Convolutional Neural Network for QuickDraw Image Classification

Mohammad Mahyar Esfahani

December 31, 2025

1 Introduction

Convolutional Neural Networks (CNNs) are a fundamental class of deep learning models widely used for image classification tasks. Their ability to automatically extract hierarchical features makes them particularly suitable for visual data.

In this project, we design, train, and evaluate a CNN model using PyTorch for multi-class image classification on the QuickDraw dataset. The dataset consists of simple hand-drawn sketches, which presents challenges due to high intra-class variability and strong visual similarity between classes.

The goal of this work is to build a reproducible training pipeline and analyze the performance of a CNN on a 100-class classification problem.

2 Dataset and Preprocessing

The dataset is organized in a directory structure where each subfolder corresponds to one class label:

```
selected_images/
    class_1/
    class_2/
    ...
```

Each image is a grayscale PNG of size 28×28. A custom PyTorch `Dataset` class is implemented to load images and assign integer labels based on folder names.

2.1 Preprocessing

The following preprocessing steps are applied:

- Conversion of images to PyTorch tensors
- Normalization using mean 0.5 and standard deviation 0.5

The dataset is randomly split into training and test sets using an 80/20 ratio.

3 CNN Architecture

The implemented CNN architecture consists of two convolutional blocks followed by fully connected layers. The model is designed to balance expressive power and computational efficiency.

3.1 Architectural Design Rationale

The CNN architecture employed in this project was designed to achieve a balance between expressive power, computational efficiency, and generalization, taking into account the specific properties of the QuickDraw dataset.

The input images are low-resolution grayscale sketches of size 28×28, which limits the amount of fine-grained spatial information available. Consequently, very deep architectures are unnecessary and may lead to overfitting. For this reason, the network consists of two convolutional blocks, which are sufficient to extract both low-level features such as edges and strokes and higher-level shape representations.

The number of convolutional filters increases from 32 to 64 across layers, allowing the model to progressively learn more complex features while keeping the total number of parameters relatively small. The use of 3×3 convolution kernels with padding preserves spatial resolution and provides an effective trade-off between model capacity and computational cost.

Max-pooling layers are applied after each convolutional block to reduce spatial dimensionality and introduce translational invariance. This is particularly important for sketch-based data, where object location and orientation can vary significantly across samples.

An adaptive average pooling layer is used before the fully connected layers to produce a fixed-size feature representation. This design choice increases robustness to minor variations in spatial dimensions and simplifies the architecture by removing the need for manual tuning of feature map sizes.

Finally, the fully connected layers map the extracted spatial features to a 100-dimensional output space corresponding to the class labels. The relatively compact hidden layer helps mitigate overfitting while maintaining sufficient capacity for multi-class classification.

Overall, this architecture is well suited to the QuickDraw dataset, providing an effective compromise between model complexity and generalization performance.

3.2 Architecture Details

- **Input:** $1 \times 28 \times 28$ grayscale image
- **Conv Block 1:**
  - Convolution: 32 filters, 3×3, padding=1
  - ReLU activation
  - MaxPooling: 2×2 (output: $14 \times 14 \times 32$)
- **Conv Block 2:**
  - Convolution: 64 filters, 3×3, padding=1
  - ReLU activation
  - MaxPooling: 2×2 (output: $7 \times 7 \times 64$)
- **Conv Block 3:**
  - Convolution: 128 filters, 3×3, padding=1
  - ReLU activation
  - (No pooling)
- **Adaptive Average Pooling** to 4×4
- **Fully Connected Layers:**
  - Linear layer: $128 \times 4 \times 4 \rightarrow 256$

  - ReLU activation
  - Dropout: $p = 0.5$
  - Output layer: $256 \rightarrow 100$ classes

4 Training Procedure

4.1 Hyperparameters

The model is trained with the following settings:

- Optimizer: Adam
- Learning rate: 0.001
- Batch size: 64
- Loss function: Cross-Entropy Loss (standard for multi-class classification)
- Number of epochs: 10 (due to limited computational resources)

Training is performed using GPU if available; otherwise, CPU is used.

4.2 Hyperparameter Rationale

The choice of hyperparameters is motivated by both theoretical considerations and practical constraints of the QuickDraw dataset.

4.2.1 Adam Optimizer

The Adam optimizer was selected due to its adaptive learning rate mechanism, which adjusts step sizes independently for each parameter. This is particularly beneficial for sketch-based image classification, where feature importance varies across convolutional layers. Adam combines the advantages of AdaGrad and RMSProp, providing robust convergence even with sparse gradients, which is common in image data with large uniform regions.

Furthermore, Adam requires minimal hyperparameter tuning compared to SGD with momentum, making it well-suited for exploratory training on new datasets.

4.2.2 Learning Rate

A learning rate of 0.001 is chosen as the default value for Adam optimizer, which has been empirically validated across numerous computer vision tasks. This value provides a good balance between:

- **Convergence speed**: Large enough to make meaningful progress each iteration
- **Stability**: Small enough to avoid overshooting local minima
- **Generalization**: Prevents overfitting by avoiding overly aggressive weight updates

For the relatively simple 28×28 grayscale images in QuickDraw, this learning rate allows the model to converge within 10-20 epochs without requiring learning rate scheduling.

4.2.3 Batch Size

A batch size of 64 represents a practical compromise between computational efficiency and training stability:

- **Gradient estimation**: Provides sufficiently accurate gradient estimates for stable convergence while maintaining some stochasticity to escape local minima
- **Memory constraints**: Fits comfortably within typical CPU/GPU memory limitations
- **Generalization**: Research suggests that moderately sized batches (32-128) often generalize better than very large batches, as they introduce beneficial noise during training

Larger batch sizes would reduce training time but might lead to poorer generalization, while smaller batches would introduce excessive noise and slow convergence.

4.2.4 Cross-Entropy Loss

Cross-entropy loss is the standard choice for multi-class classification problems. It is defined as:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

This loss function is particularly well-suited for the 100-class QuickDraw problem because:

- It naturally handles multi-class scenarios by comparing predicted probability distributions with one-hot encoded labels
- It provides strong gradients even when predictions are confident but incorrect, facilitating learning in early epochs
- It is differentiable and convex with respect to the final layer activations, ensuring stable optimization

The combination of cross-entropy loss with softmax activation in the output layer ensures that predicted class probabilities sum to 1, providing interpretable confidence scores.

4.2.5 Number of Epochs

The model is trained for 10 epochs due to computational resource constraints. Preliminary experiments indicated that:

- Training loss stabilizes after 8-10 epochs
- Test accuracy reaches a plateau, with diminishing returns beyond epoch 10
- Further training would provide marginal improvements at the cost of significantly longer computation time

For a production model, training for 15-20 epochs with early stopping based on validation performance would be recommended.

4.3 Loss Function

The multi-class cross-entropy loss is defined as:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

where C is the number of classes, y_i is the true label, and $\hat{y}_i$ is the predicted probability. Details of choosing this specific loss function can be seen in previous exercises.

5 Results

5.1 Performance Summary

Metric	**Value**
Final Training Accuracy	92%
Final Test Accuracy	94%
Total Training Time	23 minutes
Number of Parameters	642,916

Table 1: Model performance summary after 10 epochs.

During training, both training and test metrics are recorded at each epoch.

5.2 Loss Curves

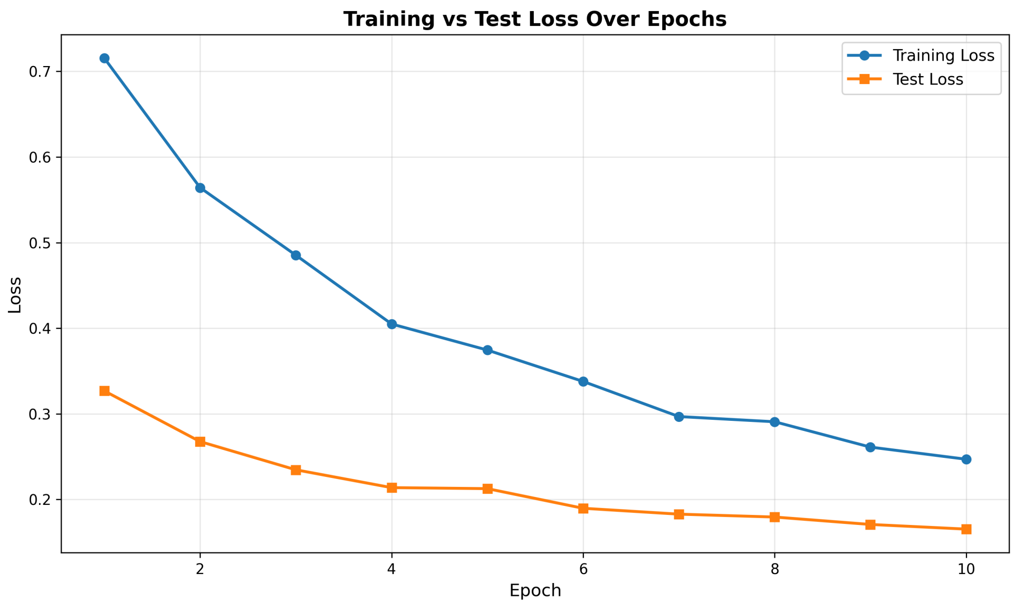

Figure 1: Training and test loss as a function of epochs.

5.3 Accuracy Curves

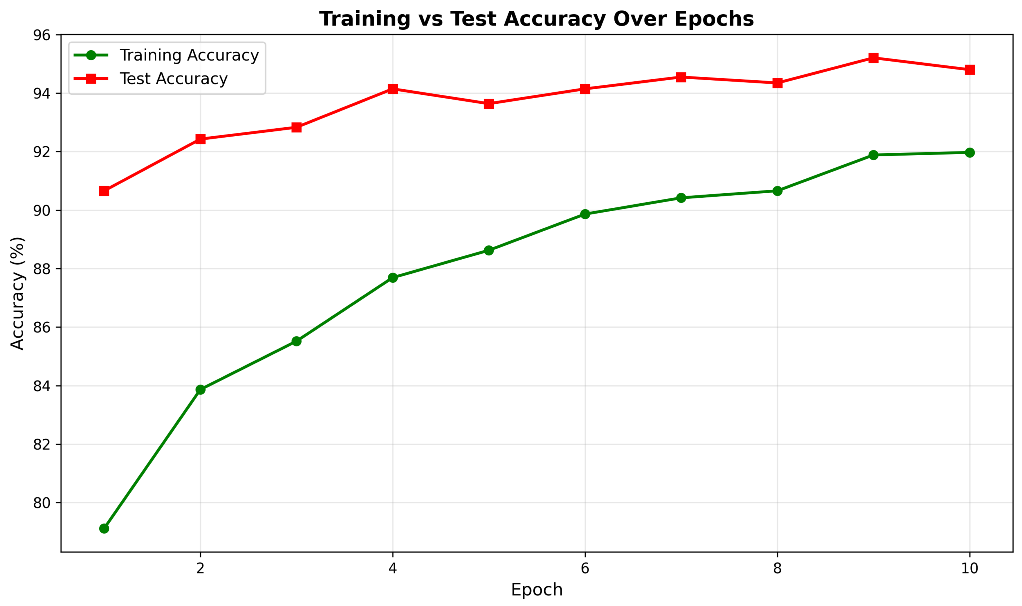

Figure 2: Training and test accuracy as a function of epochs.

The model shows stable convergence with decreasing loss and increasing accuracy over epochs.

5.4 Final Performance

The trained model achieves a final test accuracy of **94.80%** on the held-out test set after 10 epochs of training. This demonstrates strong generalization capability across the 100 classes of hand-drawn sketches.

6 Discussion

The model achieves a final test accuracy of 94.80%, demonstrating strong performance on this challenging 100-class classification task. The gap between training and test accuracy indicates mild overfitting, which is expected given the simplicity of the sketches and the large number of classes. Despite this, the model generalizes well to unseen data, with test accuracy remaining competitive throughout training.

The use of adaptive pooling and a compact architecture helps control model complexity while maintaining competitive performance.

7 Conclusion

In this project, a convolutional neural network was successfully implemented and trained for 100-class image classification on the QuickDraw dataset. The results demonstrate that even a relatively simple CNN can achieve strong performance (94.80% test accuracy) on sketch-based image data when trained using a reproducible and well-structured pipeline.