

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 №2

Дисциплина: «Инженерия данных»

Тема: «**Инференс и обучение НС**»

Выполнил: Каспаров И.А.

Группа: 6232-010402D

Самара 2024

Задание на лабораторную работу

Пайплайн для инференса данных

В рамках данного задания предлагается построить пайплайн, который реализует систему "Автоматического распознавания речи" для видеофайлов.

Построенный пайплайн будет выполнять следующие действия поочередно:

1. Производить мониторинг целевой папки на предмет появления новых видеофайлов.
2. Извлекать аудиодорожку из исходного видеофайла.
3. Преобразовывать аудиодорожку в текст с помощью нейросетевой модели.
4. Формировать конспект на основе полученного текста.
5. Формировать выходной .pdf файл с конспектом.

Пайплайн для обучения модели

В рамках данного задания предлагается построить пайплайн, который реализует систему автоматического обучения/дообучения нейросетевой модели.

Предлагается самостоятельно выбрать набор данных и модель для обучения. Например, можно реализовать пайплайн для обучения модели, которую вы планируете использовать в вашей НИР или ВКРМ. Это также позволит вам добавить отдельный пункт в ваш отчет.

Итак, пайплайн будет выполнять следующие действия:

1. Читать набор файлов из определенного источника (файловой системы, сетевого интерфейса и т.д.).
2. Формировать пакет данных для обучения модели.
3. Обучать модель.

4. Сохранять данные результатов обучения (логи, значения функции ошибки) в текстовый файл

Для успешного выполнения задания необходимо продемонстрировать успешность обучения модели и приложить файл `.ipynb`, в котором продемонстрирован процесс инференса данной модели.

СОДЕРЖАНИЕ

Часть 1. Построение пайплайн для инференса данных.	5
Шаг 1. Разработка и реализация DAG-а	5
Шаг 2. Регистрация на huggingface и получения токена API.....	6
Шаг 3. Создание Docker образа с необходимыми библиотеками.	7
Шаг 4. Подготовка DAG-а.	9
Шаг 5. Запуск DAG-а.	10
Часть 2. Пайплайн, который реализует систему автоматического обучения/дообучения нейросетевой модели.....	12
Шаг 1. Разработка DAG	12
Шаг 2. Запуска DAG-а.	13
Заключение	14

Часть 1. Построение пайплайн для инференса данных.

Шаг 1. Разработка и реализация DAG-a

В рамках первого задания необходимо реализовать пайплайн, который реализует систему "Автоматического распознавания речи" для видеофайлов. Построенный пайплайн будет выполнять следующие действия поочередно:

- Производить мониторинг целевой папки на предмет появления новых видеофайлов.
- Извлекать аудиодорожку из исходного видеофайла.
- Преобразовывать аудиодорожку в текст с помощью нейросетевой модели.
- Формировать конспект на основе полученного текста.
- Формировать выходной .pdf файл с конспектом.

Для реализации описанных действий мы будем использовать DockerOperator, а также FileSensor для получения необходимого видеофайла.

Для работы task-a по ожиданию получения нового видео необходимо создать новое подключение к airflow. Для создания подключения переходим в Airflow по адресу <http://localhost:8080/connection/list/> или мы можем в Airflow пройти по пути Admin-->Connections, как на рисунке ниже.

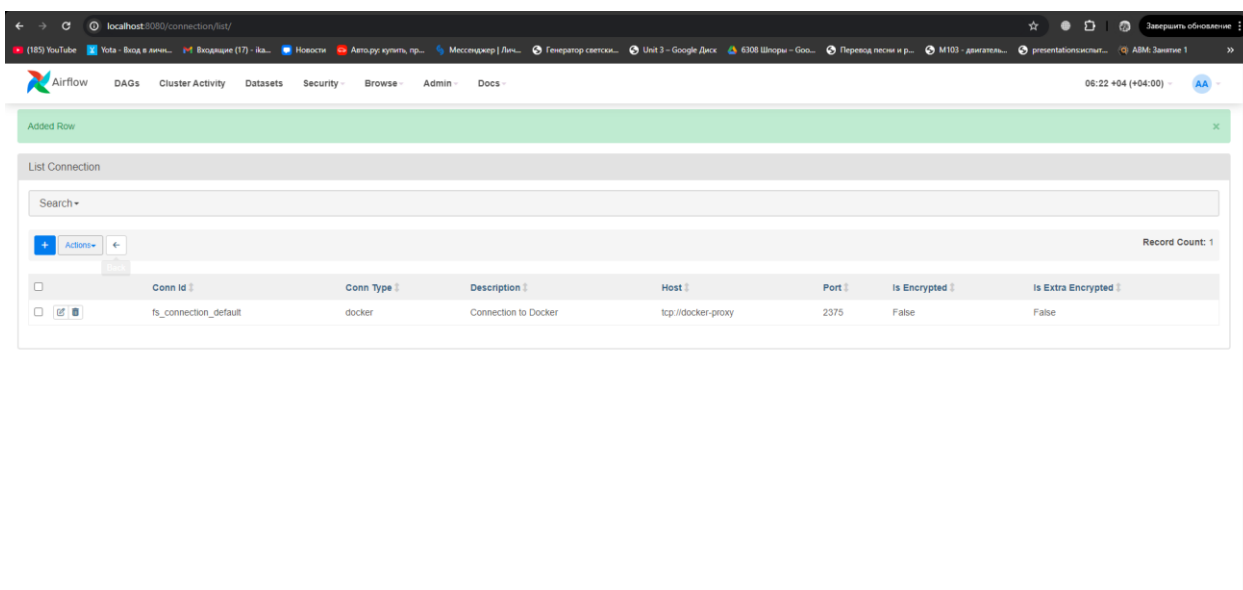


Рисунок 1 – Создание Connection

Шаг 2. Регистрация на *huggingface* и получения токена API.

Далее для того, чтобы можно было преобразовать наш аудиофайл в текст, а после получить из него summary, необходимо зарегистрироваться на <https://huggingface.co/> и получить токен API с правами записи для возможности отправки и получения запросов к сайту.

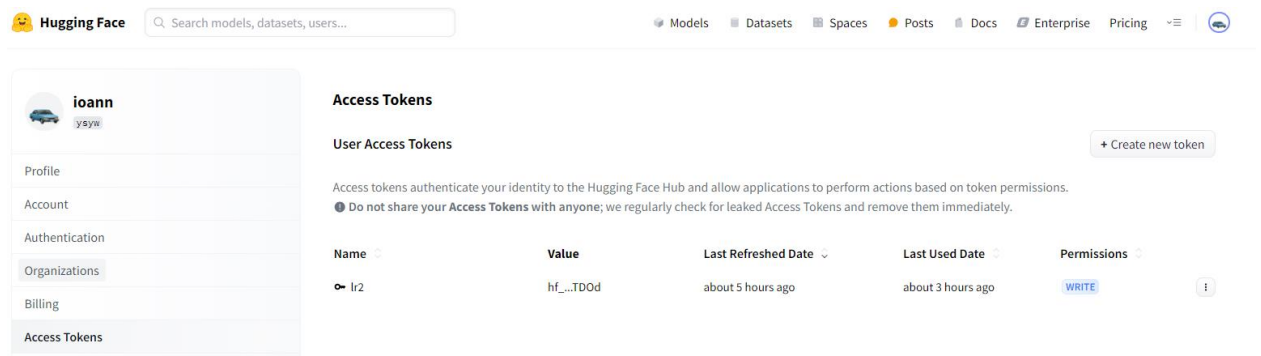


Рисунок 2 – Получение токена API

API токен = hf_njgYVOqRYokePEAGaWbGCyYWFyuyDCTDd

Шаг 3. Создание Docker образа с необходимыми библиотеками.

Для сохранения конспекта в PDF, необходимо было использовать библиотеку fpdf. Создадим необходимый для этого образ в Docker, который будет содержать в себе необходимые библиотеки для выполнения всей лабораторной работы. Процесс представлен ниже.

Вначале создадим Dockerfile который будет содержать в себе инструкции по сборке и разворачиванию контейнера. Контейнер мы создаем на основе tensorflow, который нам пригодится при выполнении второй части работы. Так же добавим следующие библиотеки, которые нам понадобятся в будущем:

- Scikit-learn
- Numpy
- Pandas
- FPDF

Первым делом произведем сборку образа, при помощи команды:

`$ docker build . -t our_tensorflow_container`

```
Start a build
PS C:\Users\ikasp\vscode\labs\docker> docker build -t our_tensorflow_container .
[+] Building 3026.7s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 358B
=> [internal] load metadata for docker.io/tensorflow/tensorflow:latest
=> [auth] tensorflow/tensorflow:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/3] FROM docker.io/tensorflow/tensorflow:latest@sha256:f3be5db24f080b3b228a23eb24f586058b3bec445d81500f95167c70e5473d4e
=> => resolve docker.io/tensorflow/tensorflow:latest@sha256:f3be5db24f080b3b228a23eb24f586058b3bec445d81500f95167c70e5473d4e
=> => sha256:3a6e466541337435bc0d7230b38695810ab694d60dcca510462a5e187e5d52c 1.13kB / 1.13kB
=> => sha256:6a0550e4123ec5614f97f69ebb5ca767c1c43b213a52af3087ab15bba018ae9c 1.13kB / 1.13kB
=> => sha256:9cc72bd2daf540ae775691c7190a1af3919a09b60f39b06cb4a4bd0aa5f9460d 330.31MB / 330.31MB
=> => sha256:a9eb4219fce39e31343dd629ad7697f6a46e41e25ff4933709d7bf5e41bfb885 37.41MB / 37.41MB
=> => sha256:e806dd517e77346add853700810a931c5d5b220c38f9b84125e253e69858be34 127B / 127B
=> => sha256:cf450bbc2f912c0a2236789627065e3e96b53d51763e7182653557d43ba98ba7 968B / 968B
=> => sha256:cbdff062747acba4a231665a488acc0aef0cd1cdac9ab0eda6f73f113b7424b15 133.29MB / 133.29MB
=> => sha256:0b3e4bf45acf2ea793433a225dcfdcab61562eaa52abee05f7d9794f5dbddef 44.28MB / 44.28MB
=> => sha256:2443cd0be7bb6a9c615e32e3c44d3a3e8bcab2f73190721c08a033a10ca2e797 164B / 164B
=> => sha256:70464040456b253f3544f76746d197434778dabc313e26cc07f2d70d45fecdc39 766B / 766B
=> => sha256:d877f681d4feb5f423c9014e161cf85e9f5bb37de840f7d9e62b1b0645b31c86 858B / 858B
=> => sha256:6414378b647780fee8fd903ddb9541d134a1947ce092d08bdeb23a54cb3684ac 29.54MB / 29.54MB
=> => extracting sha256:6414378b647780fee8fd903ddb9541d134a1947ce092d08bdeb23a54cb3684ac
=> => extracting sha256:d877f681d4feb5f423c9014e161cf85e9f5bb37de840f7d9e62b1b0645b31c86
=> => extracting sha256:70464040456b253f3544f76746d197434778dabc313e26cc07f2d70d45fecdc39
=> => extracting sha256:2443cd0be7bb6a9c615e32e3c44d3a3e8bcab2f73190721c08a033a10ca2e797
=> => extracting sha256:0b3e4bf45acf2ea793433a225dcfdcab61562eaa52abee05f7d9794f5dbddef
=> => extracting sha256:cbdff062747acba4a231665a488acc0aef0cd1cdac9ab0eda6f73f113b7424b15
=> => extracting sha256:cf450bbc2f912c0a2236789627065e3e96b53d51763e7182653557d43ba98ba7
=> => extracting sha256:e806dd517e77346add853700810a931c5d5b220c38f9b84125e253e69858be34
=> => extracting sha256:a9eb4219fce39e31343dd629ad7697f6a46e41e25ff4933709d7bf5e41bfb885
=> => extracting sha256:9cc72bd2daf540ae775691c7190a1af3919a09b60f39b06cb4a4bd0aa5f9460d
=> => extracting sha256:6a0550e4123ec5614f97f69ebb5ca767c1c43b213a52af3087ab15bba018ae9c
=> => extracting sha256:3a6e466541337435bc0d7230b38695810ab694d60dcca510462a5e187e5d52c
=> [auth] tensorflow/tensorflow:pull token for registry-1.docker.io
=> [2/3] RUN pip install scikit-learn numpy pandas fpdf
=> [3/3] WORKDIR /usr/app/src
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:dd867e11660a22eb68ad1bae829df2e82b635fa359a8396b39b6ddd1271454e
=> => exporting config sha256:397dcd74f3e3bdd30088e1fe19e9f6e93b025f598649079326467436599dbcaf
=> => exporting attestation manifest sha256:703e2ee9aac43d192eccbbca4a8114cb786dc27e750723ff4eda2a4e1661e5cd8
=> => exporting manifest list sha256:afe000f5c984862208809b9e1651ceaa1490b622eb71af8a2b4fe106681f443c
=> => naming to docker.io/library/our_tensorflow_container:latest
=> => unpacking to docker.io/library/our_tensorflow_container:latest
```

Рисунок 3 – Сборка образа

После успешной сборки, необходимо произвести проверку того, что Docker образ был создан. Для этого используем команду

```
$ docker images
```

После проверки произведем присвоение tag нашему образу, для того чтобы можно было произвести отправку нашего контейнера в DockerHub.

Для этого воспользуемся командой:

```
$ docker tag our_tensorflow_container ysyw/our_tensorflow_container:1.0
```

В итоге после всех приготовлений, произведем отправку нашего образа в DockerHub, при помощи команды:

```
$ docker push ysyw/our_tensorflow_container:1.0
```



```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/1xapk05ozh8w7g2hq7pp2w3yc
PS C:\Users\ikasp\vscode\labs\docker> docker tag our_tensorflow_container ysyw/our_tensorflow_container:1.0
PS C:\Users\ikasp\vscode\labs\docker> docker push ysyw/our_tensorflow_container:1.0
The push refers to repository [docker.io/ysyw/our_tensorflow_container]
96bb988fde95: Pushed
70464040456b: Pushed
6414378b6477: Pushed
e806dd517e77: Pushed
cf450bbc2f91: Pushed
2443cd0be7bb: Pushed
d877f681d4fe: Pushed
a9eb4219fce3: Pushed
3a6e46654133: Pushed
0b3e4bf45acf: Pushed
bc2f9fee8911: Pushed
37d4470aa3ec: Pushed
9cc72bd2daf5: Pushed
cbdf062747a: Pushed
6a0550e4123e: Pushed
1.0: digest: sha256:afe000f5c984862208809b9e1651ceaa1490b622eb71af8a2b4fe106681f443c size: 856
PS C:\Users\ikasp\vscode\labs\docker>
```

Рисунок 4 – Присвоение тега образу и отправка образа в DockerHub

Шаг 4. Подготовка DAG-a.

В результате выполнения данной части работы был разработан DAG, состоящий из 5 task:

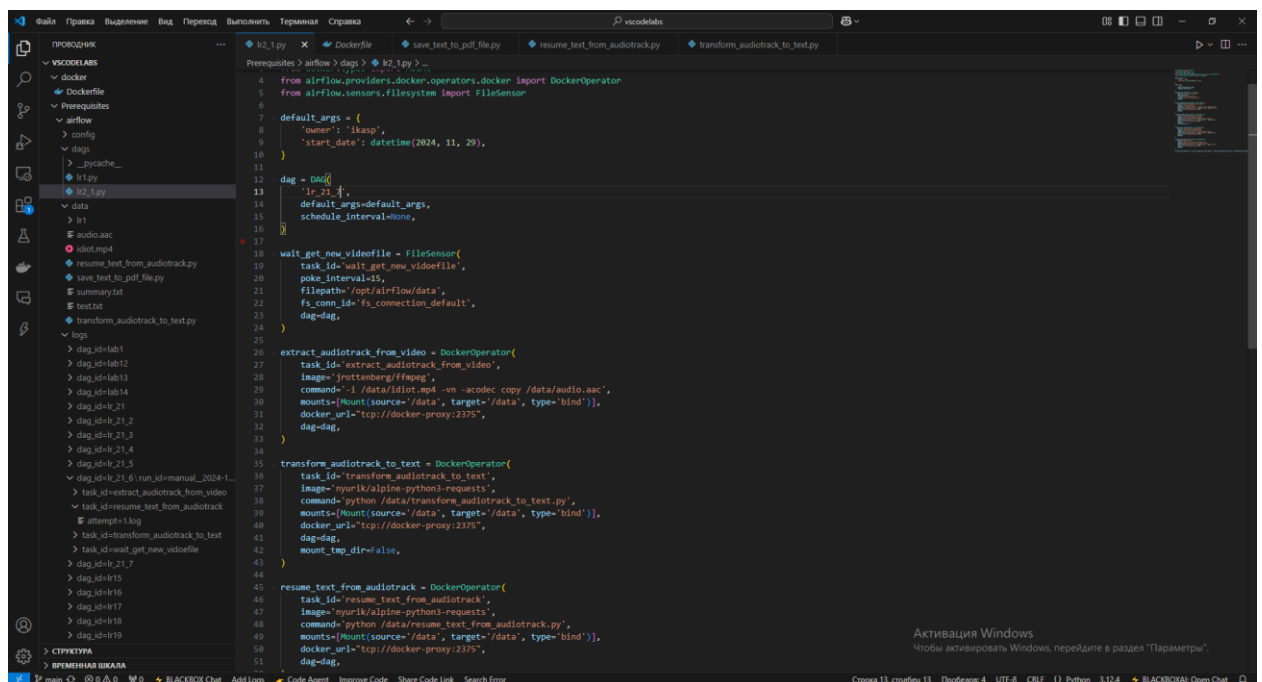
wait_get_new_videofile – осуществляет «прослушивание» указанной директории, на предмет появления в ней видеофайла, который будет принят далее в работу.

extract_audiotrack_from_video – осуществляет извлечение аудиодорожки из исходного видеофайла для дальнейшей работы. Для извлечения аудиодорожки из видео была использована библиотека ffmpeg, которая была получена из Docker-образа jrottenberg/ffmpeg.

transform_audiotrack_to_text – осуществляет обработку, распознавание и трансформацию аудиофайла в текстовый файл. Данная операция осуществлялась при помощи запросов в сервис huggingface.

resume_text_from_audiotrack – осуществляет суммаризацию текстового файла, который получен на предыдущих этапах.

save_get_text_from_txt_to_pdf – осуществляет сохранение полученного результата в файл формата pdf.



```
1 from airflow.providers.docker.operators.docker import DockerOperator
2 from airflow.sensors.filesystem import FileSensor
3
4
5
6
7 default_args = {
8     'owner': 'ikasp',
9     'start_date': datetime(2024, 11, 29),
10 }
11
12 dag = DAG(
13     'k2_1',
14     default_args=default_args,
15     schedule_interval=None,
16 )
17
18 wait_get_new_videofile = FileSensor(
19     task_id='wait_get_new_videofile',
20     poke_interval=15,
21     filepath='/opt/airflow/data',
22     fs_conn_id='fs_connection_default',
23     dag=dag,
24 )
25
26 extract_audiotrack_from_video = DockerOperator(
27     task_id='extract_audiotrack_from_video',
28     image='jrottenberg/ffmpeg',
29     command='1 /data/idiot.mp4 -vn -acodec copy /data/audio.aac',
30     mounts=[Mount(source='/data', target='/data', type='bind')],
31     docker_url='tcp://docker-proxy:2375',
32     dag=dag,
33 )
34
35 transform_audiotrack_to_text = DockerOperator(
36     task_id='transform_audiotrack_to_text',
37     image='nyurik/alpine-python3-requests',
38     command='python /data/transform_audiotrack_to_text.py',
39     mounts=[Mount(source='/data', target='/data', type='bind')],
40     docker_url='tcp://docker-proxy:2375',
41     dag=dag,
42     mount_tmp_dir=False,
43 )
44
45 resume_text_from_audiotrack = DockerOperator(
46     task_id='resume_text_from_audiotrack',
47     image='nyurik/alpine-python3-requests',
48     command='python /data/resume_text_from_audiotrack.py',
49     mounts=[Mount(source='/data', target='/data', type='bind')],
50     docker_url='tcp://docker-proxy:2375',
51     dag=dag,
```

Шаг 5. Запуск DAG-а.

Теперь после всех необходимых настроек и приготовлений, мы можем запустить наш DAG. Для этого переходим в airflow: <http://localhost:8080/home> и находим наш DAG.

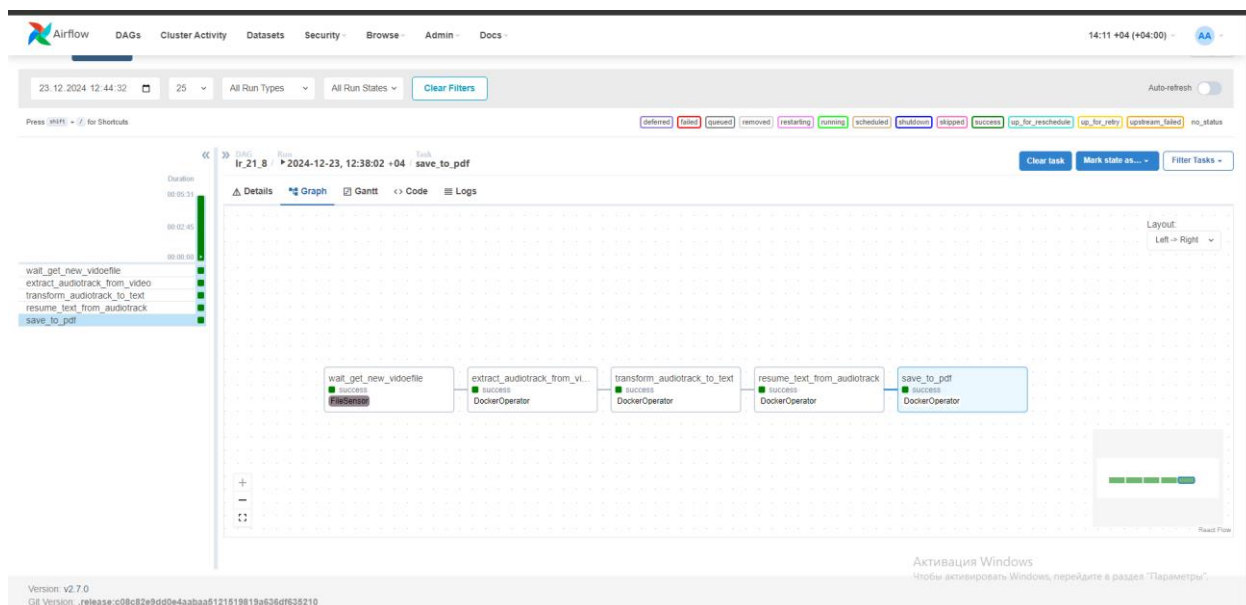


Рисунок 5 – Запуск DAG-а.

В качестве исходного видео использовался фрагмент из известного автомобильного шоу «The Grand Tour». На фрагменте видео один из ведущих тестировал голосового ассистента в машине, который трансформировал сказанное в сообщении, после чего отправлял выбранному контакту. Очевидно, что в данной лабораторной работе распознавание аудио-фрагмента происходит лучше.

Мы получаем аудиодорожку, которая используется в качестве основы для получения текстового файла.

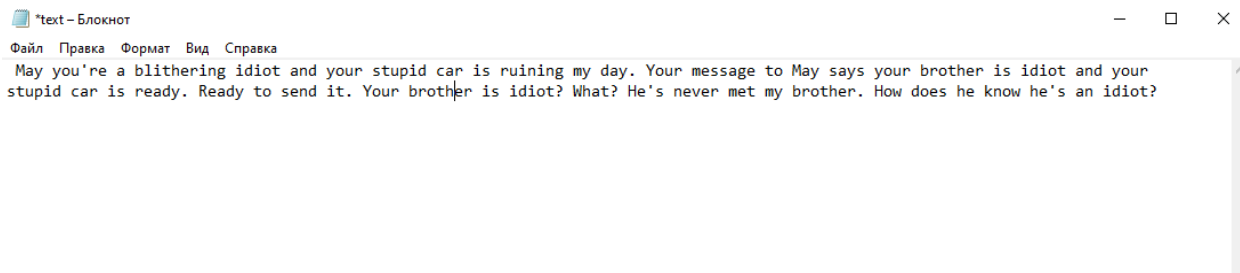


Рисунок 6 – Результат работы huggingface по преобразованию аудио в текст

Далее полученный результат мы еще раз передавали huggingface для получения уже конспекта по отправленному файлу. Полученный результат записывали pdf-файл.

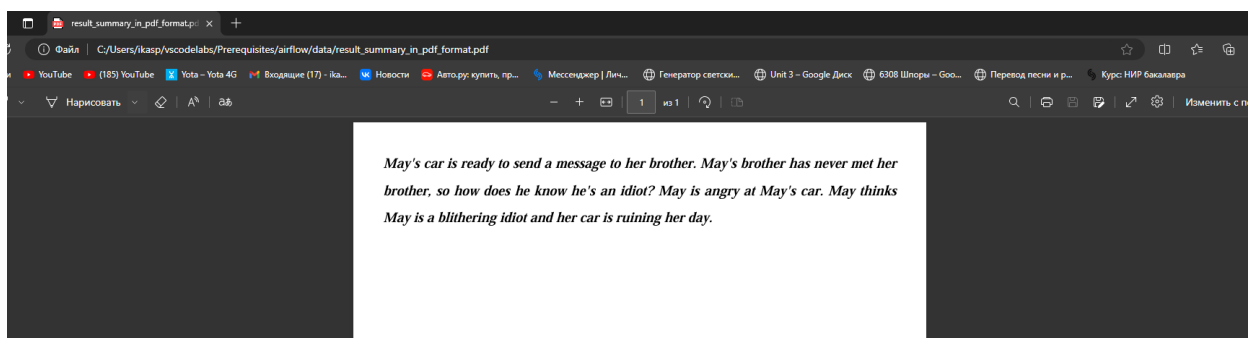


Рисунок 7 – Конспект текстового файла.

Часть 2. Пайплайн, который реализует систему автоматического обучения/дообучения нейросетевой модели

В рамках второй части лабораторной работы нам необходимо было разработать пайплайн, который реализует систему автоматического обучения/дообучения нейросетевой модели.

Шаг 1. Разработка DAG

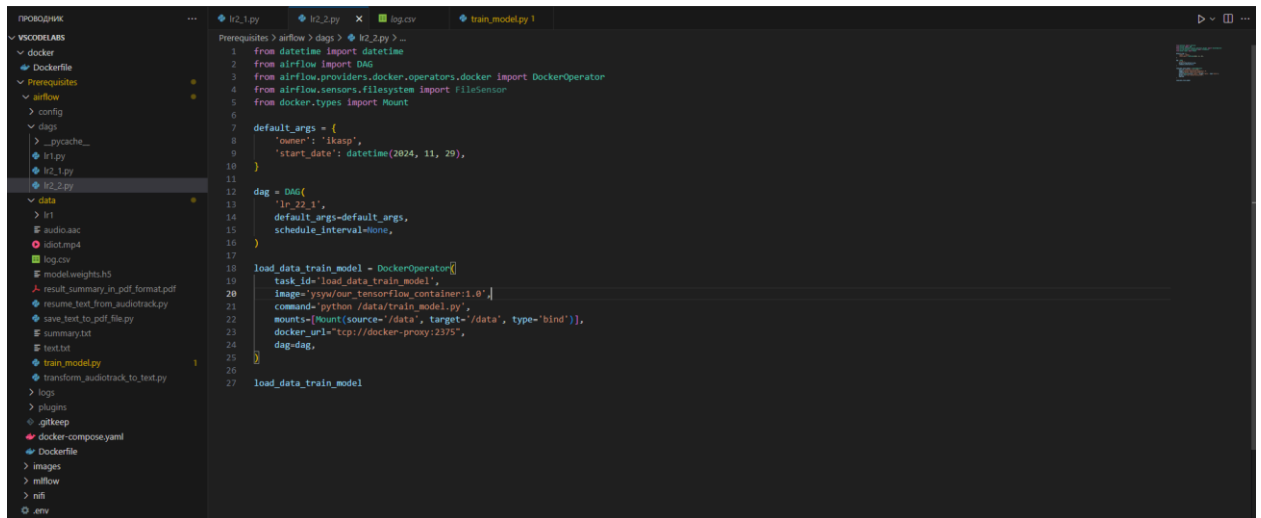


Рисунок 8 – Пайплайн

DAG запускал код, который получал датасет вин `load_wine` из `sklearn.datasets`, после чего мы проводили разбиение данных. Которые передаются в нейросеть, после чего модель проходит обучение. Процесс обучения логируется.

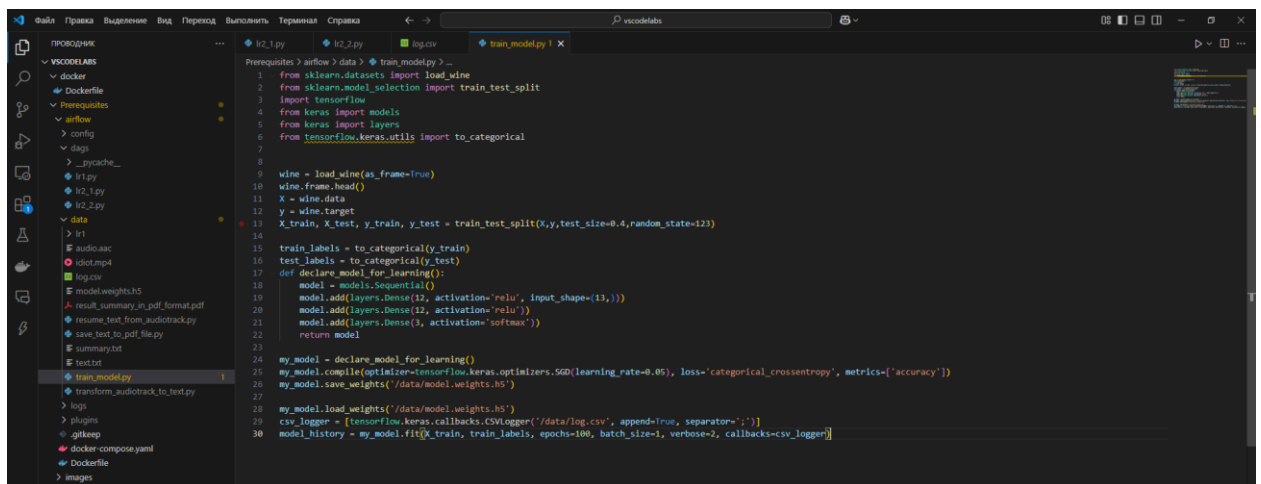


Рисунок 9 – Код обучения модели.

Шаг 2. Запуска DAG-а.

В процессе запуска DAG-а модель была обучена и показала результаты, которые мы записали в файл. В итоге получился вот такой лог обучения:

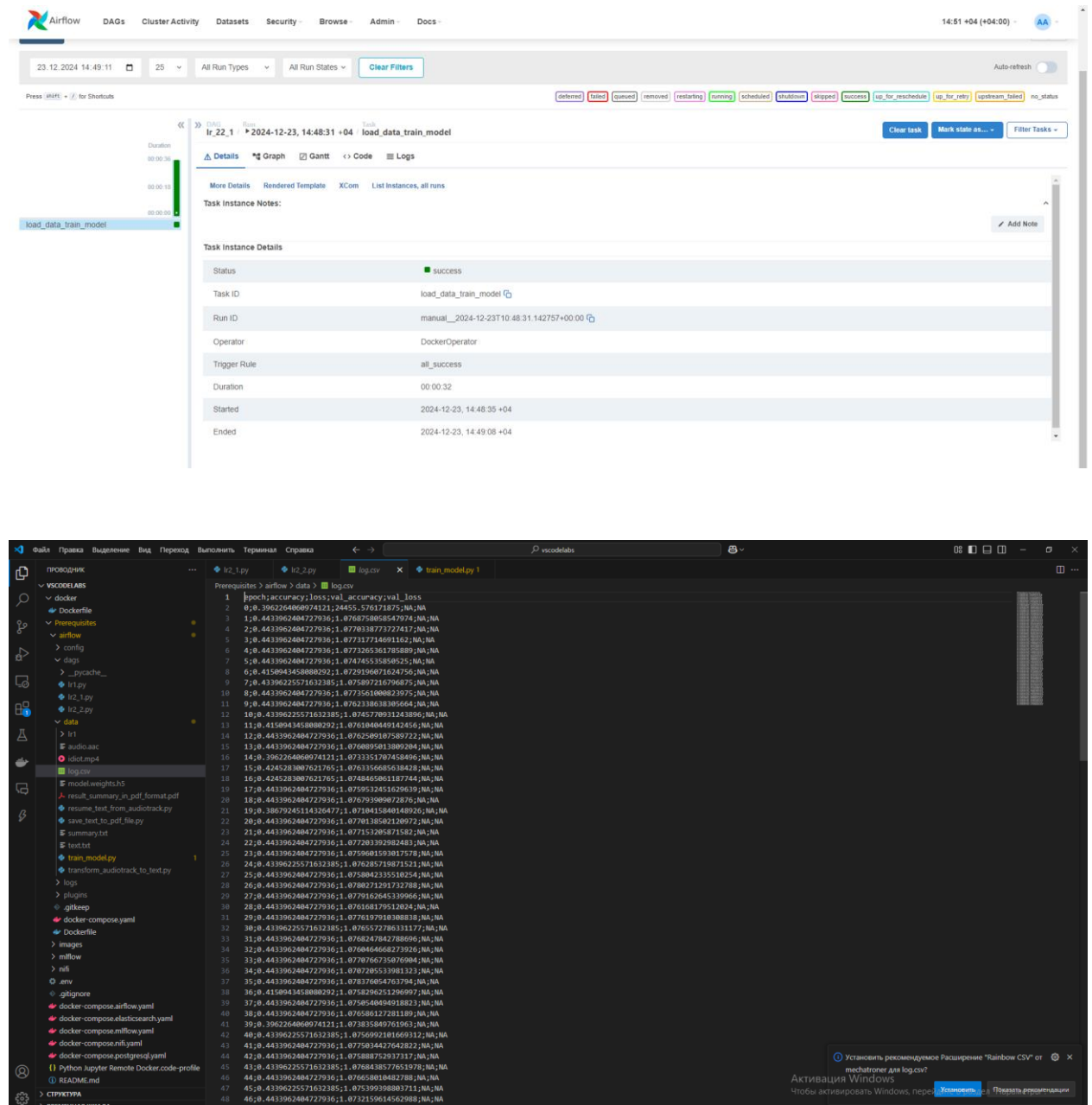


Рисунок 15 – Лог процесса обучения нейросети.

Заключение

В ходе выполнения лабораторной работы получены навыки:

1. Работа с DAG в Airflow
2. Работа с сетями на huggingface