🧠 Handwritten Digit Recognition with C++ and SFML: A Reflection on Machine Learning

# 📌 Introduction

This project implements a handwritten digit recognition system using C++, the SFML library for GUI interaction, and a custom neural network built from scratch — no external ML frameworks. It allowed me to explore both machine learning theory and practical implementation, while working with my own hand-drawn data rather than a pre-packaged dataset like MNIST.

# 🔧 What I Did

* - Built an interactive GUI using SFML that allows the user to draw digits (0–9) on a canvas.
* - Implemented logic to label and save the drawings as 28x28 grayscale images.
* - Created a CSV dataset of labeled drawings, eventually building a small dataset of 200 samples (20 for each digit).
* - Built a feedforward neural network in C++ with:
* - 784 input nodes (28x28 grayscale)
* - 128 hidden nodes
* - 10 output nodes (digits 0–9)
* - Implemented training using simple backpropagation with mean squared error.
* - Saved the trained model weights to a file and used them to predict digits in real time within the GUI.

# 🧠 Machine Learning Details

* - The network was trained for 100 epochs with a learning rate of 0.01.
* - Each training sample was a flattened 28x28 pixel array, normalized between 0 and 1.
* - The output of the network was a vector of size 10, with the predicted digit corresponding to the index with the highest value.
* - Activation functions used:
* - Sigmoid for the output layer
* - ReLU for the hidden layer (with simple backward ReLU logic)
* - Loss function: simple sum-of-squares error.

# 🔍 Accuracy & Limitations

After training and testing with my small dataset (200 samples total), the prediction performance was inconsistent. Often it worked for clearly written digits, but it would fail on digits with slightly different styles or stroke patterns.

# ❌ Why Accuracy Was Low

* - Small dataset: Only 20 samples per class is far from enough for a generalizable model.
* - Data variability: My drawings were done by one person (myself), so the model may be overfitting to my own style.
* - No data augmentation: Real datasets use noise, rotation, shifts, etc. to expand variety.
* - Simple architecture: A single hidden layer is often insufficient for complex visual tasks like digit classification.
* - No cross-validation or test set: All evaluation was informal via live prediction.

# 📚 What I Learned

* - How to build a basic neural network from scratch, including matrix operations, forward pass, and backpropagation.
* - The importance of good data preprocessing, especially when dealing with pixel-based images.
* - How to use SFML for drawing and input capture, turning user actions into training samples.
* - The challenges of managing ML pipelines without libraries like TensorFlow or PyTorch.
* - The importance of good data and volume in training ML models.

# 🔁 What I Would Change (If I Had More Time)

* - Build a larger dataset with more diverse samples, ideally crowdsourced or generated with noise/rotation.
* - Switch from CSV to a more efficient format (e.g., raw binary or `.npz`).
* - Improve preprocessing:
* - Center the digit
* - Normalize line thickness or crop bounding box
* - Explore more advanced architectures:
* - Add a second hidden layer
* - Try convolutional layers (if implementing a basic CNN in C++)
* - Evaluate model performance using confusion matrices, accuracy reports, and test/train splits.
* - Potentially integrate a Python backend for training, while keeping the GUI in C++.

# ✅ Conclusion

This project was a great deep dive into both machine learning fundamentals and practical implementation in a low-level language. By building everything from scratch — from the neural net to the GUI — I developed a much deeper appreciation for what ML frameworks handle behind the scenes.

Though the final accuracy was limited, the experience helped me solidify key concepts like:

* - Forward/backward propagation
* - Activation functions
* - Normalization
* - Data representation and labeling

With better data and more experimentation, this project could evolve into a robust handwritten digit recognizer — and it was a fantastic learning experience regardless of accuracy.