POLITECNICO
MILANO 1863

AN2DL - First Homework Report
Synapses

Federico Arcelaschi, Giacomo Delera, Andrea Varesi, Paolo Riva
Codabench fedearce, Codabench gdele, Codabench andreavaresi, Codabench pr414
233301, 259904, 252584, 233240

1 Introduction

Blood cell identification is critical in medical diagnostics, particularly for understanding diseases affecting the blood[1].
This project focuses on the **classification** of different types of **blood cell images** into eight distinct categories using **neural networks and deep learning techniques**. The primary objective is to develop a model capable of achieving **high accuracy** in this task using image data.

The project began with the construction by hand of a **Convolutional Neural Network (CNN)**, which provided baseline results. To improve performance, we decided to use **tranfer learning techniques** using pre-trained models and we found that the EfficientNet family[2] was to best one to employ, with **EfficientNetV2S**[3], towards techniques of **data augmentation**, achieving the best performance.
Additionally, other dataset augmentation techniques were applied using keras-cv[4] to enhance the training dataset and further improve model robustness. In the end, also **test time augmentation** techinques were employed to improve the accuracy.

2 Problem Analysis

- **Dataset**: The dataset contained **13759** labeled images of eight distinct blood cell types. The original input size of the images in the dataset was $96 \times 96 \times 3$. The distribution of classes was evaluated but we decided not to perform class balance because after some try it resulted not to be worthy.
- **Preprocessing**: Images were resized to the standardized input size of $128 \times 128 \times 3$, (but not only, in the end ...) better for many of the pretrained neural networks architectures used during the development phase. **Normalization** was applied to scale pixel values.
- **Main challenges**

1. **Dataset Cleaning**: as first thing we analyzed the original dataset and found some outliers (as shown in Figure 1) so we decided to clean up the dataset from those images (**1800** in total).
2. **Dataset split**: to train our model we applied **one-hot encoding** to labels to suit the classification task, and we initially split the dataset in 80% for training and 20% for validation and test. Then we favored a ratio of 90%-10% to enhance the training of our models.
3. **Pretrained model and Parameters selection**: we tried to use many of the models seen during lecture to find the one that better fits this problem and after some not satisfying results we found in the **EfficientNet**[2] family our best model. After the selection of the best model we focused also on finding the best parameters to use with our model, such as **learning rate, batch size, epochs** and so on.
4. **Data Augmentation**: this is one of the most critical points of the challenge: building a good structure of augmentations to produce the best augmented dataset as possible. Much time in the development phase was focused on finding the best techniques of data augmentation for our problem.

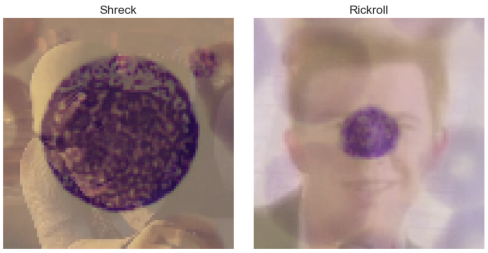

Figure 1: Outliers in the dataset

3 Method

After a first try building our personal CNN from scratch that, despite adding different combinations of layers or changing the parameters, reached a low upper bound on the overall accuracy performance, we started to employ **transfer learning** by leveraging pre-trained **Keras models**[5]. These models were **trained** and **fine-tuned** to adapt their learned representations to the blood cell dataset. In paragraph 4 we show the models that we tried to improve the accuracy score.
As regards the transfer learning phase, our focus was mainly to select the **best optimizer** as possible for our problem and the **right number and type of layers** in addition to the ones belonging to the pre-trained model selected.
We started using Adam as base optimizer but then we tried also Lion and **AdamW**[6]. This last one was the optimizer with which we achieved our best result on the platform, so we definitely adopted it.
The combination of layers that we found pretty soon to be a very good one was using one **Global Average Pooling** layer and then a three-layer structure repeated twice, composed by **Dense**, **Batch Normalization** (to stabilize and accelerate training) and **Dropout** (to prevent overfitting) layers. The second Dense layer has half of the units of the first one, to have a bottleneck and prevent overfitting.
Fine-tuning involved unfreezing selected layers of the model and training them on our dataset with a small learning rate, enabling the model to capture **task-specific features**[7]. In this case the main challenge was to select the right number of layers to unfreeze. After trying a completely unfreezed structure for fine tuning with bad results, we found that the standard number of **124** freezed layers produced the best results. Even changing to 100 or 150 we obtained lower accuracy results.
To improve the model's **robustness** and **generalization**, we applied **data augmentation** techniques such as **translation**, **shear**, **contrast**, **brightness**, **cut mix** and **mix up** during training. We spent a lot of time on this aspect since from the first augmentations we experienced very high improvements in the accuracy over the test set[8].
We also included a validation-based **early stopping** callback in both the transfer learning phase and the fine-tuning phase, in all considered models. This way we prevented the models of choice from overfitting the training dataset, directly benefiting the accuracy observed on the test set[9].
To introduce more variability on the test data, **test time augmentation** was also used. We tried many functions to reach a good combination of them, and finally the best ones for our model revealed to be **flipping, saturation, central crop and shift**[10]. It too led to achieving an higher, even if only slightly, accuracy score on the platform, more during the development phase with respect to the final phase.

4 Experiments

Dataset Experiments

Dataset pre-processing and augmentation played a crucial role in improving the model's generalization and robustness[11]. Below are the dataset-related experiments and their impacts:

- **RandAugment**: Applying random augmentations using `keras-cv` had a positive impact on the model, increasing accuracy (only on train set).
- **CutMix and MixUp**: These advanced augmentation techniques enriched the dataset by blending features and labels from multiple images. Both approaches positively impacted model performance, yielding accuracy improvements (all the dataset).
- **Other Transformations**: Experiments with additional transformations such as GridMask or GrayScale resulted in a slight decrease in performance, as these distortions reduced the quality and interpretability of the dataset.
- **Resizing Images to 300x300**: Increasing the input image size to 300x300 enabled the model to capture finer details, resulting in a significant accuracy boost.

Model Experiments

In parallel, several experiments were conducted on the model architecture and hyperparameters. These are summarized below:

- **Exploring Different Architectures**: Various neural network architectures were evaluated, including:
  - **Custom Convolutional Neural Network (CNN)**: Served as the baseline but achieved limited accuracy.
  - **MobileNetV2**: the first pre-trained model that we used, but also with data augmentation it reached low accuracy results (0.63)[12].
  - **EfficientNet Family**: Testing the first generation of networks (EfficientNet-B0, EfficientNet-B4 and EfficientNet-B7) allowed to gradually increase the observed performance up to 0.88, at a cost of slower training times per epoch with the biggest models. Testing the second generation of EfficientNet models provided progressive improvements from the previous generation, with EfficientNet-V2S achieving the highest accuracy of 0.95 with 1/3 the parameters size compared to the previous generation's biggest model (EfficientNet-B7)[2][3].

  - **ConvNext Family**: ConvNextTiny, ConvNextBase and ConvNextLarge performed well (around 0.80-0.85) but did not surpass the EfficientNet models in accuracy or efficiency[13].
  - **ResNet**: in the last days we tried this architecture but it was not possible to test it on the platform due to website problems[14].
- **Parameter and Hyperparameter Tuning**: Key parameters such as learning rate, batch size, and regularization factors were optimized. This optimization significantly contributed to performance improvements.
  We chose to use **200 epochs** both for transfer learning and fine tuning to have a sufficient window to let the model train. Then we set the batch size for the transfer learning to **64** and the one for fine tuning to **32**.
  As regarding the learning rate, during training we also used **ReduceLROnPlateau** as learning rate scheduler, to refine learning in the later stages of training and to reduce oscillations during the optimization process[15].

5 Results

Results from testing different models can be seen in Table 1). The reported results were obtained during the development phase, with the same optimal configurations that we found and that we exposed in the previous sections[2][3][12][13].

Model	**Top-1 ImageNet Accuracy**	**Result from testing**
Custom CNN	N/D	0.45
MobileNetV2	0.71	0.63
EfficientNet-B0	0.77	0.84
EfficientNet-B4	0.82	0.86
EfficientNet-B7	0.84	0.88
EfficientNet-V2M	0.85	0.91
EfficientNet-V2S	0.84	0.95
ConvNextBase	0.85	0.80

Table 1: Performance w.r.t. pretrained models

Model	**Augmentation**	**Accuracy**
B0	randAugment	0.83
B0	randAugment + cut mix + mix up	0.84
B4	randAugment	0.84
B4	randAugment + cut mix + mix up	0.86
B7	randAugment	0.87
B7	randAugment + cut mix + mix up	0.88

Table 2: Performance w.r.t. dataset augmentations

Model	**Optimizer**	**Accuracy**
V2S	Lion	0.91
V2S	Adam	0.93
V2S	AdamW	0.95

Table 3: Performance w.r.t. tested optimizers

6 Discussion

From the early development stage, considering increasingly-complex pre-trained models gave a direct accuracy boost with respect to smaller networks. Augmentation on the dataset came out to give the real big improvements in terms of performance, so the focus of our optimization strongly relied on dataset augmentation as key to reach above-90% accuracy.
Since our development-phase tests strongly showed that the EfficientNet's family of pre-trained CNNs were the ones scoring the highest accuracies, our work focused on adjusting our solution based on such models to best fit the presented classification problem. Models belonging to ConvNeXt's family of CNNs were tested to provide a comparison with the set of performance data that we had acquired on EfficientNet solutions, considering the same classification layers and similar model parameters. Presented with worse results, we decided to focus our efforts on EfficientNetV2 solutions, although ConvNeXt could be a solid alternative to push the performance delivered by our assumptions even further.

7 Conclusions

The EfficientNet-V2S model, in conjunction with advanced data augmentation techniques, successfully achieved 95% accuracy in the assigned classification problem. This approach demonstrates the effectiveness of combining modern architectures with robust data preprocessing and augmentation for challenging image classification tasks.
Some other things can be done to try to achieve a better accuracy on the test set, such as:

- Exploring deeper the various optimizers available and the combinations between them
- Trying to experiment some model ensemble technique
- Try to change the loss function during transfer learning and fine tuning since we used only the Categorical Crossentropy function

As regards the team, everyone of us contributed in the training part of the problem, testing the various options as exposed in this report. Giacomo and Federico focused also on the data augmentation part, Andrea on the test time augmentation, while Paolo managed the largest models in each family.

References

[1] Kolhatkar, Dixit, and Nisha Wankhade. «Detection and counting of blood cells using image segmentation: A review». In 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave), 1–5, 2016. https://doi.org/10.1109/STARTUP.2016.7583931.

[2] Tan, Mingxing, and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." arXiv, September 11, 2020. https://doi.org/10.48550/arXiv.1905.11946.

[3] Tan, Mingxing, and Quoc V. Le. «EfficientNetV2: Smaller Models and Faster Training». arXiv, June 23, 2021. https://doi.org/10.48550/arXiv.2104.00298.

[4] F. Chollet et al., "Keras," 2015. [Online]. Available: https://keras.io

[5] Team, Keras. "Keras Documentation: KerasCV Models." Accessed November 24, 2024. https://keras.io/api/keras_cv/models/.

[6] Team, Keras. "Keras Documentation: AdamW." Accessed November 24, 2024. https://keras.io/api/optimizers/adamw/.

[7] "Fine-Tuning LLMs: Overview, Methods Best Practices." Accessed November 24, 2024. https://www.turing.com/resources/finetuning-large-language-models.

[8] Team, Keras. "Keras Documentation: Image Augmentation Layers." Accessed November 24, 2024. https://keras.io/api/layers/preprocessing_layers.

[9] Team, Keras. "Keras Documentation: EarlyStopping." Accessed November 24, 2024. https://keras.io/api/callbacks/early_stopping/.

[10] "Understanding Test-Time Augmentation." Accessed November 24, 2024. https://arxiv.org/html/2402.06892v1.

[11] TensorFlow. "Data Augmentation — TensorFlow Core." Accessed November 24, 2024. https://tensorflow.org/tutorials/images/data_augmentation.

[12] Sandler, Mark, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks." arXiv, March 21, 2019. https://doi.org/10.48550/arXiv.1801.04381.

[13] Liu, Zhuang, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. "A ConvNet for the 2020s." arXiv, March 2, 2022. https://doi.org/10.48550/arXiv.2201.03545.

[14] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." arXiv, December 10, 2015. https://doi.org/10.48550/arXiv.1512.03385.

[15] Team, Keras. "Keras Documentation: ReduceLROnPlateau." Accessed November 24, 2024. https://keras.io/api/callbacks/reduce_lr_on_plateau/.