POLITECNICO
MILANO 1863

ANNDL - First Homework Report
Neural Network November

Bettiati Matteo, Bianchi Lorenzo, Ostidich Francesco

betti, lolly, kello

258730, 259946, 259863

November 24, 2024

1 Introduction

This project[7][3] focuses on developing a neural network for multi-class classification of images representing eight distinct blood cell states. The main challenges included addressing class imbalance, generalizing from a relatively small dataset, and ensuring accurate classification. To tackle these, we conducted extensive exploratory data analysis (EDA) to identify and resolve outliers, ambiguities, and class imbalances. A multi-stage data augmentation pipeline, culminating in dynamic augmentation during training, was employed to enhance model generalization.

We adopted transfer learning, leveraging pretrained architectures to accelerate learning and adapt the models to the dataset's specific features. Model optimization involved hyperparameter tuning, selective layer unfreezing, and mixed precision training. Our final submission achieved a local accuracy of 96% and an online evaluation accuracy of 92%.

2 Dataset Processing

The raw dataset consists of total of 13759 images designed for the classification of different types of blood cells. Each image is labeled with one of eight classes, representing various blood cell types: Basophil, Eosinophil, Erythroblast, Immature granulocytes, Lymphocyte, Monocyte, Neutrophil and Platelet. Images size is 96x96, color space is RGB (3 channels), with *uint8* data type.

2.1 Exploratory Data Analysis (EDA)

The first step in the pipeline was performing exploratory data analysis (EDA). We began by plotting a grid sorted by class, which helped us identify two outliers: one image with a Shrek watermark appearing in multiple classes, and another with a Rickroll watermark in the Monocyte class. Both were easily removed as repeated, identical images.

An analysis of class distribution revealed a mild class imbalance, which we addressed through image duplication and data augmentation to ensure more balanced class representation.

We also identified some ambiguously labeled images. Since no images were clearly mislabeled and considering the dataset's size, we chose to retain these ambiguous samples to preserve the dataset's diversity, which can help improve model robustness.

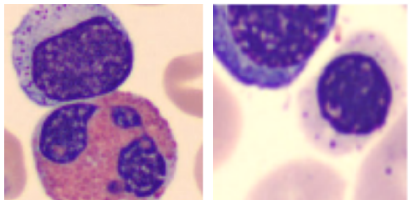

Figure 1: Ambiguous samples n°1 and n°2

2.2 Data Augmentation

To address class imbalance, we initially used the *class_weights* argument in *model.fit()* to balance class contributions. However, after balancing the dataset through sample duplication and augmentation, class weighting became unnecessary.

Data augmentation played a crucial role in improving model performance. The pipeline was refined in three stages:

1. **Initial Augmentation**: Basic transformations (flip, rotation, brightness, contrast, hue, etc.) were applied, expanding the dataset to 2,500 images per class.

2. **Refined Augmentation**: The second round introduced additional functions (grid mask, color degeneration, cutout, shear), Models were retrained on a larger dataset (5,000 images per class, including the augmented test set).

3. **Dynamic Augmentation**: The final method used real-time augmentation during training with *TensorFlow Dataset* and *keras_cv*[1] layers like Augmix[4], RandomZoom, AutoContrast and RandAugment[2]. wrapped in RandomAugmentationPipeline of *keras_cv* This approach improved generalization but increased training time.

3 Model experiments

Each team member began by selecting a pre-trained model, and while tweaking hyperparameters and top layers structure, we started to look for the best resulting ones to use.

We started with relatively small models, like *MobileNetV2*, a model known for its favorable trade-off between parameter count and accuracy, as the foundation for our experiments, to leverage fast training and less overfitting risk given the small dataset.

Following this initial setup, we proceeded to design the top layers architecture. Overall, while the specific structures varied slightly between attempts, the core design remained consistent.

Layer type
Input layer
Pre-trained base model
Global average pooling 2D
Normalization (Group or Batch)
Dropout (0.2 to 0.4)
Dense (128 to 1024 neurons, ReLU or GeLU)
Normalization (Group or Batch)
Dropout (0.2 to 0.4)
Dense (64 to 512, ReLU or GeLU)
Normalization (Group or Batch)
Dropout (0.2 to 0.4)
Output layer

For each dense layer we experimented also with kernel regularizers, with an average value of 1e-3.

By the end we tried a wide range for all parameters as choices like dataset size, complexity, model size and optimizers varied a lot through the experiments. Here after a recap with the most used ones.

Parameter	**Toptraining**	**Fine tuning**
Learning rate	1e-3 to 1e-4	1e-5 to 1e-6
Weight decay	1e-1 to 1e-3	1e-1 to 1e-3
Epochs	10 to 20	20 to 30
Batch size	32 to 512	32 to 512

We experimented with two optimizers, *Lion* and *AdamW*, tuning their hyperparameters for optimal performance. *Lion* performed best with a low learning rate and high weight decay, while *AdamW* favored a higher learning rate and lower weight decay.

Extensive training test runs were conducted to identify the best model parameters based on local test set accuracy. Although the final model used alternative parameters, the hyperparameter tuning provided valuable guidance, with detailed results available in the "*misc/hyperparameter_tuning.pdf*" file.

3.1 Transfer Learning

The transfer learning process was structured into two phases: training the top layers followed by fine-tuning. The first phase primarily focused on tuning the top layers' architecture and parameters. To enhance training efficiency, we incorporated callbacks such as early stopping, learning rate schedulers, and checkpoints. Training was conducted on Colab[5] and Kaggle[6] GPUs, utilizing mixed precision and full RAM dataset loading. When necessary, we employed the *TensorFlow Dataset*[9] class to optimize

RAM usage and address resource limitations.

3.2 Fine Tuning

Fine-tuning began once performance plateaued during top layer training. This involved unfreezing base model layers, lowering the learning rate, and adjusting weights to fit dataset-specific patterns. To minimize instability, normalization layers were kept frozen, following Keras[1] Transfer Learning best practices.

Initially, all layers were unfrozen at once, but we adopted gradual unfreezing. This approach preserved the pre-trained model's general knowledge, reducing the risk of catastrophic forgetting and overfitting[8]a Early layers, capturing fundamental features, remained stable to ensure smoother fine-tuning.

4 Comparison overview

The initial model was trained and tested on the early-stage augmented datasets. Through progressive enhancements to both the datasets and the model architecture, we achieved a steady improvement in accuracy throughout the challenge.

4.1 Base model evaluation

Initially, we selected a set of pre-trained models to evaluate as foundations for our end-goal model. Below, we summarize the key metrics observed during this evaluation phase, which ultimately led us to focus on the *MobileNet*, *EfficientNet*, and *ConvNeXt* models.

The reported model accuracies correspond to the local test accuracy and the Condabench development phase accuracy.

Model Name	Local Accuracy	Online Accuracy
Early Stage Models		
MobileNetV3	0.98	0.36
InceptionV3	0.98	0.40
Middle Stage Models		
EfficentNetV2M	0.96	0.60
MobileNetV3Large	0.98	**0.74**
Late Stage Models		
ConvNeXtTiny	0.97	**0.86**
ConvNeXtSmall	0.98	0.85

5 Final model

For our final submission model, we achieved the best performance by utilizing the previously described dynamical data augmentation technique. The model was trained with a batch size of 512, a learning rate of 1e-3, a weight decay of 1e-4, and the AdamW optimizer. The model architecture is summarized in the following table:

Layer	Description
Input layer	`input_shape`
Pre-trained base model	`convnext_tiny`
Global average pooling 2D	`GlobalAveragePooling2D`
Normalization 1	`groups=32, axis=-1`
Dropout 1	`p=0.2`
Dense 1	`512, act='gelu', kernel_reg=l2(0.001)`
Normalization 2	`groups=32, axis=-1`
Dropout 2	`p=0.2`
Dense 2	`128, act='gelu', kernel_reg=l2(0.001)`
Normalization 3	`groups=32, axis=-1`
Output layer	`num_classes, act='softmax'`

This model achieved a end top-layers training validation set accuracy of **81%**.

5.1 Fine tuning approach

Fine-tuning was performed unconventionally by unfreezing the entire model at once, rather than progressively unfreezing layers. This approach enabled faster convergence, reduced overfitting, and allowed for simultaneous adaptation of all layers, enhancing the model's ability to learn complex patterns tailored to the task, ultimately improving accuracy and generalization.

The submission of this final model made us able to achieve a final local accuracy of **96%** and an online one of **92%**

6 Contributions

Each team member contributed equally, with Bettiati working on environment setup and model training, Bianchi on model building, dataset cleaning and augmentation, and Ostidich on data augmentation and hyperparameter tuning.

References

[1] F. Chollet. Keras: The python deep learning library, 2015. Accessed: 2024-11-24.

[2] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. Randaugment: Practical automated data augmentation with a reduced search space. *arXiv preprint arXiv:1909.13719*, 2019.

[3] P. di Milano. Artificial neural networks and deep learning (polimi, ay 2024/2025): course slides, 2024. Available from the course platform or upon request.

[4] D. Hendrycks, S. Zhao, A. Basart, D. Song, and J. Steinhardt. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*, 2019.

[5] G. Inc. Google colaboratory, 2024. Accessed: 2024-11-24.

[6] K. Inc. Kaggle: Your home for data science, 2024. Accessed: 2024-11-24.

[7] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[8] S. Overflow. What is the right way to gradually unfreeze layers in neural network while learning?, 2020. Accessed: 2024-11-24.

[9] TensorFlow. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. Accessed: 2024-11-24.