AN2DL - First Homework Report
AneuralI

Federico de Introna, Francesco Di Giore, Davide Corradina, Michele Leggieri

fdeintrona, digioref, corrapiano, leggierimichele

251274, 252126, 252117, 244615

July 22, 2025

1 Introduction

From detecting cancerous growths to making autonomous cars possible, **image classification** has been one of most iconic achievements of artificial intelligence.

In its basic form, the task consists of assigning a label from a limited set to a collection of pixels (i.e. an image). Even though it seems trivial to us, it poses a challenge to computers and a number of techniques have been developed to try and solve this problem.

This project focuses on using **deep learning**, specifically *convolutional neural networks* (CNNs), to classify RGB images of blood cells [8]. Deep learning, a subset of machine learning, uses multi-layered neural networks to automatically learn patterns and features from raw data. CNNs are particularly effective for image-based tasks, as their convolutional layers are designed to capture spatial and hierarchical features.

The goal of this project is to accurately classify each blood cell image into the correct category, addressing a **multi-class classification** problem. This has direct applications in medical diagnostics, such as identifying cell states linked to specific health conditions.

By leveraging the power of deep learning, this project aims to demonstrate how artificial intelligence can assist in tasks of clinical importance.

2 Problem Analysis

Understanding the problem is essential for designing an effective solution. This analysis explores the dataset's characteristics, highlights the primary challenges faced during development—such as implementing **data augmentation**—and reflects on initial assumptions, including early decisions that influenced model performance. These insights guided the refinement of the final approach.

The dataset we used consisted of 13759 images of blood cells divided into 8 classes shown in figure 1.

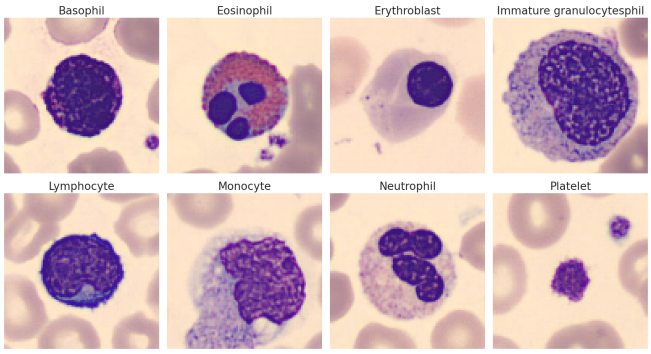

Figure 1: The eight types of blood cells

The dataset consists of RGB images with a resolution of 96x96 pixels, resulting in an input shape of (96, 96, 3).

2.1 Data analysis

While analyzing the data, we discovered that the last 1800 images were not blood cells but repetitions of a frame from the music video of the 1987 hit *"Never gonna give you up"* picturing the singer Rick Astley and a frame from the film Shrek showing the main character. All of them were removed from the dataset.

Furthermore, the balanced distribution of images across classes does not ensure fair representation during training, so we immediately used a form of **class weighting**[1] that assigns to each class a weight of "importance".

Table 1: Application of class weighting

Type of blood cell	N° images	Weight
Basophil	852	1.7557
Eosinophil	2181	0.6857
Erythroblast	1085	1.3771
Immature granulocytesphil	2026	0.7386
Lymphocyte	849	1.7599
Monocyte	993	1.5047
Neutrophil	2330	0.6413
Platelet	1643	0.9094

To enhance the performance of our model, we implemented **data augmentation** [2] as part of the preprocessing pipeline. We decided to use this method because it addresses two critical challenges: improving the model's ability to **generalize** and mitigating overfitting, especially given the relatively small dataset size.

Augmentation was also applied to the validation set to ensure consistency with the training pipeline. This decision aimed to simulate real-world variability and evaluate the model's robustness under augmented scenarios. However, transformations were applied with caution to avoid overestimating validation performance.

2.2 Initial assumption

Initially believed that designing our own architecture would give us complete control over the learning process. However it immediately became clear that our initial model design was underperforming. The lack of pre-trained features and the challenge of training a deep network from scratch resulted in slower convergence and less accurate results. This led us to reconsider our approach, and we ultimately turned to more established architectures and techniques, which significantly improved model performance.

3 Method

For our project we decided to start off by creating our own model architecture but quickly realized that transfer learning yielded the best results. To explore this approach, we started by evaluating four popular CNN architectures: MobileNetV1, InceptionV3, VGG16, ResNet50 [3].

However, after testing these models, we found that their performance was not as strong as anticipated. So, we shifted our focus to more advanced architectures, specifically **EfficientNetB0 [9], ConvNeXtBase, ConvNeXtLarge** [5].

3.1 Architecture

To help the models better adapt to our problem we did not include the top, instead we added our own layers for classification.

Table 2: Our ConvNeXtBase architecture

Layer (type)	Output Shape
InputLayer	(None, 96, 96, 3)
Functional (ConvNeXtBase)	(None, 3, 3, 1024)
GlobalAveragePooling2D	(None, 1024)
BatchNormalization	(None, 1024)
Dropout	(None, 1024)
Dense	(None, 512)
Dense	(None, 256)
Dense	(None, 128)
Dense	(None, 64)
Dropout	(None, 64)
Dense	(None, 8)

3.2 Hyperparameters

Going into more detail we used mini-batch gradient descent with batch size of 64 samples (this batch size was selected to balance computational efficiency and convergence stability), 100 epochs were far enough given an early stopping patience of 10 epochs.

With regard to the **optimizer**, we finally landed on **AdamW** with cosine annealing and warm

restarts as suggested in (I. Loshchilov and F. Hutter, 2019 and 2017) [7, 6] to speed up **convergence**.

3.3 Transfer Learning with Fine-Tuning

We used transfer learning with **fine-tuning** because it leverages pre-trained models that have already learned rich feature representations from large datasets. This approach is particularly effective when working with smaller datasets, like ours, as it reduces training time and improves performance by transferring knowledge from these pre-trained models.

To further optimize the model for our specific task, we applied fine-tuning by unfreezing the entire model except for the batch normalization layers. Batch normalization layers were kept frozen to preserve their learned statistics and avoid destabilizing the training process, which could occur due to drastic changes in the network. This strategy allowed us to adapt the model's higher-level features to our blood cell classification task while maintaining **stability** and accelerating convergence.

4 Experiments

In this section we present the results of our experiments that are summarized in Table 3.

Table 3: Results

Model	Acc.	Prec.	Rec.
MobileNet	0.5769	0.6890	0.121
InceptionV3	0.6580	0.7848	0.169
EfficientNet	0.7272	0.9765	0.253
ConvNeXtBase	**0.8570**	**0.9879**	**0.2776**
ConvNeXtLarge	0.8248	0.9926	0.2643

5 Results

Comparing the results obtained by the trained networks, we noticed that the best accuracy was obtained by the **ConvNeXtBase**, being a more complex network compared to those that precede it as shown in Table 3 and with fewer parameters compared to the Large that was more prone to overfitting due to the size of the dataset.

6 Discussion

The strengths of our network are:

- Preventing overfitting by using **dropout** in the dense part of the network and the GAP layer;
- Adaptive learning rate by the use of the **CosineDecayRestarts** schedule, instead of using a fixed learning rate for all the epochs;
- The general structure of the project, that can be reused with different networks;
- We utilized **Test Time Augmentation** [4] to ensure better predictions.

While possible limitations and weaknesses are:

- Considerable effort is required to properly tune the several **hyperparameters** used in our model;
- Augmenting the validation set is not usually advised.
- Given the size of our network, training with a larger dataset would possibly lead to higher accuracy and generalization capabilities;
- Tune ConvNeXtLarge properly with a larger dataset and a deeper and more complex structure;

7 Conclusions

To conclude, we may suggest some future improvements, directly linked to the weaknesses of our network, that are: the use of a **large dataset**, the inclusion/removal of **dense layers** at the top of the network, the use of an **oversampling technique** instead of class weighting to deal with class unbalancing, the use of a different network for the transfer learning. Nevertheless our network proved to be robust and solid to augmented images and achieved good performances overall on the test set, being a good starting point to develop a network for more complex tasks.

8 Contributors

In this section, we list the contributions of every member to the various tasks.

Member	Contributions
Davide Corradina	Class weighting, Network Training (EfficientNetB0, ConvNeXtBase, ConvNeXtLarge), Evaluation of models, Data Analysis
Federico de Introna	Augmentation techniques, Network Training (Custom Models, EfficientNetB0, ConvNeXtBase-Large), Report Writing, Data Analysis
Francesco Di Giore	TTA, Network Training (InceptionV3, MobileNet, EfficientNetB0, ConvNeXtBase-Large)
Michele Leggieri	Report writing, Network Training (EfficientNet, ConvNeXtBase-Large)

References

[1] `https://scikit-learn.org/dev/modules/generated/sklearn.utils.class_weight.compute_class_weight.html`. Class weighting of scikit_learn.

[2] `https://keras.io/api/keras_cv/layers/`. Keras_cv layers used for Data Augmentations.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. 2015.

[4] M. Kimura. Understanding test-time augmentation. 2024.

[5] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. 2022.

[6] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. *Conference paper at ICLR 2017*, 2017.

[7] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *Conference paper at ICLR 2019*, 2019.

[8] P. H. Rabia Asghar, Sanjay Kumar. Automatic classification of 10 blood cell subtypes using transfer learning via pre-trained convolutional neural networks. 2024.

[9] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. 2020.