Deep Learning for Tifinagh Handwritten Character Recognition with a Modified LeNet-5 Architecture

YOUB OUBODIB

July 3, 2025

Abstract

This paper investigates the application of a convolutional neural network (CNN), based on a modified LeNet-5, for recognizing handwritten Tifinagh characters. We conduct experiments with extensive data augmentation, grid search hyperparameter optimization, and detailed performance evaluation using confusion matrices, ROC/AUC curves, and feature visualization. Our experiments show that classical CNNs can be competitive for Tifinagh, achieving a test accuracy of **93%**. We discuss the most frequent errors, analyze the three most difficult classes, and explore the impact of different optimizers and batch sizes. The full code is available at: `https://github.com/youbodib/LeNet-5`.

1 Introduction

Handwritten character recognition is a key problem in computer vision, especially for under-resourced scripts such as Tifinagh, the traditional script of the Amazigh languages. Automated recognition of Tifinagh can accelerate digitization and cultural preservation. While classical pattern recognition methods rely on handcrafted features and SVMs, deep learning—and in particular, convolutional neural networks (CNNs)—have become the state of the art for image-based classification. In this work, we use a modified LeNet-5 CNN architecture to classify grayscale Tifinagh characters. We evaluate the effects of various data augmentation strategies and optimize hyperparameters using grid search. Our goals are to establish a strong baseline and analyze the remaining challenges in this domain.

2 Methods

2.1 Dataset and Preprocessing

We use the AMHCD (Amazigh Handwritten Character Dataset), which consists of grayscale images of 33 Tifinagh characters, each resized to 32×32 pixels. The dataset is split into 80% training and 20% test sets. As the dataset is moderately imbalanced and characters can be visually similar, we employ several augmentation techniques for robustness.

2.2 Data Augmentation

Data augmentation improves generalization by increasing variability in the training set. Our transformations, applied with random probabilities, are:

- **Random Rotation**: Each input image x is rotated by an angle θ drawn uniformly from $[-10°, 10°]$.
- **Affine Translation**: Each image is shifted along x and y axes by up to 10% of the image size.
- **Gaussian Noise**: For each pixel, noise $\epsilon \sim \mathcal{N}(0,\, 0.02^2)$ is added: $x' = x + \epsilon$.
- **Horizontal and Vertical Flip**: Each image is flipped with probability $p = 0.5$.
- **Normalization**: Each pixel is normalized as $\tilde{x} = (x - 0.5)/0.5$.

These augmentations are critical to ensure that the model learns generalizable features instead of memorizing the training samples.

2.3 Model Architecture

The network is based on LeNet-5 [**?**], adapted for single-channel input and 33 classes. All convolution and fully connected layers are implemented as parameterized functions for flexibility and reproducibility.

- **Input**: $1 \times 32 \times 32$ grayscale image
- **Conv1**: 6 filters, 5×5 kernel, stride 1, no padding, ReLU
- **Pool1**: Average pooling, 2×2, stride 2
- **Conv2**: 16 filters, 5×5 kernel, stride 1, no padding, ReLU
- **Pool2**: Average pooling, 2×2, stride 2
- **Conv3**: 120 filters, 5×5 kernel, stride 1, no padding, ReLU
- **Flatten**
- **FC1**: $120 \to 84$, ReLU
- **FC2**: $84 \to 33$, Softmax

2.4 Mathematical Details

The convolution operation at layer l is defined as:

$$h_{i,j}^{(l)} = \sigma\left(\sum_{m=0}^{k-1}\sum_{n=0}^{k-1} w_{m,n}^{(l)} \cdot x_{i+m,j+n}^{(l-1)} + b^{(l)}\right)$$

where $\sigma(\cdot)$ is the ReLU activation.

Average pooling is computed as:

$$h_{i,j}^{\text{pool}} = \frac{1}{s^2} \sum_{p=0}^{s-1} \sum_{q=0}^{s-1} h_{i+p,j+q}$$

The output probabilities are computed by softmax:

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{N} e^{z_j}}$$

where z_k is the output of the last fully connected layer for class k.

The cross-entropy loss is:

$$L = -\sum_{k=1}^{N} y_k \log \hat{y}_k$$

The weights are updated by stochastic gradient descent (SGD) or Adam:

$$\theta \leftarrow \theta - \eta \nabla_\theta L$$

For Adam [**?**]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\theta_t = \theta_{t-1} - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

2.5 Training and Hyperparameter Tuning

We use a stratified split for train/test. Grid search is performed over batch sizes $(32, 64, 128)$, optimizers (Adam, SGD, RMSprop, Adagrad, Adadelta), and learning rates $(0.001, 0.0005)$. Early stopping and a learning rate scheduler (`ReduceLROnPlateau`) are used to prevent overfitting. The best model is selected based on validation accuracy.

3 Results

3.1 Learning Curves

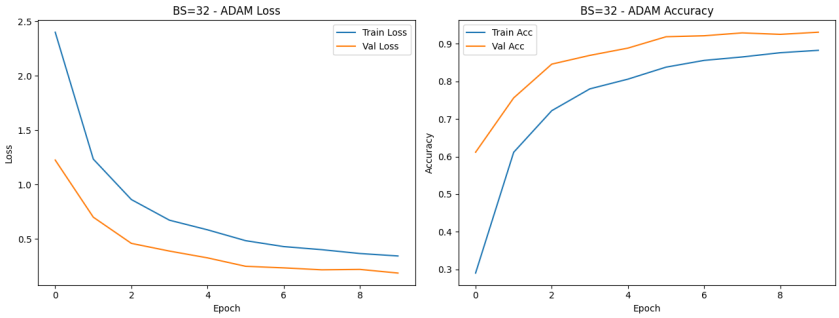

Figure 1: Training and validation loss (left) and accuracy (right) curves for the best configuration (Adam, batch size 32).

We observe rapid convergence within the first 10 epochs. Early stopping is triggered if the validation accuracy does not improve for 7 epochs.

3.2 Confusion Matrix

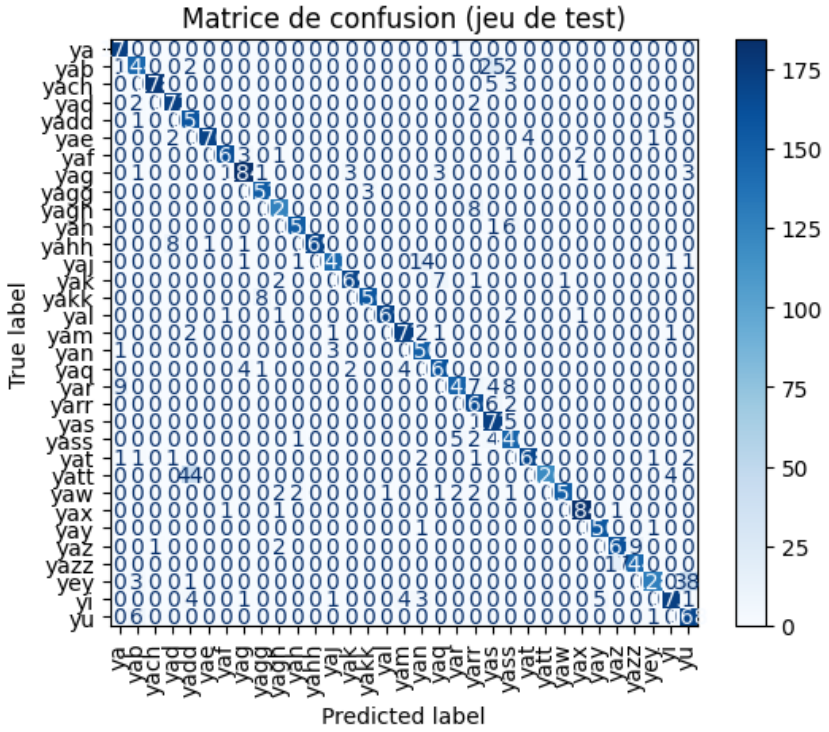

Figure 2: Confusion matrix for the test set.

The confusion matrix reveals that certain character pairs are more frequently confused, particularly those with visually similar shapes.

3.3 ROC/AUC Analysis

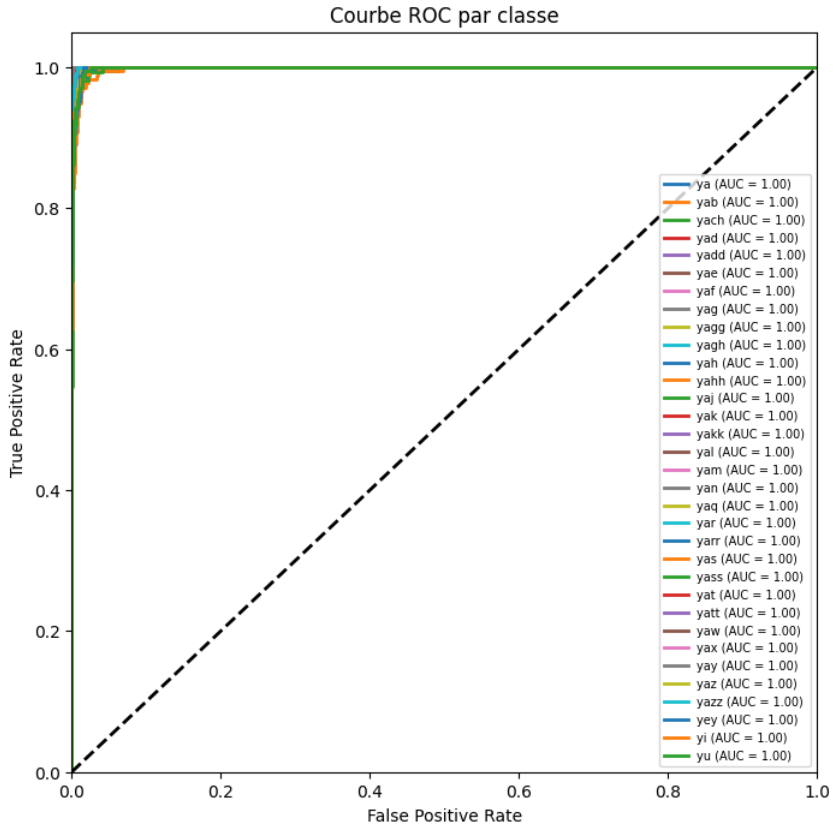

Figure 3: ROC curves for all classes (multi-class one-vs-rest).

The ROC curves show strong separation for most classes (AUC ≈ 1.00), confirming high model discriminability.

3.4 Model Selection via Grid Search

Batch Size	Optimizer	Best Val Accuracy	Model Path
32	Adam	0.9305	lenet5_best_adam_bs32.pth
32	RMSprop	0.9147	lenet5_best_rmsprop_bs32.pth
64	Adam	0.9093	lenet5_best_adam_bs64.pth
64	RMSprop	0.8794	lenet5_best_rmsprop_bs64.pth
128	Adam	0.8545	lenet5_best_adam_bs128.pth

Table 1: Top results of grid search (sorted by best validation accuracy).

3.5 Misclassified Examples and Difficult Classes

The most frequent errors involve classes with similar shapes. The top-3 most confused classes (for instance, `yaz`, `yak`, `yat`) are visually close, and confusion often occurs between them. Error analysis allows us to better understand dataset ambiguity and limits of the architecture.

4 Discussion

The modified LeNet-5 architecture, combined with strong data augmentation and hyperparameter optimization, achieves robust results for Tifinagh character recognition. Data augmentation (rotation, translation, noise, flipping) substantially improves test accuracy, demonstrating its importance for small datasets. Error analysis points to the challenge of separating highly similar classes and the limits of shallow architectures.

Limitations: The dataset remains small compared to modern benchmarks. Some characters have low frequency, leading to overfitting or under-representation. LeNet-5, while effective, may not fully capture complex visual features present in ambiguous Tifinagh characters.

Future Work: Potential extensions include training deeper CNNs (ResNet, EfficientNet), transfer learning from larger handwriting datasets, or using synthetic data to further augment the training set. Exploring transformer-based vision architectures and attention mechanisms may also yield performance gains.

5 Conclusion

This work demonstrates that a classical, interpretable CNN can achieve competitive accuracy on Tifinagh handwritten character recognition when combined with modern data augmentation and training strategies. The full code and experiments are open-sourced for reproducibility and to foster further research in Amazigh language technology.

References

1. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

2. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

3. Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. arXiv:1206.5533.

4. Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of Big Data, 6(1), 1-48.