МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»  
(Самарский университет)

Институт информатики и кибернетики

Кафедра технической кибернетики

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ

Дисциплина: «Технологии программирования на Python»

по программе бакалавриата по направлению подготовки   
01.03.02 Прикладная математика и информатика,

профиль «Компьютерные науки»

Выполнил: И.В. Якухин

Группа: 6403-010302D

Проверил: Н.В. Головастиков

Самара 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc217084941)

[1 Архитектура проекта 6](#_Toc217084942)

[1.1 Уровень абстракции и интерфейсов (interfaces/) 6](#_Toc217084943)

[1.2 Модели предметной области (implementation/dog\_image.py) 6](#_Toc217084944)

[1.3 Уровень вычислительных стратегий (implementation/) 7](#_Toc217084945)

[1.4 Уровень координации и асинхронного управления (dog\_image\_processing.py) 7](#_Toc217084946)

[1.5 Инфраструктурный слой и конфигурация 7](#_Toc217084947)

[2 Внутренняя реализация 9](#_Toc217084948)

[2.1 Асинхронное программирование (asyncio, aiohttp, aiofiles) 9](#_Toc217084949)

[2.2 Многопроцессорность (multiprocessing) 9](#_Toc217084950)

[2.3 JIT-компиляция и оптимизация (Numba, NumPy) 10](#_Toc217084951)

[2.4 Объектно-ориентированные паттерны 10](#_Toc217084952)

[2.5 Механизмы надежности (logging, unittest, dotenv) 10](#_Toc217084953)

[2.6 Инфраструктура сборки (Setuptools, Pipenv) 11](#_Toc217084954)

[3 Инструкция по установке и запуску приложения 12](#_Toc217084955)

[3.1 Предварительные требования и установка 12](#_Toc217084956)

[3.2 Запуск приложения и доступные параметры 12](#_Toc217084957)

[3.3 Пример работы и визуализация результатов 12](#_Toc217084958)

[3.4 Запуск модульных тестов 15](#_Toc217084959)

[3.5 Сборка, дистрибуция и управление жизненным циклом пакета 15](#_Toc217084960)

[ЗАКЛЮЧЕНИЕ 17](#_Toc217084961)

[ПРИЛОЖЕНИЕ А 18](#_Toc217084962)

[Ссылка на GitHub 18](#_Toc217084963)

[Список зависимостей проекта 18](#_Toc217084964)

ВВЕДЕНИЕ

Настоящий цикл лабораторных работ посвящен разработке комплексного программного решения на языке Python для автоматизированной обработки медиаданных и анализа статистической информации. В процессе выполнения работ реализуется переход от написания базовых алгоритмов «с нуля» к созданию масштабируемого приложения с использованием современных парадигм программирования. Итоговый проект представляет собой законченный программный продукт, прошедший полный цикл от проектирования архитектуры до автоматизированного тестирования и сборки.

**Цель работы:** Создание многофункционального консольного приложения для потоковой обработки изображений, с интеграцией внешних API, использованием асинхронных технологий и обеспечением промышленного стандарта качества кода.

**Осваиваемые компетенции и технологии:**

* **Алгоритмическая база:** реализация математических операций свертки, методов выделения границ (Собель) и углов (Харрис) без использования готовых библиотек.
* **Архитектура и ООП:** проектирование системы классов с инкапсуляцией логики, применением декораторов, дескрипторов и перегрузкой операторов.
* **High Performance:** оптимизация работы приложения через асинхронный ввод-вывод (asyncio, aiohttp) и распараллеливание вычислений на CPU (multiprocessing).
* **Анализ данных:** построение конвейеров обработки данных (пайплайнов) на базе генераторов для анализа CSV.
* **DevOps и QA:** использование Git, внедрение системы логирования, покрытие кода Unit-тестами и сборка приложения.

**Задачи и требования к проекту (согласно заданиям)**

На основе методических указаний проект должен отвечать следующим четким требованиям:

**Функциональные требования по обработке изображений:**

* Реализовать **самостоятельно** (без OpenCV/Pillow для вычислений): операцию свертки, перевод в Grayscale, гамма-коррекцию, оператор Собеля и детектор Харриса.
* Обеспечить интерфейс командной строки для запуска функций через ключи (например, python imageprocessing.py sobel image.png).
* Организовать автоматическое сохранение результатов в структурированные папки с использованием модуля os и именованием файлов по шаблону [номер]\_[порода]\_[тип].png.

**Требования к работе с данными и сетью:**

* Реализовать интеграцию с API (TheCatAPI/TheDogAPI) через библиотеку requests (в синхронной версии) и aiohttp (в асинхронной).
* Ключи доступа (API\_KEY) должны храниться в файлах .env и быть исключены из репозитория.
* Разработать систему анализа CSV-файлов, где каждый этап (чтение, фильтрация, агрегация) является **отдельным генератором**, объединенным в единый пайплайн.
* Обеспечить визуализацию результатов анализа (Bar plot, Line plot, доверительные интервалы) средствами matplotlib.

**Требования к архитектуре и производительности:**

* Код должен быть инкапсулирован в классах: CatImage (данные и методы обработки) и CatImageProcessor (логика API и управления).
* Внедрить **асинхронность** для скачивания/сохранения файлов и **мультипроцессорность** для выполнения ресурсоемких операций свертки.
* Добавить логирование процесса: уровень DEBUG в файл app.log (подробный ход работы) и уровень INFO в консоль (краткий статус).

**Требования к качеству и оформлению:**

* Полное отсутствие дублирования кода, соблюдение стандартов PEP8 (проверка линтером) и наличие документации к функциям.
* Наличие модуля тестов tests на базе unittest, проверяющего корректность обработки изображений и работу с API.
* Корректная структура Git-репозитория: ветвление по работам, наличие .gitignore, отсутствие мусорных файлов.
* Подготовка пакета к импортированию.

1. Архитектура проекта

Архитектура приложения организована иерархически и разделена на несколько функциональных уровней: уровень абстракции (интерфейсы), уровень предметной области (модели данных), уровень реализации алгоритмов и уровень управления (процессинг).

* 1. Уровень абстракции и интерфейсов (interfaces/)

В основе системы лежит строгое определение контрактов. Модуль i\_image\_processing.py содержит абстрактный базовый класс IImageProcessing, созданный с использованием модуля abc.

* **Назначение:** гарантирует, что любой движок обработки изображений (будь то самописный или библиотечный) будет иметь идентичный набор методов: \_convolution, edge\_detection, corner\_detection, circle\_detection и др.
* **Преимущество:** это позволяет реализовать паттерн «Стратегия», когда программа может на лету переключаться между различными способами обработки, не меняя основной код приложения.
  1. Модели предметной области (implementation/dog\_image.py)

Этот слой отвечает за представление данных и инкапсуляцию состояния объектов. Используется иерархия классов для работы с различными типами медиаконтента:

* **DogImage (ABC):** Абстрактный корень иерархии. Хранит общие атрибуты: массив пикселей (numpy.ndarray), породу собаки и URL источника.
* **ColorDogImage и GrayscaleDogImage:** Конкретные реализации для трехканальных (RGB) и одноканальных изображений.
* **Перегрузка операторов:** одной из архитектурных особенностей является реализация магических методов \_\_add\_\_ и \_\_sub\_\_. Это позволяет выполнять сложные операции (например, наложение маски или вычитание фона) с помощью естественного синтаксиса image1 + image2, скрывая внутри логику выравнивания размерностей и нормализации значений пикселей.
  1. Уровень вычислительных стратегий (implementation/)

Здесь сосредоточена логика низкоуровневых манипуляций с данными. Разделение выполнено на два независимых модуля:

* **ImageProcessing:** «ручная» реализация. Архитектурно важна интеграция с **Numba**. С помощью декораторов @njit функции обработки массивов компилируются в машинный код, что позволяет обходить медлительность стандартных циклов Python при выполнении свертки.
* **Cv2ImageProcessing:** адаптер для библиотеки OpenCV. Используется как эталонная реализация для валидации результатов и обеспечения высокой скорости в стандартных задачах.
  1. Уровень координации и асинхронного управления (dog\_image\_processing.py)

Класс **DogImageProcessor** выступает в роли управляющего комопнента. Он координирует работу всех остальных модулей:

* **Асинхронный сетевой слой:** использует aiohttp для неблокирующих запросов к API.
* **Управление процессами:** архитектура предусматривает разделение задач на IO-bound (сеть, диск) и CPU-bound (математика). Для обработки используется ProcessPoolExecutor, который распределяет объекты DogImage по разным ядрам процессора.
* **Интеграция с логированием:** класс обернут декораторами (например, async\_time\_logger), которые автоматически фиксируют время выполнения каждого этапа в app.log.
  1. Инфраструктурный слой и конфигурация

Проект включает в себя развитую инфраструктуру для поддержки жизненного цикла разработки:

* **Логирование (log\_conf.py):** централизованная настройка через RotatingFileHandler. Архитектура логгера разделена на подробный файл (уровень DEBUG) и краткую консоль (уровень INFO).
* **Тестирование (tests/):** система покрыта модульными тестами. Архитектурно тесты разделены на проверку состояния моделей (test\_dog\_image.py) и проверку поведения процессора с использованием Mock-объектов (test\_dog\_image\_processor.py).
* **Конфигурация:** использование .env для API-ключей и pyproject.toml для сборки пакета подчеркивает следование современным стандартам дистрибуции Python-приложений.

1. Внутренняя реализация

Программный комплекс «Dog Image Processor» представляет собой инженерное решение, объединяющее низкоуровневую математическую обработку сигналов с современной высоконагруженной архитектурой на базе асинхронных технологий и параллельных вычислений.

* 1. Асинхронное программирование (asyncio, aiohttp, aiofiles)

Приложение относится к категории систем с интенсивным вводом-выводом (IO-bound) на этапе сбора данных.

* **Почему это использовано:** Традиционные синхронные запросы блокируют выполнение программы, заставляя процессор простаивать в ожидании ответа от сервера. Использование asyncio позволяет приложению инициировать десятки запросов к Dog API одновременно.
* **Роль в приложении:** **aiohttp** обеспечивает неблокирующее скачивание изображений.
  + **aiofiles** гарантирует, что сохранение тяжелых массивов данных на диск не «заморозит» событийный цикл (event loop), позволяя параллельно обрабатывать следующие файлы.
  + Это сокращает общее время работы программы в несколько раз по сравнению с последовательным подходом.
  1. Многопроцессорность (multiprocessing)

В отличие от загрузки, обработка изображений (фильтры, поиск кругов) — это вычислительно сложная задача (CPU-bound).

* **Почему это использовано:** Из-за ограничений GIL (Global Interpreter Lock) в Python, потоки (threading) не могут выполняться параллельно на разных ядрах процессора. Использование ProcessPoolExecutor позволяет обойти это ограничение.
* **Роль в приложении:** Тяжелые методы, такие как \_convolution (свертка) или Hough Circles (преобразование Хафа), делегируются отдельным дочерним процессам. Это позволяет задействовать все доступные ядра CPU, распределяя нагрузку и предотвращая перегрев одного потока выполнения.
  1. JIT-компиляция и оптимизация (Numba, NumPy)

Для «ручной» реализации алгоритмов без использования готовых библиотек остро встает вопрос производительности циклов Python.

* **Почему это использовано:** Вложенные циклы по пикселям изображения в чистом Python работают крайне медленно. **Numba** компилирует эти функции в машинный код прямо во время первого запуска (Just-In-Time).
* **Роль в приложении:** Декоратор @njit в модуле image\_processing.py превращает математические расчеты свертки в высокопроизводительный код, сопоставимый по скорости с реализациями на C++. **NumPy** при этом выступает фундаментом, предоставляя эффективные структуры данных для хранения матриц пикселей.
  1. Объектно-ориентированные паттерны

Архитектура кода опирается на продвинутые возможности языка для обеспечения чистоты и гибкости.

* **Перегрузка операторов (\_\_add\_\_, \_\_sub\_\_):** В классе DogImage реализованы методы, которые позволяют буквально «складывать» изображения. Это не просто визуальный эффект — внутри происходит автоматическая нормализация весов и выравнивание размерностей массивов.
* **Интерфейсы (abc.ABC):** Использование абстрактных классов в i\_image\_processing.py реализует паттерн «Мост» или «Стратегия». Приложение может прозрачно для пользователя переключаться между модулем ImageProcessing (собственные алгоритмы) и Cv2ImageProcessing (библиотека OpenCV).
  1. Механизмы надежности (logging, unittest, dotenv)

Для перехода от учебного скрипта к полноценному пакету внедрены инструменты контроля качества:

* **logging.handlers.RotatingFileHandler:** В приложении реализована ротация логов. Это критично для асинхронных программ, генерирующих большое количество событий: старые логи сжимаются и удаляются, предотвращая переполнение диска.
* **unittest.IsolatedAsyncioTestCase:** Специализированные тесты для асинхронного кода позволяют симулировать задержки сети и проверять корректность работы DogImageProcessor в условиях конкурентного доступа.
* **python-dotenv:** Роль этой библиотеки — безопасность. Она исключает попадание чувствительных данных (API-ключей) в систему контроля версий, загружая их из локального окружения.
  1. Инфраструктура сборки (Setuptools, Pipenv)
* **Роль:** Файлы pyproject.toml и Pipfile управляют зависимостями и процессом упаковки. Благодаря им проект можно собрать в .whl или .tar.gz архив и установить в любую систему одной командой, что завершает концепцию создания полноценного программного продукта.

1. Инструкция по установке и запуску приложения

Данный раздел описывает процесс развертывания проекта из исходного кода и примеры его использования через интерфейс командной строки (CLI).

* 1. Предварительные требования и установка

Для работы приложения необходим интерпретатор **Python версии 3.11+** и менеджер зависимостей **Pipenv**.

1. **Клонирование и переход в проект:**

git clone https://github.com/BruhMano/6403yakuhiniv

cd dog\_image\_processor

1. **Установка зависимостей:** В проекте используются Pipfile и Pipfile.lock для фиксации версий библиотек (numpy, aiohttp, numba, opencv-python).

pipenv install

pipenv shell

1. **Настройка переменных окружения:** Создайте файл .env в корневой директории и добавьте ваш персональный ключ от TheDogAPI:

API\_KEY=your\_secret\_key\_from\_dog\_api

* 1. Запуск приложения и доступные параметры

Запуск осуществляется через модуль main.py. Приложение поддерживает следующие аргументы:

* -l (--limit): Количество изображений для обработки
* -odir (--output\_dir): Путь к директории для сохранения результатов.
* -grey: Флаг, активирующий обработку в одноканальном режиме.

**Пример команды для массовой обработки:**

python main.py -l 5 -odir "res " -grey

* 1. Пример работы и визуализация результатов

При запуске приложение инициализирует логгер и начинает асинхронный цикл. Пример вывода прогресса в консоль приведен на рисунке 1.

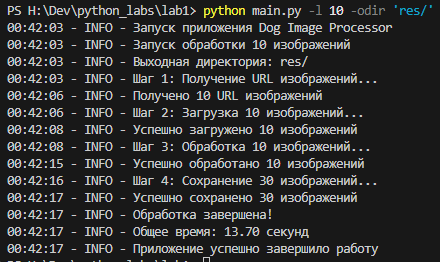


Рисунок 1 – Вывод коротких логов в консоль

**Этапы выполнения в логах:**

1. **Fetch URLs:** Запрос к API для получения ссылок на изображения.
2. **Async Download:** Параллельная загрузка файлов в оперативную память.
3. **Multiprocess Processing:** Распределение изображений по ядрам CPU для выполнения свертки и детекции границ (с указанием PID процесса).
4. **Async Save:** Запись обработанных матриц в PNG-файлы.

**Структура выходных данных:** В указанной директории (например, results/) создается иерархия файлов. Для каждой найденной собаки сохраняется три изображения: исходное, обработанное «ручным» алгоритмом и результат работы OpenCV. Примеры входного и обработанных изображений приведены на рисунках 2, 3, 4.



Рисунок 2 – Исходное изображение



Рисунок 3 – Изображение, обработанное «ручным» алгоритмом



Рисунок 4 – Изображение обработанное OpenCV

* 1. Запуск модульных тестов

Для проверки целостности системы после установки рекомендуется запустить встроенные тесты:

python run\_tests.py

Скрипт выполнит тесты из папки tests/, проверяя корректность работы асинхронного процессора и математическую точность перегруженных операторов в классе DogImage. При успешном завершении вы увидите сообщение OK. Скриншот с запуском тестов представлен на рисунке 5.

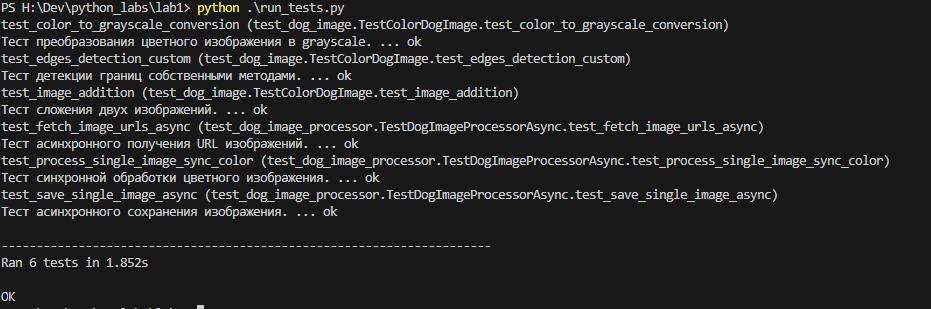


Рисунок 5 – Скриншот запуска тестов

* 1. Сборка, дистрибуция и управление жизненным циклом пакета

Завершающим этапом внутренней реализации является приведение проекта к формату готового программного продукта, пригодного для установки и использования в изолированных окружениях.

**Алгоритм сборки и установки:**

* 1. **Сборка:** для генерации дистрибутива используется модуль build.

python -m build

Командаанализирует pyproject.toml и создает в директории dist/ архивы, содержащие все необходимые модули (implementation, interfaces) и ресурсы.

* 1. **Установка:** пакет устанавливается стандартным менеджером через команду

pip install dist/\*.whl

При этом автоматически разрешаются и скачиваются все сторонние зависимости, указанные в секции dependencies.

* 1. **Запуск скомпилированного скрипта:** при сборке проекта создается исполняемый файл dog-processor.exe, ктороый можно запустить аналогично main.py с теми же аргументами командной строки.

**Пример запуска программы:**

dog-processor -l 5 -odir "res" -grey

или

python -m dog\_image\_processor -l 5 “res”

Скриншот запуска скомпилированной программы представлен на рисунке 6.

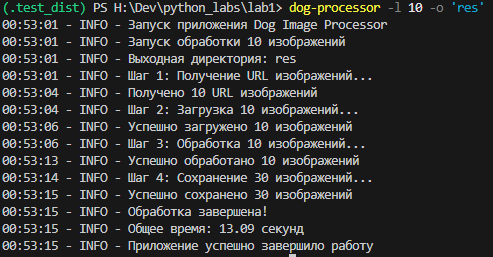


Рисунок 6 – Скриншот запуска скомпилированной программы

ЗАКЛЮЧЕНИЕ

В ходе выполнения цикла лабораторных работ было спроектировано и реализовано полнофункциональное асинхронное приложение «Dog Image Processor», предназначенное для автоматизированного сбора и анализа медиаданных. Все поставленные в начале работы цели были достигнуты в полном объеме, а архитектура приложения приведена в строгое соответствие с требованиями методических указаний.

В рамках реализации функциональности было создано консольное решение, обеспечивающее полный цикл обработки: от взаимодействия с внешними REST API и неблокирующей загрузки данных до проведения сложных математических манипуляций с изображениями и их последующего сохранения. Проект построен на фундаментальных принципах объектно-ориентированного программирования с использованием абстрактных интерфейсов, что позволило интегрировать как собственные низкоуровневые алгоритмы обработки сигналов, так и стандартные библиотечные решения. Приложение полностью отвечает промышленным стандартам разработки, так как включает в себя механизмы ротируемого логирования, автоматизированное модульное тестирование и систему управления конфигурациями через переменные окружения.

Особое внимание в процессе работы было уделено изучению и внедрению высокопроизводительных технологий. Использование асинхронного подхода и многопроцессорности позволило эффективно разделить сетевые и вычислительные задачи, значительно повысив общую скорость работы системы. Применение JIT-компиляции для оптимизации критических участков кода позволило достичь высокой производительности при выполнении ручных операций свертки и детекции объектов. Итоговым этапом стала успешная подготовка дистрибутива и его публикация в репозитории TestPyPI, что подтверждает готовность продукта к эксплуатации и дистрибуции. Полученные в ходе разработки навыки сформировали надежную базу для создания сложных масштабируемых систем на языке Python.

ПРИЛОЖЕНИЕ А

Ссылка на GitHub

https://github.com/BruhMano/6403yakuhiniv

Список зависимостей проекта

* numpy = "\*"
* aiohttp = "\*"
* aiofiles = "\*"
* matplotlib = "\*"
* opencv-python = "\*"
* dotenv = "\*"
* numba = "\*"