Milestone 2 Report: Skin Lesion Diagnosis with MLP & CNN

Leonardo Matteo Bolognese Ilias Bouraoui Yann Gaspoz

June 1, 2025

1 Introduction

Early detection of malignant skin lesions is critical for patient survival. In this project, we address a 7-class classification task on the DermaMNIST dataset (7007 train / 2005 test images, $28 \times 28 \times 3$), implementing two deep-learning models in PyTorch: a multi-layer perceptron (MLP) and a convolutional neural network (CNN). Alongside standard training and evaluation, we contribute several bonus features, including data augmentation, GPU/multi-CPU optimisation, and hyperparameter grid search.

2 Data Preparation Pipeline

2.1 Data Loading

We use the `load_data()` function to load the DermaMNIST dataset via the `medmnist` API. It returns NumPy arrays of images and labels for both training and test sets, each resized to 28×28 with 3 color channels.

2.2 Data Augmentation (*bonus*)

When the `--augment_data` flag is enabled, the training set is expanded approximately $4\times$ (e.g., from 7,007 to over 28,000 samples) using the `augment_data` function with the following transformations:

- 90°, 180°, and 270° rotations (controlled via `--aug_allow_rotation`)
- Horizontal and vertical flips (controlled via `--aug_flip_h`, `--aug_flip_v`)
- Random brightness scaling in the range $[0.8, 1.2]$

Augmented samples are combined and shuffled with the original data to avoid ordering bias.

2.3 Architecture-Specific Reshaping

Input data is reshaped to match the requirements of each neural network architecture:

- **MLP:** Images are flattened to shape $(N, H \times W \times C)$ to produce 1D input vectors.
- **CNN:** Images are transposed from (N, H, W, C) to (N, C, H, W) to match PyTorch's channels-first convention.

2.4 Normalization

We normalize the training and test data using global feature-wise means and standard deviations computed from the training set: $x_{\text{norm}} = \frac{x-\mu}{\sigma}$. For CNNs, data is flattened for normalization and reshaped back to preserve spatial structure.

2.5 MLP-Specific Processing

A bias term is appended to each input vector using `append_bias_term()`, enabling the model to learn offsets without modifying weights.

3 Models

3.1 Multi-Layer Perceptron (MLP)

Architecture. A fully-connected stack with user-definable hidden dimensions (default $[512, 256, 128]$) and ReLU activations. The input is a 2352-dimensional vector ($28 \times 28 \times 3$), and the output layer has $C = 7$ logits. Implementation in `MLP` (`deep_network.py`).

3.2 Convolutional Neural Network (CNN)

Architecture. The CNN architecture, implemented in the `CNN` class in `src/methods/deep_network.py`, consists of four convolutional blocks followed by three fully-connected layers. Each convolutional block comprises: `Conv2d` → `BatchNorm2d` → ReLU → `MaxPool2d`.

- **Convolutional Layers:** All convolutional layers use a kernel size of 3 with padding of 1. The filter counts for the four layers are $(128, 256, 512, 1024)$.
- **Max Pooling:** Each `MaxPool2d` layer has a kernel size of 2, which halves the spatial dimensions. After four such layers, the spatial resolution (height and width) is reduced by a factor of $2^4 = 16$.
- **Fully-Connected Layers:** The flattened features from the last pooling layer feed into a sequence of three fully-connected layers with output dimensions ($256 \rightarrow 128 \rightarrow 7$). BatchNorm (`BatchNorm1d`) is applied after the first two fully-connected layers, followed by dropout with a probability of $p = 0.3$.

The final layer outputs 7 logits for the 7 classes.

3.3 Trainer

Optimiser. We replaced vanilla SGD by `AdamW` (with a learning rate `lr`, weight-decay 10^{-4}, betas $(0.9, 0.999)$, and epsilon 10^{-8}) and added GPU/MPS support, worker pin-memory, and verbose logging. The `Trainer` class also exposes a `--grid_search_lr_batch` option that sweeps learning-rates (e.g., $\{10^{-5}, 10^{-4}, 10^{-3}\}$) and batch-sizes (e.g., $\{16, 32, 64, 128\}$), recording accuracy in a heat-map (see Fig. 1).

3.3.1 MLP

For the `MLP`, we further evaluated performance across multiple hidden layer configurations: $\{[128], [256, 128], [512, 256, 128]\}$, where we found the latter to achieve the highest validation accuracy.

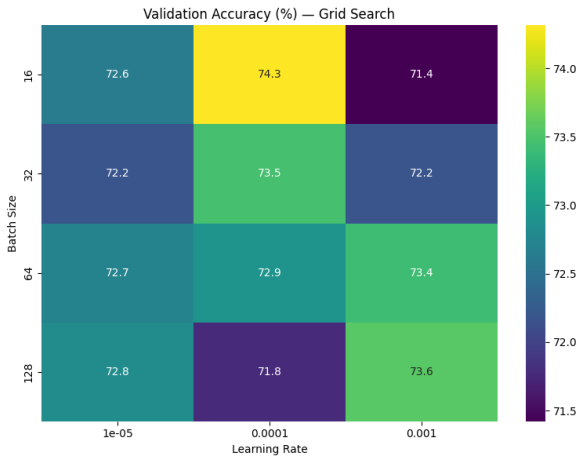

Figure 1: Validation accuracy (%) across the learning-rate and batch-size grid for the MLP hidden layer configuration $[512, 256, 128]$. The best performance is achieved with `nn_batch_size=16` and `lr=1e-4`, followed closely by `nn_batch_size=128` and `lr=1e-3`. In the end, we opted for the second option as it trains in ∼8s instead of 60s, with minimal difference in accuracy.

3.3.2 CNN

CNN was evaluated across various combinations of learning rates and batch sizes, while keeping the architecture fixed (four convolutional blocks and dropout in the fully connected layers). Although some configurations achieved very high training accuracy, we prioritized models with strong validation macro F1-scores. This metric better reflects generalization performance, especially for our imbalanced dataset, where accuracy alone can be misleading. Our final model (Val F1: 0.608, Val Acc: 77.307%) was selected based on this balanced evaluation.

4 Experimental Setup

Unless stated otherwise, models are trained for a specified number of epochs (e.g., 200 epochs for the reported CNN results). We use a fixed split of 10% of the original training data for validation if the `--test` flag is not used.

Training for the `CNN` runs on an NVIDIA RTX 5000 Ada Generation (specified via `--device cuda:0`) with up to sixteen threads for data loading (`--workers 16`). MLP training was conducted on an NVIDIA RTX 2060 Desktop GPU.

5 Results

Model	Train Acc	Val Acc	Train F1	Val F1
MLP	99.971%	74.165%	0.999	0.486
CNN	91.788%	77.307%	0.870	0.608

Table 1: Performance on DermaMNIST. MLP results correspond to a run with LR 1e-3, Batch Size 128, 70 Epochs. For CNN we used LR 1e-3, Batch Size 200, 200 Epochs, and data augmentation on.

5.1 Runtime

On the NVIDIA RTX 5000 Ada Generation, the `CNN` (LR 1e-3, Batch Size 200, 200 Epochs) trains in ∼331s. Inference for the 2005 test images is typically completed in under 50ms. The lighter `MLP` trains in only ∼8s on a RTX 2060 Desktop GPU.

6 Discussion

The CNN performs better than the MLP, showing the advantage of convolutional layers in capturing spatial features. Using data augmentation usually improves macro F1-scores and helps prevent overfitting, as seen in our experiments. Grid search (mainly shown for the MLP in Fig. 1) shows that using a medium batch size and learning rate (e.g., batch size between 32 and 200, learning rate from 10^{-3} to 10^{-5} for CNN) helps balance training speed and stability.

7 Conclusion

In conclusion, we implemented MLP and CNN and compared the performance for skin lesion classification on the DermaMNIST dataset. The CNN outperformed MLP, because it could model spatial features. Data augmentation is another simple way to increase generalization and prevent overfitting. We have implemented our methods using GPU training, data augmentation, and grid search to facilitate efficient experiments. This baseline establishes a solid foundation for future enhancements through more sophisticated models or learning methods.