

This is a web app that recognises handwritten digits between 0 and 9. The prediction is made using a Convolutional Neural Network (CNN) trained on the well-known MNIST database.

The user draws a digit on the slate and then presses the predict button. If the app predicts wrong, the user types in the correct number and presses the submit feedback button. To try another digit, click on the delete icon below the slate and draw another digit as before. A history of previous digits drawn by the user is shown in a table below the slate together with a running prediction accuracy of the app. However this table is lost once the app is refreshed. Persistent storage of past predictions is done in a Postgres database hosted on the same web server,

Try it out here: URL: <http://138.199.200.113:8501>

The screenshot shows a web browser window with the URL 'localhost:8501' in the address bar. The page title is 'Handwritten Digit Recognition'. It features a drawing slate where a handwritten digit '3' is drawn. Below the slate is a 'Predict' button. To the right, a box displays 'Prediction: 3' and 'Confidence: 99%'. There is also a text input field for 'Correct digit if incorrect:' and a 'Submit Feedback' button. Below the slate, there's a section titled 'Example Drawing from Training Data' showing a grid of handwritten digits. At the bottom, a table titled 'Previous Predictions' shows two entries. A footer at the bottom of the page displays 'Total Attempts: 2 | Correct Predictions: 2 | Realized Accuracy: 100.00%'.

Index	Timestamp	Prediction	Confidence	Actual
0	1 2025-04-01 08:34:56	4	99%	4
1	2 2025-04-01 08:35:06	3	99%	3

Project Structure

graphql

Copy
 Edit

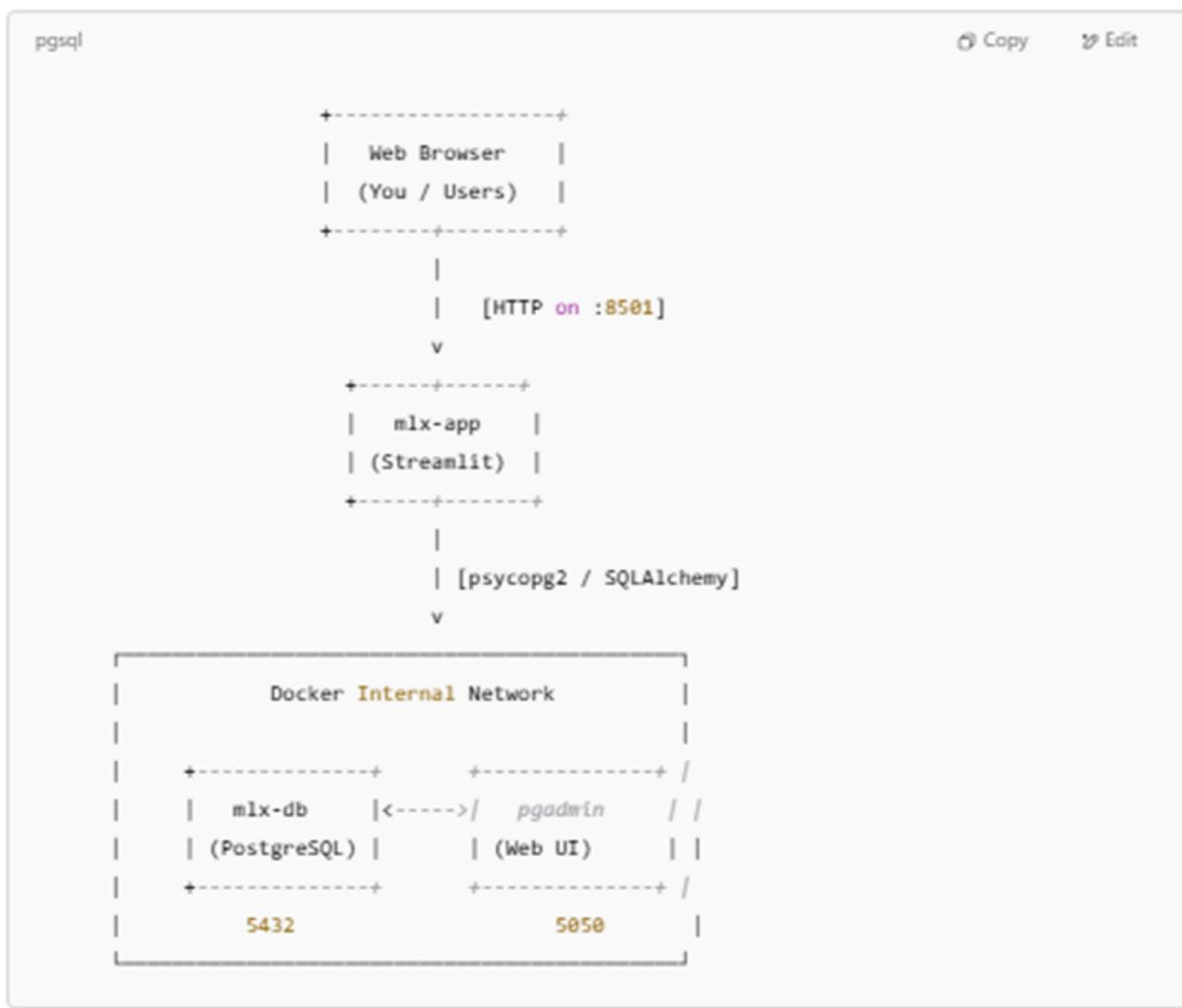
```

MLX_PROJECT/
├── app/                                // Streamlit + ML Logic
│   ├── main.py                           // UI + prediction logic
│   ├── CNNModelMNIST.py                 // PyTorch model
│   ├── model.pth                         // Trained model weights
│   ├── Dockerfile                        // Docker config for Streamlit app
│   └── ...
|
├── db/                                   // Postgres SQL init scripts
│   ├── init.sql                          // CREATE DATABASE script
│   └── create_table.sql                  // CREATE TABLE IF NOT EXISTS ...
|
├── .env                                  // App + DB + pgAdmin environment variables
├── docker-compose.yaml                  // Defines services: app, db, pgadmin
├── requirements.txt                     // Python dependencies
└── .gitignore
└── README.md

```

Container	Purpose	Port(s) Exposed	Volume Persistence
mlx-app	Your Streamlit app for digit recognition	8501 → host	Not needed
mlx-db	PostgreSQL database	5432 (internal only)	pgdata:/var/lib/postgresql/data <input checked="" type="checkbox"/>
pgadmin	pgAdmin UI to inspect the DB	5050 → host	pgadmin-data:/var/lib/pgadmin <input checked="" type="checkbox"/>

✳️ MLX Project – Container Interaction Diagram



Data

The MNIST dataset was used

Preprocessing:

Model Architecture

-

```

1 # Define the CNN model by subclassing nn.Module
2 class CNNModel(nn.Module):
3     def __init__(self):
4         super().__init__() # Initialize the base class
5         # Convolutional layer 1: Input channels = 1 (grayscale), Output channels = 16
6         self.conv1 = nn.Conv2d(1, 16, kernel_size=3, padding=1)
7         # Convolutional layer 2: Input channels = 16, Output channels = 32
8         self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
9         # Max pooling layer: Reduces spatial dimensions
10        self.pool = nn.MaxPool2d(2, 2)
11        # Fully connected layer 1
12        self.fc1 = nn.Linear(32 * 7 * 7, 128)
13        # Fully connected layer 2 (Output layer)
14        self.fc2 = nn.Linear(128, 10)
15        # Dropout layer to prevent overfitting
16        self.dropout = nn.Dropout(0.25)
17
18    def forward(self, x):
19        # Convolutional layer 1 followed by ReLU activation and pooling
20        x = self.pool(F.relu(self.conv1(x)))
21        # Convolutional layer 2 followed by ReLU activation and pooling
22        x = self.pool(F.relu(self.conv2(x)))
23        # Flatten the output for the fully connected layers
24        x = x.view(-1, 32 * 7 * 7)
25        # Apply dropout
26        x = self.dropout(x)
27        # Fully connected layer 1 with ReLU activation
28        x = F.relu(self.fc1(x))
29        # Output layer
30        x = self.fc2(x)
31
32    return x

```

Training

- In order to make use of GPUs and so speed up training, model was run in a Google Colab Notebook, Model was run for 10 epochs. Othe training details including hyper-parameters are as per the screenshot below:

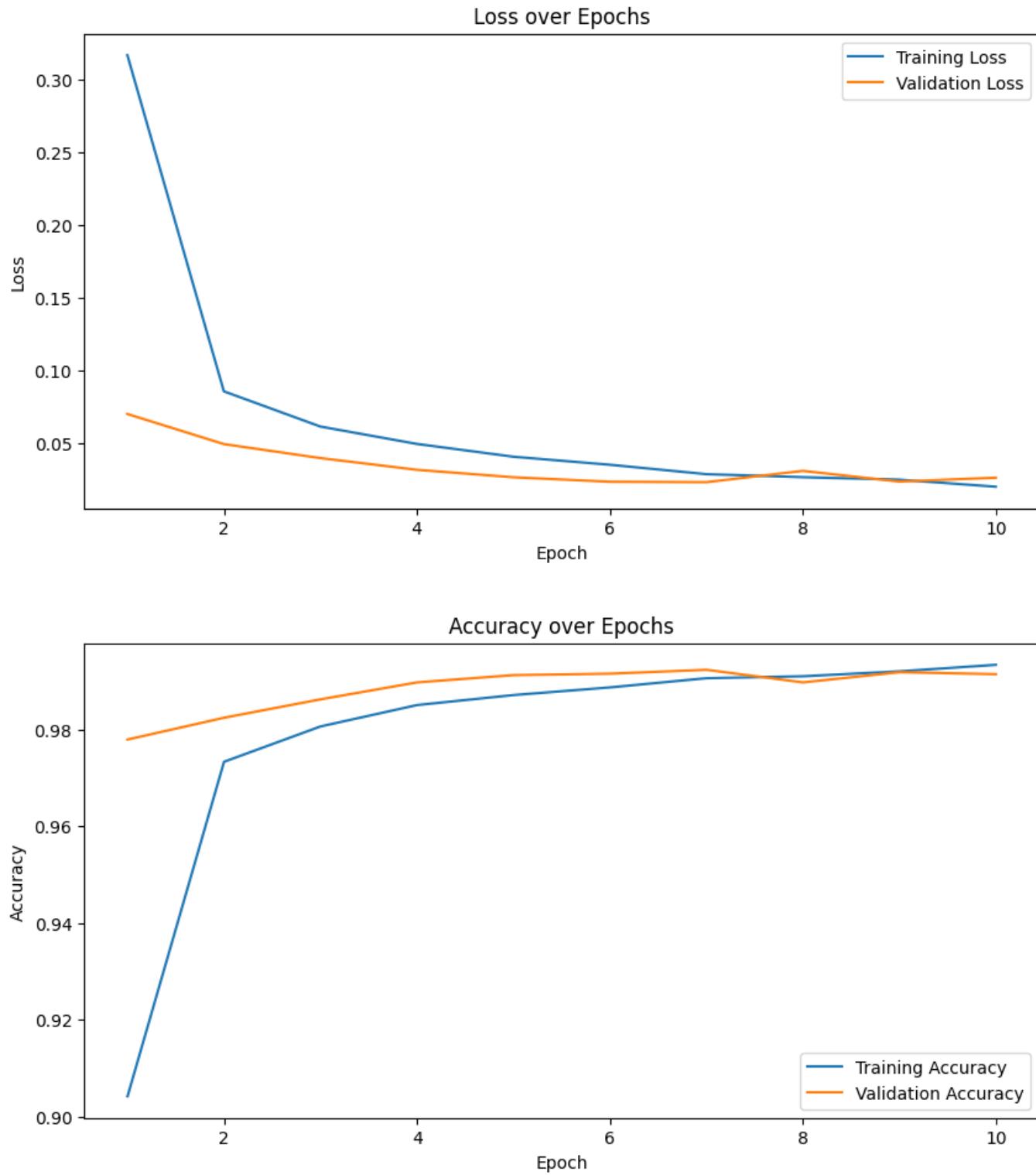
```

1 # Instantiate the model and move it to the device (CPU or GPU)
2 model = CNNModel().to(device)
3
4 # Define the loss function (Cross-Entropy Loss for multi-class classification)
5 criterion = nn.CrossEntropyLoss()
6
7 # Define the optimizer (Adam optimizer with a learning rate of 0.001)
8 optimizer = optim.Adam(model.parameters(), lr=0.001)
9
10 # Number of epochs to train
11 num_epochs = 10
12

```

Model Evaluation

Model accuracy of over 90% was achieved on the MNIST test dataset.



Inference

File

_MODEL WEIGHTS

This repo includes a pre-trained CNN for MNIST digit recognition:

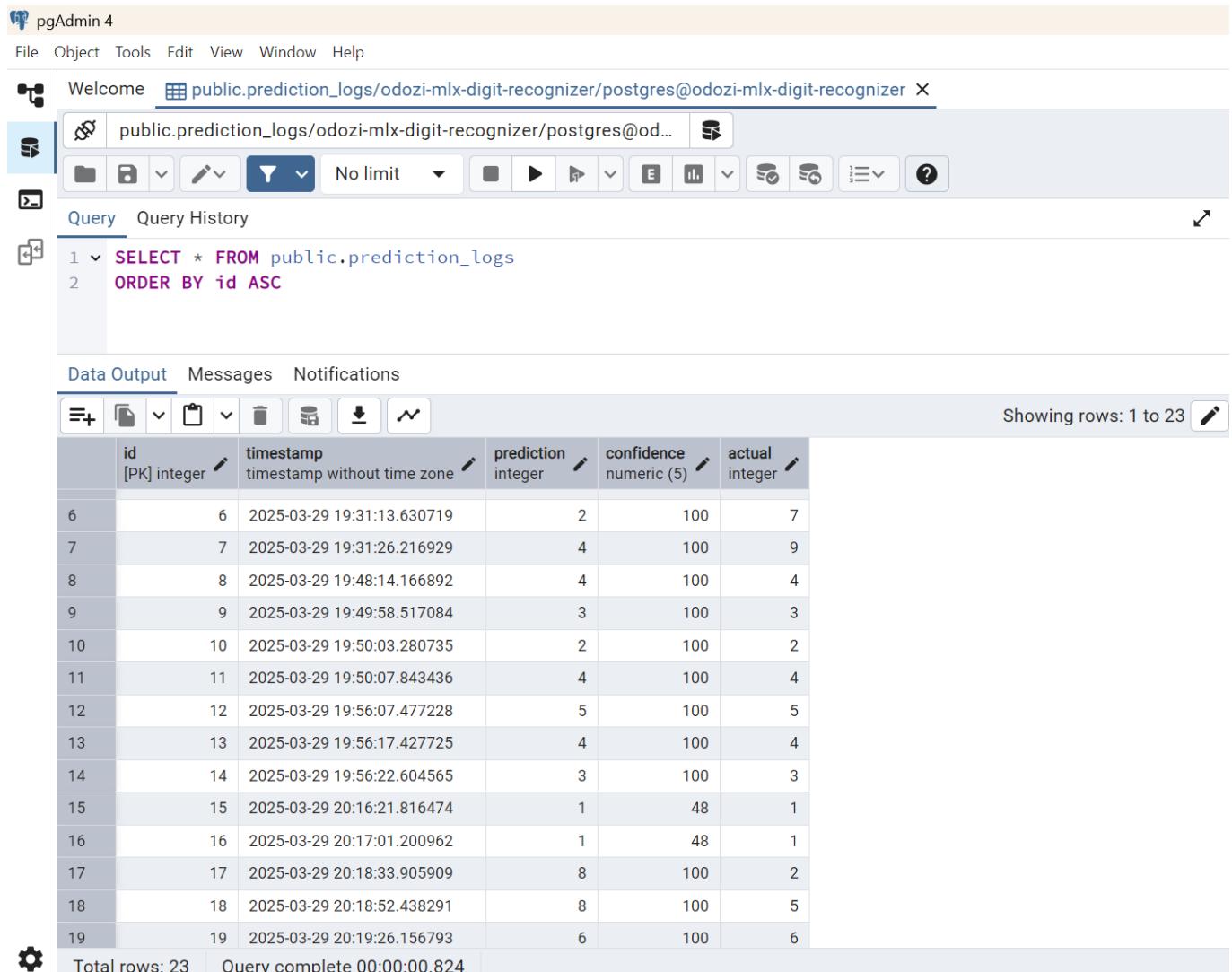
- `app/mnist_cnn.pth` — weights for a small CNN trained on the MNIST dataset

You can retrain your own using `CNNModelMNIST.py` or swap in a different `.pth`.

Deployment

Three Docker containers were used as per the project structure. One for the PyTorch Model and Streamlit App the second for the initialisation of the Postgres DB and Table and the 3rd for a PGAdmin tool to be used for viewing the predicton logs ona browser The Containers were built and hosted on Hetzner VPS Instance Git Hub actions were used to push any changes to the app folder automatically to the VPS using ssh login and there to re-build the container Docker-compose was used to automatically restart the App anytime the VPS was rebooted On first run, the `init.sql` script (in `db/init.sql`) sets up the PostgreSQL schema for logging predictions. Docker Compose handles this automatically.

The Postgres database table is shown below after over 20 attempts had been logged



The screenshot shows the pgAdmin 4 interface. The title bar says "pgAdmin 4". The menu bar includes File, Object, Tools, Edit, View, Window, Help. The main window shows a connection to "public.prediction_logs/odozi-mlx-digit-recognizer/postgres@odozi-mlx-digit-recognizer". The toolbar has various icons for database management. The left sidebar shows "Query History" and the current query is:

```
1 ✓ SELECT * FROM public.prediction_logs
2 ORDER BY id ASC
```

The bottom status bar shows "Showing rows: 1 to 23" and "Total rows: 23 Query complete 00:00:00.824".

	<code>id [PK] integer</code>	<code>timestamp timestamp without time zone</code>	<code>prediction integer</code>	<code>confidence numeric (5)</code>	<code>actual integer</code>
6	6	2025-03-29 19:31:13.630719	2	100	7
7	7	2025-03-29 19:31:26.216929	4	100	9
8	8	2025-03-29 19:48:14.166892	4	100	4
9	9	2025-03-29 19:49:58.517084	3	100	3
10	10	2025-03-29 19:50:03.280735	2	100	2
11	11	2025-03-29 19:50:07.843436	4	100	4
12	12	2025-03-29 19:56:07.477228	5	100	5
13	13	2025-03-29 19:56:17.427725	4	100	4
14	14	2025-03-29 19:56:22.604565	3	100	3
15	15	2025-03-29 20:16:21.816474	1	48	1
16	16	2025-03-29 20:17:01.200962	1	48	1
17	17	2025-03-29 20:18:33.905909	8	100	2
18	18	2025-03-29 20:18:52.438291	8	100	5
19	19	2025-03-29 20:19:26.156793	6	100	6

Ideas for future

- Trying out different models eg LeNet-5, Vision Transformers, Capsule Networks, Google Vision API, LLMs like ChatGPT
- Splitting out the model from the Streamlit app so that there are 4 containers instead of 3