POLITECNICO
MILANO 1863

AN2DL - First Homework Report
Johnny Deep (Learning)

Andrea Giangrande, Marta Giliberto, Emanuele Greco,
andreagg01, marta23gili, lelegreco2002,
252647, 258876, 252080,

November 23, 2024

1 Introduction

The aim of this project is to develop and test a **deep learning model** to solve a *multi-class classification problem* of 96x96 RGB images of blood cells, in the most effective and accurate way. The model will need to load an unseen dataset and be able to correctly classify the provided images into the corresponding classes. Other factors to consider are the reliability of the model and its computational cost, since the goal is to get the **most reliable** solution while **preserving storage** occupancy. The platform **CodaBench** was used to test the model on an unknown, possibly augmented, dataset.

2 Problem Analysis

The primary operation performed was to inspect and analyze the dataset: it was made of roughly 14 thousand images and labels, no consideration about class balancing was done. After plotting the dataset, some **outliers** were noted and removed, with a subsequent split of the dataset into 80% train set, 10% validation set and 10% test set to maximize the train set volume. Expecting the final test set to be different from our training dataset, the use of **augmentation** was taken into account. The initial augmentation was a very simple sequence of transformations, which was substituted afterwards by a more complete pipeline. This pipeline included various layers, applied sequentially: *AugMix*[2], *GridMask*[5], *RandAugment*[14], *ChannelShuffle*[3], *RandomHue*[15], *RandomShear*[16], *RandomFlip* and *RandomRotation*.

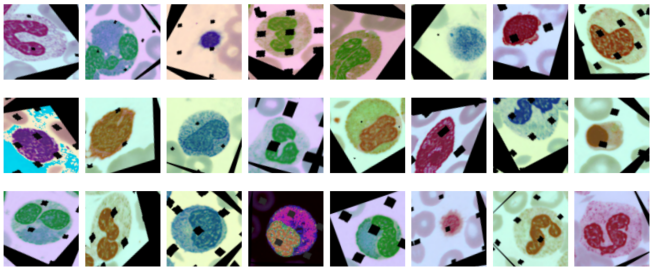

Figure 1: A visualization of the augmentation applied

3 Method

1. The primary idea was to try to **build a CNN**, with a structure resembling a *VGG16*[12], but with fewer layers. This CNN included $\mathcal{L}2$ *regularization* to reduce overfitting, *Adam optimizer*[1], *EarlyStopping*[4], *ReduceLROnPlateau*[17] and **no augmentation**. The goal of this initial test was to see how "off" a standard model was with the unknown test set; local runs were successful, but given the real test set nature, the model could achieve just 0.26 accuracy on CodaBench.

2. The following move was to add a **simple augmentation** code inside the model struc-

ture, before all the layers, similar to the one seen at the exercise sessions, and train the model directly on the full augmented train set. The local results were poor, the CNN was not able to generalize its knowledge, probably because of the constantly-changing train set. Therefore, the "self-made" CNN was ditched in favor of **transfer learning**. At this point, a decision was made to **divide the team's efforts** and to ensure comprehensive case coverage: each one of the team members tried a different pre-trained model, but with the same parameters (batch size, learning rate, callbacks etc.). After a thorough documentation read[6] about all the possible models adoptable, the chosen ones were **MobileNetV3**[10], **ResNet50V2**[11] and **InceptionV3**[9].

3. The 3 models have been trained with **transfer learning**, both on augmented and non augmented datasets, without significant improvements. Therefore, the subsequent attempt was to change the optimizer: the choices were **SGD**[18], as it was more appropriate for multi-class problems, and **Lion**[13], as it converged faster with noisy datasets. After running some simulations locally, not much progress was made since the models were not able to correctly learn from the augmented dataset.

4. Eventually, the focus shifted towards **EfficientNetB7**[7]; from the very beginning, even with no augmentation, this model performed well with the test set, so all resources were invested on it. **Weighted Loss** was also applied: at first, the model seemed to benefit from it, returning higher performances and reaching up to 77% accuracy with the final test set. This model was heavier than the predecessor and, consequently, the training time grew immensely. In order to mitigate it, the application of the **augmentation pipeline** was moved from the model's layers to the preprocessing phase. This allowed to expand the initial dataset, adding to it its augmented version and thus doubling or tripling its size. The final improvements involved expanding the training set, upgrading the model to **EfficientNetV2M**[8], switching the optimizer to Lion, and removing the Weighted Loss, which had proven counterproductive in some cases, negatively affecting the model's performance. The results obtained were promising, being able to obtain **up to 89% accuracy** with the real test set on CodaBench.

4 Experiments

Table 1: Models tested with relative results obtained **LOCALLY**, with and without Augmentation (**Aug.**)

Model	Validation Accuracy	Test Accuracy
Custom VGG16	98.41%	98.58%
Custom VGG16+**Aug.**	22.41%	22.83%
MobileNetV3	92.64%	91.05%
MobileNetV3+**Aug.**	74.16%	74.41%
ResNet50V2	88.71%	90.13%
ResNet50V2+**Aug.**	41.56%	43.65%
InceptionV3	18.14%	17.47%
InceptionV3+**Aug.**	14.30%	14.72%
EfficientNetV2M	97.83%	**97.66%**
EfficientNetV2M+**Aug.**	98.66%	**98.41%**

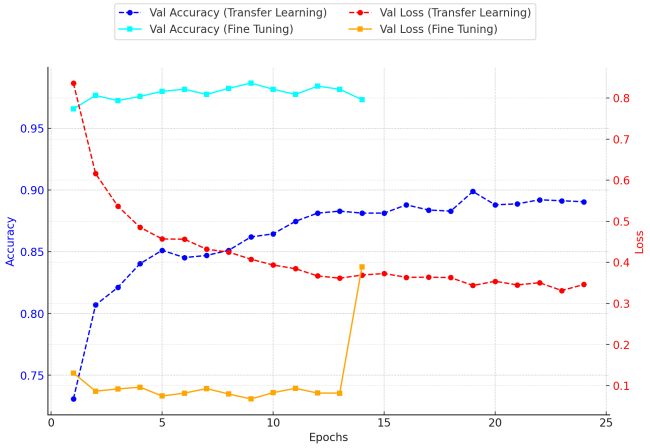

Figure 2: Graphical representation of EfficientNetV2M training

5 Results

Given the accuracies shown in Table 1, it is noticeable how all the models, **without augmentation**, are fully capable of learning from the clean dataset and achieve very high accuracies on the local

test set. However, the performances on CodaBench were not satisfying. When **augmentation** comes in place, the differences between models are significant: the **Custom VGG** performed poorly, similarly to **InceptionV3**, whose performance was notably underwhelming. On the other hand, **ResNet50V2** and, most importantly, **EfficientNetV2M**, showed stronger performances both locally and on CodaBench. Also applying augmentation during the **pre-processing** instead that in a dedicated layer seemed to not affect too much the training abilities of the model.

6 Discussion

- **Architecture**: The comparison of various models proved to be highly beneficial in identifying a robust architecture such as **EfficientNetV2M**. This represents a significant strength, as once this model was selected, the focus shifted to fine-tuning **hyperparameters** and selecting the best **optimizer**, consistently achieving high levels of accuracy.

- **Weight loss**: The use of **weight loss** proved to be a double-edged sword. Initially, it appeared to **enhance** model **accuracy** consistently by balancing the dataset. Unexpectedly, its **removal** lead to **better results**. This highlights a potential limitation of the project: overlooking the impact of **data imbalance**. It is possible that applying balancing with **advanced techniques** could improve the performance of the model.

- **Augmentation**: A particularly noteworthy aspect of the project was the approach taken to augment the images of the dataset, which was lately used to expand it. Augmentation was performed through an advanced data augmentation pipeline which combined KerasCV Layers and other **geometric** and **chromatic transformations**, that ultimately allowed the creation of a richer and more diverse dataset. This strategy significantly enhanced the model's ability to generalize, enabling it to learn more robust and transferable features, even for unseen data. This approach reduced the **risk of overfitting** to a limited or overly specific dataset. By concatenating **multiple layers of augmentation**, the final dataset became highly diversified, offering the model multiple perspectives on the same problem. As a result, it was possible to achieve significantly higher model accuracy compared to the initial results obtained with the original, not augmented dataset.

7 Conclusions

This project successfully addressed the task of multi-class classification. The work was evenly split across the group member: the three main focus (one for each member) were:

- Initialize the whole project with the basics functions, such as preprocessing functions, splitting and analyzing the dataset, define the basic structure of the model, testing the first models and identify the best direction to undertake (key contributor: **lelegreco2002**)

- Apply and adapt the basis for transfer learning and fine tuning, testing the model locally (key contributor: **andreagg01**)

- Find and apply the best optimization and improvement for the model, such as finding the best model to use for transfer learning and test weight loss functions (key contributor: **marta23gili**)

The key factor was identifying **EfficientNetV2M** as the most effective model, developing an advanced data augmentation pipeline to augment and expand the dataset, and leveraging **transfer learning** and hyper-parameter **optimization** to achieve a test accuracy of 89% on CodaBench. The systematic testing of models allowed for a **robust comparison** and optimization process. However, challenges such as handling class imbalance and the underperformance of certain pre-trained models like InceptionV3 highlighted areas for **improvement**. **Future work** could focus on exploring advanced techniques for balancing datasets, creating custom augmentations with even more advanced techniques, and testing additional lightweight models to enhance computational efficiency while maintaining high accuracy.

References

[1] K. Team. Adam Optimizer Documentation, 2024.

[2] K. Team. AugMix Layer documentation, 2024.

[3] K. Team. ChannelShuffle Layer Documentation, 2024.

[4] K. Team. EarlyStopping Callback Documentation, 2024.

[5] K. Team. GridMask Layer Documentation, 2024.

[6] K. Team. Keras Applications Documentation, 2024.

[7] K. Team. Keras EfficientNetB7 Documentation, 2024.

[8] K. Team. Keras EfficientNetV2M Documentation, 2024.

[9] K. Team. Keras InceptionV3 Documentation, 2024.

[10] K. Team. Keras MobileNetV3 Documentation, 2024.

[11] K. Team. Keras ResNet50V2 Documentation, 2024.

[12] K. Team. Keras VGG16 Documentation, 2024.

[13] K. Team. Lion Optimizer Documentation, 2024.

[14] K. Team. RandAugment Layer Documentation, 2024.

[15] K. Team. RandomHue Layer Documentation, 2024.

[16] K. Team. RandomShear Layer Documentation, 2024.

[17] K. Team. ReduceLROnPlateau Callback Documentation, 2024.

[18] K. Team. SGD Optimizer Documentation, 2024.