

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	5
1.1 Методы сжатия изображения	5
1.2 Алгоритмы сингулярного разложения	13
2. ПРАКТИЧЕСКАЯ ЧАСТЬ	18
2.1 Разработка и тестирование приложения	18
ЗАКЛЮЧЕНИЕ	28
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	29
ПРИЛОЖЕНИЯ	31

ВВЕДЕНИЕ

В течение последних десятилетий человечество создало столько же данных, сколько за всю предыдущую историю цивилизации. При этом темпы роста объемов хранимых данных продолжают увеличиваться: их удвоение происходит, по разным оценкам, каждые 4 или 10 лет.

Разумеется, аппаратные возможности хранения и передачи данных растут. Емкость устройств внешней памяти домашних компьютеров измеряется не мегабайтами, как было 20 лет назад, и не гигабайтами, как это было 10 лет назад, а терабайтами.

Скорость обмена с внешними устройствами за этот же период выросла на два-три порядка. Скорость передачи данных по сети Интернет сегодня составляет десятки и сотни мегабит в секунду, тогда как еще 10 лет назад она была на порядок меньше. Однако рост аппаратных возможностей компьютеров и сетей передачи не поспевает за ростом лавины данных. Для борьбы с этой проблемой необходимы алгоритмы и программное обеспечение сжатия (уменьшения объема) сохраняемых данных и распаковки сжатых данных для последующего использования.

Цель курсовой работы — разработать графический интерфейс для сжатия изображений с помощью языка программирования R.

Задачи, решаемые в данной курсовой работе:

- изучение документации и технической литературы по исследуемой теме;
- реализовать алгоритм сжатия изображений методом сингулярного разложения;
- разработка программного кода приложения с использованием языка программирования R;
- тестирование приложения и замеры результатов;
- изучить правила качественного оформления документации.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Методы сжатия изображения

Все существующие алгоритмы можно разделить на два больших класса:

- алгоритмы сжатия без потерь;
- алгоритмы сжатия с потерями.

Когда мы говорим о сжатии без потерь, мы имеем в виду, что существует алгоритм, обратный алгоритму сжатия, позволяющий точно восстановить исходное изображение. Для алгоритмов сжатия с потерями обратного алгоритма не существует. Существует алгоритм, восстанавливающий изображение, не обязательно точно совпадающее с исходным. Алгоритмы сжатия и восстановления подбираются так, чтобы добиться высокой степени сжатия и при этом сохранить визуальное качество изображения. [1.1]

Алгоритм RLE (Run Length Encoding). Все алгоритмы серии RLE основаны на очень простой идее: повторяющиеся группы элементов заменяются на пару (количество повторов, повторяющийся элемент). Хотя этот алгоритм и очень прост, но эффективность его сравнительно низка. Более того, в некоторых случаях применение этого алгоритма приводит не к уменьшению, а к увеличению длины последовательности.

Этот алгоритм наиболее эффективен для чёрно-белых изображений. Также он часто используется, как один из промежуточных этапов сжатия более сложных алгоритмов. [1.2]

LZ77 — один из наиболее простых и известных алгоритмов в семействе LZ. Назван так в честь своих создателей: Авраама Лемпеля (Abraham Lempel) и Якова Зива (Jacob Ziv). Цифры 77 в названии означают 1977 год, в котором была опубликована статья с описанием этого алгоритма.

Основная идея заключается в том, чтобы кодировать одинаковые последовательности элементов. Т.е., если во входных данных какая-то цепочка

элементов встречается более одного раза, то все последующие её вхождения можно заменить «ссылками» на её первый экземпляр.

Как и остальные алгоритмы этого семейства, LZ77 использует словарь, в котором хранятся встречаемые ранее последовательности. Для этого он применяет принцип т.н. «скользящего окна»: области, всегда находящейся перед текущей позицией кодирования, в рамках которой мы можем адресовать ссылки. Это окно и является динамическим словарём для данного алгоритма — каждому элементу в нём соответствует два атрибута: позиция в окне и длина. Хотя в физическом смысле это просто участок памяти, который мы уже закодировали.

Словарные алгоритмы. Идея, лежащая в основе словарных алгоритмов, заключается в том, что происходит кодирование цепочек элементов исходной последовательности. При этом кодировании используется специальный словарь, который получается на основе исходной последовательности.

Существует целое семейство словарных алгоритмов, но мы рассмотрим наиболее распространённый алгоритм LZW.

Словарь в этом алгоритме представляет собой таблицу, которая заполняется цепочками кодирования по мере работы алгоритма. При декодировании сжатого кода словарь восстанавливается автоматически, поэтому нет необходимости передавать словарь вместе со сжатым кодом.

Словарь инициализируется всеми одноэлементными цепочками, т.е. первые строки словаря представляют собой алфавит, в котором мы производим кодирование. При сжатии происходит поиск наиболее длинной цепочки, уже записанной в словарь.

Каждый раз, когда встречается цепочка, ещё не записанная в словарь, она добавляется туда, при этом выводится сжатый код, соответствующий уже записанной в словаре цепочки. В теории на размер словаря не накладывается никаких ограничений, но на практике есть смысл этот размер ограничивать, так как со временем начинают встречаться цепочки, которые больше в тексте не встречаются.

Кроме того, при увеличении размеров таблицы вдвое, мы должны выделять лишние биты для хранения сжатых кодов. Для того чтобы не допускать таких ситуаций, вводится специальный код, символизирующий инициализацию таблицы всеми одноэлементными цепочками. [1.3]

Кодирование Хаффмана — один из наиболее известных методов сжатия данных, который основан на предпосылке, что в избыточной информации некоторые символы используются чаще, чем другие. Как уже упоминалось выше, в русском языке некоторые буквы встречаются с большей вероятностью, чем другие, однако в ASCII-кодах мы используем для представления символов одинаковое количество битов. Логично предположить, что если мы будем использовать меньшее количество битов для часто встречающихся символов и большее для редко встречающихся, то мы сможем сократить избыточность сообщения. Кодирование Хаффмана как раз и основано на связи длины кода символа с вероятностью его появления в тексте.

Алгоритм LZW (Название алгоритм получил по первым буквам фамилий его разработчиков — Lempel, Ziv и Welch), предложенный сравнительно недавно (в 1984 году), запатентован и принадлежит фирме Sperry.

LZW-алгоритм основан на идее расширения алфавита, что позволяет использовать дополнительные символы для представления строк обычных символов. Используя, например, вместо 8-битовых ASCII-кодов 9-битовые, вы получаете дополнительные 256 символов. Работа компрессора сводится к построению таблицы, состоящей из строк и соответствующих им кодов.

Алгоритм сжатия сводится к следующему: программа прочитывает очередной символ и добавляет его к строке. Если строка уже находится в таблице, чтение продолжается, если нет, данная строка добавляется к таблице строк. Чем больше будет повторяющихся строк, тем сильнее будут сжаты данные. [1.4]

Алгоритмы сжатия с потерями. Не смотря на множество весьма эффективных алгоритмов сжатия без потерь, становится очевидно, что эти

алгоритмы не обеспечивают (и не могут обеспечить) достаточной степени сжатия. [1.5]

Идея, лежащая в основе всех алгоритмов сжатия с потерями, довольно проста: на первом этапе удалить несущественную информацию, а на втором этапе к оставшимся данным применить наиболее подходящий алгоритм сжатия без потерь.

Основные сложности заключаются в выделении этой несущественной информации. Подходы здесь существенно различаются в зависимости от типа сжимаемых данных. Для звука чаще всего удаляют частоты, которые человек просто не способен воспринять, уменьшают частоту дискретизации, а также некоторые алгоритмы удаляют тихие звуки, следующие сразу за громкими, для видеоданных кодируют только движущиеся объекты, а незначительные изменения на неподвижных объектах просто отбрасывают.

Рекурсивное сжатие. Этот вид архивации известен довольно давно и напрямую исходит из идеи использования когерентности областей. Ориентирован алгоритм на цветные и черно-белые изображения с плавными переходами. Идеален для картинок типа рентгеновских снимков. Коэффициент сжатия задается и варьируется в пределах 5-20 раз. При попытке задать больший коэффициент, на резких границах, особенно проходящих по диагонали, проявляется «лестничный эффект» — ступеньки разной яркости, размером в несколько пикселей.

JPEG на данный момент один из самых распространенных способов сжатия изображений с потерями. Опишем основные шаги, лежащие в основе этого алгоритма. Будем считать, что на вход алгоритма сжатия поступает изображение с глубиной цвета 24 бита на пиксель (изображение представлено в цветовой модели RGB). [1.6]

В цветовой модели YCbCr мы представляем изображение в виде яркостной компоненты (Y) и двух цветоразностных компонент (Cb,Cr). Человеческий глаз более восприимчив к яркости, а не к цвету, поэтому алгоритм JPEG вносит по возможности минимальные изменения в яркостную

компоненту (Y), а в цветоразностные компоненты могут вноситься значительные изменения.

После перевода в цветовое пространство YCbCr выполняется дискретизация. Возможен один из трёх способов дискретизации:

- 4:4:4 — отсутствует субдискретизация;
- 4:2:2 — компоненты цветности меняются через одну по горизонтали;
- 4:2:0 — компоненты цветности меняются через одну строку по горизонтали, при этом по вертикали они меняются через строку.

При использовании второго или третьего способа мы избавляемся от 1/3 или 1/2 информации соответственно. Очевидно, что чем больше информации мы теряем, тем сильнее будут искажения в итоговом изображении.

Изображение разбивается на компоненты 8*8 пикселей, к каждой компоненте применяются ДКП (Дискретно-косинусное преобразование). Это приводит к уплотнению энергии в коде. Преобразования применяются к компонентам независимо.

Человек практически не способен замечать изменения в высокочастотных составляющих, поэтому коэффициенты, отвечающие за высокие частоты, можно хранить с меньшей точностью. Для этого используется покомпонентное умножение (и округление) матриц, полученных в результате ДКП, на матрицу квантования. На данном этапе тоже можно регулировать степень сжатия.

Далее следует зигзаг-обход матрицы. Зигзаг-обход матрицы — это специальное направление обхода, представленное на Рисунке 1.1.

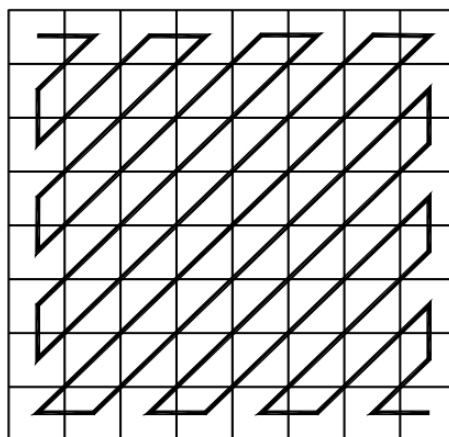


Рисунок 1.1 — Зигзаг-обход матрицы

Получаем одномерный массив с числами. Мы видим, что в нем много нулей, их можно убрать. Для этого вместо последовательности из множества нулей мы вписываем 1 ноль и после него число, обозначающее их количество в последовательности. Таким образом, можно сбросить до 1/3 размера всего массива. А дальше просто сжимает этот массив методом Хаффмана и вписываем уже в сам файл. [1.7]

На Рисунке 1.2 представлен конвейер операций, используемый в алгоритме JPEG.

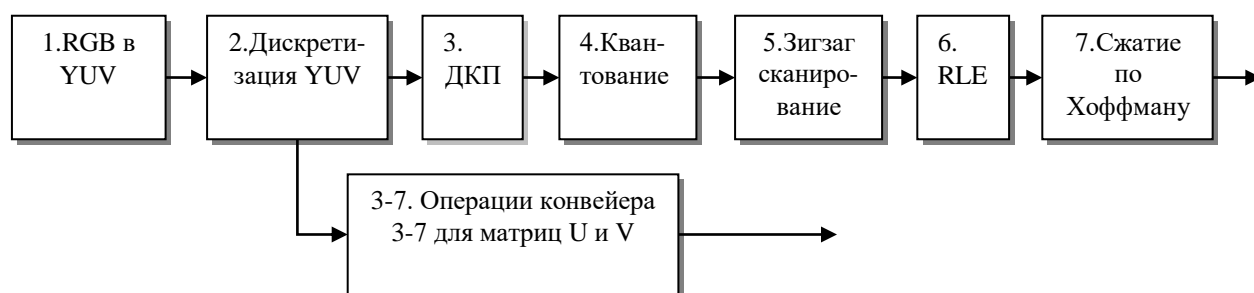


Рисунок 1.2 — Конвейер операций, используемый в алгоритме JPEG

Алгоритм JPEG 2000 разработан той же группой экспертов в области фотографии, что и JPEG. В 1997 стало ясно, что необходим новый, более гибкий и мощный стандарт, который и был доработан к зиме 2000 года. Базовая схема JPEG-2000 очень похожа на базовую схему JPEG. Отличия заключаются в следующем:

Вместо дискретного косинусного преобразования (DCT)

1. Вместо дискретного косинусного преобразования (DCT) используется дискретное вэйвлет-преобразование (DWT).
2. Вместо кодирования по Хаффману используется арифметическое сжатие.
3. В алгоритм изначально заложено управление качеством областей изображения.
4. Не используется уменьшение разрешения цветоразностных компонент U и V.

5. Кодирование с явным заданием требуемого размера наряду с традиционным методом кодирования по качеству.
6. Поддержка сжатия без потерь. Поддержка сжатия однобитных (2-цветных) изображений.
7. На уровне формата поддерживается прозрачность.

На Рисунке 1.3 продемонстрирован конвейер операций, используемый в алгоритме JPEG-2000.

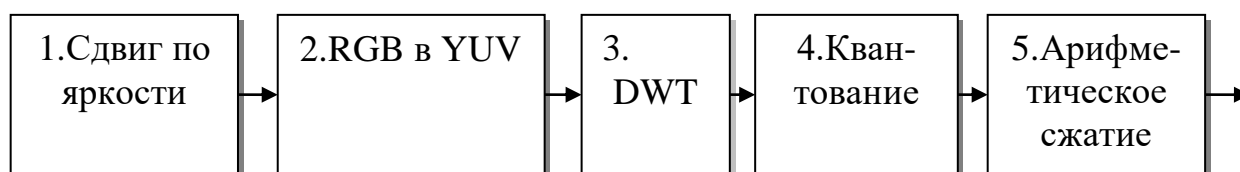


Рисунок 1.3 — Конвейер операций, используемый в алгоритме JPEG-2000

Вейвлеты — математические функции, предназначенные для анализа частотных компонент данных. В задачах сжатия информации вейвлеты используются сравнительно недавно, тем не менее исследователям удалось достичь впечатляющих результатов.

Вейвлеты не требуют предварительного разбиения исходного изображения на блоки, а могут применяться к изображению в целом. Если исходные данные представлены в виде матрицы, то сначала выполняется преобразование для каждой строки, а затем для полученных матриц выполняется преобразование для каждого столбца.

Цвет пропорционален значению функции в точке (чем больше значение, тем темнее). В результате преобразования получается четыре матрицы: одна содержит аппроксимацию исходного изображения (с уменьшенной частотой дискретизации), а три остальных содержат уточняющую информацию об изображении.

Сжатие достигается путём удаления некоторых коэффициентов из уточняющих матриц.

Очевидно, что представление изображения с помощью вейвлетов позволяет добиваться эффективного сжатия, сохраняя при этом визуальное качество изображения. [1.8]

Фрактальное сжатие — это относительно новая область. Фрактал — сложная геометрическая фигура, обладающая свойством самоподобия. Алгоритмы фрактального сжатия сейчас активно развиваются, но идеи, лежащие в их основе, можно описать следующей последовательностью действий.

Процесс сжатия:

1. Разделение изображения на неперекрывающиеся области (домены). Набор доменов должен покрывать всё изображение полностью.
2. Выбор ранговых областей. Ранговые области могут перекрываться и не покрывать целиком всё изображение.
3. Фрактальное преобразование: для каждого домена подбирается такая ранговая область, которая после аффинного преобразования наиболее точно аппроксимирует домен.
4. Сжатие и сохранение параметров аффинного преобразования. В файл записывается информация о расположении доменов и ранговых областей, а также сжатые коэффициенты аффинных преобразований. [1.9]

Аффинное преобразование — отображение плоскости или пространства в себя, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся — в пересекающиеся, скрещивающиеся — в скрещивающиеся.

Существенный недостаток фрактального сжатия — большое время сжатия, т.е. на сжатие рисунка уходят часы на мощных компьютерах.

Таким образом, мы рассмотрели два вида алгоритмов сжатия — сжатие с потерями и без них. Ознакомились с разными алгоритмами, принадлежащими этим двум классам, у каждого из которых рассмотрели достоинства и недостатки. Алгоритмы сжатия без потери, такие как RLE, LZW или LZ77,

имеют общую проблему — недостаточное сжатие изображения. Алгоритмы сжатия с потерями тоже не лишены проблем. К примеру, у фрактального сжатия проблема во времени, даже мощные компьютеры сжимают изображение часами. Более того, отсутствие возможности обратной операции сжатия ограничивает полезность данных алгоритмов.

Тем не менее, наиболее популярными являются алгоритмы сжатия изображений с потерями и, в целях курсовой работы, мы рассмотрим часто используемый на практике алгоритм сжатия сингулярным разложением (svd).

1.2 Алгоритмы сингулярного разложения

Возможно, самый известный и широко используемый метод декомпозиции матрицы — это декомпозиция по сингулярному значению, или SVD. Все матрицы имеют SVD, что делает его более стабильным, чем другие методы, такие как собственное разложение. Как таковой, он часто используется в широком спектре приложений, включая сжатие, удаление шума и уменьшение объема данных.

Сингулярное разложение (Singular Value Decomposition) — декомпозиция вещественной матрицы с целью ее приведения к каноническому виду. Сингулярное разложение является удобным методом при работе с матрицами. Оно показывает геометрическую структуру матрицы и позволяет наглядно представить имеющиеся данные. Сингулярное разложение используется при решении самых разных задач — от приближения методом наименьших квадратов и решения систем уравнений до сжатия изображений.

При этом используются разные свойства сингулярного разложения, например, способность показывать ранг матрицы, приближать матрицы данного ранга. SVD позволяет вычислять обратные и псевдообратные матрицы большого размера, что делает его полезным инструментом при решении задач регрессионного анализа.

$$A = U \cdot d \cdot V^T, \quad (1.1)$$

где A — действительная матрица $M \times N$, которую мы хотим разложить;

U — матрица $M \times M$;

d — диагональная матрица $M \times N$;

V — матрица $N \times N$.

Диагональные значения в матрице d называются сингулярными значениями исходной матрицы A . Столбцы матрицы U называются левосингулярными векторами A , а столбцы V называются правосингулярными векторами A . Число ненулевых элементов на диагонали матрицы A есть фактическая размерность матрицы d .

На Рисунке 1.4 продемонстрировано схематическое представление сингулярного разложения при $m \leq n$.

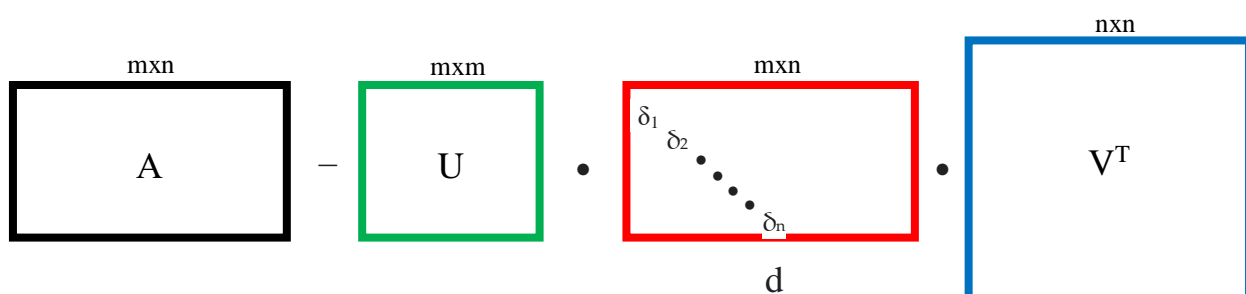


Рисунок 1.4 — Схематическое представление svd при $m \leq n$

На Рисунке 1.5 продемонстрировано схематическое представление сингулярного разложения при $m > n$.

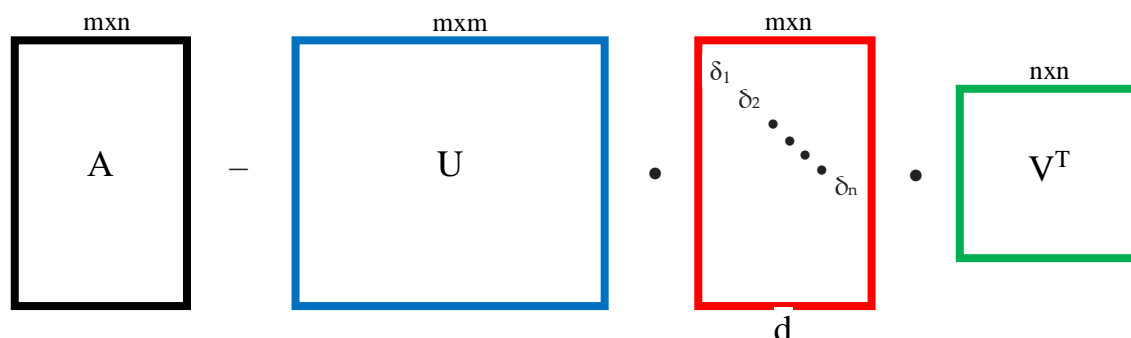


Рисунок 1.5 — Схематическое представление svd при $m > n$

Матрица левых сингулярных векторов U содержит информацию о линейной комбинации столбцов исходной матрицы, которая наиболее сильно влияет на ее разложение. Каждый столбец матрицы U является собственным вектором, соответствующим сингулярному значению матрицы.

Матрица правых сингулярных векторов V содержит информацию о линейной комбинации строк исходной матрицы, которая наиболее сильно влияет на ее разложение. Каждый столбец матрицы V также является собственным вектором, соответствующим сингулярному значению матрицы.

Диагональная матрица сингулярных значений d содержит сингулярные значения, которые представляют собой величины, отражающие важность каждой компоненты в разложении. Более того, можно выбрать U и V так, чтобы диагональные элементы d имели вид

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_r > \lambda_{r+1} = \dots = \lambda_n = 0,$$

где r — ранг матрицы A .

В частности, если A невырождена, то

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n > 0.$$

Индекс r элемента λ_r есть фактическая размерность матрицы d .

Столбцы матриц U и V называются соответственно левыми и правыми сингулярными векторами, а значения диагонали матрицы A называются сингулярными значениями.

Рассмотрим геометрический смысл сингулярного разложения. Пусть A — $(M \times N)$ -матрица и ей в соответствие поставлен линейный оператор, также обозначаемый d . Формулу сингулярного разложения (1.1) можно переформулировать в геометрических терминах.

Линейный оператор, отображающий элементы пространства \mathbb{R}^M в элементы пространства \mathbb{R}^N представим в виде последовательно выполняемых линейных операций растяжения и вращения.

Поэтому компоненты сингулярного разложения наглядно показывают геометрические изменения при отображении линейным оператором d множества векторов из одного векторного пространства в другое.

Критерием качества разложения, выражающим насколько много информации было сохранено по итогу алгоритма, служит близость к единице коэффициента детерминации, рассчитываемого по формуле

$$Q(r) = \frac{\sum_{k=1}^r \lambda_k}{\sum_{k=1}^n \lambda_k},$$

где λ_k — сингулярные значения.

Зависимость коэффициента детерминации от числа главных компонент позволяет оценить эффективность алгоритма. [1.10]

Свойства сингулярного разложения матрицы:

1. Увеличение (уменьшение) всех значений матрицы A на константу $k \neq 0$ нелинейно изменяет все сингулярные числа, изменяются также матрицы U и V .
2. Умножение матрицы на константу $k \neq 0$ изменяет в k раз значения сингулярных чисел, не меняя матрицы U и V .
3. Изменение одного элемента матрицы A может нелинейно изменить все сингулярные числа и матрицы U и V .
4. Перестановка двух элементов матрицы A может изменить все множество сингулярных чисел.
5. От перестановки строк или столбцов матрицы A массив ее сингулярных чисел не изменяется.

SVD широко используется как при вычислении других матричных операций, таких как обратная матрица, так и в качестве метода сокращения

данных в машинном обучении. SVD также может быть использован в линейной регрессии наименьших квадратов, сжатии изображений и шумоподавлении данных.

Единственность сингулярного разложения матрицы не гарантирована. Однозначно определена только матрица d (при условии упорядочения сингулярных чисел по убыванию).

Так как сингулярные значения быстро убывают, то, начиная с некоторого номера p , их вклад в общую сумму достаточно мал, и эти оставшиеся значения можно не учитывать при восстановлении исходного изображения. В формуле (1.1) нет необходимости целиком сохранять матрицы U , d , V , что позволяет производить сжатие изображения по компонентам сжатия.

Эффективность алгоритма существенно повышается при наличии на изображении похожих элементов с простой геометрической формой. Преобладание мелких деталей приводит к необходимости использования большего числа главных компонент для получения изображения требуемого качества.

Таким образом, алгоритм сингулярного разложения для изображений имеет свои плюсы и минусы. Он крайне эффективен при наличии на изображении похожих элементов с простой геометрической формой. Также он крайне эффективен при работе с черно-белыми изображениями. Но алгоритм сингулярного разложения уступает другим в сжатии цветных фото или фото с большим количеством разных элементов.

Подводя итоги, мы можем сказать, что svd по праву заслуживает своей популярности. В параграфе выше мы рассмотрели принцип работы алгоритма сингулярного разложения и теперь можем программно реализовать его для сжатия изображений и выполнения целей данной курсовой работы. Программный код будет написан на языке R.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Разработка и тестирование приложения

Для создания приложения нам потребуется запустить RStudio и создать новый проект. [2.1]

Внутри проекта создадим папку под названием «compressGUI», в ней будут храниться наши скрипты. Каждое приложение располагается в отдельной папке, а имя этой папки совпадает с именем приложения.

Установим специальные библиотеки shiny и jpeg. Нам понадобится перейти во вкладку Packages и нажать кнопку Install. В появившейся графе поиска введем сначала shiny, а затем jpeg, после чего RStudio автоматически скачает и установит библиотеки.

Библиотека shiny дает возможность работы и проектирования с GUI. Библиотека jpeg дает возможность чтения и записи изображений. [2.2]

Далее создадим четыре скрипта:

- compress.R;
- ui.R;
- server.R;
- activator.R.

Первый скрипт является функцией сжатия фотографии алгоритмом сингулярного разложения.

compress — данная функция отвечает за сжатие изображений. Опишем ее подробнее.

Сначала записываем в переменную photo_copу изображение, которое подается на вход функции. Функцией if определяем, как мы будем сжимать фото (по количеству компонент сжатия или же по проценту суммарной дисперсии).

Если мы сжимаем фото по количеству компонент сжатия, то вводим функцию `for` для отдельного сжатия по всем трем каналам `rgb`. Далее применяем функцию `svd`, которая разбивает матрицу цвета на три матрицы, одна из которых диагональная. При последовательном перемножении эти матрицы дают исходную матрицу цвета. В заключение, ограничиваем матрицы по элементам компонент сжатия и перемножаем, получая новую уже сжатую матрицу цвета.

Если же мы сжимаем изображение по проценту суммарной дисперсии, то по красному каналу с помощью цикла `for` сравниваем сумму элементов диагональной матрицы, ограниченной по итератор `k`, деленное на сумму всей диагональной матрицы с нашим процентом суммарной дисперсии, который подается на вход функции. Далее, когда мы узнали количество компонент сжатия, перемножаем матрицы последовательно и получаем новую матрицу цвета. Завершая функцию, мы возвращаем новое сжатое фото.

Второй скрипт является пользовательским интерфейсом. Он отвечает за внешнюю оболочку приложения.

Теперь опишем страницу приложения библиотеке «shiny». Ниже приведены добавленные функции:

- `titlePanel` — функция вывода заголовка панели приложения;
- `radioButtons` — функция вывода на экран кнопок-переключателей, которые будут давать возможность выбора что мы будем задавать (количество компонент или процент суммарной дисперсии);
- `textInput` — функция вывода на экран текста, в нашем случае количество компонент или процент суммарной дисперсии;
- `fileInput` — функция вывода на экран поля для загрузки файлов, в нашем случае изображения;
- `imageOutput` — функция вывода на экран сжатого изображения.

В ходе выполнения описанных выше действий, заполнен скрипт пользовательского интерфейса.

На Рисунке 2.1 представлен получившийся графический интерфейс.

Compress your photo by svd.

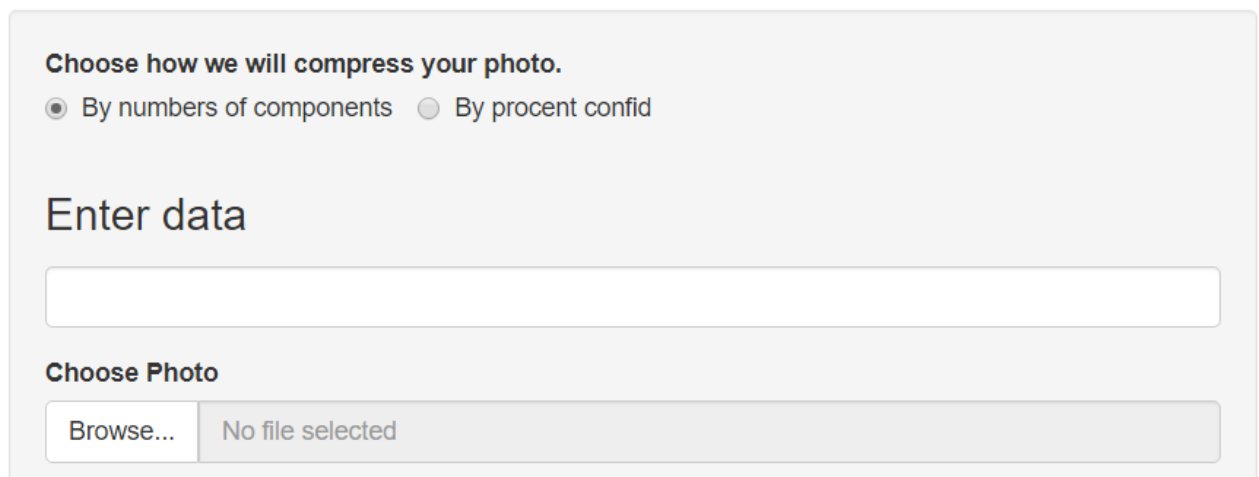


Рисунок 1

Рисунок 2.1 — Графический интерфейс

После создания экрана приложения пропишем третий скрипт, который позволит реагировать на действия пользователя при взаимодействии с приложением.

Поскольку библиотеки установлены, их нужно подключить. Для этого воспользуемся командами `library(shiny)` и `library(jpeg)`.

Для визуализации результатов воспользуемся функцией `renderImage` — данная функция отвечает за обработку и вывод на экран полученной фотографии из функции `fileInput`. Опишем ее подробнее.

Первым делом убираем ошибку пустой переменной с помощью функции `tryCatch`. Далее запоминаем ширину и длину окна пользователя, для динамического масштабирования фото под экран пользователя. Вводим временную переменную `outpfile` хранения изображения. Читаем и запоминаем изображение в переменную `pic`. Далее сжимаем фото с помощью нашего алгоритма сжатия. Записываем изображение в временную переменную `outfile`. Последним действием функции является возвращение списка информации об изображении и удаление фото из памяти с помощью функции `deleteFile`.

Последним скриптом является активатор нашего приложения, который содержит в себе функцию `runApp`, которая позволяет запустить наше приложение.

Итак, программный код реализован, теперь можно приступить к тестированию приложения.

На Рисунке 2.2 представлено изображение, которое мы подаем программе.



Рисунок 2.2 — Исходное изображение

Введем данные сжатия: выберем сжатие по количеству компонент и впишем в строку ввода число 20 (20 компонент сжатия). Теперь загрузим на обработку наше изображение.

На Рисунке 2.3 представлен пример графического интерфейса.

Compress your photo by svd.

Choose how we will compress your photo.

☒ By numbers of components ☐ By procent confid

Enter data

20

Choose Photo

Browse... Night.jpg

Upload complete

Рисунок 2.3 — Пример приложения

Теперь надо некоторое время подождать, пока алгоритм сожмет наше изображение. Через некоторое время получаем результат.

На Рисунке 2.4 представлен пример графического интерфейса.

Compress your photo by svd.


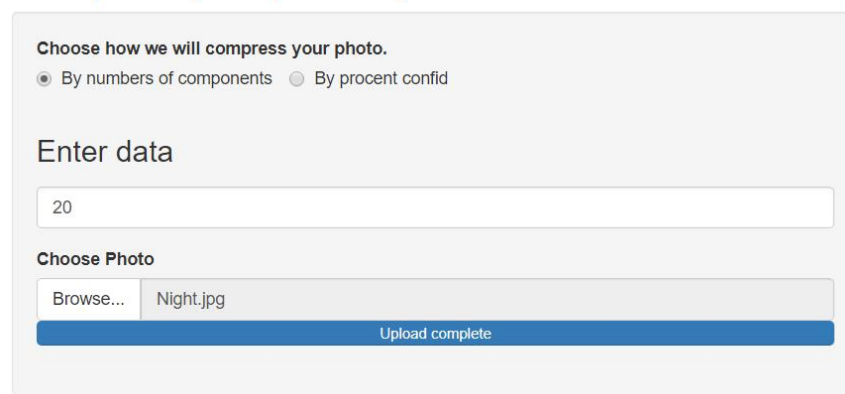


Рисунок 2.4 — Результат работы программы

Сжатое изображение получено. Сжатие по количеству компонент работает верно. Данное фото сжалось более чем в 2,5 раза (исходное изображение весило 1,28МБ, а полученное фото весит 529КБ).

Из полученного изображения видна основная проблема алгоритма сингулярного разложения. В особо контрастных местах, например, на самых светлых участках картинке появляются артефакты в виде цветных пятен.

В качестве более заметного примера загрузим фото с большим количеством участков, переходящих резко в другой цвет.

На Рисунке 2.5 представлено новое изображение, которое мы подаем программе.



Рисунок 2.5 — Исходное изображение

Введем данные сжатия: выберем сжатие по количеству компонент и впишем в строку ввода число 20 (20 компонент сжатия). Теперь загрузим на обработку наше изображение.

На Рисунке 2.6 представлен пример графического интерфейса.

Compress your photo by svd.

Choose how we will compress your photo.

☒ By numbers of components ☐ By procent confid

Enter data

20

Choose Photo

Browse... priroda_kartinki_foto_03.jpg

Upload complete

Рисунок 2.6 — Пример графического интерфейса

Теперь надо снова некоторое время подождать, пока алгоритм сожмет наше изображение. Через некоторое время получаем результат.

На Рисунке 2.7 представлен пример графического интерфейса.

Compress your photo by svd.

Choose how we will compress your photo.
☒ By numbers of components ☐ By procent confid

Enter data

Choose Photo
 priroda_kartinki_foto_03.jpg



Рисунок 2.7 — Пример графического интерфейса

Теперь на фото хорошо видны артефакты. Мы рассмотрели результат, когда алгоритм работает недостаточно хорошо. Для устранения этой проблемы можно загружать фото в черно-белом формате.

На Рисунке 2.8 представлено изображение в черно-белом формате.



Рисунок 2.8 — Изображение в черно-белом формате

Проделаем всю ту же процедуру, которую делали ранее и получим результат.

На Рисунке 2.9 представлен пример графического интерфейса.

Compress your photo by svd.

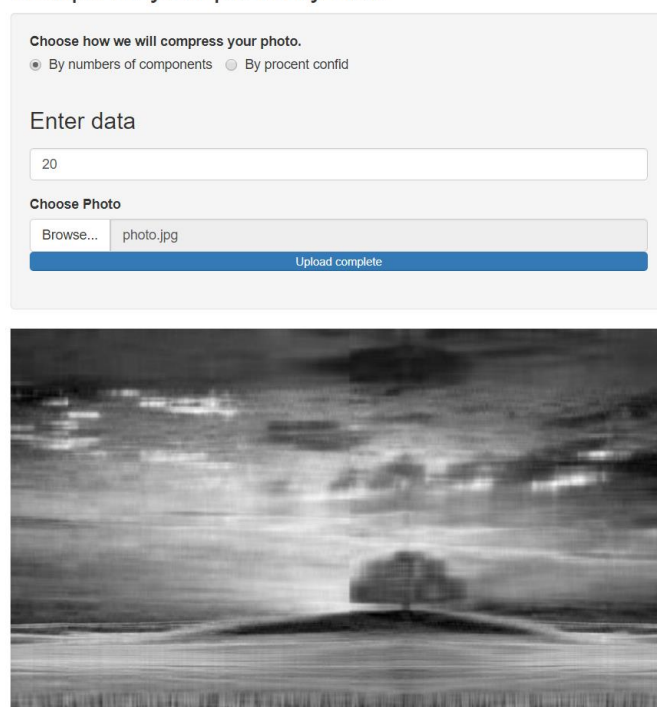


Рисунок 2.9 — Пример графического интерфейса

В результате мы видим, что артефакты с изображения устранены.

Теперь протестируем приложение, задав изначальноными данными процент суммарной дисперсии (оставшийся процент значимой информации на изображении).

На Рисунке 2.10 представлено изображение, которое мы подаем программе.

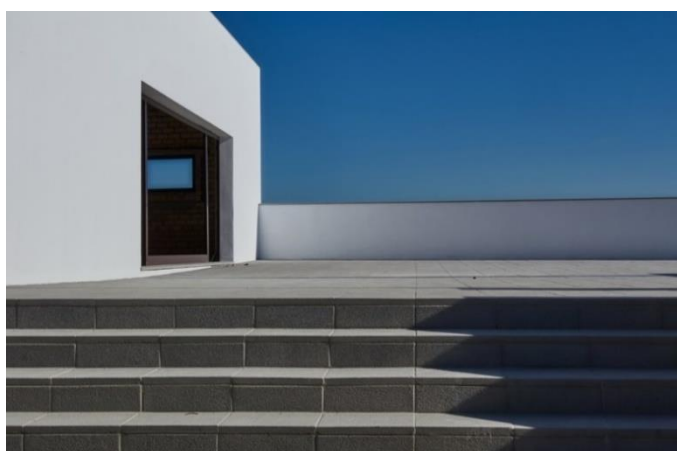


Рисунок 2.10 — Исходное изображение

Введем данные сжатия: выберем сжатие по проценту суммарной дисперсии и впишем в строку ввода число 0.8 (80 процентов значимой

информации останется на фото). Теперь загрузим на обработку наше изображение.

На Рисунке 2.11 представлен пример графического интерфейса.

Compress your photo by svd.

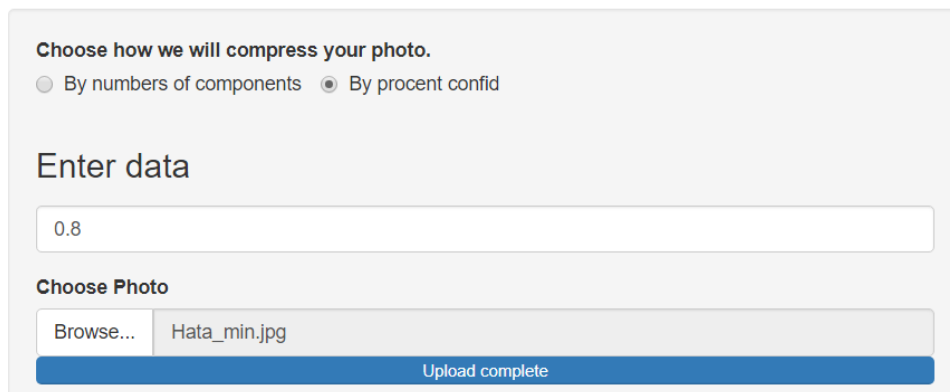


Рисунок 2.11 — Пример графического интерфейса

Немного подождав, получаем результат.

На рисунке 2.12 представлен пример графического интерфейса.

Compress your photo by svd.

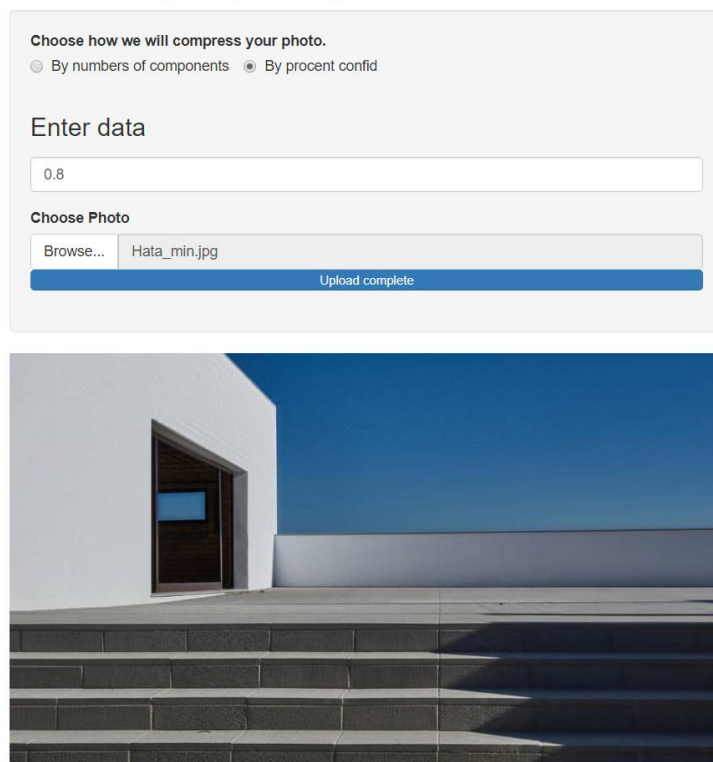


Рисунок 2.12 — Пример графического интерфейса

Сжатое изображение получено. Сжатие по проценту суммарной дисперсии работает верно. В результате получаем фото почти в 10 раз меньше по размеру, чем исходное (исходное фото весило 426КБ, а полученное 45,9КБ).

Графический интерфейс полностью работает верно.

На Рисунке 2.13 представлен график количества компонент сжатия к значимой информации изображения. По оси ОХ изображено количество компонент сжатия, а по оси ОУ процент от суммарной дисперсии.

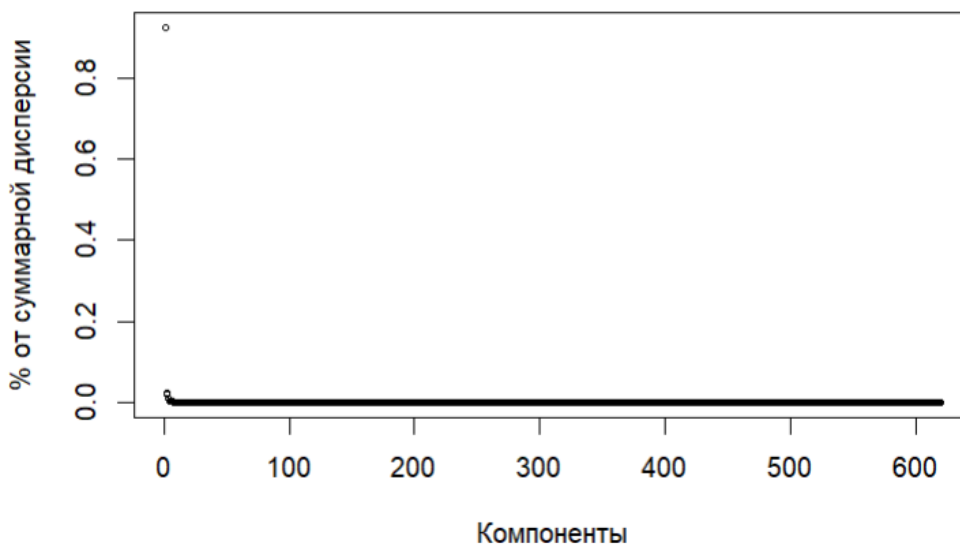


Рисунок 2.13 — График количества компонент сжатия к проценту от суммарной дисперсии

В результате, можем отметить, что после 50 компонент сжатия фото уже практически не будет сжиматься, поэтому нерационально брать больше 50 компонент сжатия для большинства фотографий.

Итак, все параметры установлены согласно указаниям. Графический интерфейс настроен и полностью готов к работе. Функционал приложения может легко дополняться за счет разбиения основной программы на 4 главных компонента. Реализован функционал приёма задания параметров сжатия двумя способами: по количеству компонент и по проценту суммарной дисперсии. Проведена настройка и тестирование приложения. Проведена обработка изображения, в результате которого фотография подверглась сжатию почти в 10 раз (исходное фото весило 426КБ, а полученное 45,9КБ).

ЗАКЛЮЧЕНИЕ

Жизнь современного общества невозможно представить без широкого применения информационных технологий. Бурное развитие Интернета, компьютерной, копировальной и фототехники привело к широкому использованию цифровых изображений и обусловило большой интерес к разработке эффективных алгоритмов их сжатия.

С повсеместным ростом объёма данных, должны эволюционировать и оптимизироваться и способы их представления. Применение алгоритмов, обеспечивающих высокую степень сжатия изображений, позволяет увеличить скорость передачи данных по каналам связи, эффективность их хранения и использования.

В ходе данной курсовой работы на языке программирования R была разработана и протестирована программа для сжатия изображений методом сингулярного разложения, разработан графический интерфейс для взаимодействия с ней. Таким образом, цель данной курсовой работы достигнута.

В ходе данной курсовой работы выполнены следующие задачи:

- изучена документация и техническая литература по исследуемой теме;
- реализован алгоритм сжатия изображений методом сингулярного разложения
- разработан программный код приложения с использованием языка программирования R;
- приложение протестировано и результаты собраны;
- изучены правила качественного оформления документации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

- 1.1 Хабр [Электронный ресурс] / Алгоритмы сжатия изображений. — Режим доступа: <https://habr.com/ru/post/116697/>
- 1.2 Аль-Бахдили Х.К., Макейчик Е.Г., Цветков В.Ю., Конопелько В.К. Сжатие полутонных изображений без потерь на основе кодирования длин серий. // Доклады БГУИР. — 2019. — 96 с.
- 1.3 Поддубный А.П., Холуев М.А., Галактионов Н.С. Использование файла в качестве избыточного словаря для препроцессинга данных на основе словарных методов сжатия // Известия вузов. Поволжский регион. Технические науки. — 2013. — 8 с.
- 1.4 Компьютер пресс. [Электронный ресурс] / Сжатие информации с потерями и без. — Режим доступа: <https://compress.ru/article.aspx?id=10581#06>
- 1.5 Шепиль Руслан Олегович. Сжатие данных и как устроены архиваторы // StudNet. — 2020. — 6 с.
- 1.6 Дизер А.Е., Дизер Е.С., Опарина Т.М. Модификация метода Куттера-Джордана-Боссена скрытого хранения информации в изображениях формата JPEG. — Москва: МСМ, 2019. — 39 с.
- 1.7 Хабр [Электронный ресурс] / JPEG. Алгоритм сжатия изображений. — Режим доступа: <https://habr.com/ru/post/482728/>
- 1.8 Садыхов Р.Х., Козловский А.Н. Алгоритм сжатия изображений на базе вейвлет-преобразования // Доклады БГУИР. — 2018. — 19 с.
- 1.9 Temofeev.ru [Электронный ресурс] / Фрактальное сжатие изображений. — Режим доступа: <https://temofeev.ru/info/articles/fraktalnoe-szhatie-izobrazheniy/>
- 1.10 Богданова Н.А., Зыбина Ю.С., Шпакова Е.С. Использование сингулярного разложения матриц для сжатия электронно-

микроскопических изображений. — Москва: Национальный исследовательский университет «МИЭТ», 2018. — 5 с.

ПРАКТИЧЕСКАЯ ЧАСТЬ

- 2.1 RStudio [Электронный ресурс]. — Режим доступа:
<https://www.rstudio.com>
- 2.2 Shiny from RStudio [Электронный ресурс]. — Режим доступа:
<https://shiny.rstudio.com>

ПРИЛОЖЕНИЯ

Приложение А — программный код

Приложение А

```
#UI.R

#Подключение пакета
library(shiny)
shinyUI(fluidPage(
  titlePanel("Compress your photo by svd."),
  sidebarLayout(
    sidebarPanel(
      #Вывод бокса с выбором метода сжатия
      radioButtons('format', 'Choose how we will compress your photo.',
        c('By numbers of components',
          'By procent confid'),
        inline = TRUE),

      #Вывод поля для ввода количества компонент или процента суммарной
      дисперсии
      textInput("data", label = h3("Enter data"),
        value = ""),

      #Вывод поля для загрузки фото
      fileInput("file", "Choose Photo",
        multiple = FALSE),
    ),
    mainPanel(
      #Вывод получившейся картинки
      imageOutput("photo")
    )
  )
))

#server.R

#Загрузка пакетов
```

```

library(shiny)
library(jpeg)
source("compress.R")
server <- function(input, output, session) {
  output$photo <- renderImage({
    req(input$file)
    #Убираем ошибку пустой переменной input$file
    tryCatch(
      {
        df <- readJPEG(input$file$datapath)
      },
      error = function(e) {
        stop(safeError(e))
      }
    )
    #Получаем ширину и длину изображения
    width <- session$clientData$output_photo_width
    height <- session$clientData$output_photo_height
    outfile <- tempfile(fileext = ".jpg")
    #Читаем изображение
    pic <- readJPEG(input$file$datapath)
    #Сжимаем фото
    if (input$format == 'By numbers of components') {
      result <- compress(pic, input$data , NULL)}
    else if (input$format == 'By procent confid') {
      result <- compress(pic, NULL , input$data)}
    writeJPEG(result$compressed_image, target = outfile)
    # Возвращаем список информации о изображении
    list(src = outfile,
         contentType = "data",

```

```

        width = width,
        height = height,
        alt = "This is alternate text")
    }, deleteFile = TRUE) #удаляем фото из памяти
}
#compress.R
compress <- function(image, n_components = 100, confid = NULL) {
  photo_copy <- image
  sum_full = 0
  sum_comp = 0
  if (is.null(confid)) {
    #Разбиваем изображение на три канала
    for (channel in 1:3) {
      photo_svd <- base::La.svd(image[, , channel])
      photo_copy[, , channel] <- photo_svd$u[,1:n_components] %*%
        diag(photo_svd$d[1:n_components]) %*%
        photo_svd$vt[1:n_components,]
      #Считаем данные для вычисления потери изображения
      sum_full = sum_full + sum(photo_svd$d)
      sum_comp = sum_comp + sum(photo_svd$d[1:n_components])
    }
    return(list("compressed_image" = photo_copy, "confid" = sum_comp / sum_full))
  } else {
    photo_svd <- base::La.svd(image[, , 1])
    for (k in 1:length(diag(photo_svd$d))) {
      if (sum(photo_svd$d[1:k])/sum(photo_svd$d)>=confid) {
        comp <- k
        break
      }
    }
  }
}

```



```

for (channel in 1:3) {
  photo_svd <- base::La.svd(image[, , channel])
  photo_copy[, , channel] <- photo_svd$u[,1:comp] %*%
    diag(photo_svd$d[1:comp]) %*%
    photo_svd$vt[1:comp,]
}
return(list("compressed_image" = photo_copy, "n_components" = comp))
}
}
#activator.R
library(shiny)
runApp("compressGUI")

```