AN2DL - First Homework Report NoPainNoTrain

Noemi Bongiorni, Alessandro Pedone, Simone Licciardi, Federico Maria Riva

noemibongiorni, alepedone, simonelicciardi, feder

278173, 273814, 214875, 280447

November 24, 2024

1 Introduction

This project aims to classify RGB images of blood cells among eight disjoint classes (*basophil [0], eosinophil [1], erythroblast [2], immature granulocytes [3], lymphocyte [4], monocyte [5], neutrophil [6], platelet [7]*) through a CNN.

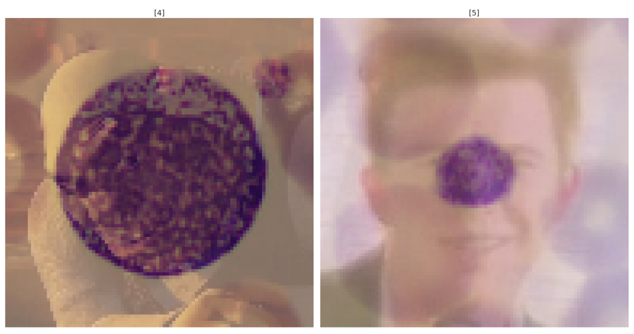
2 Problem Analysis

To assess the task, we perform a statistical and visual inspection of the dataset, determining its relevant characteristics and noticing several outliers.

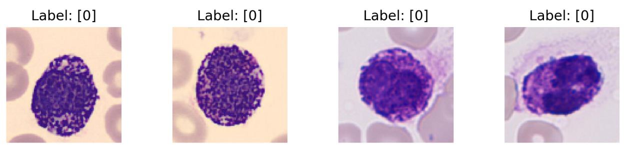
Dataset inspection. The dataset consists of a `(13759, 96, 96, 3)` tensor, the 3-channel RGB images, and a `(13759,)` tensor of labels.

Main challenges. Visual inspection reveals the main learning challenges. Images differ by *details*, not global pattern; formally, high-frequency differences are more important than low-frequency ones. Moreover, there is an *unbalanced* distribution of samples among classes and some images are corrupted by the *superposition* of another image (see Figure 1a). Some images are corrupted by *colour filtering* (see Figure 1b). To address the above, we preprocess the dataset by detecting and **removing superpositioned images** (resulting in a clean `(11959, 96, 96, 3)` tensor), while we leave blue-filtered ones untouched due to difficult detection and overall small impact (indeed, the filtering preserves the high-frequency changes). The latter seems to account for 2% of the database. We also choose high-pass filter layers in the model, and add class weights to contrast the other highlighted challenges.

Initial assumptions. We assume that for each image there is no ambiguity on its unique class (equivalently, that either the model is correct, or is mistaken), that all mistakes weight the same towards our downstream tasks, that the images have no preferred orientation and that zoom is constant.



(a) Images with superposition noise.



(b) In-class comparison between standard images (left) and blue filtered ones (right).

3 Method

To develop our solution, we split the filtered dataset into test (20%), validation (8%) and training (72%) sets. We fine-tune hyperparameters, such as the number of **epochs, batch size, learning rate** (including strategies like reducing the learning rate on plateaus), and **patience** levels for early stopping. And we implement mixed precision, to balance numerical stability and speed of operations. We choose **categorical cross-entropy** as the loss function, which aligns with evaluation metrics of the external assessment. In order to mitigate overfitting, we use **early stopping, regularization, dropout** and weight initialization to ensure stable convergence. We focus on **data augmentation** too, with the scope of enhancing model robustness, and we utilize standard transformations (flip, rotation, translation, zoom, contrast, etc.), tuning their parameters, so that the they are relevant to the dataset's characteristics and we are avoiding misleading distortions. Additionally, we incorporate **class weights** to address the imbalance between classes. The modeling process is devided into two parts, corresponding to two different strategies. First, we design a **simple architecture** that we can consider a baseline, emphasizes interpretability and allows us to test the effectiveness of some of the already mentioned techniques. Next, we shift our attention towards **transfer learning** (TL) and **fine-tuning** (FT) using the available pre-trained networks. The key decisions we make in this step are selecting an appropriate base network, configuring dense layers (e.g., number of units, use of **batch normalization**, and dropout), specifying input pre-processing and shape and opting for max pooling to capture fine details. For FT, we determine trainable parameters based on a heuristic percentage (between 30% and 1%) to balance computational efficiency and feature adaptation. Finally, we build an **ensemble** of two neural networks to combine the (complementary) features learned by different architectures and improve overall performance and robustness.

4 Experiments

As summarized in Table 1, we conduct a series of experiments by progressively modifying a baseline CNN architecture to assess the impact of each change on model performance.

Baseline network was trained on the complete dataset comprising 13,759 images. This architecture consists of two blocks, each including a single convolutional layer, followed by a softmax activation layer and a max-pooling layer. While this configuration yields reasonable performance, the results can certainly be improved.

Network 1 has the primary modification of training the model on a filtered subset of the dataset (11,959 images). This adjustment consistently improves model performance, demonstrating the benefits of dataset refinement.

Network 2 introduces a **resizing** layer to downscale the input images from 96x96 to 32x32 before the convolutional layers. However, this alteration results in decreased model accuracy, likely due to the loss of critical features caused by the aggressive reduction in spatial resolution.

Network 3 incorporates **LecunNormal** weight initialization and **L2-norm** regularization for the weights. While these additions don't significantly improve local accuracy, they are hypothesized to stabilize training and potentially enhance generalization.

Network 4 adds dropout and batch normalization layers to the architecture. These layers aim to improve generalization and accelerate convergence by reducing overfitting and normalizing feature distributions during training.

When evaluated on CodaBench, none of the models achieves an overall accuracy exceeding 25%, suggesting suboptimal generalization to unseen test sets. Nevertheless, our experiments provide valuable insights. Specifically: training on the filtered dataset proved beneficial for model performance; downscaling inputs adversely affected accuracy, likely due to the removal of key features; weight initialization, regularization, and the inclusion of dropout and batch normalization, while not yielding noticeable improvements in local accuracy, contributed to better performance on the CodaBench evaluation. Based on these findings, we retain the most effective techniques and applied them to TL and FT experiments using pre-trained networks to further improve model performance. As summarized in Table 2 ["/" means that we couldn't upload the model due to CodaBench related issues.], we compare different architectures, using them for TL and FT our model. The **percentage of FT** corresponds to the percentage of parameters we decide to unfreeze and re-train for our model. We tested between 1% and 30%, reporting in the table the best one for each architecture. We test the SGD [5], Adam [8] and **Lion** [6] optimizers. The best-performing is the latter, with a learning rate schedule from 5e−4 to 1e−5 for the TL phase, and from 1e−4 to 5e−6 for the FT. The difference in parameters is due to the difference in aim: in the first case we need to train layers from scratch, and so we initially prioritize significant learning steps, while in the second case we are already located near a local minimum and thus the alignment of the source architecture

Table 1: Comparison on a simple CNN of different choices of parameters/layers/dataset

Model	Accuracy	Precision	Recall	F1
Baseline	86.59	89.71	86.59	87.17
Network 1	94.15	94.38	94.15	94.15
Network 2	91.81	91.94	91.81	91.7
Network 3	91.76	91.98	91.76	91.73
Network 4	**91.43**	**92.02**	**91.43**	**91.61**

Table 2: Comparison of different nets for TF and FT

Model	Accuracy Codabench score	Precision Recall	F1	FT percentage
DenseNet201	87.04 /	88.31 87.04	86.59	10
ConvNeXtTiny	96.20 53.00	96.36 96.20	96.13	1
ConvNextSmall	95.74 64.00	95.88 95.74	95.76	10
InceptionResNetV2	91.39 48.00	91.76 91.39	91.43	1
EfficientNetV2B3	94.57 63.00	94.58 94.57	94.55	10
EfficientNetV2S	**93.23** **70.00**	**93.75** **93.23**	**93.33**	**10**
EfficientNetV2M	92.02 63.00	92.04 92.02	91.98	10

requires smaller stepsizes.
Notably, with Lion a schedule increases performance dramatically: its stepsize is fixed and thus cannot converge satisfactorily. Then we want to choose the number of layer and neurons, for the last dense layers. In the end, a single layer with 64 neurons turns out to be the best choice.

5 Results

The **network** we submitted (CodaBench score of 0.76) was an ensemble of three models based on the best performing networks: ConvNeXSmall (model1), EfficientNetV2S (model2) and EfficientNetv2B3 (model3). This result demonstrate significant improvements over the baselines, outperforming simpler architectures and basic TF/FT implementations.

6 Discussion

Strengths: advanced architectures, effective optimization, and an ensemble approach enhanced generalization. *Weakness*: we couldn't build a model that generalizes well enough so that the local accuracy could match the one on CodaBench. *Limitation*: we weren't able to test all our ideas to improve our model after the network comparison for TL/FT due to CodaBench not working properly.

7 Conclusions

Each team member worked on every task and gave his/her best contribution as follows: Noemi Bongiorni and Alessandro Pedone in architecture design, Simone Licciardi in hyperparameters tuning and ensemble building and Federico Maria Riva in pre-processing and augmentation. Future areas of development include testing **additional networks** for TL to improve accuracy by extracting different features, applying **Grad-CAM** to inspect the model's heat map, and improving image augmentation by allowing **Keras-CV** library to act directly within the model, after resolving implementation issues.

References

[1] M. R. Hossain, D. Timmer, and H. Moya. Machine learning model optimization with hyperparameter tuning approach. In *Proceedings of the International Conference on Advanced Computing*, August 2021.

[2] O. Islam, M. Assaduzzaman, and M. Z. Hasan. An explainable ai-based blood cell classification using optimized convolutional neural network. *Department of EEE and HIRL, Daffodil International University, Bangladesh*, 2023. Available online.

[3] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[4] H. Li, C. Fowlkes, H. Yang, O. Dabeer, Z. Tu, and S. Soatto. Guided recommendation for model fine-tuning. In *AWS AI Labs*, 2023. Available via AWS Research.

[5] S. Mandt, M. D. Hoffman, and D. M. Blei. Stochastic gradient descent as approximate bayesian inference, 2018.

[6] D. H. E. R. K. W. Y. L. H. P. X. D. T. L. C.-J. H. Y. L. Q. V. L. Xiangning Chen, Chen Liang. Symbolic discovery of optimization algorithms. May 2023. Available online.

[7] Q. Yang, Y. Zhang, W. Dai, and S. J. Pan. *References*. Cambridge University Press, 2020.

[8] J. Yun. Stochgradadam: Accelerating neural networks training with stochastic gradient sampling, 2024.