



به نام خدا

تمرین درس تحلیل و سیستم های داده های حجیم

دانشجو:

پریا تاری

شماره دانشجویی:

40211415012

استاد مربوطه:

دکتر آرمین رشنو

تیر 1403

CNN

```
1 from fastapi import FastAPI, File, UploadFile
2 from fastapi.responses import JSONResponse
3 from tensorflow.keras.models import load_model
4 from PIL import Image
5 import io
6 import tensorflow as tf
7 import os
8 import numpy as np
9
10 app = FastAPI()
11
12 # CIFAR-10 class names
13 class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
14                'dog', 'frog', 'horse', 'ship', 'truck']
15
```

این خطها کتابخانه های مورد نیاز و نامهای دسته های مختلف موجود در دیتاست CIFAR-10 را تعریف می کنند.

```
17 model_path = 'cifar10_model.h5'
18
19 if not os.path.exists(model_path):
20     # Load CIFAR-10 dataset from local directory
21     usage new *
22     def load_local_cifar10(path):
23         new *
24         def unpickle(file):
25             import pickle
26             with open(file, 'rb') as fo:
27                 dict = pickle.load(fo, encoding='bytes')
28                 return dict
29
30         train_data = []
31         train_labels = []
32         for i in range(1, 6):
33             batch = unpickle(os.path.join(path, 'data_batch_' + str(i)))
34             train_data.append(batch[b'data'])
35             train_labels += batch[b'labels']
36         train_data = np.concatenate(train_data)
37         train_data = train_data.reshape((50000, 32, 32, 3), order='F')
38         train_labels = np.array(train_labels)
39
40         test_batch = unpickle(os.path.join(path, 'test_batch'))
41         test_data = test_batch[b'data'].reshape((10000, 32, 32, 3), order='F')
42         test_labels = np.array(test_batch[b'labels'])
43
44         return (train_data, train_labels), (test_data, test_labels)
45
```

خط 17 مسیر فایل مدل ذخیره شده را مشخص می کند.

خط 19 بررسی می کند که آیا فایل مدل ذخیره شده وجود دارد یا خیر. اگر وجود نداشته باشد، کد زیرش اجرا می شود تا مدل را از ابتدا آموزش دهد.

تابع `load_local_cifar10` برای بارگذاری داده ها از فایل های پیکلی شده. این تابع از ماژول `pickle` برای خواندن داده ها استفاده می کند.

خطوط 30 تا 36 داده های آموزشی را از پنج فایل `data_batch` بارگذاری می کند، آنها را با هم ترکیب کرده و به شکل مناسب $(32 \times 32 \times 3)$ تغییر می دهد.

خطوط 38 تا 40 داده های تست را از فایل `test_batch` بارگذاری و به شکل مناسب تغییر می دهد.

خط 42 داده های آموزشی و تست را به عنوان خروجی تابع بازمی گرداند.

```

44
45 (train_images, train_labels), (test_images, test_labels) = load_local_cifar10('./cifar-10-batches-py')
46 train_images, test_images = train_images / 255.0, test_images / 255.0
47
48 # Define the model with additional Conv2D layers and Dropout
49 model = tf.keras.Sequential([
50     tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
51     tf.keras.layers.BatchNormalization(),
52     tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'),
53     tf.keras.layers.BatchNormalization(),
54     tf.keras.layers.MaxPooling2D((2, 2)),
55     tf.keras.layers.Dropout(0.25),
56
57     tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
58     tf.keras.layers.BatchNormalization(),
59     tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
60     tf.keras.layers.BatchNormalization(),
61     tf.keras.layers.MaxPooling2D((2, 2)),
62     tf.keras.layers.Dropout(0.25),
63
64     tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
65     tf.keras.layers.BatchNormalization(),
66     tf.keras.layers.MaxPooling2D((2, 2)),
67     tf.keras.layers.Dropout(0.25),
68
69     tf.keras.layers.Flatten(),
70     tf.keras.layers.Dense(units=256, activation='relu'),
71     tf.keras.layers.Dropout(0.5),
72     tf.keras.layers.Dense(10)
73 ])
74

```

داده‌های آموزشی و تست را بارگذاری کرده و آنها را به محدوده [0, 1] نرمالایز می‌کند.

مدل شبکه عصبی با استفاده از Keras تعریف می‌شود که شامل چندین لایه Conv2D، BatchNormalization، MaxPooling2D، Dropout و Dense است.

خط 50 لایه اول:

نوع لایه: کانولوشن دو بعدی (Conv2D)

تعداد فیلترها: 32

اندازه فیلترها: 3 x 3

تابع فعال‌سازی: ReLU (Rectified Linear Unit)

شکل ورودی: 32x32x3 (تصاویر رنگی 32 x 32 با 3 کانال رنگی)

این لایه 32 فیلتر 3x3 را بر روی تصاویر ورودی اعمال می‌کند و خروجی را از تابع فعال‌سازی ReLU عبور می‌دهد.

خط 51 لایه دوم:

نوع لایه: نرمال‌سازی دسته‌ای (Batch Normalization)

این لایه نرمال‌سازی دسته‌ای را انجام می‌دهد که به تسریع آموزش و پایداری شبکه کمک می‌کند.

خط 52 لایه سوم:

نوع لایه: کانولوشن دو بعدی (Conv2D)

تعداد فیلترها: 32

اندازه فیلترها: 3×3

تابع فعال‌سازی: ReLU (Rectified Linear Unit)

این لایه 32 فیلتر 3×3 دیگر را بر روی خروجی لایه قبلی اعمال می‌کند و سپس خروجی را از تابع فعال‌سازی ReLU عبور می‌دهد.

خط 53 لایه چهارم:

نوع لایه: نرمال‌سازی دسته‌ای (Batch Normalization)

این لایه نرمال‌سازی دسته‌ای را انجام می‌دهد.

خط 54 لایه پنجم:

نوع لایه: مکس‌پولینگ دو بعدی (MaxPooling2D)

اندازه پنجره: 2×2

این لایه اندازه مکانی خروجی را با استفاده از عملیات مکس‌پولینگ 2×2 کاهش می‌دهد.

خط 55 لایه ششم:

نوع لایه: دراپ‌اوت (Dropout)

نرخ دراپ‌اوت: 0.25 (25%)

این لایه به منظور جلوگیری از بیش‌برازش (overfitting) در طول آموزش، به صورت تصادفی 25% از نورون‌ها را در هر گام آموزشی نادیده می‌گیرد.

خط 57 لایه هفتم:

نوع لایه: کانولوشن دو بعدی (Conv2D)

تعداد فیلترها: 64

اندازه فیلترها: 3×3

تابع فعال‌سازی: ReLU (Rectified Linear Unit)

این لایه 64 فیلتر 3×3 را بر روی خروجی لایه قبلی اعمال می‌کند و سپس خروجی را از تابع فعال‌سازی ReLU عبور می‌دهد.

خط 58 لایه هشتم:

نوع لایه: نرمال‌سازی دسته‌ای (Batch Normalization)

این لایه نرمال‌سازی دسته‌ای را انجام می‌دهد.

خط 59 لایه نهم:

نوع لایه: کانولوشن دو بعدی (Conv2D)

تعداد فیلترها: 64

اندازه فیلترها: 3×3

تابع فعال‌سازی: ReLU (Rectified Linear Unit)

این لایه 64 فیلتر 3×3 دیگر را بر روی خروجی لایه قبلی اعمال می‌کند و سپس خروجی را از تابع فعال‌سازی ReLU عبور می‌دهد.

خط 60 لایه دهم:

نوع لایه: نرمال‌سازی دسته‌ای (Batch Normalization)

این لایه نرمال‌سازی دسته‌ای را انجام می‌دهد.

خط 61 لایه یازدهم:

نوع لایه: ماکس‌پولینگ دو بعدی (MaxPooling2D)

اندازه پنجره: 2×2

این لایه اندازه مکانی خروجی را با استفاده از عملیات ماکس‌پولینگ 2×2 کاهش می‌دهد.

خط 62 لایه دوازدهم:

نوع لایه: دراپ‌اوت (Dropout)

نرخ دراپ‌اوت: (25%) 0.25

این لایه به منظور جلوگیری از بیش‌برازش (overfitting) در طول آموزش، به صورت تصادفی 25% از نورون‌ها را در هر گام آموزشی نادیده می‌گیرد.

خط 64 لایه سیزدهم:

نوع لایه: کانولوشن دو بعدی (Conv2D)

تعداد فیلترها: 128

اندازه فیلترها: 3×3

تابع فعال‌سازی: ReLU (Rectified Linear Unit)

این لایه 128 فیلتر 3×3 دیگر را بر روی خروجی لایه قبلی اعمال می‌کند و سپس خروجی را از تابع فعال‌سازی ReLU عبور می‌دهد.

خط 65 لایه چهاردهم:

نوع لایه: نرمال‌سازی دسته‌ای (Batch Normalization)

این لایه نرمال‌سازی دسته‌ای را انجام می‌دهد.

خط 66 لایه پانزدهم:

نوع لایه: ماکس‌پولینگ دو بعدی (MaxPooling2D)

اندازه پنجره: 2×2

این لایه اندازه مکانی خروجی را با استفاده از عملیات ماکس‌پولینگ 2×2 کاهش می‌دهد.

خط 67 لایه شانزدهم:

نوع لایه: دراپ‌اوت (Dropout)

نرخ دراپ‌اوت: (25%) 0.25

این لایه به منظور جلوگیری از بیش‌برازش (overfitting) در طول آموزش، به صورت تصادفی 25% از نورون‌ها را در هر گام آموزشی نادیده می‌گیرد.

خط 69 لایه هفدهم:

نوع لایه: تخت کردن (Flatten)

این لایه خروجی سه بعدی لایه‌های قبلی را به یک بردار یک بعدی تبدیل می‌کند که برای لایه‌های Dense قابل استفاده باشد.

خط 70 لایه هجدهم:

نوع لایه: Dense

تعداد نرون‌ها: 256

تابع فعال‌سازی: ReLU (Rectified Linear Unit)

این لایه یک لایه کاملاً متصل (Dense) با 256 نرون و تابع فعال‌سازی ReLU است.

خط 71 لایه نوزدهم:

نوع لایه: دراپ‌اوت (Dropout)

نرخ دراپ‌اوت: 0.5 (50%)

این لایه به منظور جلوگیری از بیش‌برازش (overfitting) در طول آموزش، به صورت تصادفی 50% از نرون‌ها را در هر گام آموزشی نادیده می‌گیرد.

خط 72 لایه بیستم:

نوع لایه: Dense

تعداد نرون‌ها: 10

این لایه آخرین لایه Dense است که 10 نرون دارد، هر کدام مربوط به یکی از 10 کلاس CIFAR-10 است. تابع فعال‌سازی در اینجا تعریف نشده، زیرا از logits برای محاسبه تابع هزینه استفاده خواهد شد.

```

74
75     # Compile the model
76     model.compile(optimizer='adam',
77                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
78                   metrics=['accuracy'])
79
80     # Train the model
81     model.fit(train_images, train_labels, epochs=70, validation_data=(test_images, test_labels))
82
83     # Save the model
84     model.save(model_path)
85
86     # Evaluate the model on test data
87     _, test_accuracy = model.evaluate(test_images, test_labels)
88     print(f"Test accuracy: {test_accuracy}")
89
90 else:
91     model = load_model(model_path)
92

```

در خط 76 مدل با استفاده از بهینه‌ساز Adam، تابع هزینه SparseCategoricalCrossentropy و معیار دقت، کامپایل می‌شود.

در خط 81 مدل به مدت 70 دوره (epoch) بر روی داده‌های آموزشی آموزش داده می‌شود و داده‌های تست برای ارزیابی استفاده می‌شوند.

در خط 84 مدل آموزش داده شده در مسیر مشخص شده ذخیره می‌شود.

در خط 87 مدل بر روی داده‌های تست ارزیابی می‌شود و دقت آن چاپ می‌شود.

در خط 90 اگر مدل ذخیره شده وجود داشته باشد، مدل از فایل بارگذاری می‌شود.

```

new *
94 @app.post("/predict/")
95 async def predict(file: UploadFile = File(...)):
96     # Read the image file
97     image = await file.read()
98     image = Image.open(io.BytesIO(image)).convert("RGB")
99     image = image.resize((32, 32))
100    image = np.array(image) / 255.0
101    image = np.expand_dims(image, axis=0)
102
103    # Predict the class of the image
104    predictions = model.predict(image)
105    predicted_class = class_names[np.argmax(predictions[0])]
106
107    return JSONResponse(content={"class": predicted_class})
108

```


یک API تعریف می‌شود که تصویری را از کاربر دریافت کرده، آن را به شکل مناسب تغییر می‌دهد، پیش‌بینی مدل را انجام داده و دسته‌بندی پیش‌بینی شده را برمی‌گرداند.

<http://103.216.61.177:8000/predict/>

POST <http://localhost:8000/predict/> Send

200 OK TIME 281 ms SIZE 16 B

Multipart 1 Auth Query Header 1 Docs

file 

name value

name value

New name New value


Preview Header 4 Cookie Timeline

```
1 {  
2   "class": "frog"  
3 }
```

POST <http://localhost:8000/predict/> Send

200 OK TIME 360 ms SIZE 16 B

Multipart 1 Auth Query Header 1 Docs

file 

name value

name value

New name New value

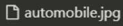
Preview Header 4 Cookie Timeline

```
1 {  
2   "class": "bird"  
3 }
```

POST <http://localhost:8000/predict/> Send

200 OK TIME 329 ms SIZE 22 B

Multipart 1 Auth Query Header 1 Docs

file 

name value

name value

New name New value

Preview Header 4 Cookie Timeline

```
1 {  
2   "class": "automobile"  
3 }
```


خروجی با سرور

POST ▼ http://45.252.182.94:8000/predict/ Send

200 OK TIME 813 ms SIZE 16 B

Multipart 1 ▼ Auth ▼ Query Header 1 Docs

file deer.jpg ▼ ✓ 🗑

name value ▼ □ 🗑

New name New value

Preview ▼ Header 4 ▼ Cookie Timeline

```
1 {  
2   "class": "deer"  
3 }
```

POST ▼ http://45.252.182.94:8000/predict/ Send

200 OK TIME 3.25 s SIZE 16 B

Multipart 1 ▼ Auth ▼ Query Header 1 Docs

file frog.jpg ▼ ✓ 🗑

name value ▼ □ 🗑

New name New value

Preview ▼ Header 4 ▼ Cookie Timeline

```
1 {  
2   "class": "frog"  
3 }
```

POST ▼ http://45.252.182.94:8000/predict/ Send

200 OK TIME 844 ms SIZE 15 B

Multipart 1 ▼ Auth ▼ Query Header 1 Docs

file dog.jpg ▼ ✓ 🗑

name value ▼ □ 🗑

New name New value

Preview ▼ Header 4 ▼ Cookie Timeline

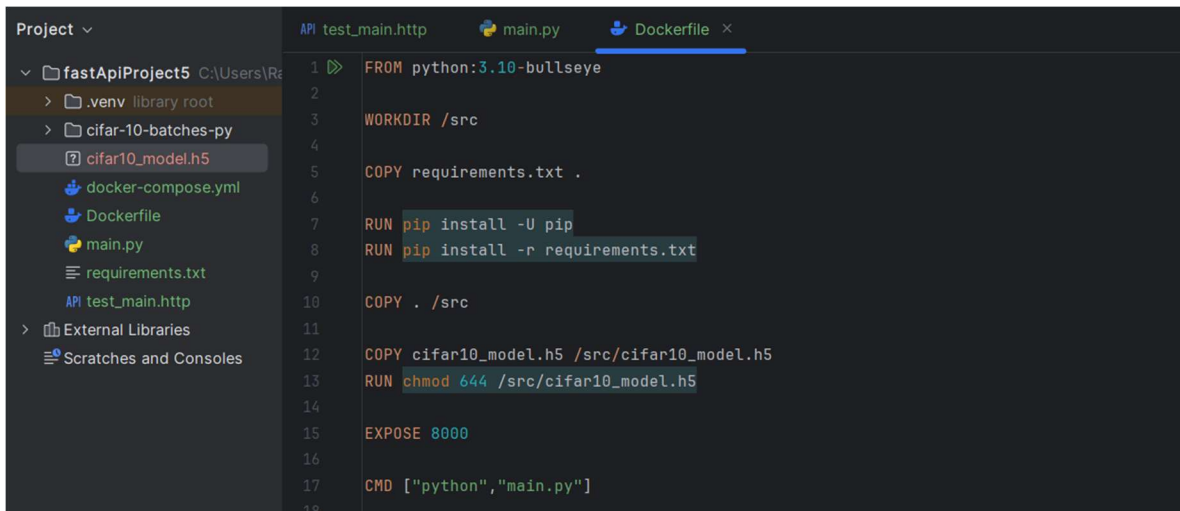
```
1 {  
2   "class": "dog"  
3 }
```

داکرایز

در فایل پروژه فایل های Dockerfile و docker-compose.yml را ایجاد میکنیم.

در فایل Dockerfile مسیری که پروژه باید طی کند را مینویسیم.

در فایل docker-compose.yml کانتینرها و image هایمان را مینویسیم.



```
1 FROM python:3.10-bullseye
2
3 WORKDIR /src
4
5 COPY requirements.txt .
6
7 RUN pip install -U pip
8 RUN pip install -r requirements.txt
9
10 COPY . /src
11
12 COPY cifar10_model.h5 /src/cifar10_model.h5
13 RUN chmod 644 /src/cifar10_model.h5
14
15 EXPOSE 8000
16
17 CMD ["python", "main.py"]
18
```

در خط 1 گفتیم که برنامه برای اجرا به پایتون و ورژن مورد نظر نیاز دارد.

در خط 3 مسیری را تعیین کردیم که فایل پروژه در آن قرار گیرد.

در خط 5 فایل requirements.txt که در آن کتابخانه های لازم قرار دارند، در همان مسیر کپی میشود.

در خط 7 کتابخانه pip که برای نصب کتابخانه های پایتونی مورد نیاز است نصب میشود.

در خط 8 کتابخانه های داخل فایل requirements.txt را نصب میشوند.

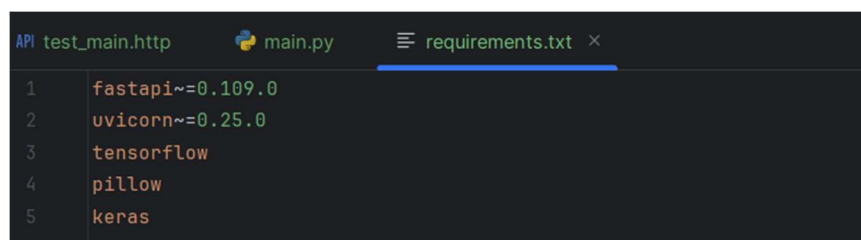
در خط 10 همه ی فایل هایی که در فایل پروژه وجود دارند، در آن مسیری که تعیین کردیم کپی میشوند.

در خطوط 12 و 13 فایل مدل را کپی و اجرا میکنیم.

در خط 15 پورتهی برنامه روی آن اجرا میشود گفته شده است.

در خط 17 دستوری که برنامه با آن اجرا میشود نوشته شده است.

فایل requirements.txt :



```
1 fastapi~=0.109.0
2 uvicorn~=0.25.0
3 tensorflow
4 pillow
5 keras
```

```

1 version: "3"
2
3 services:
4   app:
5     build: .
6     container_name: bigdataproject
7     command: uvicorn --host 0.0.0.0 --port 8000 main:app --reload
8     ports:
9       - "8000:8000"
10    networks:
11      - mainnetwork
12    restart: always
13
14 networks:
15   mainnetwork:
16     driver: bridge

```

در خط 4 image پروژه را نوشتیم که app است.

در app image در خط 5 گفتیم که در همان مسیر فعلی ساخته شود و بعد برای این container نامی تعیین کردیم. خط 7 برای طریقه اجرا شدن پروژه است که میگوید با استفاده از uvicorn از هر هاستی و پورت 7000 و از فایل main برنامه را اجرا کن. در خط 8 پورت 8000 سیستم را به پورت 8000 این app وصل میکند. در خطوط بعدی نام network مورد نظر را نوشتیم و در آخر در خط 12 تعیین کردیم که container در صورت به وجود آمدن مشکل خودش همیشه restart شود.

در نهایت در خط 14، network را تشکیل میدهیم.

برای ساخت image پروژه وارد پوشه پروژه میشویم و با دستور زیر image را میسازیم:

```

C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5>docker-compose up -d
[+] Building 117.1s (13/13) FINISHED
docker:default
=> [app internal] load .dockerignore 0.1s
=> => transferring context: 2B 0.1s
=> [app internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 318B 0.1s
=> [app internal] load metadata for docker.io/library/python:3.10-bullseye 0.0s
=> [app 1/8] FROM docker.io/library/python:3.10-bullseye 0.0s
=> [app internal] load build context 7.2s
=> => transferring context: 3.25MB 6.7s
=> CACHED [app 2/8] WORKDIR /src 0.0s
=> CACHED [app 3/8] COPY requirements.txt . 0.0s
=> CACHED [app 4/8] RUN pip install -U pip 0.0s
=> CACHED [app 5/8] RUN pip install -r requirements.txt 0.0s
=> [app 6/8] COPY . /src 86.4s
=> [app 7/8] COPY cifar10_model.h5 /src/cifar10_model.h5 0.3s
=> [app 8/8] RUN chmod 644 /src/cifar10_model.h5 1.2s
=> [app] exporting to image 21.3s
=> => exporting layers 21.2s
=> => writing image sha256:46cdb17ac7622aeb6c7018d2df43dcfa8867c7b130512c7dfe58abd960cafca7 0.0s
=> => naming to docker.io/library/fastapiproject5-app 0.0s
[+] Running 2/2
  Network fastapiproject5_mainnetwork Created 0.2s
  Container bigdataproject Started 0.5s

```

```
C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
77f26aac943c   fastapiproject5-app   "uvicorn --host 0.0..." 36 seconds ago Up 33 seconds 0.0.0.0:8000->8000/tcp   bigdataproject

C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5>docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
77f26aac943c   fastapiproject5-app   "uvicorn --host 0.0..." 46 seconds ago Up 43 seconds 0.0.0.0:8000->8000/tcp   bigdataproject
a144af408ac9   fastapiproject4-app   "uvicorn --host 0.0..." 31 minutes ago Exited (0) 11 minutes ago   parallelproject
2beb0bba70f3   fastapiproject2-app   "uvicorn --host 0.0..." 5 months ago   Exited (0) 25 minutes ago   fastprojectmongo
2c83c7ceae21   mongo:6.0.13         "docker-entrypoint.s..." 5 months ago   Exited (0) 25 minutes ago   fastmongo
d301dc3429ad   hello-world         "/hello"                  5 months ago   Exited (0) 5 months ago     eloquent_austin

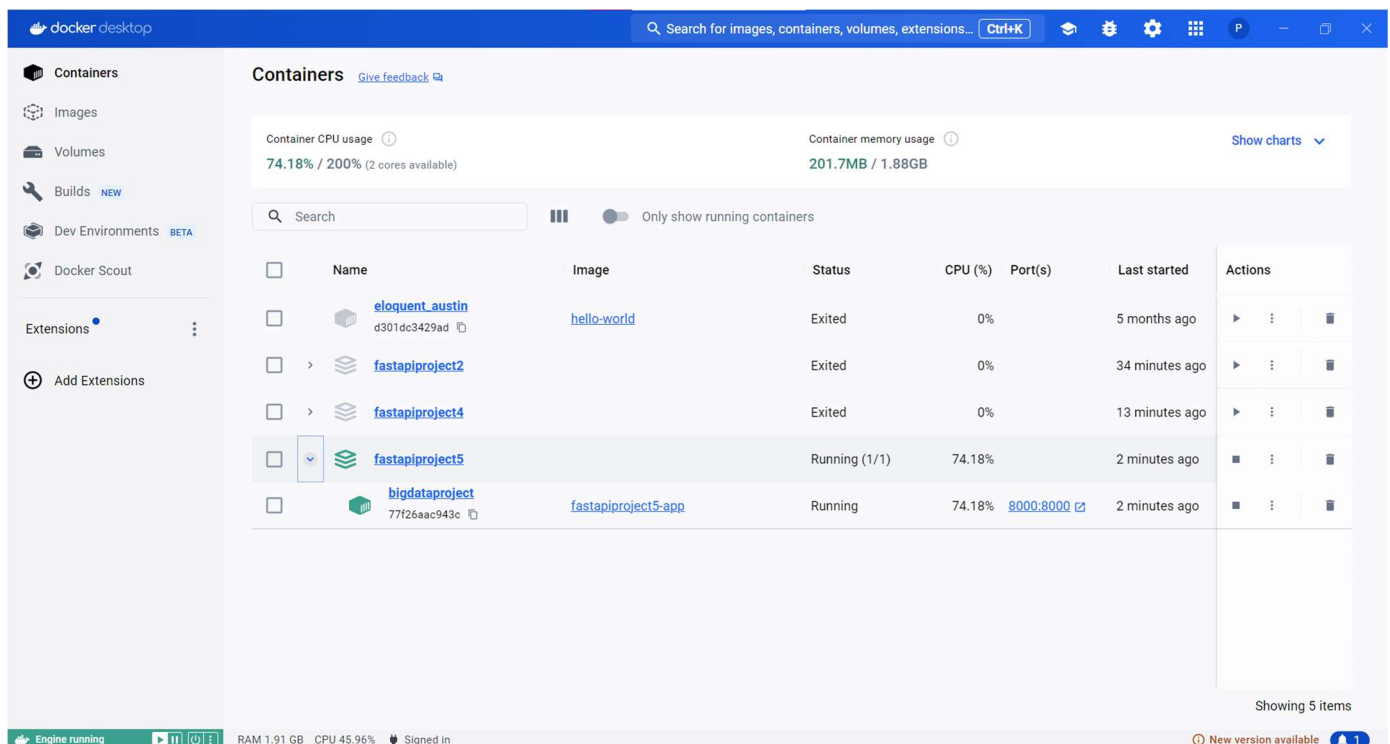
C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5>docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
fastapiproject5-app   latest    ec0c2b7bb120   About a minute ago   5.47GB
fastapiproject4-app   latest    064c25f13628   31 minutes ago      992MB
fastapiproject2-app   latest    19cb94ac15ab   5 months ago        999MB
fastapiproject3-app   latest    941ba8fa3200   5 months ago        1.02GB
mongo              6.0.13    23e54c868737   5 months ago        690MB
postgres          16.1      75b7bff7c3ad   6 months ago        425MB
python             3.10-bullseye 2efb40c32a8e   6 months ago        911MB
hello-world        latest    d2c94e258dcb   14 months ago       13.3kB
```

دستور `docker ps` لیست container های فعال را به ما نشان میدهد که همانطور که مشاهده میکنید `contanner` که ما ساختیم هم با نام های `fastapipeobject5-app` فعال است.

دستور `docker ps -a` لیست همه ی container ها چه فعال و چه غیر فعال را نشان میدهد.

و در آخر با دستور `docker images` لیست image ها را میبینیم که `fastapipeobject5-app` هم که مربوط به پروژه ما هست اضافه شده است.

همانطور که مشاهده میکنید در Docker هم container ساخته شده را میتوانیم ببینیم:



داکر در سرور

Recycle Bin

Microsoft Edge

Docker Desktop

BigDataPro...

ParallelPro...

New Text Document

docker desktop

Search for images, containers, volumes... Ctrl+K

Containers

Give feedback

Container CPU usage 3.23% / 400% (4 CPUs available)

Container memory usage 259.9MB / 2.64GB

Show charts

Search

Only show running containers

	Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	bigdatap		Running (1/1)		3.23%	11 minutes ago	<input type="checkbox"/> ⋮ <input type="checkbox"/>
<input type="checkbox"/>	bigdat bb590f	bigdatapproject-n	Running	8000:8000	3.23%	11 minutes ago	<input type="checkbox"/> ⋮ <input type="checkbox"/>
<input type="checkbox"/>	parallelp		Exited		0%	43 minutes ago	<input type="checkbox"/> ⋮ <input type="checkbox"/>

Showing 3 items

Walkthroughs

Multi-container applications

8 mins

Containerize your application

3 mins

View more in the Learning center

Engine running

RAM 2.69 GB CPU 2.79%

v4.31.1

Search

5:40 AM 7/10/2024

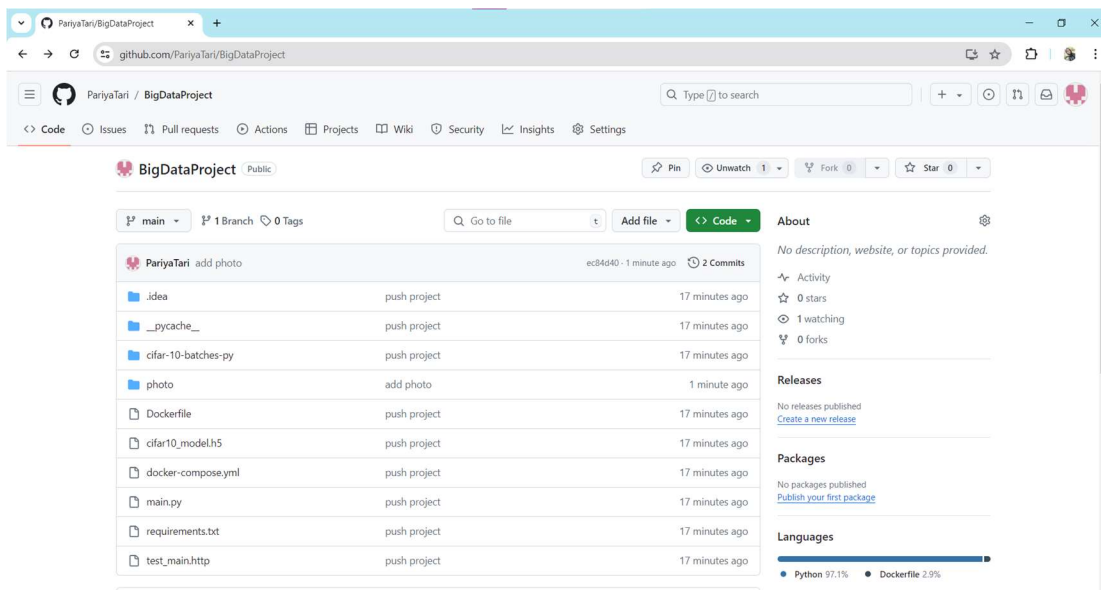
Github

```
INFO: Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5> git init
Reinitialized existing Git repository in C:/Users/RatinRayaneh/PycharmProjects/fastApiProject5/.git/
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5> git remote add origin git@github.com:PariyaTari/BigDataProject.git
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5> git branch -M main
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5> git add .
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5> git commit -m "push project"
[main (root-commit) 8ea523f] push project
 21 files changed, 358 insertions(+)
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/fastApiProject5.iml
 create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/modules.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 Dockerfile
 create mode 100644 __pycache__/main.cpython-38.pyc
 create mode 100644 cifar-10-batches-py/batches.meta
 create mode 100644 cifar-10-batches-py/data_batch_1
 create mode 100644 cifar-10-batches-py/data_batch_2
 create mode 100644 cifar-10-batches-py/data_batch_3
 create mode 100644 cifar-10-batches-py/data_batch_4
 create mode 100644 cifar-10-batches-py/data_batch_5
 create mode 100644 cifar-10-batches-py/readme.html
 create mode 100644 cifar10_model.h5
 create mode 100644 docker-compose.yml
 create mode 100644 main.py
 create mode 100644 requirements.txt
 create mode 100644 test_main.http
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5> git push -u origin main
```

```
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5> git push -u origin main
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 8 threads
Compressing objects: 100% (25/25), done.
Writing objects: 100% (27/27), 164.00 MiB | 1.66 MiB/s, done.
Total 27 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:PariyaTari/BigDataProject.git
 * [new branch]    main -> main
branch 'main' set up to track 'origin/main'.
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject5>
```

عکس مراحل پوش کردن پروژه به repository در گیت‌هاب را نشان میدهد.

ثبت پروژه در github :



پایان