AN2DL - First Homework Report
RagingNeurons

Abdullah Nasr, Mara Tortorella, Sabrina Cesaroni, Samuele Vignolo

abiii, maratortorella, sabri, SamueleVigno

1 Introduction

This study addresses a *multi-class image classification problem* using deep learning to classify 96x96 RGB images of blood cells into eight types, aiming to correctly label each image based on its features.

2 Problem Analysis

The dataset provided, "training_set.npz", initially contained 13,759 blood cell images.
The first preprocessing step involved removing *outliers*, images that do not truly represent the cells (Figure 1), resulting in a reduced dataset of 11,959 cells.

Cell	Count	Cell	Count
Basophil	852	Eosinophil	2181
Erythroblast	1085	Granulocytes	2026
Lymphocyte	849	Monocyte	993
Neutrophil	2330	Platelet	1643

As evidenced by the table above, there was a problem of **class imbalance**: the unequal distribution of blood cell images posed a challenge for the model, making it more difficult to learn the features of the underrepresented classes and potentially affecting its overall performance and ability to generalize effectively.

Moreover, there was **limited variety in images:** the lack of diversity in visual features, such as lighting conditions, orientations, and other factors, could prevent the model's ability to generalize and reduce its robustness on different scenarios and datasets.
Furthermore, the **small dataset size** made training a deep learning model particularly challenging, especially in achieving strong generalization. For all the reasons mentioned above, the model was likely to **over fit** specific features.

Figure 1: Outliers

3 Method

3.1 Image Augmentation

Image augmentation was used to enrich the dataset, enhance model generalization, and prepare the data for training. In this project, augmentation was achieved in three ways: increasing the number of images, applying color and geometric transformations along with image combinations, and incorporating an augmentation layer directly into the model during training.

3.2 Drop-out layer

To reduce overfitting, dropout layers were added to the classifier. Dropout randomly deactivates neurons during training, helping prevent the model from relying too much on specific features and encouraging it to learn more general patterns, which

Table 1: Model's results considering **all main experiments** that led to the **best test score**.

Model	Train Accuracy / Val. Accuracy	Precision / Val. Precision	Recall / Val. Recall	AUC / Val. AUC	Online test Accuracy
CNN	99.04/83.40	99.07/83.34	99.04/83.40	100/95.24	≈ 0.23
MobileNetV3Small	91.59/97.03	95.24/95.24	88.60/96.57	99.36/99.86	≈ 0.43
EfficientNetB0	46.68/69.69	73.20/76.39	17.66/69.69	83.86/93.54	≈ 0.60
EfficientNetV2S	75.91/93.23	83.92/93.49	67.67/93.23	96.12/99.54	≈ 0.78
EfficientNetV2M	76.51/96.34	86.29/96.72	66.98/95.91	96.09/99.84	≈ 0.83
ConvNeXtBase	**81.91/96.95**	**90.83/97.10**	**74.56/96.73**	**97.52/99.83**	$\approx$ **0.88**

improves performance on new data.

3.3 Loss Function and Weight Decay

The Categotical Crossentropy loss function was employed:

$$L = -w_k \cdot \sum_{i=1}^{C} y_i \cdot \log(p_i) + \frac{\lambda}{2} \sum_{j=1}^{N} \theta_j^2 \qquad (1)$$

where w_k is a weight factor for the k-th class which can be used to assign more importance to certain classes during training. This is especially useful in imbalanced datasets, where some classes may be underrepresented. Assigning higher weights to underrepresented classes helps mitigate bias towards more frequent classes. C is the total number of classes, y_i is the true probability distribution for the i-th class, where $y_i = 1$ if the sample belongs to class i and $y_i = 0$ otherwise. p_i is the predicted probability for class i obtained from the model's output after applying softmax. N is the total number of parameters, θ_j is the j-th weight in the model and λ is the regularization factor.

3.4 Optimizer

The *Lion optimizer* was chosen based on the analysis conducted by [2], which demonstrates that Lion significantly outperforms Adam on various architectures. The update rule of Lion for minimizing a loss $f(x)$as described in [1] is:

$$m_{t+1} = \beta_2 m_t - (1-\beta_2)\nabla f(x_t) \qquad (2)$$
$$x_{t+1} = x_t + \epsilon(\text{sign}(\beta_1 m_t \beta_1)\nabla f(x_t)) - \lambda x_t) \qquad (3)$$

where $m_t \in \mathbf{R}^d$ is momentum, $\epsilon > 0$ is the learning rate, $\beta_1, \beta_2 \in [0,1]$ are two momentum related coefficients, and $\lambda \geq 0$ is a weight decay coefficient. A default value of $\beta_1 = 0.9$ and $\beta_2 = 0.99$ was suggested in [2], with which the Lion update rule can be written directly as:

$$x_{t+1} \leftarrow (1-\epsilon\lambda)x_t - \epsilon\,\text{sign}\Big((10+1)g_t + 0.99g_{t-1} + + 0.99^2 g_{t-2} + \cdots + 0.99^k g_{t-k} + \dots\Big) \qquad (3)$$

where $g_t = \nabla f(x_t)$.

4 Experiments

4.1 Convolutational Neural Network with no pre-trained

Convolutional neural networks (CNNs) were chosen because they perform particularly well with image-based inputs compared to other types of neural networks. The first approach used a custom CNN with two convolutional layers, each followed by *ReLU activation* and pooling layers. The output was then flattened into a one-dimensional vector, passed through a dense layer, and finished with a *softmax activation* to produce class probabilities. However, the model showed clear signs of over fitting, as it performed poorly during online testing. This led to the decision to switch to pre-trained models, which offered a more reliable starting point for the task. Moreover, in this early stage of the study, no augmentation was added to the dataset nor to the network.

4.2 Transfer Learning and Fine Tuning with different pre-trained models

The pre-trained models tested were MobileNetV3Small, EfficientNetB0, EfficientNetV2S, EfficientNetV2M and ConvNeXtBase. *Transfer learning* alone does not work well for classifying blood cells because pretrained models are not designed for this specific task, leading to poor performance. To improve results, some layers of the pretrained models were *fine-tuned* using the augmented training set created earlier. Among the tested models, the one with the ConvNeXtBase pretrained network performed the best in this study.

4.3 Augmentation pipeline

At the beginning, basic augmentation layers were applied only during training, including geometric adjustments, image mixing, and color modifications. It was later found that using more diverse and substantial augmentations significantly improved test performance. Experimentation led to the identification of the most effective augmentation pipeline. To ensure a balanced dataset, the number of images per class was increased to approximately 5500. This was accomplished using `ImageDataGenerator` class from the Keras library, which allows augmentation by generating new images with slight geometric modifications. The dataset was then passed through the pipeline illustrated in Figure 2, where random augmentation layers, taken from Keras CV, were applied to the training set. The augmentation pipeline was selected by balancing two key objectives: enhancing generalization to improve model robustness and preserving image integrity to ensure representativeness of their respective classes.

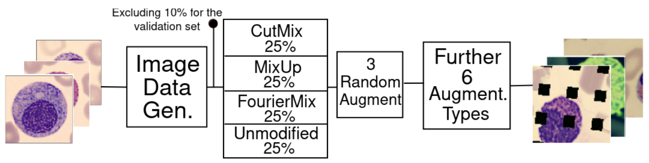

Figure 2: Data Augmentation Pipeline

5 Results

The dataset was mostly split into training and validation sets, with 90%-10% or 80%-20%, respectively. Table 1 shows the best results obtained for each pre-trained model, based on the various experiments conducted for each model, including different classifiers, augmentation pipelines, learning rates, optimizers, and the number of unfrozen layers during fine-tuning, among other factors. In most of the tested models, validation accuracy was significantly higher than training accuracy. This occurred because the validation data underwent extreme augmentation, while the training set data was left plain. This suggests that the augmentation layers still allow the training of a model that performs well on the original blood cell images. The two key factors that had the biggest impact on improving test accuracy were strong augmentations and the choice of the pre-trained model. Specifically, the results showed that more extreme augmentations and more complex networks, like EfficientNetV2S and EfficientNetV2M, led to better test accuracy.

6 Discussion

Due to the many factors that influence network construction not all experiments were conducted across every model. For example, when a particular augmentation pipeline improved (or degraded) performance on one pre-trained model, it was assumed that it would have a similar effect on other models of the same type. In the three best-performing models, which used the optimal augmentation pipeline, a good balance was achieved between recognizing the original cells and avoiding overfitting. However, increasing training accuracy beyond a certain point proved challenging, leading to the decision to limit training at almost 80% accuracy.

7 Conclusions

The best model, which has ConvNeXtBase achieved an accuracy of *88%* on the provided testing set. As already said the factors that had the most impact on the model's performance were image augmentation and the choice of the pre-trained model. This trend suggests that enhancing the variety of training data through augmentation, along with using more sophisticated models, helps the model generalize better, resulting in higher performance on test datasets. With more resources available, larger pre-trained models, such as ConvNeXtLarge or ConvNeXtXLarge, could be considered, as they might significantly improve the model's performance.

8 Contributors

The division of tasks was organized in order to be able to work in parallel.

8.1 Abdullah Nasr

Tests with different augmentation techniques, tests with CNN and ConvNeXtBase, report

8.2 Mara Tortorella

Augmentation performance testing, tests with MobileNetV3Small, papers/report

8.3 Sabrina Cesaroni

Augmentation performance testing, tests with EfficientNetV2S, papers/report

8.4 Samuele Vignolo

Parameters tuning, tests with EfficientNetV2M, EfficientNetB0, report

References

[1] L. Chen, B. Liu, K. Liang, and Q. Liu. Lion secretly solves constrained optimization, as lyapunov predicts. 2024.

[2] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, Y. Liu, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le. Symbolic discovery of optimization algorithms. 2023.