AN2DL - First Homework Report
LOS PINGUINOS

Daniele Vozza, Francesco Tomasi, Zeno Peracchione, Lorenzo Galatea

danyvois, francescotomasi, zenopera, zenzero27

252574, 252533, 250643, 271594

November 25, 2024

1 Introduction

This project addresses a **multi-class classification problem** involving 96x96 RGB images of blood cells. The dataset contains eight distinct classes, each representing a specific blood cell type, such as *basophils*, *neutrophils*, and *lymphocytes*.
Our approach started with a simple *from-scratch model* to establish a baseline. We then progressively advanced to **transfer learning with fine-tuning**, leveraging pre-trained networks to improve classification accuracy and efficiency.

2 Inspect Data and Data Cleaning

In this section, we provide a detailed overview our initial data exploration process. Blood cells are categorized into the following eight classes:

- 0: *Basophil*
- 1: *Eosinophil*
- 2: *Erythroblast*
- 3: *Immature granulocytes*
- 4: *Lymphocyte*
- 5: *Monocyte*
- 6: *Neutrophil*
- 7: *Platelet*

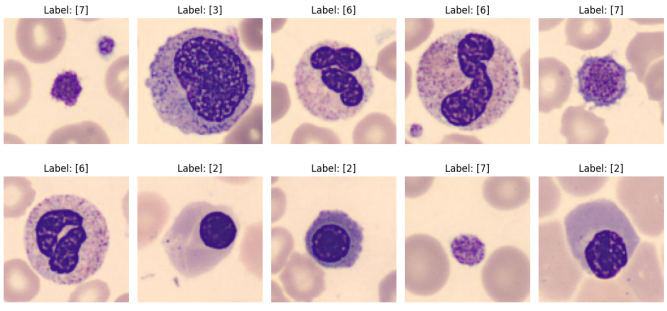

Figure 1: First 10 images of dataset and their labels

We visualized the first and last 10 images, quickly identifying outliers unrelated to blood cells, such as *Shrek* and *Rick Astley*, concentrated at the dataset's end.
After reviewing the entire dataset, we determined a `cutoff_idx = 11959` to exclude all outliers. A cleaned file, `training_set_trimmed.npz`, was created, containing only valid images for subsequent analysis and training.

3 Dataset Splitting and Evaluation Strategy

Initially, we split the dataset into training and validation sets using an 80-20 ratio, intending to use Codabench submissions as the test set for an unbiased evaluation of the model's performance. However, technical issues with Codabench prevented us from uploading submissions for testing. To address this, we adjusted our data split to include training, validation, and test sets locally, ensuring that we could still evaluate the model's performance independently of Codabench.

Figure 2: Ouliers examples

4 First Model: From Scratch

4.1 Basic Model

Our initial *Convolutional Neural Network* (CNN) model, inspired by the lab sessions, included three convolutional layers with 32, 64, and 128 filters of size 3×3, each followed by **max pooling** to reduce spatial dimensions. We used *"valid"* padding to maintain consistency in feature maps, then **flattened** the output and added a dense layer with a *softmax* activation for classification.
The model was compiled with the **Adam optimizer** (learning rate: 0.001) and **sparse categorical crossentropy loss**, suitable for integer labels. To prevent overfitting, we applied *early stopping*, halting training when validation performance plateaued and also a basic implementation of *augmentation*.
This setup achieved **95.44% accuracy** on local training and validation data in nearly 100 epochs, but suffered from overfitting, with only 0.21 accuracy on the *Codabench* test dataset.

4.2 Addressing Overfitting: Regularization and Data Augmentation

To address *overfitting*, we added a **dropout layer** (rate: 0.5) and applied *L2 regularization* on the output layer's weights. While these adjustments brought some improvement, overfitting persisted.
We further applied **data augmentation** using `ImageDataGenerator`, introducing *rotations*, *flips*, *translations*, and adjustments to brightness and contrast to increase data variability.
These techniques slightly improved the model's test performance, achieving an accuracy of 0.23 on the hidden test set, though overfitting remained a challenge.

Class	Images
0	852
1	2181
2	1085
3	2026
4	849
5	993
6	2330
7	1643

4.3 Addressing Class Imbalance

A deeper analysis revealed an imbalance in class distribution, which we addressed by calculating *class weights* to ensure underrepresented classes contributed more to the loss function.
This adjustment slightly improved the model's performance on the *Codabench* test dataset, achieving a maximum accuracy of **0.29**, but highlighted the need for more advanced techniques to overcome the dataset's challenges.

4.4 Experiments

We experimented with adding a fourth convolutional layer to extract more complex features and tested alternative optimizers, such as *Lion* instead of *Adam*, to evaluate their impact on accuracy.
However, these changes failed to improve accuracy and occasionally worsened performance.

5 Transfer Learning and Fine Tuning

Given the availability of pre-trained neural networks, we utilized transfer learning and fine-tuning to improve performance. Each team member started with a different pre-trained model: **ResNet50V2**, **Efficient-**

NetB3, **InceptionV3**, and **ConvNeXtSmall**. Contrary to expectations, models with fewer parameters did not perform better.
Surprisingly, **ConvNeXtSmall** excelled in local evaluations (Codabench was non-functional). Following the project notebook, we implemented **mixed precision training** to accelerate execution and reduce memory usage. We also applied **early stopping** and a **learning rate scheduler** that decayed the learning rate exponentially after 10 epochs, yielding more precise results.
To address overfitting, we designed and implemented **seven custom data augmentation pipelines**, dynamically selecting one of them during training. These pipelines included techniques such as random cropping, flipping, brightness and contrast adjustments, and solarization. This strategy increased data diversity, improved generalization, and strengthened the model's robustness against overfitting.

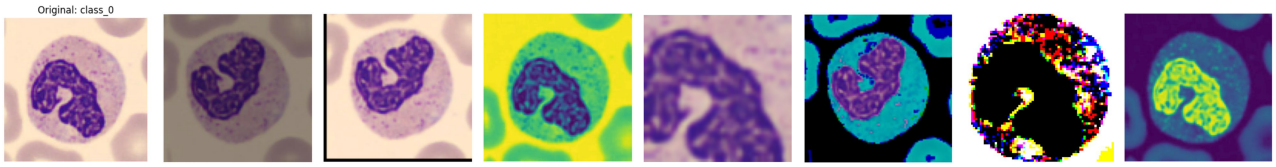

Figure 3: Examples of advanced augmentation

To reduce overfitting, we added a **Dropout layer** after the **ConvNeXtSmall** output and two **Dense layers** with **ReLU activation**, **L1L2 regularization**, and the **HeUniform initializer**. Another Dropout layer preceded the final Dense layer, which used the **GlorotUniform initializer**.

Initially, freezing all pre-trained layers yielded 0.84 validation accuracy. Fine-tuning by unfreezing the convolutional layers improved performance, maintaining a low number of trainable parameters while adapting to blood cell classification. The final model achieved significant improvement, reaching **97.66%** accuracy on the local test set.

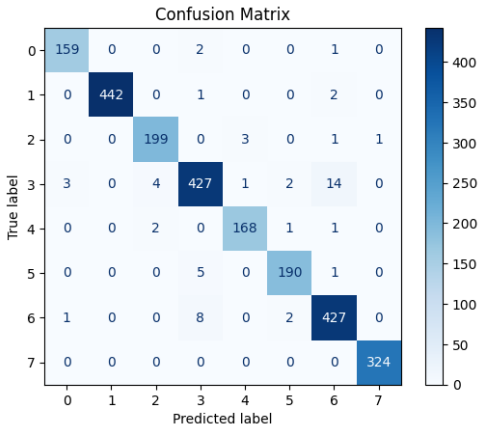

Figure 4: Confusion matrix of the final model

6 Conclusion

In the final phase of the challenge, we achieved our best results, with a local test accuracy of **98.6%**, demonstrating the potential of our approach. However, during testing on *Codabench*, we repeatedly encountered errors that resulted in failed submissions. Unfortunately, the limited time available prevented us from resolving these issues. Despite these challenges, we believe our model shows great promise, and with further debugging and refinement, it could achieve strong performance on external test sets.

7 Contribution

Our work was a collaborative effort: **Francesco** developed an effective augmentation strategy, **Daniele** implemented the core structure of the code with ConvNeXtSmall, forming the foundation of our final model, **Zeno** identified the best layers to unfreeze for fine-tuning, and **Lorenzo** tested various optimizers to improve convergence. Together, we refined our approach iteratively, achieving a robust and high-performing model.