Absolutely! Here's a complete **end-to-end FastAPI project** setup for deploying your **MNIST CNN digit recognizer**, including:

📦 Model training & saving

🚀 FastAPI app with image upload and prediction

🐳 Dockerfile for containerization

✅ GitHub Actions for auto-deploy (optional)

🧪 Testable API endpoint

## 📁 Project Structure

mnist\_fastapi/

├── app/

│ ├── main.py ← FastAPI app

│ ├── model.py ← Model load & predict

│ └── utils.py ← Preprocessing functions

├── models/

│ └── mnist\_cnn\_model.h5 ← Trained CNN model

├── requirements.txt

├── Dockerfile

└── train\_model.py ← (Optional) model training script

## ✅ Step 1: train\_model.py (One-time training)

import numpy as np

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten

from tensorflow.keras.utils import to\_categorical

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

X\_train = X\_train.reshape(-1, 28, 28, 1).astype("float32") / 255

X\_test = X\_test.reshape(-1, 28, 28, 1).astype("float32") / 255

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

Flatten(),

Dense(100, activation='relu'),

Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=5, batch\_size=128, verbose=1)

model.save("models/mnist\_cnn\_model.h5")

## ✅ Step 2: FastAPI App

### app/main.py

from fastapi import FastAPI, File, UploadFile

from fastapi.responses import JSONResponse

import numpy as np

from PIL import Image

from app.model import predict\_digit

from app.utils import preprocess\_image

app = FastAPI(title="MNIST Digit Recognizer")

@app.post("/predict")

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

try:

image = Image.open(file.file).convert("L") # grayscale

img\_array = preprocess\_image(image)

pred = predict\_digit(img\_array)

return {"digit": int(pred)}

except Exception as e:

return JSONResponse(content={"error": str(e)}, status\_code=500)

### app/model.py

import numpy as np

from tensorflow.keras.models import load\_model

model = load\_model("models/mnist\_cnn\_model.h5")

def predict\_digit(image\_array):

image\_array = image\_array.reshape(1, 28, 28, 1)

predictions = model.predict(image\_array)

return np.argmax(predictions)

### app/utils.py

import numpy as np

import cv2

from PIL import Image

def preprocess\_image(pil\_image):

image = pil\_image.resize((28, 28))

image = np.array(image)

image = cv2.bitwise\_not(image)

image = image.astype("float32") / 255.0

return image

## ✅ Step 3: requirements.txt

fastapi

uvicorn

tensorflow

pillow

opencv-python-headless

## ✅ Step 4: Dockerfile

FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "7860"]

## ✅ Step 5: Run Locally

uvicorn app.main:app --reload

Visit: <http://localhost:8000/docs>

Upload an image (28x28 or any digit drawing), get prediction.

## ✅ Optional: GitHub Actions + HF Space (Docker SDK)

You can deploy this FastAPI Docker app to Hugging Face just like the Gradio version using the **Docker + GitHub Actions** method shared earlier.

Let me know if you want that **auto-deploy workflow YAML** with this setup too — I’ll tailor it to this structure. Or if you'd like this packaged as a template repo ready to clone and deploy.

Awesome! Here's how to complete the **FastAPI + Docker + GitHub Actions** setup for automatic deployment (e.g., to **Hugging Face Spaces**, **Render**, or **any Docker-compatible server**).

## ✅ 1. Dockerfile (already shared)

Make sure this is in your root directory (mnist\_fastapi/):

FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "7860"]

Note: --port 7860 is used by Hugging Face Spaces for FastAPI.

## ✅ 2. Add a .github/workflows/deploy.yml

Here’s a sample GitHub Actions workflow that **builds your Docker image** and pushes it to **DockerHub**.

name: Deploy FastAPI App

on:

push:

branches: [main]

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout repo

uses: actions/checkout@v3

- name: Set up Docker Buildx

uses: docker/setup-buildx-action@v3

- name: Login to DockerHub

uses: docker/login-action@v2

with:

username: ${{ secrets.DOCKERHUB\_USERNAME }}

password: ${{ secrets.DOCKERHUB\_TOKEN }}

- name: Build and push Docker image

uses: docker/build-push-action@v5

with:

context: .

push: true

tags: ${{ secrets.DOCKERHUB\_USERNAME }}/mnist-fastapi:latest

## ✅ 3. Set up secrets in GitHub

Go to your GitHub repo → **Settings → Secrets → Actions** and add:

DOCKERHUB\_USERNAME → your Docker Hub username

DOCKERHUB\_TOKEN → [DockerHub Access Token](https://hub.docker.com/settings/security)

## ✅ 4. Push and Deploy

Once this is in place:

git add .

git commit -m "Add GitHub Actions CI/CD workflow"

git push origin main

This will:

Build your FastAPI Docker image

Push it to Docker Hub

## 🚀 Deploy to Hugging Face Spaces (Docker mode)

Go to: <https://huggingface.co/spaces>

Click **Create New Space**

Choose:

SDK: **Docker**

Space name: e.g., mnist-fastapi

Visibility: **Public** or **Private**

Connect your repo or upload the files manually.

Hugging Face will read your Dockerfile and host it!

## 📦 FastAPI App Endpoint

Once deployed, you’ll get a link like:

https://username-mnist-fastapi.hf.space/docs

From here, you can test /predict using image uploads.

Do you want me to zip up a **ready-to-upload GitHub repo** with all of this included? Or walk you through a specific deployment platform like **Render**, **Railway**, or **AWS** too?