POLITECNICO
MILANO 1863

AN2DL - First Homework Report
NeuralNexus

Andrea Codazzi, Alessio Onori, Pasquale Serrao, Anna Simeone

andreacodazzi, ironoa01, pasqualeserrao, annasimeone

247291, 224955, 250803, 245744

November 24, 2024

1 Introduction

Accurate *identification* and *quantification* of blood cells are critical in clinical diagnostics, as abnormalities in cell counts often indicate underlying health conditions. (For example, irregular white blood cell counts may suggest infections [1]).
In this context, this project aims to develop a deep learning model capable of distinguishing between eight distinct blood cell types. To achieve this, our approach engaged the development of both a **custom-designed** convolutional neural network and **pretrained models**, such as *VGG-16* and *EfficientNet*, known for their strong performance in image classification tasks [2].
During the development phase, **data augmentation techniques** were implemented to enhance the model's robustness. This is particularly critical for practical applications, where datasets may deviate significantly from the training data—for instance, images captured via electron microscopy or containing pathological cells.

2 Problem Analysis

2.1 Dataset Characteristics

The provided dataset consists of microscopic images of blood cells, categorized into eight distinct classes: Basophil, Eosinophil, Erythroblast, Immature Granulocytes, Lymphocyte, Monocyte, Neutrophil, and Platelet. The dataset contains approximately 13,000 samples. Each image has a resolution of 96×96 pixels with three color channels (RGB).

2.2 Main Challenges

1. **Class imbalance**: Some blood cell types, like Basophil and Lymphocyte, are underrepresented. This class imbalance may cause the model to rely on statistical inference rather than learning the unique characteristics of each class, introducing so a bias in the prediction.

2. **Limited variability in the Dataset**: Since all the images are clean, with low intra-class variability and minimal noise, the model may risk *overfitting* to the specific characteristics of the training data.

2.3 Initial Assumptions

This project is based on two key assumptions. First, it is assumed that all images originate from a consistent domain, captured using the same microscope, resolution, and staining protocol. This uniformity is expected to ensure high performance on the local training and validation sets, though it may limit the model's ability to generalize to data acquired under different conditions. Second, it is assumed that the

resolution of the images is sufficient to capture the relevant cellular features necessary for classification, without the need for additional preprocessing to enhance the resolution.

3 Method

This section describes the approach used for training the model and evaluating its performance, including the choice of model architecture, preprocessing steps, data handling strategies, and training procedure.

3.1 Data inspection and processing

We performed a visual inspection of the images to assess their overall characteristics and identify potential outliers or inconsistencies. Approximately 2,000 images were identified as outliers during this process and subsequently removed to ensure dataset integrity.
As a first step, we split the dataset into training (80%), validation (10%), and testing (10%) using stratification. Then, we balanced the training set by performing the *upsampling* of the underrepresented classes by applying common geometric augmentation techniques (rotations, flipping, zooming, etc.).
To avoid the model learning to classify the augmented classes simply due to their transformations, rather than their intrinsic features, we decided to implement dynamic augmentation, which changes at each epoch on every batch, across the entire dataset.
We chose to use *RandAugment* [3], a powerful advanced augmentation technique that, with a simple pipeline, allows to apply a random combination of transformations with variable magnitudes.

3.2 Building models

We explored multiple architectures to evaluate their effectiveness for the specific task. We followed two distinct approaches: one based on the development of a **custom Convolutional Neural Network** (CNN) and the other involving **transfer learing** and **fine tuning**.
The custom CNN was made of four convolutional blocks, where each of them included a convolutional layer, a batch normalization layer to stabilize the training, and a max-pooling layer to reduce spatial dimensions. After the convolutional blocks, the resulting feature maps was flattened into a one-dimensional vector, which is then passed through a dense layer, a dropout layer and finally a softmax activation layer.
The second approach involved the use of pre-trained models, such as *VGG16* and *EfficientNet*, which were originally trained on the *ImageNet* dataset. These models were employed as feature extractors. After the convolutional base of each pre-trained network, a Global Average Pooling (GAP) layer was added to obtain a compact and efficient representation of the learned features. To further prevent overfitting, a Dropout layer was introduced. The final step involved a Dense layer with softmax activation.

3.3 Training models

For the pre-trained models, we adopted a gradual fine-tuning approach to preserve the integrity of the pre-trained weights. Initially, we trained only the top-added layer until convergence. Once the top layer was trained, we selectively fine-tuned a subset of layers. The layers closer to the top of the network capture deep semantic features relevant for each specific task, so their fine-tuning helped improving the classification performance [4]. For the training process, we tuned the following parameters:

- **Loss function**: We selected *Categorical Crossentropy* as the loss function. It effectively penalizes incorrect predictions, encouraging the model to output high probabilities for the correct class.

- **Optimizer**: We used the *Adam* optimizer by default, as it is widely recognized for its robustness and adaptability [5]. We experimented with various learning rate values to fine-tune the model's performance. Additionally, we explored the use of the *weight decay* and *Lion* optimizer to further prevent overfitting [6].

- **Early stopping**: To prevent overfitting and save computational resources, we incorporated Early Stopping as a callback.

4 Experiments

We tested our models both on the local test set and on the hidden test set, evaluating its performance in terms of accuracy.
To address the data imbalance, both **upsampling** and downsampling were implemented. Models using upsampling with augmentations significantly outperformed the one engaging downsampling, which strongly reduced the size of the training set.
According to the state of the art [7], we tried combining *RandAugment* with *CutMix* and *MixUp*, but the model failed to converge. This was likely due to an excessive alteration of the image's semantic information, which hindered the learning process. While using *RandAugment* and *CutMix* together allowed the model to eventually converge, it did not lead to performance improvements compared to using *RandAugment* alone. We also explored the *AugMix* technique but it delivered consistently worse results than *RandAugment*, even when combined.
We adjusted the **batch size** to assess its influence on model performance, observing that smaller batch sizes yielded slight performance improvements. However, they also introduced greater training instability and slower convergence. Performing **fine-tuning**, we experimented with unfreezing varying numbers of layers across networks to identify the optimal balance between performance gains and computational efficiency.

5 Results

The custom CNN model was the lightest in terms of architecture and exhibited the fastest training time among the models tested. Initially, it achieved excellent performance on the validation set, however, when evaluated on the test set from CodaBench, a significant drop in performance was observed.
Among the pre-trained models, VGG16 performed reasonably well despite its relatively lower efficiency. When combined with random augmentation techniques, VGG16 achieved a good performance on both the validation and test sets. However the different EfficientNet versions tested (B0, B2,V2B2,V2M) outperformed VGG16, consistently achieving higher accuracy. To further improve generalization capability we tried also to tune the *weight decay* parameter using Adam optimizer, showing no significant improvements. After that, we switched from Adam to Lion, achieving a great improvement in the model performances. The EfficientNetV2-M model, with Lion, performed particularly well, achieving accuracy of 85% on the hidden test set. In the following table is showed a comparison between different tested models:

Model	Local Accuracy	Codabench Accuracy
Custom CNN	88.8%	38%
VGG-16	94.73%	66%
EfficientNetB0	96.24%	75%
EfficientNetB2	96.74%	78%
EfficientNetV2-M (Adam)	95.15%	74%
EfficientNetV2-M (Lion)	**96.49%**	**85%**

6 Discussion

The constant performance drop observed on the CodaBench test set with respect to the local test set, especially for the custom CNN, is likely due to overfitting and limited generalization capability caused by the limited variability of the training data.
Pre-trained models effectively addressed these issues by extracting robust and transferable features, reducing the risk of overfitting. Among them, better performances of EfficientNet models could be attributed to their more advanced architecture, which better balanced depth and computational efficiency, in comparison with VGG16 and custom CNN. Furthermore, fine-tuning the deeper layers of EfficientNet models was particularly impactful, as these layers specialize in capturing complex and task-specific semantic features.

7 Conclusions

The best result for this project was achieved with the deepest network tested, but it was also largely influenced by training and optimization choices, such as the optimizer. Moreover the reached accuracy still showed large improvement margins. With more time and a greater computational power, it would have been possible to test more complex networks(e.g. VGG19 [8]), explore more deeply the tuning of our hyperparameters, and include alternative approaches (e.g. crossvalidation) to try to surpass this performance.

References

[1] Honda, Takayuki, et al. "Neutrophil left shift and white blood cell count as markers of bacterial infection." Clinica chimica acta 457 (2016): 46-53.

[2] Aggarwal, Shivam, et al. "Image Classification using Deep Learning: A Comparative Study of VGG-16, InceptionV3 and EfficientNet B7 Models." 2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE). IEEE, 2023.

[3] Cubuk, Ekin D., et al. "Randaugment: Practical automated data augmentation with a reduced search space." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. 2020.

[4] Kim, Hee E., et al. "Transfer learning for medical image classification: a literature review." BMC medical imaging 22.1 (2022): 69.

[5] Postalcıoğlu, Seda. "Performance analysis of different optimizers for deep learning-based image recognition." International Journal of Pattern Recognition and Artificial Intelligence 34.02 (2020): 2051003.

[6] Masadeh, Raja, Basel A. Mahafzah, and Ahmad Sharieh. "Sea lion optimization algorithm." International Journal of Advanced Computer Science and Applications 10.5 (2019).

[7] Park, Juyong, et al. "Methodology of Applying Randomness for Boosting Image Classification Performance." IEMEK Journal of Embedded Systems and Applications 19.5 (2024): 251-257.

[8]HENI, Ashraf; JDEY, Imen; LTIFI, Hela. Blood Cells Classification Using Deep Learning with customized data augmentation and EK-means segmentation. J. Theor. Appl. Inf. Technol, 2023, 101.3.

Individual Contributions

Codazzi Andrea:

- Rried the custom CNN architecture
- Tested different models for transfer learning and their different implementation
- testing the effect of different batch size

Onori Alessio : Alessio's primary contributions included setting up and debugging the local GPU environment, which he utilized to explore transfer learning and fine-tuning strategies. Additionally, he implemented the Lion optimizer with EfficientNetV2-M, leading to our best results. The Dockerfile used for the GPU setup has been open-sourced and is available at `https://github.com/ironoa/tensorflow-gpu-docker`.

Serrao Pasquale:

- Performed data preprocessing
- Tested different data augmentation methods(customized layers and combinations of methods)
- Tested and compared different models for transfer learning
- Tested effects of weight decay
- Tested effects of a top-added dense layer (see notebook)

Simeone Anna:

- Performed data inspection and trials with upsampling and downsampling
- Tested data augmentation (static and dynamic) methods with custom pipeline and with advanced techniques

-
- Tested various fine-tuning strategies on the EfficientNet architecture by unfreezing different numbers of layers
- Worked on model explainability by detailing and interpreting predictions (see notebooks for further details)