POLITECNICO
MILANO 1863

AN2DL - First Homework Report
Team Name: Hyper Parameters Guessers

Paolo Biolghini, Loris Di Girolamo, Sara Cavallini, Riccardo Camellini

Codabench: fazpaolo, Codabench: lorisdigirolamo, Codabench: saracava, Codabench: riccardo

246816, 249851, 244980, 244779

November 24, 2024

1 Introduction

This project focuses on *image classification* using **deep learning** techniques. The dataset provided was a challenging, unclean dataset of blood cell images. The primary objective was to develop a neural network capable of accurately classifying these images. The key steps in achieving this included:

- **Data preparation**: Cleaning the dataset and applying normalization techniques to ensure consistency and improve model performance.
- **Data augmentation**: Enhancing the model's generalization capabilities through various transformations. This was particularly crucial, as some models performed well on a local test set but struggled with the competition test set.
- **Hyper-Parameter tuning**: Experimenting with different configurations of the model's architecture and hyperparameters to optimize performance.

A noteworthy augmentation technique involved non-standard **saturation adjustments**, which allowed clearer extrapolation of blood cell shapes, significantly boosting performance. We further investigated the application of various **transfer learning** techniques, including the use of state-of-the-art convolutional neural network architectures of the *EfficientNet*, *ResNet*, and *ConvNeXt* families.

2 Dataset and Preprocessing

The given dataset was made by 13759 of blood cell images of shape (96,96,3) and the corresponding labels ranging from 0 to 7. Among the correct images there were several clearly outliers, like images mixed with *"Rick Astley"* and *"Shrek"*, and some other more hidden, such as same images (spotted with image hashing technique) but with different labels. In order to evaluate our architecture, given the inaccessibility of Codabench, we created a local test set where the images were augmented with white noise to better evaluate the model's ability to generalize. With the remaining data, we split them into training and validation sets when creating a single model. Alternatively, for improved robustness, we used 5-fold cross-validation, which allows for better generalization by combining the results through majority voting.

3 Data Augmentation

For data augmentation, we explored different approaches:
In the first approach, we consistently applied commonly used random transformations, such as rota-

tion, horizontal and vertical flips, resizing, cropping, and zooming. These augmentations help generalize the model over variations in the position and size of the cells.

For generalizing over colors, we tested two techniques:

- Random alterations to the color features of the images, including adjustments to brightness, contrast, hue, random channel shifts, and occasional conversion to grayscale.
- An ad-hoc saturation and contrast technique using the `convertScaleAbs` function in OpenCV with parameters $\alpha = 1.5$ and $\beta = 0$.

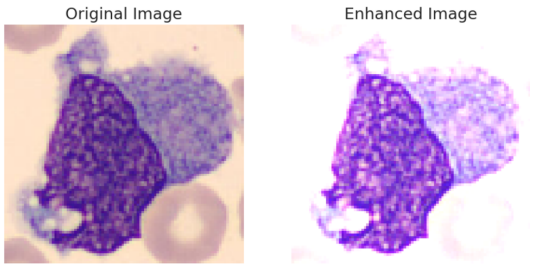

Figure 1: Image Augmentation

The final technique that we used is implemented using the RandAugmentation Layer, as suggested in the notebook, with a magnitude of 0.32 to achieve a balance: the images are augmented sufficiently to promote good generalization while avoiding excessive distortion.

4 Method

Our method is based on creating a neural network that leverage the usage of the transfer learning techniques. We used an input layer with a shape of (96, 96, 3) and an output layer consisting of 8 neurons with a Softmax activation function.

As regularization techniques we mainly utilized early stopping, dropout layers and batch normalization to stabilize the intermediate values within the neural network.

In order to find the best weights, we adopt the Lion optimizer using learning rate equal to 0.001 and a categorical cross-entropy loss function.

$$\mathcal{L} = -\sum_{i=1}^{N} w_i y_i \log(\hat{y}_i) \tag{1}$$

The loss function incorporates class weights, which ensure high-quality learning even when the classes are imbalanced.

$$w_{t+1} = w_t - \eta \cdot \text{sgn}(m_t) \tag{2}$$

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla f(w_t) \tag{3}$$

(1) Categorical cross-entropy loss function. (2) Lion Weight update rule. (3) Lion Momentum update rule.

5 Experiments

Reaching our final solution involved some struggle as we navigated the maze of possible combinations of data augmentation techniques and neural network architectures.

Using the ad-hoc augmentation we were fairly confident that it was a good model because with VGG16 and 2 dense layers we were able to reach over the 90% if accuracy. Unfortunately, this occurred during a downtime of Codabench, so we couldn't verify our hypothesis. Given that it turned out to be a poor solution, obtaining only 31% accuracy on Codabench, we decided to try RandAugmentation, albeit with limited time remaining.

To determine an optimal magnitude, we experimented with different values, observing how the images changed and assessing whether they remained usable for prediction. We eventually set the magnitude to 0.32 and the number of augmentations per image to 5.

For the architecture exploration, although we knew that transfer learning would likely yield better results, we initially experimented with some simple architectures as well as state-of-the-art models without pretrained weights. As we were expecting, these approaches resulted to have low accuracy.

The next step was trying different architectures implemented in `keras.applications` with pretrained weights on the ImageNet dataset. We primarily focused on VGG16 and ResNet50, and then on the EfficientNet and ConvNeXt family. The pretrained architecture used in our final model was ConvNeXtBase, as it provided an excellent trade-off between performance and model size. It delivered better results and was also one of the top-performing models according to the documentation [1].

The final step was to find the optimal hyperparameters for the number of dense layers, number of

neurons, dropout rate and number of unfreezing layers. To find a good solution, we manually performed part of the tuning, achieving 90% accuracy on the local test set. Additionally, we used a network architecture search tool called Optuna that allowed us to automatically search for the best combination of hyperparameters that maximizes the accuracy on the test set. we were able to reach 93.27%

In table, a resume of some experiments

6 Results

The final model we obtained consists of an input layer that accepts inputs of shape (96, 96, 3), followed by the ConvNeXtBase model with pretrained weights and 15 unfreezed layers. After that, there are three dense layers with 512, 256, and 128 neurons, with dropout rates of 0.5, 0.4, and 0.3, respectively. All dense layers use ReLU activation and batch normalization, while the output layer has 8 neurons with softmax activation.

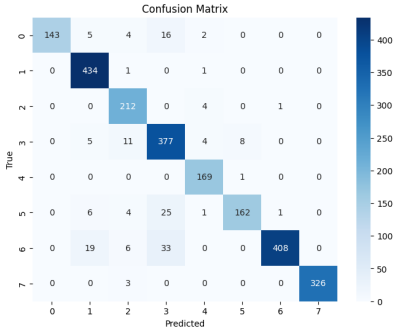

Figure 2: Confusion matrix of ConvNeXtBase model

7 Conclusions

During our journey, we occasionally got stuck on beliefs that seemed correct and led us to good results. The inability to test these beliefs further only hindered our progress. We began making remarkable progress once Codabench was unfrozen, but the limited number of model submissions still posed a significant constraint.

In the end created a model that performed quite well on our test set, achieving an accuracy of 93.27%. Despite this, our model still struggles with generalization, as evidenced by the result of 71%. This outcome could be attributed to our augmentation technique, which may not have been optimized for better generalization. Additionally, we may not have found the best combination of hyperparameters for our model.

To improve our model, we could explore other augmentation techniques, such as AugMix, experiment with different architectures by adjusting the hyperparameters, try different optimizers, and consider changing the pretrained network.

Another potential improvement would be to use stratified splitting to create multiple models with the same hyperparameters and then apply a majority voting strategy to combine their predictions.

8 Contribution

Every person in this team collaborated to reach the final solution, everyone with different responsibilities:

Loris is responsible for data cleaning, data augmentation, and network structure exploration. Paolo focuses on data augmentation, network exploration, and automated hyperparameter tuning. Sara handles hyperparameter tuning, testing, and network structure exploration. Riccardo is dedicated to hyperparameter tuning and model testing.

Table 1: Experiments. Best results are highlighted in **bold**.

Model	Accuracy	Precision	Recall	F1 Score
EfficientNetV2B0	70.56 ± 2.40	65.29 ± 1.19	54.34 ± 0.86	53.85 ± 0.95
ConvNextTiny	82.60 ± 1.19	73.52 ± 1.02	69.28 ± 1.15	68.68 ± 1.09
EfficientNetV2B0 (5 Fold)	82.27 ± 0.57	84.14 ± 0.78	82.27 ± 0.68	82.25 ± 0.72
VGG19	89.07 ± 1.19	89.67 ± 0.52	89.07 ± 1.15	88.86 ± 0.76
ConvNeXtBase	**93.27 ± 0.85**	**93.77 ± 0.47**	**93.27 ± 0.98**	**93.26 ± 0.72**

References

[1] Keras. Keras applications. https://keras.io/api/applications/, 2024. Accessed: 2024-11-24.