



دانشگاه لرستان

به نام خدا

تمرین درس داده های حجیم

دانشجو:

حانیه بیگی

شماره دانشجویی:

40211415015

استاد مربوطه:

دکتر عبدالرضا رشنو

تیرماه 1403

```

import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import uvicorn
from fastapi import FastAPI, File, UploadFile
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
import io
import os
from PIL import Image
from fastapi import APIRouter
import sys
import io

sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

router = APIRouter()

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

def process_image(img):
    img = Image.open(io.BytesIO(img))
    img = img.resize((32, 32))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0
    return img_array

@router.post("/predictc/")
async def predict(file: UploadFile = File(...)):
    contents = await file.read()
    img_array = process_image(contents)
    predictions = model.predict(img_array)
    score = np.argmax(predictions[0])
    predicted_label = class_names[score]
    return {"label": predicted_label}

def load_and_predict_model():
    model = load_model('cifar10_model.keras')
    return model

if not os.path.exists('cifar10_model.keras'):
    train_and_save_model()

model = load_and_predict_model()

def train_and_save_model():
    (train_images, train_labels), (test_images, test_labels) =
datasets.cifar10.load_data()
    train_images, test_images = train_images / 255.0, test_images / 255.0

    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32,
32, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))

```

```

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
validation_data=(test_images, test_labels))

model.save('cifar10_model.keras')

```

وارد کردن کتابخانه‌ها و تنظیمات اولیه

```

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import uvicorn

from fastapi import FastAPI, File, UploadFile

from tensorflow.keras.models import load_model

from tensorflow.keras.preprocessing import image

import numpy as np

import io

import os

from PIL import Image

from fastapi import APIRouter

import sys

import io

sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

```

tensorflow و tensorflow.keras برای ساخت و آموزش مدل‌های شبکه عصبی استفاده می‌شوند.

uvicorn یک سرور ASGI است که برای اجرای برنامه‌های FastAPI استفاده می‌شود.

FastAPI, File, UploadFile و APIRouter از کتابخانه FastAPI برای ساخت API‌های وب استفاده می‌شوند.

load_model و image از tensorflow.keras برای بارگذاری مدل آموزش‌دیده و پردازش تصاویر استفاده می‌شوند.

numpy برای انجام عملیات عددی استفاده می‌شود.

io و os برای عملیات فایل استفاده می‌شوند.

PIL (Python Imaging Library) برای پردازش تصاویر استفاده می‌شود.

```
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
```

اطمینان حاصل می‌کند که خروجی استاندارد با کدگذاری UTF-8 تنظیم شده است.

نام کلاس‌ها و تابع پردازش تصویر

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',  
'dog', 'frog', 'horse', 'ship', 'truck']
```

```
def process_image(img):
```

```
    img = Image.open(io.BytesIO(img))
```

```
    img = img.resize((32, 32))
```

```
    img_array = image.img_to_array(img)
```

```
    img_array = np.expand_dims(img_array, axis=0)
```

```
img_array /= 255.0  
return img_array
```

`class_names` لیست برچسب‌های کلاس‌ها برای مجموعه داده **CIFAR-10** را تعریف می‌کند.

تابع `process_image`:

تصویر آپلود شده را از بایت‌ها باز می‌کند.

تصویر را به اندازه **32x32** پیکسل تغییر اندازه می‌دهد.

تصویر را به آرایه تبدیل می‌کند.

ابعاد را برای تطابق با شکل ورودی مدل گسترش می‌دهد.

مقادیر پیکسل‌ها را بین **0** و **1** نرمال‌سازی می‌کند.

مسیر **FastAPI** برای پیش‌بینی

```
@router.post("/predictc/")
```

```
async def predict(file: UploadFile = File(...)):
```

```
    contents = await file.read()
```

```
    img_array = process_image(contents)
```

```
    predictions = model.predict(img_array)
```

```
    score = np.argmax(predictions[0])
```

```
    predicted_label = class_names[score]
```

```
    return {"label": predicted_label}
```

`router.post("/predictc/")` یک نقطه پایانی **POST** را تعریف می‌کند.

تابع `predict`:

فایل تصویری آپلود شده را می‌خواند.

تصویر را با استفاده از `process_image` پردازش می‌کند.

با استفاده از مدل آموزش‌دیده پیش‌بینی می‌کند.

کلاسی که بالاترین امتیاز را دارد پیدا می‌کند.

برچسب پیش‌بینی شده را به عنوان پاسخ JSON بازمی‌گرداند.

بارگذاری و آموزش مدل

```
def load_and_predict_model():
```

```
    model = load_model('cifar10_model.keras')
```

```
    return model
```

```
if not os.path.exists('cifar10_model.keras'):
```

```
    train_and_save_model()
```

```
model = load_and_predict_model()
```

```
def train_and_save_model():
```

```
    (train_images, train_labels), (test_images, test_labels) =  
    datasets.cifar10.load_data()
```

```
    train_images, test_images = train_images / 255.0, test_images /  
    255.0
```

```
    model = models.Sequential()
```

```

model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),

metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
validation_data=(test_images, test_labels))

model.save('cifar10_model.keras')

```

load_and_predict_model مدل را از یک فایل بارگذاری می‌کند. بررسی می‌کند که آیا فایل مدل موجود است یا خیر؛ اگر موجود نیست، با استفاده از **train_and_save_model** مدل را آموزش داده و ذخیره می‌کند.

تابع `train_and_save_model`:

مجموعه داده CIFAR-10 را بارگذاری می‌کند.

داده‌های تصویری را نرمال‌سازی می‌کند.

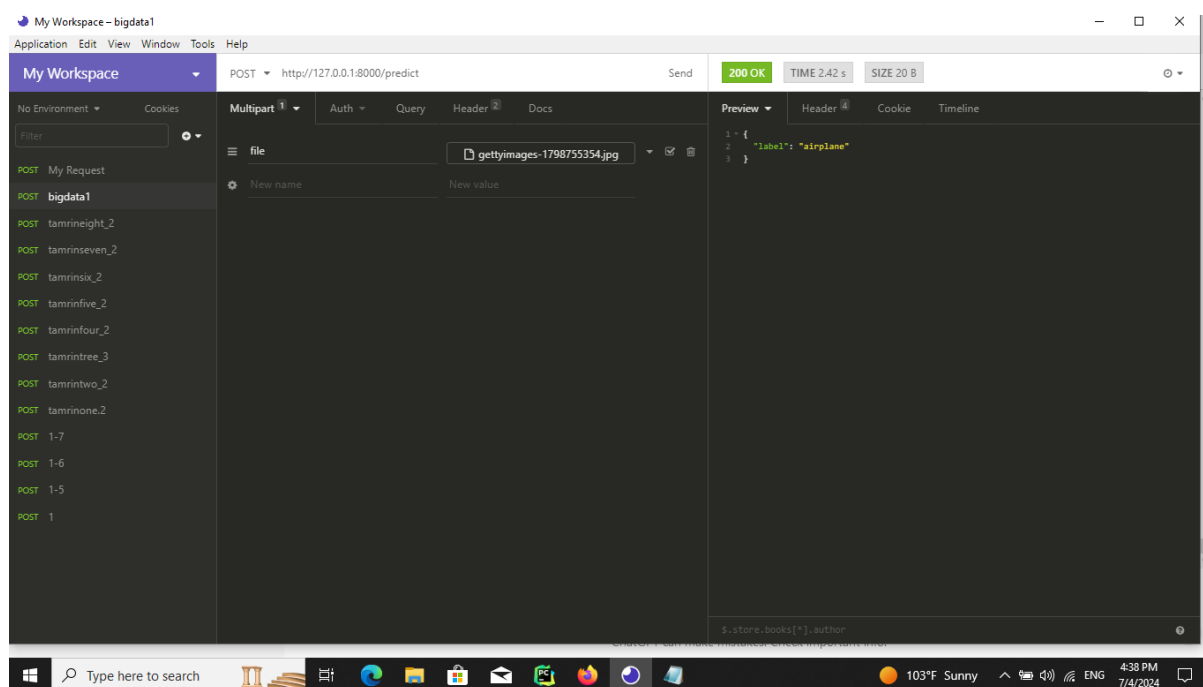
یک مدل شبکه عصبی کانولوشنی (CNN) را تعریف می‌کند.

مدل را با استفاده از بهینه‌ساز Adam و تابع هزینه cross-entropy کامپایل می‌کند.

مدل را برای 10 دوره آموزشی تمرین می‌دهد.

مدل آموزش‌دیده را در یک فایل ذخیره می‌کند.

خروجی



تمرین دوم

```
from fastapi import APIRouter, FastAPI, File, UploadFile, Response
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
import tensorflow_datasets as tfds
import io
from PIL import Image
import logging
import os
```



```

import sys

# کدگذاری از پشتیبانی برای خروجی تنظیمات UTF-8
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

router = APIRouter()

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[logging.StreamHandler(sys.stdout)]
)
logger = logging.getLogger(__name__)

# داده‌ها و مدل تنظیمات
NUM_CLASSES = 4
INPUT_HEIGHT = 128
INPUT_WIDTH = 128
LEARNING_RATE = 1e-3
EPOCHS = 10
BATCH_SIZE = 16
SHUFFLE = True
MODEL_WEIGHTS_FILE = 'fcn32s_model.weights.h5' # وزن‌ها فایل نام تغییر
AUTOTUNE = tf.data.AUTOTUNE

(train_ds, valid_ds, test_ds) = tfds.load(
    "oxford_iiit_pet",
    split=["train[:85%]", "train[85%:]", "test"],
    batch_size=BATCH_SIZE,
    shuffle_files=SHUFFLE,
)

def unpack_resize_data(section):
    image = section["image"]
    segmentation_mask = section["segmentation_mask"]
    resize_layer = keras.layers.Resizing(INPUT_HEIGHT, INPUT_WIDTH)
    image = resize_layer(image)
    segmentation_mask = resize_layer(segmentation_mask)
    return image, segmentation_mask

train_ds = train_ds.map(unpack_resize_data, num_parallel_calls=AUTOTUNE)
valid_ds = valid_ds.map(unpack_resize_data, num_parallel_calls=AUTOTUNE)
test_ds = test_ds.map(unpack_resize_data, num_parallel_calls=AUTOTUNE)

input_layer = keras.Input(shape=(INPUT_HEIGHT, INPUT_WIDTH, 3))
vgg_model = keras.applications.VGG19(include_top=False, weights="imagenet",
input_shape=(INPUT_HEIGHT, INPUT_WIDTH, 3))

fcn_backbone = keras.models.Model(
    inputs=vgg_model.input,
    outputs=[
        vgg_model.get_layer(block_name).output
        for block_name in ["block3_pool", "block4_pool", "block5_pool"]
    ],
)

fcn_backbone.trainable = False

x = fcn_backbone(input_layer)
units = [4096, 4096]

```

```

dense_convs = []

for filter_idx in range(len(units)):
    dense_conv = keras.layers.Conv2D(
        filters=units[filter_idx],
        kernel_size=(7, 7) if filter_idx == 0 else (1, 1),
        strides=(1, 1),
        activation="relu",
        padding="same",
        use_bias=False,
        kernel_initializer='he_normal',
    )
    dense_convs.append(dense_conv)
    dropout_layer = keras.layers.Dropout(0.5)
    dense_convs.append(dropout_layer)

dense_convs = keras.Sequential(dense_convs)
dense_convs.trainable = False

x = dense_convs(x[-1])

pool5 = keras.layers.Conv2D(
    filters=NUM_CLASSES,
    kernel_size=(1, 1),
    padding="same",
    strides=(1, 1),
    activation="relu",
) (x)

fcn32s_conv_layer = keras.layers.Conv2D(
    filters=NUM_CLASSES,
    kernel_size=(1, 1),
    activation="softmax",
    padding="same",
    strides=(1, 1),
)

fcn32s_upsampling = keras.layers.UpSampling2D(
    size=(32, 32),
    data_format=keras.backend.image_data_format(),
    interpolation="bilinear",
)

final_fcn32s_pool = pool5
final_fcn32s_output = fcn32s_conv_layer(final_fcn32s_pool)
final_fcn32s_output = fcn32s_upsampling(final_fcn32s_output)

fcn32s_model = keras.Model(inputs=input_layer, outputs=final_fcn32s_output)

fcn32s_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=LEARNING_RATE),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)

if os.path.exists(MODEL_WEIGHTS_FILE):
    fcn32s_model.load_weights(MODEL_WEIGHTS_FILE)
    logger.info(f"Model weights loaded from {MODEL_WEIGHTS_FILE}")
else:
    fcn32s_model.fit(
        train_ds,
    
```

```

        validation_data=valid_ds,
        epochs=EPOCHS,
    )
    fcn32s_model.save_weights(MODEL_WEIGHTS_FILE)
    logger.info(f"Model weights saved to {MODEL_WEIGHTS_FILE}")

test_loss, test_accuracy = fcn32s_model.evaluate(test_ds)
logger.info(f"Test Loss: {test_loss}")
logger.info(f"Test Accuracy: {test_accuracy}")

@router.post("/predict1/")
async def predict(file: UploadFile = File(...)):
    image = Image.open(file.file).convert("RGB")
    image = image.resize((INPUT_WIDTH, INPUT_HEIGHT))
    image = np.array(image)
    image = image / 255.0
    image = np.expand_dims(image, axis=0)

    prediction = fcn32s_model.predict(image) [0]
    segmentation_mask = np.argmax(prediction, axis=-1)

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title("Input Image")
    plt.imshow(image[0])
    plt.subplot(1, 2, 2)
    plt.title("Segmentation Mask")

```

وارد کردن کتابخانه‌ها و تنظیمات اولیه

```

from fastapi import APIRouter, FastAPI, File, UploadFile, Response

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

import tensorflow_datasets as tfds

import io

from PIL import Image

import logging

import os

import sys

```

تنظیم خروجی برای پشتیبانی از کدگذاری UTF-8

```
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
```

```
router = APIRouter()
```

تنظیمات برای نمایش ورودی‌ها و خروجی‌ها در کنسول

```
logging.basicConfig()
```

```
level=logging.INFO,
```

```
format='%(asctime)s - %(levelname)s - %(message)s,'
```

```
handlers=[logging.StreamHandler(sys.stdout)]
```

```
(
```

```
logger = logging.getLogger(__name__)
```

در این بخش، کتابخانه‌های مورد نیاز برای اجرای برنامه وارد می‌شوند. همچنین تنظیمات برای پشتیبانی از کدگذاری UTF-8 و برای نمایش پیام‌های log در کنسول انجام می‌شود.

تنظیمات مدل و داده‌ها

```
NUM_CLASSES = 4
```

```
INPUT_HEIGHT = 128
INPUT_WIDTH = 128
LEARNING_RATE = 1e-3
EPOCHS = 10
BATCH_SIZE = 16
SHUFFLE = True
MODEL_WEIGHTS_FILE = 'fcn32s_model.weights.h5' # تغییر نام فایل
# وزن‌ها
```

```
AUTOTUNE = tf.data.AUTOTUNE
```

```
(train_ds, valid_ds, test_ds) = tfds.load
```

```
" oxford_iiit_pet,"
```

```
split=["train[:85%]", "train[85%:]", "test"],
```

```
batch_size=BATCH_SIZE,
```

```
shuffle_files=SHUFFLE,
```

(

در این بخش، پارامترهای مدل مانند تعداد کلاس‌ها، ابعاد تصویر ورودی، نرخ یادگیری، تعداد اپاک‌ها، اندازه دسته و فایل وزن‌های مدل تعریف می‌شوند. همچنین مجموعه داده "oxford_iiit_pet" برای آموزش، اعتبارسنجی و آزمایش بارگذاری می‌شود.

```
def unpack_resize_data(section):  
    image = section["image"]  
    segmentation_mask = section["segmentation_mask"]  
    resize_layer = keras.layers.Resizing(INPUT_HEIGHT,  
INPUT_WIDTH)  
    image = resize_layer(image)  
    segmentation_mask = resize_layer(segmentation_mask)  
    return image, segmentation_mask
```

```
train_ds = train_ds.map(unpack_resize_data,  
num_parallel_calls=AUTOTUNE)  
valid_ds = valid_ds.map(unpack_resize_data,  
num_parallel_calls=AUTOTUNE)  
test_ds = test_ds.map(unpack_resize_data,  
num_parallel_calls=AUTOTUNE)
```

تابع `unpack_resize_data` تصاویر و ماسک‌های `segmentation` را به ابعاد مورد نظر تغییر اندازه می‌دهد و سپس داده‌های آموزشی، اعتبارسنجی و آزمایش به کمک آن تغییر اندازه می‌یابند.

```
input_layer = keras.Input(shape=(INPUT_HEIGHT, INPUT_WIDTH,
3))
```

```
vgg_model = keras.applications.VGG19(include_top=False,
weights="imagenet", input_shape=(INPUT_HEIGHT,
INPUT_WIDTH, 3))
```

```
fcn_backbone = keras.models.Model()
```

```
inputs=vgg_model.input,
```

```
outputs]=
```

```
    vgg_model.get_layer(block_name).output
```

```
    for block_name in ["block3_pool", "block4_pool", "block5_pool"]
```

```
.,[
```

```
(
```

```
fcn_backbone.trainable = False
```

```
x = fcn_backbone(input_layer)
```

```
units[4096, 4096] =
```

```
dense_convs[] =
```

```
for filter_idx in range(len(units)):
```

```
    dense_conv = keras.layers.Conv2D()
```

```
    filters=units[filter_idx],
```

```

        kernel_size=(7, 7) if filter_idx == 0 else (1, 1)
        strides=(1, 1)
        activation="relu,"
        padding="same,"
        use_bias=False,
        kernel_initializer='he_normal,'
    )
    dense_convs.append(dense_conv)
    dropout_layer = keras.layers.Dropout(0.5)
    dense_convs.append(dropout_layer)

dense_convs = keras.Sequential(dense_convs)
dense_convs.trainable = False

x = dense_convs(x[-1])

pool5 = keras.layers.Conv2D(
    filters=NUM_CLASSES,
    kernel_size=(1, 1),
    padding="same,"
    strides=(1, 1),
    activation="relu,"

```


)X(

```
fcn32s_conv_layer = keras.layers.Conv2D)
```

```
    filters=NUM_CLASSES,
```

```
    kernel_size=(1 ,1)=
```

```
    activation="softmax,"
```

```
    padding="same,"
```

```
    strides=(1 ,1)=
```

```
(
```

```
fcn32s_upsampling = keras.layers.UpSampling2D)
```

```
    size=(32 ,32)=
```

```
    data_format=keras.backend.image_data_format(),
```

```
    interpolation="bilinear,"
```

```
(
```

```
final_fcn32s_pool = pool5
```

```
final_fcn32s_output = fcn32s_conv_layer(final_fcn32s_pool)
```

```
final_fcn32s_output = fcn32s_upsampling(final_fcn32s_output)
```

```
fcn32s_model = keras.Model(inputs=input_layer,
```

```
outputs=final_fcn32s_output)
```

در این بخش، مدل شبکه عصبی با استفاده از شبکه VGG19 به عنوان بخش اصلی و لایه‌های segmentation و Conv2D و UpSampling2D برای تبدیل خروجی‌های VGG19 به ماسک ساخته می‌شود.

آموزش یا بارگذاری مدل

```
fc32s_model.compile)

optimizer=keras.optimizers.Adam(learning_rate=LEARNING_RATE)
,
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
(

if os.path.exists(MODEL_WEIGHTS_FILE):
    fc32s_model.load_weights(MODEL_WEIGHTS_FILE)
    logger.info(f"Model weights loaded from
{MODEL_WEIGHTS_FILE}")
else:
    fc32s_model.fit)
    train_ds,
    validation_data=valid_ds,
    epochs=EPOCHS,
(
    fc32s_model.save_weights(MODEL_WEIGHTS_FILE)
```

```
logger.info(f"Model weights saved to {MODEL_WEIGHTS_FILE}")
```

در این بخش، مدل تعریف شده با داده‌های آموزشی و اعتبارسنجی آموزش داده می‌شود و وزن‌های آن ذخیره می‌شوند یا اگر از قبل وجود داشته باشند، بارگذاری می‌شوند.

پیش‌بینی با استفاده از مدل

```
@router.post("/predict1/")
```

```
async def predict(file: UploadFile = File(...)):
```

```
    image = Image.open(file.file).convert("RGB")
```

```
    image = image.resize((INPUT_WIDTH, INPUT_HEIGHT))
```

```
    image = np.array(image)
```

```
    image = image / 255.0
```

```
    image = np.expand_dims(image, axis=0)
```

```
    prediction = fcn32s_model.predict(image)[0]
```

```
    segmentation_mask = np.argmax(prediction, axis=-1)
```

```
    plt.figure(figsize=(10, 5))
```

```
    plt.subplot(1, 2, 1)
```

```
    plt.title("Input Image")
```

```
    plt.imshow(image[0])
```

```
    plt.subplot(2, 2, 1)
```

```
    plt.title("Segmentation Mask")
```

```
plt.imshow(segmentation_mask, cmap="inferno")

buf = io.BytesIO()

plt.savefig(buf, format='png')

plt.close()

buf.seek(0)

return Response(content=buf.getvalue(),
media_type="image/png")
```

در این بخش، تصاویر ورودی از طریق وب سرویس با استفاده از FastAPI پیش‌بینی می‌شوند و ماسک segmentation برای هر تصویر با استفاده از مدل fcn32s_model تولید می‌شود. سپس تصویر ورودی و ماسک segmentation در کنار هم نمایش داده می‌شود و به عنوان پاسخ به درخواست، تصویر نهایی به کلاینت ارسال می‌شود.

```
()app = FastAPI

app.include_router(router)

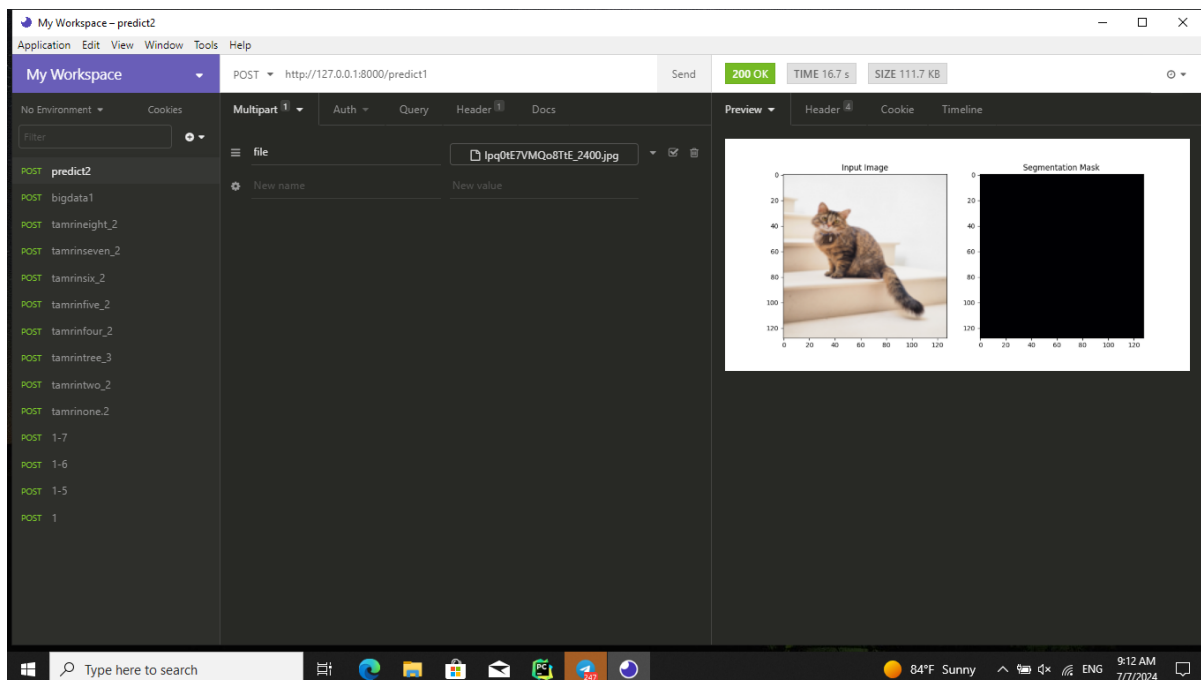
:"__if __name__ == "__main

import uvicorn

uvicorn.run(app, host="0.0.0.0", port=8000)
```

در این بخش، یک نمونه از FastAPI ایجاد شده و روتر که شامل نقطه پایانی `/predict1` برای پیش‌بینی تصاویر است، به آن اضافه می‌شود. سپس اپلیکیشن با استفاده از `uvicorn` اجرا می‌شود.

خروجی



نحوه داکرایز کردن

درابتدا با استفاده از دستورات

`Docker pull python:3.10-bullseye`

در اینجا، دستور `pull docker` برای دریافت تصویر `Python` با نسخه `3.10` و سیستم عامل `Bullseye Debian` از `Hub Docker` به کار می‌رود.

این دستور باعث دریافت تصویر `python:3.10-bullseye` از `Hub Docker` میشود و آن را در محیط محلی `Docker` ذخیره می‌کند.

در قسمت بعد در محیط پای چرم در کنار فایل‌هایی که برای تمرین ایجاد کردیم برای داکرایز کردن

ابتدا دوفایل `dockerfile`, `dockercompose` را ایجاد میکنیم

داکر فایل: درواقع هنگامی که می‌خواهیم داکرایز کنیم مسیر اصلی برنامه را به ما نشان می‌دهد و چه اقداماتی انجام بده رو به ما نشان می‌دهد

داکر فایل

. انتخاب تصویر پایه:

dockerfile

FROM python:3.10-bullseye

در این خط، شما تصویری از Python نسخه 3.10 که بر اساس توزیع Debian Bullseye ساخته شده است را به عنوان تصویر پایه (Base Image) انتخاب می‌کنید. این تصویر شامل تمام پیش‌نیازهای Python و کتابخانه‌های سیستم‌عامل Debian Bullseye است.

2. تنظیم دایرکتوری کاری:

dockerfile

WORKDIR /code

این خط دایرکتوری کاری را در داخل کانتینر به /code تنظیم می‌کند. تمامی دستورات بعدی در این دایرکتوری اجرا خواهند شد. به عبارتی، شما تعیین می‌کنید که محل قرارگیری کدها و فایل‌ها در داخل کانتینر کجا باشد.

3. کپی فایل requirements.txt:

dockerfile

COPY requirements.txt.

این خط فایل `requirements.txt` را از دایرکتوری محلی (جایی که `Dockerfile` قرار دارد) به دایرکتوری کاری (`code/`) در داخل کانتینر کپی می‌کند. این فایل شامل لیست کتابخانه‌های `Python` است که پروژه شما به آن‌ها نیاز دارد.

4. نصب پیش‌نیازهای پروژه:

`dockerfile`

`RUN pip install -r requirements.txt`

این خط با استفاده از ابزار `pip`، کتابخانه‌های مورد نیاز پروژه را که در فایل `requirements.txt` مشخص شده‌اند، نصب می‌کند. دستور `RUN` در `Dockerfile` برای اجرای دستورات در زمان ساخت تصویر `Docker` استفاده می‌شود.

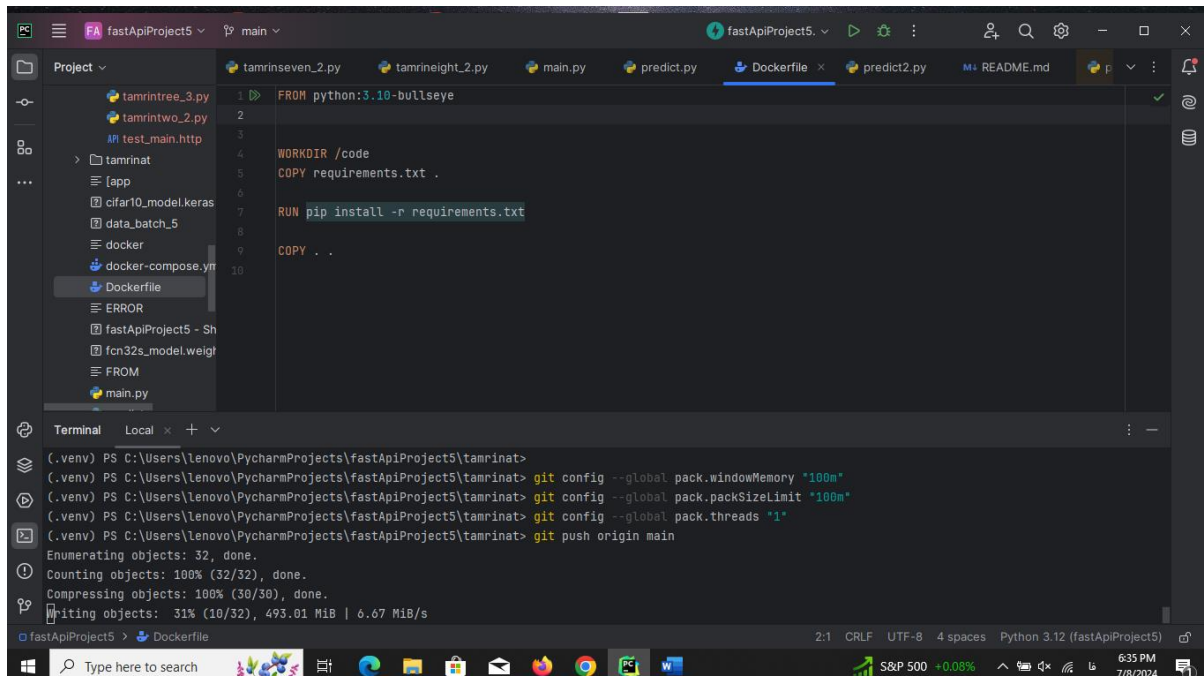
5. کپی کل پروژه به دایرکتوری کاری:

`dockerfile`

`COPY .`

این خط تمامی فایل‌ها و دایرکتوری‌های موجود در دایرکتوری محلی (جایی که `Dockerfile` قرار دارد) را به دایرکتوری کاری (`code/`) در داخل کانتینر کپی می‌کند. به این ترتیب، کل کدهای پروژه شما به داخل کانتینر منتقل می‌شود.

به طور کلی، این Dockerfile یک محیط Python را با استفاده از یک تصویر پایه Python 3.10 بر اساس Debian Bullseye تنظیم می‌کند، پیش‌نیازهای پروژه را نصب می‌کند و سپس کل پروژه را به داخل کانتینر کپی می‌کند تا آماده اجرا باشد.



The screenshot shows the PyCharm IDE interface. The top pane displays a Dockerfile with the following content:

```
FROM python:3.10-bullseye
WORKDIR /code
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
```

The bottom pane shows a terminal window with the following commands and output:

```
(.venv) PS C:\Users\lenovo\PycharmProjects\fastApiProject5\tamrinat>
(.venv) PS C:\Users\lenovo\PycharmProjects\fastApiProject5\tamrinat> git config --global pack.windowMemory "100m"
(.venv) PS C:\Users\lenovo\PycharmProjects\fastApiProject5\tamrinat> git config --global pack.packSizeLimit "100m"
(.venv) PS C:\Users\lenovo\PycharmProjects\fastApiProject5\tamrinat> git config --global pack.threads "1"
(.venv) PS C:\Users\lenovo\PycharmProjects\fastApiProject5\tamrinat> git push origin main
Enumerating objects: 32, done.
Counting objects: 100% (32/32), done.
Compressing objects: 100% (30/30), done.
Writing objects: 31% (10/32), 493.01 MiB | 6.67 MiB/s
```

```
FROM python:3.10-bullseye

WORKDIR /code
COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .
```

و داکر کامپوز

```
version: "3"

services:
  app:
    build: .
    container_name: fastProject5
    command: uvicorn --host 0.0.0.0 --port 8000 main:app --reload
    ports:
      - "8080:8000"

    networks:
      - fastnetwork
    restart: always

networks:
```



```
fastnetwork:  
  driver: bridge
```

. نسخه Docker Compose:

version: "3"

در این خط، نسخه‌ای از Docker Compose که می‌خواهید استفاده کنید را مشخص می‌کنید. در اینجا از نسخه ۳ استفاده شده است.

2. تعریف سرویس‌ها:

services:

در این بخش، سرویس‌هایی که قرار است در Docker Compose استفاده شوند را تعریف می‌کنید.

3. تعریف سرویس app:

app:

build.:

container_name: fastProject5

command: uvicorn --host 0.0.0.0 --port 8000 main:app --reload

ports:

"8080:8000" –

networks:

– fastnetwork

restart: always

جزئیات سرویس app:

build. :

این خط مشخص می‌کند که Dockerfile موجود در دایرکتوری فعلی برای ساخت تصویر Docker استفاده شود.

container_name: fastProject5

این خط نامی برای کانتینر ایجاد شده مشخص می‌کند. در اینجا نام کانتینر fastProject5 است.

command: uvicorn --host 0.0.0.0 --port 8000 main

-- reload

این خط فرمانی است که در داخل کانتینر برای اجرای اپلیکیشن شما اجرا می‌شود. در اینجا، uvicorn به عنوان سرور وب استفاده می‌شود تا اپلیکیشن FastAPI شما را اجرا کند.

ports:

این بخش پورت‌های مورد نیاز را تعریف می‌کند.

"8080:8000"

این خط مشخص می‌کند که پورت ۸۰۸۰ بر روی میزبان به پورت ۸۰۰۰ در داخل کانتینر نگاشت شود. به این ترتیب، شما می‌توانید از طریق پورت ۸۰۸۰ به اپلیکیشن دسترسی پیدا کنید.

`:networks`

این بخش شبکه‌هایی که سرویس به آن متصل است را تعریف می‌کند.

`fastnetwork`

این خط مشخص می‌کند که سرویس `app` به شبکه `fastnetwork` متصل شود.

`restart: always`

این خط تنظیم می‌کند که کانتینر همیشه در صورت خرابی مجدداً راه‌اندازی شود.

4. تعریف شبکه‌ها:

`networks:`

`fastnetwork:`

`driver: bridge`

جزئیات شبکه `fastnetwork`:

`:fastnetwork`

این خط یک شبکه به نام `fastnetwork` تعریف می‌کند.

`driver: bridge`

این خط مشخص می‌کند که شبکه از نوع **bridge** باشد، که یک نوع شبکه پیش‌فرض در Docker است و امکان ارتباط بین کانتینرهای مختلف را فراهم می‌کند.

این فایل به طور کلی یک سرویس **FastAPI** را پیکربندی می‌کند که بر روی پورت ۸۰۸۰ قابل دسترسی است و در صورت خرابی به طور خودکار مجدداً راه‌اندازی می‌شود و از یک شبکه‌ی **bridge** به نام **fastnetwork** استفاده می‌کند

وبعد بعد از این که فایل‌ها را ایجاد کردیم به قسمتی که در آن پروژه‌های پای‌چرم ذخیره می‌شوند می‌رویم و فایل پروژه را در درایو دلخواه مان که من درایو C را انتخاب کردم وارد می‌شویم و فایل را در آن قسمت کپی کرده و بعد رو فایل داکر کامپوز کلیک را ست می‌کنیم یک فایل تکست از آن ایجاد می‌کنیم و نام آن را **requirements** می‌گذاریم و در آن پکیج‌هایی که موردنیاز می‌باشد را می‌نویسیم و دونه به دونه در هنگام ران کردن اگر به این پکیج‌ها نیاز باشد اون‌ها رو نصب می‌کنه.

وبعد وارد محیط **cmd** می‌شویم

حال می‌خواهیم داکرایز کنیم با دستور

docker compose up -d الان شروع می‌کنه به داکرایز کردن و ایمج را می‌سازد و داخل کانتینر قرار می‌دهد ایمج‌ها ساخته شدن و درون کانتینر‌ها قرار داده شده‌ان و باید به ش‌بک‌ه وصل کند

و با دستور **docker images** مشاهده می‌کنیم چه ایمج‌هایی ساخته شدن حال انتظار داریم که برنامه ران شود و آن‌رو روی ایمیسونیا مشاهده کنیم

```
Command Prompt - docker-compose up
Untagged: fastapiproject5-app:latest
Deleted: sha256:deca08f005c151b094cc8662172619263849d7fa9203b5917789ffe6a977695b

C:\Users\lenovo\PycharmProjects\fastApiProject5>docker-compose up
[+] Building 687.2s (10/10) FINISHED
=> [app internal] load build definition from Dockerfile
=> => transferring dockerfile: 159B
=> [app internal] load metadata for docker.io/library/python:3.10-bullseye
=> [app internal] load .dockerignore
=> => transferring context: 2B
=> [app 1/5] FROM docker.io/library/python:3.10-bullseye@sha256:1e759abe54585ffa460d09ef2e77abf693d95d77b038dcca23ccbb7d7a4da585
=> => resolve docker.io/library/python:3.10-bullseye@sha256:1e759abe54585ffa460d09ef2e77abf693d95d77b038dcca23ccbb7d7a4da585
=> [app internal] load build context
=> => transferring context: 5.50MB
=> CACHED [app 2/5] WORKDIR /code
=> CACHED [app 3/5] COPY requirements.txt .
=> CACHED [app 4/5] RUN pip install -r requirements.txt
=> [app 5/5] COPY . .
=> [app] exporting to image
=> => exporting layers
=> => writing image sha256:3668453d726365fb26ef353d54506c4bbe9d0c39962520eb41f0e21b6303edbd
=> => naming to docker.io/library/fastapiproject5-app
[+] Running 1/1
 Container fastProject5  Recreated
Attaching to fastProject5
fastProject5 | INFO: Will watch for changes in these directories: ['/code']
fastProject5 | INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
fastProject5 | INFO: Started reload process [1] using WatchFiles
fastProject5 | 2024-07-07 20:42:49.895734: I external/local_tsl/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
fastProject5 | 2024-07-07 20:42:50.439552: I external/local_tsl/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
fastProject5 | 2024-07-07 20:42:51.189391: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:479] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
fastProject5 | 2024-07-07 20:42:51.937451: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:10575] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
fastProject5 | 2024-07-07 20:42:51.951727: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1442] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
fastProject5 | 2024-07-07 20:42:52.909312: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
fastProject5 | To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
fastProject5 | 2024-07-07 20:43:01.316307: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
fastProject5 | INFO: Started server process [8]
fastProject5 | INFO: Waiting for application startup.
fastProject5 | INFO: Application startup complete.
```

پروژه با موفقیت اجرا شد.

پوش کردن (push) در Git به معنای ارسال تغییرات از ریپازیتوری محلی به ریپازیتوری ریموت است. این عملیات معمولاً برای اشتراک‌گذاری کار با دیگران یا به‌روزرسانی ریپازیتوری ریموت با آخرین تغییرات انجام می‌شود. در **GitHub**، شما می‌توانید تغییرات خود را به یک ریپازیتوری ریموت ارسال کنید تا دیگران نیز بتوانند به آن دسترسی داشته باشند.

مراحل پوش کردن به **GitHub**

ایجاد یا کلون کردن ریپازیتوری

اگر از قبل یک ریپازیتوری در **GitHub** ایجاد کرده‌اید، آن را کلون کنید:

```
git clone https://github.com/username/repository.git
```

اگر هنوز ریپازیتوری ندارید، می‌توانید یک ریپازیتوری جدید در **GitHub** ایجاد کنید و سپس آن را کلون کنید.

ایجاد تغییرات در ریپازیتوری محلی فایل‌های جدید اضافه کنید، فایل‌های موجود را ویرایش کنید یا فایل‌هایی را حذف کنید. اضافه کردن تغییرات به منطقه استیجینگ برای اضافه کردن تغییرات به منطقه استیجینگ، از دستور **git add** استفاده کنید:

`git add filename`

برای اضافه کردن همه تغییرات:

`git add.`

انجام یک کامیت پس از اضافه کردن تغییرات به منطقه استیجینگ، باید یک کامیت ایجاد کنید:

`git commit -m "توضیح تغییرات"`

پوش کردن تغییرات به ریپازیتوری ریموت برای پوش کردن تغییرات به ریپازیتوری ریموت، از دستور `git push` استفاده کنید:

`git push origin main`

اگر شاخه‌ی شما `main` نیست، نام شاخه مورد نظر را جایگزین کنید. استفاده از `Git Large File Storage (Git LFS)` برای فایل‌های بزرگ اگر فایلی بزرگ‌تر از 100 مگابایت دارید، باید از `Git LFS` استفاده کنید:

نصب `Git LFS`:

`git lfs install`

تبع کردن فایل‌های بزرگ:

فایل‌های بزرگ خود را با استفاده از `Git LFS` تتبع کنید:

`"# یا نام دقیق فایل *.h5" git lfs track`

اضافه کردن فایل‌های بزرگ به منطقه استیجینگ

```
git add .gitattributes
```

```
git add path/to/largefile
```

انجام یک کامیت:

```
git commit -m "اضافه کردن فایل بزرگ با استفاده از Git LFS"
```

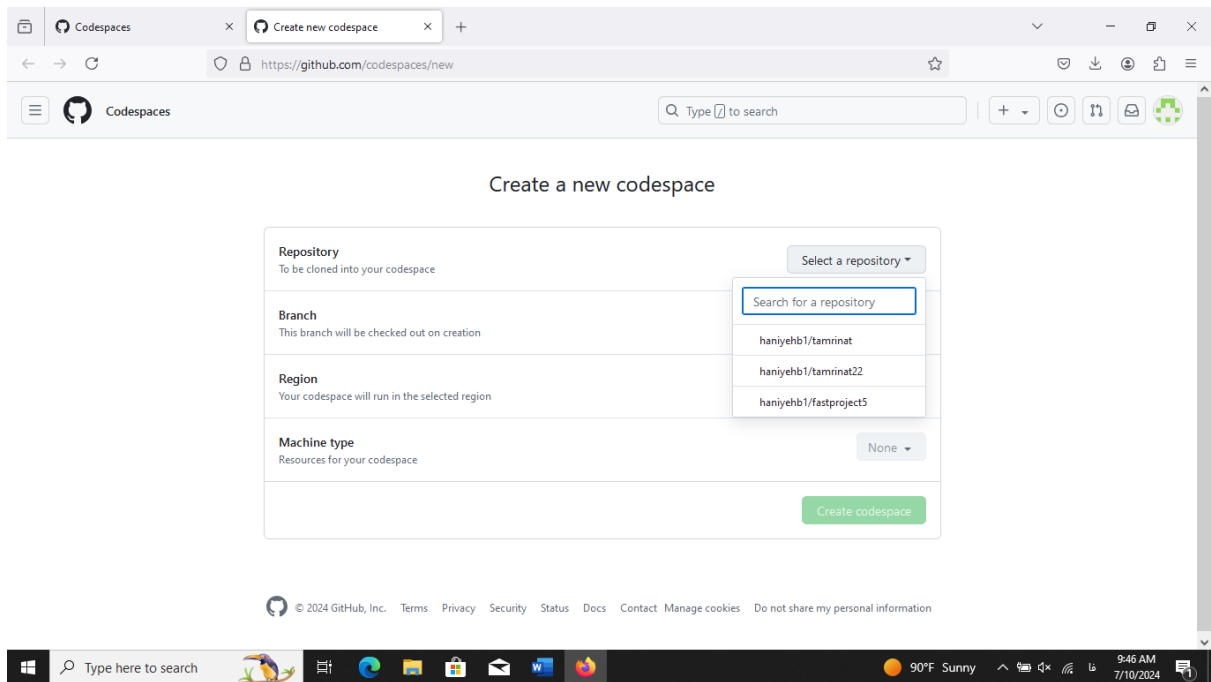
پوش کردن تغییرات به ریپازیتوری ریموت:

```
git push origin main
```

سرور

برای اتصال به سرور هم وارد سایت [codespaces](https://codespaces.com) می‌شویم

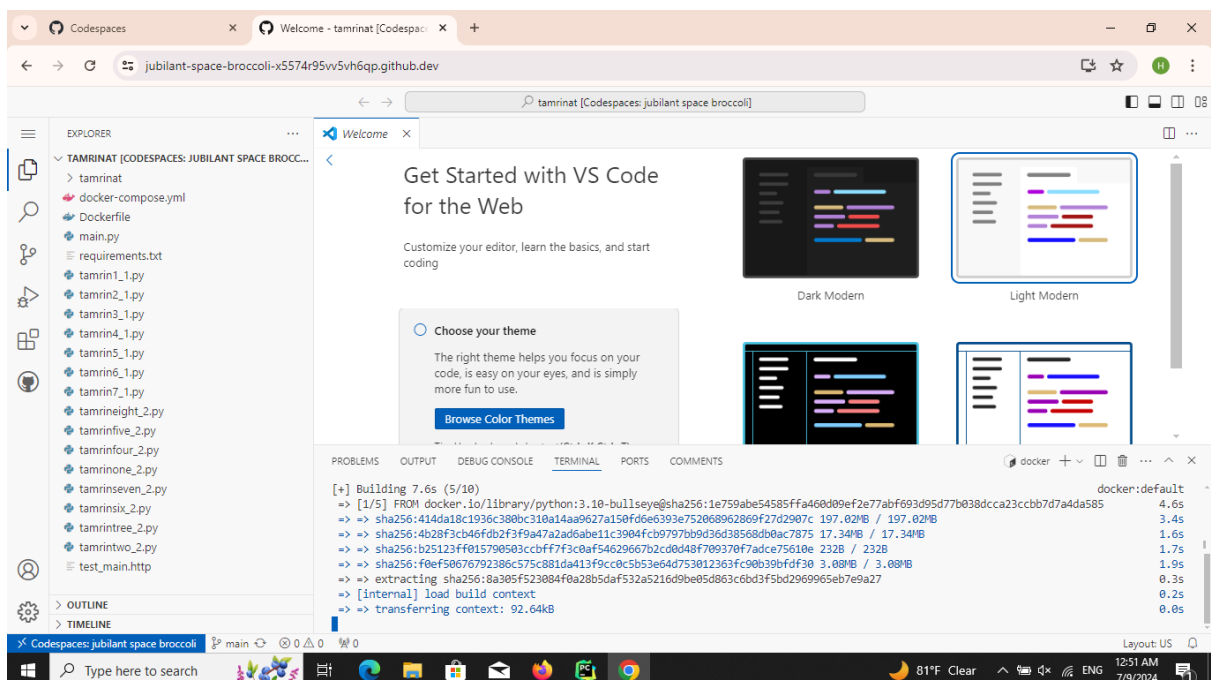
در این بخش ریپازیتوری خودمون رو انتخاب می‌کنیم

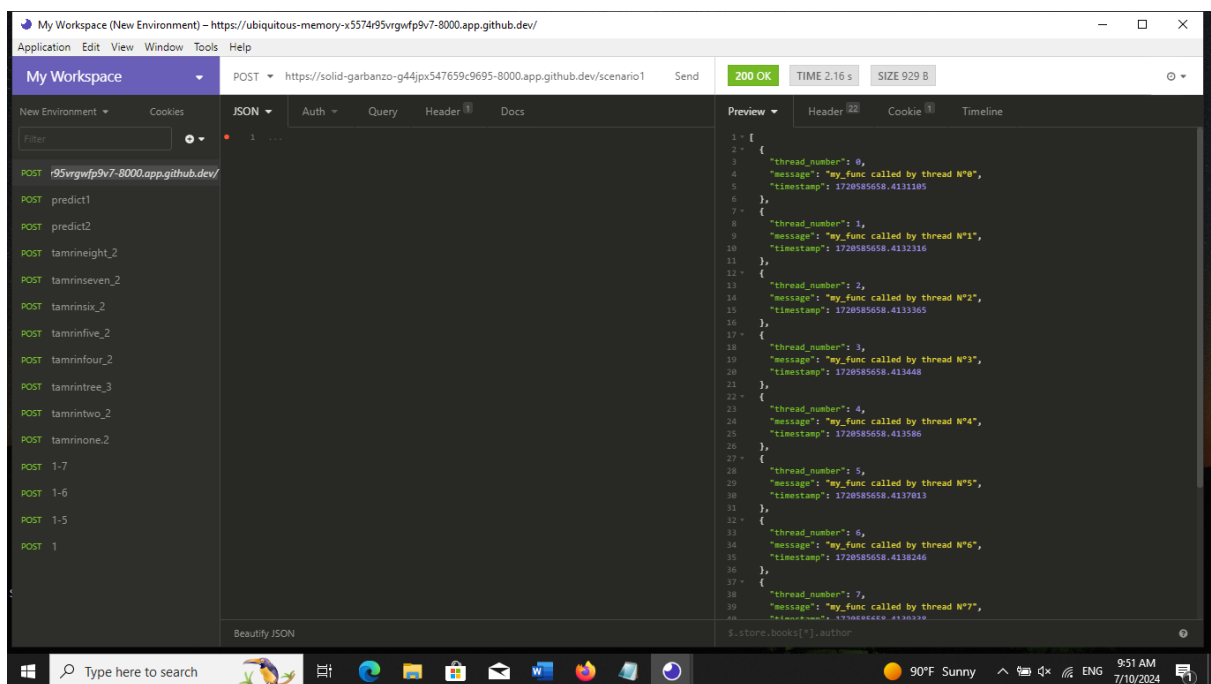
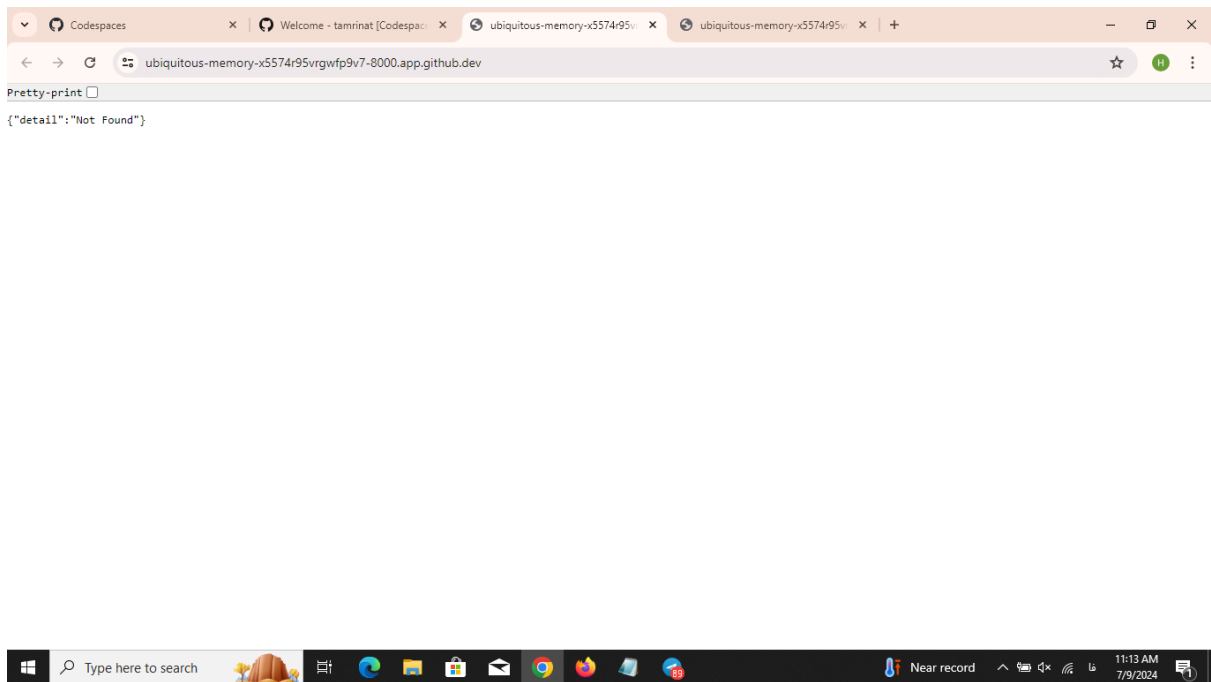


وارد محیط زیر که شدید دستور زیر را وارد میکنیم

docker-compose up

وبعد از آن به ما به یک پورت میدهد و یک آدرس سایت که وارد اون میشیم و میتونیم با استفاده از ایمسونیا ازش خروجی بگیریم





همینطور که مشاهده میشود خروجی کدهارو درایمسونیا با استفاده از لینک و یوآرال میتوان مشاهده کرد.

پایان