A06 - Build a FastAPI for MNIST Digit Prediction

Vinayak Gupta
EE20B152

Introduction

This report details the development and evaluation of a FastAPI application designed for MNIST digit classification. The application leverages a pre-trained convolutional neural network (CNN) model to predict the digit depicted in a user-uploaded image. This report builds upon the findings of a previous assignment, where various CNN configurations were explored for MNIST classification. The best performing model from that exploration serves as the foundation for this project.

The report is structured into two key tasks. The first task focuses on building the core API functionality. It outlines the creation of the FastAPI application, including loading the trained model, defining prediction logic, and establishing an endpoint for receiving and processing upload requests. This initial implementation allows users to submit images for digit classification through a REST API call.

The second task expands upon the core functionality by incorporating image pre-processing. A new function is introduced to handle uploaded images, ensuring they are resized to the required 28x28 format and converted to a grayscale representation before being fed into the model for prediction. This pre-processing step enhances the accuracy and robustness of the API across various image formats.

Finally, the report evaluates the performance of the API/model combination. We assess the ability of the system to correctly identify hand-drawn digits submitted by the user. By testing with a set of ten drawings, the report aims to provide insights into the effectiveness of the developed FastAPI application for MNIST digit classification.

Task #1: Detecting the Digit given MNIST Images

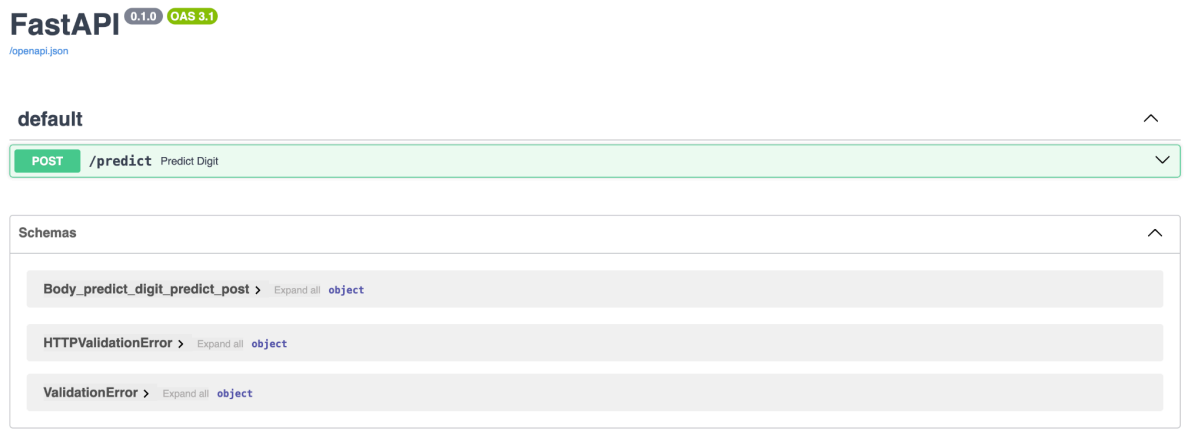
- load_model() function

```python
def load_model(model_path):
    model = Net()
    model.load_state_dict(torch.load(model_path))
    model.eval()

    return model
```

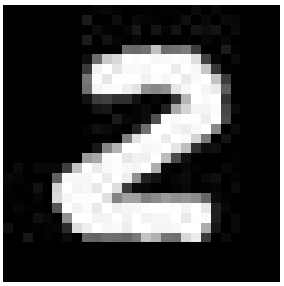
- predict_digit() function

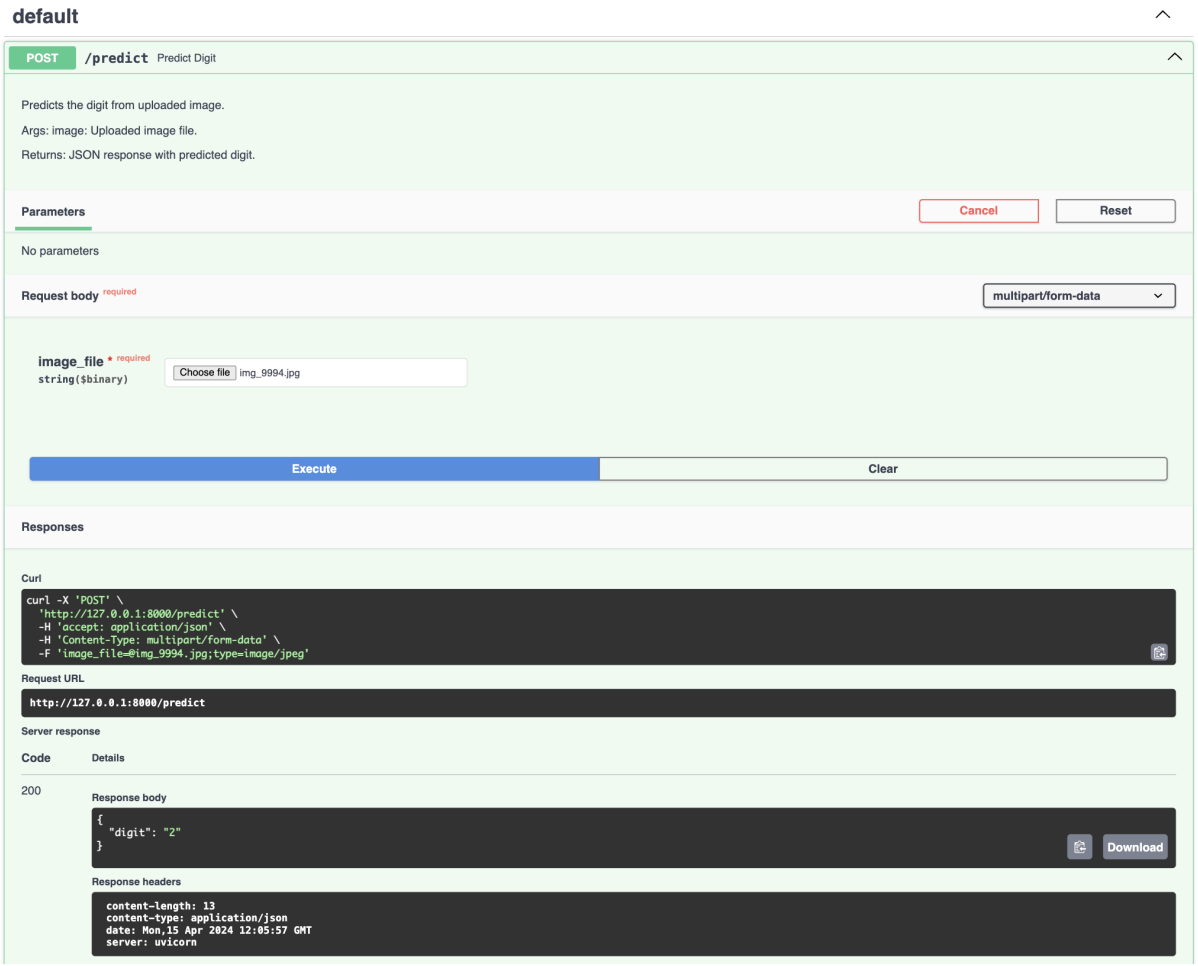
```python
async def predict_digit(image_file: UploadFile = File(...)):

    image_bytes = image_file.file.read()
    class_id = get_prediction(image_bytes)
    result = {
            "digit": class_id
        }
    return result
```

Creating an endpoint with '/predict' and checking that in **http://127.0.0.1:8000/docs**



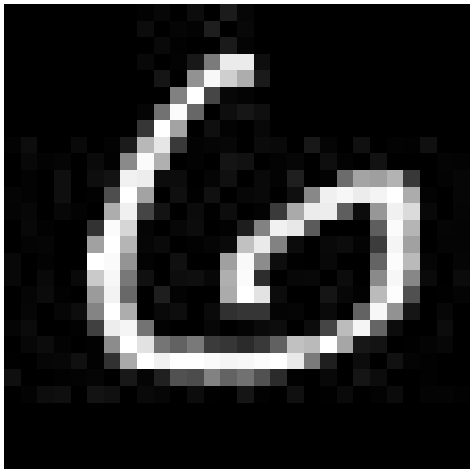
Given the below input image, we get the correct output by our model.



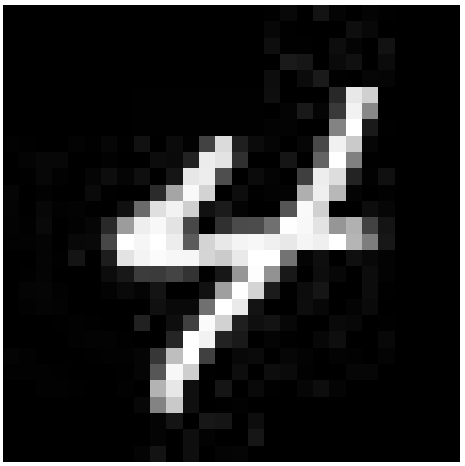


The predicted digit is 2 as seen in the response body.

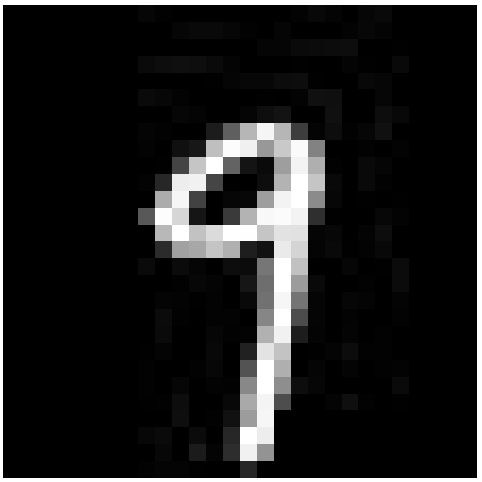
Some More Examples



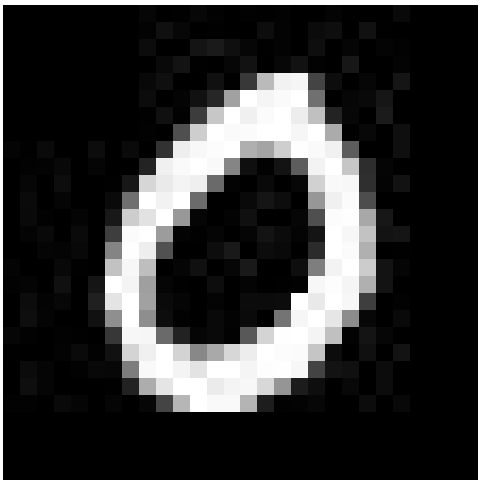
Predicted Digit - 6



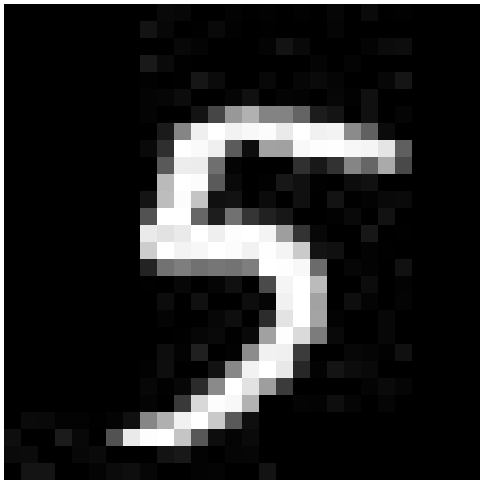
Predicted Digit - 4



Predicted Digit - 9



Predicted Digit - 0



Predicted Digit - 5

Predicted Digit - 3

Our model achieved 97% accuracy on Test Data

Task #2: Detecting the Digit given Hand Drawn images

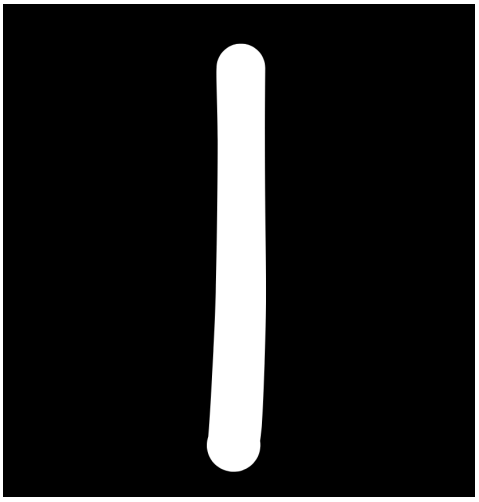
- format_image() function

```
def format_image(image_bytes):
    image = Image.open(io.BytesIO(image_bytes))
    image = image.resize((28, 28))
    image = np.array(image)
    try:
        if image.shape[2] == 3 or image.shape[2] == 4:
            image = 0.2989*image[:, :, 0] + 0.5870*image[:, :, 1] + 0.1140*image[:, :, 2]
    except:
        pass
    image = image.astype(np.float32)
    return image
```

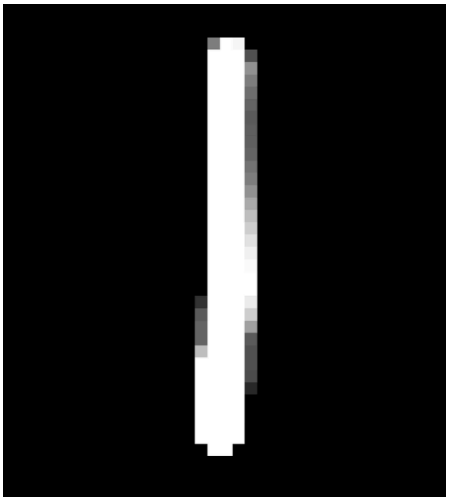
The above takes in any sized image and resizes it and converts it to a grayscale image.

Examples -

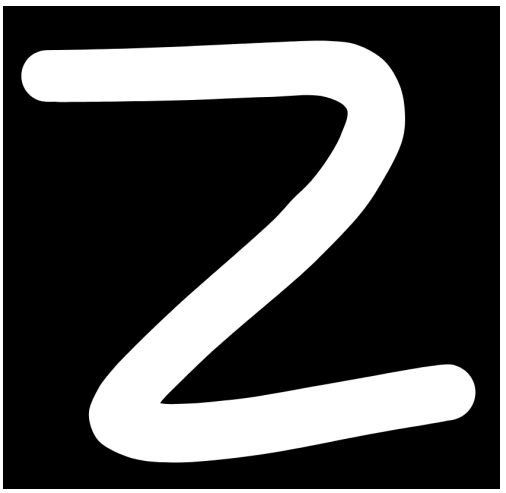
Shown examples on Images directly drawn on Canva and also side-by-side shown results after pixelating the image in Canva and then giving to the model.



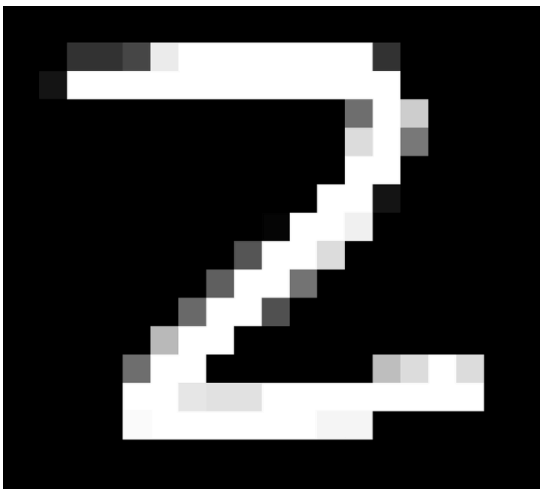
Predicts - 1



Predicts - 1

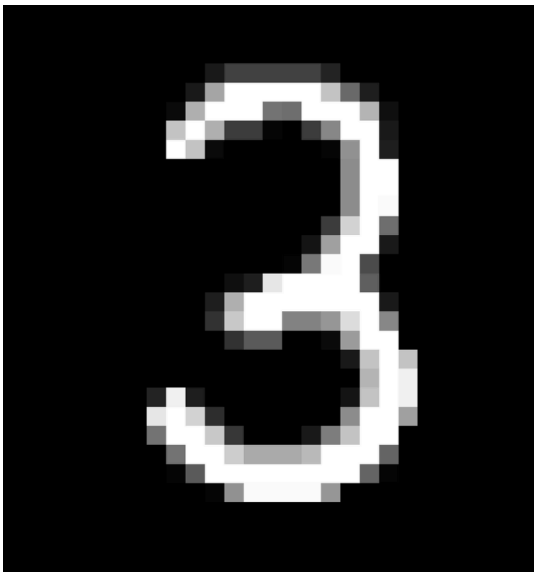


Predicts - 2

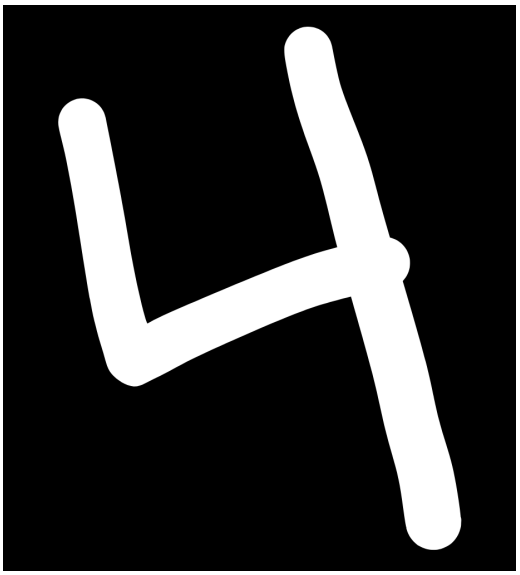


Predicts - 2

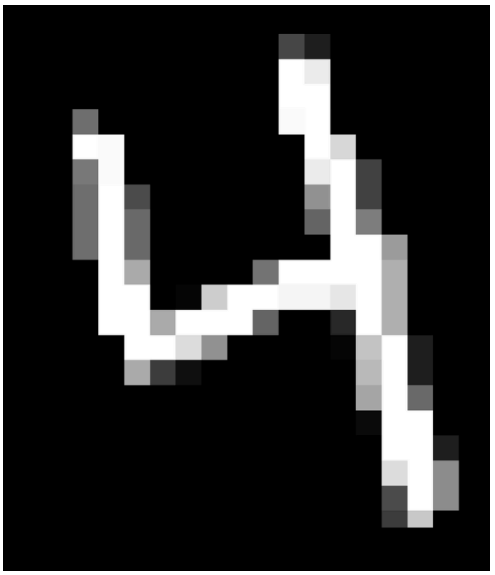
Predicts - 3



Predicts - 3

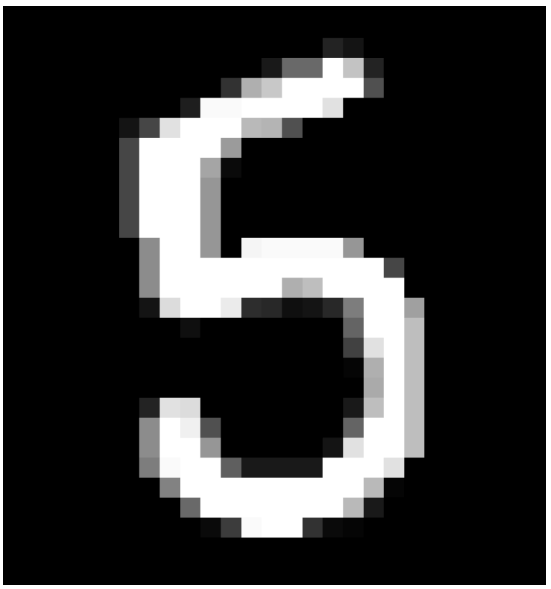


Predicts - 2

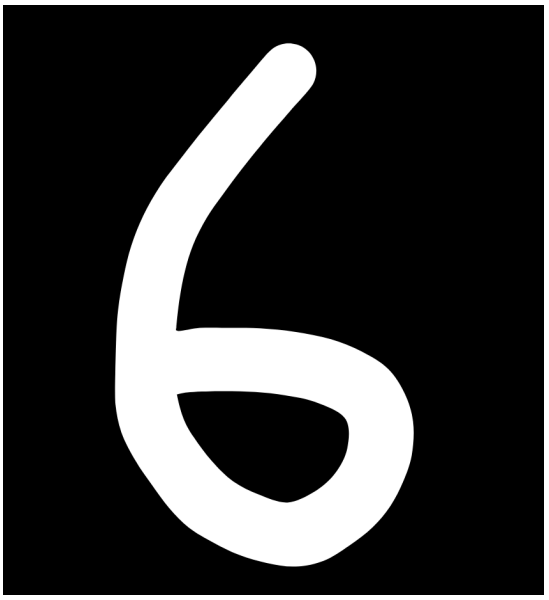


Predicts - 9

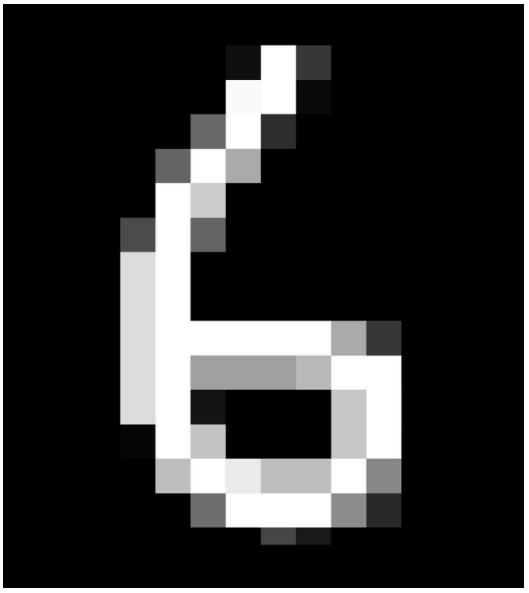
Predicts - 3



Predicts - 5

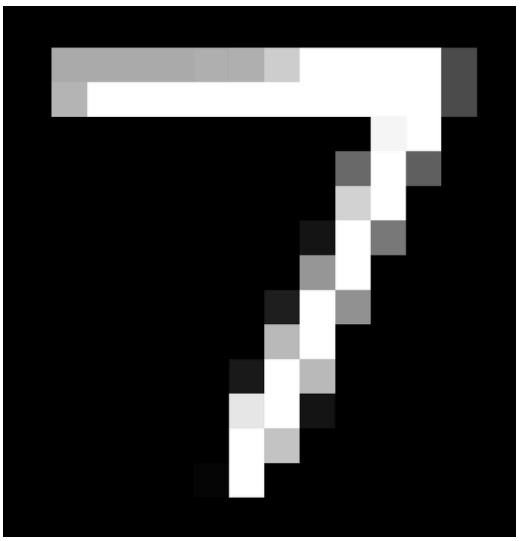


Predicts - 8

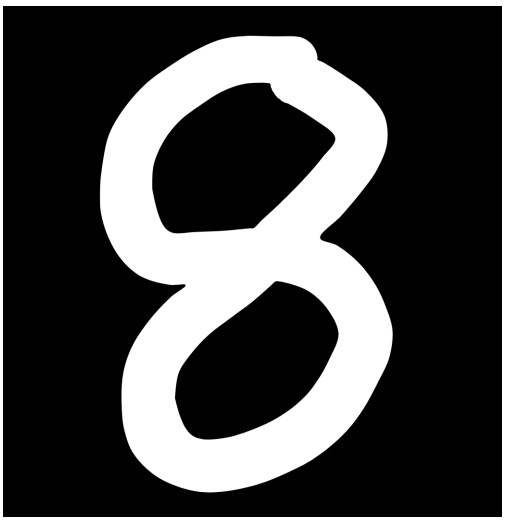


Predicts - 5

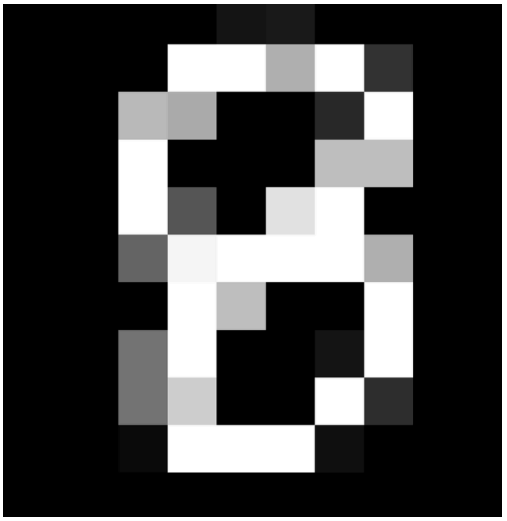
Predicts - 3



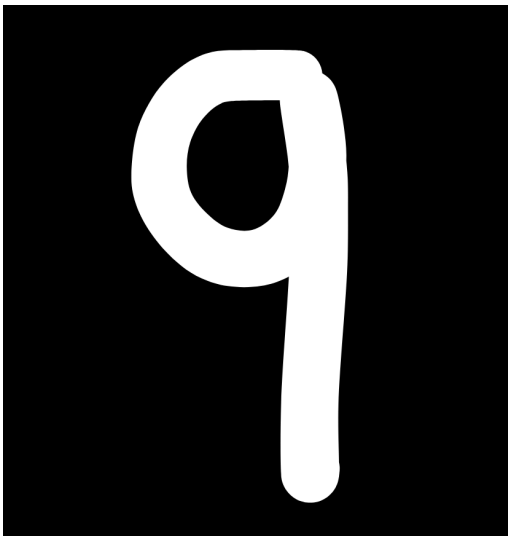
Predicts - 3



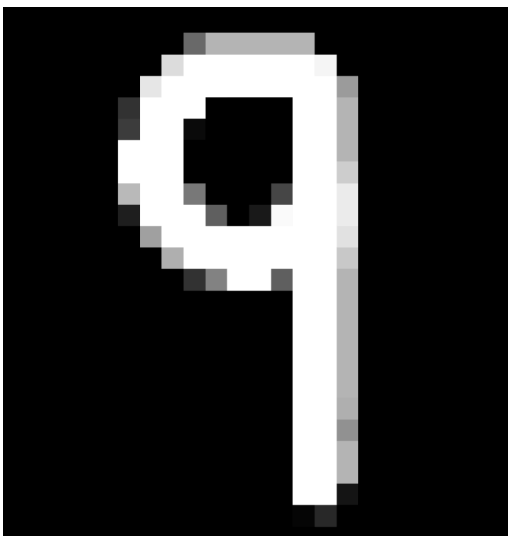
Predicts - 3



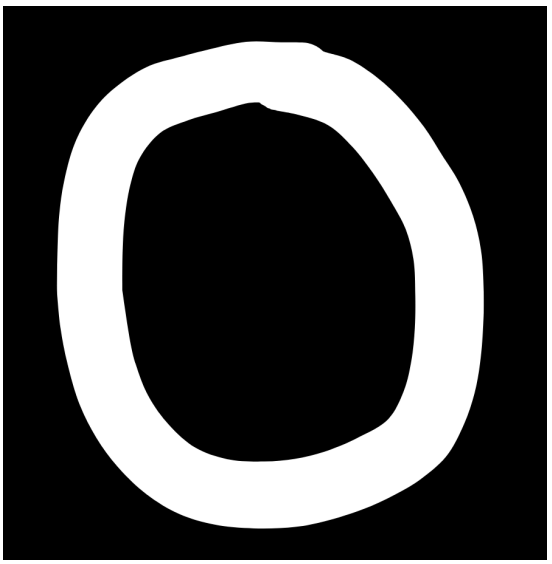
Predicts - 8



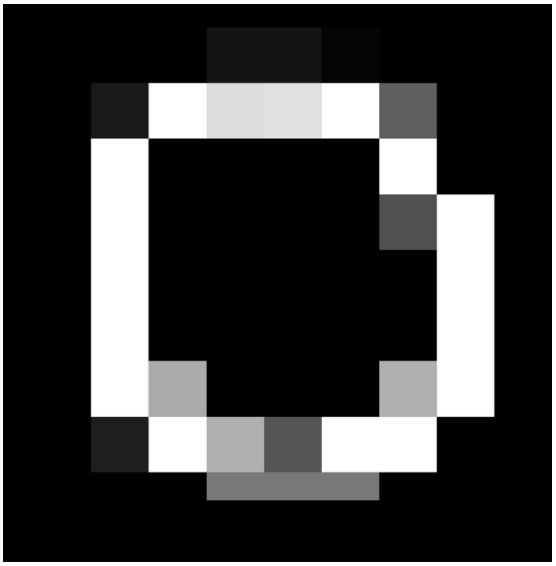
Predicts - 3



Predicts - 3



Predicts - 2



Predicts - 0

We observe that a few times for both hand-drawn and pixelated images, the model predicts correctly. Few times, the pixelated image is only predicted correctly and at few places both don’t get predicted properly. The model seems to be very data sensitive. Overfitted to the original data.