AN2DL - First Homework Report
TensorTribe

Matteo Figini, Caterina Motti, Andrea Grassi, Marco Gervatini

teofigio, cthequeen, andrea, shift

248094, 252240, 252516, 251749

1 Introduction

In the first homework of the Artificial Neural Networks and Deep Learning course, we faced an image classification problem. This homework aimed to classify different images representing white blood cells into 8 categories: basophil, eosinophil, erythroblast, immature granulocytes, lymphocyte, monocyte, neutrophil, and platelet.

The goal of the project was to build and train a multi-class classifier able to distinguish between different classes, by using deep neural network techniques. The quality of the designed model was to be assessed over a separate, not visible, test set. The main metric used for evaluating the model was the accuracy on the test set: other metrics (precision, recall, F1 score) were taken into consideration too.

2 Problem Analysis

2.1 Data Analysis and Cleaning

The dataset we were provided with consisted of 13759 96x96 images. Each image was labelled with an integer number from 0 to 7, representing the specific type of white blood cell.

First, we manually inspected the dataset to find possible outliers: indeed, all the images from index 11959 to index 13558 were mixed up with the background of the animated film character Shrek, while from index 13559 until the end, images were mixed up with the background of the singer Rick Astley. Thus, we removed the outliers from the dataset to have only clean images.

Afterwards, we ran a procedure to detect duplicated images with the same label; in the end, 8 images were duplicated in the dataset. Thus, we removed the duplicated copies, and the clean dataset had 11951 images.

2.2 Data Rebalancing

After cleaning the dataset, we noticed that the resulting classes were unbalanced in size: the largest class had 2330 members, whereas the smallest had about 800. To rebalance the dataset, we considered three different techniques: Undersampling, Oversampling, and SMOTE.

Among these techniques, the one that seem to provide the best result for rebalancing the dataset was SMOTE (Synthetic Minority Oversampling Technique) [1].

SMOTE performs a re-balancement of the dataset, aligning all the classes to the one which has the most number of samples, by generating new synthetic samples (in our problem, generating samples is equivalent to generating new tri-dimensional arrays 96 x 96 x 3 corresponding to images) by considering other samples of the same class. The general setting of SMOTE rebalancing is the following:

1. Take the difference between a sample and its nearest neighbour.

2. Multiply the difference by a random number between 0 and 1.

3. Add this difference to the sample to generate a new synthetic example in the feature space.

4. Continue with the next nearest neighbour up to the user-defined number, in our case the size of the most popular class.

2.3 Splitting Data

We split the data for the training and validation sets with a 70/30 ratio.

2.4 Augmentations

Dataset augmentations were the most critical part, as they improved model performance and generalization capabilities. We employed Keras layers to generate and save samples before training.
On the validation set, we used RandAugment to generate a broad range of samples.
On the training set, we used one specific augmentation at the time, concatenating it to the original one, to see its impact on the accuracy. Among the ones we tried, the following were the most promising ones: AutoContrast, RandomSaturation, Solarization and RandomCutout. Concatenating all the resulting images we end up with 65240 samples.
Once we identified the best-performing model, we applied RandAugment on the augmented training set to improve accuracy and robustness which led us to substantial gains in the competition result.
A sketch of the final training set, shown in Figure 1, illustrates the variety of augmented samples.

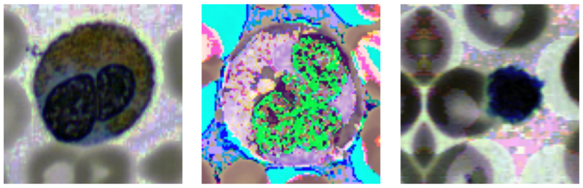

Figure 1: Images from augmented training set

3 Experiments

We started with a simple convolutional architecture designed from scratch.
To evaluate the loss in each epoch of the model we used the Categorical Cross-Entropy function, a standard choice for multi-class classification tasks, defined as follows:

$$CE = -\sum_{i}^{C} t_i log(s_i) \tag{1}$$

Where t_i is the ground truth for each class i, s_i is the CNN score for each class i and C it the total number of classes.
The final dense layer employed a Softmax activation function, which converts a vector of K real numbers into a probability distribution of K possible outcomes.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j}^{K} e^{z_j}} \tag{2}$$

Despite experimenting with filters, layers and configuration, we got only 0.40 accuracy on the test set. Those models were unsatisfactory but they helped us a lot in finding the most effective augmentations for the dataset.
We then moved towards transfer learning to try a bigger network without building it from zero.

3.1 Transfer Learning

Following the insights provided in [5] and [4] we were able to narrow our choices efficiently, focusing on nets that had already shown good performance on medical image analysis. We initially explored InceptionV3, VGG16, ResNet50 and DenseNet121. We appended only a minimal set of layers after the network, as our primary goal was to leverage the pre-trained model's capabilities.
The models were trained for 10 epochs with a batch size of 16, as we wanted a rough idea of which model was the best for our dataset. Among these models, DenseNet121 showed the best performance.

We further experimented with ConvNeXtBase and MobileNetV3Large, to evaluate the performance of a larger network with a better augmented dataset.
In summary:

- **DenseNet121** was the best network in terms of efficiency and performance. It could run in a couple of hours and yielded around 0.88 accuracy in the development phase.

- **MobileNetV3Large** is comparable to the DenseNet121 in terms of accuracy but with much lower computational cost.

- **ConvNeXtBase** outperformed all other models, achieving 0.91 accuracy; however, it is a lot more computationally demanding.

3.2 Fine Tuning

We implemented fine-tuning, a technique to adapt pre-trained networks to our specific dataset while focusing on learning relevant features. We've focused mainly on fine-tuning the DenseNet121 layer, which is composed of roughly 400 layers, including Convolutional2D and BatchNormalization layers.

Following the suggestion of the Tensorflow Fine-Tuning guide at this link, we implemented fine-tuning but leaving BatchNormalization layers frozen, while later on we tried to make also BatchNormalization layers trainable: indeed, as suggested by the theory, the validation accuracy with BN left as trainable was slightly worse than the one with BN layers frozen, as shown in Table 1. Later on, when we applied fine-tuning to the ConvNeXtBase model, this wasn't longer a problem, since the LayerNormalization layer is not affected by the same issue regarding the trainability of BatchNormalization.

To find the best setting for fine tuning we experimented with the number of layers and types to freeze, following the lines of [3].

Table 1: Different layers used in fine-tuning.

Type trainable	Accuracy
CNN	80.6%
BN	78.9%
CNN & BN	78%

In both transfer learning and fine-tuning, to prevent overfitting we employed early stopping (monitoring validation accuracy) and reduced learning rate on plateau (monitoring validation loss), with the callbacks provided by Keras.
We used AdamW optimizer [2] since it decouples the weight decay from the gradient updates, which prevents overfitting by penalizing large weights.
We also experimented with GridSearch to find the best hyperparameters, particularly for the learning rate and the weight decay.

4 Results and Discussion

Ultimately, among the models we've tried, the best working one was ConvNeXtBase, which achieved the maximum accuracy observed during this challenge. DenseNet121 and MobileNetV3Large followed closely, as summarized in table 2.

Combining multiple augmentations contributed to improving robustness and better generalization: we can state that our model seems to be good at generalizing since it performed well both in the development phase and in the test phase, which relied on different datasets.

Table 2: Accuracy on the final phase test set.

Model	Accuracy
ConvNeXtBase	91%
DenseNet121	87%
MobileNetV3Large	86%

5 Conclusions

In this project, we successfully developed a deep-learning model to classify white blood cells into eight categories. While our model achieved high accuracy, there is still room for improvement, especially regarding computational efficiency. Future work could aim to achieve similar performance levels while reducing complexity.

Team contributions:

- **Figini Matteo**: Data Cleaning and preprocessing, basic CNN training, model fine-tuning with ConvNeXtBase and MobileNetV3Large.

- **Gervatini Marco**: Data Augmentation, model training with DenseNet201, integration of callbacks.

- **Grassi Andrea**: Transfer learning (DenseNet121), fine-tuning, data augmentation, GridSearch for hyperparameters optimization.

- **Motti Caterina**: Data Augmentation, transfer learning with multiple architectures (DenseNet121, VGG16, ResNet50, InceptionV3, ConvNeXtBase), fine tuning DenseNet121.

References

[1] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res. (JAIR)*, 2002.

[2] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. 2017.

[3] P. Peng and J. Wang. How to fine-tune deep neural networks in few-shot learning? 2020.

[4] O. Saidani, M. Umer, N. Alturki, A. Alshardan, M. Kiran, S. Alsubai, T.-H. Kim, and I. Ashraf. White blood cells classification using multi-fold pre-processing and optimized cnn model. *Scientific Reports*, 2024.

[5] M. Sharma, A. Bhave, and R. Janghel. White blood cell classification using convolutional neural network: Methods and protocols. 2019.