POLITECNICO
MILANO 1863

AN2DL - First Homework Report
Gradient Gangstars

Giorgio Algisi, Simone Bresciani, Ilaria Campestrini, Federico Dagani

giorgioalgisi, simonebresciani, ilacampestrini, federicodagani

250859, 275432, 252865, 243989

November 24, 2024

1 Introduction

The project discussed in these pages focused on solving an *image classification* problem using the deep learning techniques learned during the Artificial Neural Networks and Deep Learning course.

Our objective was to develop a Convolutional Neural Network (CNN) that could tackle with good performances an image classification challenge on a blood cells dataset.

2 Problem Analysis

The challenge we had to overcome was a multi-class classification problem. The dataset we worked on, indeed, was made up of 13759 96x96 RGB (0,255) images that were categorised into 8 classes representing different types of blood cells (*Basophil, Eosinophil, Erythroblast, Immature granulocytes, Lymphocyte, Monocyte, Neutrophil, Platelet*). Given this limited number of samples, the problem of **overfitting** was more than just a possibility.

Another challenge was the class imbalance in the dataset: some classes were much more represented than others (18.39% of Neutrophil and 17.31% of Eosinophil w.r.t 7.65% of Basophil and 7.62% of Lymphocyte). This composition, replicated in the training set, would have biased the model toward the majority classes, neglecting the minority ones. This was another important challenge to face.

3 Our work

3.1 Data inspection

Using an hash-based analysis of the dataset we noticed that there was a huge number of duplicated images. A more accurate inspection of these elements revealed that the two most replicated samples were quite different with respect to the others. Indeed, it seemed that some blood cells were mixed up with some famous internet memes.

These **outliers** would have hurt significantly the performances of our models so they had to be removed from the dataset. At first, we tried to implement a "general" solution based on PCA[1] and Mahalanobis distance but it didn't provide the expected results. For this reason we moved to a "manual" algorithm which was able to delete both the outliers and the other duplicates from the dataset. A following pixel-based inspection confirmed that no more outliers seemed to present.

This solution lead to an important decrease of the number of samples in the dataset (and a small change in percentage of composition of the dataset around 1%), which made even more realistic the problem of overfitting.

[1] Principal Component Analysis

3.2 Data preprocessing

At first, we applied to the dataset *classical* data preprocess: **normalization** to the range [0, 1] of the images and **one-hot encoding** transformation of the labels.

In order to deal with the problem of data scarcity, we decided to apply **augmentation**. We started using classical and soft data augmentation [2] but we soon realized that we had to adopt a more aggressive and elaborate methodology. The principal operations that we applied were **AugMix** and **RandAugment** combined in a unique pipeline. Initially, we used several differently augmented dataset concatenated into a single one but, even if it allowed us to work with a larger and more heterogenous dataset, it also introduced too high computational costs. Moreover, performances didn't significantly improve so we decided to adopt a less computational expensive solution with just one augmented training set. Instead, increasing the number of augmentations applied on each image led to important improvements. In particular, in our final experiments, applying 2, 3, and 4 RandAugment transformations per image led to an improvement in performance on Codabench, with accuracy increasing from 0.71 and 0.78 to the final score of 0.86.

Another important problem was the **dataset imbalance**. In order to fix this issue, two different approaches were considered: evaluating classes' weights or oversampling the dataset[3]. In the final model neither of the two solutions was adopted because they seemed to bring no improvements. To face this problem we used a different approach based on Categorical Focal Loss.

3.3 Model architecture

Our first approach was trying to build **ad-hoc CNN** models (combined with a soft augmentation). These models showed promising results locally but constantly performed very poorly on Codabench's test set (see Table 1). These performances showed us that there was something wrong with our approach and we needed a drastic change.

Then we moved to **pre-trained** models to be used as backbones for a custom classifier like: VGG16, Xception, ConvNeXtBase, ConvNeXtLarge[4]. These models showed better performances since the beginning and they also enabled us to perform a **deeper manipulation of the layers** during the fine-tuning, reaching higher accuracies in the test set.

We started analysing in parallel the first three models in order to see if we could have obtained better performances with an huge model (like VGG16), a light one (Xception) or a new one (ConvNeXtBase). Over the same augmented dataset, ConvNeXtBase was the one that performed better, even if still poorly. For this reason we moved to a more aggressive augmentation thus to a more complex model like ConvNeXtLarge.

Model	Test accuracy	Codabench accuracy
Baseline	0.80 ± 0.06	0.15
VGG16	0.78 ± 0.05	0.38
Xception	0.87 ± 0.04	0.40
ConvNeXtBase	0.90 ± 0.01	0.56
ConvNeXtLarge	**0.98 ± 0.01**	**0.86**

Table 1: Accuracy of the tested models

3.4 Model Loss and optimizer

During the model selection and development phase, different **loss functions** and **optimizers** were considered with the aim of improving score and time performances.

Initially, we used the Categorical Cross Entropy **loss function** but we soon moved to **Categorical Focal Loss**. It is an evolution of the Cross Entropy Loss *designed to address scenario in which there is an imbalance between foreground and background classes during training*[1], then extended to multi-class case.

$$FL(p_t) = -\alpha_t(1-p_t)^\gamma log(p_t) \tag{1}$$

As shown in equation 1, the main difference with respect to the classical Categorical Cross Entropy Loss lies on the α-balanced modulating factor $(1-p_t)^\gamma$ that *reshape the loss function to down-weight easy examples and thus focus training on hard negatives*[1].

Regarding the optimizer, different method were evaluated and tested, passing from **Adam** to its last

[2]RandomFlip, RandomRotation, RandomTranslation,...

[3]Different methodologies were tried. From the most classical image duplication, to more complex solution such as SMOTE or BorderlineSMOTE for synthetic samples generation[3]

[4]Look at `https://keras.io/api/applications/` for models documentation

evolution like **Lion** and **AdamW**. The last one in particular seemed us to best suit the problem we were facing. Indeed, the main feature of AdamW is that it *decouples the optimal choice of weight decay factor from the setting of the learning rate [...] and substantially improves Adam's generalization performance [...] on image classification datasets*[2].

3.5 Fine-Tuning

The Fine-Tuning phase was meant to improve accuracy by updating the weights of the last layers of the pretrained model. Initially, we *unfroze* the last 10/20 layers but it didn't bring any improvement. To address this issue, we conducted further experiments by unfreezing a larger number of layers and selectively unlocking specific types of layers.

4 Final Model

4.1 Model architecture

The final model was composed of **ConvNeXt-Large pretrained** model (including finals **GAV**[5] and **Normalization** layers) as backbone for a custom classifier made of two **Dropout** layers interleaved with two **Dense** layers and a **BatchNormalization** layer.

4.2 Model Loss and optimizer

The loss function adopted in the final model was the **Categorical Focal Loss** 1. The main reason behind this decision was that we considered this solution as the best way for solving the dataset imbalance problem.

AdamW was the optimizer we choose for the final model. This decision was taken considering the huge model we had to train (especially during fine tuning). This solution allowed us to introduce into our model a more precise control over regularization, enhanced generalization and more stability.

In addiction, during the fine tuning phase we introduced a **Learning rate scheduler** in order to have a complete control over the learning rate updates during training. Indeed, customizing the LearningRateScheduler provided by keras, we were able to define an exponential decreasing behaviour of the learning rate which depended on the current training epoch.

[5]Global Average Pooling

4.3 Training and Fine-Tuning

The training phase was performed with a fixed learning rate of 10^{-4}, resulting in an accuracy of 83.07% on the validation set. Instead, in the fine-tuning phase, we *unfroze* the last **200 layers** of the network, focusing exclusively on dense and convolutional layers. The training was conducted with a dynamic learning rate, reducing it from 10^{-4} to 10^{-5}, resulting in a final accuracy of 98.22%. Both training and fine-tuning were carried out with a maximum of 400 epochs and an early stopping function, with a patience of 20, based on the maximum validation accuracy. Furthermore, we also implemented a **second fine-tuning** phase, unlocking only 100 layers; however, it did not lead to a significant improvement in accuracy.

5 Results & discussion

A good balancing between augmentations, dynamic learning-rate and early stopping patience made the model able to correctly classify **our** dataset with a good generalisation but this wasn't enough. Indeed, despite the excellent performances achieved by our model on local test set (98.37% accuracy), accuracy on Codabench reached just an 86%. A first consideration is that our model is still not able to generalize completely and so doesn't classify correctly a different dataset (such as the Codabech one).

Furthermore, we noticed that our model (like early trained models) wrongly classified a small part of Neutrophils as Immature granulocytes, maybe because of the similarities in the cells shape.

6 Conclusions

In order to parallelize the effort of training models with different features, we worked on the project together, progressing simultaneously throughout all phases of its development.

A future work may implement a more aggressive regularization in order to prevent the model to be prone to overfitting. Besides, the model could implement more augmentations to achieve better general results.

References

[1] *Focal Loss for Dense Object Detection*, Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár, 7 February 2018 [*arXiv:1708.02002v2*]

[2] *Decoupled Weight Decay Regularizationar*, Ilya Loshchilov, Frank Hutter, 4 January 2019 [*arXiv:1502.03167v3*]

[3] *https://imbalance-learn.org/stable/references/over_sampling.html)*