AN2DL - First Homework Report
SL4ANN

Giorgia Arena, Sara Auletta, Giulia Di Vincenzo, Christian Frigerio

gioare, saraauletta, giuliadivinc, chrifriz

279088, 280911, 274976, 279082

November 24, 2024

1 Introduction

This project focuses on *image classification* using **deep learning** techniques. The given dataset consists of **13759 images**, belonging to **8 different classes**, designed for the classification of different types of blood cells: Basophil, Eosinophil, Erythroblast, Immature granulocytes, Lymphocyte, Monocyte, Neutrophil and Platelet. The aim of the project is to build a model that assign the correct class label to each RGB image. Specifically, we started with a baseline and slowly made choices from there, seeing whether the performance improved or not, trying to understand why and if the outcome was expected. We used the **accuracy score** as principle metric, but also the precision, recall and F1 scores in order to understand better the weaknesses of the model.

2 Data Preprocessing

2.1 Inspect Data

We analyzed the dataset by plotting class distributions and inspecting samples, uncovering two issues. Images 11958–13558 were duplicated with conflicting labels, risking overfitting, while image 13559 was an outlier repeated throughout the dataset, adding noise. To improve data quality, we removed these problematic images.

2.2 Data Distribution

The refined dataset revealed an issue of *class imbalance*. To mitigate this problem, we calculated **class weights** to regularize the training process. Consequently, the cross-entropy loss function was adjusted to include these class weights:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} w_{y_i} \log(\hat{y}_{i,y_i}) \tag{1}$$

Where y_i is the true class label, $\hat{y}_{i,y_i}$ is the predicted probability for class y_i, w_{y_i} is the weight for class y_i, and N is the total number of samples.

2.3 Data Augmentation

We implemented an extensive data augmentation pipeline using the Albumentations library, tailored for blood cell images to address variations in staining, lighting, and imaging noise. Techniques included color adjustments (hue, saturation, brightness, and contrast), geometric transformations (cropping, rotation, translation), and quality variations (downscaling, blurring). Augmentations like A.RandomCrop emphasized cell features, while A.HueSaturationValue and A.CLAHE focused on enhancing color accuracy and improving image contrast. Parameters were carefully adjusted to balance augmentation intensity, significantly improving model generalization and accuracy by enabling it to handle diverse conditions effectively [2]. We found all these augmentation techniques in this article [1].

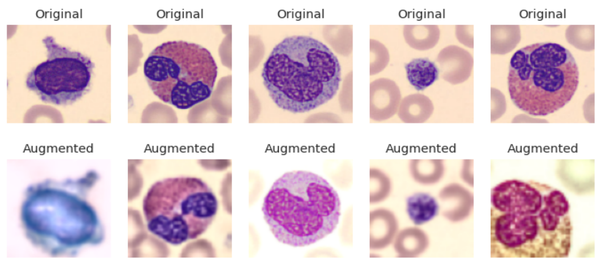

Figure 1: Examples of Augmented images

3 Convolutional Neural Network

We initially started with a baseline model based on the classical architecture of modern CNNs. The model consisted of 5 blocks, each with a combination of `Conv2D` and `MaxPooling2D` layers, followed by a `Dense` layer. However, the accuracy achieved by this model was not satisfactory: 24% (roughly twice as good as a random guess). We chose this as a baseline to improve our model. To improve performance, we turned to transfer learning, starting with the pre-trained MobileNetV3Small model. Along with this, we incorporated an Inception block, and used the Lion optimizer (see section 3.2). This approach resulted in a starting accuracy of 65.00%, confirming that these modifications were the right direction to improve performance.

3.1 Feature extraction

In order to choose the best feature extractor some famous architecture (in their medium level size) with the same model as before, and run the simulation for 30 epochs.

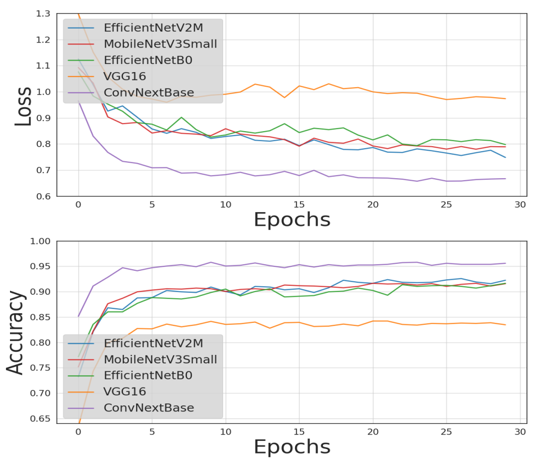

Figure 2: Architectures comparison

Based on the results shown in the table below, ConvNextBase emerged as the best-performing model. However, we decided to retain EfficientNetV2M for further evaluation to observe how both models perform after fine-tuning and on the test set. This approach aims to provide deeper insights into their behavior and guide the final model selection. Both models were submitted for this purpose.

EfficientNetV2M	**92.58%**
MobileNetV3Small	91.64%
EfficientNetB0	91.64%
VGG16	84.21%
ConvNextBase	**95.79%**

3.2 Optimizer and learning rate

To determine the most suitable optimizer, we compared several well-known optimizers using the same simple model and ran simulations for 30 epochs, as in the previous paragraph. We decided also to apply, for each optimizer, the `ReduceLROnPlateau` that reduces the learning rate when the validation accuracy stops improving. This helps the model converge more precisely and avoid vanishing gradient and overshooting as it gets closer to the optimal solution. It's important because it allows the model to adaptively adjust the learning rate during training, improving performance and stability.
Based on the results shown in Table 1, **Lion** emerged as the best-performing optimizer, so we decided to proceed with it for our work.

4 Training Techniques

4.1 Fine Tuning

We increased the complexity of our model by adding additional `Dense` layers. However, with transfer learning, the accuracy improved only marginally to 0.67. To address this, we applied fine-tuning to the two best-performing models.
We began with **EfficientNetV2M**, freezing 500 out of 740 layers. This led to a significant improvement, with accuracy increasing to **0.80**. Encouraged by this result, we unfroze all layers, but the accuracy dropped to 0.73, likely due to overfitting.
Next, we focused on ConvNextBase, unfreezing 150 out of 250 layers (60%, the same proportion that worked best for EfficientNet). However, this yielded a less satisfying accuracy of 0.77. When we unfroze

Table 1: Optimizers comparison

Optimizer	Train loss	Train accuracy	Val. loss	Val accuracy
Lion	**0.7645**	**87.35%**	**0.7559**	**92.04%**
Adam	0.8990	82.01%	0.8245	91.24%
AdamW	0.9247	80.67%	0.8325	90.43%
SGD	2.0728	21.36%	1.7265	39.93%

all layers, the accuracy declined again, further suggesting overfitting.
These experiments revealed that while ConvNextBase initially appeared more effective during transfer learning, its performance declined after fine-tuning. This suggests it overfit the training and validation datasets, leading to poorer results on the test set.
Confident that EfficientNetV2M was the most suitable pretrained model for our study, we explored its larger variant, **EfficientNetV2L**. This resulted in another improvement, with the accuracy reaching **0.82**.

4.2 Gradient-Based Activation Analysis for Fine-Tuning Phase

To optimize the fine-tuning of the pre-trained EfficientNetV2L model, we used Grad-CAM [3] to visualize activation regions and determine the optimal number of layers to unfreeze. Heatmaps revealed that earlier layers captured generic features like edges, while deeper layers focused on blood cell structures critical for classification. Consequently, we froze the first 700 layers to preserve general feature extraction while adapting deeper layers to our dataset, given the significant differences from ImageNet, which required substantial adaptation of the deeper layers.

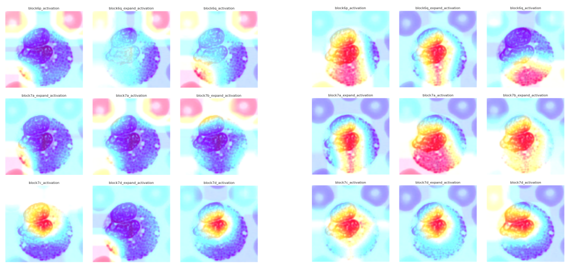

Figure 3: Heatmap comparison in transfer learning and fine tuning

5 Conclusion

5.1 Performance

The results of our final model are as follows:

Train acc.	99.63%
Val acc.	97.66%
Local test acc.	97.39%
Remote test acc.	81.67%

We are satisfied with our model's performance, as the local accuracy values are high. However, we acknowledge the gap between the accuracy scores of the two tests, which may be due to significant differences in the datasets' distributions. To mitigate this issue, we attempted to apply Test-Time Augmentation (TTA), but it did not yield the desired results. Undoubtedly, other approaches could have been explored to address this challenge more effectively.

5.2 Further Developments

To improve our workflow, we recognize the potential benefits of advanced augmentation techniques to enhance the model's generalization. To address class imbalance, we applied class weights (section 2.2) and experimented with oversampling by generating samples for underrepresented classes below the dataset's average threshold. This approach aimed to avoid excessive dataset alterations but proved challenging due to technical constraints. We also experimented with augmentations during training via a generator, but this caused a 0.02 drop in accuracy, likely due to training instability, slower convergence, and uneven augmentation distribution, which hindered the model's ability to learn key patterns effectively.
Taking these considerations into account, along with our resource limitations, and thanks to our continuous collaboration and teamwork, we find the results achieved to be satisfactory.

References

[1] Albumentations-team. Albumentations documentation. 2024.

[2] P. B. David Tellez, Geert Litjens. Quantifying the effects of data augmentation and stain color normalization in convolutional neural networks for computational pathology. 2019.

[3] A. D. Ramprasaath R. Selvaraju, Michael Cogswell. Grad-cam: Visual explanations from deep networks via gradient-based localization. 2019.