Machine Intellect Society of GIT

Tech Team - Task 2 : Report

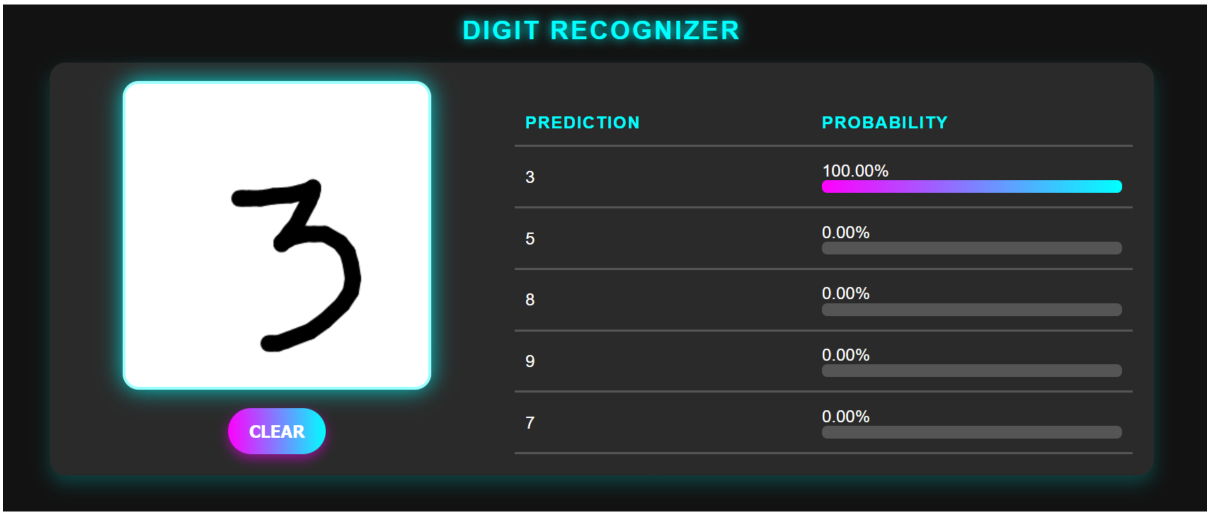
– Rahul Samedavar

Realtime Digit Recognizer

Overview

The **DigitRecognizer** project focuses on developing a **Convolutional Neural Network (CNN)** model for recognizing handwritten digits (0-9) using the **MNIST dataset**. The model is trained for multi-class classification and deployed with an interactive UI for real-time digit recognition.

The final model achieves an impressive **99.37% accuracy** with a **99.94% Top-K accuracy**, making it highly reliable for digit classification.



Github: https://github.com/Rahul-Samedavar/DigitRecognizer

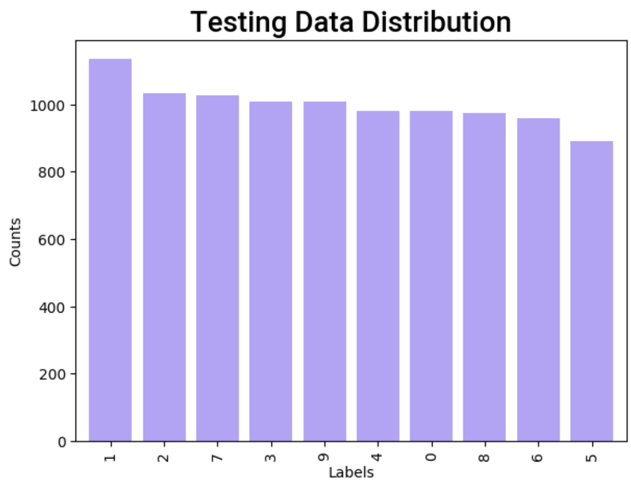
Deployed in HuggingFace space: https://huggingface.co/spaces/Rahul-Samedavar/DigitRecognizer

Note: Refer to Readme of Github repo for in-depth Explaination.

Project Implementation

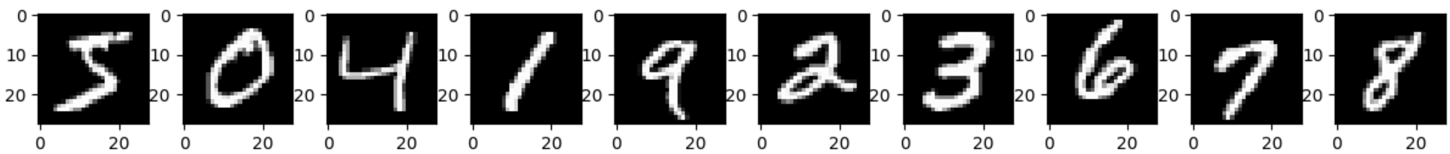
A) Dataset Preparation

- The project uses the **MNIST dataset**, which consists of:
    - **60,000 training images**
    - **10,000 testing images**
- Each image is a **28x28 grayscale** representation of a handwritten digit.



Preprocessing Steps:

1. **Resizing & Normalization:**
    - Images resized to **28x28** pixels.
    - Pixel values normalized to the **[0, 1]** range for improved convergence.
    - Digits part was cantered using the bounding box method for uniformity.



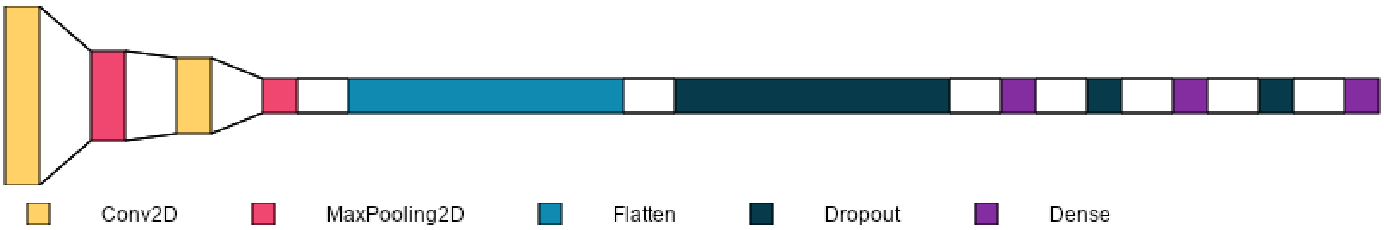
2. **Label Encoding:**
    - Labels were **one-hot encoded** for multi-class classification.

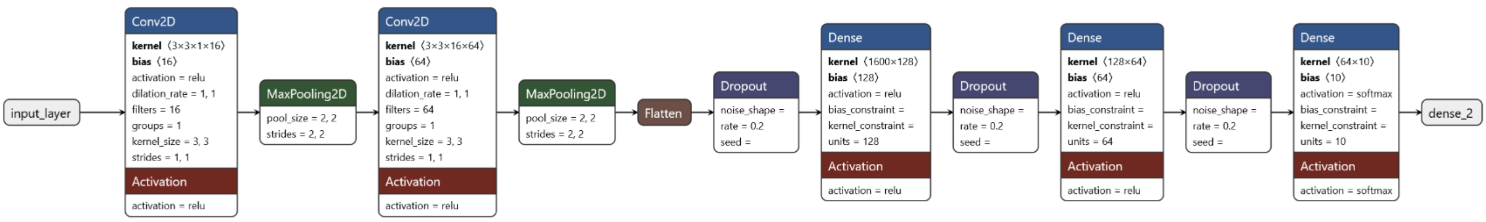
B) Model Architecture

The CNN model was designed with the following layers:

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Conv2D, Flatten, Dense, Dropout


model = Sequential([
    Conv2D(16, (3,3), activation='relu', input_shape=(28,
28, 1)),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(10, activation='softmax')
])
```





C) Loss Function & Optimization

Loss Function: Categorical Crossentropy (suitable for multi-class classification)

Optimizer: Adamax (optimal for noisy data and stability)

Learning Rate: 0.001 (fine-tuned for optimal performance)

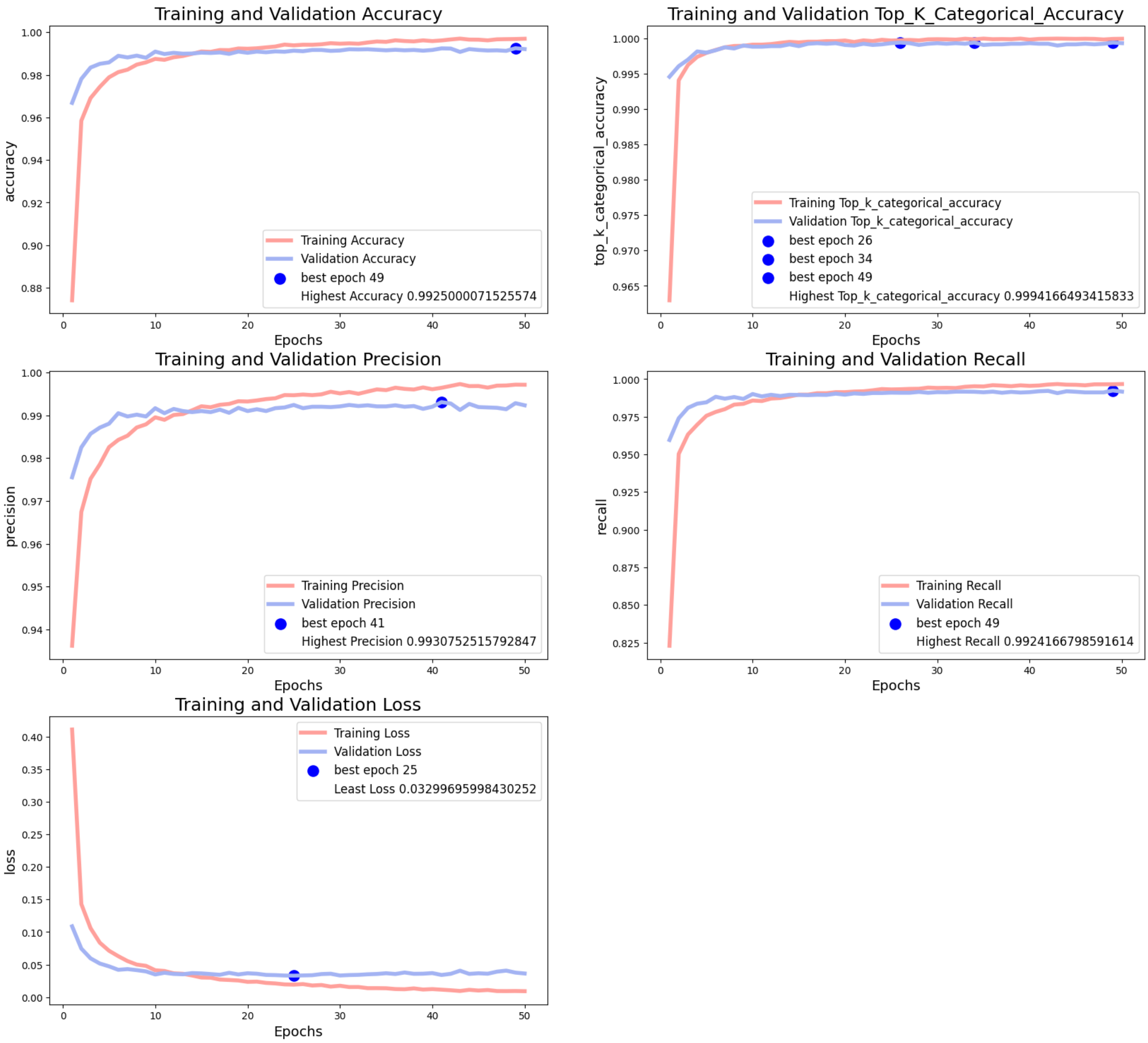
```
model.compile(
   optimizer=Adamax(0.001),
   loss='categorical_crossentropy',
   metrics=['accuracy',  TopKCategoricalAccuracy(3),
Precision(), Recall()]
)
```

D) Training Process

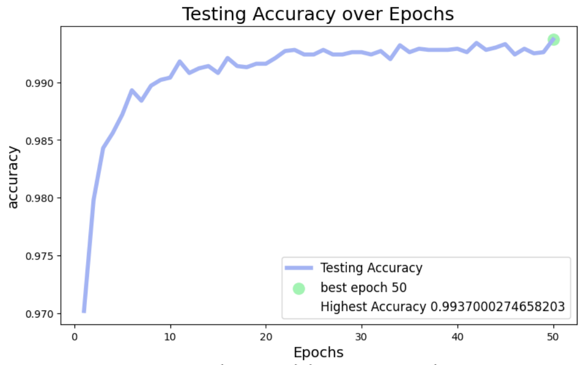
- **Train-Test Split:** 85% training, 15% validation.
- **Batch Size:** 32 (efficient for GPU memory usage).
- **Epochs:** 50 (to ensure model convergence).
- **Model Checkpointing:**
  - Saved the model at each epoch to preserve the best-performing weights.

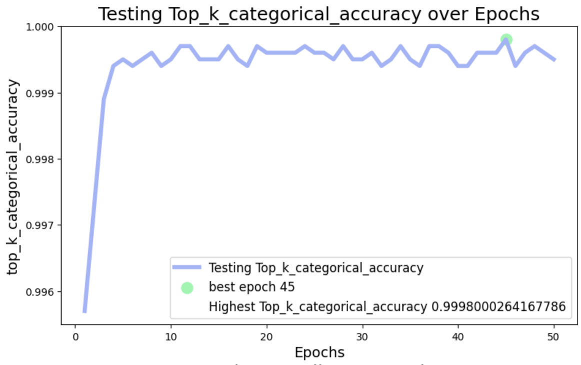
```
from tensorflow.keras.callbacks import ModelCheckpoint
checkpoint = ModelCheckpoint(
    filepath = "Models/model_epoch_{epoch:02}.keras",
    save_weights_only = False,
    save_best_only = False,
    monitor = 'val_loss',
    mode = 'min',
    verbose = 0
)
```

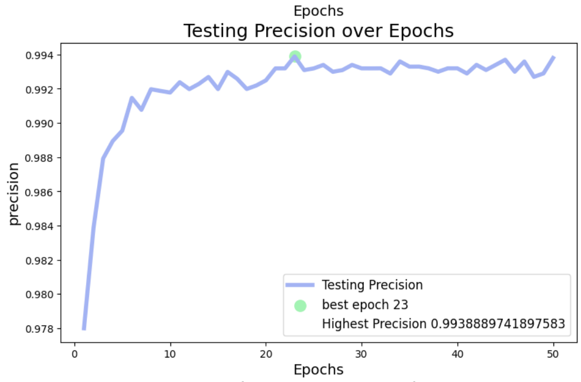
Training and Validation Metrics

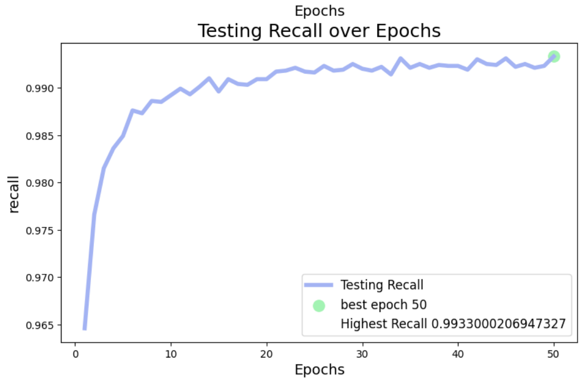


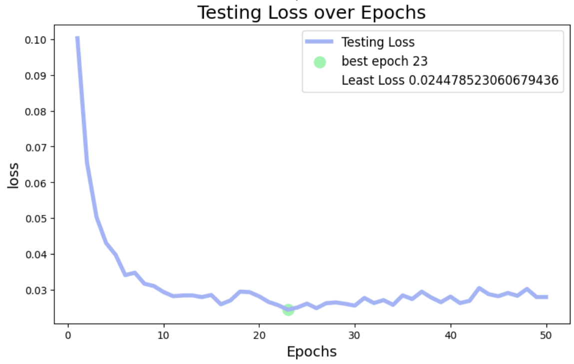
Testing Metrics





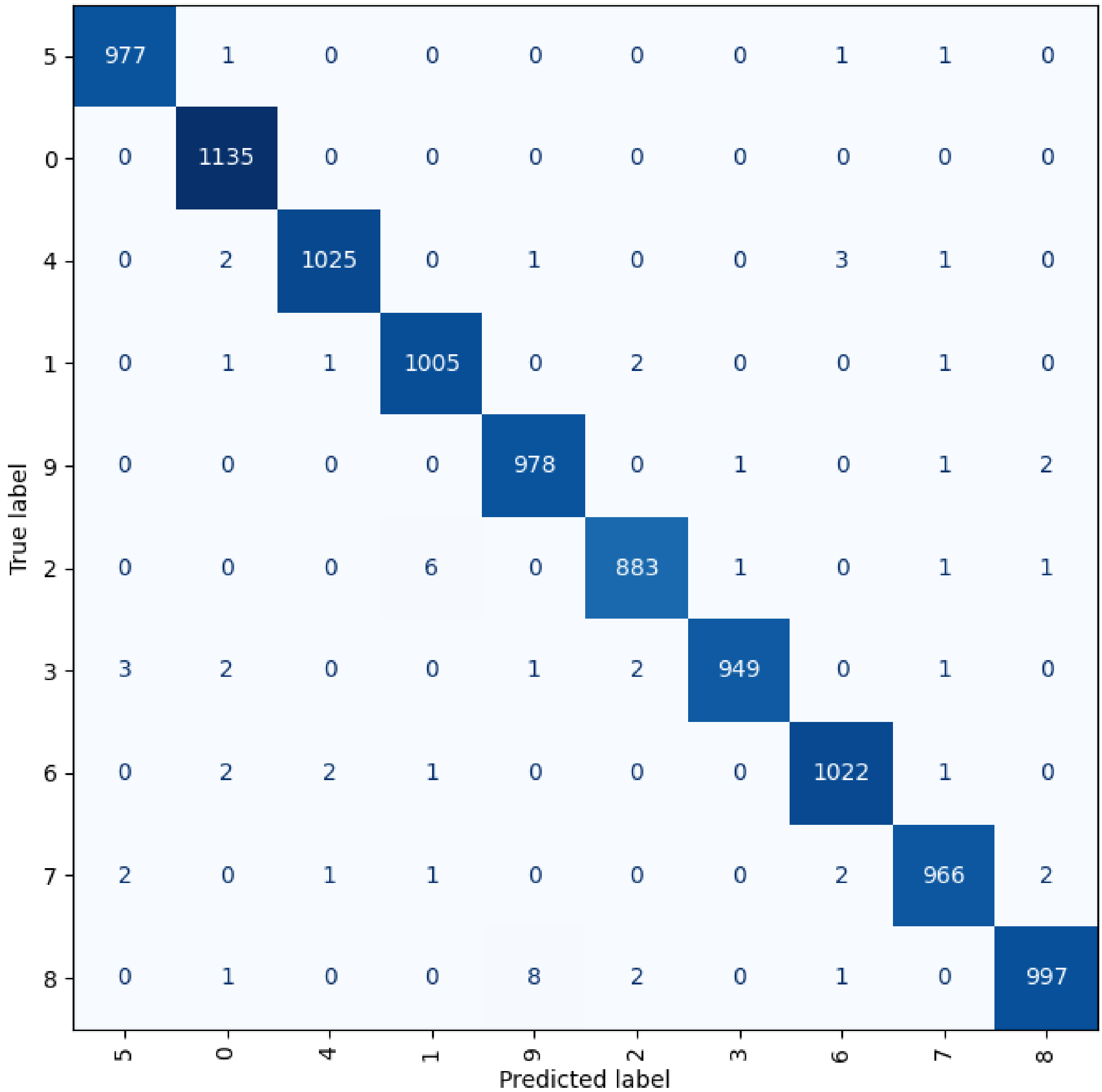






Results:

- **Best Accuracy:** 99.37% at epoch 50.
- **Best Top-K Accuracy:** 99.94% at epoch 45.
- **Precision:** 99.39%.
- **Recall:** 99.33%.
- **Inference Speed:** ~40ms per image (real-time).

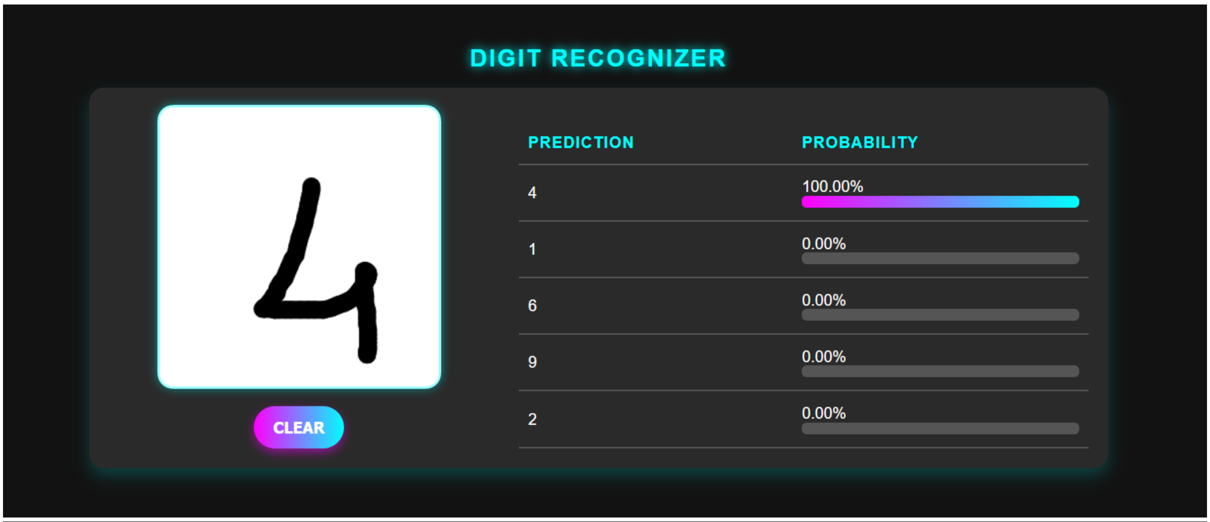


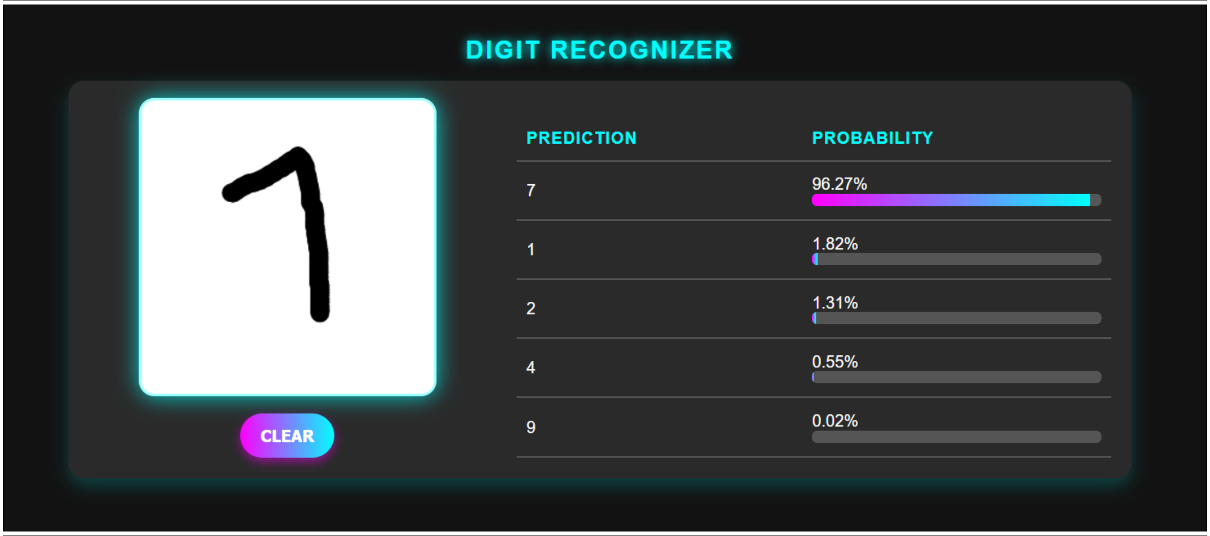
Confusion Matrix

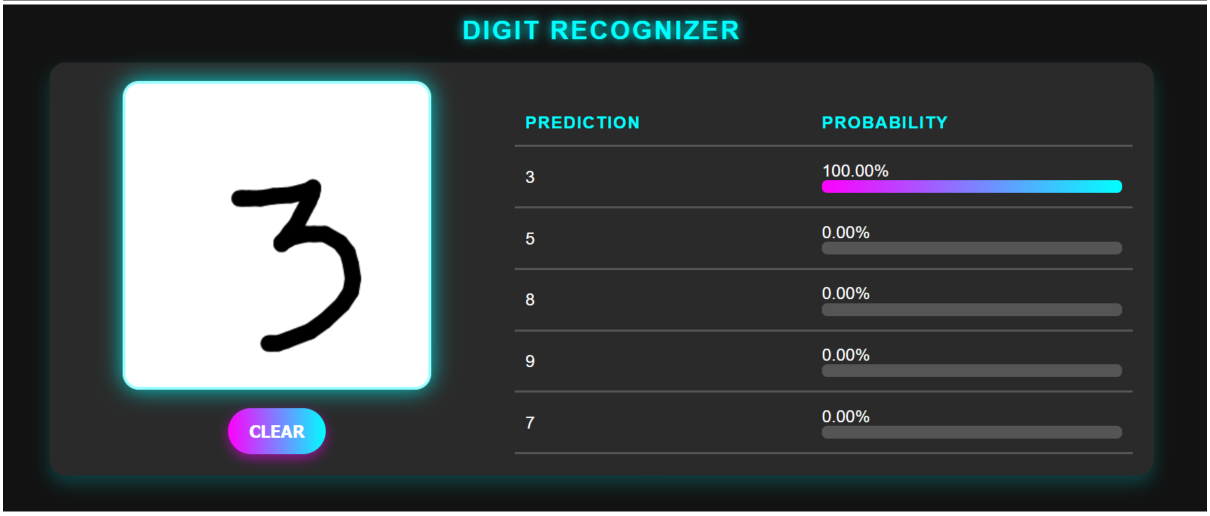
F) Deployment

- The model and GUI were deployed on **Hugging Face Spaces** for easy accessibility.
- The interactive UI allows users to:
    - Draw digits on a canvas.
    - View predictions in real-time with confidence scores.
    - See top-5 predictions with probability bars.

Images:


DIGIT RECOGNIZER
PREDICTION
PROBABILITY
4
100.00%
1
0.00%
6
0.00%
9
0.00%
2
0.00%
CLEAR


DIGIT RECOGNIZER
PREDICTION
PROBABILITY
7
96.27%
1
1.82%
2
1.31%
4
0.55%
9
0.02%
CLEAR


DIGIT RECOGNIZER
PREDICTION
PROBABILITY
3
100.00%
5
0.00%
8
0.00%
9
0.00%
7
0.00%
CLEAR

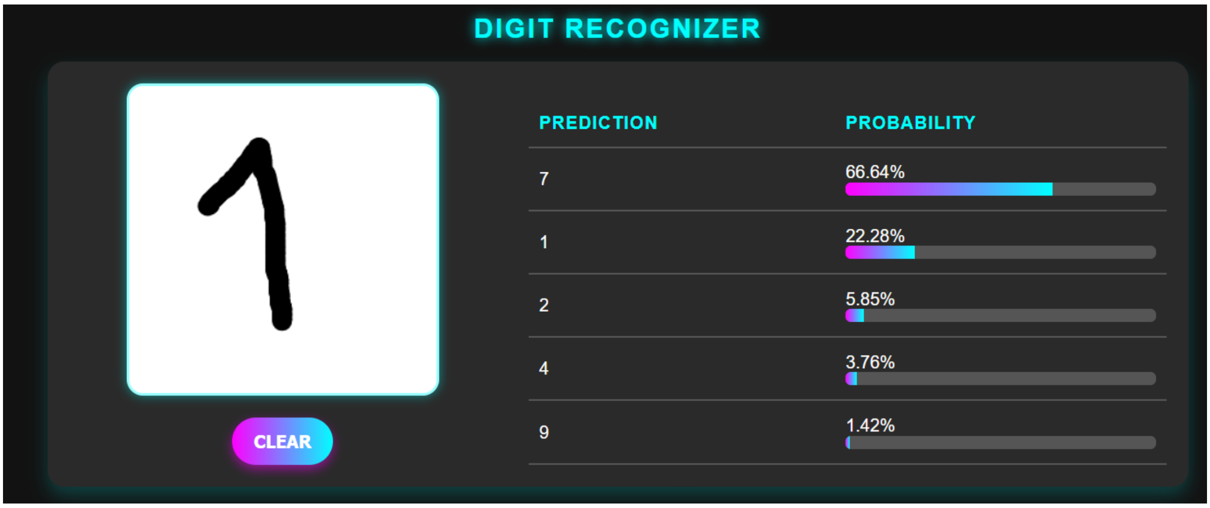
Tools & Technologies Used

- **Programming Languages & Frameworks:**
  - Python
  - TensorFlow - Keras
  - Flask with (HTML, CSS and JS).
- **Libraries:**
  - numpy
  - matplotlib
  - tensorflow – keras
  - visualkeras: model visualisation
  - Flask
- **Deployment:**
  - **Hugging Face Spaces** for hosting with gunicorn.
- **Version Control:**
  - GitHub
- **Visualizing Model:**
  - Netron

Challenges Faced:

1. Similarity Between different classes:

When written by hand there might be unambiguity. For example 4 and 9 are similar, 1 and 7 are similar. Sometimes it is harder for model to discriminate between them.

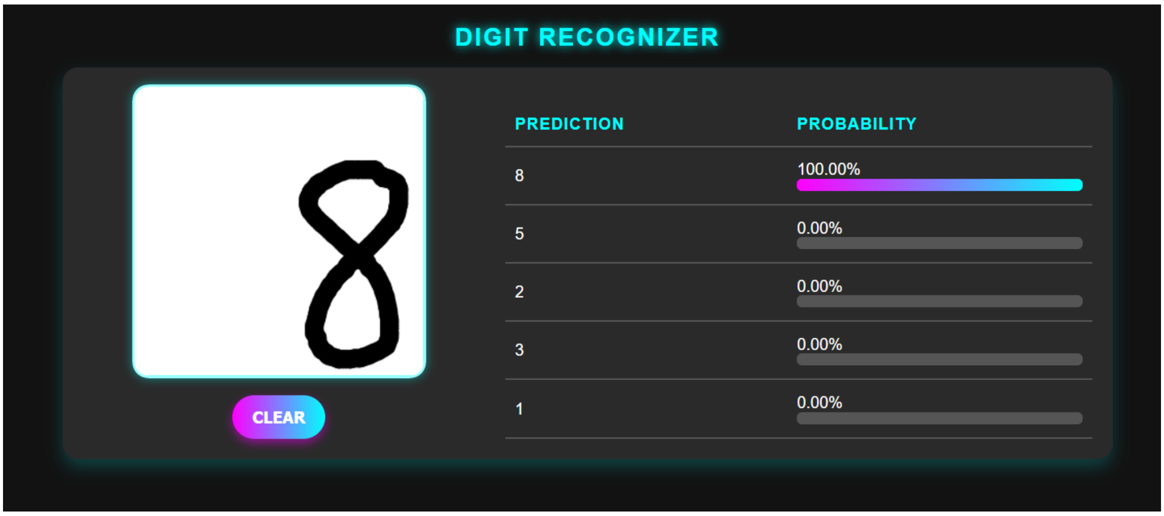


Solution : Included Top k Accuracy metrics, with k =3, to atleast include the correct class in best 3 predictions in such cases.

2. Digit Positioning

Initially although the model excelled in testing data, it struggled a bit when I tried realtime drawing with canvas. Specifically, when drawn near edges.

Solution: MNIST data preparation details hinted that they had centred the digit for uniformity. So I followed same preprocessing steps and centred the digit from canvas. Hence the problem was fixed.



3. Fancy Digits

When my Friends were playing with this canvas, I Observed that, when My friends draw digits in fancy way, The model was predicting wrong occasionally. Because This fancy digits had unnecessary features that my model recognized.

Solution: I had couple of Dropout Layers with decent rate, to ignore unnecessary features.