AN2DL - First Homework Report
ReLUsive Minds

Nicola Chiarani, Nastaran Ghaffari Elkhechi, Andrea Parenti, Alessandro Staffaroni Biagetti

nicolachiarani, Nastarangfr4, anpare, alestaffa

11001700, 10900702, 10620763, 10659399

November 24, 2024

1 Introduction

This report outlines the work completed for the first homework project of the "Artificial Neural Networks and Deep Learning" course at Politecnico di Milano. The project focuses on the classification of 96×96 pixel RGB images of blood cells into eight predefined categories.

2 Problem Analysis

Our dataset consisted of 13,759 images for the classification of blood cells into eight distinct classes: Basophil, Eosinophil, Erythroblast, Immature Granulocytes, Lymphocyte, Monocyte, Neutrophil, and Platelet. The images were 96 x 96 pixels in size and stored in RGB format. Initially provided in .npz format as NumPy arrays, we converted them to .png format using Python's Pillow (PIL) library for easier processing while preserving their labels.

During analysis, we identified several challenges. Certain classes, like Basophil and Lymphocyte or Erythroblast and Neutrophil, displayed similar shapes, complicating classification. Additionally, a histogram of class distributions revealed a significant imbalance, with some categories underrepresented, necessitating balancing strategies.

Using Image Hashing, we detected 1,800 duplicate images clustered at the end of the dataset, corresponding to only two unique hash indices. After removing these duplicates, the dataset was reduced to 11,959 unique images. This smaller dataset posed a higher risk of overfitting, reinforcing the need for robust preprocessing and augmentation methods.

To address the class imbalance, we explored both the SMOTE [1] technique and a random oversampling method, ultimately choosing the latter for two main reasons:

1. SMOTE relies on the interpolation between the k-nearest neighbors of a sample, but it's nontrivial to define what a neighbor of an image is due to the lack of spatial correlation between different images.
2. Under the consideration we would surely have to apply some kind of augmentation on data, simply injecting duplicates inside the dataset should not be a problem.

Given the relatively small size of our filtered dataset and the availability of the Codabench platform for testing, we split the dataset into 80% training and 20% validation sets. (We experimented with different splitting percentages, which will be discussed in the hyperparameter-tuning section).

After normalizing the pixel values to the range [0, 1], and applying one-hot encoding to our labels, we used several data augmentation techniques either together or using them separately. We used data augmentation to address the challenges of having an unbalanced dataset and limited training data.

These techniques aimed to expand the diversity of the dataset, helping to enhance the model's generalization capabilities and mitigate overfitting. So, we focused on enlarging the dataset as much as possible, to give to the feature extraction layer more information to work on. Here are the augmentation techniques we worked on:

- *ImageDataGenerator*: Augments datasets by applying transformations like rotation, zoom, flipping, brightness changes, and shifting [5].
- *AugMix*: Creates blended images with random augmentation chains while keeping semantic content intact.
- *CutMix*: Combines images by cropping a region from one and replacing it with a patch from another, blending images and labels [7]. However, we noticed that, in some cases, the patch applied to the original image was simply meaningless and the cropped-out region contained most of the discriminating features for the original class. Hence, we decided to move to Mixup.
- *Mixup*: Creates blended image-label pairs by linearly interpolating between two random images and their labels to improve robustness [8].
- *Random Translation*: Shifts images horizontally, vertically, or both within a range to enhance positional invariance.

3 Method

The method that we adopted to tackle this classification problem is an iterative one. We started with building and training custom CNNs and gradually transitioned to pre-trained models, leveraging transfer learning. Here we briefly list our attempts, to highlight the strategy that brought us to the final model.

4 Custom CNN

For our Custom CNN, we designed an architecture with four convolutional layers using filters ranging from 32 to 256, ReLU activation, *MaxPooling* layers, and dropout layers with rates of 0.25 for the convolutional layers and 0.5 for the classification layers. After these, we added a GAP layer followed by two dense layers: one with 256 neurons and the output layer with 8 neurons. We used *HeUniform* as the kernel initializer, categorical cross-entropy as the loss function, Adam as the optimizer, and categorical accuracy as the evaluation metric. Initially, the model showed expected overfitting, so we tried several approaches to address this issue:

- Increasing the validation split from 10% to 20%, which reduced fluctuations in validation loss and accuracy;
- Experimenting with stratified splitting and k-fold cross-validation, including stratified k-fold, though the latter two significantly increased training time;
- Applying L2 regularization with different lambda values (0.002, 0.0002, 0.005);
- Adding batch normalization and adjusting the Adam optimizer's learning rates (10^{-3} and 10^{-4});
- Testing different dropout rates and data augmentation techniques;
- Evaluating metrics beyond accuracy, such as F1 score, due to the unbalanced dataset;
- Assigning class weights during model fitting.

5 EfficientNet

We first used the *EfficientNetB0* [6] model with two dense layers: one with 256 neurons and the output layer with 8 neurons, applying a dropout layer before each of those to prevent overfitting. The *Adam* optimizer was used with a learning rate of 10^{-4}, along with early stopping. To deal with the imbalance in our dataset, we assigned class weights to give more importance to the minority classes. We also added simple random augmentations, like rotation, brightness adjustments, and flips. With this setup, we achieved a test accuracy of around 47%. Then, we switched to EfficientNetB2 using the same setup as before and this change increased the test accuracy to 51%.
We started tuning *EfficientNetB2*, gradually unfreezing layers starting from the last. The best results were achieved by unfreezing all layers and

adding the *ReduceLROnPlateau* callback (monitoring validation accuracy, with a factor of 0.5, patience of 3 and a minimum learning rate of 10^{-6} [3]). Using the same approach, we also tested other architectures including *ResNet50*, *Xception* [2], and *DenseNet121*, but *EfficientNetB2* performed the best. For augmentations, we added more advanced techniques beyond the simple ones mentioned earlier. The first was Mixup, implemented using keras_cv, which we tried in three different ways:

1. Replacing the entire original dataset with the mixed dataset;
2. Doubling the dataset by applying *Mixup* to one copy and merging it with the original dataset;
3. Applying *Mixup* to only 20-30% of the dataset.

Option 2 gave the best results. As a final step, we combined *RandomTranslation* with *Mixup* (Option 2), and with these techniques, we achieved a test accuracy of **72%**. Lastly, for the Fine-Tuning part, we tried a few other changes, but due to issues with Codabench, we couldn't test all the adjustments together. As a result, we were uncertain if some of these changes would have improved the results. The changes included:

- Using the *Lion* optimizer instead of *Adam*, testing different learning rates (0.0001, 0.00001, and 0.00005);
- Creating a local test set and adding white noise to the validation set, which made it slightly more accurate for testing the model locally;
- Experimenting with additional augmentations in keras_cv, such as *AutoContrast*, Cutout, *ChannelShuffle*, and *RandomShear*;
- Manually tuning class weights to give higher importance to classes with lower recall.

6 ConvNeXt

Another promising model was coming from the *ConvNeXt* family since its architecture gathers many key blocks from others models. We started working on the *Tiny* variant against a balanced (for the sake of *RandomOverSampler*) augmix-augmented dataset with both dropout and L2 regularization applied to each layer of the classifier made of a dense layer with 256 neurons, *HeUniform* kernel initialization and swish activation and the head with 8 neurons, *GlorotUniform* kernel initialization and softmax activation. This time the chosen optimizer was *Nadam*, which introduces the Nesterov Accelerated Gradient [4] into Adam, an early stopping callback with patience between 10 and 20 and we start to experiment with the *CosineDecayRestart* scheduler for the learning rate, trying values from 10^{-3} to 10^{-6}. After training the classifier only, we started unfreezing layers and found out that good performances and a feasible training time were both reached by unfreezing the third stage convolutional and dense layers of the pretrained model.
The same procedure was also repeated with the *Small* and *Base ConvNeXt* variants and the latest permormed the best, achieving a **73%** accuracy on the test set.

7 Final Model

Given the similar results obtained by the two models described above, we decided to build an ensemble classifier and take the mean of the two predictions in order to reduce both the bias and the variance of the two single models.
As expected, the ensemble performed better and achieved a **78%** accuracy on the test set.

8 Personal Contribution

All the members tried different solutions to build their own CNNs, using Transfer Learning and Fine Tuning, all of them tried to solve overfitting in different ways and tried different data augmentation techniques. In the end the two best models have been chosen for ensemble.

References

[1] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 2002.

[2] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of*

the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[3] K. Documentation. Keras api documentation: Cosine decay restarts, n.d.

[4] T. Dozat. Incorporating nesterov momentum into adam. *arXiv preprint arXiv:1502.07284*, 2015.

[5] P. Rana, A. Sowmya, E. Meijering, and Y. Song. Data augmentation for imbalanced image classification. *University of New South Wales (UNSW), Australia*, n.d.

[6] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for cnns. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[7] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[8] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *International Conference on Learning Representations (ICLR)*, 2018.