AN2DL - First Homework Report
I ragazzi del Bav

Stefano Corti, Andrea Catelli, Alessandro Ciotti, Marco Giovanni Barbero

stefanocorti, andreacatelli, aleciotti, marcogbarbero

245554, 247446, 247754, 260314

November 24, 2024

1 Problem introduction

The project concerns **supervised image classification** using **Deep Learning** techniques. Our goal is to develop a **Neural Network model** that is able to correctly classify an image of a **blood cell**. This task required us to focus on two different aspects: **data pipeline** and **models**.

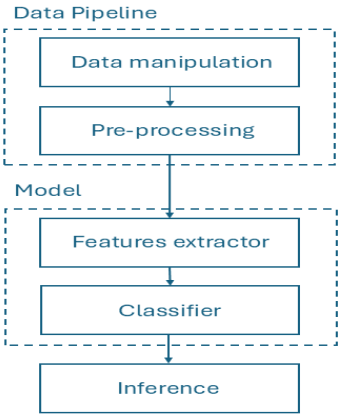

Figure 1: Development Process.

The starting point is a dataset of 13759 RGB labeled images of size (96x96), which we will address as **original dataset**. Overall, the number of labels is **eight** that represents different type of cells (basophils, neutrophils, lymphocytes...).

As we will discuss in the following sections, the correct manipulation of the dataset turns out to be crucial. Indeed, exploiting the simple and standard CNN architecture we were able to obtain high accuracy on the local test set; while the performance on new images in the submission was poor. In other words, the main challenge has been creating a model that was able to **generalize** and to **learn the specific features** of each cell type. Overcoming this challenge allowed us to obtain good results in the development and final phase.

2 Data pipeline

The first step in our analysis was to inspect our **original dataset** to understand the characteristics of each type of blood cell. Upon examining some random images, we noticed some **outliers**: images that were **identical** (pixel by pixel across all channels), but were assigned **different labels**. Consequently, we decided to remove these duplicates from the original dataset and focus our studies on the remaining images (**cleaned dataset**).

We noticed that the cleaned dataset was not balanced, with some classes being three times as frequent as others. Minding that it is not always the case, we balanced the dataset, gaining accuracy.

The **balanced dataset** was created by equalizing the occurrences across classes using rotation, shift, zoom and flip, which obviously does not characterize the cell class.

To help the model **focus on blood cell features** avoiding influences of the background, we explored two approaches. (i) Images were cropped to various dimensions, focusing on areas containing blood cells (**Cropped Datasets**). (ii) Using **unsupervised K-Means clustering**, we generated **binary masks** (0/1 valued matrices) that highlight the blood cell regions. The RGB images are multiplied pixel-wise by these masks before being fed into the classifier.

The most effective data processing turned out to be the **data augmentation**, using:

$$kcv.layers.RandAugment(...)$$

This function allowed us to generate for each image a predetermined number of **random augmentation** whose magnitude is distributed normally, exploiting heavily the information available from the dataset. Finally, a **large augmented dataset** fed in our models has been more effective than augment the images in a layer inside our architecture.

3 Saliency maps

To gain **insights** into the inner workings of our convolutional neural network (CNN) and improve the **explainability** of its predictions, we computed **saliency maps** following the approach introduced in the paper [1].

The saliency map highlights the regions of the input image that the model considers most important for making a specific prediction. We used **TensorFlow's GradientTape** to compute the gradient of the predicted class score with respect to the input image. We took the maximum absolute value of the computed gradient across all color channels to generate a single-channel saliency map. This highlights the **most influential regions**.

The resulting saliency maps were overlaid on the original images, enabling us to qualitatively assess which parts of the image contributed most to the model's predictions.

4 Models

4.1 Baseline CNN architecture

As an initial approach to solve the classification task, we implemented a **simple CNN architecture** consisting of two convolutional layers followed by a final dense layer with softmax activation. Despite its simplicity, this model performed well on our cleaned dataset, achieving a high accuracy on a local test set.

However, as expected, this basic architecture was insufficient to capture the **relevant features required for accurate cell classification**, resulting in an accuracy of only 0.27 on CodaBench. This limitation was further evidenced by examining the saliency maps (Figure 2), which revealed that the model failed to correctly focus on the cells.

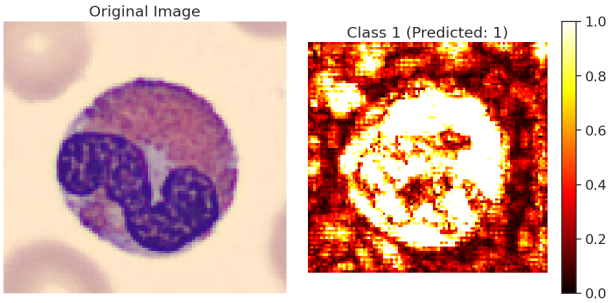

Figure 2: Saliency map for Baseline CNN

4.2 MobileNet features extractor

After conducting some experiments with this architecture to address overfitting and improve generalization, we decided to leverage **well-known pretrained models** on ImageNet as feature extractors. Our initial attempts focused on the **MobileNet architecture**. After applying the standard **Transfer Learning** procedure, including **fine-tuning** specific layers at the end of the model, we observed a strong improvement of accuracy on the competition test set. As expected, this model was more capable of **extracting meaningful characteristics** of the cells.

4.3 InceptionV3 features extractor

The Inception exctractor models are characterized by multiple filter size whose aim is to capture relevant information that may appear at a **different scales** in the images. This is done by layer that run **multiple convolution in parallel**. For this reason we have decided to perform transfer learning and fine tuning on the **InceptionV3** model.
Even if this property of the architecture fit well our data since each type of cells have a different size, as shown in Table 1 we have not achieved high accuracy with this model.

4.4 EfficientNet features extractor

We then decided to try the EfficientNet family, as it offers a great mix of **high accuracy** and **fewer parameters** compared to other neural networks. We started with **EfficientNetB0**, which gave solid results on the test set. However, trying other versions in the EfficientNet, it didn't make a significant difference in performance. Hence, we switched to the newer **EfficientNetV2** models. Among the options, we went with *V2S* because it's less demanding in terms of computation. Two dense layers are

Table 1: Codabench accuracy and metrics on local and Codabench test set

Model	Dataset	Codabench Accuracy	Local Test Set Accuracy	Precision	Recall	F1-Score
Baseline	Cleaned	0.27	0.9264	0.9308	0.9264	0.9266
MobileNet	Balanced	0.58	0.9846	0.9846	0.9846	0.9846
InceptionV3	Balanced	0.53	0.6241	0.6440	0.6241	0.6169
EfficientNetB0	Balanced	0.76	0.9473	0.9491	0.9473	0.9474
EfficientNetB2V3	Augmented	0.85	0.9553	0.9564	0.9553	0.9554
EfficientNetV2S	Augmented	**0.93**	0.9865	0.9865	0.9865	0.9865

concatenated with the net in order to solve the classification problem. The overall architecture was optimized using the **AdamW optimizer** and **L2 regularization** was applied to the convolutional layers to help **prevent overfitting**. As shown in Table 1, this final model matches the accuracy of the others on the balanced dataset but really **stood out on the test set**, making it the best choice for our task.

5 Results

In Table 1 we reported the model, the dataset used and their **performances** both on **CodaBench** and on a **local test set** created from the provided dataset. The first models highlight a **high difference between the accuracies**, underlying that our networks **lacked in generalization** and **suffered of overfitting**. We dealt with it by using standard techniques such as dropout on the dense layer and L2 regularization while performing the fine tuning. We leveraged different **optimizers like AdamW** and **Lion** which turn out to perform very well.

6 Conclusions

The problem of classifying blood cell images proved to be challenging due to the similarity between the images and the background that could be misleading. While the first issue was addressed by **applying data augmentation**, the second challenge was more difficult to resolve. Attempts to crop the images to isolate only the cells led to misclassifications by the tested models. To address this, we experimented with various network families to find the most accurate solution. Our analysis showed that the optimal architecture is a CNN using **EfficientNetV2S** as a feature extractor. The network is trained through a **direct fine-tuning** while preserving the parameters **pre-trained on ImageNet** for the first layers. We tried this procedure since the images of cells have **different characteristics**, that should be **learned from scratch**, with respect to the images in Imagenet. The strength of this model lies in its superior ability to focus on and **extract detailed structural information** from the cells, effectively **minimizing the influence of the background**, as demonstrated in Figure 3. This allowed us to achieve a **classification accuracy of 93%** on the test set, outperforming all previously implemented models. Further analysis of this model could focus on the behavior of the loss function and its stability.

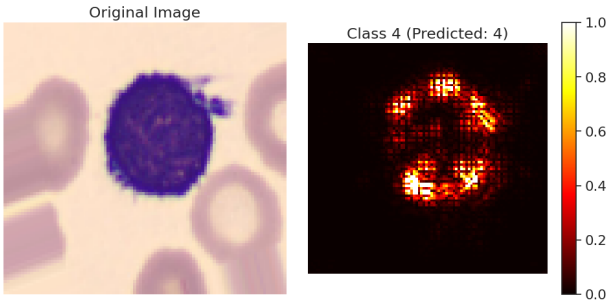

Figure 3: Saliency map for EfficientNetV2S

7 Members contribution

We began by analyzing the problem together and gaining initial insights from the dataset. **Andrea** started building a baseline CNN. Next, **Alessandro** created a new balanced dataset, which became the new starting point for the team. **Stefano** focused on the Inception model, and both he and Andrea worked on cropped datasets and binary masks, but without significant improvements. **Marco** and Alessandro first tested MobileNetV3, then moved to EfficientNet, achieving the first good results. The latter also augmented the dataset to further improve performance. Finally, we combined our datasets, models, code, and report into one **unified solution**.

References

[1] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

8 Annex

The zip folder includes the Jupyter notebooks used to run the models presented in the report (refer to the Table of Performances 1) excluding EfficientNetB0 as it was the first attempt in this models family. Additionally, there is a notebook titled *Dataset_Manipulation,* which documents the various preprocessing attempts made to enhance the image quality and improve task performance. Please note that we have not included any datasets (.npz) or model files (.keras) due to storage limitations. Unforntunately, we noticed that by importing the notebooks from Google Colab, Saliency Maps are not shown.