AN2DL - First Homework Report
regionaleveloce

Simone Calzolaro, Gabriele Clara Di Gioacchino, Tommaso Galimberti, Sara Massarelli

simonecalzolar0, gabboclaa, tomgalimberti, saramassarelli

260121, 259518, 259629, 249387

November 24, 2024

1 Introduction

The project aim was to classify 96x96 RGB images of blood cells in eight different classes, each representing a particular kind of cell, using deep learning techniques. We started using a Convolutional Neural Network and then we decided to shift our focus to Transfer Learning in order to achieve better results. Our main goals were:

- Accurate image classification: To correctly classify the given images.
- Efficient training process: Optimize the training process to be compatible with the limited computational power at our disposal.
- Model evaluation and validation: To consistently assess model's performance using standard evaluation metrics such as confusion matrix.

2 Dataset Analysis

The cells dataset available to us consisted of 13,759 images from 8 different categories. A dataset of such limited size exposes the training of any model to several risks: overfitting on the training set, inability to generalize learned features, and failure to converge with larger models. Recognizing the limitations of the available data, we knew from the beginning that augmentation techniques and fine-tuning would be essential.

The dataset was cleaned using a hashing technique, which allowed us to identify and remove duplicate images and outliers. The cleaned dataset had a **shape** of (11,951, 96, 96, 3), and the 8 classes were slightly unbalanced.

We acknowledged this and introduced a `class_weights` parameter to rebalance the classes during training.

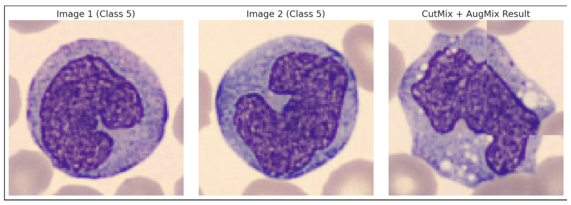
3 Methods and Training Techniques

Using our theoretical knowledge as a compass, we adopted a learning-by-doing approach for the construction of our classifier. We experimented with different models, as detailed in the next section. For each model, it was equally important to experiment with and tune parameters as well as the accompanying techniques.

In this section, we will discuss the techniques and strategies we used to optimize our model.

3.1 Data augmentation

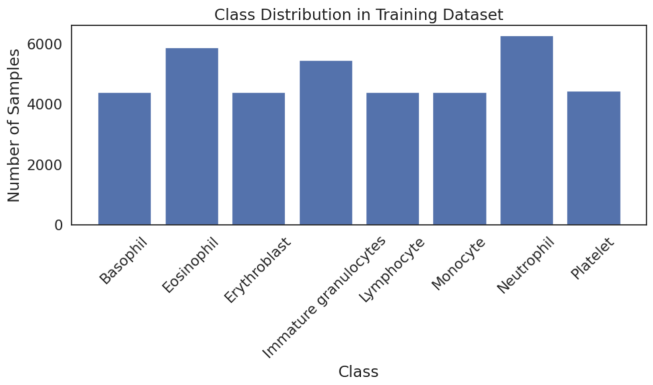
We initially implemented an online augmentation pipeline, introducing random rotations, translations, and adjustments to brightness and contrast. However, this approach yielded limited performance improvements due to persistent overfitting. To mitigate this issue, we transitioned to an offline augmentation strategy, which not only reduced training time but also enhanced the effectiveness of our augmentations. Specifically, we employed advanced techniques such as CutMix and AugMix.



These methods involved combining two randomly selected samples from the same class with minor random rotations to modify pixel positions, effectively creating more diverse and robust training examples. For under-represented classes, we applied targeted augmentations to increase their sample size to 90% of the most represented class. For the remaining classes, we augmented up to 50%

of the size of the most represented class. This balanced augmentation strategy aimed to address class imbalance and improve the generalization of the model.

This approach improved performance by generating diverse samples that retained class-defining features. Finally, we enhanced dataset robustness by applying three random augmentations per sample using RandAugment. The resulting balanced and robust dataset provided the boost needed to improve rankings.



3.2 Optimizers

Adam (Adaptive Moment Estimation) was our initial optimizer choice due to its widespread use and computational efficiency. While it provided satisfactory results, the model exhibited a slight tendency to overfit, prompting us to explore alternatives.

AdamW (Adam with Decoupled Weight Decay) addressed a key limitation of Adam by separating weight decay from gradient-based updates. This switch improved hidden-test performance, enhancing both generalization and robustness.

Lion (EvoLved Sign Momentum) a newer optimizer employing sign-based updates for gradients and momentum, demonstrated faster convergence and promising results. However, its performance proved unstable with minor changes to the network structure. Ultimately, AdamW was chosen for its consistency and reliability.

3.3 Learning rate schedulers

Plain scheduling, where the learning rate remains constant, provided a good baseline. **Exponential decay** improved convergence and accuracy. By gradually reducing the learning rate, it helped stabilize training in later stages and approach the minima. However, the most performant results were achieved using **cosine annealing**, which adjusts the learning rate cyclically, promoting smoother convergence and better generalization. This strategy proved particularly effective in avoiding local minima and maintaining stability throughout training.

3.4 Fine Tuning

Our initial fine-tuning approach involved training only the head of the model, keeping the pre-trained backbone frozen and updating only the final convolutional layers (excluding normalization layers). As we progressively unfroze more layers, we observed a clear improvement in performance, with better results as more layers were included in training.

We implemented a gradual unfreezing strategy, where a fixed number of layers were unfrozen, trained to convergence, and repeated until the entire network was trainable. While this method yielded the best results, it was abandoned in later stages due to its high computational cost.

Interestingly, contrary to common recommendations, fine-tuning the entire network, including normalization layers, produced better outcomes for larger models. Before training the unfrozen model, we stabilized the classifier head by training it alone for three epochs with Xavier weight initialization, which improved performance. During this phase, we used a higher initial learning rate (1e-4) to guide the weights effectively, while the fine-tuning phase used a lower learning rate (1e-5) to avoid destabilizing the pre-trained backbone.

3.5 Data Preprocessing

When using pretrained models, we applied the appropriate preprocessing steps to the inputs, including rescaling and resizing.

4 Models

To arrive to our final solution we conducted a series of experiments. In this section we will summarize the path we followed through different and progressively more powerful models. The results are collected in the table below.

4.1 Convolutional Neural Networks from scratch

We started from what we saw in class, so we used a baseline model adopting the classical architecture of a modern CNNs. The results obtained from this baseline model were suboptimal, leading us to explore transfer learning as an alternative strategy. From this experiment, we were able to retain the classifier head, which was then used with the following transfer learning model. The head consisted of a dropout layer with a dropout rate of 0.3, followed by two dense layers and an output layer with a softmax classification. The number of neurons in the dense layers was adapted based on the imported network's dimensions, starting from (128, 64, 8) to (512, 256, 8).

4.2 Transfer Learning

We experimented with various pre-trained models, including VGG16, EfficientNetB2 and MobileNet . While many of these models performed exceptionally well on our local validation set, only a few achieved good results on the hidden dataset provided by the competition.

This discrepancy was likely due to overfitting, where the models learned patterns too specific to the training or local validation data but failed to generalize effectively to unseen data. To address this issue, we identified that the models were not learning robust features. As a corrective measure, we implemented more 'aggressive' data augmentation techniques, as mentioned in the dedicated section, in order to improve the model's generalization ability and help it perform better on diverse and unseen examples.
ResNet50 stood out as a strong performer after applying these improvements. Its deep architecture and residual connections enabled it to extract meaningful features more effectively. However our choice shifted to a ConvNeXt small since we noticed that it was giving a better performance compared to the ResNet50. It incorporates several key design innovations, drawing inspiration from the latest advancements in vision models, particularly in terms of scalability, feature extraction, and generalization [1]. When combined with our data augmentation strategy, this model enabled us to achieve a score of 0.89 on the hidden test set. This result was not surprising, as ConvNeXt is fundamentally built upon ResNet50, designed to address the limitations of traditional ConvNets while preserving their simplicity and efficiency, combined with our augmentation that was builded and tested meticolously in order to provide an effective advantage without confusing the model.

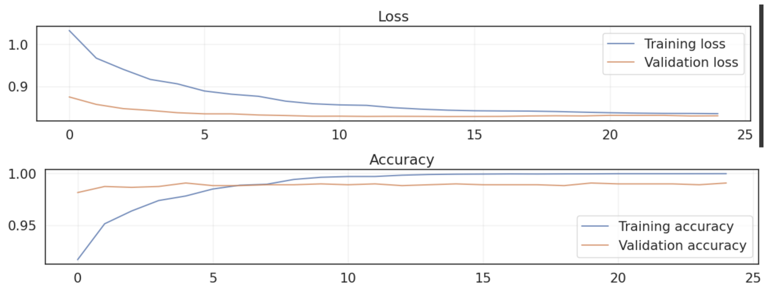
5 Final Model

The best-performing model was configured as follows:

- **Backbone:** ConvNeXt Large.
- **Classifier Architecture:**
  - A Dense layer with 512 neurons, followed by a Dropout layer (rate = 0.4).
  - A second Dense layer with 256 neurons, followed by another Dropout layer (rate = 0.4).
  - An output Dense layer with 8 neurons for the final classification.
- **Data Augmentation:** As described in the relevant section.
- **Optimizer:** AdamW.
- **Fine-Tuning Strategy:** As detailed in the dedicated section.
- **Batch Size:** 128.
- **Learning Rate Schedule:**
  - Initial learning rate of 10^{-4} for weight alignment.
  - Learning rate reduced to 10^{-5} during finetuning.
  - Both phases utilized a Cosine Decay Restart schedule.
- **Weight Initialization:** Glorot Normal.
- **Regularization:**
  - L2 regularization with $\lambda = 10^{-4}$ applied to the output layer.
  - Dropout regularization as specified in the classifier architecture.

Performance Metrics:

- Final local validation accuracy: **99.41%**.
- Final hidden test accuracy: **92.00%**.



6 Contributions

We all contributed on training and testing the models, here more details are included.

Tommaso Galimberti inspected and cleaned the dataset removing the duplicates and the outliers.
Gabriele Clara Di Gioacchino worked on testing various types of augmentations.
Sara Massarelli studied and tested various types of pretrained models in order to find the most suitable for our task.
Simone Calzolaro worked on generating new images for the augmentation and refined the parameters and layers for the models.

References

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

Table 1: Models performances. Here we report the interesting experiments.

Model	Local Validation Accuracy(%)	Codabench Hidden Test Accuracy(%)	Main Updates
MobileNet	89.7	41	No fine tuning
ResNet50(1)	89.8	38	New model
ResNet50 (2)	95.9	48	Fine tuned (last 40 layers)
ResNet50(3)	97.74	52	+Gradual unfreezing
ResNet50(4)	91.81	55	+image data generator
ConvNeXtSmall(1)	90.13	63	New model
ConvNeXtSmall(2)	98.08	77	New augmentated dataset (45k+ images)
ConvNeXtSmall(3)	97.99	89	Total unfreezing of layers
ConvNeXtLarge	**99.41**	**92**	Same architecture as ConvNeXtSmall(3)
EfficientNetB2	97.07	51	Without aggressive data augmentation
EfficientNetB7	97.07	80	