Advanced Machine Learning Project #1 Report

Rodrigo Nogueira

University of Coimbra

Abstract. This project investigates the use of Multi Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) for classifying pigmented skin lesions using the DermaMNIST dataset, which includes over 10,000 dermatoscopic images categorized into seven skin lesion types. Various hyperparameters were explored to evaluate their influence on model performance. Results showed that CNNs consistently outperformed MLPs, particularly on the training set, although the performance gap narrowed on validation and test sets due to overfitting. Among the hyperparameters tested, activation function, learning rate, and optimizer had the most impact, while batch size and pooling type were less influential. Despite CNNs achieving better overall results, neither model reached high accuracy.

1 Introduction

Image classification is a key problem in computer vision, where machine learning models analyze and categorize images. Traditional methods relied on handcrafted feature extraction with models like Adaboost, Random Forests, SVMs, and MLPs. Deep learning, particularly CNNs, has since revolutionized the field by automating feature learning and achieving superior performance.

CNNs have been especially effective in medical image analysis, including tasks involving the MedMNIST dataset—a benchmark for lightweight medical image classification which covers multiple medical imaging modalities.

This project explores supervised image classification on MedMNIST using Artificial Neural Networks (ANNs), specifically MLPs and CNNs. In this study, for each of these models, different architectures were tested and compared with the goal of evaluating their effectiveness in medical image classification.

2 Problem Description

Medical image classification is an important task in automated disease diagnosis, helping healthcare professionals identify conditions from medical scans. This project focused on classifying pigmented skin lesions using the DermaMNIST dataset, which is derived from HAM10000. The dataset contained 10,015 dermatoscopic images, each categorized into one of seven skin lesion types, including both benign and malignant conditions. The images were preprocessed to a standardized size of $3\times28\times28$ pixels and split into training (7,007 images), validation

(1,003 images), and test (2,005 images) sets. The goal was to develop and evaluate machine learning models, specifically MLPs and CNNs, for classifying these images.

- **MLPs** are fully connected neural networks that process images by flattening them into one-dimensional vectors and passing them through multiple layers of neurons, each applying weights and activations to learn patterns.
- **CNNs** are specialized for image data, using convolutional layers to detect spatial features like edges, textures, and shapes before passing information through deeper layers for classification. Their ability to capture local patterns makes them highly effective for image recognition tasks.

This involved designing a machine learning pipeline, experimenting with different model architectures, and testing various hyperparameter combinations. A key aspect of the project was ensuring the models generalized well, requiring careful validation and the selection of appropriate performance metrics to assess classification accuracy and reliability in a medical setting.

3 Methodology

The methodology that will be followed in this project can be seen in Figure 1. It consists of five main steps:

1. **Pre-processing**: The dataset will be pre-processed to address potential inconsistencies.
2. **Hyperparameter Tuning**: Various configurations of MLPs and CNNs will be tested using grid search with 5-fold cross-validation.
3. **Model Selection**: The best-performing configuration for each model will be chosen based on the validation results and then a model with that configuration will be trained on the entire training set.
4. **Model Evaluation**: We will analyze the various metrics (confusion matrix, loss function, ...) of the best performing models on both the training and validation set. If some problem is detected, we will address it (even returning to the pre-processing step if necessary).
5. **Model Testing**: The best performing models will be evaluated on the testing data.

In all steps, we used accuracy as the main evaluation metric. This made sense because, after preprocessing, the dataset was balanced. In this kind of setting, accuracy gives a fair picture of how well the model is performing overall. The goal here was to correctly identify the specific skin condition in each image, and since no class is more important than the others, we weren't looking to prioritize one type of error over another. What mattered most was simply whether the model got the diagnosis right, making accuracy a straightforward and appropriate choice for evaluating our models.

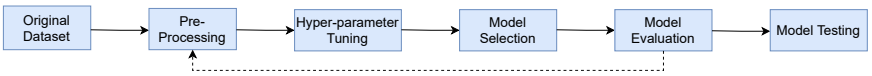

Fig. 1. Overview of the methodology

4 Experimentation

4.1 Pre-Processing

Joining Training and Validation Data Cross-validation will be used to compare different model configurations, meaning that a separate validation set is unnecessary. As such, the training and validation sets were combined, resulting in an 80-20 train-test split.

Undersampling After merging these sets, we examined the class distribution. The dataset was heavily imbalanced, with most images belonging to class 5. This imbalance could skew the model's predictions, making it favor the majority class while struggling with the underrepresented ones. To mitigate this issue, we applied undersampling, limiting the number of samples per class at 500.

While this meant discarding some training data that might have contained useful information, it was a necessary compromise. Keeping a larger number of samples, such as 1500 per class, would have significantly increased training time, making it impractical to test multiple parameter configurations. The 500-sample limit also meant that we would not rely on as much artificial data.

Data Augmentation After applying the undersampling we have removed excess samples from overrepresented classes. However, no new examples have been created for the classes with very few examples. To address this, we used data augmentation to generate additional images for those under-represented classes.

We defined four types of transformations: a random rotation between 0 and 30 degrees, a shift of up to 10% of the image width in any direction, mirroring the image from left to right, and mirroring the image from top to bottom.

For each class with fewer than 500 images, we repeated the following steps until the class reached 500 images: First, we randomly select an image from the class and then we apply a randomly chosen transformation (from the ones described above) to create a new image. By the end of this process, we had a balanced dataset with 3,500 images—500 per class.

Matrix flattening Currently, the images are represented as 28×28 matrices in RGB format, meaning they have three dimensions ($3\times28\times28$), one for each color channel. While CNNs can process images in this format, MLPs require one-dimensional input. Therefore, we flatten each image into a 1×2352 array before feeding it into the MLP model.

Table 1. Hyper-parameters used for both models

Parameter	MLP	CNN
Num epocs	50	50
Batch size	32, 64	32, 64
Learning rate	0.001, 0.01, 0.1	0.001, 0.01, 0.1
Num layers	3, 15	3, 15
Activation func	Relu, Sigmoid	Relu, Sigmoid
Loss function	Cross Entropy, Multi Margin	Cross Entropy, Multi Margin
Optimizer	ADAM. SGD, RMSProp	ADAM. SGD, RMSProp
Pooling	Not applicable	Max Pooling, Average Pooling
Num conv. layers	Not applicable	1, 2
Conv out channels	Not applicable	16
Conv Kernel Size	Not applicable	3
Conv padding	Not applicable	1

4.2 Hyper-parameter tuning

As for the Hyper-parameter tuning step, 5-fold cross validation was used and the parameters tested can be seen in Table 4.2.

5 Results

5.1 Initial Results

Impact of the different Hyperparameters Figure 2 presents the impact of various hyperparameters on the accuracy of CNNs and MLPs. Each subplot compares the average performance of CNNs (red bars) and MLPs (blue bars) for different values of a given hyperparameter. The bars represent the average accuracy across all configurations where the hyperparameter takes a specific value. Both training and validation accuracy are reported, which allows us to assess how each hyperparameter setting influences the model's ability to generalize, and whether a model is overfitting or underfitting.

Some hyperparameters clearly have a strong influence on model performance. For example, ReLU consistently outperforms Sigmoid in both CNNs and MLPs, especially in CNNs. Learning rate is another key factor: lower values like 0.001 result in higher accuracy, while higher rates often lead to unstable training. The choice of optimizer also matters—ADAM and RMSprop perform better than SGD, probably because they adapt the learning rate during training and improve convergence. Additionally, models with fewer layers (three) tend to perform better than deeper ones (fifteen), which may be harder to train and more prone to overfitting in this setup.

Other hyperparameters, however, seem to have less impact. Batch size (32 vs. 64), the number of convolutional layers (1 vs. 2), and pooling type (Max vs. Average) all lead to similar results. Likewise, Cross-Entropy and Multi-Margin loss functions produce comparable accuracy in both CNNs and MLPs.

Fig. 2. Average model performance for each hyperparameter value

Best Configuration We now examine the configurations that led to the best validation accuracy for each model. For the MLP, the optimal configuration included the following hyperparameters: activation function: ReLU, number of hidden layers: 3, batch size: 64, learning rate: 0.001, loss function: Multi Margin, and optimizer: Adam.

For the CNN, the best configuration was as follows: activation function: ReLU, pooling: MaxPool, number of convolutional layers: 2, convolutional output channels: 16, number of hidden layers: 3, padding: 1, kernel size: 3, batch size: 64, learning rate: 0.1, loss function: Cross Entropy, and optimizer: SGD.

The best configuration for the models aligns well with the previous analysis of hyperparameter effects. Most choices—such as the use of ReLU and a three hidden layers—are consistent with findings that these settings tend to lead to better performance. The only notable exception is the learning rate of 0.1, which is higher than the previously suggested optimal value of 0.001. This can be explained by the limited training duration: with only 50 epochs, a higher learning rate may have allowed the model to converge more quickly, even if at the cost of some stability. In contrast, lower learning rates typically require more epochs to reach optimal performance but tend to offer more stable and precise convergence.

Table 2. Results of the best performing configuration for both MLPs and CNNs

Metric	Accuracy	
Data	Train	Validation
MLP	0.53	0.48
CNN	0.97	0.65

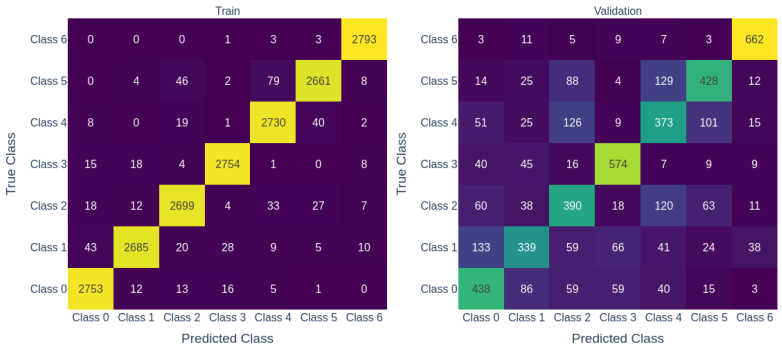

Fig. 3. Confusion Matrix for the best performing CNN

As shown in Table 5.1, the CNN outperforms the MLP in both training (0.97) and validation (0.65) accuracy (as expected). However, the large gap between training and validation accuracy in the CNN indicates overfitting, where the model performs well on training data but struggles to generalize. The MLP, on the other hand, shows more consistent performance with a smaller difference between training (0.53) and validation (0.48) accuracy, though it achieves lower overall accuracy.

At first, we suspected the overfitting could be caused by the data augmentation. The dataset consists mainly of round signals, so some augmentation techniques, like rotation and translation, don't significantly alter the images. As a result, the model may have been memorizing these patterns instead of learning generalizable features. In order to check whether this was the case, we looked at the confusion matrices for both train and validation (Figure 3).

As we can see, overfitting is not present in the classes where more data augmentation has been applied. This suggests that data augmentation was not the primary cause of the overfitting.

When looking at the training loss curve across the different folds (Figure 4), two conclusion can be be made:

1. **More epochs are needed**: The loss curve is still showing a considerable slope, indicating that the model has not yet fully converged. Increasing the number of epochs could potentially improve the model's performance.
2. **Learning instability**: Although a learning rate of 0.1 led to the best results, this is likely due to the limited number of training epochs. As previously discussed, lower learning rates generally offer more stable convergence, but require more epochs to reach optimal performance. Therefore, increasing the

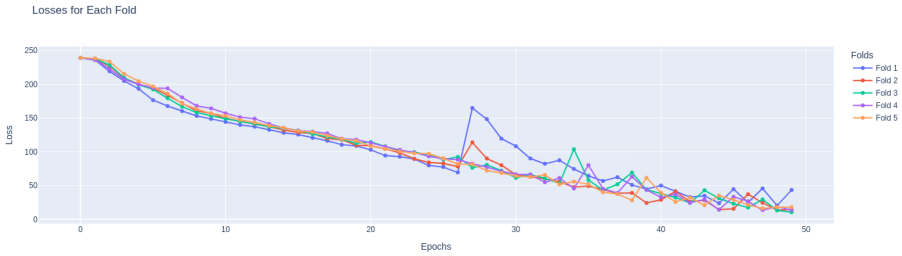

Fig. 4. Training loss across the different folds

number of epochs while reducing the learning rate could potentially yield better results with improved stability.

5.2 Fixing Attempts

Based on the observed results, three different strategies were tested to try to reduce overfitting: adding weight decay, adding early stopping based on the validation loss and testing additional network configurations (1 and 2 hidden layers, 32 and 64 output channels, and 1 and 3 convolutional layers, all the other hyperparameters were the same as in the best performing CNN)

After applying all these techniques, we will take the best performing configuration and train it using a reduced learning rate (0.01) and an increased number of epochs (300) and analyse the final performance.

5.3 Final Results

In Table 3, we can see that performances obtained by the best model configuration when each of the mentioned strategies was applied.

As we can observe, the first two techniques did not lead to an improvement in the validation accuracy. However, one the of the additional configurations we tested lead to an improvement in performance (the configuration with only one hidden layers, three convolutional layers, and 64 convolutional output channels).

We then trained a CNN with these hyperparameters but with a learning rate of 0.01 for 300 epochs. This model achieved an accuracy of 1.0 on the training data and 0.70 on the validation data, meaning that the decrease in the learning rate and the increase in the number of epochs improved the model performance. When looking at the loss functions (Figure 5), we can see that this number of epochs was appropriate, as both the loss functions seem to have stabilized.

5.4 Model Testing

Once the best performing model on the validation data for both MLPs and CNNs had been identified, we tested both these models on the testing data. The MLP achieved an accuracy of 0.48, while the CNN achieved an accuracy

Table 3. Results

Metric	Accuracy	
Strategy	Train	Validation
Weight Decay	0.95	0.65
Early Stopping	0.84	0.63
Different Network Structure	0.94	0.67

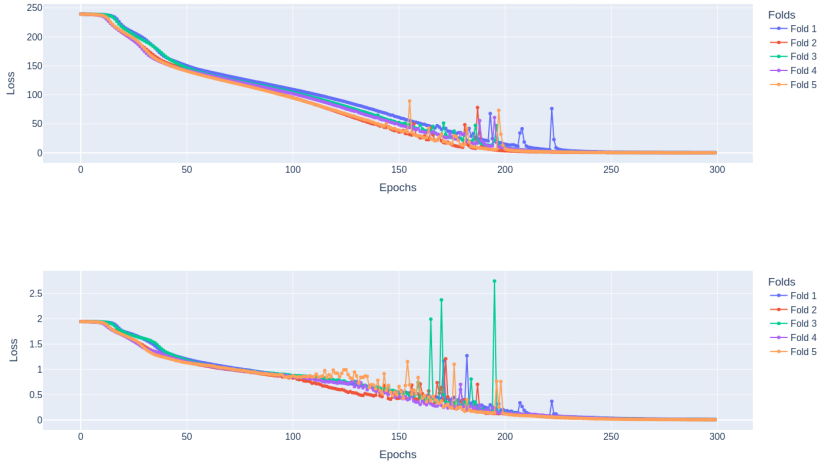

Fig. 5. Train (top) and validation (bottom) loss curves

of 0.6. This drop in accuracy from the validation set to the testing set can be explained by the way data augmentation was applied: we augmented the entire training dataset (e.g., by rotating images) before splitting it into training and validation subsets. As a result, the training and validation sets contained more similar images than the training and testing set.

6 Conclusion

In this work, we explored the use of MLPs and CNNs to classify skin lesion images into one of seven distinct classes, each corresponding to a specific dermatological condition. Through systematic experimentation with different hyperparameters, we observed that certain choices—such as activation function, learning rate, and optimizer—had a clear impact on model performance, while others, like batch size and pooling type, played a less significant role.

Overall, CNNs outperformed MLPs across all datasets, with the largest performance gap observed on the training set. However, this advantage was somewhat reduced on the validation and test sets, likely due to overfitting. Despite this, neither model achieved particularly high accuracy, highlighting the complexity of the task and suggesting that further improvements may be necessary for reliable real-world application.