Министерство образования и науки 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» (ИУ)

КАФЕДРА «Системы обработки информации и управления» (ИУ5)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Поисковая система научных информационных материалов на основе открытых источников сети интернет»

Студент группы ИУ5-32М _____ А. В. Силаев

Руководитель _____ Ю. Е. Гапанюк

2021 г.

Министерство образования и науки 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)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ5

_____ В.М. Черненький

« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме «Сервис импорта и экспорта данных из репозитория информационных материалов»

Студент группы ИУ5-32М

Чиженков Борис Михайлович

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

исследовательская

Источник тематики (кафедра, предприятие, НИР) _____

График выполнения НИР: 25% к 5 нед., 50% к 9 нед., 75% к 13 нед., 100% к 17 нед.

Техническое задание _____

1. Обзор XML, используемых для создания базы данных.
2. Проектирование базы данных.
3. Разработка прототипа системы.

_____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 7 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

_____

_____

_____

_____

Дата выдачи задания «1» сентября 2021 г.

Руководитель НИР _____ Ю. Е. Гапанюк

Студент _____ А. В. Силаев

ОГЛАВЛЕНИЕ

Введение

Зрительная система человека (HVS) очень чувствительна к визуальным средствам, таким как изображения и видео. Изображения сегодня используются в широком спектре приложений, от социальных сетей, таких как Facebook, Instagram и Snapchat, до области машинного обучения и искусственного интеллекта для различных задач, таких как распознавание образов и обнаружение опухолей на медицинских изображениях. Недавние достижения и принятие таких приложений частично стали возможными благодаря увеличению пропускной способности и улучшению возможностей сжатия и обработки изображений. Сжатие изображений является широко исследованной и изученной областью, при этом многие стандарты сжатия изображений доступны как для кодирования с потерями, так и без потерь, при этом также предлагаются стандарты и методы сжатия для конкретных приложений [1] [2]. В своей работе для всех трех форматов я выбрал метод сжатия с потерями.

Формат JPEG

JPEG, который используется с 1992 года, в настоящее время является наиболее широко используемым стандартом сжатия изображений с потерями, особенно для интернет-приложений и цифровых камер, и в настоящее время его используют почти 70 % веб-сайтов. Алгоритм JPEG наиболее эффективен для сжатия фотографий и картин, содержащих реалистичные сцены с плавными переходами яркости и цвета. Наибольшее распространение JPEG получил в цифровой фотографии и для хранения и передачи изображений с использованием Интернета.

Формат JPEG в режиме сжатия с потерями малопригоден для сжатия чертежей, текстовой и знаковой графики, где резкий контраст между соседними пикселами приводит к появлению заметных артефактов. Такие изображения целесообразно сохранять в форматах без потерь, таких как TIFF, GIF, PNG.

Алгоритм сжатия

При сжатии изображения в формате JPEG, оно преобразуется из цветового пространства RGB в YCbCr. Стандарт JPEG (ISO/IEC 10918-1) не регламентирует выбор именно YCbCr, допуская и другие виды преобразования (например, с числом компонентов [3], отличным от трёх), и сжатие без преобразования (непосредственно в RGB), однако спецификация JFIF предполагает использование преобразования RGB -> YCbCr.

После преобразования RGB -> YCbCr для каналов изображения Cb и Cr, отвечающих за цвет, может выполняться «прореживание» (subsampling[4]), которое заключается в том, что каждому блоку из 4 пикселей (2x2) яркостного канала Y ставятся в соответствие усреднённые значения Cb и Cr (схема прореживания «4:2:0»[5]). При этом для каждого блока 2x2 вместо 12 значений (4 Y, 4 Cb и 4 Cr) используется всего 6 (4 Y и по одному усреднённому Cb и Cr). Если к качеству восстановленного после сжатия изображения предъявляются повышенные требования, прореживание может выполняться лишь в каком-то одном направлении — по вертикали

(схема «4:4:0») или по горизонтали («4:2:2»), или не выполняться вовсе («4:4:4»).

Стандарт допускает также прореживание с усреднением Cb и Cr не для блока 2x2, а для четырёх расположенных последовательно (по вертикали или по горизонтали) пикселей, то есть для блоков 1x4, 4x1 (схема «4:1:1»), а также 2x4 и 4x2 (схема «4:1:0»). Допускается также использование различных типов прореживания для Cb и Cr, но на практике такие схемы применяются исключительно редко.

Далее яркостный компонент Y и отвечающие за цвет компоненты Cb и Cr разбиваются на блоки 8x8 пикселей. Каждый такой блок подвергается дискретному косинусному преобразованию (ДКП). Полученные коэффициенты ДКП квантуются (для Y, Cb и Cr в общем случае используются разные матрицы квантования) и пакуются с использованием кодирования серий и кодов Хаффмана. Стандарт JPEG допускает также использование значительно более эффективного арифметического кодирования, однако из-за патентных ограничений (патент на описанный в стандарте JPEG арифметический QM-кодер принадлежит IBM) на практике оно используется редко. В популярную библиотеку libjpeg последних версий включена поддержка арифметического кодирования, но с просмотром сжатых с использованием этого метода изображений могут возникнуть проблемы, поскольку многие программы просмотра не поддерживают их декодирование.

Матрицы, используемые для квантования коэффициентов ДКП, хранятся в заголовочной части JPEG-файла. Обычно они строятся так, что высокочастотные коэффициенты подвергаются более сильному квантованию, чем низкочастотные. Это приводит к огрублению мелких деталей на изображении. Чем выше степень сжатия, тем более сильному квантованию подвергаются все коэффициенты.

При сохранении изображения в JPEG-файле кодеру указывается параметр качества, задаваемый в некоторых условных единицах, например,

от 1 до 100 или от 1 до 10. Большее число обычно соответствует лучшему качеству (и большему размеру сжатого файла). Однако, в самом JPEG-файле такой параметр отсутствует, а качество восстановленного изображения определяется матрицами квантования, типом прореживания цветоразностных компонентов и точностью выполнения математических операций как на стороне кодера, так и на стороне декодера. При этом даже при использовании наивысшего качества (соответствующего матрице квантования, состоящей из одних только единиц, и отсутствию прореживания цветоразностных компонентов) восстановленное изображение не будет в точности совпадать с исходным, что связано как с конечной точностью выполнения ДКП, так и с необходимостью округления значений Y, Cb, Cr и коэффициентов ДКП до ближайшего целого.

Формат WebP

WebP — это формат файла, разработанный компанией Google в 2010 году. Его особенностью является продвинутый алгоритм сжатия, позволяющий сократить размер картинки без видимых потерь в качестве.

Алгоритм сжатия

Сжатие состоит из двух этапов. На первом делается попытка «предсказать» содержимое одних блоков по уже декодированным (три блока над текущим и один блок слева от него), на втором кодируется ошибка предсказания. Блоки отрисовываются в порядке слева направо и сверху вниз [6]. Режимы предсказания работают с 3 размерами макроблоков [7]:

- 4x4 в канале яркости
- 16x16 в канале яркости
- 8x8 в канале цветности
- В VP8 и WebP реализованы режимы предсказания [7]:

Горизонтальное, H_PRED. Каждый столбец текущего блока есть копия столбца слева от текущего блока.

Вертикальное, V_PRED. Каждая строка текущего блока есть копия строки над текущим блоком.

DC предсказание, DC_PRED. Заполняет весь блок одинаковыми значениями, полученными за счет усреднения значений пикселов вышележащей строки и столбца слева от текущего блока.

Предсказание TrueMotion, TM_PRED. Разработано в On2 Technologies. Кроме строки над блоком и столбца слева от него, используется пиксел, расположенный сверху-слева от блока. Разница между угловым пикселом и строкой сверху записывается в строки блока, при этом к значениям добавляется значение соответствующего пиксела из столбца. Xij = Столбецi + Рядj - Угл. Пиксел.

Для блоков 4x4 реализовано 6 дополнительных режимов, сходных с V_PRED и H_PRED, но с диагональными направлениями.

Для сжатия ошибок предсказания и подблоков, которые не были предсказаны, используется дискретное косинусное преобразование DCT (и, изредка, преобразование Уолша—Адамара, WHT). Оба преобразования работают с подблоками размером 4x4 пиксела. Реализация преобразований выполнена на представлении чисел с фиксированной точностью, чтобы уменьшить ошибки округления [8]. Коэффициенты DCT и WHT пакуются энтропийным кодеком.

WebP не работает в цветовом пространстве RGB, перед кодированием изображение переводится в YUV с глубиной 8 бит и форматом 4:2:0. Перевод осуществляется согласно стандарту ITU-R BT.601[8].

Для некоторых изображений может использоваться алгоритм upscaling, когда кодируется не само изображение, а его отмасштабированная (уменьшенная) версия. Декодер проводит обратное преобразование (увеличение изображения) [8].

Формат AVIF

AVIF, или формат изображения AV, представляет собой бесплатный формат изображений с открытым исходным кодом, основанный на кодеке AV1, и, как и AV1, AVIF обеспечивает очень высокую степень сжатия. Тот факт, что он не требует лицензионных отчислений, выделяет его среди конкурентов. Кодек AV1 уже поддерживается многими веб-платформами, включая Android, Chrome, Microsoft Edge и Firefox, и несколько поставщиков веб-видеоуслуг, включая YouTube, Netflix, Vimeo и Bitmovin, начали масштабное развертывание потоковых сервисов AV1.

Алгоритм сжатия

Изначально происходит разбиение изображения на блоки.

Содержимое кадра разделяется на смежные блоки одинакового размера, называемые суперблоками. Подобно концепции макроблока, суперблоки имеют квадратную форму и могут иметь размер 128 × 128 или 64 × 64 пикселей. Суперблоки можно разделить на более мелкие блоки в соответствии с различными шаблонами разделения. Шаблон четырехстороннего разделения - единственный шаблон, разделы которого могут быть рекурсивно разделены. Это позволяет разделить суперблоки на разделы размером 4 × 4 пикселя.

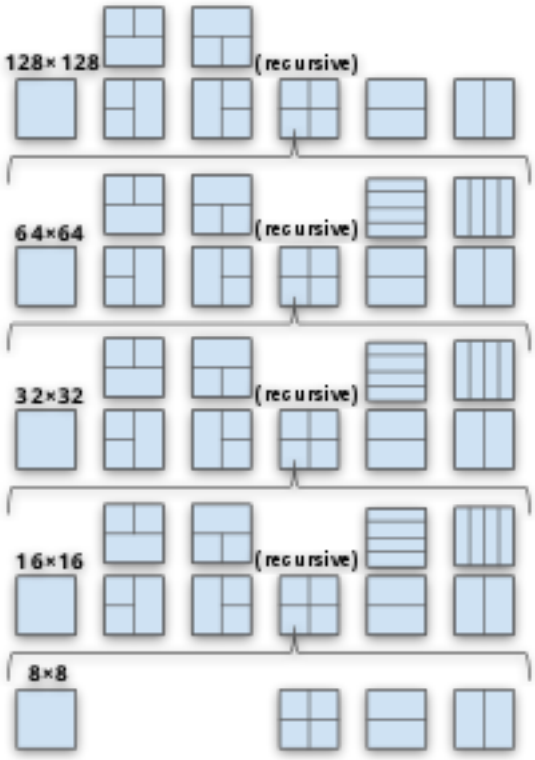


Схема разбиения суперблока AV1. Он показывает, как суперблоки 128 × 128 могут быть полностью разделены до блоков 4 × 4. В особых случаях блоки 128 × 128 и 8 × 8 не могут использовать разделения 1: 4 и 4: 1, а блоки 8 × 8 не могут использовать Т-образные разделения.

Есть "Т-образные" шаблоны разделения, а также горизонтальное или вертикальное разделение на четыре полосы с соотношением сторон 4:1 и 1:4. Доступные шаблоны разделения различаются в зависимости от размера блока, блоки 128×128 и 8×8 не могут использовать разделение 4:1 и 1:4. Более того, блоки 8×8 не могут использовать Т-образное разбиение.

На следующем этапе происходит предсказание.

AV1 выполняет внутреннюю обработку с более высокой точностью (10 или 12 бит на выборку), что приводит к улучшению сжатия за счет меньших ошибок округления в эталонных изображениях.

Предсказания могут быть объединены более продвинутыми способами (чем равномерное среднее значение) в блоке (составное предсказание), включая плавные и резкие градиенты перехода в разных направлениях (предсказание с разделением клином), а также неявные маски, основанные на

разнице между двумя предсказателями. Это позволяет использовать комбинацию либо двух промежуточных прогнозов, либо промежуточного и внутреннего прогнозирования в одном блоке. [10]

Кадр может ссылаться на 6 вместо 3 из 8 доступных буферов кадров для временного (промежуточного) прогнозирования, обеспечивая при этом большую гибкость при двунаправленном прогнозировании.

После этих этапов происходит преобразование данных.

Чтобы преобразовать ошибку, оставшуюся после прогнозирования, в частотную область, кодеры AV1 могут использовать квадратные, 2:1/1:2 и 4:1/1:4 прямоугольные DCT (rect_tx), а также асимметричное летнее время[11][12][13] для блоков, где ожидается, что верхний и/или левый край будет иметь меньшую ошибку благодаря прогнозированию из соседних пикселей, или выбрать не преобразовывать (преобразование идентичности).

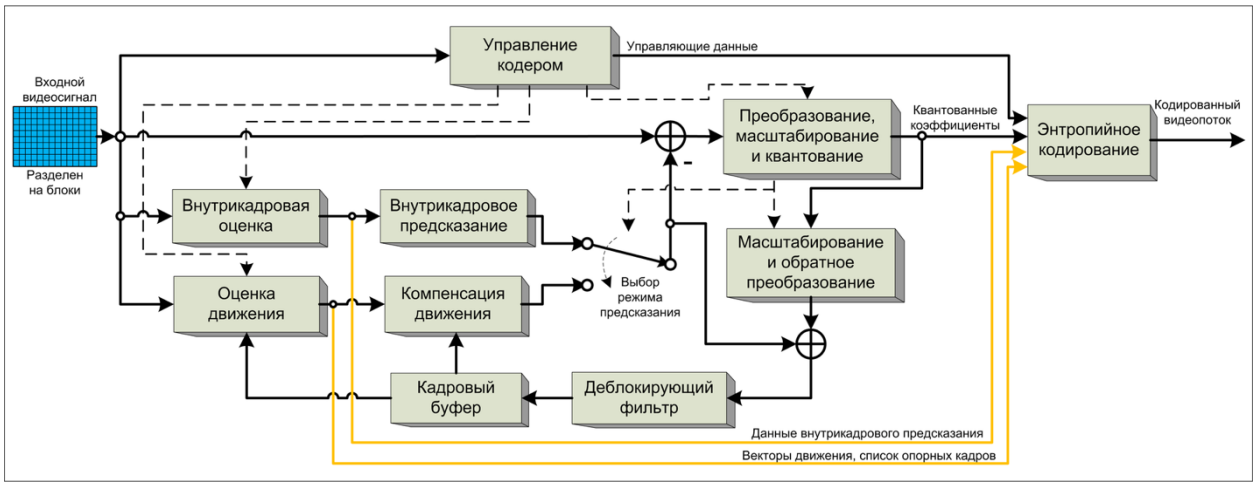
Он может комбинировать два одномерных преобразования, чтобы использовать разные преобразования для горизонтального и вертикального измерения (ext_tx).[9]

Формат HEIF

Высокоэффективный формат файлов изображений (HEIF) — это контейнерный формат для хранения отдельных изображений и последовательностей изображений. Стандарт охватывает мультимедийные файлы, которые также могут включать другие потоки мультимедиа, такие как синхронизированный текст, аудио и видео.

HEIF может хранить изображения, закодированные в нескольких форматах кодирования, например изображения SDR и HDR. Изображение HEIF с использованием HEVC требует меньше места для хранения, чем JPEG эквивалентного качества. [14] [15]

Алгоритм сжатия



При кодировании фото в HEVC применяется такой же «гибридный» подход, что и во всех современных кодеках. Он заключается в применении предсказания и двумерного кодирования с преобразованием.

В кодере HEVC каждое фото делится на блоки. Первый кадр кодируется с использованием только внутрикадрового предсказания, то есть применяется пространственное предсказание ожидаемого уровня отсчёта внутри кадра по соседним отсчётам, при этом отсутствует зависимость от других кадров. Для большинства блоков всех остальных кадров последовательности, как правило, используется режим межкадрового

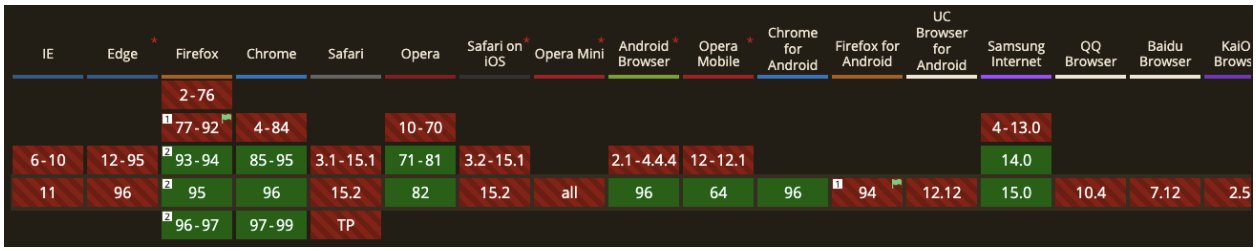
временного предсказания. В режиме межкадрового предсказания на основании данных о величине отсчётов опорного кадра и вектора движения оцениваются текущие отсчёты каждого блока. Кодер и декодер создают идентичные межкадровые предсказания путём применения алгоритма компенсации движения с помощью векторов движения и данных выбранного режима, которые передаются в качестве дополнительной информации.

Разностный сигнал предсказания, который представляет собой разницу между опорным блоком кадра и его предсказанием, подвергается линейному пространственному преобразованию. Затем коэффициенты преобразования масштабируются, квантуются, применяется энтропийное кодирование, и затем передаются вместе с информацией предсказания.

Кодер в точности повторяет цикл обработки декодером так, что в обоих случаях будут генерироваться идентичные предсказания последующих данных. Таким образом, преобразованные квантованные коэффициенты подвергаются обратному масштабированию и затем обратному преобразованию, чтобы повторить декодированное значение разностного сигнала. Разность затем добавляется к предсказанию, и полученный результат фильтруется для сглаживания артефактов, полученных делением на блоки и при квантовании. Окончательное представление кадра (идентичное кадру на выходе декодера) хранится в буфере декодированных кадров, которое будет использоваться для прогнозирования последующих кадров. В итоге порядок кодирования и декодирования обработки кадров часто отличается от порядка, в котором они поступают из источника.

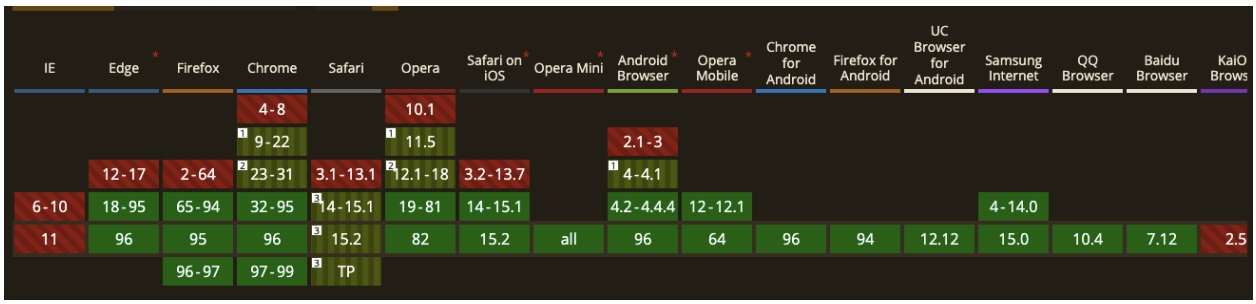
Поддерживаемые браузеры

AVIF



На данный момент AVIF не так популярен как его конкурент WebP всего лишь 69.38% против 95,7% но наибольший рост AVIF, скорее всего, произойдет в ближайший год, когда он будет постепенно внедряться в веб-браузеры. Оказавшись в веб-браузерах, операторы веб-сайтов смогут быть уверены, что смогут перейти к новому формату на действующих производственных сайтах, что, в свою очередь, еще больше ускорит его распространение.

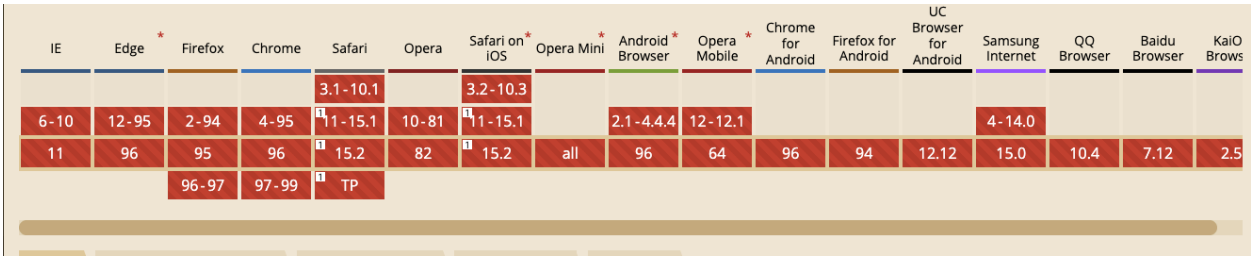
WebP



По данным caniuse, в настоящее время 95,7% браузеров поддерживают формат изображений WebP. Сюда входят Chrome, Firefox и Edge. За последнее десятилетие WebP значительно улучшился, и теперь почти все наиболее часто используемые браузеры поддерживают этот формат. Это большое достижение, потому что теперь больше пользователей могут воспользоваться преимуществами, которые WebP, безусловно, предлагает по сравнению с JPEG. Многие крупные компании, такие как eBay и Facebook, активно используют этот формат изображений, чтобы сэкономить трафик благодаря улучшениям сжатия. Кроме того, поддержка WebP выходит за рамки поддержки только веб-браузеров, поскольку все больше и больше CMS и платформ предоставляют пользователям методы загрузки изображений в формате WebP.

Поскольку WebP может уменьшать размеры изображений, этот формат делает его очень привлекательным для владельцев веб-сайтов, которые хотят как снизить стоимость полосы пропускания, так и повысить удовлетворенность посетителей.

HEIF



На данный момент ни один из браузеров не поддерживает данный формат, однако есть возможность работать с ним через сторонние плагины, что не совсем удобно.

Анализ и сравнение форматов

Установка библиотек:

```
[ ] !pip install pillow-avif-plugin # для работы с форматом AVIF
    !pip install matplotlib==3.5.1 # для визуализации данных
```

Подключение библиотек:

```
from google.colab import drive # для подключения гугл-диска
import cv2 # для работы с изображениями
import glob # для работы с файлами
import matplotlib.pyplot as plt # для построения гистограммы
from matplotlib.patches import Rectangle # для создания легенды
import shutil # для копирования файлов
import numpy as np # для преобразования типов
from PIL import Image # для работы с изображениями
import pillow_avif # для работы с форматов AVIF
import os # для подсчёта веса файлов

# import tensorflow as tf # для оценки качества изображений
# import requests # для запросов
# import time # для работы с временем
# import urllib.request # для загрузки файлов
```

Подключение гугл-диска:

```
[ ] drive.mount('/content/gdrive') # монтирование гугл-диска
```

Подключение изображений:

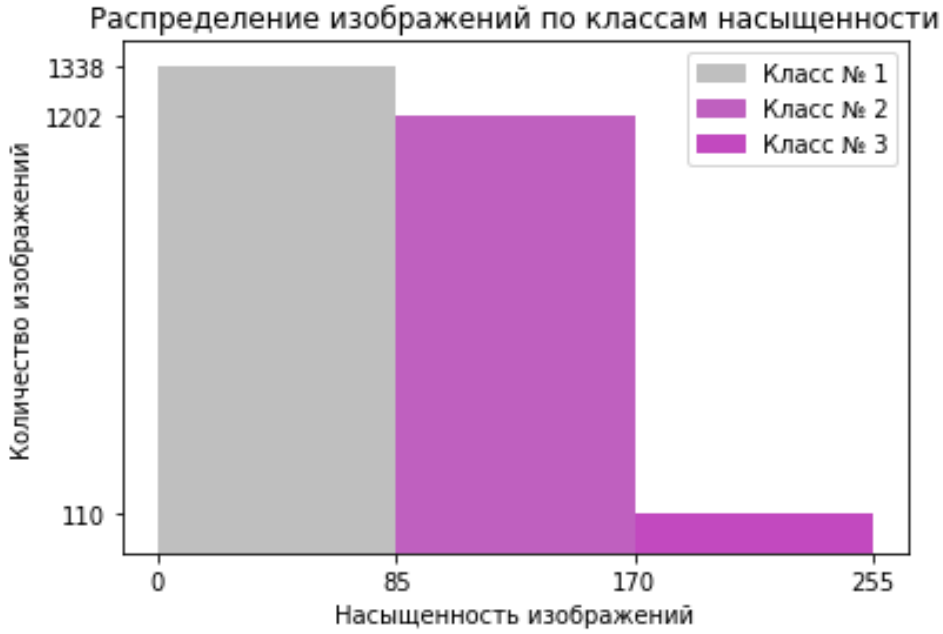
```
[ ] files = glob.glob("gdrive/MyDrive/DATA/*.png") # список изображений
    files.sort() # сортировка изображений по возрастанию

    # files = files[0:20] # ограничение выборки для ускорения работы программы
```

Вычисление насыщенности изображений:

```
data = [] # список для результатов ([наименование файла][насыщенность])

for i in files: # цикл по изображениям

  img = cv2.imread(i) # чтение изображения
  img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # перевод изображения из RGB в HSV
  saturation = img_hsv[:, :, 1].mean() # вычисление насыщенности изображения
  data.append([i, saturation]) # добавление полученных данных в список для результатов

  print(i, '-', saturation) # вывод промежуточного результата
```

Гистрограмма:

```
n = [item[1] for item in data] # список насыщенностей изображений
counts, bins, patches = plt.hist(n, bins = 3, range = (0, 255)) # настройки гистрограммы (3 диапазона в интервале от 0 до 255)

patches[0].set_fc((0.75, 0.75, 0.75)) # цвет класса № 1
patches[1].set_fc((0.75, 0.375, 0.75)) # цвет класса № 2
patches[2].set_fc((0.75, 0, 0.75)) # цвет класса № 3

plt.xticks(bins) # настройка оси X
plt.yticks(counts) # настройка оси Y
plt.title('Распределение изображений по классам насыщенности') # наименование гистограммы
plt.xlabel('Насыщенность изображений') # наименование оси X
plt.ylabel('Количество изображений') # наименование оси Y

handles = [] # список для легенды
handles.append(Rectangle((0, 0), 1, 1, color = (0.75, 0.75, 0.75))) # прямоугольник класса № 1
handles.append(Rectangle((0, 0), 1, 1, color = (0.75, 0.375, 0.75))) # прямоугольник класса № 2
handles.append(Rectangle((0, 0), 1, 1, color = (0.75, 0, 0.75))) # прямоугольник класса № 3

labels= ["Класс № 1", "Класс № 2", "Класс № 3"] # наименования классов

plt.legend(handles, labels) # вывод легенды
plt.show() # вывод гистограммы
```



Выбор 110 изображений на каждый класс насыщенности:

```
low_saturated = glob.glob("gdrive/MyDrive/DATA_CLASSES/Low-saturated PNG/*.png") # список низконасыщенных изображений
medium_saturated = glob.glob("gdrive/MyDrive/DATA_CLASSES/Medium-saturated PNG/*.png") # список средненасыщенных изображений
high_saturated = glob.glob("gdrive/MyDrive/DATA_CLASSES/High-saturated PNG/*.png") # список высоконасыщенных изображений

count_low_saturated = 0 # количество выбранных низконасыщенных изображений
count_medium_saturated = 0 # количество выбранных средненасыщенных изображений
count_high_saturated = 0 # количество выбранных высоконасыщенных изображений

for i in data: # цикл по списку для результатов

  flag = False # было ли выбрано текущее изображение для дальнейшей обработки?
  temp = i[1].astype(np.int64) # насыщенность текущего изображения

  if temp > 0 and temp <= 85 and count_low_saturated < 110: # если насыщенность низкая

    count_low_saturated += 1 # увеличение соответствующего счетчика
    shutil.copy2(i[0], 'gdrive/MyDrive/DATA_CLASSES/Low-saturated PNG/' + str(count_low_saturated) + '.png') # копирования изображения в соответствующую папку
    flag = True # изменение флага

  if temp > 85 and temp < 170 and count_medium_saturated < 110: # если насыщенность средняя

    count_medium_saturated += 1 # увеличение соответствующего счетчика
    shutil.copy2(i[0], 'gdrive/MyDrive/DATA_CLASSES/Medium-saturated PNG/' + str(count_medium_saturated) + '.png') # копирования изображения в соответствующую папк
    flag = True # изменение флага

  if temp >= 170 and temp < 255 and count_high_saturated < 110: # если насыщенность высокая

    count_high_saturated += 1 # увеличение соответствующего счетчика
    shutil.copy2(i[0], 'gdrive/MyDrive/DATA_CLASSES/High-saturated PNG/' + str(count_high_saturated) + '.png') # копирования изображения в соответствующую папку
    flag = True # изменение флага

  if flag: # если текущее изображениевыбрано для дальнейшей обработки

    print((count_low_saturated + count_medium_saturated + count_high_saturated) / 330 * 100, '%') # вывод промежуточного результата

  if (count_low_saturated + count_medium_saturated + count_high_saturated == 330): # если набралось нужное количество данных

    break # остановить цикл
```

Преобразование низконасыщенных исходных изображений в другие форматы:

```
files = glob.glob("gdrive/MyDrive/DATA_CLASSES/Low-saturated PNG/*.png") # список низконасыщенных изображений
files.sort() # сортировка списка
count = 0 # счётчик преобразованных изображений

for i in files: # цикл по низконасыщенным изображениям

  count += 1 # увеличение счётчика

  img = cv2.imread(i) # чтение изображения
  cv2.imwrite('gdrive/MyDrive/DATA_CLASSES/Low-saturated TIFF/' + str(count) + '.tiff', img) # сохранение в формат TIFF
  cv2.imwrite('gdrive/MyDrive/DATA_CLASSES/Low-saturated JPEG/' + str(count) + '.jpeg', img) # сохранение в формат JPEG
  cv2.imwrite('gdrive/MyDrive/DATA_CLASSES/Low-saturated WEBP/' + str(count) + '.webp', img) # сохранение в формат WEBP

  img = Image.open(i) # чтение изображения
  img.save('gdrive/MyDrive/DATA_CLASSES/Low-saturated AVIF/' + str(count) + '.avif','AVIF') # сохранение в формат AVIF

  print(count / 110 * 100, '%') # вывод промежуточного результата
```

Преобразование средненасыщенных исходных изображений в другие форматы:

```
[ ] files = glob.glob("gdrive/MyDrive/DATA_CLASSES/Medium-saturated PNG/*.png") # список средненасыщенных изображений
    files.sort() # сортировка списка
    count = 0 # счётчик преобразованных изображений

    for i in files: # цикл по средненасыщенным изображениям

      count += 1 # увеличение счётчика

      img = cv2.imread(i) # чтение изображения
      cv2.imwrite('gdrive/MyDrive/DATA_CLASSES/Medium-saturated TIFF/' + str(count) + '.tiff', img) # сохранение в формат TIFF
      cv2.imwrite('gdrive/MyDrive/DATA_CLASSES/Medium-saturated JPEG/' + str(count) + '.jpeg', img) # сохранение в формат JPEG
      cv2.imwrite('gdrive/MyDrive/DATA_CLASSES/Medium-saturated WEBP/' + str(count) + '.webp', img) # сохранение в формат WEBP

      img = Image.open(i) # чтение изображения
      img.save('gdrive/MyDrive/DATA_CLASSES/Medium-saturated AVIF/' + str(count) + '.avif','AVIF') # сохранение в формат AVIF

      print(count / 110 * 100, '%') # вывод промежуточного результата
```

Функция подсчёта среднего веса:

```
[ ] sizes = [] # список весов

    def Count_size(folder_path): # функция для подсчёта веса

      size = 0 # начальный вес

      for path, dirs, files in os.walk(folder_path): # цикл к папке

        for f in files: # цикл по файлам

          fp = os.path.join(path, f) # вес файла
          size += os.path.getsize(fp) # суммирование веса

      sizes.append(round(size/1024/110)) # добавление веса в список
```

Преобразование высоконасыщенных исходных изображений в другие форматы:

```
[ ] files = glob.glob("gdrive/MyDrive/DATA_CLASSES/High-saturated PNG/*.png") # список высоконасыщенных изображений
    files.sort() # сортировка списка
    count = 0 # счётчик преобразованных изображений

    for i in files: # цикл по высоконасыщенным изображениям

      count += 1 # увеличение счётчика

      img = cv2.imread(i) # чтение изображения
      cv2.imwrite('gdrive/MyDrive/DATA_CLASSES/High-saturated TIFF/' + str(count) + '.tiff', img) # сохранение в формат TIFF
      cv2.imwrite('gdrive/MyDrive/DATA_CLASSES/High-saturated JPEG/' + str(count) + '.jpeg', img) # сохранение в формат JPEG
      cv2.imwrite('gdrive/MyDrive/DATA_CLASSES/High-saturated WEBP/' + str(count) + '.webp', img) # сохранение в формат WEBP

      img = Image.open(i) # чтение изображения
      img.save('gdrive/MyDrive/DATA_CLASSES/High-saturated AVIF/' + str(count) + '.avif','AVIF') # сохранение в формат AVIF

      print(count / 110 * 100, '%') # вывод промежуточного результата
```

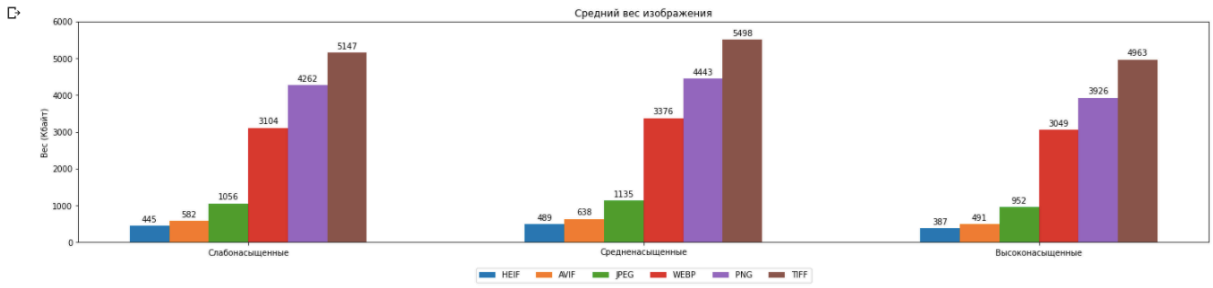
Подсчёт среднего веса изображений:

```
    Count_size('gdrive/MyDrive/DATA_CLASSES/Low-saturated HEIF') # подсчёт среднего веса низконасыщенного HEIF
    Count_size('gdrive/MyDrive/DATA_CLASSES/Medium-saturated HEIF') # подсчёт среднего веса средненасыщенного HEIF
    Count_size('gdrive/MyDrive/DATA_CLASSES/High-saturated HEIF') # подсчёт среднего веса высоконасыщенного HEIF

    Count_size('gdrive/MyDrive/DATA_CLASSES/Low-saturated AVIF') # подсчёт среднего веса низконасыщенного AVIF
    Count_size('gdrive/MyDrive/DATA_CLASSES/Medium-saturated AVIF') # подсчёт среднего веса средненасыщенного AVIF
    Count_size('gdrive/MyDrive/DATA_CLASSES/High-saturated AVIF') # подсчёт среднего веса высоконасыщенного AVIF

    Count_size('gdrive/MyDrive/DATA_CLASSES/Low-saturated JPEG') # подсчёт среднего веса низконасыщенного JPEG
    Count_size('gdrive/MyDrive/DATA_CLASSES/Medium-saturated JPEG') # подсчёт среднего веса средненасыщенного JPEG
    Count_size('gdrive/MyDrive/DATA_CLASSES/High-saturated JPEG') # подсчёт среднего веса высоконасыщенного JPEG

    Count_size('gdrive/MyDrive/DATA_CLASSES/Low-saturated WEBP') # подсчёт среднего веса низконасыщенного WEBP
    Count_size('gdrive/MyDrive/DATA_CLASSES/Medium-saturated WEBP') # подсчёт среднего веса средненасыщенного WEBP
    Count_size('gdrive/MyDrive/DATA_CLASSES/High-saturated WEBP') # подсчёт среднего веса высоконасыщенного WEBP

    Count_size('gdrive/MyDrive/DATA_CLASSES/Low-saturated PNG') # подсчёт среднего веса низконасыщенного PNG
    Count_size('gdrive/MyDrive/DATA_CLASSES/Medium-saturated PNG') # подсчёт среднего веса средненасыщенного PNG
    Count_size('gdrive/MyDrive/DATA_CLASSES/High-saturated PNG') # подсчёт среднего веса высоконасыщенного PNG

    Count_size('gdrive/MyDrive/DATA_CLASSES/Low-saturated TIFF') # подсчёт среднего веса низконасыщенного TIFF
    Count_size('gdrive/MyDrive/DATA_CLASSES/Medium-saturated TIFF') # подсчёт среднего веса средненасыщенного TIFF
    Count_size('gdrive/MyDrive/DATA_CLASSES/High-saturated TIFF') # подсчёт среднего веса высоконасыщенного TIFF
```

Гистограмма средних весов изображений:

```
labels = ['Слабонасыщенные', 'Средненасыщенные', 'Высоконасыщенные'] # подгруппы

HEIF = sizes[0:3] # данные HEIF
AVIF = sizes[3:6] # данные AVIF
JPEG = sizes[6:9] # данные JPEG
WEBP = sizes[9:12] # данные WEBP
PNG = sizes[12:15] # данные PNG
TIFF = sizes[15:18] # данные TIFF

x = np.arange(len(labels)) # определение координат
width = 0.1 # ширина столбца
fig, ax = plt.subplots() # подгруппы гистограммы

rects_1 = ax.bar(x - width * 2.5, HEIF, width, label = 'HEIF') # визуализация данных HEIF
rects_2 = ax.bar(x - width * 1.5, AVIF, width, label = 'AVIF') # визуализация данных AVIF
rects_3 = ax.bar(x - width * 0.5, JPEG, width, label = 'JPEG') # визуализация данных JPEG
rects_4 = ax.bar(x + width * 0.5, WEBP, width, label = 'WEBP') # визуализация данных WEBP
rects_5 = ax.bar(x + width * 1.5, PNG, width, label = 'PNG') # визуализация данных PNG
rects_6 = ax.bar(x + width * 2.5, TIFF, width, label = 'TIFF') # визуализация данных TIFF

ax.set_ylabel('Вес (Кбайт)') # подпись оси Y
ax.set_title('Средний вес изображения') # наименование гистограммы
ax.set_xticks(x, labels) # подписи подгрупп
ax.legend() # построение легенды

ax.bar_label(rects_1, padding = 3) # подписи HEIF
ax.bar_label(rects_2, padding = 3) # подписи AVIF
ax.bar_label(rects_3, padding = 3) # подписи JPEG
ax.bar_label(rects_4, padding = 3) # подписи WEBP
ax.bar_label(rects_5, padding = 3) # подписи PNG
ax.bar_label(rects_6, padding = 3) # подписи TIFF
```

```
ax.bar_label(rects_6, padding = 3) # подписи TIFF

plt.yticks([0, 1000, 2000, 3000, 4000, 5000, 6000]) # шкала оси Y
plt.rcParams["figure.figsize"] = (25, 5) # размер гистограммы
plt.legend(loc = "lower center", ncol = 6, bbox_to_anchor = (0.5, -0.2)) # выравнивание легенды

plt.show() # отображение гистограммы
```



Исходя из полученных результатов можно сделать вывод, что лучший графический формат по весу и качеству является HEIF, но этот формат на данный момент плохо распространен, также это закрытый формат и лицензией владеет компания apple и на данный момент ни один браузер не поддерживает этот формат. Следующий формат после HEIF является AVIF который не имеет этих недостатков, но при этом имеет чуть больший вес изображения и на данный момент имеет большую совместимость со сторонним ПО.

Список литературы

- [1] F. Y. Shih and Y.-T. Wu, “Robust watermarking and compression for medical images based on genetic algorithms,” Information Sciences, vol. 175, no. 3, pp. 200–216, 2005.
- [2] R. Kumar, K. Singh, and R. Khanna, “Satellite image compression using fractional fourier transform,” International Journal of Computer Applications, vol. 50, no. 3, pp. 0975–8887, 2012.
- [3] ГОСТ 34.003-90 в области информационных
- [4] ISO/IEC 10918-1 : 1993(E) p.28.
- [5] Kerr, Douglas A. «Chrominance Subsampling in Digital Images»
- [6] VideoBits.org — Prediction.
- [7] Inside WebM Technology: VP8 Intra and Inter Prediction — The WebM Open Media Project Blog
- [8] http://www.webmproject.org
- [9] "Analysis of the emerging AOMedia AV1 video coding format for OTT use-cases" (PDF). Archived from the original (PDF)
- [10] Mukherjee, Debargha; Su, Hui; Bankoski, Jim; Converse, Alex; Han, Jingning; Liu, Zoe; Xu (Google Inc.), Yaowu (2015), Tescher, Andrew G (ed.), "An overview of new video coding tools under consideration for VP10 – the successor to VP9", SPIE Optical Engineering+ Applications, Applications of Digital Image Processing XXXVIII, International Society for Optics and Photonics, 9599: 95991E, Bibcode:2015SPIE.9599E..1EM, doi:10.1117/12.2191104, S2CID 61317162
- [11] Han, Jingning; Saxena, Ankur; Melkote, Vinay; Rose, Kenneth (29 September 2011). "Jointly Optimized Spatial Prediction and Block Transform for Video and Image Coding" (PDF) 1884. CiteSeerX 10.1.1.367.5662. doi:10.1109/tip.2011.2169976. PMID 21965209. S2CID 9507669. Archived from the original (PDF)

- [12] "Mozilla shares how AV1, the new open source royalty-free video codec, works"
- [13] "Into the Depths:The Technical Details Behind AV1" (PDF).
- [14] Shankland, Stephen (June 16, 2017). "How Apple is squeezing more photos into your iPhone – FAQ: Apple's newest iPhone software attempts to move the world out of the JPEG era". CNET. Archived from the original on 2017-11-16. Retrieved 2017-11-21.
- [15] Shu, Les (September 19, 2017). "Here's what HEIF and HEVC are, and why they'll improve your iPhone with iOS 11". Digital Trends. Archived from the original on 2021-01-22. Retrieved 2017-09-30.