

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 лабораторной работе №1

Дисциплина: «Технологии искусственного интеллекта»

Тема: «**Docker. Сборка OpenCV**»

Выполнил: Каспаров И.А.

Группа: 6232-010402D

Самара 2024

Задание 1

На базе образа [Ubuntu 22.04](#) создать Docker контейнер со сборкой [OpenCV](#) с non free contrib модулями (пример из лекции). Рабочее сочетание версий (можно выбрать посвежее):

- Версия Убунты: 22.04
- OpenCV: 4.8.0
- CUDA: 12.2

Данные для сборки:

- скрипт сборки [build.sh](#)
 - скрипт [build_env.sh](#) для задания определенных переменных среды (environment vars)
- докер файл [OpenCVDockerFile](#)

1. Отредактировать скрипт сборки [build.sh](#), заменить значения:

- image_tag – название тега

Дабы избежать пересечения в тегах, имеет смысл добавить уникальный префикс, пусть будет имя учетной записи: stud<N>_<что угодно>, где <N> – номер учетной записи на сервере.

- build_thread_count – количество потоков для сборки библиотеки

лучше указать $n-1$, где n – количество *физических* ядер CPU.

- Версии Ubuntu, OpenCV, CUDA при желании

Скрипт [build_env.sh](#) использовать как есть, он необходим для установки питонячих путей для компиляции OpenCV и последующей установки библиотеки.

2. С CUDA возможны различные приколы при установке, особенно на старые релизы типа 18.04. Если в системе нет GPU Nvidia, то установку CUDA можно вырезать из скрипта сборки и докер файла.

На сервере GPU есть. Компиляция с использованием CUDA опциональна.

3. Изменить права доступа, выдать разрешение для запуска скриптов build.sh, build_env.sh:

4. `chmod +x build.sh`

`chmod +x build_env.sh`

5. Дописать в конец [докер файла](#) (перед CMD) команды для установки необходимых либ Python 3 при необходимости.

Кроме opencv-python и opencv-contrib-python!

6. Запустить build.sh для сборки контейнера.

7. Реализовать [алгоритм обработки изображений](#), скрипт на питоне положить в папку на хосте.

8. Запустить контейнер командой:

`docker run -v <путь на хосте>:<путь внутри контейнера> -it <имя тега>`

9. Запустить скрипт с реализованным алгоритмом в контейнере в примонтированной внутри контейнера папке. Результат обработки сохранить в локальной директории контейнера.

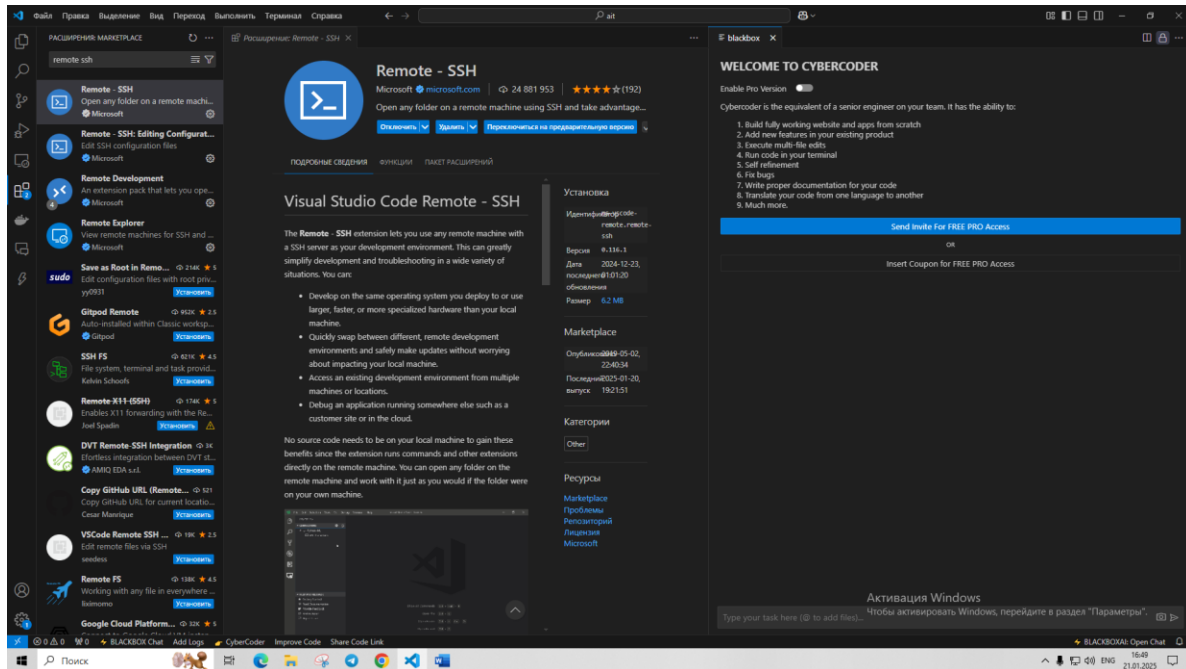
10. Убедиться в появлении результата в директории хоста (на сервере).

Содержание

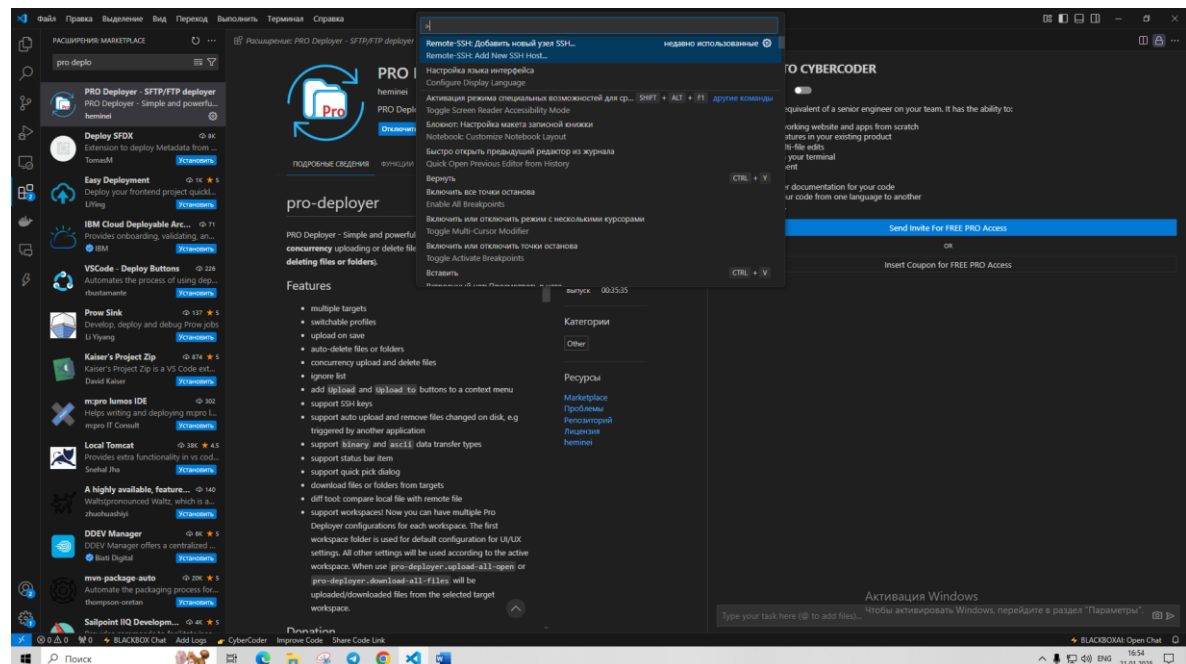
Шаг 1. Настройка SSH и установка плагинов в Visual Studio Code.....	5
Шаг 2. Загрузка файлов с GitHub	8
Шаг 3. Выгрузка исходного изображения на сервер	9
Шаг 4. Редактирование докерфайла	10
Шаг 5. Редактирование файла build.sh	11
Шаг 6. Выдача прав и просмотр содержимого data.....	12
Шаг 7. Запуск build.sh для сборки контейнера.....	13
Шаг 8. Алгоритм обработки изображений	14
Шаг 9. Запуск контейнера, запуск скрипта с реализованным алгоритмом и проверка результата	17
Шаг 10. Просмотр изображений	18
Способ с помощью Pro Deployer	21
Заключение	27

Шаг 1. Настройка SSH и установка плагинов в Visual Studio Code

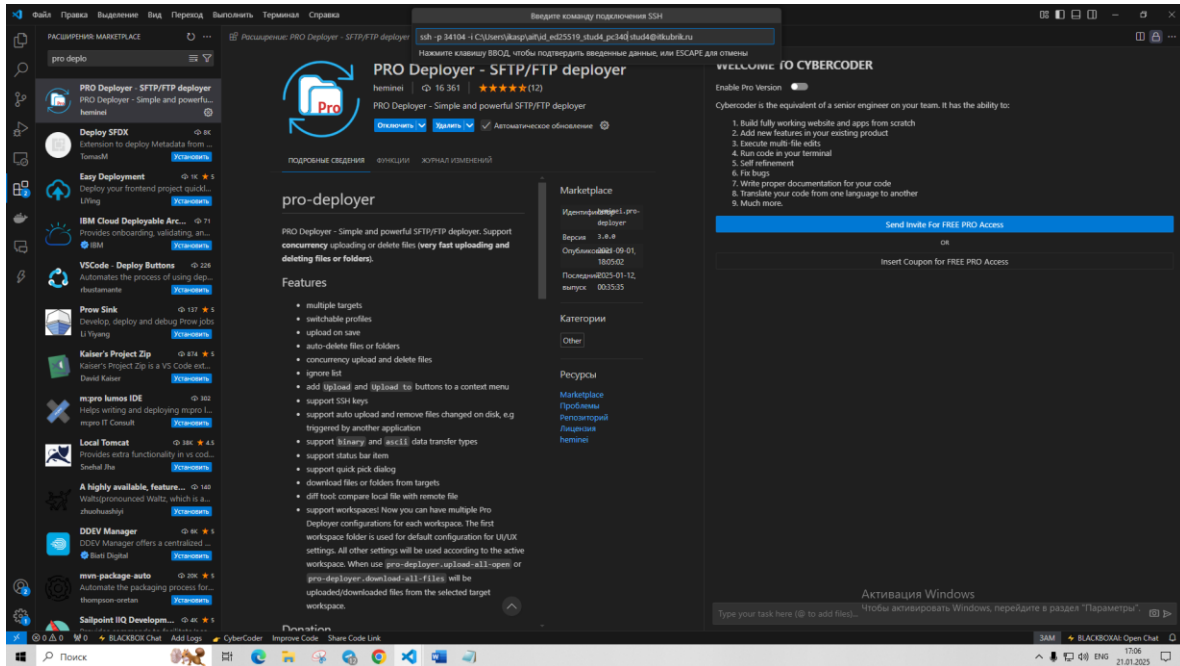
1) Установка расширения Remote - SSH:



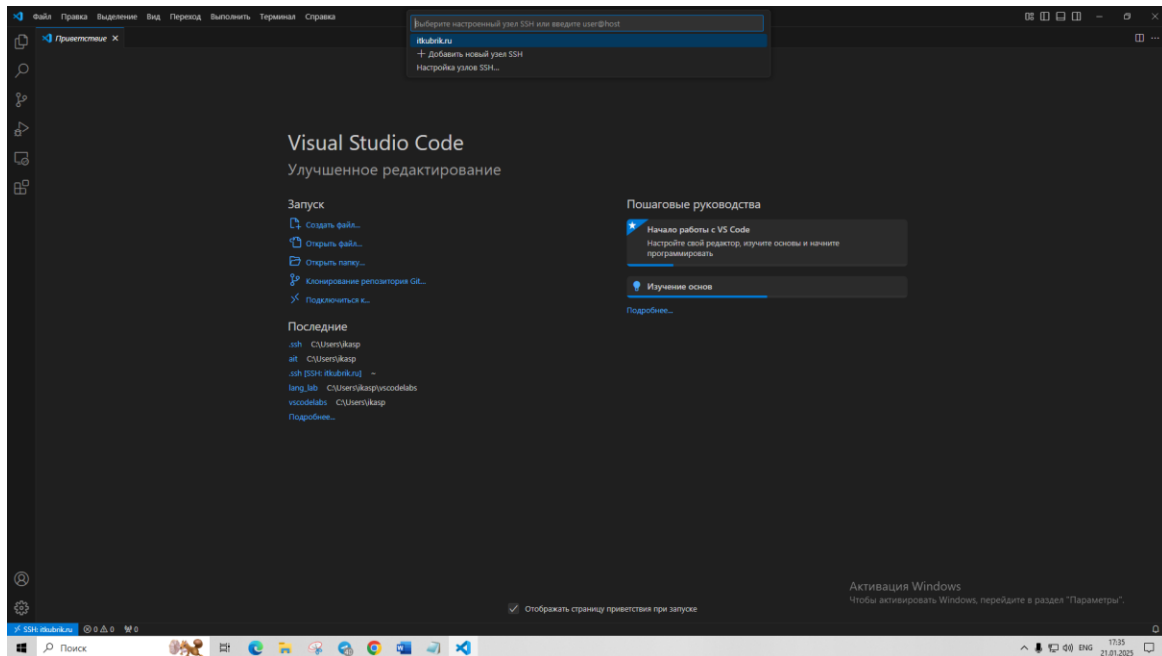
2) Добавление SSH-конфигурации:



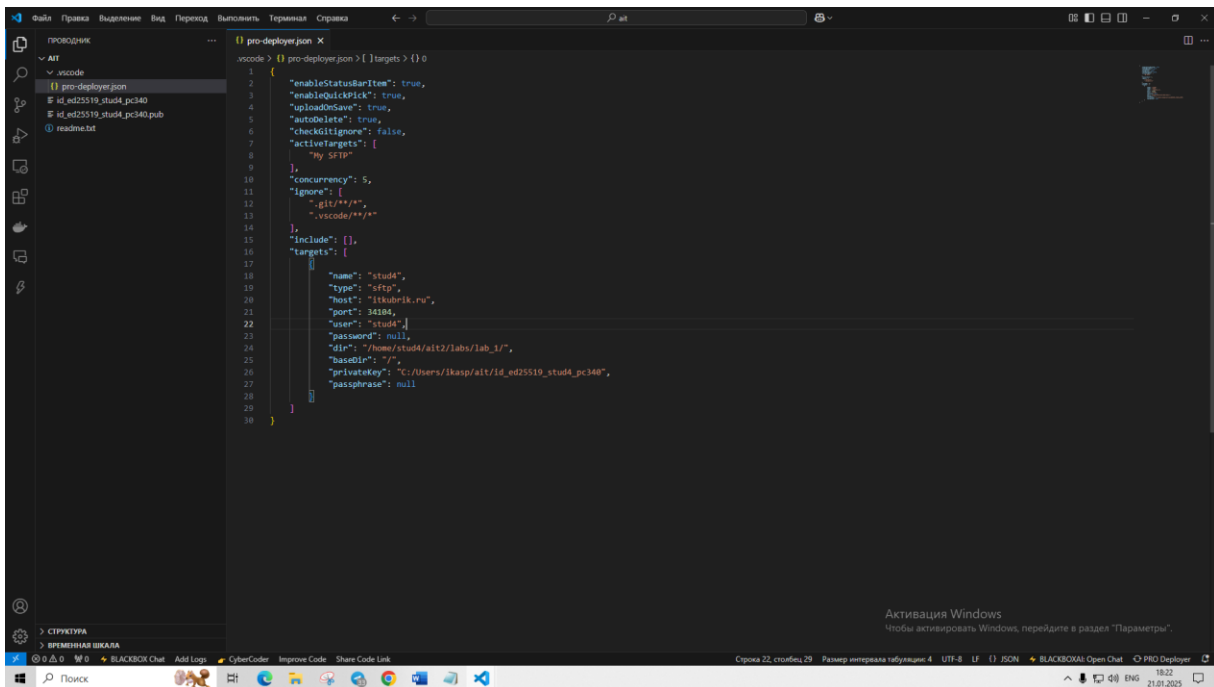
3) Add New SSH Host.



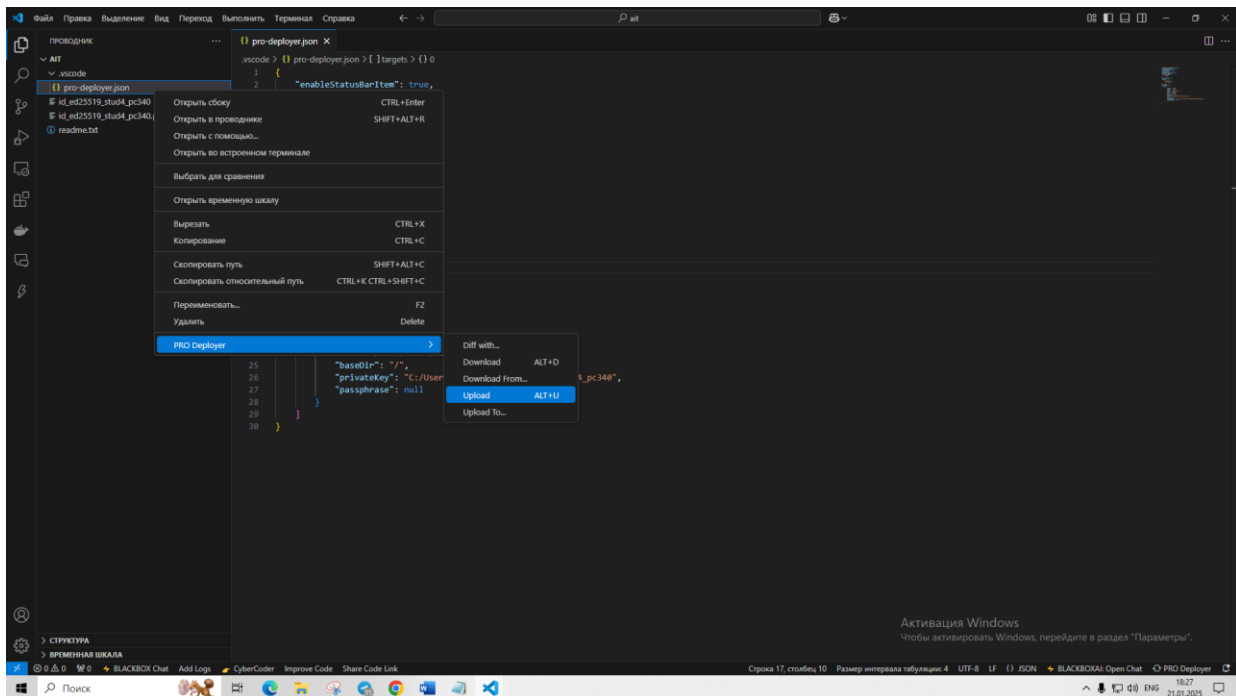
4) Подключение к серверу:



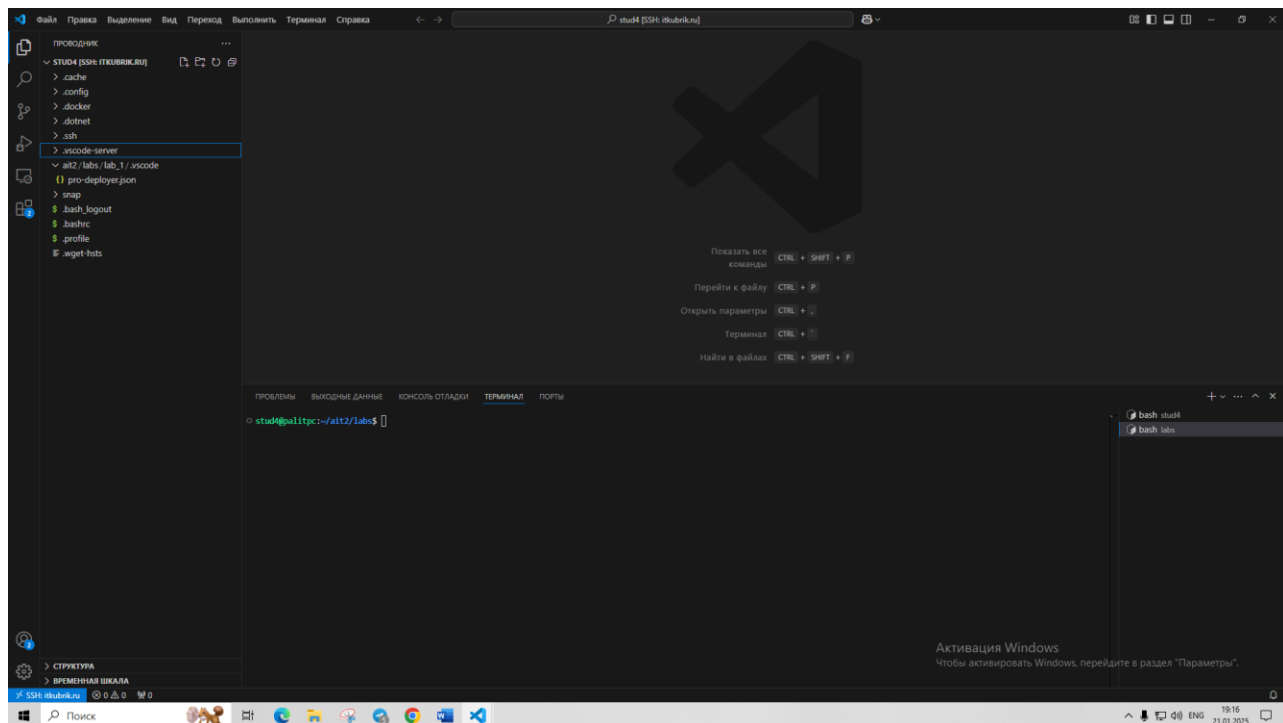
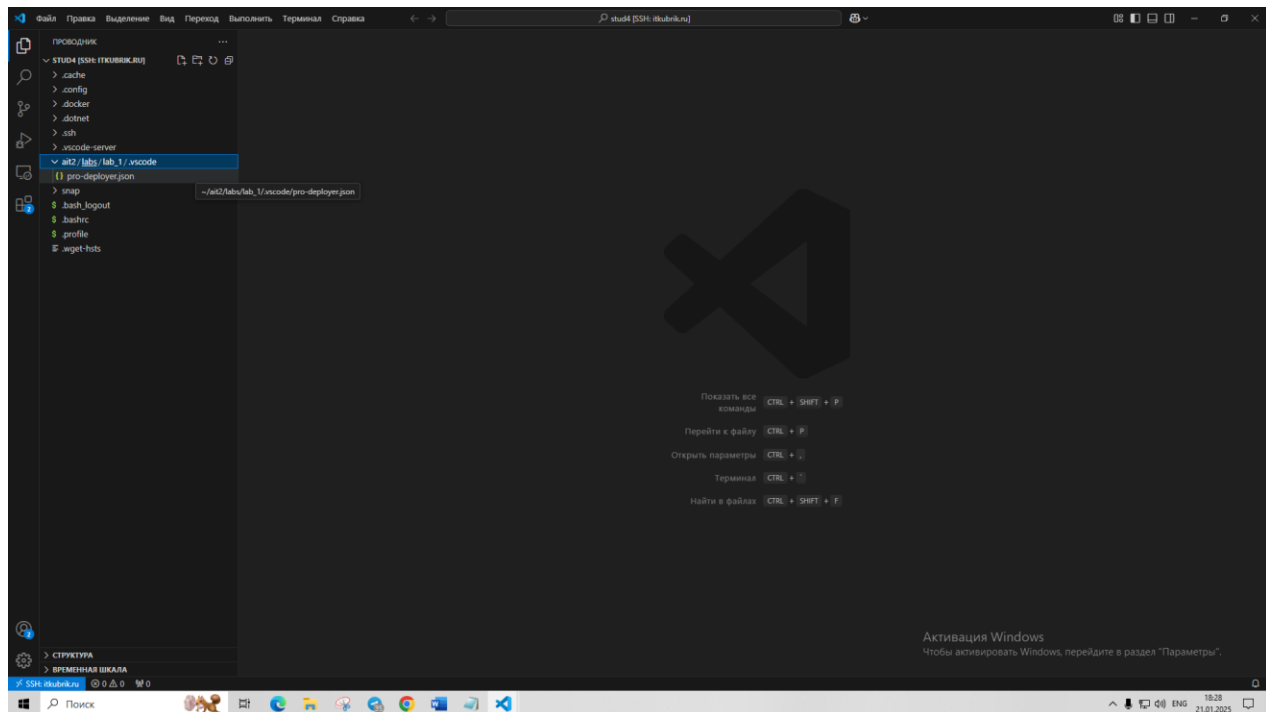
5) Делаем Pro Deployer: Generate Config File



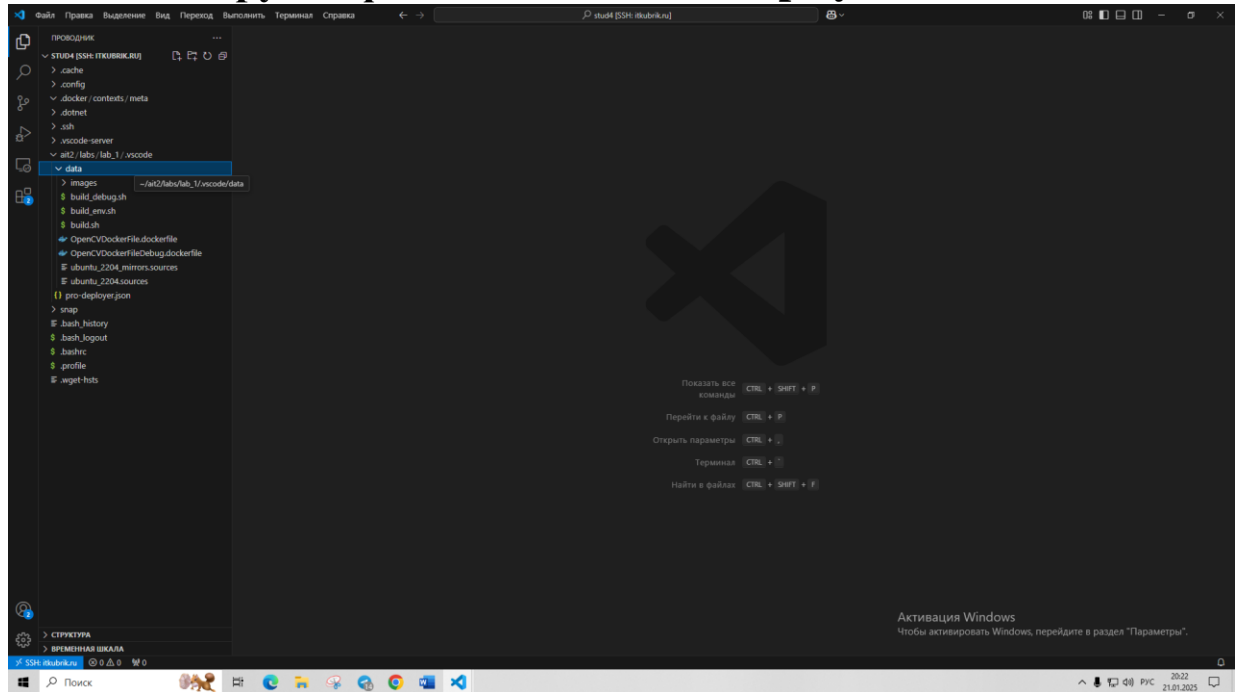
6) Делаем Pro Deployer: Upload



7) Проверка наличия файла на сервере в папке /home/stud4/ait2/labs/lab_1



Шаг 2. Загрузка файлов с помощью Deployer



Шаг 3. Выгрузка исходного изображения на сервер

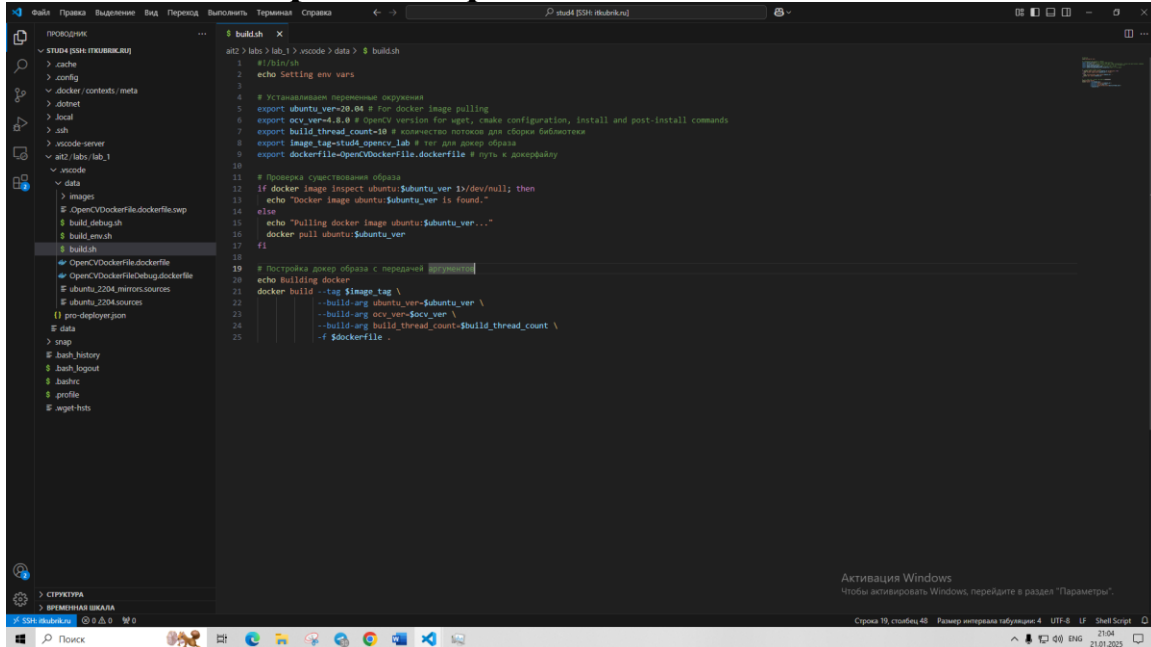
```
Windows PowerShell
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS C:\Users\ikasp> scp -P 34104 "C:\Users\ikasp\ait\car.jpg" stud4@itkubrik.ru:/home/stud4/ait2/labs/lab_1/data
no such identity: C:\Users\ikasp\ait_ed25519_stud4_pc340: No such file or directory
no such identity: \342\200\252C:\Users\ikasp\ait_ed25519_stud4_pc340: No such file or directory
stud4@itkubrik.ru's password:
Permission denied, please try again.
stud4@itkubrik.ru's password:
car.jpg
100% 181KB 39.5KB/s 00:04

PS C:\Users\ikasp>
```

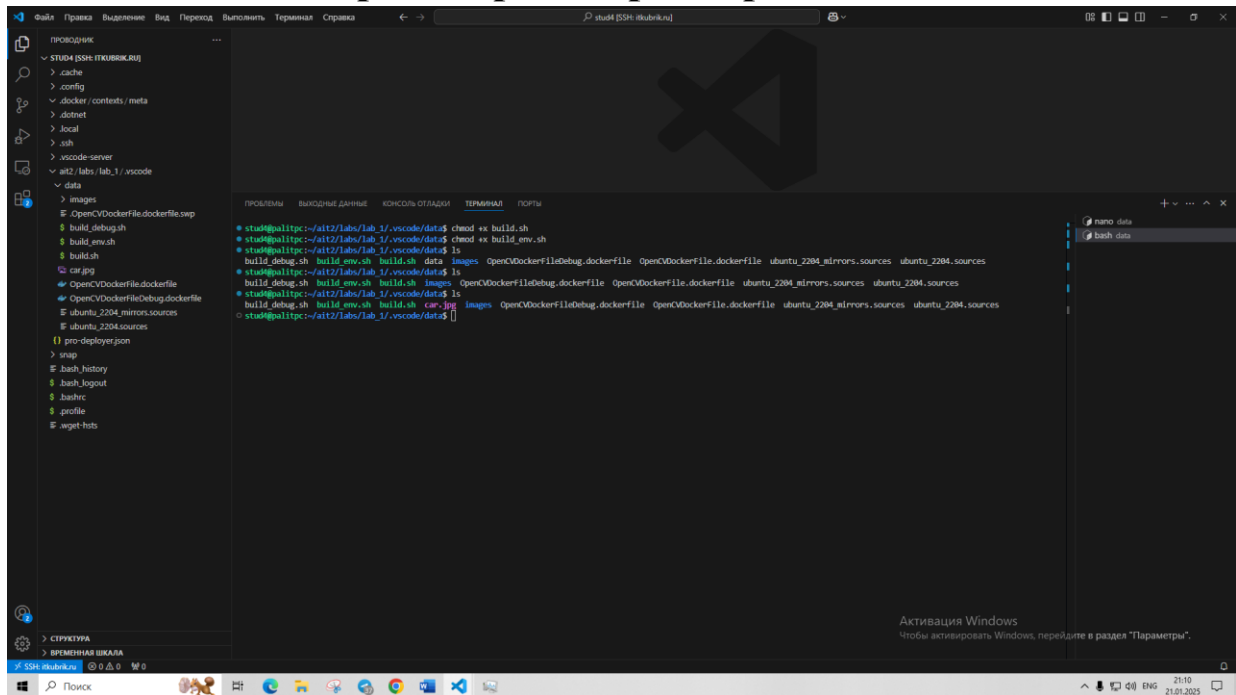
Шаг 4. Редактирование файла build.sh



The screenshot shows the VS Code editor with the file explorer on the left displaying the project structure. The main editor window shows the `build.sh` file with the following content:

```
1 #!/bin/sh
2 echo Setting env vars
3
4 # Установка переменных окружения
5 export ubuntu_ver=20.04 # for docker image pulling
6 export soc_ver=4.0.0 # socu version for agent, make configuration, install and post-install commands
7 export build_thread_count=10 # количество потоков для сборки бинарников
8 export image_tag=stud4_opency_lab # ver для динер образа
9 export dockerfile=OpenCVDockerfile.dockerfile # путь к dockerfile
10
11 # Проверка существования образа
12 if docker image inspect ubuntu:$ubuntu_ver 1>/dev/null; then
13     echo "Docker image ubuntu:$ubuntu_ver is found."
14 else
15     echo "Pulling docker image ubuntu:$ubuntu_ver..."
16     docker pull ubuntu:$ubuntu_ver
17 fi
18
19 # Настройка динер образа с помощью make
20 echo Building docker
21 docker build --tag $image_tag \
22     --build-arg ubuntu_ver=$ubuntu_ver \
23     --build-arg soc_ver=$soc_ver \
24     --build-arg build_thread_count=$build_thread_count \
25     -f $dockerfile .
```

Шаг 6. Выдача прав и просмотр содержимого data



The screenshot shows the VS Code editor with the file explorer on the left displaying the project structure. The main editor window shows the terminal output of the following commands:

```
stud4@allitpc:~/at2/labs/lab_1/.vscode/data$ chmod +x build.sh
stud4@allitpc:~/at2/labs/lab_1/.vscode/data$ chmod +x build_env.sh
stud4@allitpc:~/at2/labs/lab_1/.vscode/data$ ls
build_debug.sh  build_env.sh  build.sh  data  images
stud4@allitpc:~/at2/labs/lab_1/.vscode/data$ ls
build_debug.sh  build_env.sh  build.sh  images  OpenCVDockerfileDebug.dockerfile  OpenCVDockerfile.dockerfile  ubuntu_2204_mirrors.sources  ubuntu_2204.sources
stud4@allitpc:~/at2/labs/lab_1/.vscode/data$ ls
build_debug.sh  build_env.sh  build.sh  car.jpg  images  OpenCVDockerfileDebug.dockerfile  OpenCVDockerfile.dockerfile  ubuntu_2204_mirrors.sources  ubuntu_2204.sources
```

Шаг 7. Запуск build.sh для сборки контейнера

```
stud4@palitpc:~/ait2/labs/lab_1/.vscode/data$ ./build.sh
Setting env vars
Docker image ubuntu:22.04 is found.
Building docker
[+] Building 0.1s (31/31) FINISHED
=> [internal] load build definition from OpenVContainerFile.dockerfile 0.0s
=> transferring dockerfile: 3.25kB 0.0s
=> [internal] load metadata for docker.io/library/ubuntu:22.04 0.0s
=> [internal] load .dockerignore 0.0s
=> transferring context: 28 0.0s
=> [internal] load build context 0.0s
=> transferring context: 34B 0.0s
[ 1/27] FROM docker.io/library/ubuntu:22.04 0.0s
=> CACHED [ 2/27] RUN apt update && apt -y upgrade 0.0s
=> CACHED [ 3/27] RUN mkdir /usr/local/Dev 0.0s
=> CACHED [ 4/27] RUN apt install -y curl python3-testresources python3-wget gnupg2 software-properties-common 0.0s
=> CACHED [ 5/27] WORKDIR /usr/local/Dev/ 0.0s
=> CACHED [ 6/27] RUN curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py && python3 get-pip.py 0.0s
=> CACHED [ 7/27] RUN apt -y install build-essential cmake unzip git pkg-config libgtk2.0-dev libavcodec-dev libavformat-dev 0.0s
=> CACHED [ 8/27] RUN wget -O opencv.zip https://github.com/opencv/opencv/archive/4.8.0.zip 0.0s
=> CACHED [ 9/27] RUN wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.8.0.zip 0.0s
=> CACHED [10/27] RUN unzip opencv.zip 0.0s
=> CACHED [11/27] RUN unzip opencv_contrib.zip 0.0s
=> CACHED [12/27] RUN apt -y install mlocate && updatedb 0.0s
=> CACHED [13/27] COPY build_env.sh /usr/local/Dev 0.0s
=> CACHED [14/27] RUN chmod +x /usr/local/Dev/build_env.sh 0.0s
=> CACHED [15/27] RUN cd /usr/local/Dev/ && ./build_env.sh 0.0s
=> CACHED [16/27] WORKDIR /usr/local/Dev/opencv-4.8.0 0.0s
=> CACHED [17/27] RUN mkdir build 0.0s
=> CACHED [18/27] WORKDIR /usr/local/Dev/opencv-4.8.0/build 0.0s
=> CACHED [19/27] RUN rm -rf /usr/local/Dev/opencv-4.8.0/build/* 0.0s
=> CACHED [20/27] RUN pip3 install numpy 0.0s
=> CACHED [21/27] RUN ./usr/local/Dev/build_env.sh && cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local/OpenCV-4.8.0 -D OPENCV_SKIP_PYTHON_LOADER=OFF -D OPENCV_PYTHONX_INSTALL_PATH=/ 0.0s
=> CACHED [22/27] RUN make -j10 0.0s
=> CACHED [23/27] RUN make install 0.0s
=> CACHED [24/27] RUN ldconfig 0.0s
=> CACHED [25/27] RUN ./usr/local/Dev/build_env.sh && ln -sf /usr/local/OpenCV-4.8.0/lib/python${py3_ver_mm}/site-packages/cv2/python${py3_ver_mm}/${ls /usr/local/OpenCV-4.8.0/lib/python${py3_ver_mm}/site-pack 0.0s
=> CACHED [26/27] RUN echo ${python3} -c "import cv2 as cv; print(cv.__version__)" 0.0s
=> exporting to image 0.0s
=> exporting layers 0.0s
=> writing image sha256:b43e8133e96af94adc46849b69d192938ad2d93315970f71d3c33c8f995ad34 0.0s
=> naming to docker.io/library/stud4_opencv_lab 0.0s

What's Next?
1. Sign in to your Docker account + docker login
2. View a summary of image vulnerabilities and recommendations + docker scout quickview
stud4@palitpc:~/ait2/labs/lab_1/.vscode/data$
```

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Шаг 8. Алгоритм обработки изображений

```
python3.py
def process_image(gamma=1.0, brightness_factor=1.0, contrast_factor=1.0):
    # Загрузка изображения
    image = cv2.imread(input_path)

    # Применение гамма-коррекции
    image_gamma_corrected = np.power(image / 255.0, 1 / gamma) * 255.0

    # Преобразование в целочисленный формат и ограничение значений
    image_gamma_corrected = np.clip(image_gamma_corrected, 0, 255).astype(np.uint8)

    # Сохранение изображения после гамма-коррекции
    gamma_output_name = f"{output_path}bits_gamma(gamma).jpg"
    cv2.imwrite(gamma_output_name, image_gamma_corrected)
    print(f"Image after gamma saved as: {gamma_output_name}")

    # Преобразование в оттенки серого
    gray = cv2.cvtColor(image_gamma_corrected, cv2.COLOR_BGR2GRAY)

    # Сохранение изображения после гамма-коррекции в оттенках серого
    gray_output_name = f"{output_path}bits_gamma(gamma).gray.jpg"
    cv2.imwrite(gray_output_name, gray)
    print(f"Image after gamma and grayscale saved as: {gray_output_name}")

    # Применение контрастности
    img_array2 = contrast_factor * (image_gamma_corrected - 128) + 128

    # Ограничиваем значения в диапазоне [0, 255]
    img_after_contrast = np.clip(img_array2, 0, 255).astype(np.uint8)

    # Сохранение изображения после гамма-коррекции и контрастности
    contrast_output_name = f"{output_path}bits_gamma(gamma)_contrast(contrast_factor).jpg"
    cv2.imwrite(contrast_output_name, img_after_contrast)
    print(f"Image after gamma and contrast saved as: {contrast_output_name}")

    # Применение яркости
    img_array3 = img_array2 * brightness_factor

    # Ограничиваем значения в диапазоне [0, 255]
    img_after_brightness = np.clip(img_array3, 0, 255).astype(np.uint8)

    # Сохранение изображения после гамма-коррекции, контрастности и яркости
    brightness_output_name = f"{output_path}bits_gamma(gamma)_contrast(contrast_factor)_brightness(brightness_factor).jpg"
    cv2.imwrite(brightness_output_name, img_after_brightness)
    print(f"Image after gamma, contrast, and brightness saved as: {brightness_output_name}")
```

2. View a summary of image vulnerabilities and recommendations + docker scout quickview

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Код представляет собой скрипт для обработки изображений с использованием библиотеки OpenCV и библиотеки NumPy. Он демонстрирует применение нескольких преобразований к изображению, включая гамма-коррекцию, изменение контрастности, яркости и преобразование в оттенки серого.

Этот скрипт реализует алгоритм обработки изображений, выполняющий следующие шаги:

- Загрузка изображения:

Используется функция `cv2.imread()` для загрузки изображения по пути, указанному в переменной `input_path`.

- Гамма-коррекция:

Преобразует интенсивности пикселей, чтобы компенсировать эффект нелинейности яркости.

Применяется формула

$$I_{out} = \left(\frac{I_{in}}{255}\right)^{\frac{1}{\gamma}} \cdot 255$$

где γ (гамма) регулирует степень коррекции.

Результирующее изображение сохраняется в файл.

- Преобразование в оттенки серого:

Используется функция `cv2.cvtColor()` с параметром `cv2.COLOR_BGR2GRAY` для получения черно-белого изображения.

Результат сохраняется отдельно.

- Изменение контрастности:

Реализуется линейной трансформацией интенсивностей пикселей по формуле:

$$\text{output} = \text{contrast_factor} \times (\text{input} - 128) + 128$$

Значения пикселей ограничиваются диапазоном [0, 255].

- Изменение яркости:

Применяется масштабирование интенсивностей пикселей с использованием коэффициента яркости:

$$\text{output} = \text{brightness_factor} \times \text{input}$$

Также используется ограничение значений в диапазоне [0, 255].

- Сохранение результатов:

Результаты на каждом этапе сохраняются в файлы в формате JPEG. Пути сохраняемых файлов формируются с использованием входных параметров (gamma, контрастность и яркость).

Шаг 9. Запуск контейнера, запуск скрипта с реализованным алгоритмом и проверка результата

```
stud4@palitpc:~/ait2/labs/lab_1/data$ docker run -v /home/stud4/ait2/labs/lab_1/data:/workspace -it stud4_opencv_lab
root@b47239811af3:/usr/local/Dev/opencv-4.8.0/build# cd /workspace
root@b47239811af3:/workspace# python3 piton.py
Image after gamma saved as: /workspace/car_gamma2.2.jpg
Image after gamma and grayscale saved as: /workspace/car_gamma2.2_gray.jpg
Image after gamma and contrast saved as: /workspace/car_gamma2.2_contrast1.2.jpg
Image after gamma, contrast, and brightness saved as: /workspace/car_gamma2.2_contrast1.2_brightness1.5.jpg
root@b47239811af3:/workspace#
```

```
root@b47239811af3:/workspace# ls /workspace
/workspace
OpenCVDockerFile.dockerfile  build.sh  build_env.sh  car_gamma2.2.jpg  car_gamma2.2_contrast1.2_brightness1.5.jpg  images  ubuntu_2204.sources
OpenCVDockerFileDebug.dockerfile  build_debug.sh  car.jpg  car_gamma2.2_contrast1.2.jpg  car_gamma2.2_gray.jpg  piton.py  ubuntu_2204_mirrors.sources
root@b47239811af3:/workspace# exit
exit
stud4@palitpc:~/ait2/labs/lab_1/data$ ls /home/stud4/ait2/labs/lab_1/data/
/workspace
build_debug.sh  build.sh  car_gamma2.2_contrast1.2.jpg  car_gamma2.2.jpg  images
build_env.sh  car_gamma2.2_contrast1.2_brightness1.5.jpg  car_gamma2.2_gray.jpg  car.jpg  OpenCVDockerFileDebug.dockerfile  piton.py
stud4@palitpc:~/ait2/labs/lab_1/data$
```

Шаг 10. Просмотр изображений

Исходное изображение:



Изображение после гамма-коррекции:



Изображение после гамма-коррекции в оттенках серого:



Изображение после гамма-коррекции и контрастности:



Изображение после гамма-коррекции, контрастности и яркости:



Заключение

В ходе выполнения лабораторной работы были изучены базовые методы обработки изображений с использованием библиотеки OpenCV. Реализованный алгоритм позволил выполнить последовательные преобразования изображения.