Build a FastAPI for MNIST digit prediction

Let’s build a FastAPI that exports the MNIST digit classification functionality for users to consume over REST call. You can use the best MNIST model (out of 10 variants), based on the test data performance from our last assignment. Recall our classwork where we saw how to build an API that allows upload of a file. Users may upload the picture of a digit to your API and wait for your API to reply back with the “digit” as the output.

Task 1: [30 pts]

1. Create a FastAPI module.
2. Take the path of the model as a command line argument.
3. Create a function “def load_model(path:str) -> Sequential” which will load the model saved at the supply path on the disk and return the keras.src.engine.sequential.Sequential model.
4. Create a function “def predict_digit(model:Sequential, data_point:list) -> str” that will take the image serialized as an array of 784 elements and returns the predicted digit as string.
5. Create an API endpoint “@app post(‘/predict’)” that will read the bytes from the uploaded image to create an serialized array of 784 elements. The array shall be sent to ‘predict_digit’ function to get the digit. The API endpoint should return {“digit”:digit”} back to the client.
6. Test the API via the Swagger UI (<api endpoint>/docs) or Postman, where you will upload the digit as an image (28x28 size).

Task 2: [20 pts]

1. Create a new function “def format_image” which will resize any uploaded images to a 28x28 grey scale image followed by creating a serialized array of 784 elements.
2. Modify Task 1 to incorporate “format_image” inside the “/predict” endpoint to preprocess any uploaded content.
3. Now, draw an image of a digit yourself using tools such as “ms paint” or equivalent using your touch screen or the mouse pointer. Upload your hand drawn image to your API and find out if your API is able to figure out the digit correctly. Repeat this exercise for 10 such drawings and report the performance of your API/model combo.

Important Pointers

1. You should strictly follow the function prototypes when you build the functions.
2. You are expected to submit the python scripts with the necessary inline comments.
3. You have to maintain this project in github with proper documentation and folder structure.
4. Here is the allocation of points per task
   10% for a clean coding style
   30% for the correctness of the implementation
   10% for github project
   20% for readable comments
   10% for input arguments’ validation/boundary check
   20% for unit test modules

Best wishes.