Blood Cell Classification Using Deep Learning: MLNN and CNN Architectures

Miguel Cabral Pinto & Tiago Silva

University of Coimbra

Abstract. Blood cell classification is a critical task in medical diagnosis and hematological analysis. The present work conducts a comparative study between two deep learning architectures for blood cell classification supported by the BloodMNIST dataset. We implement and evaluate two neural network architectures: a Convolutional Neural Network (CNN) and a Multi-Layer Neural Network (MLNN). Besides implementation details we created an evaluation framework composed by three dimensions: quality (accuracy, precision, recall, F1-score), efficacy (sensitivity, specificity), and efficiency (training time, inference time). Experimental results support the theoretical superiority of CNN architectures for image-based classification tasks, showing significantly better performance than traditional fully-connected networks although the imbalanced nature of the dataset poses challenges for both architectures.

Keywords: Blood Cell Classification · Deep Learning · Multi-Layer Neural Networks · Convolutional Neural Networks · Medical Imaging · BloodMNIST

1 Methodology

1.1 Dataset

We utilize the BloodMNIST dataset from the MedMNIST collection. The dataset characteristics are:

- **Image Size**: 28×28 pixels, RGB (3 channels)
- **Classes**: 8 blood cell types
- **Training Set**: 11,959 images
- **Validation Set**: 1,712 images
- **Test Set**: 3,421 images
- **Total**: 17,092 images

Because the dataset is already preprocessed and normalized, we focus on the model architectures and training procedures rather than data augmentation techniques. As expected in medical datasets, class imbalance exists, with some cell types being underrepresented. This is addressed in the evaluation phase using metrics sensitive to class distribution to ensure a more robust assessment of model performance.

1.2 Model Architectures

Convolutional Neural Network (CNN) Our CNN architecture consists of three convolutional blocks followed by fully-connected layers. The design follows the principle of progressively increasing feature channels while reducing spatial dimensions:

Algorithm 1 CNN Architecture

Input: $x \in \mathbb{R}^{B\times 3\times 28\times 28}$
Conv₁: Conv2d($3 \to 32$, kernel=3, padding=1) + ReLU + MaxPool(2)
  Output: $\mathbb{R}^{B\times 32\times 14\times 14}$
Conv₂: Conv2d($32 \to 64$, kernel=3, padding=1) + ReLU + MaxPool(2)
  Output: $\mathbb{R}^{B\times 64\times 7\times 7}$
Conv₃: Conv2d($64 \to 128$, kernel=3, padding=1) + ReLU + MaxPool(2)
  Output: $\mathbb{R}^{B\times 128\times 3\times 3}$
Flatten: $\mathbb{R}^{B\times 1152}$
FC₁: Linear($1152 \to 256$) + ReLU
FC₂: Linear($256 \to 128$) + ReLU
FC₃: Linear($128 \to 8$)
Output: Logits $\in \mathbb{R}^{B\times 8}$

The architecture has 422,344 total parameters, and its rationale is as follows:

- **Progressive channel expansion** (32→64→128): Captures increasingly complex features
- **Spatial dimension reduction** (28→14→7→3): Achieved through max-pooling with stride 2
- **Unitary padding**: Preserves spatial dimensions within each convolutional layer
- **ReLU activation**: Introduces non-linearity and mitigates vanishing gradients
- **Multi-stage FC layers**: Provides smooth transition from feature maps to class predictions

The number of convolutional layers resulted from balancing model complexity and computational efficiency. Given the input, we recognized that three convolutional layers would provide enough feature extraction while not over-reducing the spatial dimensions. The channel progression and kernel sizes were selected based on established (state-of-the-art) CNN design principles proven and discussed in theoretical classes. Additionally, the fully-connected layers were sized to gradually reduce the feature representation to the final class logits (preventing abrupt bottlenecks). Though the design could certainly be further optimized through hyperparameter tuning, this architecture serves as a solid baseline (experimentally validated in the results section).

Multi-Layer Neural Network (MLNN) The MLNN architecture uses only fully-connected layers:

Algorithm 2 MLNN Architecture

1: **Input:** $x \in \mathbb{R}^{B\times 3\times 28\times 28}$
2: **Flatten:** $x \in \mathbb{R}^{B\times 2352}$ (where $2352 = 3 \times 28 \times 28$)
3: **FC_1:** Linear($2352 \to h_1$) + ReLU
4: **FC_2-FC_n:** Linear($h_{i-1} \to h_i$) + ReLU (for n hidden layers)
5: **Output Layer:** Linear($h_n \to 8$)
6: **Output:** Logits $\in \mathbb{R}^{B\times 8}$

The size of the hidden layers in the network is calculated using the average of the input and output dimensions:

$$h = \frac{n_{input} + n_{classes}}{2} = \frac{2352 + 8}{2} = 1180 \tag{1}$$

This heuristic provides reasonable capacity without excessive parameters. For a single hidden layer configuration, it results in 1,180 hidden units, creating a bottleneck architecture ($2352 \to 1180 \to 8$) that forces the network to learn compressed representations. While this approach can work for simple classification tasks, it fundamentally treats each pixel as an independent feature, ignoring the 2D spatial relationships that characterize cell morphology.
The MLNN architecture serves primarily as a pedagogical baseline to demonstrate why CNNs are preferred for image classification. The lack of translation invariance, the enormous first-layer parameter count (2.77M parameters just in the first layer), and the immediate loss of spatial structure highlight the architectural disadvantages of this approach when compared to convolutional ones. As such, this work validates theoretical understanding through empirical evidence.

1.3 Training Configuration: MLNN

Table 1: Experimental Hyperparameters

Parameter	**MLNN-Test1**	**MLNN-Test2**
Learning Rate	0.001	0.01
Optimizer	Adam	SGD w/ Momentum (0.9)
Loss Function	CrossEntropyLoss	CrossEntropyLoss
Epochs	15	15
Hidden Layers	1	1
Hidden Size	1180	1180
Activation Function	ReLU	ReLU

Table 1 presents the complete hyperparameter configuration used in our experiments, chosen to both experiment with different training dynamics and provide a solid baseline for comparison with the CNN architecture.

Here are the main reasons behind our choices:

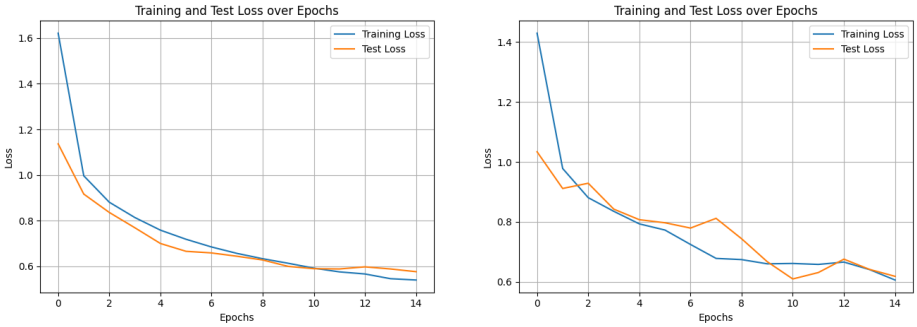
- CrossEntropyLoss was chosen as the standard loss function for multi-class classification, combining LogSoftmax and NLLLoss for numerical stability. This is particularly important given the class imbalanced nature of the dataset, CrossEntropyLoss poses a powerful approach to handle such scenarios by adjusting class influence through customized weighting if necessary.
- The state of the art Adam optimizer handles the varying gradient magnitudes across different network depths was selected due to its innovative adaptive learning rate mechanism, combining the benefits of RMSProp and momentum-based SGD. To compare with the forementioned momentum-based SGD optimizer, we also implemented the same MLNN using this optimizer (with momentum set to 0.9) and a higher learning rate (0.01 because in contrast to Adam, SGD lacks adaptive learning rates, therefore, a higher initial learning rate is necessary to achieve comparable performance).
- The 0.001 learning rate represents the standard default for Adam in computer vision tasks. Preliminary experiments (not detailed here) validated this choice: higher rates (0.01) caused training instability, while lower rates (0.0001) converged too slowly. The selected rate achieves a smooth loss decrease ($1.58 \rightarrow 0.313$) without oscillations, with test performance plateauing around epoch 12, indicating appropriate convergence behavior.
- The 15-epoch training duration was determined empirically.
- He initialization was used throughout, as it is specifically designed for ReLU activations and prevents vanishing/exploding gradients by maintaining variance across layers.

2 Results

In this section we first present and analyze the results obtained with each type of deep learning architecture: MLNN and CNN. Then we proceed to a comparative analysis between both.

2.1 MLNN

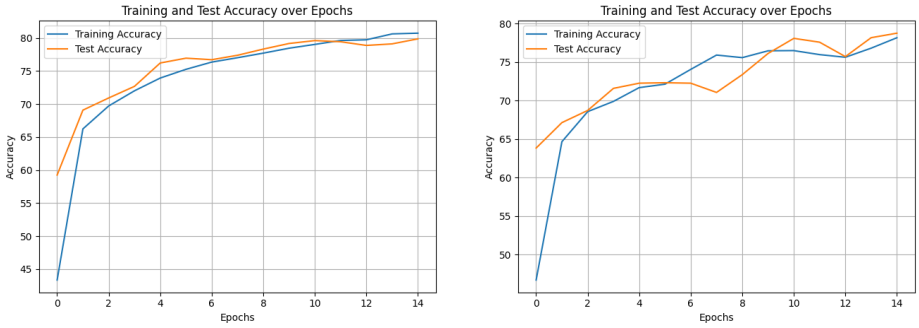
Two MLNN configurations were tested, differing in optimizer and learning rate (as detailed in Table 1). Let's analyze the results and make some observations.



(a) (Adam, lr=0.001) (b) (SGD+Momentum, lr=0.01)

Fig. 1: Loss curves comparison between MLNN configurations

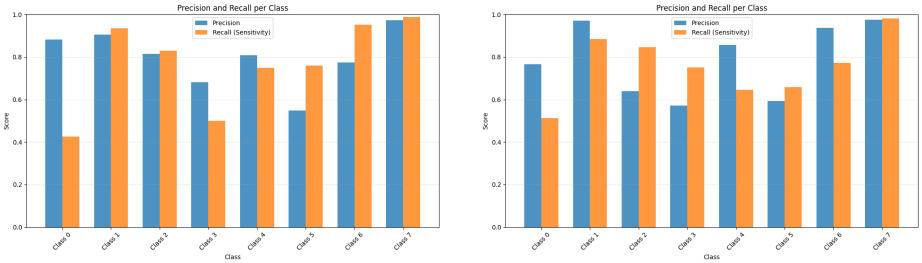
Looking at the loss curves in Figure 1, we can see that both configurations show a decreasing trend, indicating that the models are learning. However, the Adam optimizer (Figure 1a) demonstrates a more stable and consistent decrease in loss compared to the SGD with Momentum (Figure 1b), which exhibits a more erratic pattern, oscillating significantly during training. This suggests that Adam is better suited because not only does it adapt the learning rate for each parameter, but it also combines the benefits of both momentum and RMSProp, leading to more efficient convergence. This is a relatively simple task, hence both optimizers manage to reduce the loss. If this were a more complex task, the differences would be more pronounced.



(a) (Adam, lr=0.001) (b) (SGD+Momentum, lr=0.01)

Fig. 2: Accuracy curves comparison between MLNN configurations

Similarly, the accuracy curves in Figure 2 show that the Adam optimizer achieves higher accuracy more quickly than SGD with Momentum. The Adam configuration reaches approximately 85% accuracy by epoch 10, while the SGD configuration lags behind, only reaching around 75% accuracy by the same epoch. This further supports the conclusion that Adam's adaptive learning rates and momentum mechanisms provide a significant advantage in training efficiency.



(a) MLNN-Test1 (b) MLNN-Test2

Fig. 3: Precision and Recall comparison across classes

The bar charts in Figure 3 illustrate the precision and recall for each class in both MLNN configurations. We observe that the Adam optimizer (Figure 3a) generally outperforms the SGD with Momentum (Figure 3b) across most classes. Notably, classes with fewer samples (such as class 0) exhibit lower precision and recall in both configurations, highlighting the challenges posed by class imbalance. Despite in most classes the recall and precision values are relatively high, indicating that the models thresholds are effective at identifying the majority of

instances correctly, there is still room for improvement, especially for underrepresented classes.

(a) MLNN-Test1 (b) MLNN-Test2

Fig. 4: Confusion matrices for both MLNN configurations

The confusion matrix in Figure 4b shows that the momentum-based SGD optimizer is a bit less stable in comparison to Adam because, for exmaple, it tends to classify more samples as class 3. This could be due to the optimizer's inability to adapt learning rates for individual parameters, leading to suboptimal updates. In contrast, the Adam optimizer (Figure 4a) demonstrates a more balanced classification across classes, with fewer misclassifications. This further emphasizes Adam's effectiveness across several metrics.

The F1 score, is around 0.80 for both configurations, with Adam slightly outperforming SGD with Momentum. This indicates that while both models are effective, and for simple tasks like this one the differences are not very pronounced, Adam provides a marginal advantage in balancing precision and recall (the thresholds are better optimized).

2.2 CNN

2.3 Comparative Analysis

References

1. Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S., Dean, J.: A guide to deep learning in healthcare. Nature medicine 25(1), 24–29 (2019)
2. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature 521(7553), 436–444 (2015)
3. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25 (2012)
4. Habibzadeh, M., Krzyżak, A., Fevens, T.: Comparative study of shape, intensity and texture features and support vector machine for white blood cell classification. Journal of Theoretical and Applied Computer Science 5(1), 20–35 (2011)
5. Acevedo, A., Merino, A., Alférez, S., Molina, Á., Boldú, L., Rodellar, J.: A dataset of microscopic peripheral blood cell images for development of automatic recognition systems. Data in brief 30, 105474 (2020)
6. Yang, J., Shi, R., Ni, B.: MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis. IEEE 18th International Symposium on Biomedical Imaging (ISBI), 191–195 (2021)
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)