Challenge ANNDL

PoliMi Dropouts

Mattia Piccinato - 10717274
Matteo Salari - 10794158
Davide Salonico - 10774487
Federica Topazio - 10985760

1 Introduction

The goal of this challenge is to tackle a multi-class classification problem, distinguishing peripheral blood smears across eight classes, each representing a particular smear type: basophils, neutrophils, lymphocytes, monocytes, eosinophils, erythroblasts, immature granulocytes, and platelets.

In this report, we will outline the process behind the development of our deep learning model, examining both the data provided to us and the methodology adopted throughout the challenge.

2 Dataset

2.1 Analysis and Pre-Processing

We conducted a thorough examination of the dataset, revealing 13758 images related to pheripheral blood smears. Closer inspection identified 1799 duplicate or poorly informative images, which were excluded using a Python script to enhance the performance of our preliminary model.

We also examined the class distribution of labels in the dataset and tested various strategies to improve model accuracy, including uniform distribution and selective augmentation of the least frequent classes. Models trained with the original class distribution consistently performed best (usually improving accuracy by 1% to 5% depending mainly on the model and the data augmentation pipeline), likely because it reflects real-world scenarios, enabling the model to prioritize common features while recognizing rarer classes.

2.2 Data Augmentation

To enhance the robustness and diversity of our training dataset, we implemented data augmentation from the outset. This involved applying various transformations to generate new variations of the original images, improving the model's ability to generalize across different conditions.

Our augmentation pipeline includes random transformations and conventional techniques like rotation and flipping. Layers such as `RandomContrast`, `RandomColorDegeneration`, and `RandomBrightness` were tuned to avoid excessive adjustments, ensuring label fidelity by preserving essential features like shape and texture. Gaussian noise was also applied to improve model robustness without overemphasizing color variability.

Additionally, we customized the `keras_cv` `RandAugmentation` pipeline to dynamically and randomly select transformations for each image, ensuring that these augmentations enriched the dataset while preserving the semantic consistency of the labels. Importantly, these randomly applied transformations were designed to operate independently from the fixed augmentations in our pipeline, such as rotation and flipping. This separation allowed us to introduce diverse, unpredictable variations into the dataset without compromising the systematic augmentations essential for maintaining key morphological and structural features of the blood cells. With our final augmentation pipeline managed to improve VGG-based model from 0.61 to 0.72 accuracy score on the private test set (first phase) and from 0.88 to 0.93 for a ConvNeXt-based model on the same test set. For further details, refer to our Data Pipeline and Augmentation notebook.

2.3 Augmentation Parameters Tuning

To maximize the impact of data augmentation on model generalization, we fine-tuned augmentation parameters and adopted a technique inspired by Test Time Augmentation. This involved generating multiple augmented versions of each image in the local test split and averaging the model's predictions to simulate diverse scenarios and assess which transformations improved accuracy.

Through iterative trials, we found that transformations altering color properties, such as `RandomContrast`, `RandomColorDegeneration`, and `RandomBrightness`, could degrade performance by reducing the clarity of important smears features. Moreover, misprediction analysis with Grad-CAM revealed instances where these transformations led the model to focus on irrelevant regions, impacting

accuracy. To address this, we adjusted parameters to preserve key attributes like smears shape and texture, balancing transformation diversity with label fidelity.

2.4 Other methods

We explored additional augmentation techniques such as MixUp, CutMix, and MixCut to enhance performance. However, these methods did not yield any significant improvement in model accuracy, so we prioritized other augmentation strategies.

3 Models

3.1 Architecture Exploration

At the start of our architecture exploration, we reviewed current literature on cells image classification and state-of-the-art deep CNNs, building models from scratch using Keras layers like Conv2D and MaxPooling2D (e.g. [DD23] or the one we called VanillaCNN when reporting results). While nearly every model achieved high accuracy on the provided dataset, significant performance differences emerged on the private test data, likely due to specific transformations.

To estimate actual model accuracy, we employed a method inspired by Test Time Augmentation (TTA), identifying augmentation parameters that could enhance performance on the private dataset, providing a pseudo-validation step.

Not all models adapted equally well to the more challenging classification task. The top-performing architectures, ranked in increasing order, were from the VGG, EfficientNet, and ConvNeXt families. The latter two likely outperformed VGG due to their integration of Batch Normalization layers, which we did not add manually to the VGG models.

A final Softmax layer with 8 neurons was obviously added to any architecture.

3.2 Transfer Learning

Since the given dataset appeared not to be sufficient, we thought about exploiting the knowledge of models pre-trained on other image datasets through Transfer Learning, and initializing with pre-trained weights from the ImageNet dataset proved indeed effective with all the architectures we adopted.

In general, we imported the Feature Extraction half leveraging Transfer Learning, while implementing the classifier trivially, by manually adding a dense layer on top, right before the Softmax layer. This configuration provided the best performance for us, probably because this is a simple classification task and a single dense layer was probably enough.

3.3 Test Time Augmentation

To further regularize our model, we implemented Test Time Augmentation (TTA). Although TTA adds considerable time, increasing the number of test images by a constant factor and extending inference time, it provided useful insights. Due to the Codabench verifier's 6000-second limit, we used only three augmentations per test image, and only with our VGG-based model. As a result, the accuracy score worsened by 2% but the strong limitations imposed by the server make difficult to asses if this could lead to an actual improvement of the model. However, users can enable TTA in our main notebook by setting `tta = True` and specifying `tta_steps` to the desired value.

3.4 Hyperparameter Tuning

Given the high dimensionality of the hyperparameter space, combined with limited computational resources and relatively long training times, we adopted a focused approach to hyperparameter tuning. Instead of an exhaustive search, we selected a subset of critical hyperparameters that would most significantly impact model performance, and, through iterative testing, we manually optimized these parameters.

3.5 Other methods

While initializing with pre-trained weights from the ImageNet dataset proved effective, fine-tuning the final layers of the network did not lead to improved results. We hypothesize that this is due to the substantial differences between the ImageNet dataset and our blood smears dataset. Consequently, the early features learned on ImageNet, when frozen during training, may not align well with the specific requirements of this task.

Finally, we explored the use of L2 regularization and additional Dropout layers. However, these techniques did not yield any further improvements in performance and were therefore excluded from the final model configuration.

4 Training

4.1 Methods

We decided to use a training, validation, and test split of 0.8, 0.1, and 0.1, respectively. After experimenting with various splits, this configuration consistently yielded the best results by ensuring sufficient training data while retaining a meaningful validation set for hyperparameter tuning and a test set for final evaluation.

To further optimize training, we have incorporated callbacks such as `EarlyStopping` and `ReduceLROnPlateau`, which dynamically adjusted hyperparameters like learning rates. `EarlyStopping`

monitored validation accuracy metric to halt training when improvements plateaued, preventing overfitting; we tried with several other metrics, such as Recall and Loss, but results did not vary significantly. On the other hand, `ReduceLROnPlateau` reduced the learning rate when validation loss stagnated, enabling more precise convergence to optimal parameters.

The batch size was set to the biggest value which could be handled by your development environment, and we found that it could be either set to 128 or 256.

4.2 Optimizer

Across different model architectures, our experiments consistently demonstrated that Nestorov-Adam optimizer, when training with optimized initial learning rates and weight decay parameters, consistently outperformed other widely used optimizers such as standard Adam, SGD, and RMSprop. AdamW was also considered, but it provided similar performance to NAdam.

4.3 Challenge-Driven submissions

Completely aware that it is generally not considered a best practice for training deep neural networks, to maximize the amount of data available for training, we opted to omit the test split during training, using only training and validation splits when submitting a model. This allowed us to train models on a larger portion of the dataset, enhancing their ability to generalize and achieving better results on the private test. Validation data was still retained to monitor performance and prevent overfitting during training.

4.4 Other methods

To enhance model robustness and reduce variance caused by specific data splits, we implemented K-Fold Cross-Validation, which can be enabled in our main notebook by setting `k_fold = True` and specifying the desired number of splits with `splits`. Using five splits, we successfully mitigated variance; however, we were unable to evaluate accuracy for heavier models due to RAM constraints in our development environment.

Additionally, we implemented an AdaBoost-based approach, aiming to leverage the ensemble of weak classifiers to improve performance. While this method allowed us to match the results of our preceding models, it failed to outperform them, as the inherent simplicity of the weak classifiers in AdaBoost and their limited capacity to capture the complex features of blood smears images likely constrained its effectiveness. This method was thus also not included in our final submission.

5 Results

The following table shows some of our most relevant submissions (not in chronological order).

Net	Optimizer	Other notes	Score
VanillaCNN	Adam	Wide Data Augmentation	0.31
VGG16	NAdam	RandomAug + NO TEST SPLIT	0.72
VGG16	NAdam	Wide Data Augmentation + RandomAug + NO TEST	0.81
DenseNet	NAdam	Wide Data Augmentation + RandomAug + NO TEST SPLIT	0.77
EfficientNetV2B3	NAdam	Wide Data Augmentation + RandomAug + NO TEST SPLIT	0.80
ConvNextTiny	NAdam	Wide Data Augmentation + RandomAug + NO TEST SPLIT	0.88
ConvNextTiny	AdamW	Wide Data Augmentation + RandomAug + NO TEST SPLIT	0.88
ConvNextTiny	NAdam	Wide Data Augmentation + RandomAug + NO TEST SPLIT+ Preserve Original Distribution of Training Data	0.90
ConvNextBase	NAdam	Wide Data Augmentation + RandomAug + NO TEST SPLIT + Preserve Original Distribution of Training Data	0.93

Figure 1: Some relevant submissions

The table below shows the value of some metrics for our final model when using the test split.

Accuracy	Precision	Recall	F1
0.9427	0.9419	0.9355	0.9385

Table 1: Results on our Test

The following graph below shows the values of accuracy and loss of our final model during the training phase.

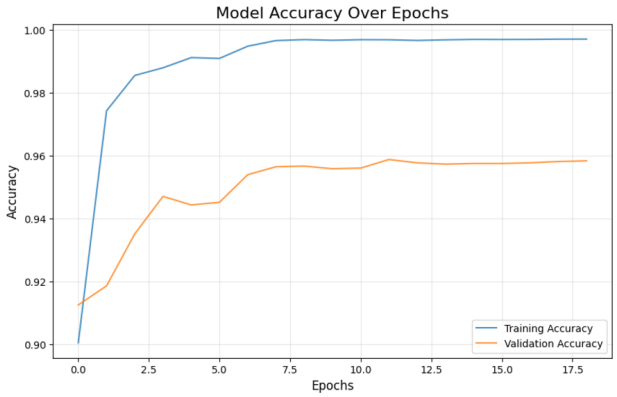

Figure 2: Accuracy history

We also report a brief comparison between different architectures tested on the local test split with TTA.

Net	Optimizer	Score
ConvNextTiny	AdamW	0.95
EfficientNetV2B3	Nadam	0.94
ResNet	Nadam	0.75
InceptionV3	Nadam	0.73
DenseNet	Nadam	0.89

Figure 3: Some local results obtained using TTA

6 Contributions

In our collaborative project, each team member played a crucial role. Federica Topazio, with a focus on literature review, delved into various papers [DD23] to guide our model selection process, while Davide Salonico experimented different techniques to reproduce private dataset scores locally to support our choices and assist our entire development process. On the other hand, Matteo Salari played a pivotal role in the Data Augmentation domain, offering valuable insights thanks to his thorough study of [WTS+22] documentation, and, finally, Mattia Piccinato constantly dedicated to data information quality, architecture choice and hyperparameters optimization, leading us to the best-performing parameters of models and optimizers. We care to clarify that even tough we mainly focused on diverse topics, each member was involved in each single choice, often discussing for hours for motivating one's suggestions.

References

[DD23] Karnika Dwivedi and Malay Kishore Dutta. Microcell-net: A deep neural network for multi-class classification of microscopic blood cell images. *Expert Systems*, 40(7):e13295, 2023.

[WTS+22] Luke Wood, Zhenyu Tan, Ian Stenbit, Jonathan Bischof, Scott Zhu, François Chollet, Divyashree Sreepathihalli, Ramesh Sampath, et al. Kerascv. https://github.com/keras-team/keras-cv, 2022.