POLITECNICO
MILANO 1863

AN2DL - First Homework Report
YNWA

Maria Aurora Bertasini, Marco Cioci, Francesco Rosnati, Luca Tramacere

mablearns, marcocioci, rosnavigator, freshtrama

260345, 252488, 252560, 247220

November 24, 2024

1. Introduction

The homework was focused on classifying eight specific classes of blood cells. We adopted a systematic trial-and-error approach to iteratively develop a convolutional neural network (CNN) model capable of achieving high accuracy.

In this report, we detail each step of our journey, emphasizing an incremental approach that focuses on steady progress and establishing a solid foundation at each stage before moving forward, resulting in a model that achieves a balanced trade-off between computational efficiency and performance. Although this paper outlines the main concepts and methodologies, it is designed to be read alongside a series of accompanying Jupyter notebooks. The `template.ipynb` notebook serves to illustrate the key components of our networks architecture. Additionally, it acts as a guide, directing the reader to other notebooks that delve into specific aspects of the process.

2. Initial Steps

2.1. Dataset preparation

The initial dataset required significant preprocessing to make it suitable for model training. In this preliminary phase, we cleaned the dataset by removing outliers and identical copies (`DatasetPreparation.ipynb`). From this point forward, all models were trained on the cleaned dataset.

Additionally, the dataset exhibited an imbalance across the eight classes. Initially, this was manually addressed by under-sampling the larger classes. However, to further optimize the model's ability to handle class imbalance, we transitioned from manual balancing to using class weights during training. This adjustment allowed the model to learn more effectively from the unbalanced data while utilizing the entire dataset.

2.2. Custom CNNs

The first set of experiments (`CustomCNNs.ipynb`) involved designing and training custom convolutional neural networks (CNNs) to classify the images. These models started from scratch with minimal customization, failing to capture the complex patterns in the images. The lack of data augmentation and proper regularization led to overfitting. The results were indeed disappointing, yielding low accuracy on the evaluation platform despite good results on local testing.

2.3. Understanding the dataset

The Custom CNN phase revealed that the dataset used for training significantly differed from the real evaluation dataset, causing overfitting and poor performance on unseen data. Despite good results on local validation, this data mismatch highlighted the models' inability to generalize well. To address this issue, we shifted towards more robust techniques such as transfer learning, using pre-trained models to better handle the data mismatch and improve performance.

3. Transfer Learning

3.1. Backbone Selection

In the backbone selection phase a variety of pre-trained models were tested among the ones available in Keras, in order to find the one with the best balance between accuracy and efficiency.

The simple idea was to use the pre-trained feature extractor initialized with ImageNet weights as the backbone, keeping it fully frozen, while training the weights of custom classification head consisting mainly of three dense layers. No data augmentation or fine-tuning was applied during this phase.

The results of all models tested during this phase are summarized in Table 1, alongside their respective sizes.

Although larger backbones like ConvNeXt showed slightly better performance, their increased computational requirements made them less desirable for this task. For this reason we opted for balance between model performance and computational efficiency.

EfficientNetB2 was then chosen as a feature extractor. The classifier was composed of three dense layers with LeakyReLU activations to prevent neuron death. Dropout layers were added at the output of the EfficientNet and after the dense layers to mitigate overfitting. The output layer used Softmax activation, as expected for multi-class classification, resulting in a simple yet functional network.

During training, class weights were used to address the class imbalance, as mentioned earlier. We employed early stopping with a patience of 15 epochs, allowing the model sufficient time to learn, with a maximum of 100 epochs. Additionally, model checkpoints were used to save the best-performing model. At this stage, we did not experiment with different optimizers, choosing instead the flexible Adam optimizer. Learning rate scheduling was implemented with exponential decay to further optimize training.

After identifying the optimal backbone, the transfer learning phase was validated and remained unchanged throughout the subsequent experiments. This allowed for a systematic comparison as we incrementally introduced additional steps. The baseline accuracy of the chosen model was established, providing a solid reference point.

In conclusion, this phase allowed us to set a starting point, achieving an accuracy of 0.63.(Table 2)

Table 1: Different backbone models.

Model	Acc.	Params	Size(MB)
EfficientNetB0	0.60	4049571	15.45
EfficientNetB1	0.59	6575239	25.08
EfficientNetB2	0.63	7768569	29.63
EfficientNetB3	0.60	10783535	41.14
ConvNeXtTiny	0.61	27820128	106.13
ConvNeXtSmall	0.64	49454688	188.65
VGG16	0.47	14714688	56.13

4. Fine-Tuning

Once the base model was selected, we decided to experiment with fine-tuning to better understand how to manage the process effectively. Fine-tuning involved unfreezing several layers of the pre-trained model, enabling the network to adjust its feature representations to better suit the new dataset. It is generally recommended to unfreeze entire blocks, as CNNs repeat similar sequences of layers within these blocks.

The fine-tuning process was gradual, starting with the unfreezing of a single block of layers (only convolutional ones) and progressively increasing the number of unfrozen blocks up to three. The model with three unfrozen blocks became the baseline for all subsequent improvements, as described in the following section. Further fine-tuning was deferred until the final phase.

The baseline got us an accuracy of 0.68.

5. Data Augmentation

After having established the proper baseline with transfer learning and fine-tuning, we recognized that the key to achieving true generalization was adopting a methodical strategy of heavy augmentation. We began with simpler augmentation techniques and progressively explored more complex combinations.

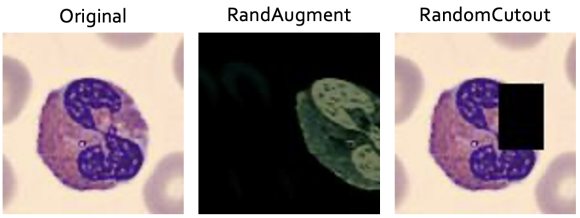
Initial Augmentation We first applied a gen-

Table 2: Summary of EfficientNetB2 results over time during development phase. (Codabench score)

Model	Description	Accuracy
Starting point	Transfer Learning only	0.63
Baseline	Transfer learning & fine tuning (three blocks unfrozen)	0.68
+Base augmentations	Added default augmentation layer avail. in Keras	0.74
+RandAugment	Added RandAugment from KerasCV	0.87
+Lowered dropout	Lowered dropout rate	0.89
+Cutout	Added cutout	0.90

eral Keras augmentation layer with light transformations, including horizontal flipping, random rotation, brightness adjustment, translation and zoom. This approach provided modest improvements in performance, bringing the accuracy to 0.74 and setting the stage for experimenting with more advanced augmentation techniques.

KerasCV Augmentation We subsequently enhanced our approach by incorporating additional data augmentation pipelines using KerasCV. Various combinations of transformations were systematically tested to determine the most effective strategies. The augmentation techniques explored are comprehensively documented in the notebook `Augmentations.ipynb`, which facilitated the visualization of augmentation effects and the selection of promising configurations for further testing. Among these, the `RandomAugmentation` function demonstrated significant effectiveness, achieving an accuracy of 0.87.



Further optimization of KerasCV parameters combined with adjustments to balance the implicit regularization introduced by augmentations, such as reducing dropout rates, resulted in an increased accuracy of 0.89. The most effective augmentation strategy identified resulted in a combination of `RandomAugmentation` and `RandomCutout`, culminating in a final accuracy of 0.90.

6. Final Phase

In the final phase, the test set on Codabench was updated, resulting in a slight decrease in accuracy (from 0.90 to 0.88). To address this, we focused on refining the **fine-tuning** process, with particular emphasis on selecting the optimal optimizer and determining the best number of layers to unfreeze.

Optimizer We experimented with several optimizers, including Adam, AdamW, and Lion. Among these, **Lion** demonstrated the best performance. Throughout the experimentation, the weight exponential decay was consistently applied.

Layer Configuration To identify the optimal number of unfrozen layers, we progressively increased the number of frozen blocks, monitoring the model's performance at each step. The optimal configuration (five out of seven) was determined when additional freezing began to reduce performance instead of improving it.

7. Discussion and Conclusion

The final model (`FinalModel.ipynb`) achieved a validation accuracy of **94%**. This outcome underscores the value of a systematic approach, where each design decision presented above was carefully isolated, enabling clear attribution of improvements and ensuring reproducibility.

Contributions The entire homework was carried out together, with team members contributing to overall development and implementation. Special mentions: Marco Cioci, Augmentation and backbone selection. Luca Tramacere, Custom CNNs and optimizer. Maria Aurora Bertasini, Baseline (transfer learning and fine-tuning structure). Francesco Rosnati, Augmentation and fine-tuning.

References

[1] Slides from course lectures and exercise sessions.

[2] Keras Team, *Keras Documentation*, Available at: `https://keras.io/`.

[3] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le, *RandAugment: Practical automated data augmentation with a reduced search space*, arXiv preprint, `arXiv:1909.13719`, 2019. Available at: `https://arxiv.org/abs/1909.13719`.

[4] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He, *A Comprehensive Survey on Transfer Learning*, arXiv preprint, `arXiv:1911.02685`, 2020. Available at: `https://arxiv.org/abs/1911.02685`.