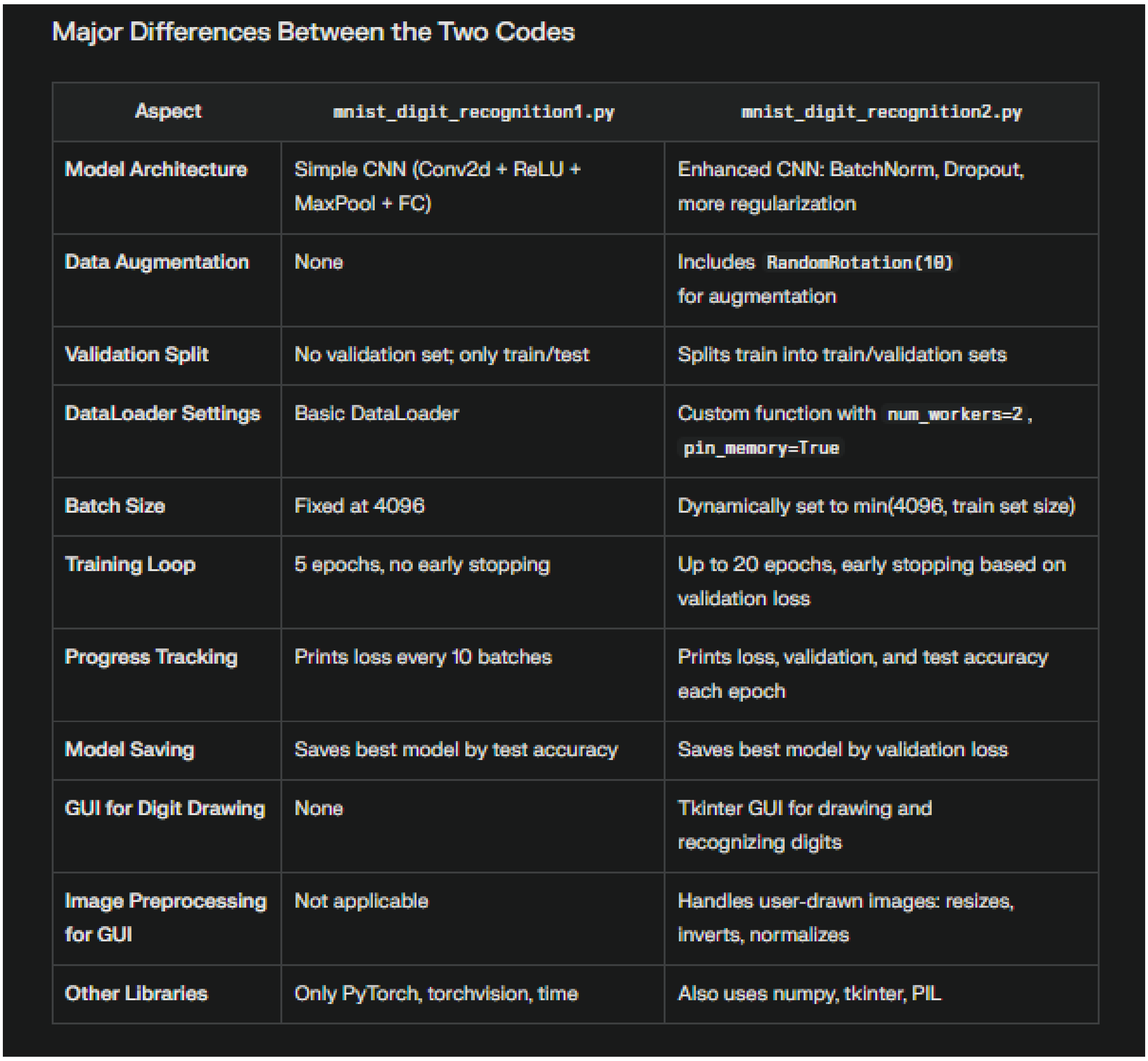

Major Differences Between the Two Codes

Aspect	`mnist_digit_recognition1.py`	`mnist_digit_recognition2.py`
Model Architecture	Simple CNN (Conv2d + ReLU + MaxPool + FC)	Enhanced CNN: BatchNorm, Dropout, more regularization
Data Augmentation	None	Includes `RandomRotation(10)` for augmentation
Validation Split	No validation set; only train/test	Splits train into train/validation sets
DataLoader Settings	Basic DataLoader	Custom function with `num_workers=2`, `pin_memory=True`
Batch Size	Fixed at 4096	Dynamically set to min(4096, train set size)
Training Loop	5 epochs, no early stopping	Up to 20 epochs, early stopping based on validation loss
Progress Tracking	Prints loss every 10 batches	Prints loss, validation, and test accuracy each epoch
Model Saving	Saves best model by test accuracy	Saves best model by validation loss
GUI for Digit Drawing	None	Tkinter GUI for drawing and recognizing digits
Image Preprocessing for GUI	Not applicable	Handles user-drawn images: resizes, inverts, normalizes
Other Libraries	Only PyTorch, torchvision, time	Also uses numpy, tkinter, PIL

Detailed Comparison

1. Model Architecture

- mnist_digit_recognition1.py uses a straightforward CNN:
    - Two convolutional layers, ReLU activations, MaxPool, two fully connected layers1.
- mnist_digit_recognition2.py introduces:
    - Batch normalization after each convolution
    - Dropout before the final layer for regularization
    - This makes the model more robust to overfitting and potentially improves generalization2.

2. Data Augmentation and Preprocessing

- The first script only normalizes the data1.
- The second script applies random rotations to training images, which helps the model generalize better by seeing slightly varied versions of digits2.

3. Data Splitting and Validation

- The first script uses the full MNIST training set for training, and the test set for evaluation1.
- The second script splits the training set into training and validation subsets (90%/10%) and uses the validation set for early stopping and model selection2.

4. DataLoader Optimization

- The first script uses basic DataLoader parameters1.
- The second script defines a custom DataLoader with num_workers=2 and pin_memory=True for faster data loading, especially on CUDA devices2.

5. Training and Evaluation Loop

- The first script runs for a fixed 5 epochs, always evaluating on the test set after each epoch, and saves the model if test accuracy improves1.
- The second script:
  - Trains for up to 20 epochs with early stopping if validation loss doesn't improve for 3 epochs
  - Tracks and prints train loss, validation loss/accuracy, and test accuracy each epoch
  - Saves the model when validation loss improves, not just based on test accuracy2.

6. GUI and User Interaction

- Only mnist_digit_recognition2.py includes a Tkinter GUI that allows users to draw digits and get real-time predictions using the trained model. It preprocesses the drawn image to match MNIST format and displays the predicted digit and confidence2.
- The first script has no GUI or interactive component1.

7. Additional Features

- The second script loads the best model after training for GUI use2.

- The first script simply prints the best test accuracy at the end1.

Summary

- **mnist_digit_recognition2.py is a more advanced, production-ready script**: It includes data augmentation, validation, early stopping, better model regularization, and a user-friendly GUI for digit recognition. It is more robust for both experimentation and demonstration.
- **mnist_digit_recognition1.py is a minimal, educational baseline**: It focuses on simplicity and speed, with no data augmentation, validation, or user interface, making it suitable for quick experiments or learning basics.