

# Андреев Александр 6233

Маленькое замечание: в репозитории отсутствует файл X\_train.csv, т.к. он имеет размер 480 МБ, а GitHub позволяет загружать файлы размером до 25 МБ.

## Первый pipeline

Работа началась стандартно с использованием рекомендованных image'ей преподавателем.

Для начала были настроены connections в airflow, далее был получен токен для записи на сайте Hugging Face, а впоследствии с использованием открытых images для докера были проведены выделение аудио из видео, его обработка и запись в pdf файл (пришлось написать свой image).

Выделение аудио из видео прошло успешно.

```
extract_audio = DockerOperator(  
    task_id='extract_audio',  
    image='jrottenberg/ffmpeg',  
    command='-i /data/input_video.mp4 -vn -acodec copy /data/audio.aac',  
    mounts=[Mount(source='/data', target='/data', type='bind')],  
    docker_url="tcp://docker-proxy:2375",  
    dag=dag,  
)
```

Рисунок 1 – Выделение аудио из видео

Преобразование из аудио в текст прошло успешно лишь отчасти, так как по какой-то причине распознавание идет толи не полностью, толи пропускаются слова. Но я считаю, что это не проблема в рамках данной ЛР, ведь я должен использовать сторонний инструмент, а не написать свой. Поэтому имеем то, что имеем – распознается лишь часть текста, но она в видео действительно есть, это не случайный набор слов.

Скрипты, которые используются на скрине ниже представлены в репозитории.

```
transform_audio_to_text = DockerOperator(  
    task_id='transform_audio_to_text',  
    image='nyurik/alpine-python3-requests',  
    command='python /data/audio_to_text.py',  
    mounts=[Mount(source='/data', target='/data', type='bind')],  
    docker_url="tcp://docker-proxy:2375",  
    dag=dag,  
)  
  
summarize_text = DockerOperator(  
    task_id='summarize_text',  
    image='nyurik/alpine-python3-requests',  
    command='python /data/text_to_summ.py',  
    mounts=[Mount(source='/data', target='/data', type='bind')],  
    docker_url="tcp://docker-proxy:2375",  
    dag=dag,  
)
```

## Рисунок 2 – Преобразование аудио в текст

Далее была проблема с преобразованием текста в pdf. Стандартные средства, которые были использовались в этом коде для преобразования в формат pdf не работали.

```
save_to_pdf = DockerOperator(
    task_id='save_to_pdf',
    image='mashupmill/text2pdf',
    command='text2pdf /data/summ.txt > /data/summ.pdf',
    mounts=[Mount(source='/data', target='/data', type='bind')],
    docker_url="tcp://docker-proxy:2375",
    dag=dag,
)
```

## Рисунок 3 – Неработающий код преобразования в pdf

Чтобы это исправить был написан свой скрипт с использованием библиотеки fpdf и создан свой образ (так как мне не удалось быстро найти образ на DockerHub с fpdf библиотекой) для решения этой задачи, который также используется в DAG файле.

```
C: > Users > sasha > Desktop > Mara > 2 год > DE > Prerequisites > airflow > data > save_to_pdf.py > ...

1  from fpdf import FPDF
2
3  file = open("/data/summ.txt","r")
4  pdf = FPDF()
5  pdf.add_page()
6  for text in file:
7      if len(text) <= 20:
8          pdf.set_font("Arial","B",size=18)
9          pdf.cell(w=200,h=10, txt=text, ln=1,align="C")
10     else:
11         pdf.set_font("Arial",size=15)
12         pdf.multi_cell(w=0,h=10, txt=text,align="L")
13     pdf.output("/data/output.pdf")
```

## Рисунок 4 – Скрипт для преобразования файла в pdf

Dockerfile – Блокнот

Файл Правка Формат Вид Справка

```
#Deriving the latest base image
FROM python:latest
```

```
RUN pip install numpy fpdf
```

```
#Labels as key value pair
LABEL Maintainer="sasha151299.fpdf"
```

```
# Any working directory can be chosen as per choice like '/' or '/home' etc
# i have chosen /usr/app/src
WORKDIR /usr/app/src
```

Рисунок 5 – Dockerfile для создание своего image файла с библиотекой fpdf

```
55 save_to_pdf = DockerOperator(  
56     task_id='save_to_pdf',  
57     image='sasha151299/my_pdf:1.0',  
58     command='python /data/save_to_pdf.py',  
59     mounts=[Mount(source='/data', target='/data', type='bind')],  
60     docker_url="tcp://docker-proxy:2375",  
61     dag=dag,  
62 )  
63  
64 wait_for_new_file >> extract_audio >> transform_audio_to_text >> summarize_text >> save_to_pdf
```

Рисунок 6 – Использование скрипта в dag файле

Итого мы имеем запущенный файл в airflow и несколько файлов в входной-выходной папке (в конце все файлы, кроме output.pdf можно было бы удалить, но я этого не делал).

Скрин со всеми файлами в папке будет приложен в самом конце отчета, так как в папке есть скрипты и файла для второй части ЛР.

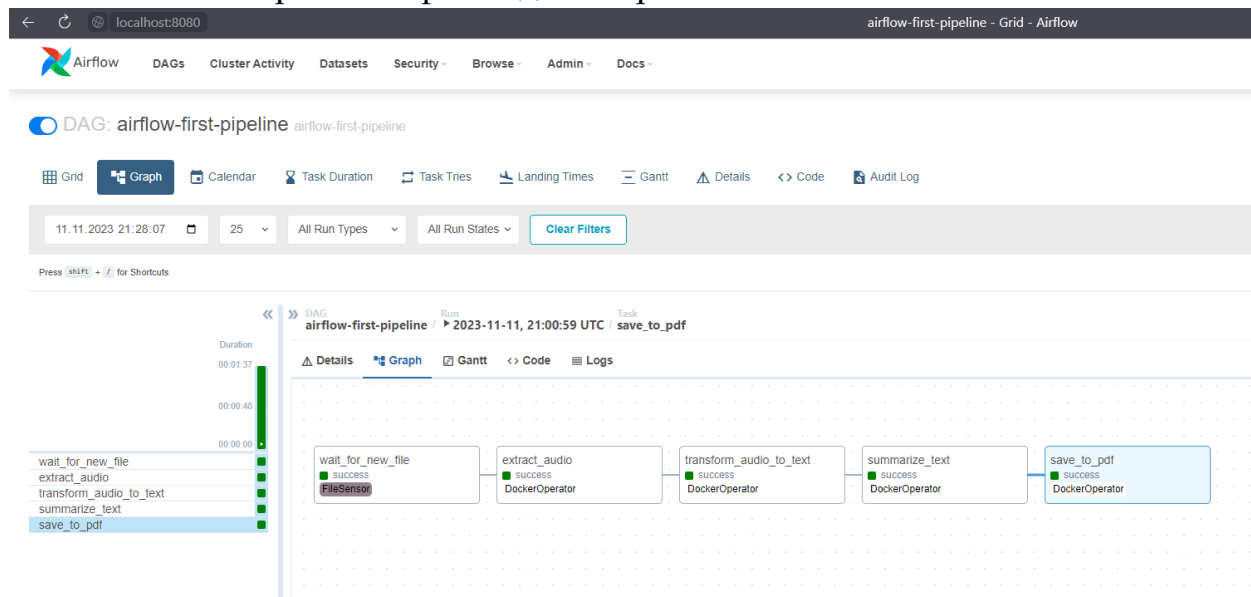


Рисунок 7 – Запущенный DAG в airflow

## Второй pipeline

В качестве решения второй задачи нам не предлагается никакой датасет и так как моя НИР не связана с сетями мною был выбран код с другого предмета этого семестра, где мы сами писали простую нейронную сеть с использованием TensorFlow и пытались ее обучить на двух датасетах (я обучал ее на mnist). Коды для чтения данных и обучения приложены в репозитории.

```
stricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

Andreev_6233_Lab4_ipynb lab1.py 5 airflow-second-pipeline.py 3 X airflow-first-pipeline.py 4

C: > Users > sasha > Desktop > Mara > 2 год > DE > Prerequisites > airflow > dags > airflow-second-pipeline.py > ...

1 from datetime import datetime
2 from airflow import DAG
3 from docker.types import Mount
4 from airflow.providers.docker.operators.docker import DockerOperator
5
6 default_args = {
7     'owner': 'airflow',
8     'start_date': datetime(2023, 1, 1),
9     'retries': 1,
10 }
11
12 dag = DAG(
13     'airflow_second_pipeline',
14     default_args=default_args,
15     description='airflow second pipeline',
16     schedule_interval=None,
17 )
18
19 read_data = DockerOperator(
20     task_id='read_data',
21     image='sasha151299/second_pipeline:1.0',
22     command='python /data/read_data.py',
23     mounts=[Mount(source='/data', target='/data', type='bind')],
24     docker_url="tcp://docker-proxy:2375",
25     dag=dag,
26 )
27
28 train = DockerOperator(
29     task_id='train',
30     image='sasha151299/second_pipeline:1.0',
31     command='python /data/train.py',
32     mounts=[Mount(source='/data', target='/data', type='bind')],
33     docker_url="tcp://docker-proxy:2375",
34     dag=dag,
35 )
36
37 read_data >> train
```

Рисунок 8 – Код DAG файла для второй части работы

Для этой части работы на просторах DockerHub мне не удалось образ, который содержал бы в себе TensorFlow, pandas и numpy, поэтому я снова сделал свой образ (использовал следующую инструкцию <https://linuxhint.com/push-and-pull-docker-image-to-docker-hub/#b>).

```

Dockerfile – Блокнот
Файл  Правка  Формат  Вид  Справка
#Deriving the latest base image
FROM tensorflow/tensorflow:latest

RUN pip install numpy pandas

#Labels as key value pair
LABEL Maintainer="sasha151299.second_pipeline"

# Any working directory can be chosen as per choice like '/' or '/home' etc
# i have chosen /usr/app/src
WORKDIR /usr/app/src

```

Рисунок 9 – Dockerfile для второй части работы

Удивительно, но в этой части лабораторной работы проблем не возникло (кроме нейминга файлов – failed запуски). DAG файл также успешно запустился и отработал.

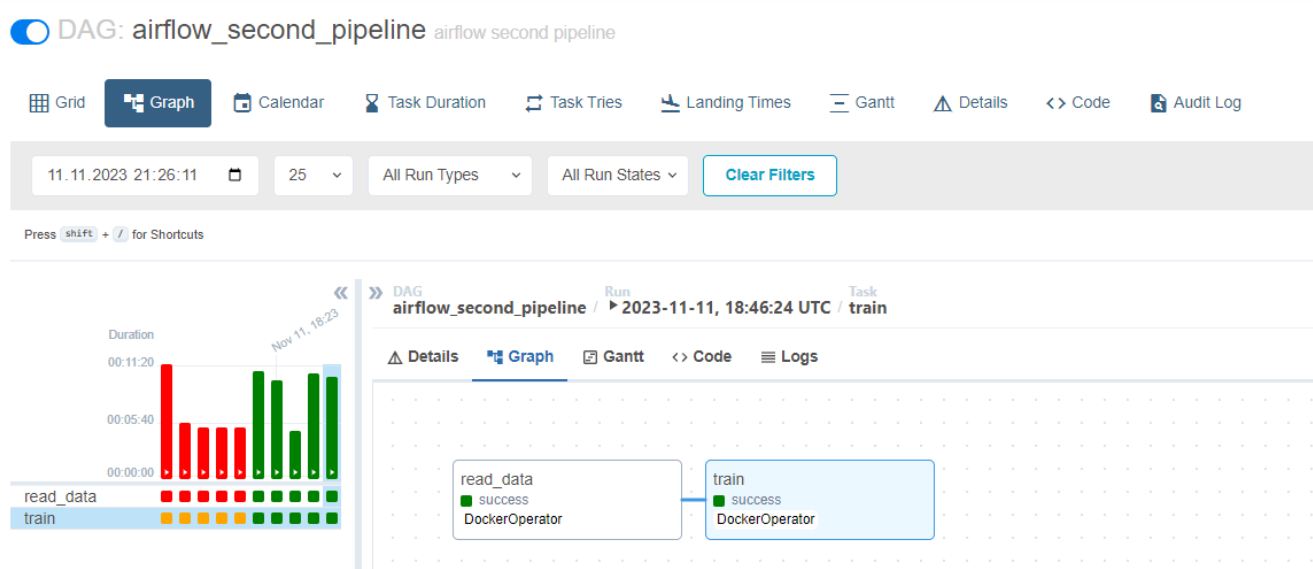


Рисунок 10 – Запущенный DAG в airflow для обучения нейронной сети

Результаты обучения нейронной сети – ассигасу на тренировочной и валидационной выборках и соответствующие loss представлены в выходном файле после обучения (как требовалось в лабораторной работе).

```
*train.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
Iteration 0, Epoch 1, Loss: 2.8507091999053955, Accuracy: 9.375, Val Loss: 2.9319071769714355, Val Accuracy: 24.0
Iteration 100, Epoch 1, Loss: 0.5858732461929321, Accuracy: 81.69863891601562, Val Loss: 0.4266330599784851, Val Accuracy: 87.5
Iteration 200, Epoch 1, Loss: 0.4498618245124817, Accuracy: 86.2018051147461, Val Loss: 0.35591527819633484, Val Accuracy: 89.30000305175781
Iteration 300, Epoch 1, Loss: 0.36324310302734375, Accuracy: 88.91715240478516, Val Loss: 0.1623111218214035, Val Accuracy: 94.19999694824219
Iteration 400, Epoch 1, Loss: 0.3065120279788971, Accuracy: 90.63668823242188, Val Loss: 0.2144419252872467, Val Accuracy: 92.79999542236328
Iteration 500, Epoch 1, Loss: 0.27495938539505005, Accuracy: 91.65731048583984, Val Loss: 0.1704723984003067, Val Accuracy: 94.80000305175781
Iteration 600, Epoch 1, Loss: 0.2503967583179474, Accuracy: 92.40068817138672, Val Loss: 0.18565353751182556, Val Accuracy: 94.5999984741211
Iteration 700, Epoch 1, Loss: 0.23237143456935883, Accuracy: 92.94088745117188, Val Loss: 0.13284637033939362, Val Accuracy: 96.5999984741211
Iteration 800, Epoch 2, Loss: 0.11665228754281998, Accuracy: 96.5625, Val Loss: 0.12090404331684113, Val Accuracy: 96.80000305175781
Iteration 900, Epoch 2, Loss: 0.09723256528377533, Accuracy: 97.25694274902344, Val Loss: 0.1147625669836998, Val Accuracy: 96.20000457763672 |
. . . . . (в репо будет полный файл - скрыто для скрина в отчет)
Iteration 6900, Epoch 10, Loss: 0.010151149705052376, Accuracy: 99.77678680419922, Val Loss: 0.09079189598560333, Val Accuracy: 97.39999389648438
Iteration 7000, Epoch 10, Loss: 0.023074693977832794, Accuracy: 99.19684600830078, Val Loss: 0.10279592871665955, Val Accuracy: 97.5999984741211
Iteration 7100, Epoch 10, Loss: 0.02097945287823677, Accuracy: 99.27536010742188, Val Loss: 0.0845332071185112, Val Accuracy: 97.0999984741211
Iteration 7200, Epoch 10, Loss: 0.01998185328888893, Accuracy: 99.27728271484375, Val Loss: 0.10293712466955185, Val Accuracy: 96.9000015258789
Iteration 7300, Epoch 10, Loss: 0.01992342621088028, Accuracy: 99.29745483398438, Val Loss: 0.11391770839691162, Val Accuracy: 97.0
Iteration 7400, Epoch 10, Loss: 0.020258478820323944, Accuracy: 99.29117584228516, Val Loss: 0.10151270031929016, Val Accuracy: 96.69999694824219
Iteration 7500, Epoch 10, Loss: 0.02077578380703926, Accuracy: 99.26637268066406, Val Loss: 0.10792820900678635, Val Accuracy: 97.0
Iteration 7600, Epoch 10, Loss: 0.02091602236032486, Accuracy: 99.26847839355469, Val Loss: 0.07339949160814285, Val Accuracy: 97.79999542236328

Стр 10, стрнб 143  120%  UNIX (LF)  UTF-8
```

Рисунок 11 – Результаты обучения нейронной сети

По итогу выполнения лабораторной работы имеется два созданных образа, два созданных и отработавших DAG файла и несколько промежуточных и финальных файлов в папке.

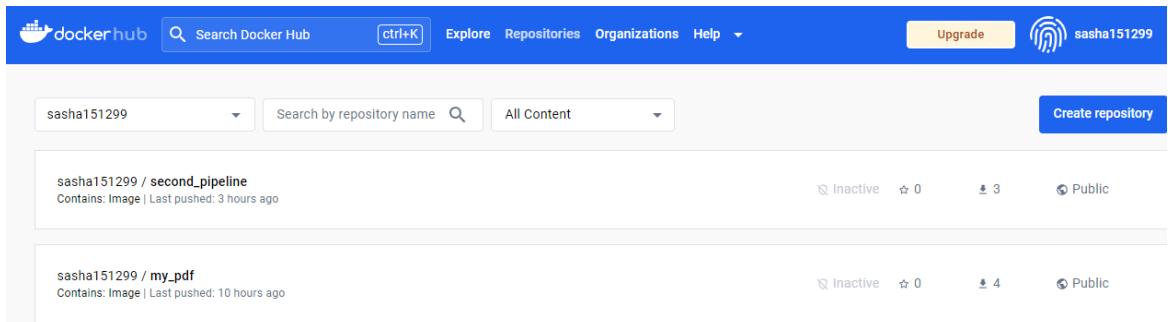


Рисунок 12 – Images для лабораторной работы

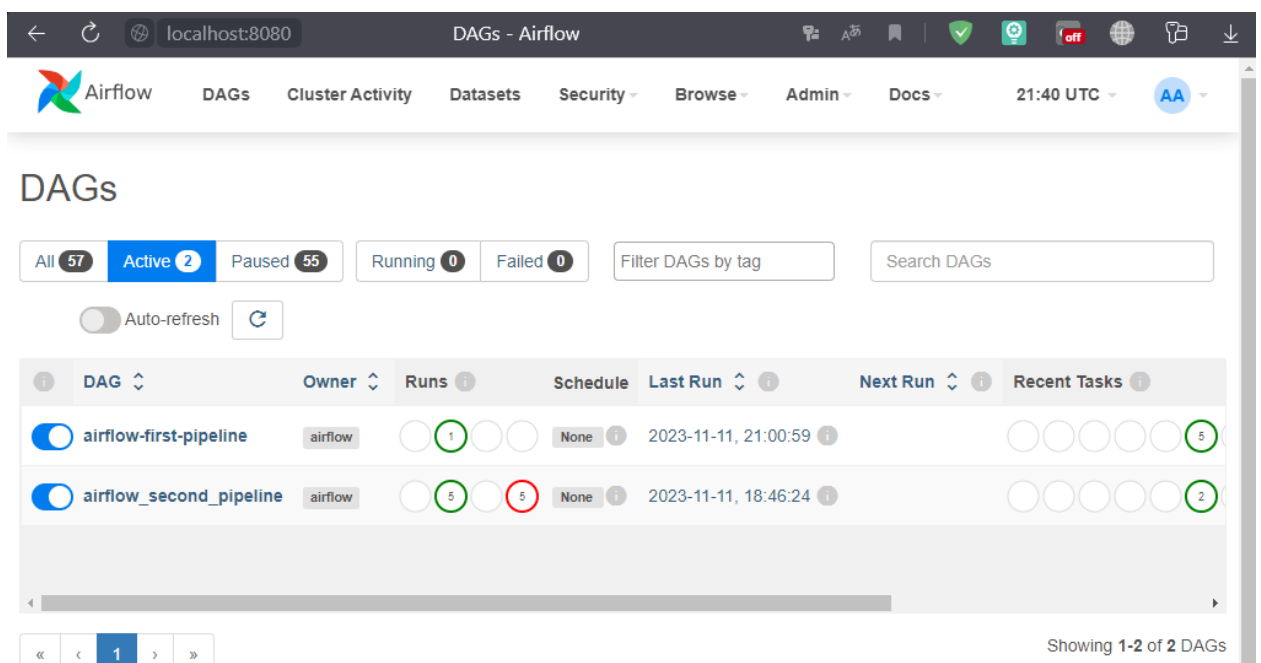


Рисунок 13 – DAG файлы в airflow

