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

Факультет информатики  
Кафедра технической кибернетики

**Отчет по лабораторной работе №2**

Дисциплина: «Технологии Искусственного Интеллекта»

Тема: **«Docker»**

Выполнил: Мелешенко И.С.

Группа: 6233-010402D

Самара 2023

**Содержание**

[Глава 1. Установка и настройка Docker Desktop 3](#_Toc157025370)

[Глава 2. Выполнение задания 1 лабораторной работы 2. 5](#_Toc157025371)

[Задание 1 лабораторной работы 2. 5](#_Toc157025372)

[Подготовка Dockerfile. 6](#_Toc157025373)

[Подготовка решения задачи Salt and Paper. 6](#_Toc157025374)

[Исходный файл для работы. 7](#_Toc157025375)

[Выполнение 1 задания лабораторной работы 2. 8](#_Toc157025376)

[1. Cборка образа контейнера. 8](#_Toc157025377)

[2. Запуск контейнера. 8](#_Toc157025378)

[3. Проверим контейнер 9](#_Toc157025379)

[4. Перенос исходников. 9](#_Toc157025380)

[5. Запуски скрипта. 10](#_Toc157025381)

[6. Результаты. 12](#_Toc157025382)

[Часть 3. Выполнение задания 2 лабораторной работы 2. 15](#_Toc157025383)

[Задание 2 лабораторной работы 2. 15](#_Toc157025384)

[Подготовка Dockerfile 16](#_Toc157025385)

[Подготовка docker-compose 16](#_Toc157025386)

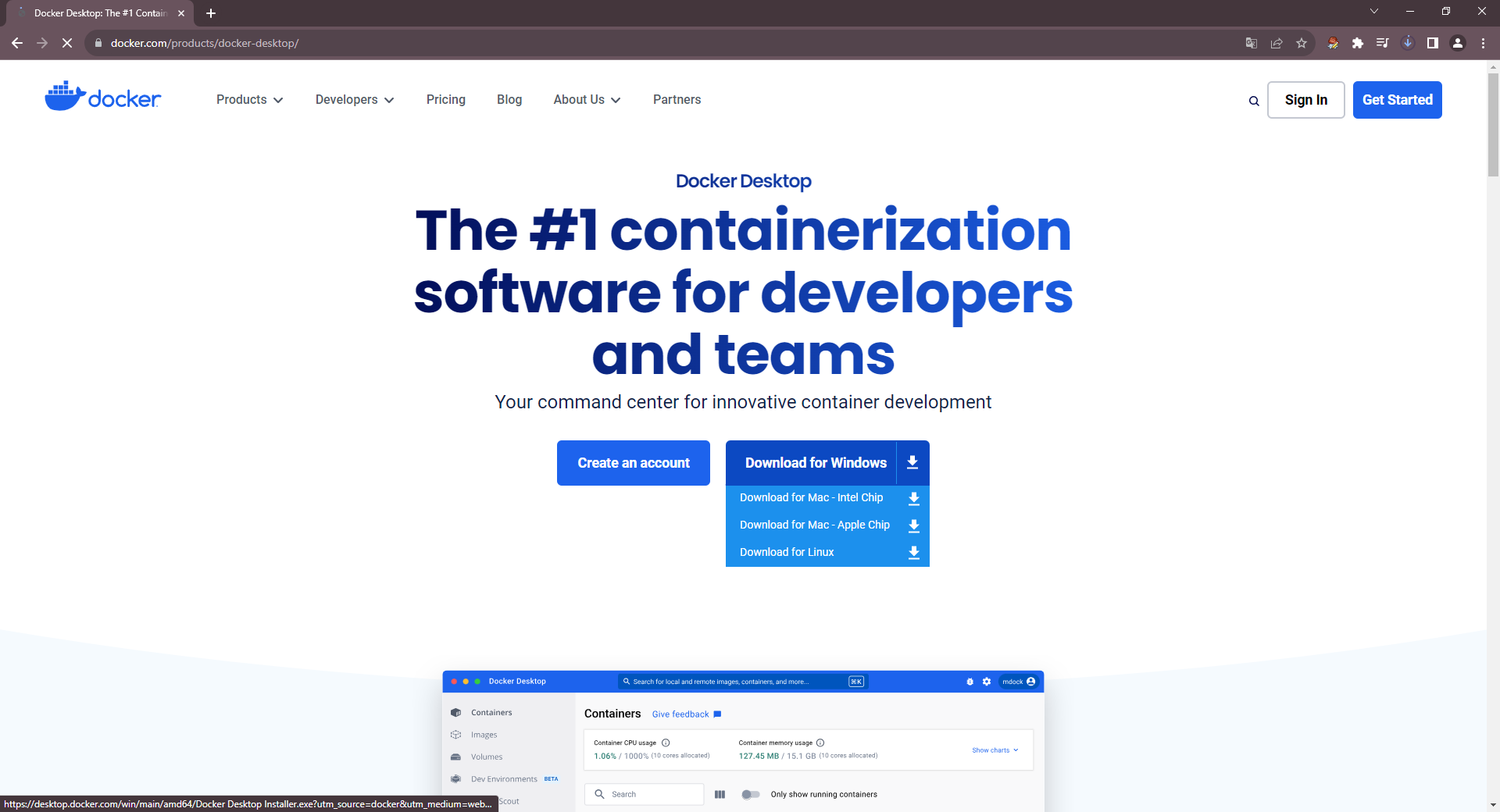
[Описание кода, для работы с нейросетью. 16](#_Toc157025387)

[Выполнение задания 2 лабораторной работы 2 18](#_Toc157025388)

[Заключение 21](#_Toc157025389)

# Глава 1. Установка и настройка Docker Desktop

1 Клиент программы Docker Desktop доступен на сайте по ссылке <https://www.docker.com/products/docker-desktop>

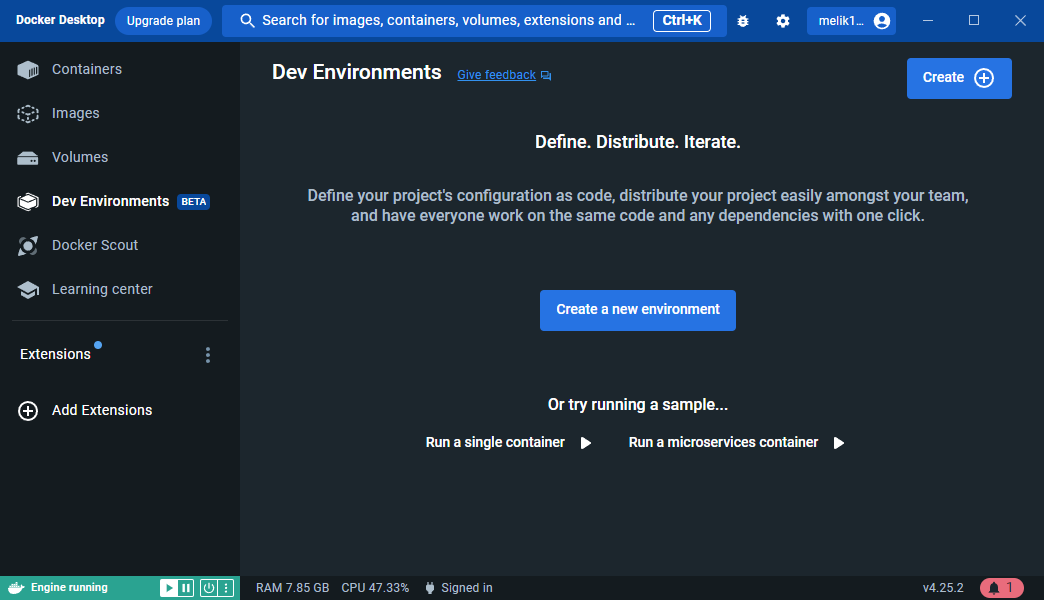
  
Рисунок 1 – Получение клиента программы Docker Desktop

2 На данном ПК потребуется произвести настройку для запуска программы Docker Desktop, более подробно об этом можно почитать на сайте по ссылке https://docs.microsoft.com/en-us/windows/wsl/install-manual.

*$ dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart*

*$ dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart*

3 После чего потребуется перезапуск машины. Теперь Docker Desktop установлен и запущен.

  
Рисунок 2 – Настроеный Docker Desktop

# Глава 2. Выполнение задания 1 лабораторной работы 2.

## **Задание 1 лабораторной работы 2.**

На базе образа Ubuntu 22.04 создать Docker контейнер со сборкой OpenCV с non free contrib модулями (пример 3 из лекции).

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

1. Отредактировать скрипт сборки build.sh, заменить значения:

* image\_tag - название тега;
* build\_thread\_count - количество потоков для сборки библиотеки, лучше указать n-1, где n - количество физических ядер CPU.

1. C 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.
2. Изменить права доступа, выдать разрешение для запуска скриптов build.sh, build\_env.sh:

***chmod +x build.sh***

***chmod +x build\_env.sh***

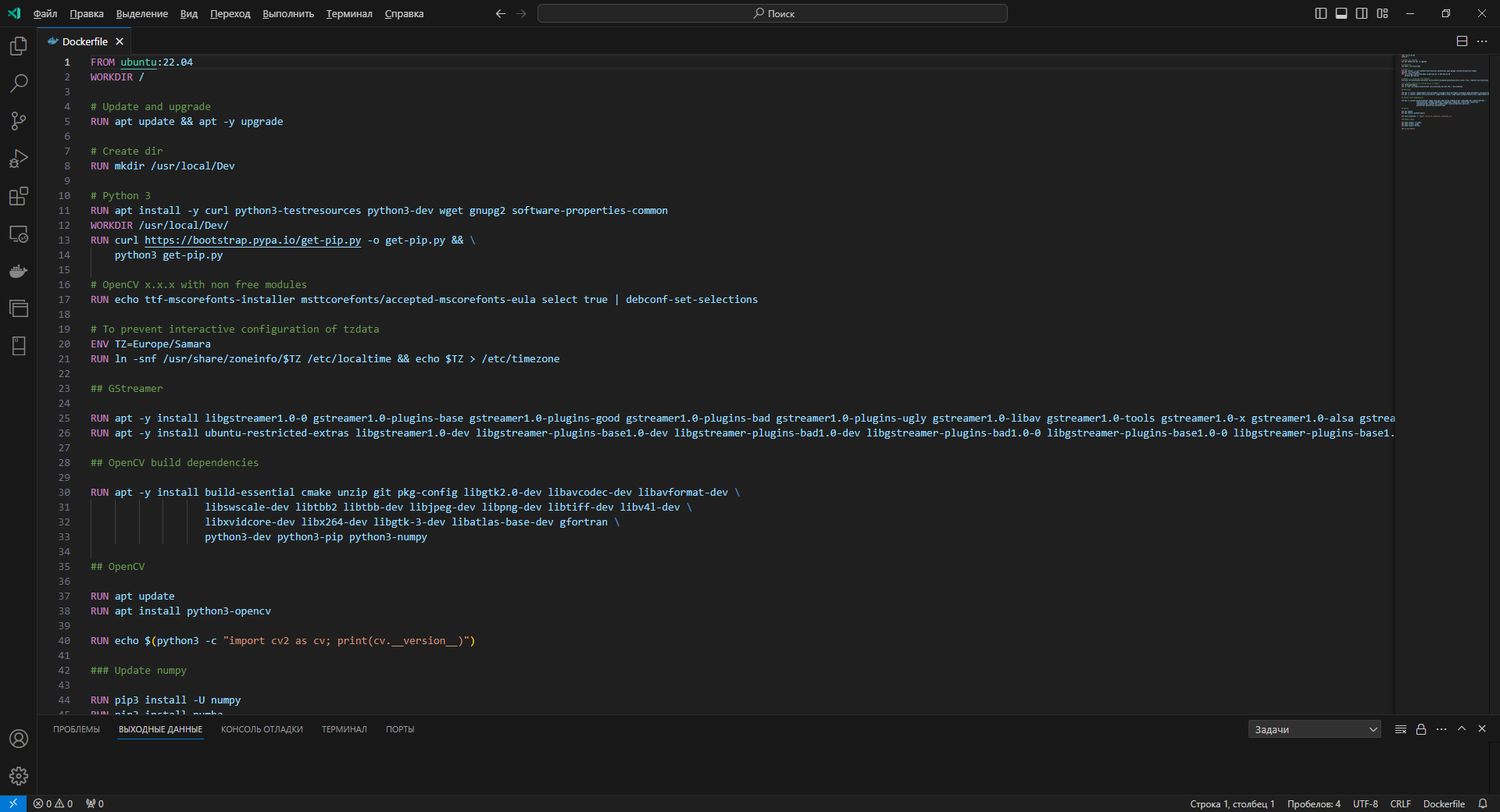
1. Дописать в конец докер файла (перед CMD) команды для установки необходимых либ Python 3.
2. Запустить build.sh для сборки контейнера.
3. Реализовать алгоритм обработки изображений, скрипт на питоне положить в папку на хосте.
4. Запустить контейнер командой:

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

1. Запустить скрипт с реализованным алгоритмом в контейнере в примонитрованной внутри контейнера папке. Результат обработки сохранить в локальной директории контейнера.
2. Убедиться в появлении результата в директории хоста.

## **Подготовка Dockerfile.**

Для того чтобы была возможность собрать образ контейнера, необходимо подготовить Dockerfile с инструкциями по сборке образа.

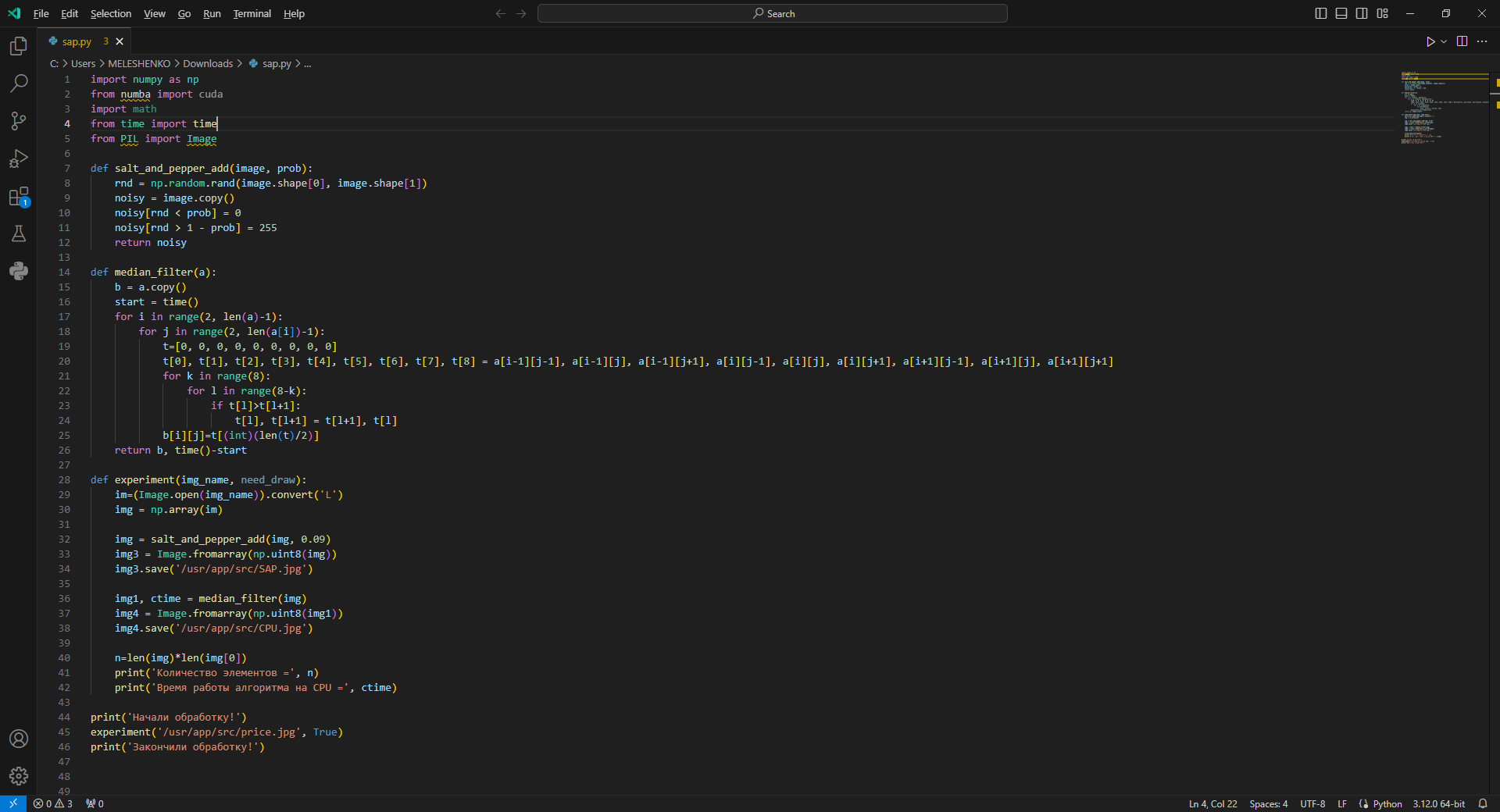
  
Рисунок 3 – Dockerfile для сборки образа контейнера.

## **Подготовка решения задачи Salt and Paper.**

За основу было взято решение задачи Salt and Paper, суть которой заключена в следующем:

Дано изображение размера M × N с шумом «Salt and Paper». Необходимо реализовать и применить версию 9-точечного медианного фильтра и сохранить результат в выходное изображение. Недостающие значения для краевых строк и столбцов берутся из ближайших пикселей. Данная задача уже рассматривалась мною в курсе «Высокопроизводительные вычисления» и решение расположено в репозитории по ссылке:  
<https://github.com/Black-Viking-63/HPC_Salt_And_Paper>.

Воспользуемся им, однако внесем не большие корректировки, из-за возможности работы на GPU. Новый вариант решения данной задачи с использованием CPU, представлено ниже.

  
Рисунок 4 – Решение задачи Salt and Paper

## **Исходный файл для работы.**

В качестве исходного изображения для работы алгоритма, я использовал скриншот из компьютерной игры, который представлен ниже.

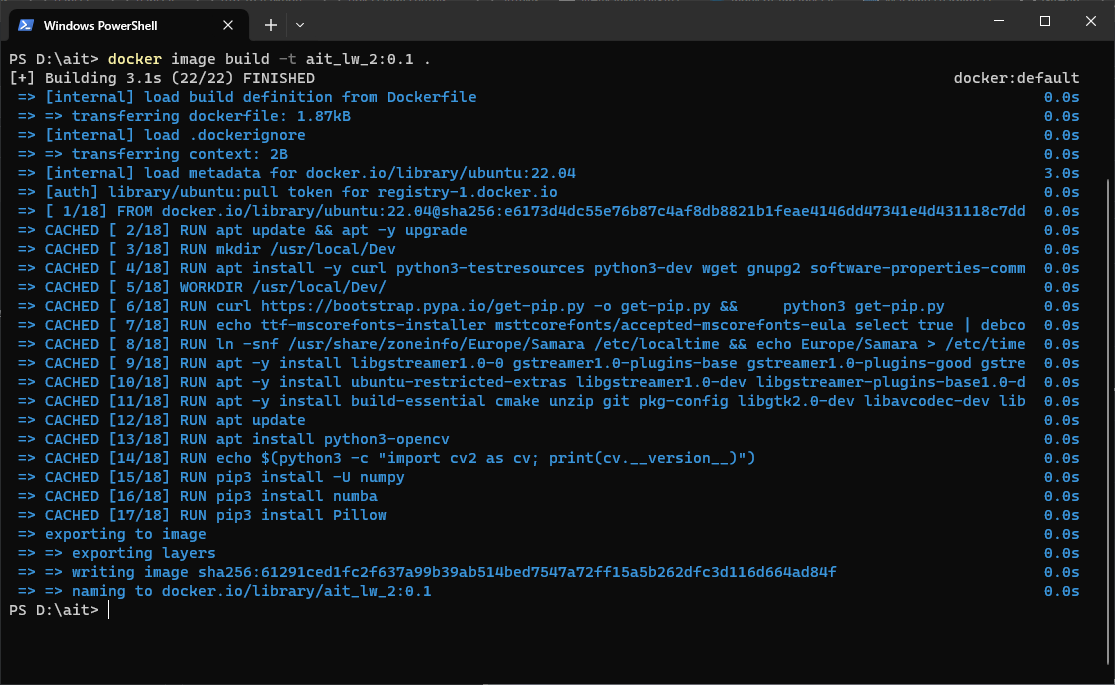
  
Рисунок 5 – Исходный файл для работы алгоритма

## **Выполнение 1 задания лабораторной работы 2.**

1. *Cборка образа контейнера.*

Для собрки образа, будем использовать команду:

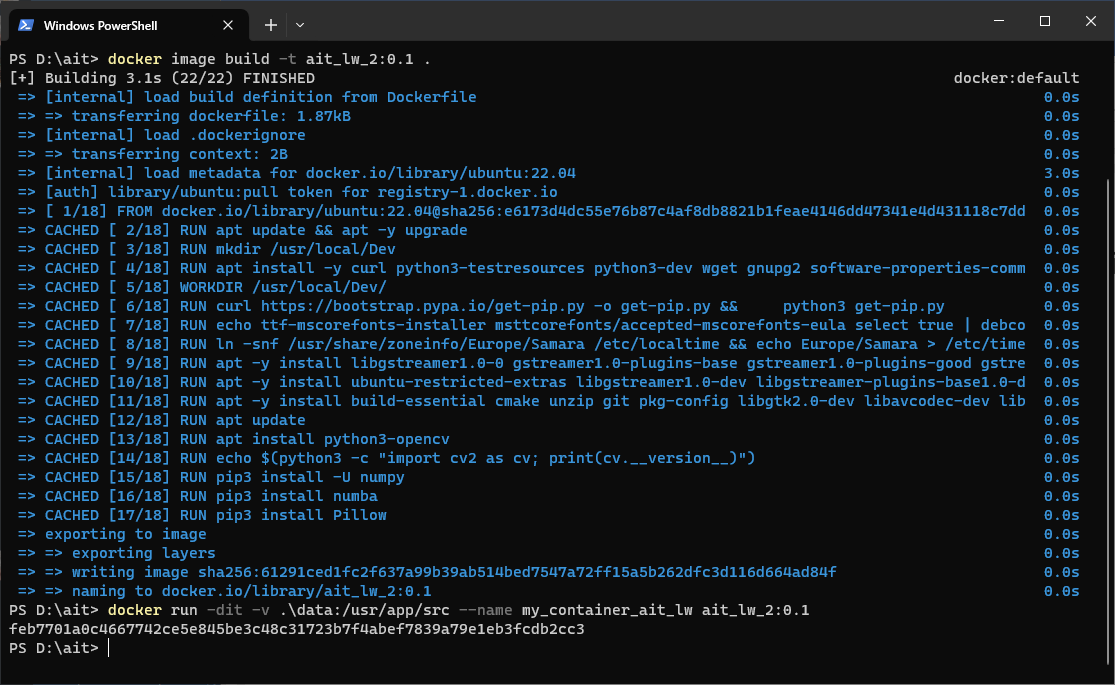
***$ docker image build -t ait\_lw\_2:0.1 .***

  
Рисунок 6 – Сборка образа контейнера

1. *Запуск контейнера.*

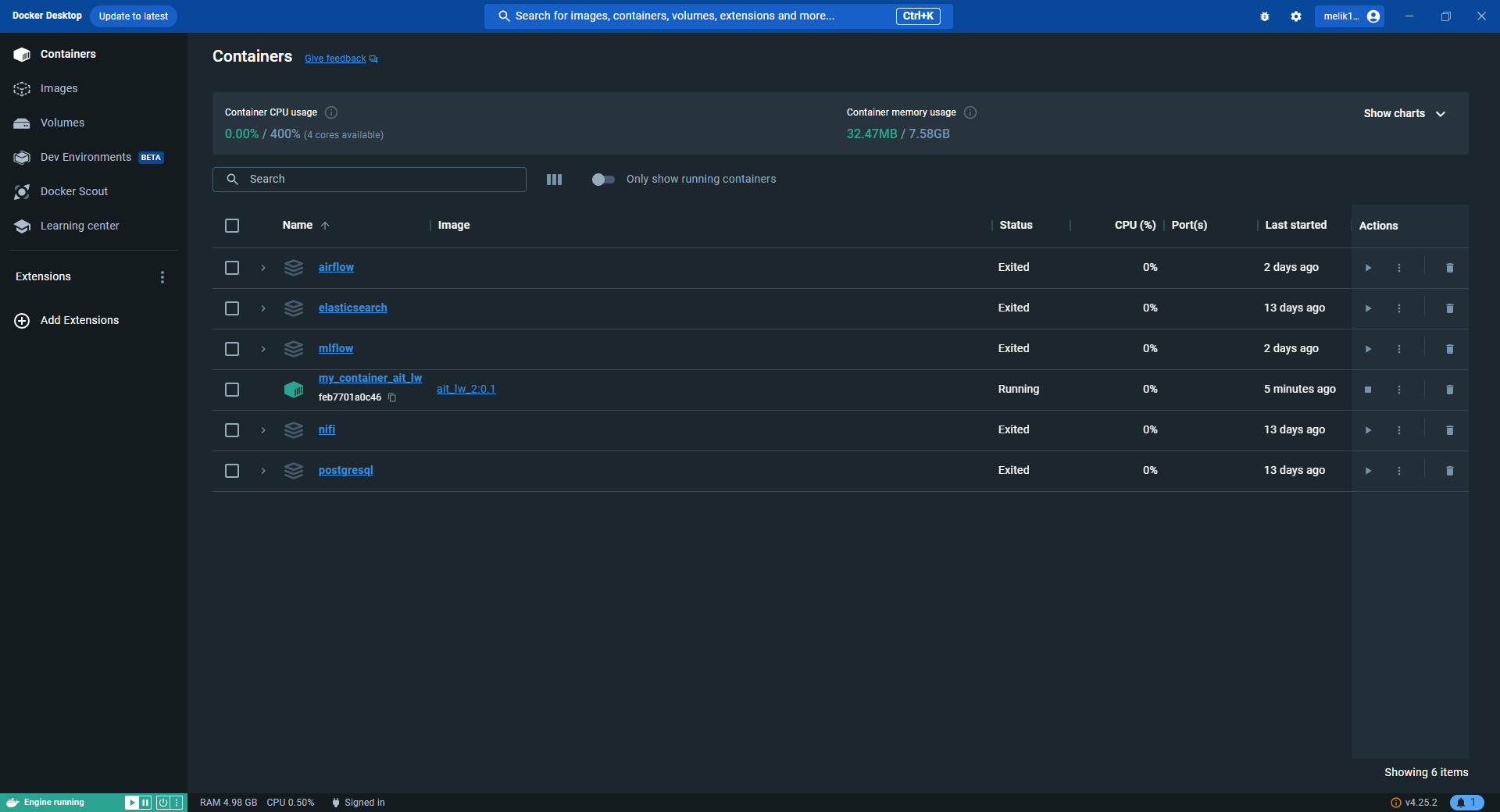
После успешной сборки, можем приступать к запуску контейнера, используя следующую команду:

***$ docker run -dit -v .\data:/usr/app/src --name my\_container\_ait\_lw ait\_lw\_2:0.1***

  
Рисунок 7 – Запуск контейнера

1. *Проверим контейнер*

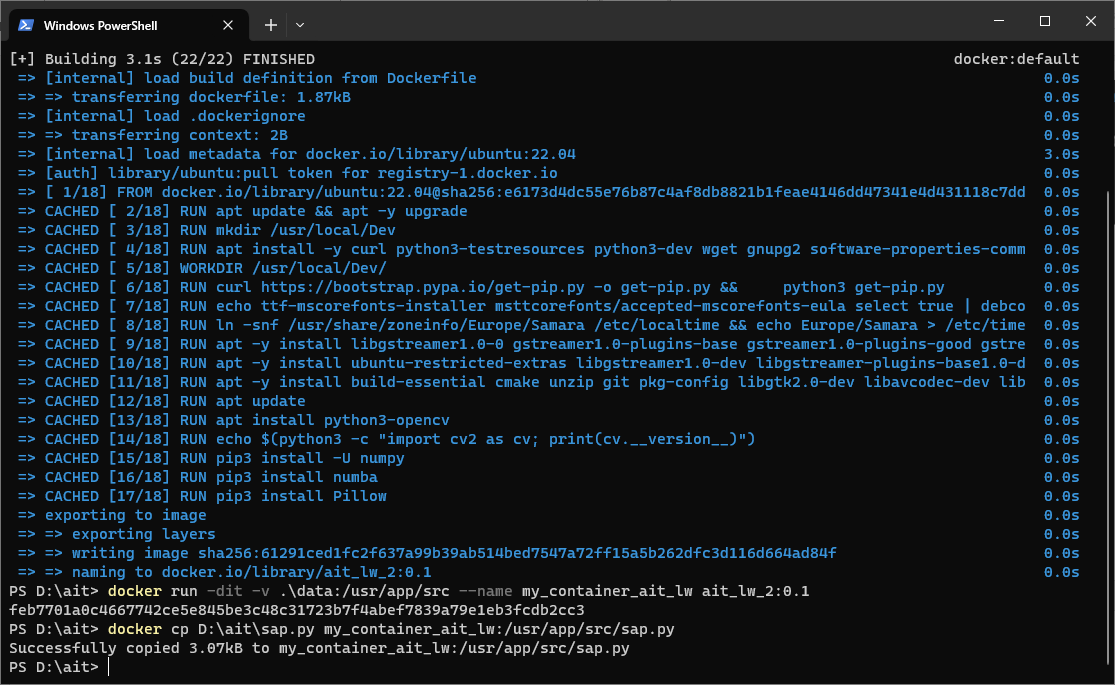
После запуска перейдем в Docker Desktop, и проверим что все получилось, наш контейнер создан, запущен и работает.

  
Рисунок 8 – Проверка контейнера

1. *Перенос исходников.*

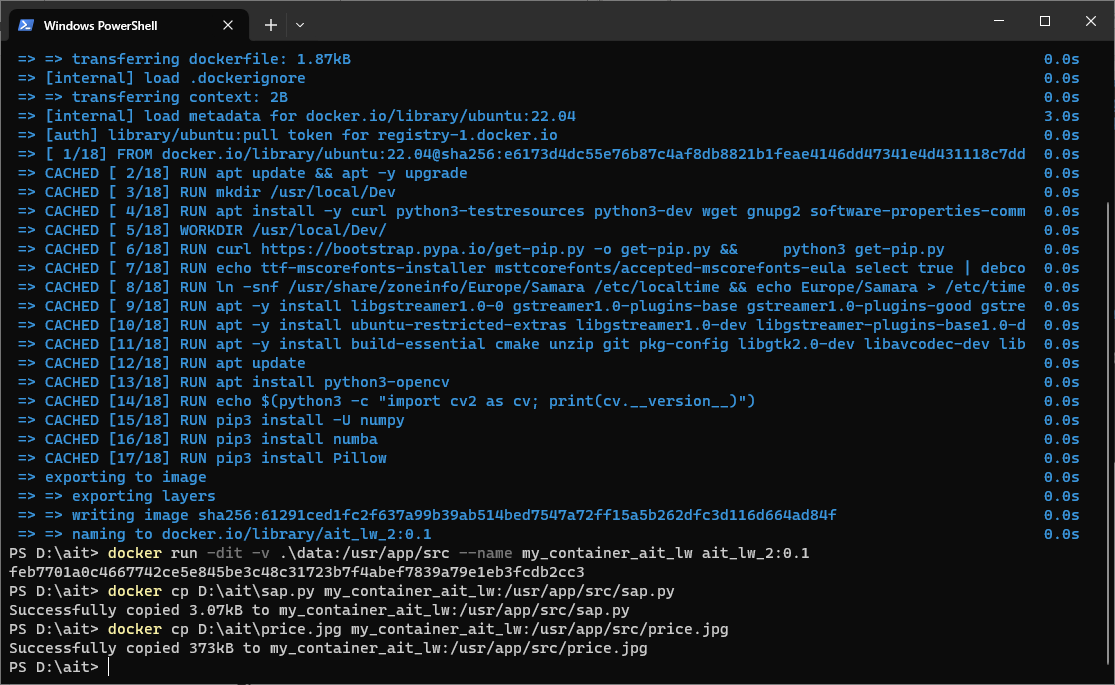
Для продолжения работы нам необходимо код и исходный файл перенести в контейнер, в котором и будет исполнятьс наш код. Файл с исходным кодом, переносим при помощи команды:

***$ docker cp D:\ait\sap.py my\_container\_ait\_lw:/usr/app/src/sap.py***

  
Рисунок 9 – Перенос исходного кода в контейнер

Файл изображения переносим аналогичной командой:

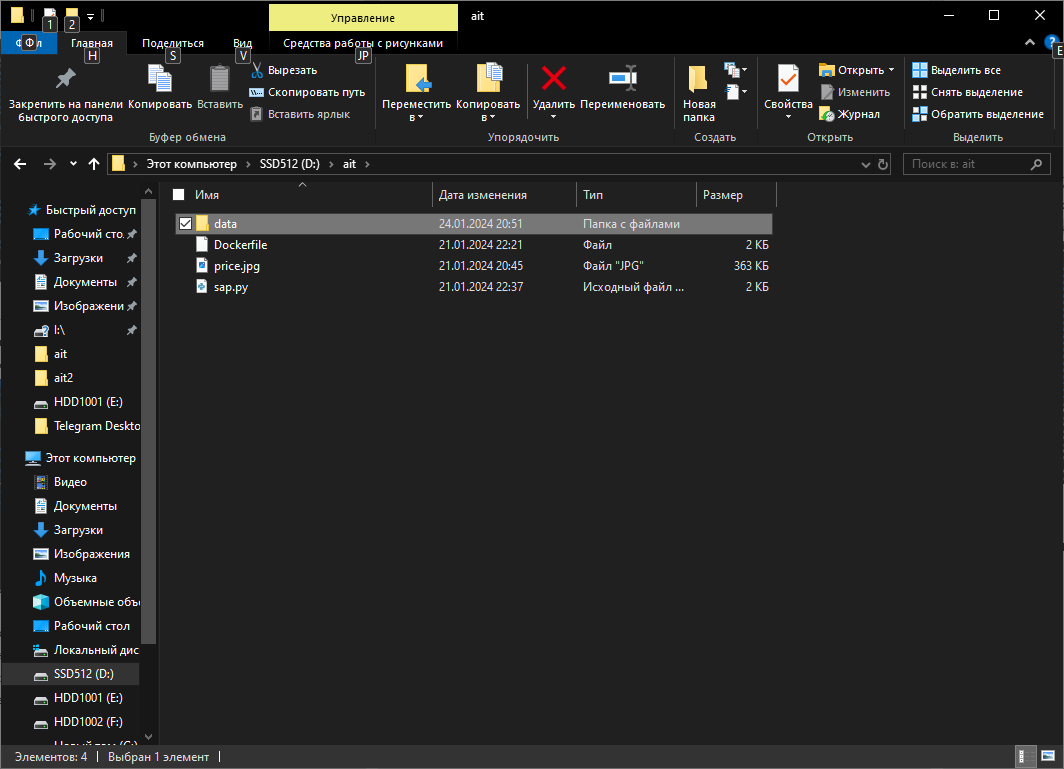
***$ docker cp D:\ait\price.jpg my\_container\_ait\_lw:/usr/app/src/price.jpg***

  
Рисунок 10 – Перенос изображения в контейнер

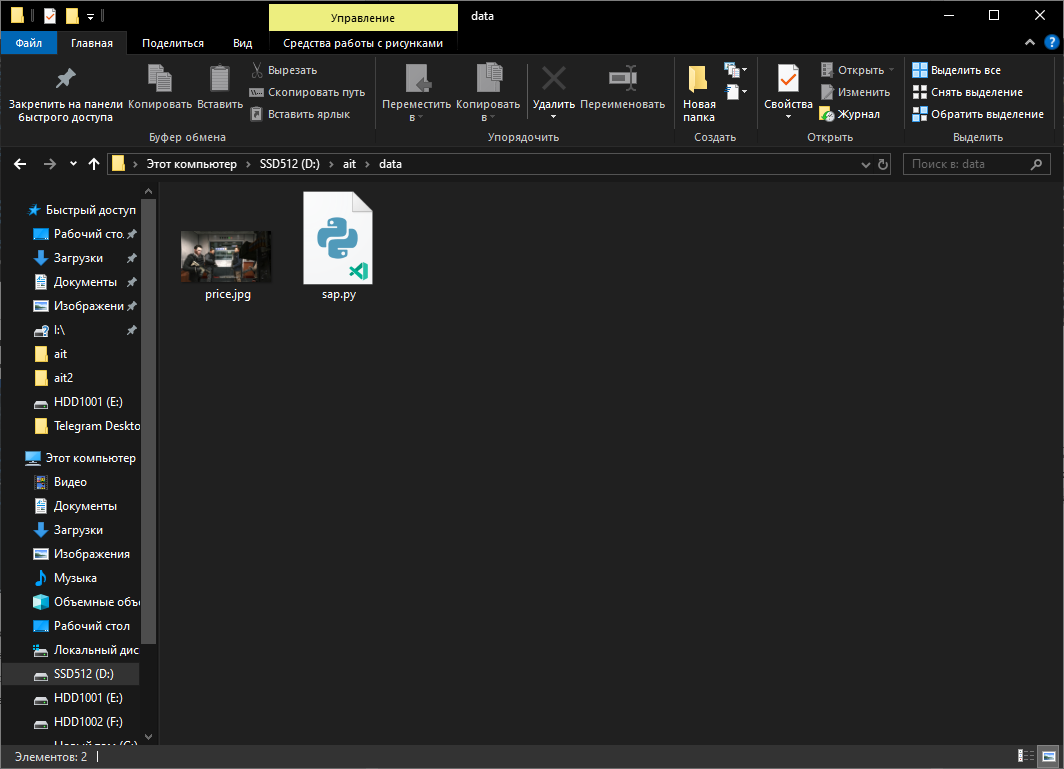
1. *Запуски скрипта.*

Перед запуском скрипта перейдем в папку и проверим нахождение файлов в нужных директориях.

Исходная папака с файлами:

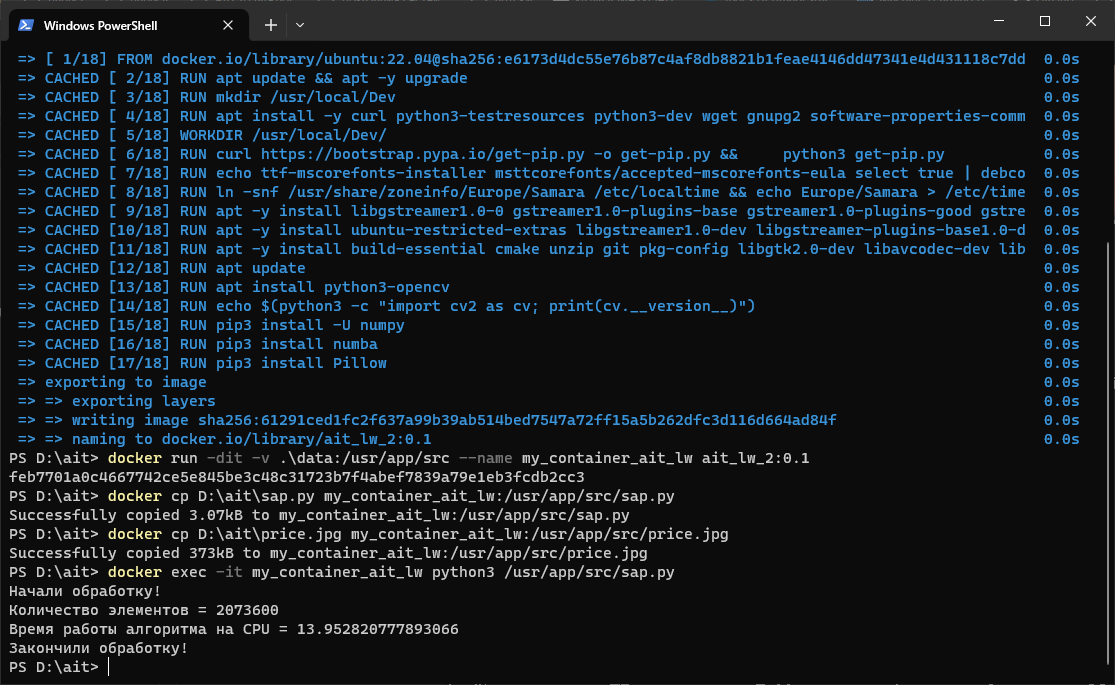
  
Рисунок 11 – Исходная пака с файлами

Папка с файлами для контейнера.

  
Рисунок 12 – Исходная папка с файлами для контейнера

Для того, чтобы запустить наш скрипт из контейнера, воспользуемся командой:

***$ docker exec -it my\_container\_ait\_lw python3 /usr/app/src/sap.py***

  
Рисунок 13 – Запуск скрипта

1. *Результаты.*

После отработки скрипта перейдем в папку с фалами контейнера и посмотрим на результаты.

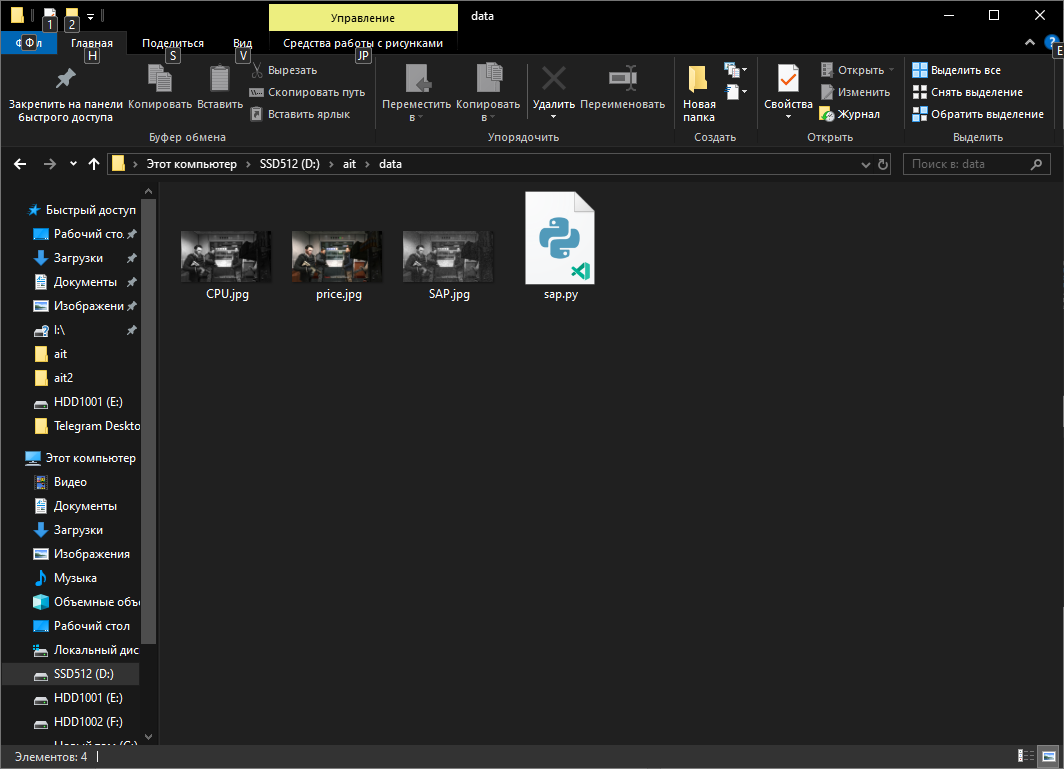
  
Рисунок 14 - Результаты

  
Рисунок 15 – Исходное изображение.

  
Рисунок 16 – Изображение с шумом «Salt and Papaer»

  
Рисунок 17 – Изображение восстановленное.

После окончания выполнения задания перейде ко второй части.

# Часть 3. Выполнение задания 2 лабораторной работы 2.

## **Задание 2 лабораторной работы 2.**

Запустить предобученную нейронку с использованием pytorch внутри контейнера. Для создания контейнера использовать Docker Compose.

1. Собрать контейнер с установленным PyTorch (CPU или GPU версия).

*Проще всего собрать контейнер на базе подготовленного образа с Docker Hub, например, pytorch:2.1.0-cuda11.8-cudnn8-devel. Можно и из обычного образа, например, Убунты. Тогда нужно будет установить CUDA определенной версии при сборке контейнера, а на самом хосте поставить NVIDIA Container Toolkit.*

1. Написать скрипт обработки изображений с использованием нейросети. Можно выбрать любую понравившуюся модель/задачу обработки изображения нейросетью.

*У niconielsen32 в репах есть целая подборка различных заготовок по Computer Vision, например, вычисление карты глубин по изображению.*

1. Запустить контейнер командой:

*docker compose -f <имя\_конфига.yaml> up*

1. Запустить скрипт с реализованным алгоритмом в контейнере в примонитрованной внутри контейнера папке. Результат обработки сохранить в локальной директории контейнера.
2. Убедиться в появлении результата в директории хоста.

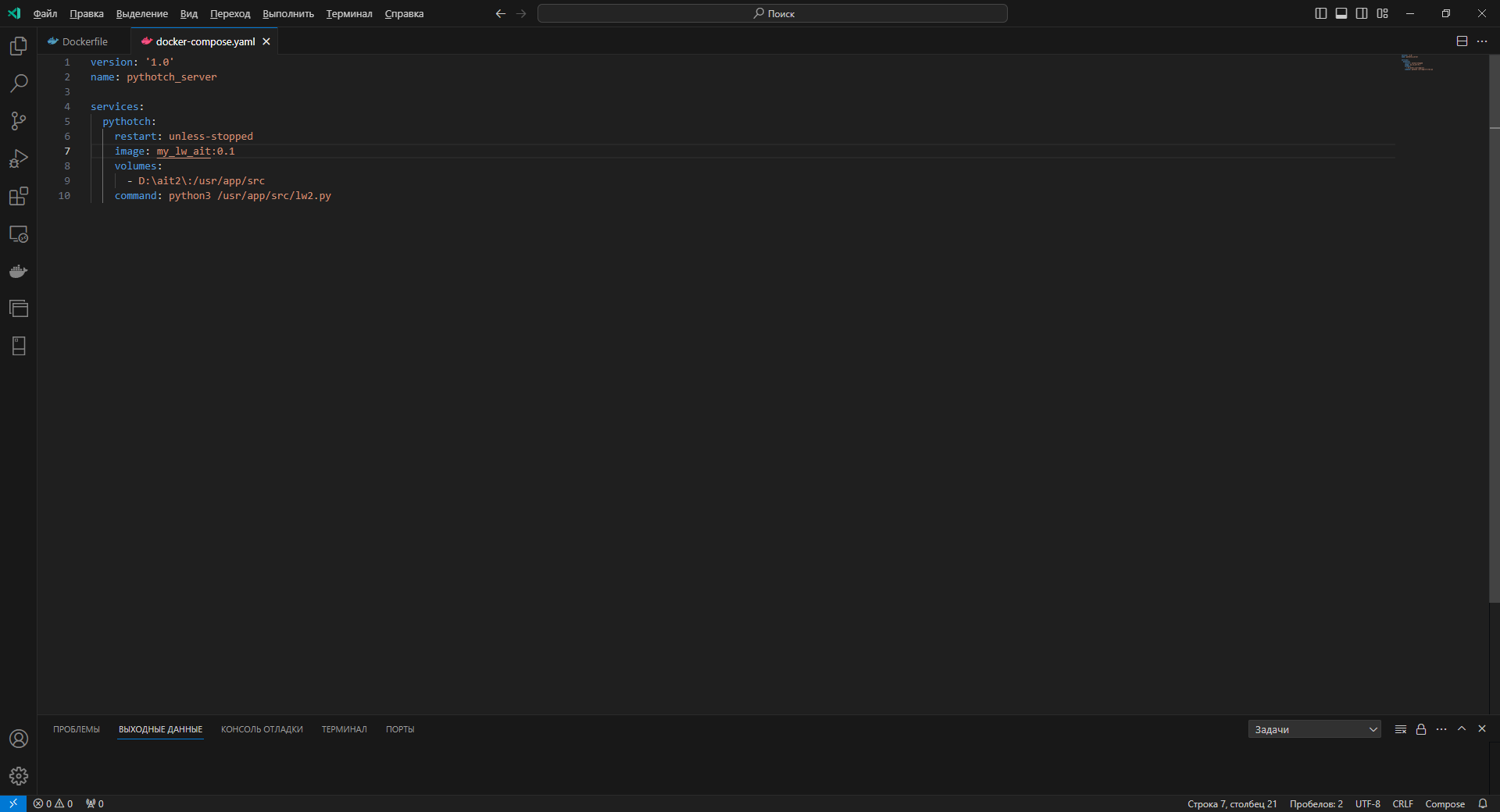
## **Подготовка Dockerfile**

Для создания образа контейнера воспользуемся следующим кодом:

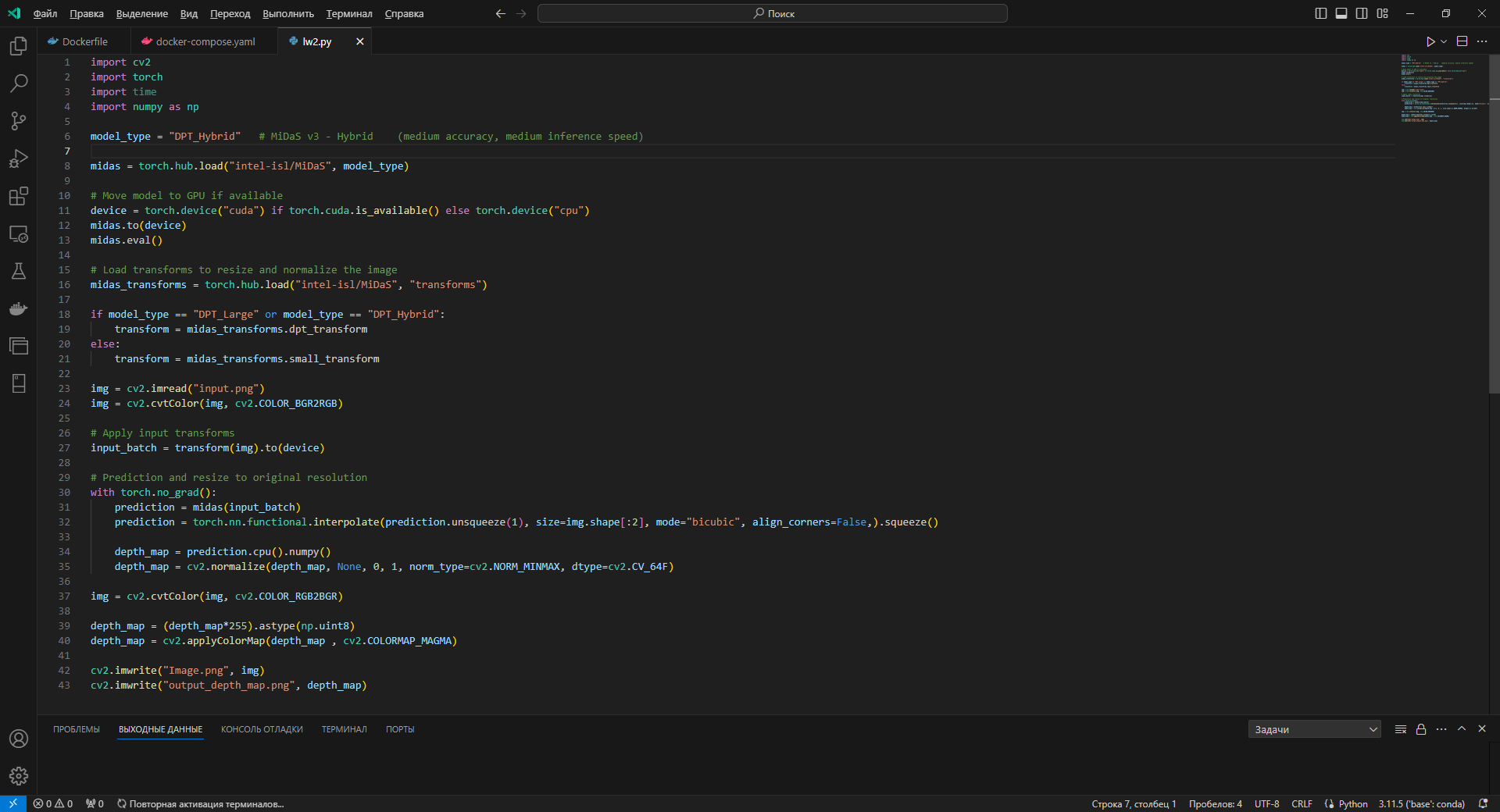
  
Рисунок 18 - Dockerfile

## **Подготовка docker-compose**

Для запуска контенера опишем docker-compose.yaml файл.

  
Рисунок 19 – Docker-Compose

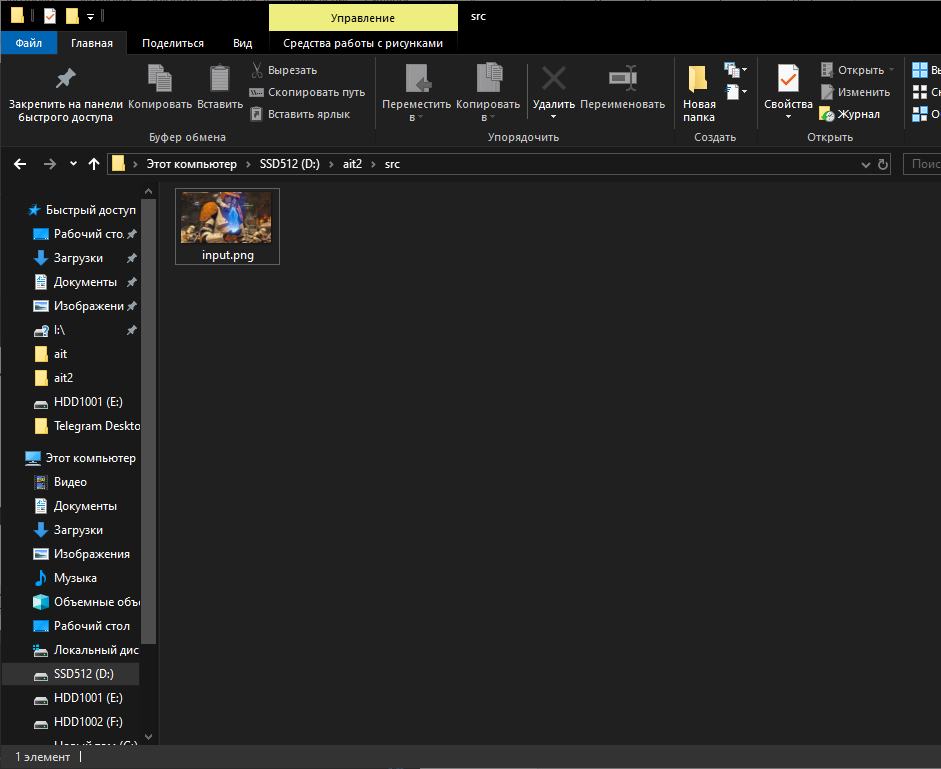
## **Описание кода, для работы с нейросетью.**

  
Рисунок 20 – Исходный код

Для работы нейросети необходимо изображение, которое будет использовано в качестве входного параметра. Будем использовать стоп кадр из кинофильма.

  
Рисунок 21 – Изображение для работы нейросети

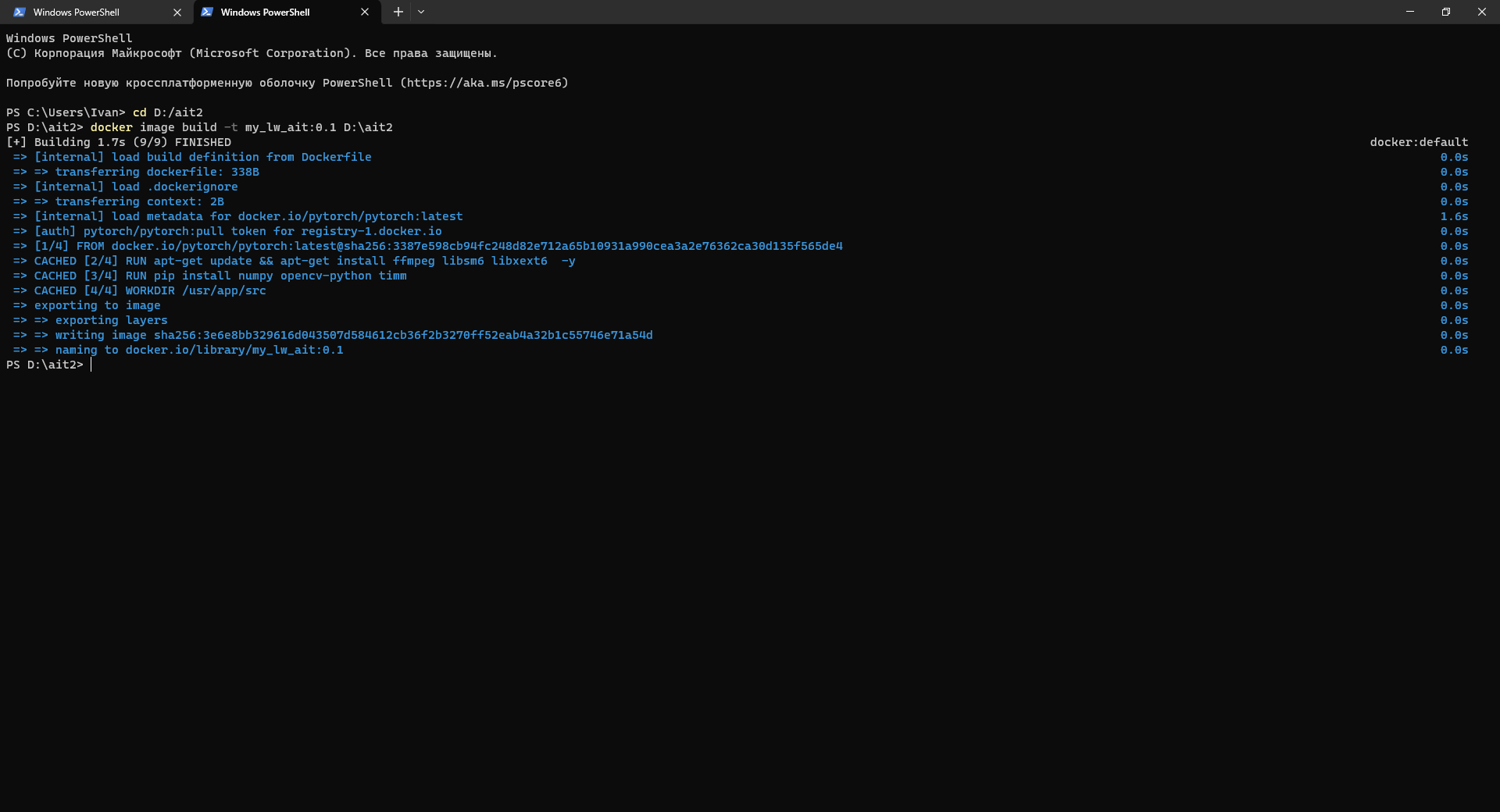
Перед началом работы переходим в исходную директорию:

  
Рисунок 22 – Исходная директория с файлами

## Выполнение задания 2 лабораторной работы 2

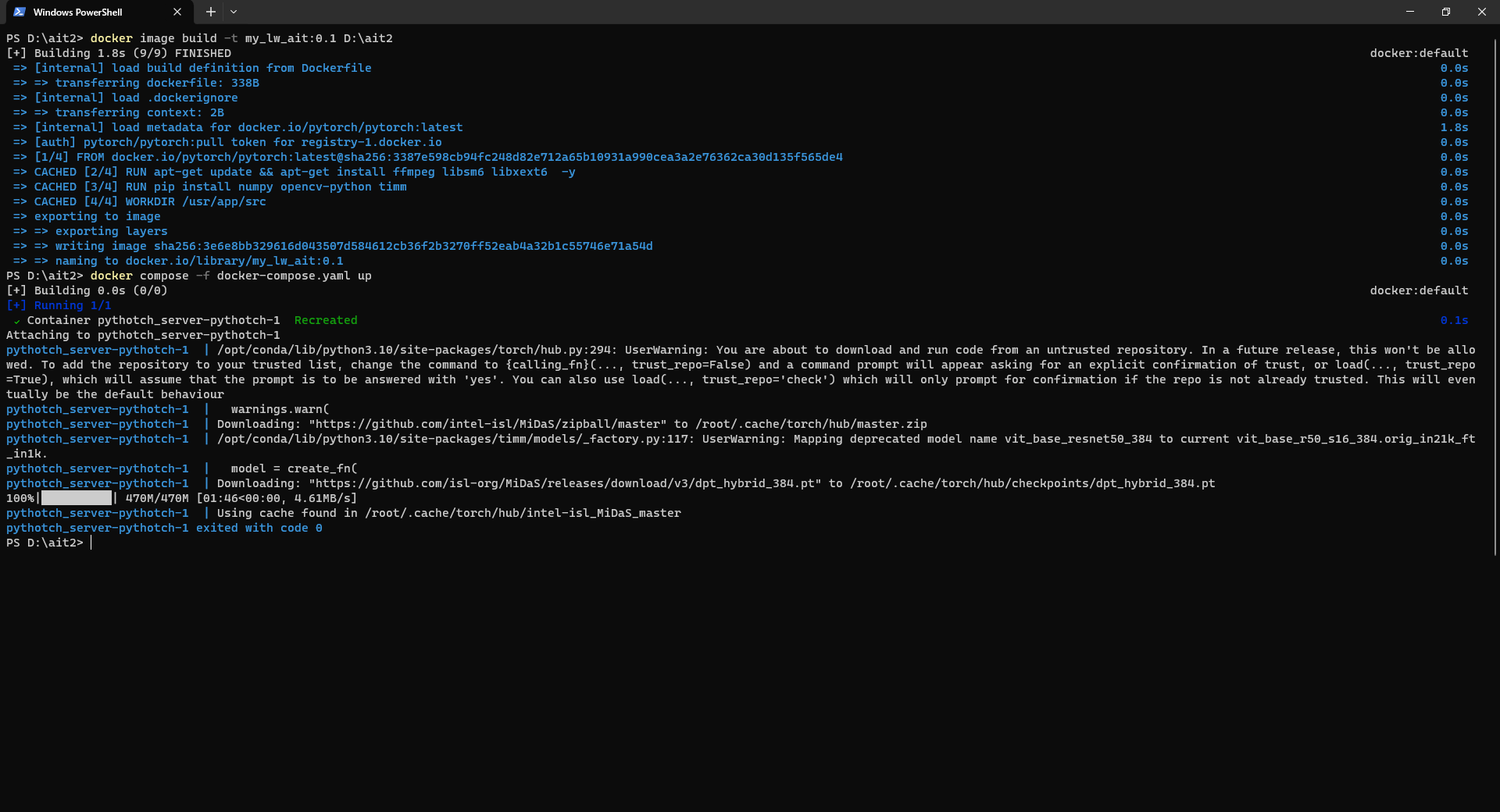
После всех приготиовлений переходим в терминал, и выполняем сборку контейнера при помощи команды:

**$ docker image build -t my\_lw\_ait:0.1 D:\ait2**

  
Рисунок 23 – Сборка образа контейнера

После окончания сборки образа запустим контейнер при помощи команды:

***$ docker compose -f docker-compose.yaml up***

  
Рисунок 24 – Запуск контейнера

По окончании работы алгоритма, перейдем в директорию и посмотрим на результаты:

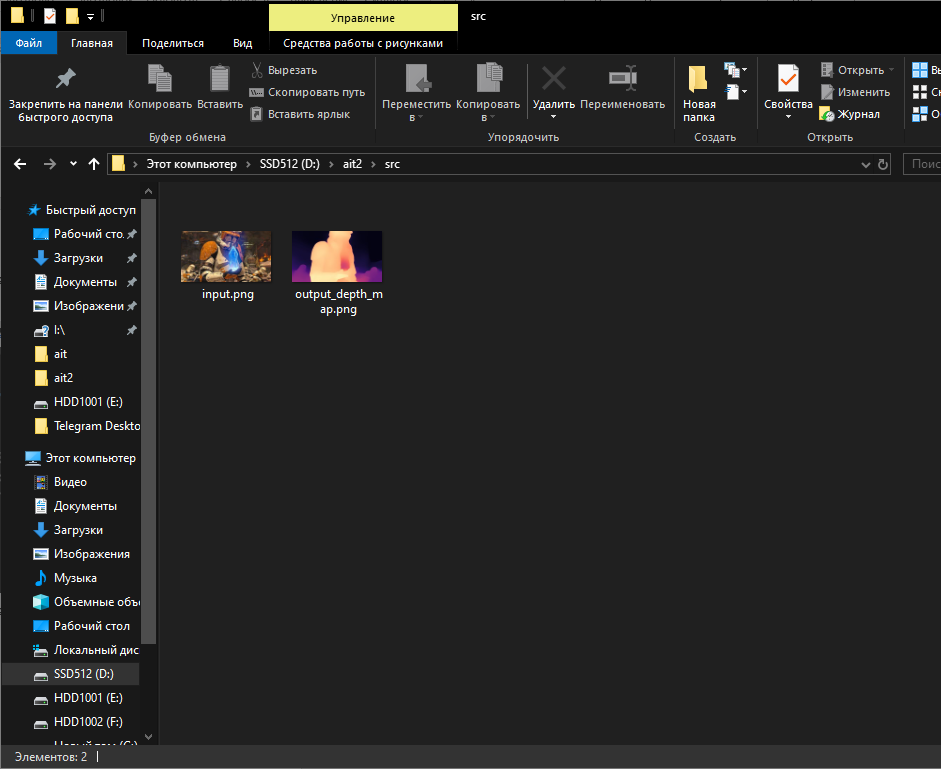
  
Рисунок 25 - Результаты

  
Рисунок 26 – Результирующее изображение.

# ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы получены навыки работы с Docker.