# AN2DL - First Homework Report
## Cortex Creators

Beatrice Adamini, Davide Cattivelli, Giorgio Coccapani, Kalle Engblom

beatriceadamini, catti01, giorcocc, kalbz2

251824, 259084, 259839, 276779

November 24, 2024

## 1 Introduction

This project aims to focus on **multi-class classification** techniques by examining a dataset of *cellular images* (in this case blood cells) using a deep-learning model based on a **convolution neural network** (CNN) specialized in this task. The points considered in the definition of the model concern:

- the analysis and pre-processing of the dataset provided to us
- the implementation of a neural network able to analyze and classify images also on an external dataset

## 2 Problem Analysis

The dataset for the convolutional neural network comprised 13.759 RGB images of blood cells, each labeled to indicate its category. The images, sized $96 \times 96$ pixels, were categorized into eight classes: basophil, eosinophil, erythroblast, immature granulocytes, lymphocyte, monocyte, neutrophil, and platelet. However, the last 2.021 images contained artifacts and inconsistencies and were subsequently removed, reducing the usable dataset to 11.738 samples.

The distribution of images in the various classes is analyzed in Figure 1. Since the distribution of elements in the classes is not uniform, a balancing of the weights of the various classes was applied during the training phase.

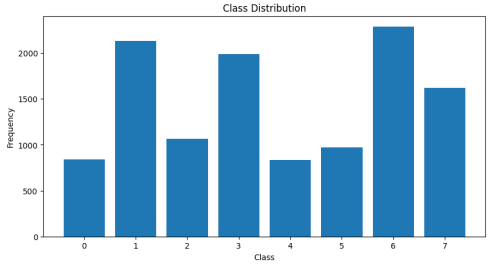


Figure 1: Distribution of the images among the 8 classes

## 3 Method

### 3.1 Data split and augmentation

Before starting any kind of data operation, we divided the dataset into 3 subcategories: a train set, a validation test and a test set. The test set contains 20% of the images of the original dataset while the remaining 80% is divided between the train and validation sets.

- Train set composed of 7981 samples
- Validation set composed of 1409 samples
- Test set composed of 2348 samples

Once the training set was obtained, a data augmentation pipeline was created and applied to compensate for the limited data availability. Using a `RandomAugmentationPipeline` (see [1]), four kind of transformations were applied to each image: geometric transformations (flip, rotation, zoom, and translation), filters (brightness, contrast, channel shift, and channel shuffle), and Gaussian noise. Each transformation (four for each image) was applied with random strength and the whole pipeline was applied on the training set only in even epochs, allowing the model to be trained on untransformed images as well. An example of the applied transformations is present in Figure 2.

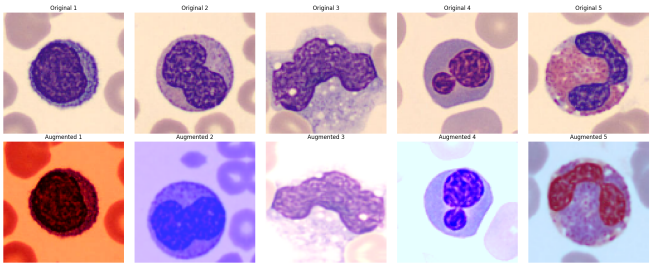


Figure 2: Example of data augmentation on some training samples

### 3.2 Model definition

The initial model was a simple convolutional neural network built from scratch. It featured convolutional layers with 32, 64, and 128 filters using ReLU activation, followed by Batch Normalization and Max Pooling layers. This was followed by a Flatten layer, a Dense layer with 128 units, Batch Normalization, a Dropout layer, and an Output layer with softmax activation.

In parallel, we started to explore other models already trained with image datasets such as `imagenet` and the results of the various tests are presented in Table 1. The model that best perform under our test was the `MobileNetV3Large` (more information about the model here [2]).

In the modified model, the feature extraction networks from `MobileNetV3Large` were retained, ensuring the preservation of pre-learned features that are critical for effective image classification.

The classification architecture began with a Squeeze-and-Excitation block, introduced to recalibrate the feature maps by emphasizing the most important features of an image and discern subtle differences between classes. Following this, a Global Average Pooling (GAP) layer was employed to reduce the spatial dimensions of the feature maps, which facilitated both classification and spatial localization training. Then, a Dense layer with a progressively smaller filter size, Swish activation function and L2 regularization was added to the chain, together with Dropout (to ensure that the model does not rely too heavily on any particular set of features) and Batch Normalization layer, added in order to stabilize and accelerate the training process.

The architecture was then fine-tuned including 140 layers of the `MobileNetV3Large` model to achieve a balance between feature extraction and task-specific classification efficacy and to better fit our specific classification task. The complete project with all the final model definition is available here [3].

## 4 Experiments

During development, we explored various aspects of the model. After selecting a pre-trained model (see Table 1), we focused on the augmentation pipeline, including the introduction of a MixUp layer. However, since this conflicted with the model, we opted for the pipeline described in Section 3.1. We also tested a more aggressive augmentation to reduce overfitting, but this yielded poorer results.

For fine-tuning, we made all layers of the pre-trained model untrainable initially, then progressively increased the number of trainable layers up to 140 to enhance accuracy. We tested alternative optimizers like Lion and AdamW to compile our model, but both led to performance degradation and uncontrolled loss computation. To reduce overfitting, we implemented `EarlyStopping` ([4]), `ReduceLROnPlateau` ([5]), Dropout layers, and L2 regularization on Dense layers.

We addressed training dataset imbalance using a class weight balancing technique, ensuring the model considered underrepresented classes. Additionally, we improved performance by replacing the ReLU activation function in dense layers with Swish (see [6]).

Table 1: Model evaluation during the experiments. The chosen model is marked in bold.

| Model | Validation Accuracy | Test Accuracy |
|---|---|---|
| Custom Model | 84.94% | 24.36% |
| VGG16 | 86.91% | 85.75% |
| VGG19 | 83.23% | 84.84% |
| MobileNetV2 | 71.33% | 71.00% |
| MobileNetV3Small | 37.01% | 13.48% |
| **MobileNetV3Large**[1] | **97.59%** | **97.40%** |
| EfficientNetB0 | 14.54% | 56.58% |

[1] these results were obtained considering the switch activation function in the dense layers of the custom classifier

## 5 Results and Discussion

Our implementation of a classifier capable of cataloging images of blood cells presented an accuracy value on our test set of 97.59% with a final value on the benchmark of 0.67 (the other results of the model are shown in the Table 2). It's possible to notice that the values recorded internally differ significantly from those found on the benchmark. This leads us to expect that the over-fitting phenomenon appeared to be prevalent on the test set used by the benchmark.

| | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| Values | 0.974 | 0.9747 | 0.974 | 0.9741 |

Table 2: Model scores on internal test set

This hypothesis is also supported by an analysis of the confusion matrix (see Figure 3) where it is clearly visible how the classes with the highest number of elements are those that stand out the most from the matrix. Probably, applying only the class weight balancing during the training phase of the model was not sufficient.

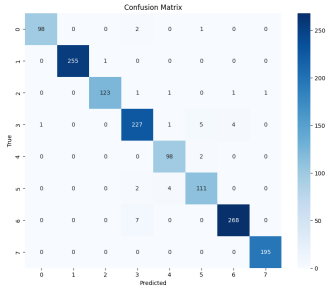


Figure 3: Confusion matrix of the model on internal test set

Finally, introducing normalization and adjustment layers between the various layers of the network could have improved the model.

## 6 Conclusions

Our project classifies blood cell images into 8 classes using a `MobileNetV3Large`-based deep learning model with a custom classifier. We implemented a data augmentation pipeline to expand the training set, techniques to prevent over-fitting, and functions to handle dataset imbalance.

The project goal was achieved locally as we managed to reach very high accuracy scores on the test set, along with other control metrics (see Table 2). On the other hand, considering a global point of view, the project was only partially successful as we achieved a score of only 0.67 on the benchmark platform.

This result makes us wonder how much more our model can be improved. First, it might be useful to investigate other pre-processing of the dataset to better fit the model and avoid biases. Second, it is possible to investigate other implementations for the custom classifier, in order to make it more effective in classification. Finally, a more careful analysis of the balance of the training test could make the model more generalist in classification, without it being able to "prefer" certain classes over others.

Special thanks to Beatrice (performance improvement and fine tuning), Giorgio (augmentation pipeline and model definition), Davide (augmentation pipeline and class weight control) and Kalle (data analysis and different model and parameters testing) [and L. for the precious hints and fundamental memes].

## References


[1] Keras. *RandomAugmentationPipeline layer*. en. https://keras.io/api/keras_cv/layers/augmentation/random_augmentation_pipeline/.

[2] Andrew Howard et al. *Searching for MobileNetV3*. 2019. arXiv: 1905.02244 [cs.CV]. URL: https://arxiv.org/abs/1905.02244.

[3] *Colab Notebook*. en. https://tinyurl.com/2btzcrnp. 2024.

[4] Keras. *EarlyStopping*. en. https://keras.io/api/callbacks/early_stopping/.

[5] Keras. *ReduceLROnPlateau*. en. https://keras.io/api/callbacks/reduce_lr_on_plateau/.

[6] Keras. *Swish activation function*. en. https://keras.io/api/layers/activations/#silu-function.