POLITECNICO
MILANO 1863

AN2DL - First Homework Report
SiumGPT

Angelo Attivissimo, Leonardo Bianconi, Mattia Busso, Armando Fiorini

angelo7672, leobianconi, mattiabusso, armaf

233484, 232004, 252501, 259459

November 24, 2024

1 Introduction

This project focused on solving an *image classification* task regarding microscope images of blood cells.
Automated classification of blood cells is a fundamental problem in medical applications: the analysis, study, and diagnosis of various blood diseases, including leukemia, anemia, lymphoma, thalassemia, and more, depend on the accurate labeling of these cells' images, which requires hours of labor from human experts.
Our aim was to tackle this problem developing Deep Learning architectures and techniques, in particular the *Convolutional Neural Network* (CNN) topology.
We approached the problem by reasoning on all aspects of it, from the dataset inconsistencies to the model selection, trying to put into practice every theoretical reasoning seen during the theory classes.

2 Problem Analysis

To perform the task a dataset of 13759 images of size 96x96 pixels was given to us. Each image in the set was labeled accordingly as one of the 8 classes.
We first found two types of *outlier images* (images that clearly deviate from the data distribution). These images were duplicated, adding noise to the problem.
Another problem that we encountered when inspecting the dataset was an *unbalanced class distribution*. The initial count and the final one of the classes can be seen in 3.2.
This set was used by us to both train and validate our models, while we tested the performances on an unknown test set.
In these first assessments our main challenges also became clear to us: the already limited size of the set, coupled with the fact that we had to get rid of outliers and reduce its size even more, presented us with a problem of **data scarcity**.
Data augmentation techniques and exploring different alternatives in *transfer learning* became one of our main focuses to try to build a model able to extract complex features from our images and to classify them correctly.

3 Method

3.1 Data Wrangling

We decided to exclude all the duplicated elements in order to purge the dataset of outliers. We used a randomized approach to search the dataset and find the outliers in order to speed up the computation. The dataset size was reduced to 11959 elements.

3.2 Data Balancing and Data Augmentation

We used five augmentation layers from keras_cv (AugMix, RandAugment, RandomHue, RandomCutout, and RandomSharpness) to augment and re-

balance the number of elements of the dataset. Each class was rebalanced to 5000 samples using the five layers in equal proportions on random sampled images from that class.
In order to increase the generalization of our neural network, we then chose to apply also the MixUp layer augmentation to the original dataset cleaned from outliers. After the MixUp layer, the resulting dataset is not exactly balanced because the original dataset was unbalanced, but there is no longer a disparity between the classes like the initial one. The final dataset is composed of 51943 elements.

Table 1: Counting occurrences of target classes.

Class	Base	Final Dataset
0: Basophil	1052	5867
1: Eosinophil	2381	7154
2: Erythroblast	1285	6066
3: Imm. granulocytes	2226	7043
4: Lymphocyte	1049	5831
5: Monocyte	1393	5985
6: Neutrophil	2530	7355
7: Platele	1843	6642

We also introduced a sequential augmentation layer directly inside the model, involving various types of transformations (Random Flip, Random Brightness, Random Rotation, Random Translation and Random Contrast from keras).

3.3 Feature Extraction Network

We tried various approaches to extract features out of our training data.

- *Custom made convolutions* quickly showed their limitations given the limited amount of data at our disposal.
- *Transfer learning* solutions, coupled with *fine-tuning*, enabled us to use the weights of famous architectures, pre-trained on the *imagenet* dataset, from which we imported in our model just the convolutional layers.
  In particular we experimented with **VGG16**, because of its simplicity and effectiveness that allowed us to see improvements while keeping our model and training lightweight; and with the **ConvNeXt** family of architectures, which gave us the best performance thanks to its modern design with increased complexity but still manageable computational training efforts. **ConvNeXtBase** turned out to be the best choice for us.

3.4 Fully Connected Layers

The convolutional network output is reduced to a 1-dimensional vector, thanks to a **GAP** layer, and fed to the dense fully connected network, which we composed of:

- 512 units, followed by Batch Normalization, ReLu activation and Dropout.
- 128 units, followed by ReLu activation and Dropout.
- 8 output units with *softmax* activation for multi-class classification.

This design proved to be the best fit out of all our experiments. In particular we used **Batch Normalization** to help stabilize learning and **Dropout** to help prevent *overfitting*.

3.5 Training Approach

We experimented with **Adam** and **Lion** optimizers on the **Categorical Cross Entropy** loss function to train our network, which is the most adequate choice when dealing with *multiclass classification*.
We also used **early stopping** techniques monitoring the accuracy of the *validation loss* to avoid overfitting.
We tried different learning rates and batch sizes in order to find the perfect one for our problem.

4 Experiments

While looking for the best possible solution in terms of accuracy, generality and size, we went through a series of steps, in which we experimented a variety of different architectures, layers, and methods, trying each time to focus on the weaknesses of our model to develop a new solution which could give us an improvement on those aspects.

4.1 Architectures

Our experiments involved various architectural approaches: starting from "self-made" convolutional architectures, which gave us optimal results in terms

of accuracy in training and validation set but resulted in a critically high level of overfitting, as the test accuracy testified. We then oriented our attention on the usage of pre-built/pre-trained models, on top of which we added customized fully connected layers to make them fit for our purposes.
On these architectures, transfer learning and fine tuning techniques were applied.
The variety of models we explored included: **VGG16** [3], **ConvNeXtTiny**, **ConvNeXtSmall** and **ConvNeXtBase** [2].

4.2 Parameters

We made a few experiments involving the selection of hyperparameters to improve various aspects of our model.

4.2.1 Overfitting

Following a number of tests, we found that the most effective strategy to reduce overfitting is:

- In the case of early stopping, using a low patience of 5.
- A dropout rate of 0.5.
- A total number of epochs for training of 50.

4.2.2 Training

To help speed up and achieve a higher accuracy during training, we used:

- A learning rate of 10^{-4}, and during fine tuning a lower learning rate of 10^{-6}.
- A batch size of 128 sample, to speed up the process.

5 Results

Although we've achieved very high accuracy on the local dataset right away (≈ 0.95), we've immediately noticed a huge difference with the low performances on the test set (≈ 0.25). This suggested us that the test data had a significantly **different distribution** w.r.t. the one of the given training data. This led us to investigate the need for augmentation, in order to learn more general patterns, to be more robust to transformations and to prevent **overfitting the training data**.

We then saw a high increase in the test set performances when starting to build transfer *learning models*, paired with *fine-tuning*, helping us break the barrier of ≈ 0.70 test set accuracy. After that, **class balancing**, employing more varied image augmentations and utilizing more **complex models** led us to a test set accuracy of **0.9** (Table 2).

5.1 Final Training

When we found our best model (ConvNextBase connected with the FC layers described in 3.4), we hardcoded the number of training and fine tuning epochs (19 and 17) that we calculated in the prior training with early stopping, and we conducted a final training with just the training set and no validation set. This led us to the final **0.91** test set accuracy.

6 Discussion

We believe that one of the biggest strengths of our model is the quantity and variety of augmented data it's been trained on: rich transformations that don't compromise semantics of the images were employed, leading to the possibility of **robustness** to future *concept drifts*.
We also consider that the usage of *Imagenet* pretrained *CNNs* contributes to robustness, due to the more general patterns coming from many natural images' domains. Moreover, the model is of adequate **size** for the majority of use cases and also **fast** at inference time.
On the other hand, we cannot be completely sure about the generality of the model accuracy since the kind of approach the challenge required could lead to a tendency to overfit the test set, which has been used several times to test the models' performances.

7 Conclusions

In the end, we've obtained a model capable of performing the task adequately.
It is not excluded that the dataset could be further improved to reduce redundancy, due to randomization, in the data by adding more augmentation layers. To improve performance, the FC layers might also be redesigned with an enhanced architecture and automatic hyperparameters adjustment.

Table 2: Performance metrics for different models.

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Test Set Accuracy**
Custom	0.9440	0.9467	0.9440	0.9444	0.6720
VGG16	0.9700	0.9701	0.9700	0.9700	0.7886
ConvNextBase	0.9833	0.9833	0.9833	0.9832	0.8991

Note: The custom model is composed by 8 convolutional layers and 2 hidden dense layers.

References

[1] J. Heaton. Applications of deep neural networks with keras, 2022.

[2] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s, 2022.

[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.

[1]