Handwritten Character Recognition Using a Custom Neural Network Report.

Student : Hala Khalifeh

ID: 12112858

This pdf file include:

1. Introduction: Problem statement and objectives.
2. Methodology: Data preprocessing, model architecture, and training process.
3. Results: Performance metrics, visualizations, and analysis.
4. Discussion: Key findings, challenges, and lessons learned.
5. Conclusion: Summary of results and potential improvements

1. Introduction

Problem Statement

Handwritten character recognition is a fundamental problem in the field of machine learning and computer vision. The goal is to develop a model that can accurately classify handwritten letters from the EMNIST dataset, which contains both uppercase and lowercase letters. This task is challenging due to the variability in handwriting styles, similarities between certain letters (e.g., 'c' and 'e'), and the need for the model to generalize well to unseen data.

Objectives

1. Build, train, and evaluate a fully connected neural network (FCNN) for handwritten letter recognition.
2. Explore the EMNIST dataset, preprocess the data, and implement a neural network from scratch.
3. Analyze the model's performance, visualize misclassifications, and interpret what the model has learned.
4. Experiment with different hyperparameters and architectures to improve the model's accuracy.
5. Compare the classification accuracy between uppercase and lowercase letters.

2. Methodology

Data Preprocessing

1. The EMNIST Letters dataset was loaded using PyTorch and converted to a NumPy format compatible with TensorFlow/Keras.
2. Images were normalized to the range [0, 1] and reshaped to 28x28 pixels.
3. Labels were converted to one-hot encoding for multi-class classification.

4. Sample images were displayed to verify the dataset's correctness.

Model Architecture

A fully connected neural network (FCNN) was implemented with the following architecture:

1. **Input Layer**: Flattened 28x28 image (784 neurons).
2. **Hidden Layers**: Two hidden layers with 128 and 64 neurons, respectively, using ReLU activation.
3. **Output Layer**: 26 neurons with softmax activation (one for each letter).

Training Process

1. The model was compiled using the Adam optimizer and categorical cross-entropy loss.
2. Early stopping and learning rate scheduling were implemented to improve training efficiency.
3. The model was trained using 5-fold cross-validation, and the average validation accuracy was reported.
4. After cross-validation, the model was retrained on the full training set and evaluated on the test set.

3. Results

Performance Metrics

- **Cross-Validation Accuracy**: The model achieved an average validation accuracy of **83.65%** across 5 folds.
- **Test Accuracy**: The final model achieved a test accuracy of **83.01%**.
- **Final Validation Accuracy**: A bar plot comparing the final validation accuracy for each batch size.

Visualizations

- **Training and Validation Curves**: The training and validation accuracy/loss curves were plotted to analyze the model's learning behavior.
- **Validation Accuracy Curves**: Plots showing the validation accuracy for batch sizes 32, 64, and 128.
- **Validation Loss Curves**: Plots showing the validation loss for batch sizes 32, 64, and 128.
- **Confusion Matrix**: The confusion matrix was visualized to identify the most misclassified letters.
- **Misclassified Images**: A sample of misclassified images was displayed, showing the true and predicted labels.
- **Hidden Layer Weights**: The weights of the first hidden layer were visualized to interpret what the model learned.

Analysis

- **Most Misclassified Letter**: The letter **'G'** was the most misclassified, with **335** misclassifications.

4. Discussion

Batch Size 32: Achieved the highest validation accuracy, making it the best choice for this dataset.
Batch Size 64: A good compromise between stability and convergence speed.
Batch Size 128: Trained faster but converged to a lower accuracy, likely due to less noise in the gradient updates.

Key Findings

1. The model performed well on uppercase letters but struggled with lowercase letters, likely due to their subtle differences.
2. The most misclassified letters were those with similar shapes (e.g., 'a' and 'e', 'n' and 'm').

3. Visualizing the hidden layer weights revealed that the model learned to detect edges, curves, and other patterns important for letter recognition.
4. Best Batch Size: Batch size 32 performed the best, achieving the highest validation accuracy.
5. Training Speed: Larger batch sizes (e.g., 128) trained faster per epoch but converged to a lower accuracy.
6. Stability: Smaller batch sizes (e.g., 32) showed more fluctuations in validation accuracy and loss, but ultimately achieved better performance.

Challenges

- The dataset's imbalance (some letters had fewer samples) may have affected the model's performance.
- Overfitting was observed during training, which was mitigated using early stopping.

Lessons Learned

- Data preprocessing and normalization are critical for model performance.
- Cross-validation provides a robust estimate of the model's generalization ability.
- Visualizing misclassified images and hidden layer weights helps interpret the model's behavior.

5. Conclusion

Summary of Results

- The custom neural network achieved a test accuracy of **83%** on the EMNIST Letters dataset.
- The most misclassified letter was **'G'**, highlighting the need for further model improvements.

Potential Improvements

1. Experiment with more complex architectures, such as convolutional neural networks (CNNs), to improve accuracy.
2. Address dataset imbalance using techniques like data augmentation or oversampling.
3. Fine-tune hyperparameters (e.g., learning rate, batch size) to optimize performance.
4. Explore transfer learning using pre-trained models for feature extraction.