POLITECNICO
MILANO 1863

AN2DL - First Homework Report
Team: fmap

Filippo Galli, Alessandro Howe, Matteo Callini, Paolo Bellezza

filippogalli, alehowe, matteocallini, paolobellezza

251720, 251125, 244635, 251364

November 24, 2024

1 Introduction

This report describes the methodologies and strategies adopted for the **blood cell classification** challenge proposed during the Artificial Neural Networks and Deep Learning course at Politecnico di Milano. The challenge focuses on classifying images into one of eight blood cell types. The main objective is to develop a robust and accurate **deep learning model** to correctly classify blood cell images. To achieve this, image **pre-processing techniques** were employed and a pre-trained convolutional neural network was leveraged for feature extraction. These features were then used to train the model for accurate classification.
NB: Code.zip includes the final code *fmap.ipynb*, which contains the finalized model, and all other notebooks which support the results discussed in the report.

2 Data Preprocessing

The given dataset contains 13759 images with all eight cell classes. After initial data exploration, two issues are identified:

1. The presence of **1800 outliers** with backgrounds showing unrelated content, such as Shrek or Rick Astley.
2. **Difference in the number of sample per classes**, going from 2530 images in class 6 to 1049 in class 0.

To solve the first problem, images with these particular background are removed. Initially, a method to **remove the background** from all the images was also tested as attention mechanism with the results in loss of valuable information so we quit it.

The second issue is addressed by creating synthetic data using:

1. **Augmentation**: a function is developed using *Tensorflow* and *Keras* to automatically generate a specific number of images per class, balancing the dataset.
2. **Custom Interpolation**: synthetic images are generated using convex combinations of images within the same class.

Augmentation Version 1. Initially, both techniques were considered, but extensive testing revealed no improvement in model performance. Therefore, only data augmentation was used in subsequent experiments.

Augmentation Version 2. This version implemented lighter transformations like contrast and brightness adjustments, flipping, and Gaussian noise. However, these limited the models' generalization capabilities. Further trials with stronger augmentations showed consistent benefits.

Augmentation Version 3. An enhanced augmentation process was developed using the *RandAugment* and *AugMix* functions from *kerasCV* (see Figure 1).

Augmentation Version 4. A hybrid solution was ultimately chosen, combining basic transformations with a custom RandomAugmentation and excluding AugMix. This approach aimed to preserve image

structure and retain detailed cell characteristics while minimizing information loss.
In any case, we obtain **3000** images for each class.

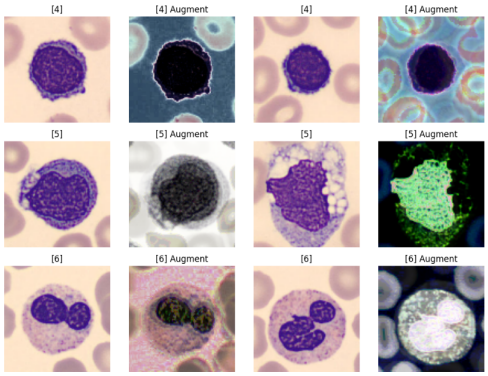

Figure 1: Some examples of before and after augmentation

3 Model Development

The selection and development of models begins with simple **custom architectures** inspired by *AlexNet0*. These architectures incorporate different blocks of layers from well-known networks, such as those used in *InceptionNet* and the *Depth Separable Convolution Layer* from *MobileNet*. The goal is to iteratively increase the complexity of the models. The custom *AlexNet* model achieves promising results on the local test set (over 90% accuracy), but its performance on the *CodaBench* test set is suboptimal. As the complexity of custom models increases, training times grow significantly because the model needs to be trained from scratch, without proportional gains in performance.

The focus shifts to **transfer learning**, leveraging pre-trained *Keras* models trained on the *ImageNet* dataset, followed by **fine-tuning** as outlined in [1].

The process starts with more standard models like *VGG16* and *MobileNetV3*, before progressing to networks such as *ResNet*, *Xception*, *EfficientNetV2S*, *NASNetMobile*.

Since larger pre-trained networks require more computation and training time, the complexity of the final part of the network is not increased. A **Global Average Polling** layer is adopted, followed by a **fully connected** layer with eight neurons and a **softmax** activation. In addition, in some cases, an **augmentation layer** is tested before the pretrained network.

3.1 Hyperparameter Optimization

Overfitting presents a significant challenge due to the distribution differences between our dataset and *CodaBench*. To address this, several techniques are employed to improve generalization. First, a **dropout** layer is added to the final layer of the network. After multiple experiments, a dropout rate of 0.3 is selected. Additionally, **L2 regularization** is applied to the final dense layer, which results in slight performance improvements on the *CodaBench* test set.

To further reduce overfitting, validation accuracy is monitored during training, and **early stopping** is introduced.

Normalization techniques are also tested by incorporating either **Batch Normalization** or **Layer Normalization** before the final dense layer.

For optimization, the default **Adam** optimizer is initially used with a learning rate of 10^{-3} for transfer learning and 10^{-4} during fine-tuning. However, experiments show that **AdamW** delivers better results. Different values of **batch size**, number of **epochs**, and **patience** for early stopping are tested: only the batch size was reduced during the fine-tuning phase to achieve a slight improvement in accuracy. However, these do not appear to have a significant impact on performance. Instead, these hyperparameters play a more critical role in memory usage, as highlighted in Section 5.

4 Final phase and results

In the final phase of the challenge, the development focuses on three pre-trained models: ***Xception***, ***NASNetMobile*** and ***EfficientNetV2s***. *Xception* is used as a baseline, as it lacks some of the distinctive features presented in the previous section. The choice of these models was driven by the aim to achieve high performance - particularly in terms of accuracy - while keeping the network relatively small and numerically efficient. After various tuning the final versions of the models are described in Table 1, while the results of the models are summarized in Table 2. The first four rows show performance scores on a local test set; the last row reports results on Codabench. Although local results are consistently high, as previously observed with simpler custom models, performance drops significantly on Codabench. This highlights the importance of model selection, augmentation techniques and strategies to prevent overfitting the local training set. Among the final models, **EfficientNetV2s achieves the best performance**. These excellent results are attributed to effective preprocessing, careful model refinement and optimization of training computation to enhance performance metrics.
Moreover, EfficientNetV2S outperforms Xception and

Model	NASNetMobile	EfficientNetV2s	Xception
Augmentation Layer	✓	✓	✗
Self-attention	✓	✗	✗
Drop-out	✓	✓	✓
L2-regularization	✓	✓	✗
Batch size	64	64	32
Patience	15	15	10
Optimizer	AdamW	AdamW	Adam
Augmentation version	3	4	2
Training time (in hours)	1,5	2,5	2
Fine Tuning Trainable parameters	8,518,085	10,916,552	15,902,921

Table 1: Model details

NASNet in efficiency and scalability. Its design innovations make it a preferred choice for tasks requiring both high accuracy and resource efficiency.

5 Discussion, challenges and further development

During the project, several challenges were encountered. The primary issue involved **limited RAM**. Initially, *Google Colab* was used, but its memory limitations often caused sessions to crash during large-scale augmentation and model fitting. Reducing the batch size in the training of the network solved the problem for smaller networks, however it was not a permanent solution. To address this, we moved to Kaggle, which offers more memory.

Another significant challenge was **computational time**. Efforts were made to optimize the code, but this process was not straightforward. Running the code on two GPUs introduced further complications, such as the **loss function diverging to NaN** during training. Additional difficulties arose from NaN values during training and issues with custom functions. To further enhance the results, it would have been beneficial to investigate potential improvements by increasing the number of images or conducting fine-tuning with fewer frozen parameters. Overall, the approach sought to balance computational cost with performance. For example in both models, NASNet and EfficientNet we retrieved great results with a fine tuning of half of the parameters.

6 Author Contributions

The main focuses of each team member are outlined below. However, the work was characterized by strong collaboration and significant overlap in tasks.

Filippo Galli: Led experiments involving the *NASNetMobile* model. Tested background removal methods. Optimized codes and functions.

Alessandro Howe: Led experiments involving the *Xception* model. Tested custom interpolation methods. Focused on strategies to prevent overfitting.

Matteo Callini: Led experiments involving the *EfficientNetV2* model. Implemented and tested custom models in the initial stages of the project. Explored the *ResNet* and *MobileNetV3* architectures during the early phases.

Paolo Bellezza: Led data exploration tasks. Focused on developing and applying augmentation techniques in combination with different models. Investigated the *VGG* architectures during the early phases.

Model	NASNetMobile	EfficientNetV2s	Xception
Accuracy	0.8757	0.9875	0.9791
Precision	0.8904	0.9875	0.9792
Recall	0.8757	0.9875	0.9791
ROC AUC	0.9896	0.9985	0.9791
CodaBench accuracy	**0.77**	**0.85**	**0.68**

Table 2: Metrics of our best models

References

[1] R. Ashgar, S. Kumar, P. Hynds, *Automatic classification of 10 blood cell using transfer learning via pre-trained convolutional neural network*, Informatics in Medicine Unlocked, 2024.

[2] M. J. Macawile, V. V. Quiñones, A. Ballado Jr., J. Dela Cruz, and M. V. Caya, *White Blood Cell Classification and Counting Using Convolutional Neural Network*, 2018 3rd International Conference on Control and Robotics Engineering, School of Electrical, Electronics and Computer Engineering, Mapua University, Manila, Philippines.