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:

«Облачный сервис по работе научных коллективов»

Студент группы ИУ5-33М _____ Н. А. Казаков

Руководитель _____ Ю. Е. Гапанюк

2020 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 «Облачный сервис по работе научных коллективов»

Студент группы ИУ5-13М

Казаков Никита Андреевич

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. Исследование типов, особенностей, состава, функций облачных СУБД а также их классификация.
2. Изучение возможностей облачных сервисов.
3. Создание базы данных
4. Изучение возможности создания кластера в БД

_____

_____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 14 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

_____

_____

_____

_____

Дата выдачи задания « 08 » сентября 2020 г.

Руководитель НИР _____ Ю. Е. Гапанюк

Студент _____ Н. А. Казаков

СОДЕРЖАНИЕ

ЗАГРУЗКА ГРАФИЧЕСКИХ ФАЙЛОВ В ОБЛАЧНОЕ ХРАНИЛИЩЕ. ОТПИМИЗАЦИЯ.

«Системный анализ форматов графических файлов»

Основная задача: Выяснить самый подходящий формат изображения для съемки на камеру спортивных мероприятий. Рассматриваемые форматы изображений - TIFF, JPEG.

Цель выбрать подходящий формат графического файла именно для спортивных мероприятий – не случайна. Следует учитывать что в современном мире не так популярна ситуация когда вес файла, а также предел заполнения буфера обмена камеры важны, серийная съемка такого мероприятия – один из тех случаев когда данные параметры действительно важны, ведь если выбрать неверный формат изображения то в самый неподходящий момент может слишком долго будет передаваться изображение на стороннее хранилище например для показа ярких моментов с камеры в прямом эфире.

Для исследования задачи будет взят набор фотографий принципиально разных конфигураций, представленных в четырех форматах: JPEG, TIFF (популярные форматы для современных камер). Затем будет произведен эргономический анализ взаимосвязи глубины цвета, его степени сжатия и разрешения. Также будет учтена возможность редактирования полученных изображений

Методы исследования: аналитический (описательный), расчётный (вес файла, степень сжатия и др.)

Работа будет состоять из пяти основных частей:

1. Какой путь преодолевает файл от момента съемки до выхода в эфир
2. Начальные условия по используемым техническим средствами
3. Хранение и передача информации в формате JPEG, TIFF в выбранном мною случае (серийная съемка спортивного мероприятия) Сжатие, глубина цвета, разрешение
4. Расчеты
5. Итог

В заключении будет сводный анализ трех форматов, а также выбор оптимального для моих условий формата.

Путь файла

На основе работы съемочный и редакторской группы можно выделить следующие, важные для моей цели инстанции:

1. Файл появляется на SD карте камеры
2. Далее передается на ноутбук оператора через USB
3. Файл попадает на дисковое пространство компьютера команды редакторов через интернет
4. Файл попадает на сервер, где идет трансляция через интернет

Используемая камера и microSD карта

Журналист, осуществляющий съемку пользуется камерой Nikon D500

Ниже будут приведены технические характеристики данной камеры:

Nikon D500

Сенсор - CMOS-матрица формата DX размером 23,5 x 15,7 мм с разрешением 21,51 Мп и максимальным размером кадра 5568 x 3712. (Дополнительно можно снимать JPEG и TIFF-файлы среднего (4176 x 2784) и малого размера (2784 x 1856);

система очистки матрицы от пыли, эталонные данные для автоматического удаления пыли на постобработке (требуется Capture NX).

Автофокусировка - модуль датчика автофокусировки Multi-CAM 20K с определением фазы TTL, тонкой настройкой и 153 точками фокусировки (включая 99 датчиков

перекрестного типа и 15 датчиков, поддерживающих светосилу f/8), из них 55 доступны для выбора: 35 датчиков перекрестного типа и 9 датчиков с поддержкой светосилы f/8.

Режимы автофокуса - покадровая следящая АФ (AF-S); непрерывная следящая АФ (AF-C); прогнозирующая следящая фокусировка, которая включается автоматически в соответствии с состоянием объекта. При выборе ручной фокусировки (M) возможно использование электронного дальномера.

Экспозамер - 180000-точечный датчик, 3D цветовой матричный замер, центровзвешенный, точечный, по ярким участкам.

ISO - от 100 до 51200 единиц, есть возможность программного расширения до 50 единиц или до величин H 0.3, 0.5, 0.7, 1, 2, 3, 4 и 5 (эквивалентно 1.640.000)

Диапазон выдержек - от 1/8000 до 30 с, с шагом 1/3, 1/2 или 1 EV, выдержка от руки, выдержка по таймеру, X250.

Режимы съемки - S (покадровый), CL (непрерывный низкоскоростной), CH (непрерывный высокоскоростной), Q (тихий спуск затвора), Автоспуск, MUP (подъем зеркала), Qc (тихий непрерывный спуск затвора).

Объективы - байонет F Nikon (с сопряжением АФ и контактами АФ).

Стабилизация изображения - в камере отсутствует.

Видоискатель - зеркальный прямой видоискатель с пентапризмой, со полным (100 процентов) покрытием кадра, диоптрийная настройка от - 3 до +1.

Дисплей - сенсорный, с поддержкой multi touch, диагональ 8 см с разрешением 2 359 000 точек.

Форматы фото - 12-bit и 14-bit RAW, доступна съемка RAW полного (5568 x 3712), среднего (4176 x 2784) или малого размера (2784 x 1856), TIFF (RGB), JPEG с высоким, средним или низким сжатием. RAW+JPEG.

Серийная съемка - однократная, непрерывная до 10 кадров/сек со следящей фокусировкой, отложенный спуск с задержкой 2, 5, 10 или 20 секунд и возможностью съемки серии от 1 до 9 кадров с интервалом 0,5, 1, 2 или 3 секунды.

Интервальная съемка - да, гибко настраиваемая.

Видеосъемка - 4K до 30 кадров/сек, Full HD до 60 кадров/сек.

Интерфейсы - SuperSpeed USB (разъем USB Micro-B 3.0), HDMI типа C, аудио вход и выход 3,5 мм, 10-контактный разъем дистанционного управления, центральный контакт.

Карты памяти - два слота памяти: один XQD и один SD, SDHC (с поддержкой интерфейса UHS-II), SDXC (с поддержкой интерфейса UHS-II).

Питание - батарея EN-EL15.

Размеры - приблизительно 147 x 115 x 81 мм.

Вес - приблизительно 760 г без аксессуаров, аккумулятора, карт памяти и крышек, около 860 г с батареей и двумя картами памяти CF.

Из важных для целей моего исследования характеристик стоит отметить то что в данной камере присутствуют интересующие меня форматы фотографий, а именно – TIFF и JPEG.

Так же из интересного – 3 вида разрешений – максимальное 5568 x 3712, среднее 4176 x 2784 и малое 2784 x 1856, которые тоже нужно будет проанализировать, ведь разрешение напрямую влияет на вес файла и на оперативность последующей его передачей в студию соответственно.

Еще стоит отметить присутствие режима съемки CL и CH – непрерывный низкоскоростной и высокоскоростной соответственно и наличие SuperSpeed USB.

Можно также было бы обратить внимание на доступные хранилища в камере, однако в нашем случае данный вопрос явно не в приоритете, ведь в мире где уже существуют SD карты на 128 и более гигабайт врятли можно представить что место на дисковом пространстве закончится. Тем более наш вопрос скорее о передаче и о скорости. Значит данную характеристику можно во внимание не брать.

Однако стоит отметить что выбор microSD также карты важен для скорости передачи файлов. Так как стоимость камеры во много раз превосходит стоимости даже очень дорогой microSD карты, то здесь экономить смысла нет. Тогда журналист и группа редакторов разумеется предпочтут качественную и подходящую карту:

SanDisk Extreme PRO SD UHS-II

Тип карты: SDXC

Емкость: до 128 ГБ

Скорость чтения: до 300 МБ / с

Скорость записи: до 260 МБ / с

Подходит для: профессионалов, снимающих быстрые снимки с высоким разрешением и видео

Тип карты – SDXC, что важно – он поддерживается нашей камерой.

Параметры ноутбука оператора, компьютера редактора и интернета

На подробных характеристиках ноутбука и компьютера останавливаться смысла для нас нет, но следует отметить что ноутбук и компьютер оснащены мощными SSD дисками. Компьютер и ноутбук подключены к оптоволоконному интернет соединению, скорость которого составляет 1000 Мбит/с. Тарифы с более высокой скоростью сейчас крайне редки, так что ограничимся этим. На SSD дисках подробнее останавливаться тоже не стоит, так как скорость чтения/записи файлов любого SSD в несколько раз выше чем скорость нашего интернета. Так что на этих этапах упор будет именно в нее.

Графические файлы

JPEG

JPEG расшифровывается как «Объединённая группа фото-экспертов» (Joint Photographic Expert Group), и он, как предполагает название, был специально разработан для хранения фотографических изображений. Он стал также стандартным форматом для хранения изображений в цифровых камерах и показа фотографий на интернет-сайтах. Файлы JPEG существенно меньше сохраняемых в TIFF, однако ценой такой экономии является использование сжатия с потерями. Великая сила файлов JPEG состоит в их гибкости. Формат JPEG по сути является набором параметров, которые могут быть настроены под нужды отдельно взятого изображения.

Формат JPEG достигает малого размера файла, сжимая изображение с помощью метода, который сохраняет наиболее значимые детали и теряет детали, оценённые как менее влияющие визуально. JPEG осуществляет это, пользуясь тем фактом, что человеческий глаз замечает вариации яркости больше, чем вариации цвета. Степень достигаемой компрессии тем самым весьма зависит от содержания изображения; высокошумные или мелкодетальные изображения непросто сжать, тогда как картинки с мягким небом и небольшой текстурой будут сжаты очень хорошо.

Полезно также получить визуальное представление о том, как различные степени сжатия влияют на качество вашего изображения. На 100% вы вряд ли заметите какие бы то ни было отличия между сжатым и несжатым изображением ниже, если они вообще есть. Обратите внимание, как алгоритм JPEG приоритизирует яркие высококонтрастные границы за счёт более тонких текстур. По мере снижения качества, алгоритм JPEG вынужден приносить в жертву всё более и более заметные текстуры, чтобы продолжать снижать размер файла.

TIFF

Формат TIFF стал стандартом для хранения изображений с большой глубиной цвета. Обычные нам 8 бит на канал для него далеко не предел: TIFF поддерживает и 16, и даже 48 бит. Кроме того, TIFF умеет хранить маски, слои, дополнительные каналы (это может быть использовано для хитрого цветоделения, например, в два пантона). Нередко ошибочно воспринимается как формат сжатия без потерь. В большинстве случаев это действительно так, но вообще-то TIFF — это контейнер для изображения, упакованного определённым алгоритмом. Алгоритм может быть как с потерями, так и без: LZW, ZIP, JPEG (!).

Итак: JPEG — отличная штука для записи финальных файлов, если степень сжатия минимальна. Алгоритм настолько хорош, что размер макета уменьшается, как правило, до десяти раз без какого-либо ухудшения качества! Тем не менее, для записи промежуточных файлов нужно применять только алгоритмы, обеспечивающие компрессию без потерь. Ну а если в макете используются высококонтрастные изображения), лучше записывать их как TIFF без компрессии. То же самое касается картинок с нестандартным цветоделением.

Глубина цвета

Далее следуют примеры изображений, рассмотрим разницу в глубине цвета. Глубина цвета на изображениях немного утрированна, но так нагляднее понимать насколько она важна.

Пример изображения с низкой глубиной цвета (8 бит на канал) – максимально поддерживаемый форматом JPEG

Такое же изображение но уже с высокой глубиной цвета (16 бит на канал) – JPEG уже такую “битность” не поддерживает, а вот TIFF - запросто

Разница в “сочности” очевидна на глаз, однако не стоит забывать что файлы TIFF весят в среднем в несколько раз больше чем файлы JPEG. Насколько такие жертвы применимы для наших условий – предстоит нам выяснить позже – в расчетах

Сжатие

Далее приведена таблица зависимости веса файла от его качества в камере Nikon D500. Большой размер изображения соответственно - 5568 x 3712, средний - 4176 x 2784, маленький - 2784 x 1856. “Кач.” В таблице означает степень сжатия выс. – 1к4, сред. – 1к8, низ. – 1к16

Качество изображения	Размер изображения	Размер файла [1]
TIFF (RGB)	Большой	62,5 МБ
	Средний	35,6 МБ
	Маленький	16,4 МБ
JPEG выс. кач. [3]	Большой	10,4 МБ
	Средний	6,4 МБ
	Маленький	3,4 МБ
JPEG сред. кач. [3]	Большой	5,3 МБ
	Средний	3,3 МБ
	Маленький	1,8 МБ
JPEG низ. кач. [3]	Большой	2,8 МБ
	Средний	1,8 МБ
	Маленький	1,0 МБ

Скриншот из документации к этой камере

К сожалению не удалось найти изображения спортивной тематики с разной степенью сжатия но я приведу в пример изображения другой тематики:

1к4

1к8

1к16

Как можно наблюдать разница в качестве сжатия не такая однозначная как в случае с глубиной цвета, тем более в условиях быстрого показа в эфире, когда в глаза скорее бросается блеклость цветов а не минимальная потеря качества в случае сжатия. Можно сделать предварительный вывод что глубина цвета важнее. Но стоит помнить что хоть большое сжатие практически незаметно, выигрыш в объема файла – также не такой большой как между TIFF и любым файлом JPEG.

Таким образом можно рассмотреть 12 вариантов выбора формата/сжатия/разрешения по логике таблицы приведенной выше, для них и будет произведен расчет

Конкретные временные рамки, расчеты

Итак, телеканалом перед съемочной группой была поставлена цель –максимум за 15 секунд 4 фотографии оператора из 8 должны попадать в эфир, произведем расчет для каждого из 12 вариантов. 4 фото которые попадут в эфир выбирает редактор. Но для начала стоит написать общую формулу

$T1 = t1 + t2 + t3 + t4$

T1 – общее время, проходящее от момента съемки до момента появления фотографий в эфире

t1 – время затрачиваемое на передачу фотографий с камеры на ноутбук

t2 – время затрачиваемое на передачу фотографий с ноутбука на компьютер редактора

t3 – время затрачиваемое на передачу фотографий на сервер где ведется эфир

t4 – время задержек, оправданное работой людей принимаем его равным 9 секундам

t1 = A1/S (Размер файлов/скорость интернета) Взята скорость интернета так как скорость интернета меньше чем скорость USB 3.0. Интернет – 125 Мб/с USB 3.0 – 625 Мб/с

t2 = (2*A1)/S Объем удваивается так как файлы сначала загружаются на google disk и затем с него выгружаются

t3 = 0,5*A1/S Объем уменьшен в два раза так как редактор выбирает 4 из 8 фото

t4 = 10с Время на работу людей (человеческий фактор)

A1 = 8*a1 (объем 8 изображений)

Формулу можно сократить, так как в знаменателе всегда будет и таже скорость – скорость интернета – 125 МБ/с, так как скорость дисков и передающих интерфейсов в несколько раз больше. Получаем:

T1 = 3,5*8*a1/S = 28*a1/S + 10с

T1(JPEG, низкое разрешение, сжатие 1к16) = 10,224с

T2(JPEG, среднее разрешение, сжатие 1к16) = 10,4032с

T3(JPEG, высокое разрешение, сжатие 1к16) = 10,6272с

T4(JPEG, низкое разрешение, сжатие 1к8) = 10,4032с

T5(JPEG, среднее разрешение, сжатие 1к8) = 10,7392с

T6(JPEG, высокое разрешение, сжатие 1к8) = 11,1872с

T7(JPEG, низкое разрешение, сжатие 1к4) = 10,7616с

T8(JPEG, среднее разрешение, сжатие 1к4) = 11,4336с

T9(JPEG, высокое разрешение, сжатие 1к4) = 12,3296с

T10(TIFF, низкое разрешение) = 13,6736

T11(TIFF, среднее разрешение) = 17,9744с

T12(TIFF, высокое разрешение) = 24с

Вывод

Итак, самые привлекательные форматы – последние 4, однако TIFF среднего и высокого разрешения не проходят по времени. В итоге остаются два наиболее привлекательных формата – JPEG высокого разрешения со сжатием 1к4 и TIFF низкого разрешения.

Остановится стоит на варианте T10 (TIFF, низкое разрешение) Так как в предыдущее исследование мы закончили выводом о том что повышенная цветность лучше выглядит на экране телевизоров тем более не смотря на низкое разрешение – оно все равно больше разрешения самого популярного экрана 1920x1080. Разрешение TIFF низкого разрешения в нашем случае 2784x1856 что гораздо больше. Так что JPEG 5568x3712 высокого разрешения в нашем случае не особо актуален. Разница во времени не существенна - меньше 1,5 секунд

Задача выполнена – наш формат TIFF с разрешением 2784x1856

АНАЛИЗ ВЕСА ИЗОБРАЖЕНИЙ В РАЗНЫХ ГРАФИЧЕСКИХ ФОРМАТАХ.

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
```

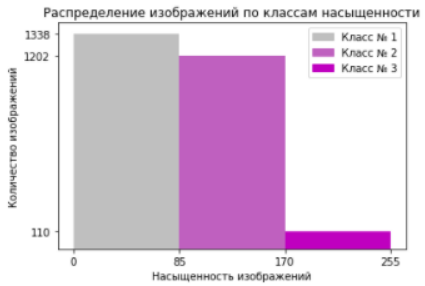
Вычисление насыщенности изображений:

```
[ ]  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 папку
        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
```

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
```

Преобразование исходных изображений в формат HEIF через API:

```
[ ] # response = requests.post('http://api.convertio.co/convert', data = "{\"apikey\": \"4b910f8b8b138dbe30a618190ea7e054\", \"input\": \"upload\", \"outputformat\": \"heif\"}") # запрос на конвертацию
    # id = response.json()['data']['id'] # получение ID конвертации

    # response = requests.put('http://api.convertio.co/convert/' + id + '/' + str(count) + '.png', data = open('gdrive/MyDrive/DATA_CLASSES/Low-saturated PNG/' + str(count) + '.png', 'rb')) # запрос на выгрузку исходного изображения

    # finish = False # завершено

    # while (not finish): # пока не завершено

      # response = requests.get('http://api.convertio.co/convert/' + id + '/status') # запрос на получение статуса конвертации

      # if (response.json()['data']['step'] == 'finish'): # если конвертация завершена

        # finish = True # завершено
        # urllib.request.urlretrieve(response.json()['data']['output']['url'], "gdrive/MyDrive/DATA_CLASSES/Low-saturated HEIF/" + str(count) + ".heif") # запрос на загрузку результата

      # else: # иначе

        # time.sleep(0.25) # подождать 0.25 секунды
```

Функция подсчёта среднего веса:

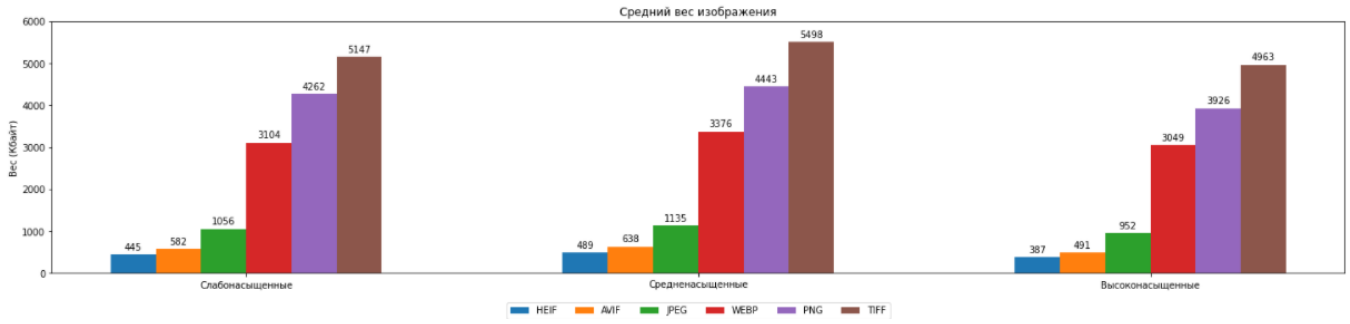
```
[ ]  sizes = [] # список весов

     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
```

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

     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
```



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
```