

AN2DL - First Homework Report

ReLUctant Learners

Mattéo Bevilacqua, Gabriele Lorenzo Singe, Vittorio Casalini, Massimiliano Botta
cyhagha, gabbo613, vittocasalini, maxbot01

275840, 275589, 981802, 259740

November 24, 2024

1 Introduction

The objective of this challenge is to develop a Neural Network for accurate **image classification** of **blood cells**. Given our limited experience, we adopted an incremental approach. We analyzed the problem, studied blood cell types, and inspected the dataset. With growing expertise, we improved model robustness through data augmentation and hyperparameter tuning, achieving significant performance gains.

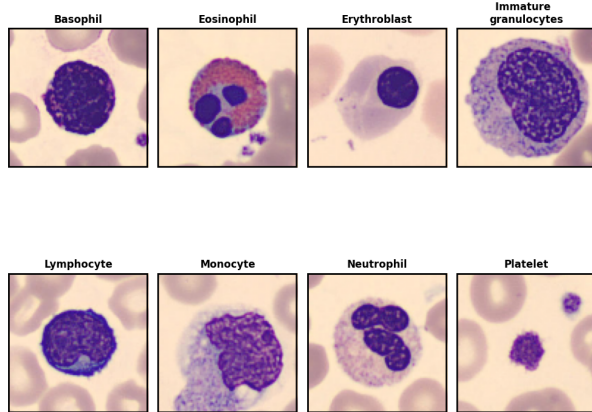


Figure 1: Distribution of the classes in the cleaned dataset

2 Problem Analysis

The proposed dataset is composed of a total of 13759 images, spread in 8 different classes.

2.1 Data cleaning

We started by inspecting the images from different classes. This allowed us to realize that several images were significantly different from the other. Further analysis allowed us to identify several **duplicate images**; 1800 of these were outliers: images layered with irrelevant pictures; and 8 other images were present twice each in the dataset.

As to not train the model on corrupted data and introducing as little bias in our dataset, we decided to get rid of all these images, bringing the total size of the dataset down to 11951.

2.2 Class imbalance

Another important observation was the significant class imbalance, with the ratio between the most and least represented class being $2330/849 \approx 2.7$.

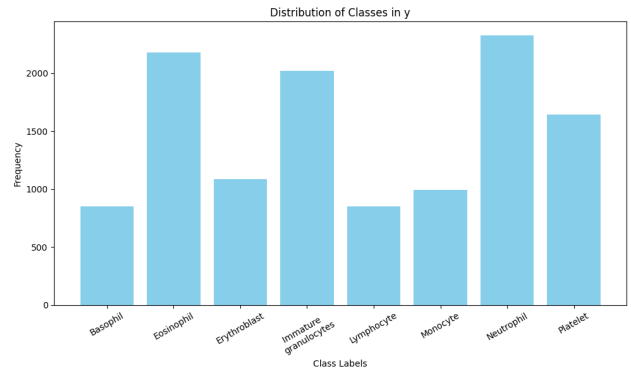


Figure 2: Class distribution in the cleaned dataset

This imbalance raised concerns about its alignment with the true probability distribution of the target classes and the risk of biased model behavior. To address this, multiple strategies were tested. Initially, class weights were adjusted, but this approach showed no notable improvements. Subsequently, two methods were compared: oversampling alone and a combination of oversampling and under-sampling.

As expected, the removal of data obtained with the combined method led to poorer outcomes during training and testing due to the loss of valuable information.

In contrast, oversampling smaller classes yielded better results. Additional experiments with oversampling the majority classes extended training times considerably without improving model performance.

2.3 Augmentation

Image duplication alone would not meaningfully enhance the dataset without **augmentation**. This technique applies transformations like rotations, flipping, saturation adjustments, and other modifications to increase sample diversity and address duplicate data. A key challenge was selecting the most suitable augmentations for the cells. We tackled this by considering transformations relevant to the medical context and concluded that rotations, flipping, and stretching were most beneficial. Experiments showed that random augmentation using Keras’s RandAug method [1] produced the best results.

3 Method

We started with a custom model to learn the environment and layer functionality, which deepened our framework knowledge. We then used a pre-trained Keras backbone with generalization layers to improve performance.

Our approach began by selecting the optimal network and structuring training into a warm-up phase, followed by fine-tuning. We developed a strategy to optimize the network’s **hyperparameters** and defined the best validation approach. A key part of our workflow was a shared Excel file, allowing team members to document updates, report scores, and track performance. This tool was

crucial in monitoring changes and preventing redundant experiments.

3.1 Network research

Our initial experiments utilized MobileNetV3Small, a compact network with approximately 3.5 million parameters, often deployed in embedded systems. Unfortunately, initial results indicated **underfitting**, suggesting that the dataset’s complexity necessitated a more robust model. We subsequently explored various versions of EfficientNet based on [2] documentation, which provided guidance through metrics like Top-1 and Top-5 accuracy on ImageNet. This led us to adopt EfficientNetB2, resulting in improved prediction accuracy. In the later stages, we transitioned to EfficientNetV2S, a more complex model with 21.6 million parameters, achieving further performance gains.

3.2 Customization

To fine-tune each base model, we tried incorporating custom **dense layers** to adapt the networks for our specific classification task, followed by **dropout layers** to enhance generalization and reduce overfitting. This led us to an optimal configuration, culminating in the addition of a **squeeze-and-excitation attention layer** to refine feature focus and improve overall performance.

We also employed a dense layer with 128 neurons, with a dropout rate of 0.4 for regularization. The custom layers concluded with a last dropout component set at 0.2, for further regularization.

4 Experiments

We dedicated most of our training instances to optimizing the hyperparameters, anticipating the outcomes of each change to achieve the desired results. Below is a summary of our observation:

Dataset split: 80% of the dataset was used for training, with the remaining 20% split evenly between validation and testing. Additionally, the Codabench platform provided extra validation.

Optimizers: We used two optimizers: Adam for the warm-up phase, ideal for feature extraction, and the more stable Lion optimizer for fine-tuning.

Learning rate: For Adam, the learning rate was set to $\alpha = 2 \times 10^{-4}$, while Lion used 5×10^{-5} for

Backbone	Data balancing	Augmentation	Codabench Score
Custom	No	RandAugment	< 0.5
MobileNetV3Small	No	RandomHue + Solarization + Posterization	0.54
EfficientNetB2	Yes	RandAugment+ AugMix	0.81
EfficientNetB2	Yes	RandAugment	0.85
EfficientNetB3	No	Colour and Geometric through pipeline	0.71
EfficientNetV2B2	Yes	RandAugment	0.86
EfficientNetV2B3	Yes	RandAugment	0.84
EfficientNetV2Small	Yes	RandAugment	0.90
EfficientNetV2Medium	Yes	RandAugment	0.89

Table 1: Table of the network architectures tried divided by type. Best results are highlighted in **bold**.

smoother convergence and better optimization, albeit requiring longer training.

Batch size: A batch size of 64 images was chosen for both optimizers, balancing stability and convergence speed. Larger batch sizes led to *overfitting*, and smaller sizes caused early loss function implosion.

Epochs: We used a 40-epoch warm-up for feature extraction, followed by 120 epochs for fine-tuning to ensure convergence.

Patience: The Adam optimizer had a patience of 5 epochs, while Lion used 13. Adam’s patience didn’t trigger due to the short feature extraction phase, while Lion’s balance between training duration and weight quality was optimal.

5 Results

Our best results with each model architecture is listed in the table above. As shown, the best results came from the **EfficientNetV2** network of type **Small** and **Medium**, an outcome confirmed by the final test, with a 0,89 score.

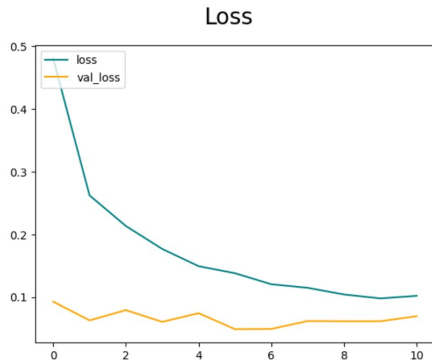


Figure 3: MobileNetV2Small: Loss function evolution during Fine Tuning phase

6 Discussion

The results obtained show the importance of using pre-trained architectures, data balancing, and augmentation strategies, with the best performance achieved using state-of-the-art models and RandAugment.

However, the reliance on a single augmentation method may limit the model’s robustness to unseen data transformations. Additionally, while data balancing improved scores, its implementation could be refined to address potential over-representation of certain samples. Further exploration of diverse augmentation pipelines and alternative balancing strategies could further enhance generalizability.

7 Conclusion

A wide range of course material was explored during this challenge, including simple CNNs, state-of-the-art models, and transfer learning. Data was thoroughly analysed, and augmentation pipelines were used to maximize its potential. Fine-tuning hyperparameters proved crucial, as small adjustments often yielded significant improvements.

With more time, enhancements such as completing K-Fold validation, implementing model bagging, and designing an on-the-fly augmentation pipeline could have further improved performance.

Group contributions

All team members contributed equally to every stage of the project, collaboratively refining the main notebooks through incremental improvements and shared insights, ensuring a cohesive and comprehensive final outcome.

References

- [1] https://keras.io/api/keras_cv/.