изображения размером 40 × 40 пикселей, так как это предусмотрено в модели. Код алгоритма разбиения модели на пачти представлен на листинге 3 (приложение А).

Для увеличения набора тестовых данных некоторые изображения подвергались предварительным преобразованиям, таким как поворот и обрезка. Общий алгоритм сбора данных для обучения нейросети представлен на листинге 4 (приложение А).

Конфигурация всех нейронных сетей основана на методе RIDNET, но количество модулей EAM было уменьшено до трех в целях уменьшения времени, потраченного на обучения моделей. При этом в качестве функции активации используется сигмоида, что позволило улучшить точность результатов, однако увеличило время обучения сети. Реализация структуры представлена на листинге 5 (приложение А).

Модель использовала адаптивную оценку момента в качестве функции оптимизации. Метод требует вычисление экспоненциального скользящего среднего градиента и квадратичный градиент.

Реализация обучения нейронной сети представлена на листинге 6 (приложение А).

3.3.2 Результаты обучения нейронной сети

Обучение производилось с помощью процессора NVIDIA Tesla V100. Объем оперативной памяти машины составлял 392 ГБ, на видеопамять выделялось 32 ГБ памяти. Это позволило ускорить процесс обучения: одна эпоха занимала 51 секунду, в то время как на процессоре Inter Core i5 аналогичный процесс длился около 2 часов. Каждая из моделей прошла обучение в 20 эпох для достижения результата.

По результатам обучения модель выделила суммарно 1515731 параметров. В процессе обучения модель выдает два параметра после каждой эпохи:

— loss – значение функции потерь для обучающей выборки, с учетом того, что некоторые нейроны отбрасывались из обучения во время эпохи;

— val_loss – значение функции потерь для обучающей выборки, но нейросеть использует все нейроны.

График для метрики loss представлен на рисунке 3.1:

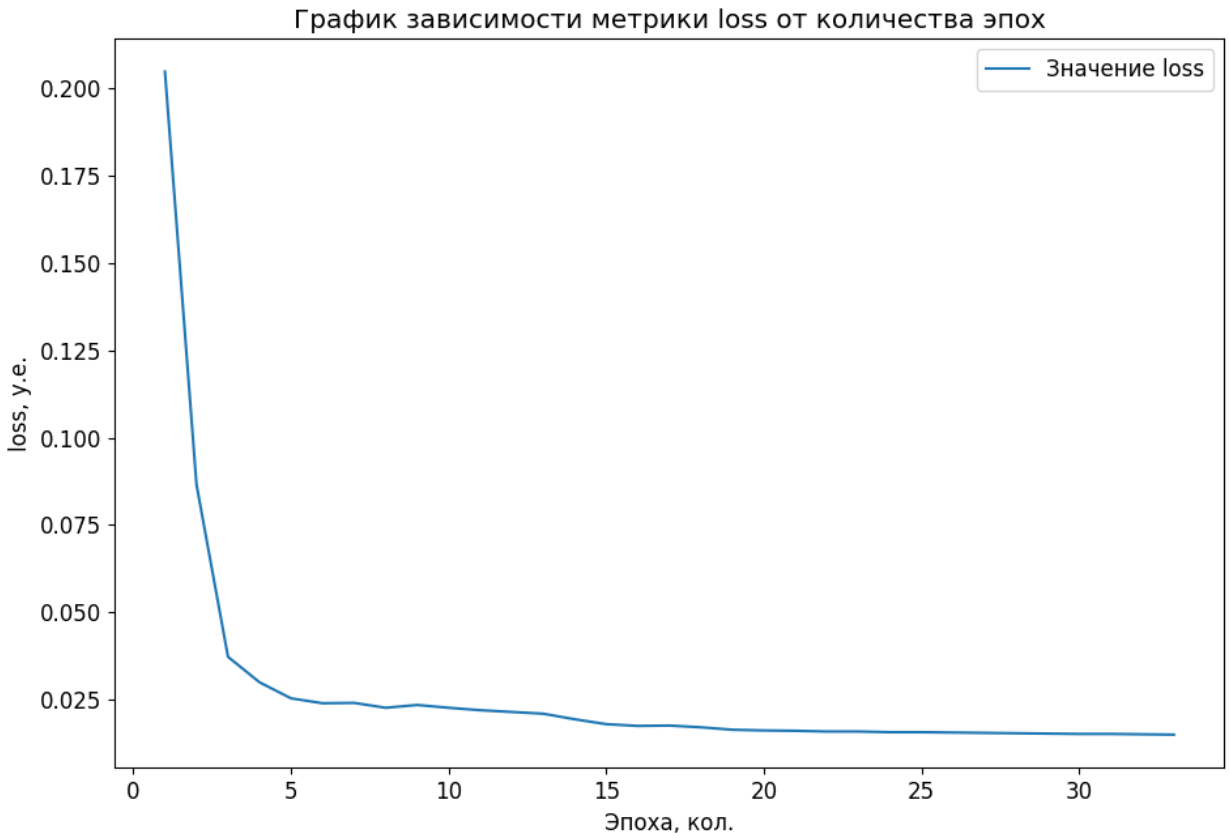

Рисунок 3.1 – График зависимости loss от эпохи

По графику можно сделать следующие выводы:

— график имеет вид гиперболы;

— уже к 5 эпохе значение loss составляет 0.025, в дальнейшем значение меняется слабо, соответственно, в случае недостатка вычислительных ресурсов пяти эпох уже достаточно для достижения результата.

График для метрики val_loss представлен на рисунке 3.2:

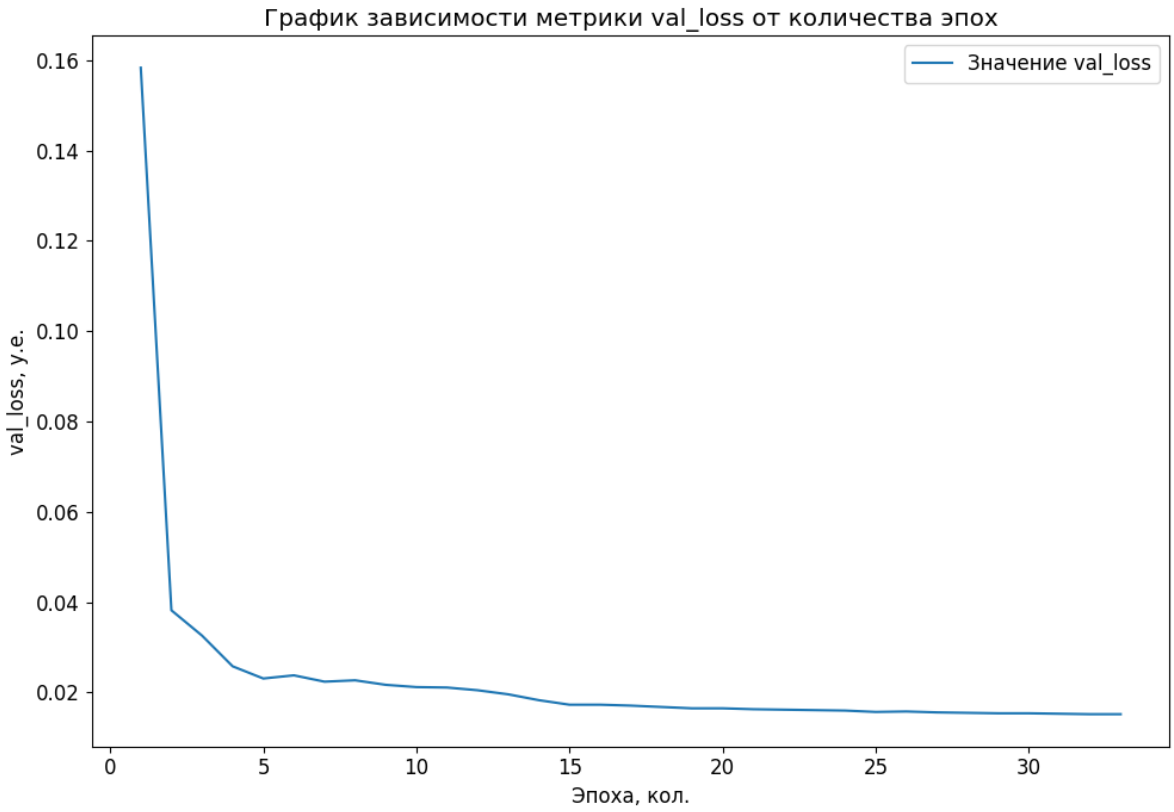

Рисунок 3.2 – График зависимости loss от эпохи

Значение метрики val_loss меньше, чем значение loss по состояние на ту же эпоху. График также имеет вид гиперболы, но до 15 эпохи график изменяется как дробно-линейная функция. Для достижения приемлемого результата также оказалось достаточно 5 эпох. Последние эпохи также не оказывают влияния на итоговые метрики модели.

Исходя из этого, можно сделать вывод, что в случае ограниченности вычислительных ресурсов достаточно обучить нейронную сеть на пяти эпохах. Для достижения результата будет достаточно 15-20 эпох, так как в дальнейшем функция потерь перестает изменяться.

3.3.3 Реализация метода

Для реализации метода требовались обученные нейронные сети, реализация которых приведена в предыдущем разделе.

Также требовалась реализация алгоритма, позволяющего оценить количество шума на патче. Реализация представлена на листинге 1:

Листинг 1 – Реализация алгоритма оценки количества шума

```
def count_noise(patch, i):
    counter = 0
    for row in patch:
        for col in row:
            if col[0] > COLOR[0] and col[1] > COLOR[1] and col[2] > COLOR[2]:
                counter += 1
            elif col[0] < COLOR[0] and col[1] < COLOR[1] and col[2] < COLOR[2]:
                counter += 1
    return counter > EPS
```

Реализация общего алгоритма работы представлена на листинге 7 (приложение Б).

Также для применимости медианного фильтра и нейронной сети для восстановления изображения нужно было провести некоторые преобразования. Исходное изображение представлено массивом из библиотеки numpy, в то время как нейронная сеть использует другое представление пикселей. Реализация алгоритма приведена на листинге 2:

Листинг 2 – Реализация медианного фильтра

```
def apply_median_filter(patch, i):
    if count_noise(patch, i):
        after_median_patch = np.array([cv2.medianBlur(patch, 3)])
        tf_type_patch = after_median_patch.astype('float32') / 255
        tf_type_patch = tf.clip_by_value(tf_type_patch)

        ridnet_median = tf.keras.models.load_model('model')
        corrected_patch = ridnet_median.predict(tf_type_patch)
        corrected_patch = (corrected_patch * 255).astype('int')
        return create_image_from_patches(corrected_patch, patch.shape)

    return patch
```

После применения всех алгоритмов к патчам требуется восстановить изображение. Код функции, выполняющий реставрацию полученного изображения из кусков представлен на листинге 8 (приложение Б).

3.4 Демонстрация работы программы

В результате разработки был программно реализован метод, позволяющий удалить шумы из изображения.

Стартовый экран для пользователя представлен на рисунке 3.3:

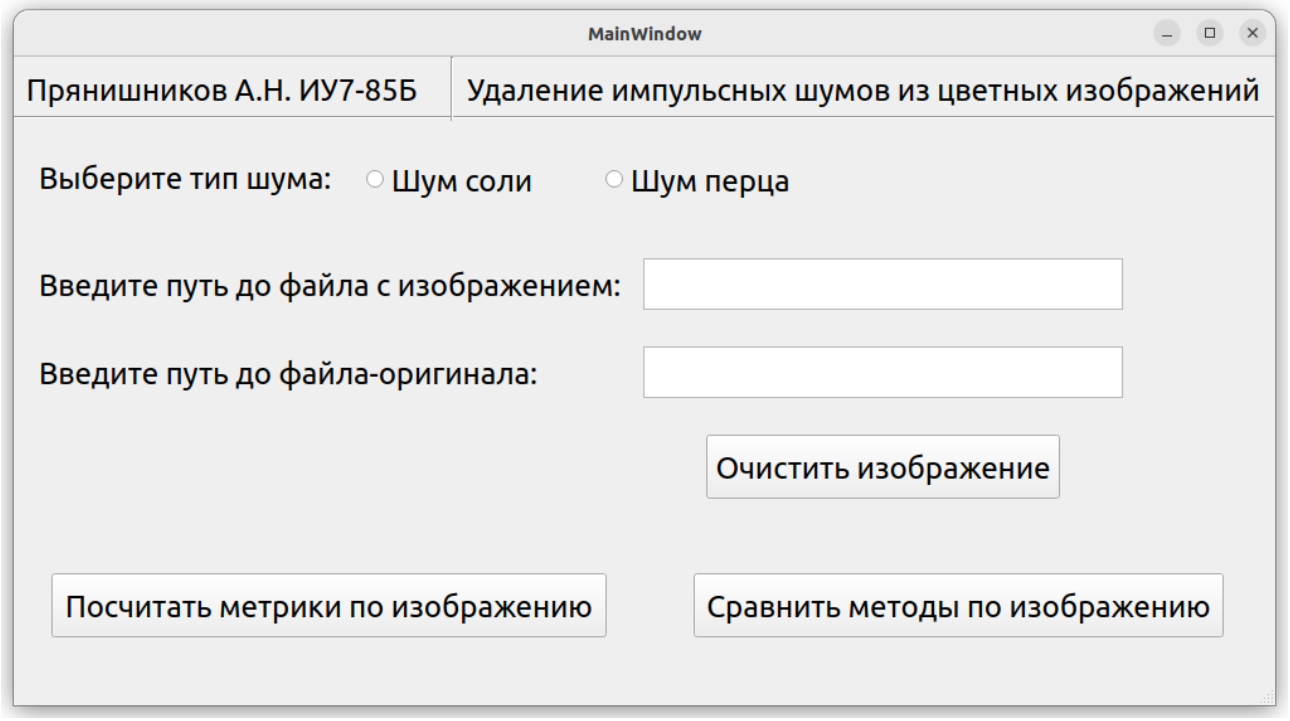

Рисунок 3.3 – Стартовый экран

Пользователю доступны следующие возможности:

1. Очистка изображений от шумов. Для этого нужно выбрать тип шума и ввести путь до загрязненного изображения.

2. Посмотреть значение метрики PSNR для конкретного изображения. Для этого нужно помимо параметров для очистки требуется указать путь до оригинального изображения.

3. Сравнить с другими методами. В этом случае не требуется вводить дополнительные данные. Программа построит график сравнения эффективности методов на заранее заданном датасете.

Пример работы по очистке изображений представлен на рисунке 3.4:

Рисунок 3.4 – Пример работы программы

3.5 Тестирование разработанного метода

Для тестирования метода были сформулированы следующие требования:

1. Тестирование происходит на тестовой выборке, загрязненной шумами нужного типа. Процент шума при этом между изображениями отличается.
2. Зашумленность изображения не превышает 30%. Этого достаточно для оценки корректности работы модели.
3. Модель должна давать по каждому изображению значение метрик большее, чем у исходного загрязненного изображения.

Тестирование проводилось на исходном датасете, но в изображения был добавлен синтетический шум. Среднее значение по каждому проценту шума усреднялось, но шум соли и перца учитывался как один вид шума.

Результаты по загрязненности шума были визуализированы с помощью графика, который представлен на рисунке 3.5:

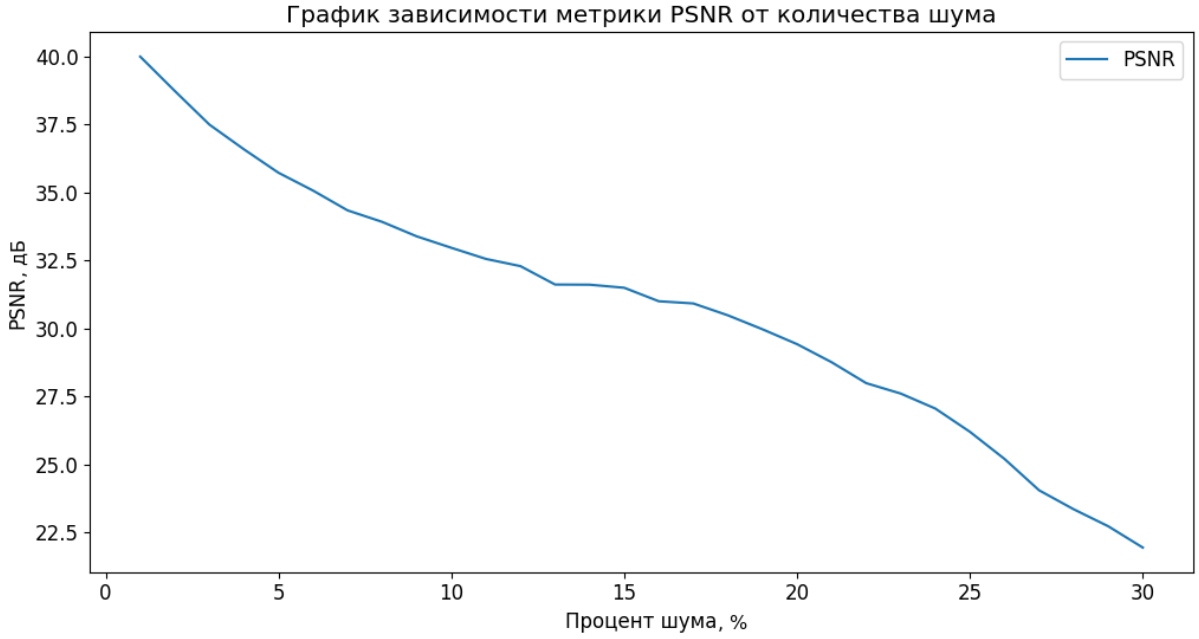

Рисунок 3.5 – Результаты тестирования разработанного метода

Видно, что метрики эффективности метода уменьшаются с ростом количества шумов, но результат соответствует поставленным требованиям.

Выводы

Был обоснован выбор средств программной реализации методы. Был реализован метод, позволяющий удалить импульсные шумы из цветных изображений. Было показано, что для обучения нейронной сети достаточно пяти эпох, чтобы достичь заметного результата, приведен график зависимости метрик нейронной сети от эпох.

Было создано приложение с графическим интерфейсом, позволяющее применить метод на практике и оценить его эффективность. Было проведено тестирование, которое показало, что метод является работоспособным на изображениях.

4 Исследовательский раздел

В этом разделе будет проведено исследование эффективности разработанного программного обеспечения. Будет исследована зависимость метрики PSNR от выбранного порога применения медианного фильтра. Будет проведено сравнение результатов работы реализованного метода с результатами, полученными с помощью известных аналогов. Будет произведена оценка разработанного программного комплекса.

4.1 Исследование зависимости метрики PSNR от порога применения

В описании алгоритма было указано, что применение к исходному изображению медианного фильтра зависит от выбранного порога, который требовалось практически.

Для исследования зависимости было провести тестирование метода на трех различных значениях фильтра: 5% загрязненности, 15% загрязненности и 25% загрязненности изначального изображения, и затем сравнить поведение метрик в зависимости от процента шума.

Требуется определить, при каком пороге значение метрик будет расти с началом применения медианного фильтра и восстанавливающей нейронной сети. В дальнейшем предполагается использование выбранного порога для сравнения с другими методами.

Результаты сравнения с аналогами представлен на графике 4.1:

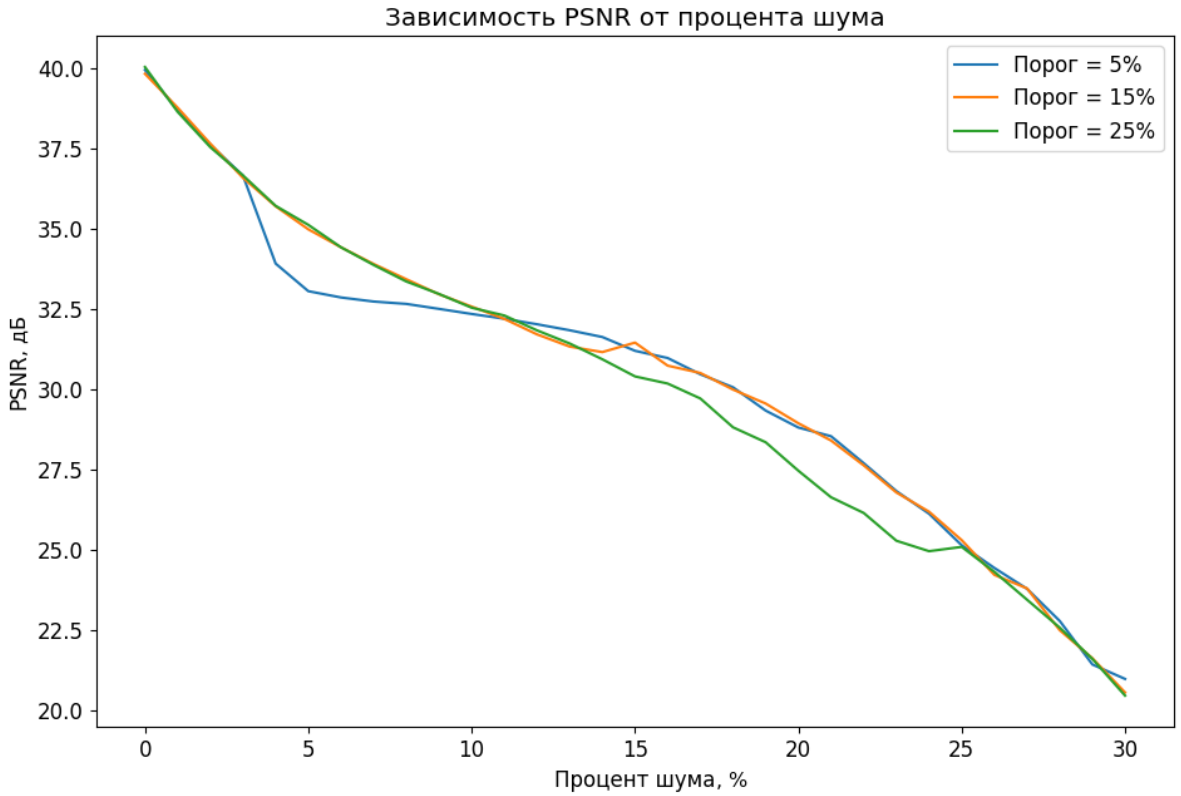

Рисунок 4.1 – Сравнение эффективности метода в зависимости от порога

Выяснилось, что наилучшие результаты показал алгоритм, применяющий фильтр в случае 15% уровня загрязненности патча. При 5% порога падение оказывается слишком резким, при 25% уровне порога значение PSNR снижается быстрее, чем при 15% пороге.

4.2 Сравнение реализованного метода с аналогами

В качестве аналогов, которые рассматривались для сравнения с реализованным методов, были выбраны:

1. Медианный фильтр.
2. Билатеральный фильтр.
3. RIDNET.

Этот выбор связан с тем, что фильтры являются самыми распространенными методами борьбы с импульсными шумами, а разработанный алгоритм позволяет устранить недостатки метода RIDNET.

Результаты сравнения с аналогами представлен на графике 4.2:

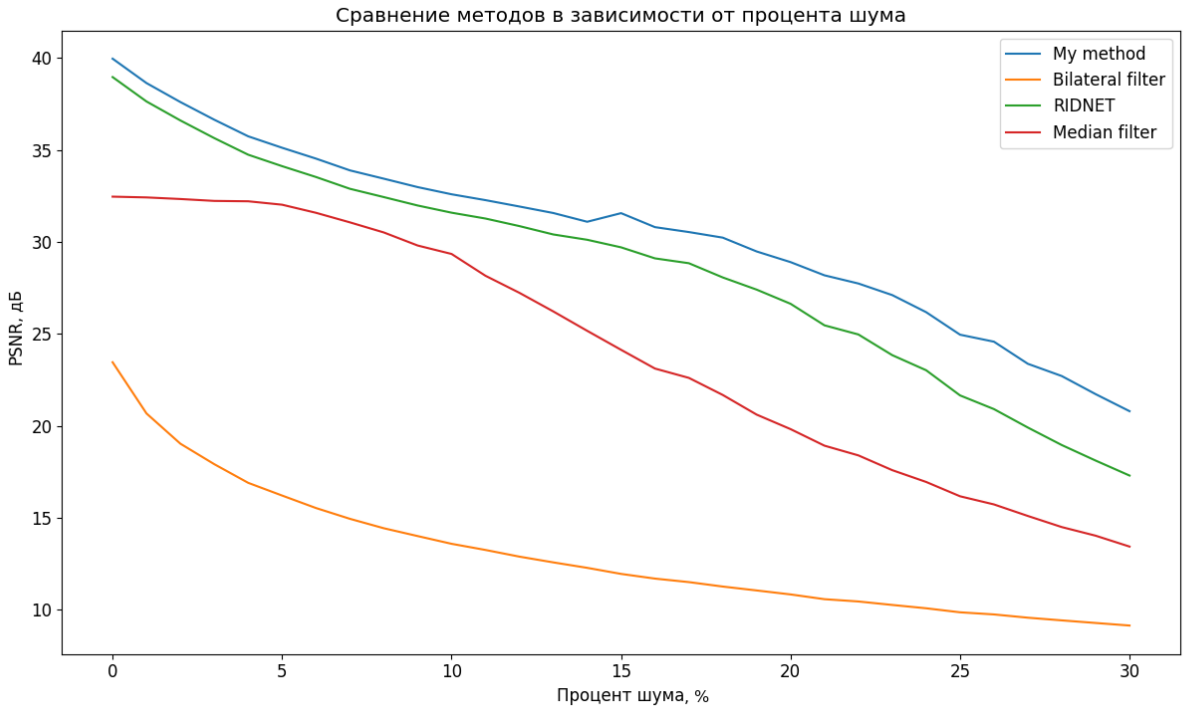

Рисунок 4.2 – Сравнение методов в зависимости от типа шума

Из графика можно сделать следующие выводы:

— все методы имеют тенденцию к снижению метрики PSNR с увеличением количества шума на изображении;

— реализованный в работе метод является самым эффективным алгоритмом среди всех перечисленных;

— разработанный метод дает в результате более высокие метрики, чем исходный RIDNET, особенно на степени загрязненности изображения 15% и больше.

Разработанный алгоритм позволил лучше справляться с изображениями, которые значительно загрязнены шумами.

4.3 Оценка разработанного программного комплекса

У разработанного программного комплекса можно выделить следующие **преимущества**:

1. Универсальность. Алгоритм работает с любым размером изображений, превышающих минимальный размер – 40 x 40 пикселей.
2. Алгоритм не требует от пользователя предварительного ввода процента шума.
3. Алгоритм позволяет работать с любым типом импульсного шума: шума соли или шума перца.
4. По сравнению с известными аналогами, алгоритм показывает более высокие результаты.

Также можно выделить некоторые **недостатки** метода:

1. Изображение на выходе получается слегка размытым, что является следствием применения медианного фильтра.
2. Медианный фильтр влияет на все пиксели изображения, в то время как можно его применять лишь к пикселям, идентифицированным как шум.
3. Не поддерживаются другие форматы, кроме JPG.

Выводы

Было проведено сравнение результатов работы реализованного метода с результатами, полученными с помощью известных аналогов. Выяснилось, что алгоритм показывает лучшие показатели метрик по сравнению с ранее рассмотренными аналогами. По результатам исследования эффективности ПО выяснилось, что оптимальный порог для применения медианного фильтра – 15% уровня загрязненности патча. Была произведена оценка разработанного программного комплекса, где были выделены основные достоинства и недостатки ПО.

ЗАКЛЮЧЕНИЕ

Была рассмотрена задача удаления импульсных шумов из цветных изображений. Была описана предметная область: определение понятия шума, причины его возникновения.

Были даны определения основных понятий нейронной сети, описан алгоритм их работы. Были описаны особенности применения нейронных сетей для борьбы с импульсными шумами в изображениях.

Были рассмотрены и проанализированы существующие методы (медианный фильтр, билатеральный фильтр, DNCNN, RIDNET) для удаления импульсных шумов из изображений, определены их недостатки.

Был разработан метод удаления импульсных шумов из цветных изображений с помощью сверточных нейронных сетей, а также программный комплекс, реализующий графический интерфейс для пользователей.

Были собраны и обработаны данные для обучения моделей, подобрана оптимальная конфигурация.

Был разработан программный комплекс, в который входят следующие модули: модуль разработки обучения моделей, модуль обработки загрязненных изображений. Приведены результаты обучения модели, основные метрики нейронной сети были показаны графически. Выбрано оптимальное количество эпох для проведения обучения сети.

Проведено исследование качества созданного программного комплекса. В результате исследования были выделены преимущества разработанного метода, в том числе: универсальность, независимость от типа импульсного шума, отсутствие предварительной информации о количестве шума, устойчивость результатов в зависимости от процента шума на изображении в сравнении с другими методами.

Таким образом, поставленная цель работы — разработать метод удаления импульсных шумов из цветных изображений с помощью сверточных нейронных сетей, была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Thanh Hien Nguyen. Image Noise Removal Method Based on Thresholding and Regularization Techniques // IEEE Access. — 2022. — С. 71584–71597.
2. Khushi H. Impulse Noise Removal Using Soft-computing // Lahore Garrison University Research Journal of Computer Science and Information Technology. — 2022. — С. 32–48.
3. Nguyen M. BoostNet: A Boosted Convolutional Neural Network for Image Blind Denoising // IEEE Access. — 2021.
4. Rabie T. Adaptive Median Filtering of CCD Sensor Noise. — 2005.
5. Зотов П.В. Цифровой шум изображения и его прикладное значение в криминалистике // Вестник СГЮА. — 2015.
6. Голдобин И.А. Влияние шумов на алгоритмы цифровой обработки изображений // Актуальные вопросы развития современной цифровой среды. — 2021. — С. 396–402.
7. Hambal Abdalla Mohamed. Image Noise Reduction and Filtering Techniques // International Journal of Science and Research (IJSR). — 2017. — С. 2033–2037.
8. Xiangming H. An improved nonlocal means based correction strategy for mixed noise removal // IET Image Processing. — 2022.
9. Ćeranić Š. Noise reduction quality test for two-photon laser scanning microscopic images // IEEE Access. — 2022.
10. Ganesan G. Noise Detection in Images using Moments // Research Journal of Applied Sciences, Engineering and Technology. — 2015. — С. 307–314.
11. Fujimoto M. Noise Removal for Degraded Images Using Neural Networks // IEEJ Transactions on Electronics, Information and Systems. — 2002. — С. 1301–1308.
12. Li X. RIDNet: Recursive Information Distillation Network for Color Image Denoising. — 2019. — С. 3896–3903.

13. Aylwin C. ReLU Neural Network Galerkin BEM // Journal of Scientific Computing. — 2023. — 03. — Vol. 95.

14. Rafidison M. Image Classification Based on Light Convolutional Neural Network Using Pulse Couple Neural Network // Computational Intelligence and Neuroscience. — 2023. — 03. — Vol. 2023. — C. 1–17.

15. Xiyu Lin. Research of Convolutional Neural Network on Image Classification // Highlights in Science, Engineering and Technology. — 2023. — 04. — Vol. 39. — C. 855–862.

16. Karrach E. Using a Convolutional Neural Network for Machine Written Character Recognition. — 2023. — 03.

17. Ziming Zhang. Sle-CNN: a novel convolutional neural network for sleep stage classification // Neural Computing and Applications. — 2023. — 04. — C. 1–16.

18. Ruishen L. Face Recognition Based on Convolutional Neural Networks // Highlights in Science, Engineering and Technology. — 2022. — 11. — Vol. 16. — C. 32–39.

19. Bandyopadhyay A. High Density Impulse Noise Removal from Color Images by K-means Clustering based Detection and Least Manhattan Distance-oriented Removal Approach // International Journal of Advanced Computer Science and Applications. — 2021. — C. 608–614.

20. Khan S. An adaptive dynamically weighted median filter for impulse noise removal // EURASIP Journal on Advances in Signal Processing. — 2017.

21. R. Kunsoth. Modified decision based median filter for impulse noise removal. — 2016. — C. 1316–1319.

22. Nabahat M. Optimization of bilateral filter parameters using a whale optimization algorithm // Research in Mathematics. — 2022.

23. Wang Jianwei. A Noise Removal Algorithm of Color Image // TELKOMNIKA Indonesian Journal of Electrical Engineering. — 2014.

24. Murali V. Image Denoising Using DnCNN: An Exploration Study // IEEE Access. — 2020. — C. 847–859.

25. Zhang K. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising // IEEE Transactions on Image Processing.

26. Loubere N. RiDNet Practical Fieldwork Notes. — 2012.

27. Setiadi D. PSNR vs SSIM: imperceptibility quality assessment for image steganography // Multimedia Tools and Applications. — 2021.

28. Nouman M. Evaluation of Image Super Resolution Deep Learning Technique based on PSNR Value. — 2022. — C. 93–122.

29. Subramanian V. SSIM Compliant Modeling Framework With Denoising and Deblurring Applications // IEEE Transactions on Image Processing. — 2021.

ПРИЛОЖЕНИЕ А

Модуль работы с нейресетью

Листинг 3 – Функция разбиения изображения на патчи

```
def get_patches(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    height, width, channels = image.shape
    crop_sizes = [1]
    patch_size = 40
    patches = []
    for crop_size in crop_sizes:
        crop_h, crop_w = int(height * crop_size), int(width * crop_size)
        image_scaled = cv2.resize(image, (crop_w, crop_h))
        for i in range(0, crop_h - patch_size + 1, int(patch_size / 1)):
            for j in range(0, crop_w - patch_size + 1, int(patch_size / 1)):
                x = image_scaled[i:i + patch_size, j:j + patch_size]
                patches.append(x)
    return patches
```

Листинг 4 – Создание датасета для нейронной сети

```
def create_dataset(src_dir, noise_level = 0.1):
    all_file_path = [filename for filename in os.listdir(src_dir)]
    for file in all_file_path:
        image = cv2.imread(src_dir+file)
        patches = get_patches(image)
        i = 0
        for patch in patches:
            processed_image = add_impulse_noise(patch, noise_level)
            cv2.imwrite(src_dir[:len(src_dir)-1]+"_data_1/"+file+"_"+str(i),
               patch)
            cv2.imwrite(src_dir[:len(src_dir)-1]+"_shum_1/"+file+"_"+str(i),
               processed_image)
            i += 1
```

Листинг 5 – Структура нейронной сети

```
def EAM(input):
    x=Conv2D(64, (3,3), dilation_rate=1,padding='same',activation='relu')(input)
    x=Conv2D(64, (3,3), dilation_rate=2,padding='same',activation='relu')(x)

    y=Conv2D(64, (3,3), dilation_rate=3,padding='same',activation='relu')(input)
    y=Conv2D(64, (3,3), dilation_rate=4,padding='same',activation='relu')(y)

    z=Concatenate(axis=-1)([x,y])
    z=Conv2D(64, (3,3),padding='same',activation='relu')(z)
    add_1=Add()([z, input])

    z=Conv2D(64, (3,3),padding='same',activation='relu')(add_1)
    z=Conv2D(64, (3,3),padding='same')(z)
    add_2=Add()([z,add_1])
    add_2 = Activation('relu')(add_2)

    z=Conv2D(64, (3,3),padding='same',activation='relu')(add_2)
    z=Conv2D(64, (3,3),padding='same',activation='relu')(z)
    z=Conv2D(64, (1,1),padding='same')(z)

    add_3=Add()([z,add_2])
    add_3 = Activation('relu')(add_3)

    z = GlobalAveragePooling2D()(add_3)
    z = tf.expand_dims(z,1)
    z = tf.expand_dims(z,1)

    z=Conv2D(4, (3,3),padding='same',activation='relu')(z)
    z=Conv2D(64, (3,3),padding='same',activation='sigmoid')(z)

    mul=Multiply()([z, add_3])

    return mul


def Model():
    input = Input((40, 40, 3),name='input')
    feat_extraction =Conv2D(64, (3,3),padding='same')(input)
    eam_1=EAM(feat_extraction)
    eam_2=EAM(eam_1)
    eam_3=EAM(eam_2)
    eam_4=EAM(eam_3)
    x=Conv2D(3, (3,3),padding='same')(eam_4)
    add_2=Add()([x, input])

    return Model(input,add_2)
```

Листинг 6 – Обучение нейронной сети

```
tf.keras.backend.clear_session()
tf.random.set_seed(6908)
ridnet = RIDNET()

ridnet.compile(optimizer=tf.keras.optimizers.Adam(1e-03), loss=tf.keras.losses.
    MeanAbsoluteError(), run_eagerly=True)

def scheduler(epoch, lr):
    return lr*0.9

checkpoint_path = "model"
checkpoint_dir = os.path.dirname(checkpoint_path)
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path)
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
lrScheduler = tf.keras.callbacks.LearningRateScheduler(scheduler)
callbacks = [cp_callback,tensorboard_callback,lrScheduler]
ridnet.fit(train_dataset, shuffle=True, epochs=2, validation_data=test_dataset,
    callbacks=callbacks)
```

ПРИЛОЖЕНИЕ Б

Модуль работы с изображениями

Листинг 7 – Общий алгоритм обработки изображений

```
def clean_image(imagefile):
    original_image = cv2.imread(imagefile)
    image = median_filter(original_image)

    patches = np.array(image).astype('float32') / 255
    patches = tf.clip_by_value(patches, clip_value_min=0., clip_value_max=1.)

    ridnet_new = tf.keras.models.load_model('models/ridnet_white_30.h5')

    denoised_image = predict_fun(ridnet_new, patches, original_image)

    src_patches = np.array(get_all_patches(original_image)).astype('float32') /
        255
    src_patches = tf.clip_by_value(src_patches, clip_value_min=0.,
        clip_value_max=1.)

    ot.plot_clean_image(original_image, denoised_image)

    return denoised_image, True
```

Листинг 8 – Функция восстановления изображения по патчам

```
def create_image_from_patches(patches, image_shape):
    image = np.zeros(image_shape)
    patch_size = patches.shape[1]
    p = 0
    for i in range(0, image.shape[0] - patch_size + 1, int(patch_size / 1)):
        for j in range(0, image.shape[1] - patch_size + 1, int(patch_size / 1)):
            image[i:i + patch_size, j:j + patch_size] = patches[p]  # Assigning
                values of pixels from patches to image
            p += 1
    return np.array(image)
```

ПРИЛОЖЕНИЕ В