|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОЙ РАБОТЕ***

***НА ТЕМУ:***

***Сравнительный анализ алгоритмов сглаживания изображения***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент ИУ9-51Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Д.А Окутин

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** А.Б Домрачева

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2023 г.*

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc125330552)

[1. Обзор выбранных алгоритмов сжатия 6](#_Toc125330553)

[1.1. Метод среднего сглаживания …………………………………………….6](#_Toc125330554)

1.2. Билатеральный фильтр……………………………………………………7

1.3. Метод медианного сглаживания……………………………………….....9

1.4. Метод минимума и максимума……………………………………….....10

[2. Особенности реализации выбранных алгоритмов на языке Python………1](#_Toc125330561)2

[2.1. Стек технологий……………..……………………………………………12](#_Toc125330554)

2.2. Описание работы программы……..……………………………………..12

[3. Сравнительный анализ резальтатов работы алгоритмов 14](#_Toc125330566)

[3.1. Описание параметров сравнения……………………………………......14](#_Toc125330554)

[3.2. Работа алгоритмов на изображениях с геометрическими фигурами....15](#_Toc125330554)

3.3. Работа алгоритмов на изображениях с шумами………………………..17

3.4. Работа алгоритмов на изображениях с документами………………….18

3.5. Работа алгоритмов на изображениях, не требущих сглаживания…….19

3.6. Общие результаты………………………………………………………..20

[ЗАКЛЮЧЕНИЕ ………………………………………………………………….22](#_Toc125330567)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ…………………………….23](#_Toc125330568)

[ПРИЛОЖЕНИЕ А……………………………………………………………….24](#_Toc125330569)

ВВЕДЕНИЕ

Известно, что любое растровое изображение, которое мы видим на экранах наших электронных устройств, представляет собой набор пикселей (англ. pixel), и так как растровая графика является дискретной, пиксель может быть либо полностью закрашенным, либо нет. Именно поэтому некоторые детали могут не отображаться, если они покрывают лишь четверть пикселя. А другие примитивы, наоборот, имеют слишком резкие переходы между парой пикселей, даже если сам предмет должен обладать плавными формами. Для решения проблемы, связанной со слишком резкими контурами изображений, были придуманы и разработаны алгоритмы сглаживания, применение которых способно давать ощутимые результаты к уровню восприятия человеком изображения на экране.

Чтобы начать разбираться с алгоритмами сглаживания прежде всего необходимо понять, что же такое сглаживание. Сглаживание (англ. anti-aliasing) — технология, используемая в обработке изображений с целью сделать границы кривых линий более гладкими и минимизировать визуальные дефекты при отображении изображений с высоким разрешением в системе с более низким разрешением [1].

Алгоритмами сглаживания называются наборы инструкций, выполняемые по средствам программного кода, направленные на улучшение визуального качества изображения. Такие алгоритмы применяются почти повсеместно: они могут быть как самостоятельной процедурой для улучшения качества изображения, так и первым шагом для более сложной процедуры, например, для распознавания объектов на изображении. Поэтому существует огромное множество способов сглаживания, использующие различные методы и подходы для достижения финального результата. Каждый из них имеет свои особенности, преимущества и недостатки. Некоторые алгоритмы основаны на математических моделях, таких как Гауссово распределение или медианный фильтр, в то время как другие основаны на интерполяции или статистическом моделировании [2].

Достаточно распространена следующая классификация алгоритмов сглаживания [3]:

1. Алгоритмы аппроксимации. Результатом работы такого рода алгоритмов является математическая функция, описывающая геометрический характер сглаживаемой линии. Параметры полученной функции могут быть сохранены и затем использованы для воссоздания контурной линии на произвольном количестве точек.

2. Алгоритмы, использующие различные геометрические отношения между точками. Такие алгоритмы могут убирать ненужные точки из исходного контура и генерировать дополнительные.

3. Алгоритмы на основе усреднения точек. Результатом работы подобного рода алгоритмов является набор точек, размер которого остаётся неизменным. Значения координат точек сглаженного контура получается путём усреднения координат соседних точек. Такие алгоритмы сглаживания относительно легко модифицировать.

В данной курсовой работе будет проведен сравнительный анализ следующих алгоритмов (каждый из приведенных алгоритмов относится к 3 типу, описанному выше): метод среднего сглаживания (mean filter), билатеральный фильтр (bilateral filter), метод медианного сглаживания (median filter), метод минимума и максимума (min-max filter).

Оцениваться данные алгоритмы будут по следующим параметрам: cредняя скорость работы алгоритма (на одинаковых входных данных при различных изображениях), метод структурного подобия (SSIM), устранение шумов (алгоритм сглаживания должен устранять шумы на изображении), устранение ступенчатости контуров (алгоритм должен сглаживать ступенчатые контуры и приводить их к более гладкому виду), заполнение деталей (оценка способности алгоритма сохранять или восстанавливать детали на изображении; желательно, чтобы сглаживание не приводило к потере информации или размытию важных элементов изображения).

Результаты данного анализа могут быть полезными при выборе оптимального алгоритма сглаживания изображений в зависимости от конкретной задачи или требований проекта в области компьютерной графики. Подходящий алгоритм сглаживания может улучшить качество и реалистичность изображений, а также повысить эффективность и скорость обработки графических данных.

Исходя из этого можно выделить объект и предмет исследования данной курсовой работы.

Объектом исследования выступают алгоритмы сглаживания изображений.

Предмет исследования – анализ их эффективности, качества и скорости работы.

Целью курсовой работы является проведение сравнительного анализа основных алгоритмов сглаживания изображений, использующих принцип усреднения точек.

Для достижения цели исследования были поставлены следующие задачи:

- рассмотреть описание каждого из выбранных алгоритмов;

- понять сильные и слабые стороны каждого из алгоритмов;

- реализовать рассматриваемые алгоритмы сглаживания на Python;

- провести тестирование на заранее подготовленном наборе изображений;

- проанализировать полученные результаты согласно описанным выше параметрам сравнения.

**1. Обзор выбранных алгоритмов сжатия**

**1.1. Метод среднего сглаживания**

Выход сглаженного линейного пространственного фильтра представляет собой простое среднее значение пикселей, содержащихся в окрестности шаблона фильтра, который является средним фильтром. Средний фильтр также является фильтром нижних частот.

Принцип работы метода среднего сглаживания следующий:

1. Определяется размер окна, используемого для сглаживания пикселей.

2. Для каждого пикселя выделяется окно указанного размера вокруг него.

3. Вычисляется среднее значение всех пикселей в этом окне.

4. Центральный пиксель заменяется на полученное среднее значение.

5. Процесс повторяется для каждого пикселя изображения.

Этот метод позволяет уменьшить шум на изображении за счет сглаживания значений пикселей в окне. Однако, он также может привести к потере деталей и резких границ на изображении, особенно если окно выбрано слишком большим [4].

Шаблон среднего фильтра делится на стандартное пиксельное среднее и средневзвешенное. Как показано на рисунке 1.

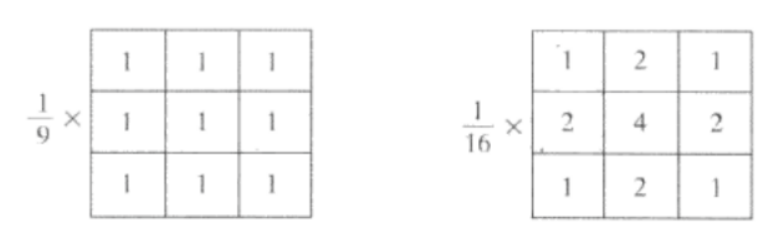


Рисунок 1. Вычисление средневзвешенного среднего [5]

Вычислительная сложность алгоритма среднего сглаживания зависит от размера окна, используемого для сглаживания, и размеров самого изображения.  
Предположим, что размер окна для сглаживания составляет n x n, а размер изображения составляет M x N пикселей.

В этом случае, вычислительная сложность алгоритма среднего сглаживания составляет O(MNn^2), так как для каждого пикселя изображения нам нужно вычислить среднее значение в окне n x n. Следовательно, время выполнения алгоритма будет пропорционально количеству пикселей в изображении и квадрату размера окна.

**1.2. Билатеральная фильтрация**

Билатеральный фильтр - это метод сглаживания изображений, который учитывает как пространственное, так и яркостное распределение пикселей. Этот фильтр использует две функции - одну для оценки различий в интенсивности пикселей и другую для оценки пространственной близости пикселей.  
Принцип работы билатерального фильтра следующий:

1. Для каждого пикселя изображения определяется окно заданного размера вокруг него.

2. Для каждого пикселя в этом окне вычисляется вес, учитывающий разницу в интенсивности пикселей и пространственное расположение пикселей.

3. Полученные веса используются для вычисления взвешенного среднего значения интенсивностей пикселей в окне. Это поведение аналогично многим другим фильтрам - например, фильтру Гаусса.

4. Центральный пиксель заменяется на взвешенное среднее значение.

5. Процесс повторяется для каждого пикселя изображения.

Двунаправленная фильтрация изображения f(x) в общем виде описывается выражением [6]:

Для лучшего понимания билатеральной фильтрации также необходимо

разобраться с тем как работает гауссова функция, т.к именно она скрывается за обозначением .

Гауссова функция - это функция, имеющая симметричную форму "колоколообразной кривой". У нее есть параметр σ, который управляет шириной "колокола". Демонстрация влияния этого параметра показана на рисунке 2.

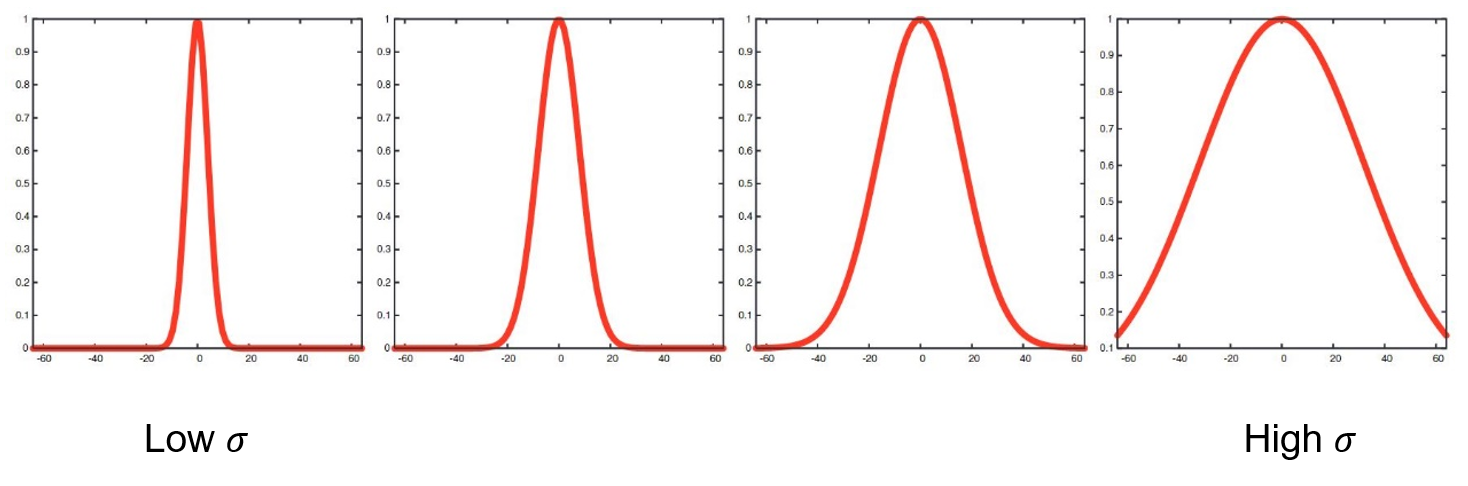


Рисунок 2. Вычисление средневзвешенного среднего

Гауссова функция описывается формулой .

Более подробно о каждом компоненте формул.

– отвечает за сглаживающий эффект

Это выражение является частью веса. Мы можем видеть, что большее расстояние между p и q приведет к меньшим значениям G\_σ (x), а меньшее расстояние между p и q приведет к более высоким значениям G\_σ (x). Это приводит к тому, что ближние пиксели "весят" больше, чем дальние.

Сглаживание означает, что для некоторого пикселя p более близкие пиксели будут иметь более близкие значения. В этой формуле, чем ближе находится соседний пиксель, тем выше будет его вес, поэтому новое значение пикселя будет больше похоже на его соседей.

Параметр σ\_s отвечает за определение расстояния между пикселями, которые все еще считаются "связанными" друг с другом.Более высокие значения σ\_s приведут к более плавному изображению.

– отвечает за эффект сохранения краев

g

Здесь, чем ближе их значения, тем выше будет вес. Другими словами, если разница в значениях пикселей велика (они менее похожи), они, вероятно, находятся на краю изображения и, следовательно, будут иметь меньший вес. если разница в их значениях невелика (пиксели похожи), то, вероятно, они не находятся на каком-либо ребре и получат более высокий вес.

Параметр σ\_r отвечает за сохранение ребер.Чем выше значение σ\_r, тем в меньшей степени фильтр сохранит края изображения.

Таким образом, произведение описывает вес каждого соседнего пикселя, первая часть делает изображение более плавным, в то время как вторая часть отменяет его, если между этими пикселями есть граница [7].

Таким образом, изображение станет более плавным на участках, которые не находятся вблизи краев (сглаживание), и не станет более плавным на участках, которые находятся вблизи края (сохранение края).

Этот алгоритм, описанный выше, является наивной версией билатеральной фильтрации, и его временная сложность равна , где S - количество пикселей в изображении.

**1.3. Метод медианного сглаживания**

Метод медианного сглаживания изображений используется для уменьшения шумов на изображении путем замены каждого пикселя на медианное значение окружающих его пикселей.

Принцип работы этого метода следующий:

1. Для каждого пикселя изображения выделяется окно заданного размера вокруг него.

2. Все интенсивности пикселей в этом окне сортируются по возрастанию.

3. Значение центрального пикселя заменяется на медианное значение интенсивностей сортированного окна.

4. Процесс повторяется для каждого пикселя изображения.

На рисунке 3 представлен пример работы алгоритма для окна 3x3.

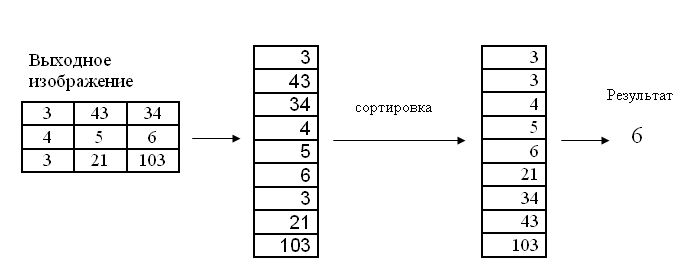


Рисунок 3. Пример работы медианного сглаживания

Медианный фильтр хорошо удаляет шумы на изображениях, поскольку медианное значение более устойчиво к выбросам, чем среднее значение. Он также сохраняет резкие границы и детали на изображении. Тем не менее, он может быть более ресурсоемким, чем другие методы сглаживания, особенно на больших изображениях или при использовании больших окон для фильтрации [8].

Сложность алгоритма медианного фильтра зависит от размера окна, используемого для обработки, и размеров самого изображения.

Допустим, размер окна для сглаживания составляет n x n пикселей, а размер изображения составляет M x N пикселей.

Тогда вычислительная сложность алгоритма медианного фильтра может быть приближенно оценена как O(MNn^2log(n^2)), так как для каждого пикселя изображения нам нужно отсортировать окно n x n, содержащее n^2 элементов, что требует O(n^2log(n^2)) операций. Предполагается, что для этого будет использована быстрая сортировка.

**1.4. Метод минимума и максимума**

Метод минимума и максимума - это методы нелинейного сглаживания изображений.

Принцип работы метода минимума заключается в том, что для каждого пикселя изображения определяется окно заданного размера вокруг него, а затем из этого окна выбирается пиксель с минимальной интенсивностью, который затем становится новым значением центрального пикселя. Этот метод применяется для удаления шумов с изображения и представляет собой простую форму медианного фильтра.

Принцип работы метода максимума аналогичен, но вместо выбора пикселя с минимальной интенсивностью, выбирается пиксель с максимальной интенсивностью в окне, и это значение затем назначается центральному пикселю.

Эти методы хорошо подходят для удаления шумов на изображениях и позволяют сохранить резкие границы, так как они не сглаживают границы и детали изображения.

В своем эксперименте я буду использовать алгоритм, сочетающий в себе два этих метода сразу. В первом проходе по изображению будет применяться метод минимума. Во втором проходе по изображению будет применятся метод максимума к матрице, полученной в результате первого прохода [9].

Аналогично методу среднего сглаживания вычислительная сложность алгоритма минимума-максимума зависит от размера окна, используемого для сглаживания, и размеров самого изображения. Предположим, что размер окна для сглаживания составляет n x n, а размер изображения составляет M x N пикселей.

В этом случае, вычислительная сложность алгоритма среднего сглаживания составляет O(MNn^2), так как для каждого пикселя изображения нам нужно вычислить среднее значение в окне n x n.

**2. Особенности реализации выбранных алгоритмов на языке Python**

**2.1. Стек технологий**

Рассмотренные в предыдущей главе методы, представляют собой итерационные алгоритмы, которые проходят по изображению в определённом порядке, обрабатывая значения пикселей. Из сторонних библиотек использовалась библиотека numpy для упрощенной и эффективной работы с матрицами (которые являются изображениями, представленными в памяти компьютера) [10], а также библиотека cv2 для считывания изображений для тестирования [11]. Для метрики SSIM был использован готовый библиотечный вариант из библиотеки skimage.

**2.2 Описание работы программы**

Помимо реализации всех описанных в предыдущей главе алгоритмов (см. описание работы алгоритмов в 1 главе) было также написано консольное приложение, чтобы удобно использовать и тестировать написанные алгоритмы, которое имеет 2 режима работы: тестовый (-m test) и обычый (-m prod).

В тестовым режиме приложение вызывает тестовый скрипт. Который проходит по каждому изображению из тестового датасета и по очереди применяет к нему все рассматриваемые в рамках курсовой работы алгоритмы сглаживания, замеряет время работы каждого алгоритма на конкретном изображении, выводит результат работы алгоритма отдельным файлом в выходную директорию. В конце прогона всех алгоритмов формирует общий файл для наглядного сравнения результатов и выводит на экран время работы. После обработки всех изображений тестового датасета вызывается скрипт, который подсчитывает метрику SSIM для каждого изображения по принципу: SSIM (исходное изображение, результат работы конкретного алгоритма).

В обычном режиме приложение можно передать путь до исходного изображения (не обязательно из тестового датасета), фильтр, который хочется использовать, гиперпараметры фильтра и выходной файл. Приложение вызовет функцию, реализующую конкретный выбранный фильтр с заданными параметрами и вернёт результат работы. Этот режим нужен для точечного запуска приложения. Все флаги приложения представлены ниже в листинге 1.

Листинг 1 – Описание аргументов командной строки приложения

pars.add\_argument('-m', type=str, default='test1', help="program mode")  
pars.add\_argument('-f', type=str, default='median', help="filters")  
pars.add\_argument('-ks', type=int, default=5, help="kernel size")  
pars.add\_argument('-a', type=int, default=0.6, help="alpha (0,1)")  
pars.add\_argument('-inp', type=str, default='./dataset/3.png', help="path to input image")  
pars.add\_argument('-out', type=str, default='./out/3\_done.png', help="path to output image")

Так как хотелось уметь работать не только с черно-белыми, но и с цветными изображениями, то для каждого изображения очевидным решением стал прогон алгоритма на каждом из каналов RGB, это несколько замедлило работу скрипта, однако, от этого можно отказаться, если нет необходимости сохранять цвет изображения.

Реализация всех алгоритмов, тестирующего скрипта и самого приложения будет доступна для самостоятельного изучения в приложении к данной записке.

**3. Сравнительный анализ резальтатов работы алгоритмов**

**3.1. Описания параметров сравнения**

В данной главе будут рассмотрены результаты работы разных фильтров на различных типах изображений.

Для проведения эксперимента я подобрал изображения из следующих категорий: 1) геометрические фигуры с однотонным фоном и ребристыми углами, 2) изображения с сильным шумом при хорошем и плохом освещении 3) изображения документов 4) абсолютно нормальное изображение в хорошем качестве.

Как было сказано во введении, сравнение будет проходить по следующим параметрам: cредняя скорость работы алгоритма (на одинаковых входных данных при различных изображениях), метод структурного подобия (SSIM), устранение шумов (алгоритм сглаживания должен устранять шумы на изображении), устранение ступенчатости контуров (алгоритм должен сглаживать ступенчатые контуры и приводить их к более гладкому виду), заполнение деталей (оценка способности алгоритма сохранять или восстанавливать детали на изображении; желательно, чтобы сглаживание не приводило к потере информации или размытию важных элементов изображения).

Средняя скорость работы вычислялась после прогона всех алгоритмов на тестовом датасете, чтобы учесть все случаи.

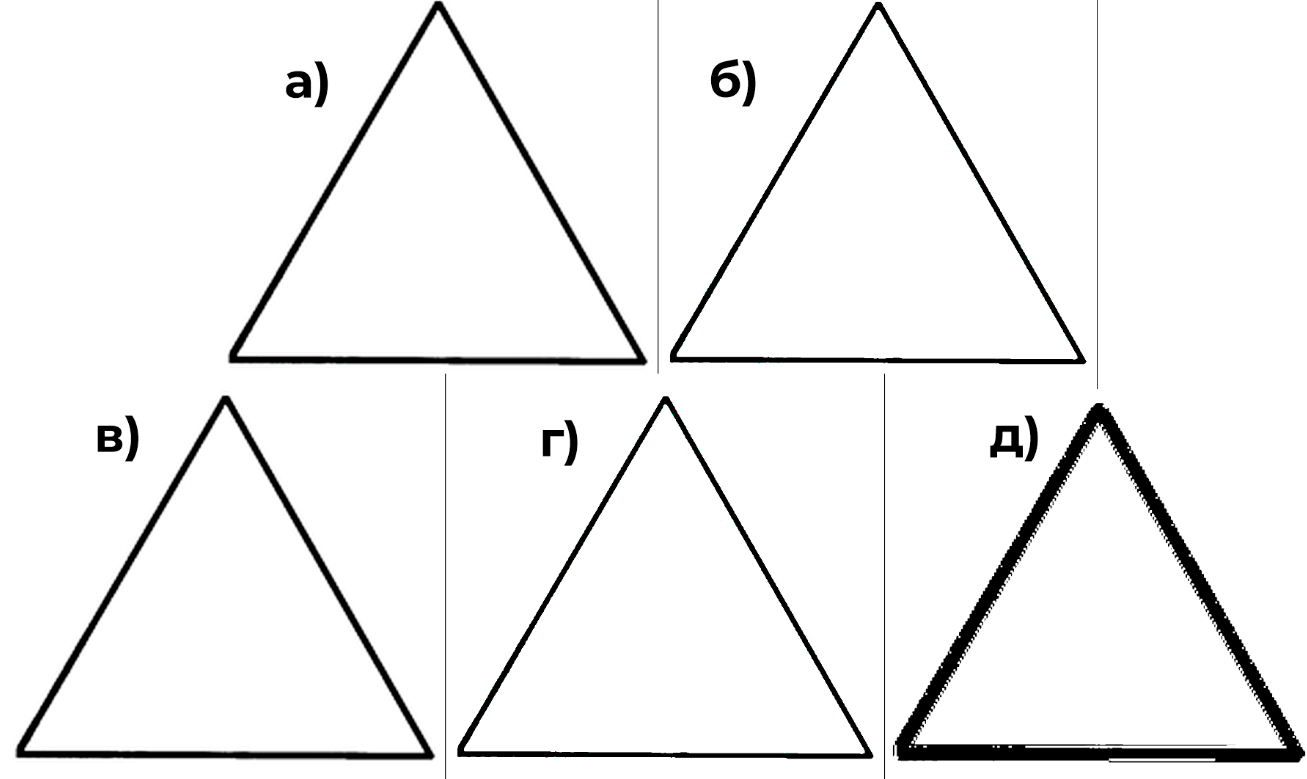
В качестве количественной оценки качества изображений был выбран метод структурного правдоподобия (SSIM). SSIM коррелирует с качеством и восприятием зрительной системы человека (цветовая модель HVS). Вместо использования традиционных методов суммирования ошибок SSIM моделирует искажение изображения как комбинацию трех факторов: потеря корреляции, искажение яркости и искажение контрастности. Чем выше параметр SSIM, тем лучше полезная сохранена информация относительно исходного изображения, и тем выше качество полученного.

Для первичного анализа я поместил алгоритмы в равные условия. Были заданы одинаковые параметры (все алгоритмы используется «скользящее окно» размером 5 на 5, однако не все алгоритмы показали себя хорошо при такой настройке, их я протестирую отдельно, чтобы понять, какая конфигурация для них будет являться оптимальной).

Также стоит отметить, что алгоритм min-max сглаживания проявил себя очень плохо во всех случаях, поэтому он рассматривать не будет, дабы не портить общую картину. Тут сразу можно сделать вывод, что идея min-max является не очень удачной, если речь идёт о сглаживании изображений.

**3.2. Работа алгоритмов на изображениях с геометрическими фигурами**

Для начала рассмотрим, как различные алгоритмы работают с геометрическими фигурами (рисунок 4). Это интересный пример, который не позволяет отвлекаться на фон и сконцентрироваться на контурах объектов, чтобы как раз сравнивать алгоритмы по этому параметру. Для этой задачи были выбраны 2 разные геометрические формы: треугольник и круг. При этом также были взяты разные цвета изображений, чтобы проверить то, как алгоритмы сохраняют цвета исходных изображений.



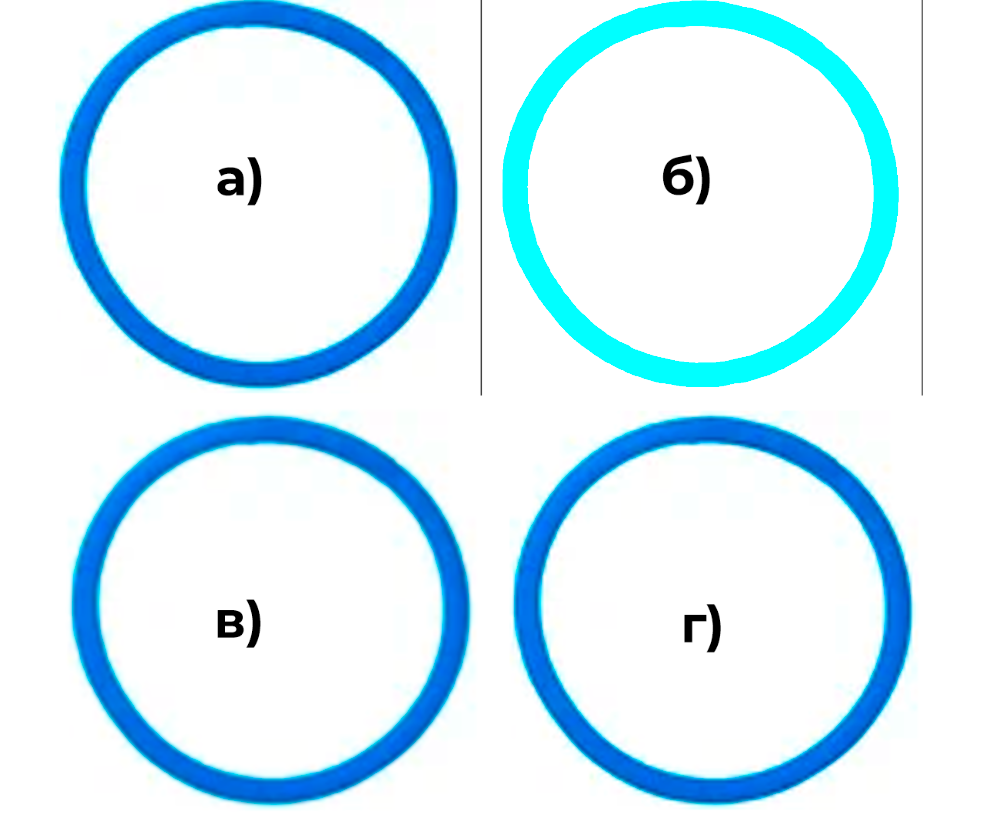


Рисунок 4. Результаты работы алгоритмов на геометрических фигурах a) исходное изображение б) mean filter в) bilateral filter г) median filter д) min-max filter

На первый взгляд все алгоритмы справляются неплохо. Однако, если рассматривать изображения более детально, то можно заметить, что mean filter обеспечивает самые чёткие границы изображения, но при этом у него существуют небольшие выбросы, которые приводят к неровностям. Плюсом к этому данный метод слишком сильно искажает цветовую передачу изображения. Поэтому в данной задаче справляется лучшего всего median filter, т.к в отличие от билатерального, он не так сильно размывает края, сохраняя их более точными. Более детально это можно увидеть на рисунке 5.





Рисунок 5. Увеличенная версия результатов работы фильтров (верхнее лево изображение-исходное)

Говоря об устранении шумов, mean фильтр справился лучше, но сохранение цвета достаточно критично в вопросе сглаживания изображений, поэтому по этому параметру опять же побеждает медианный фильтр, т.к он сохраняет цвет почти без искажения.

**3.3. Работа алгоритмов на изображениях с шумами**

Далее была рассмотрена уже реальная фотография, сделанная при плохом освещении с явными шумами. При работе с ней наглядно можно отследить, насколько хорошо тот или иной фильтр справляется с удалением шумов. Результаты прогона алгоритмов представлены на рисунке 6.

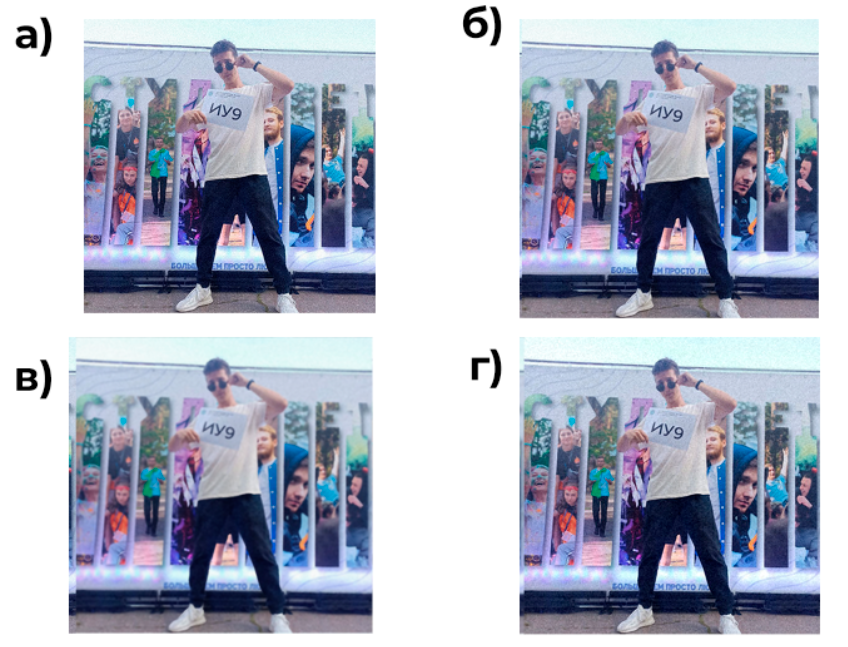


Рисунок 6. Результаты работы алгоритмов на зашумленных изображениях

a) исходное изображение б) mean filter в) bilateral filter г) median filter

Отсюда наглядно видно, что билатеральный фильтр очень хорошо сглаживает изображение, однако, при этом появляется сильный эффект размытия, что отрицательно сказывается на восприятии изображения. Лучшими здесь оказываются медианный и средний фильтр, они сохраняет четкие контуры объектов и при этом видно, что шумы стали меньше. При этом прогнав это изображение на различных параметрах медианного фильтра, можно также получить очень размытое изображение, поэтому важно подбирать оптимальные значения при работе с ним индивидуально под каждое изображение. Разницы между медианным и средним фильтром практически нет. Если говорить про метрику SSIM, то здесь мы видим следующие результаты: median filter – 0.38; bilateral filter – 0.31; mean filter – 0.42.

**3.4. Работа алгоритмов на изображениях с документами**

Следующим тестовым образцом является изображения документов, при передаче в алгоритм которого мною ожидалось то, что алгоритмы постараются восстановить недостающие детали, и на выходе я получу более качественное изображение, однако, это не так. В чём можно убедиться на рисунке 7.



Рисунок 7. Результаты работы алгоритмов на изображениях с документами

1. исходное изображение б) mean filter в) bilateral filter г) median filter

В данном случае видно, что качество документа стало чуть лучше, чем было, но это очень тяжело распознать на глаз. Билатеральный фильтр слишком сильно размыл фотографию, но в остальном тоже справился неплохо. Если говорить про метрику SSIM, то здесь мы видим следующие результаты: median filter – 0.97; bilateral filter – 0.84; mean filter – 0.92.

**3.5. Работа алгоритмов на изображениях, не требующих сглаживания**

В качестве последнего тестового изображения я выбрал абсолютно корректное качественное изображений, чтобы посмотреть, что фильтры не будут ухудшать уже заведомо хорошие изображения. Результаты можно видеть на рисунке 8.

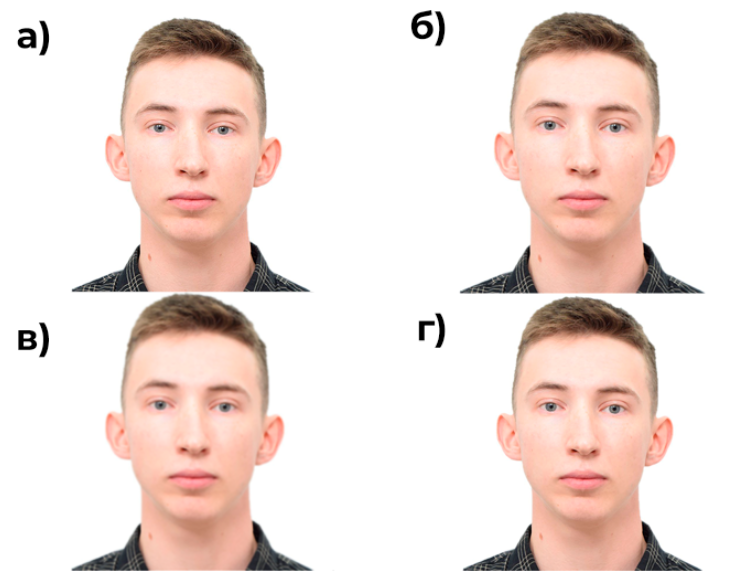


Рисунок 8. Результаты работы алгоритмов на хороших изображениях

1. исходное изображение б) mean filter в) bilateral filter г) median filter

Все фильтры показали себя хорошо, сохраняя исходное высокое качество изображения, но билатеральный опять сильно размыл картинку. Если говорить о показателях SSIM, то mean и median фильтры показали результат похожести с оригиналом 0.96, bilateral, очевидно, чуть хуже – 0.92.

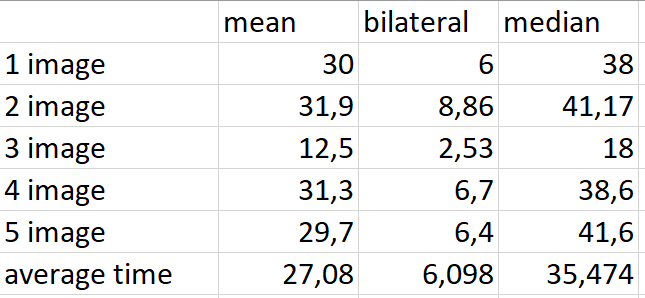
**3.6. Общие результаты**

После рассмотрения работы алгоритмов на всех типах изображений, можно сравнить их среднее время работы на изображениях из датасета. Для этого обратимся к таблице 1.

Стоит отметить, что тестирование производилось на машине с характеристиками:

* ОС: Windows 10
* ЦП: 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz 3.11 GHz
* Оперативная память: 16,0 ГБ

**Таблица 1 - Время работы алгоритмов**



Из таблицы видно, что медианный фильтр в среднем работает дольше, чем все остальные. Это вполне ожидаемый результат, т.к данный алгоритм имеет большую сложность. Менее очевидным стало то, что билатеральный фильтр отрабатывает очень быстро, относительно своих конкурентов. Mean фильтр находится примерно между ними по скорости.

**Таблица 2 – Финальная таблица сравнения**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование  алгоритма | Скорость  работы, с. | Среднее значение  SSIM | Устранение  шумов | Удаление  ступенчатости | Сохранение  деталей |
| Mean filter | 27 | 0.83 | Хорошо устраняет шумы | Заметно удаляет ступенчатость объектов | Некоторые детали искажены, проблемы с цветопередачей |
| Bilateral filter | 6 | 0.74 | Хорошо устраняет шумы, но сильно размывает изображение | Заметно удаляет ступенчатость объектов | Плохо сохраняет детали, сильное размытие |
| Median filter | 35 | 0.8 | Хорошо устраняет шумы | Заметно удаляет ступенчатость объектов | Хорошо сохраняет детали, сильное размытие только при большом размере ядра |

# ЗАКЛЮЧЕНИЕ

В результате курсовой работы поставленная задачи была выполнена, и было проведено сравнение различных алгоритмов сглаживания изображений. В процессе выполнения работы были исследованы технологии сглаживания изображений и устранения шумов, а также библиотека numpy. Удалось реализовать все рассматриваемые алгоритмы, в некоторых случая пришлось попробовать несколько различных реализаций, чтобы понять, какая из них лучше. Был получен опыт написания программного кода на Python, а также опыт работы с данными в формате растровых изображений.

В конечном итоге, можно сказать, что из рассматриваемых алгоритмов лучше всего показал себя медианный фильтр.

В дальнейшем можно попробовать сравнить медианный фильтр с другими алгоритмами сглаживания, так как их сейчас невероятное множество, чтобы выявить какой алгоритм сжатия будет наиболее качественно и эффективно справляться со своей задачей.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Виды сглаживания и их особенности - i2HARD – URL: https://i2hard.ru/publications/27401/ (дата обращения: 18.11.2023).
2. Нелокальный алгоритм для сглаживания изображений – URL: https://habr.com/ru/articles/273159/ (дата обращения: 18.11.2023).
3. Mansouryar M., Hedayati A. Smoothing Via Iterative Averaging (SIA) A Basic Technique for Line Smoothing // International Journal of Computer and Electrical Engineering, 2012. Vol. 4, No. 3. P. 307-311 (дата обращения: 19.11.2023).
4. Mean filter, or average filter – URL: http://www.librow.com/articles/article-5 (дата обращения: 20.11.2023).
5. Принцип и реализация среднего фильтра – URL: https://russianblogs.com/article/114728488/ (дата обращения: 20.11.2023).
6. Yang Q., Tan K.-H., Ahuja N. Real-time O(1) bilateral filtering // In Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, (CVPR '09), June 2009, pp. 557–564 (дата обращения: 21.11.2023).
7. Bilateral Filter for Gray and Color Images (Tomasi and Manduchi) – URL: https://users.soe.ucsc.edu/~manduchi/Papers/ICCV98.pdf (дата обращения: 21.11.2023).
8. Матричные фильтры обработки изображений – URL: https://habr.com/ru/articles/142818/(дата обращения: 22.11.2023).
9. Нелинейная фильтрация изображений   - URL: https://studbooks.net/2332535/tehnika/nelineynaya\_filtratsiya\_izobrazheniy (дата обращения: 22.11.2023).
10. Библиотека numpy python – URL: https://numpy.org/ (дата обращения: 22.11.2023).
11. Библиотека cv2 python – URL: https://gamedevacademy.org/cv2-python-tutorial-complete-guide/ (дата обращения: 22.11.2023).
12. Методы количественных оценок качества изображений – URL: https://skine.ru/articles/258694/ (дата обращения: 10.12.2023).

# ПРИЛОЖЕНИЕ А

Листинг 2 − Mean Filter

import cv2  
import numpy as np  
  
  
def get\_kernel():  
 return np.ones((3, 3), np.float32) / 9  
  
  
def get\_mean\_with\_kernel(filter\_area, kernel):  
 return np.sum(np.multiply(kernel, filter\_area))  
  
  
def mean\_filter(image):  
 kernel = get\_kernel()  
 height, width = image.shape[:2]  
 image = cv2.copyMakeBorder(image, 1, 1, 1, 1, cv2.BORDER\_REFLECT)  
  
 for row in range(1, height + 1):  
 for column in range(1, width + 1):  
 filter\_area = image[row - 1:row + 2, column - 1:column + 2]  
 res = get\_mean\_with\_kernel(filter\_area, kernel)  
 image[row][column] = res  
  
 return image

Листинг 3 –Median Filter

import numpy as np  
  
  
def median\_filter(data, filter\_size):  
 temp = []  
 indexer = filter\_size // 2  
 data\_final = np.zeros((len(data),len(data[0])))  
 for i in range(len(data)):  
  
 for j in range(len(data[0])):  
 if j + z - indexer < 0 or j + indexer > len(data[0]) - 1:  
 temp.append(0)  
 else:  
 for k in range(filter\_size):  
 temp.append(data[i + z - indexer][j + k - indexer])  
  
 temp.sort()  
 data\_final[i][j] = temp[len(temp) // 2]  
 temp = []  
 return data\_final

Листинг 4 – Bilateral Filter

import numpy as np

def gaussian(x\_square, sigma):  
 return np.exp(-0.5\*x\_square/sigma\*\*2)

def bilateral\_filter(image, sigma\_space, sigma\_intensity):  
 kernel\_size = int(2\*sigma\_space+1)  
 half\_kernel\_size = int(kernel\_size / 2)  
 result = np.zeros(image.shape)  
 W = 0  
 for x in range(-half\_kernel\_size, half\_kernel\_size+1):  
 for y in range(-half\_kernel\_size, half\_kernel\_size+1):  
 Gspace = gaussian(x \*\* 2 + y \*\* 2, sigma\_space)  
 shifted\_image = np.roll(image, [x, y], [1, 0])  
 intensity\_difference\_image = image - shifted\_image  
 Gintenisity = gaussian(  
 intensity\_difference\_image \*\* 2, sigma\_intensity)  
 result += Gspace\*Gintenisity\*shifted\_image  
 W += Gspace\*Gintenisity  
  
 return result / W

for z in range(filter\_size):  
 if i + z - indexer < 0 or i + z - indexer > len(data) - 1:  
 for c in range(filter\_size):  
 temp.append(0)  
 else:  
 if j + z - indexer < 0 or j + indexer > len(data[0]) - 1:  
 temp.append(0)  
 else:  
 for k in range(filter\_size):  
 temp.append(data[i + z - indexer][j + k - indexer])  
  
 temp.sort()  
 data\_final[i][j] = temp[len(temp) // 2]  
 temp = []  
return data\_final

Листинг 5 – Min-Max Filter

import numpy as np  
  
  
def min\_max\_filter(img, size):  
 m, n = img.shape  
  
 img\_new = np.zeros([m, n])  
 len = int((size - 1) / 2)  
 temp = list()  
 for i in range(len, m - len):  
 for j in range(len, n - len):  
 for k in range(i - len, i + len + 1):  
 for w in range(j - len, j + len + 1):  
 temp.append(img[k, w])  
  
 value = min(temp)  
 img\_new[i, j] = value  
 temp.clear()  
  
 img\_new1 = np.zeros([m, n])  
 for i in range(len, m - len):  
 for j in range(len, n - len):  
 for k in range(i - len, i + len + 1):  
 for w in range(j - len, j + len + 1):  
 temp.append(img\_new[k, w])  
  
 value = max(temp)  
 img\_new1[i, j] = value  
 temp.clear()  
  
 img\_new1 = img\_new1.astype(np.uint8)  
  
 return img\_new1

Gintenisity = gaussian(  
 intensity\_difference\_image \*\* 2, sigma\_intensity)  
 result += Gspace\*Gintenisity\*shifted\_image  
 W += Gspace\*Gintenisity  
  
return result / W

Листинг 6 – Тестирующий модуль (для тестового датасета)

import os  
import time  
  
import cv2  
import numpy as np  
  
from bilateral\_filter import bilateral\_filter  
from mean\_filter import mean\_filter  
from median\_filter import median\_filter  
from min\_max\_filter import min\_max\_filter  
  
  
def mean\_filter\_RGB(input\_image):  
 start = time.time()  
 R\_mf = mean\_filter(input\_image[:, :, 0])  
 G\_mf = mean\_filter(input\_image[:, :, 1])  
 B\_mf = mean\_filter(input\_image[:, :, 2])  
 output\_image\_00 = np.stack([R\_mf, G\_mf, B\_mf], axis=2)[:1080,:1080]  
 mf\_time = time.time() - start  
 return output\_image\_00, mf\_time  
  
  
def bilateral\_filter\_RGB(input\_image):  
 start = time.time()  
 R\_bf = bilateral\_filter(input\_image[:, :, 0], 5, 0.6)  
 G\_bf = bilateral\_filter(input\_image[:, :, 1], 5, 0.6)  
 B\_bf = bilateral\_filter(input\_image[:, :, 2], 5, 0.6)  
 output\_image\_01 = np.stack([R\_bf, G\_bf, B\_bf], axis=2)  
 bf\_time = time.time() - start  
 return output\_image\_01, bf\_time  
  
  
def median\_filter\_RGB(input\_image):  
 start = time.time()  
 R\_medf = median\_filter(input\_image[:, :, 0], 5)  
 G\_medf = median\_filter(input\_image[:, :, 1], 5)  
 B\_medf = median\_filter(input\_image[:, :, 2], 5)  
 output\_image\_02 = np.stack([R\_medf, G\_medf, B\_medf], axis=2)  
 medf\_time = time.time() - start  
 return output\_image\_02, medf\_time

output\_image\_02 = np.stack([R\_medf, G\_medf, B\_medf], axis=2)  
 medf\_time = time.time() - start  
 return output\_image\_02, medf\_time  
  
  
def min\_max\_filter\_RGB(input\_image):  
 start = time.time()  
 R\_mmf = min\_max\_filter(input\_image[:, :, 0], 5)  
 G\_mmf = min\_max\_filter(input\_image[:, :, 1], 5)  
 B\_mmf = min\_max\_filter(input\_image[:, :, 2], 5)  
 output\_image\_03 = np.stack([R\_mmf, G\_mmf, B\_mmf], axis=2)  
 mm\_time = time.time() - start  
 return output\_image\_03, mm\_time  
  
def test\_algo():  
 for i in range(1, 8):  
 if not os.path.isdir(f'./output/{i}'):  
 os.mkdir(f'./output/{i}')  
  
 print(f'-----------------STARTING PROCESSING IMAGE {i}------------------')  
 image\_name = f'./dataset/{i}.png'  
 input\_image = cv2.imread(image\_name,  
 cv2.IMREAD\_UNCHANGED).astype(np.float32) / 255.0  
  
 *# mean filter* print("Starting mean filter...")  
 output\_image\_00, mf\_time = mean\_filter\_RGB(input\_image)  
 cv2.imwrite(f'./output/{i}/mean.png', output\_image\_00 \* 255)  
 print("Mean filter end")  
  
 *# bilateral* print("Starting bilateral filter...")  
 output\_image\_01, bf\_time = bilateral\_filter\_RGB(input\_image)  
 cv2.imwrite(f'./output/{i}/bilateral.png', output\_image\_01 \* 255)  
 print("Bilateral filter end")

Листинг 7 – Скрипт по подсчету SSIM

*# median*print("Starting median filter...")  
output\_image\_02, medf\_time = median\_filter\_RGB(input\_image)  
cv2.imwrite(f'./output/{i}/median.png', output\_image\_02 \* 255)  
print("Median filter end")  
  
*# # min-max*print("Starting min-max filter...")  
output\_image\_03, mm\_time = min\_max\_filter\_RGB(input\_image)  
cv2.imwrite(f'./output/{i}/min\_max.png', output\_image\_03 \* 255)  
print("Min-max filter end")  
  
print(f'\n--------------------TIME-----------------------\n')  
print(f'{mf\_time}\t{bf\_time}\t{medf\_time}\t{mm\_time}')  
  
input\_image = np.stack([input\_image[:,:,0],input\_image[:,:,1],input\_image[:,:,2]],axis=2)  
  
Row1 = np.hstack([input\_image,output\_image\_00, output\_image\_01,  
 output\_image\_02,output\_image\_03])  
cv2.imwrite(f'./output/{i}\_all.png', Row1\*255)

from skimage.metrics import structural\_similarity  
import cv2  
  
  
def count\_SSMI():  
 filters = ["median", "bilateral", "mean"]  
 for i in range(1, 8):  
 for filter in filters:  
 before = cv2.imread(f'./dataset/{i}.png')  
 after = cv2.imread(f'./output/{i}/{filter}.png')  
  
 before\_gray = cv2.cvtColor(before, cv2.COLOR\_BGR2GRAY)  
 after\_gray = cv2.cvtColor(after, cv2.COLOR\_BGR2GRAY)  
  
 (score, diff) = structural\_similarity(before\_gray, after\_gray, full=True)  
 print(f'image similarity source with {filter}', score)

Листинг 8 – Приложение

import argparse  
  
import cv2  
import numpy as np  
  
import testing  
from ssim import count\_SSMI  
  
  
def create\_parser():  
 pars = argparse.ArgumentParser()  
 pars.add\_argument('-m', type=str, default='test1', help="program mode")  
 pars.add\_argument('-f', type=str, default='median', help="filters")  
 pars.add\_argument('-ks', type=int, default=5, help="kernel size")  
 pars.add\_argument('-a', type=int, default=0.6, help="alpha (0,1)")  
 pars.add\_argument('-inp', type=str, default='./dataset/3.png', help="path to input image")  
 pars.add\_argument('-out', type=str, default='./out/3\_done.png', help="path to output image")  
  
 return pars  
  
  
def main():  
 parser = create\_parser()  
 console\_args = parser.parse\_args()  
 if console\_args.m == "test":  
 testing.test\_algo()  
 count\_SSMI()  
 elif console\_args.m == "prod":  
 input\_image = cv2.imread(console\_args.inp, cv2.IMREAD\_UNCHANGED).astype(np.float32) / 255.0  
 filters = console\_args.f.split()  
 for filter in filters:  
 if filter == "median":  
 output\_image, \_ = testing.median\_filter\_RGB(input\_image, console\_args.ks)  
 cv2.imwrite(console\_args.out, output\_image \* 255)  
 elif filter == "mean":  
 output\_image, \_ = testing.mean\_filter\_RGB(input\_image)  
 cv2.imwrite(console\_args.out, output\_image \* 255)  
 elif filter == "bilateral":  
 output\_image, \_ = testing.bilateral\_filter\_RGB(input\_image, console\_args.ks, console\_args.alpha)  
 cv2.imwrite(console\_args.out, output\_image \* 255)  
 else:  
 print("cant use this filter")  
 else:  
 print("unknown mode")  
  
  
main()

after\_gray = cv2.cvtColor(after, cv2.COLOR\_BGR2GRAY)  
  
 (score, diff) = structural\_similarity(before\_gray, after\_gray, full=True)  
 print(f'image similarity source with {filter}', score)

elif console\_args.m == "prod":  
 input\_image = cv2.imread(console\_args.inp, cv2.IMREAD\_UNCHANGED).astype(np.float32) / 255.0  
 filters = console\_args.f.split()  
 for filter in filters:  
 if filter == "median":  
 output\_image, \_ = testing.median\_filter\_RGB(input\_image, console\_args.ks)  
 cv2.imwrite(console\_args.out, output\_image \* 255)  
 elif filter == "mean":  
 output\_image, \_ = testing.mean\_filter\_RGB(input\_image)  
 cv2.imwrite(console\_args.out, output\_image \* 255)  
 elif filter == "bilateral":  
 output\_image, \_ = testing.bilateral\_filter\_RGB(input\_image, console\_args.ks, console\_args.alpha)  
 cv2.imwrite(console\_args.out, output\_image \* 255)  
 else:  
 print("cant use this filter")  
 else:  
 print("unknown mode")  
  
  
main()