AN2DL - First Homework Report
GradientGang

Luca Bordin, Cosimo Giovanni Negri, Mattia Menegale,
lucabordin, cosimonegri, mattymene,
272482, 278015, 273813

November 24, 2024

1 Introduction

This project focuses on *image classification* using *deep learning* techniques and *convolutional neural networks*. Our approach consisted into:

1. Cleaning the dataset
2. Developing a custom **CNN** from scratch
3. Gradually proceeding towards more advanced models, applying ***Transfer Learning*** and ***Fine-Tuning*** techniques

2 Problem Analysis

The provided dataset consists of **13759 RGB** images of **96x96** pixels, labeled into **8 classes** representing different types of blood cells (as shown in Figure 1).

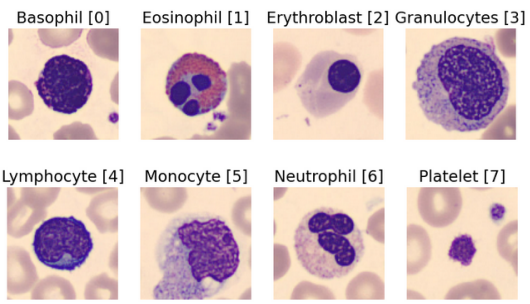

Figure 1: Blood cell types

In order to develop a robust image classifier, we needed to address some challenges related to the dataset:

1. We removed **1800 contaminated images** and **8 duplicated images**. In order to find duplicates we hashed every image in the dataset and looked for duplicated hashes.
2. We observed **class imbalance** in the dataset (as shown in Figure 2) and decided to later address it using using **class-weight**[1] to penalize the misclassification of underrepresented classes.

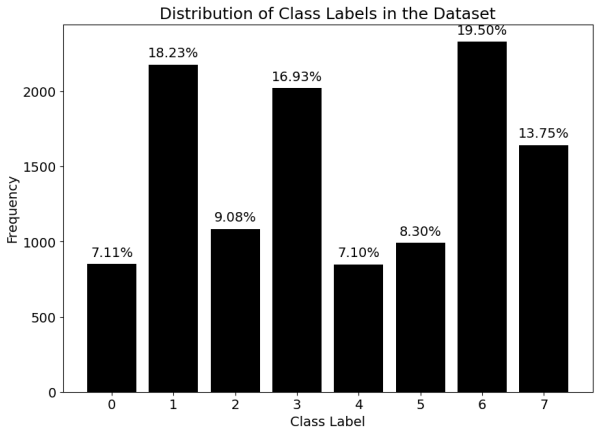

Figure 2: Distribution of images across the 8 classes in the cleaned dataset.

Data contamination would lead to a **poor performance** on real data, whereas duplicate data and class imbalancement are primarily related to **overfitting**.

3 Method

3.1 Custom CNN

We split the dataset into a training set (90%) and a validation set (10%), and set up an **Early Stopping** callback in order to avoid overfitting during training.
Then we developed a custom CNN with 4 blocks, each composed of 3 layers:
- A **Convolutional** layer to detect specific features.
- An Activation layer with **Relu [2]** to add nonlinearity.
- A **MaxPooling** layer to reduce the spacial dimensions of the feature maps.

After the convolutional blocks, we added a **Flatten** layer to convert the feature maps into a 1D vector to feed into the fully connected layers, which consisted by 2 **Dense** layers, the last followed by an Activation with **softmax** for class predictions among the 8 classes.
To mitigate overfitting, we also addedd a **Dropout**[6] layer after the Flatten layer and each Dense layer, ensuring better generalization to unseen data.
Then we added a basic **augmentation**[9] pipeline consisting of the following layers: horizontal and vertical RandomFlip, RandomTranslation, RandomRotation and RandomZoom.
We also experimented different weight regularization techniques (such as L1 and L2) with different values of the regularization parameter λ (0.01, 0.001, 0.0001), reaching the best results with **L2 regularization** and $\lambda = 0.001$.
Finally, we submitted our best custom CNN model, achieving a test accuracy of **24%**.

3.2 Model with Transfer Learning

Then we decided to move to Transfer Learning to exploit pretrained networks. We started by using MobileNetV3Small, with weights pre-trained on ImageNet to provide feature extraction capabilities. We replaced its fully connected layers with the ones we tailored in the custom CNN (as detailed in section 3.1), obtaining an accuracy of **41%** on the test set. We attempted to increase the number of neurons in the Dense layers, but this did not result in any significant improvement in accuracy. We hypothesized that the limited performance was due to MobileNetV3Small having too few parameters to effectively learn the complexity of the dataset, so we decided to try other networks such as: EfficientNetV2S, EfficientNetV2M and ConvNeXtSmall. After hyperparameter tuning, we achieved a test accuracy of 59% with both **EfficientNetV2M [7]** and **ConvNeXtSmall [5]**. Instead, with **EfficientNetV2S [7]**, we obtained 61%. For this reason we decided to continue with **EfficientNetV2S**, as it also has fewer parameters than the other two models and takes less time to train.

3.3 Model with Fine-Tuning

Starting from the last developed Transfer-Learning model (**EfficientNetV2S**), we followed a step-by-step fine-tuning strategy in order to enhance its performance.
Initially, we froze its first 480 layers, leaving only the convolutional layers in the last 30 layers trainable. This approach increased the test accuracy to **70%**, while also reducing the risk of overfitting.
As we thought our model was still not generalizing well on images different from those in the dataset, we introduced new data augmentation strategies to enrich the training dataset: **RandAugment** [3] and **AugMix [4]**. We started with a light augmentation strenght, and we gradually increased it while also unfreezing more layers in order to avoid underfitting. We achieved the best results with magnitude = 0.5 for RandAugment, severity = 0.3 for AugMix and **all the layers unfrozen**. This approach significantly boosted the test accuracy to **87%**.
Finally, we decided to slightly increased the augmentation by adding **RandomBrightness [11]** and **RandomContrast [12]**, achieving our best performance with a final test accuracy of **89%**.

4 Failed Experiments

During our experiments, we tested various strategies and techniques. However, some approaches did not yield the expected results:

- **Lion optimizer [10]:** did not improve ei-

ther accuracy or training speed compared to Adam.

- **Batch Normalization and Group Normalization [13]:** did not yield the desired results, likely due to the specific configuration of the hyperparameters in our setup.
- **Combination of CutMix and MixUp [3]:** the combined use of these augmentations negatively impacted performance.
- **Global Average Pooling [8]:** did not improve classification accuracy, likely because it reduced the model's capacity to learn complex features.
- **Leaky ReLu [2]:** failed to provide any noticeable improvement over ReLU.

5 Results & Discussion

Our work highlights the progression from a custom CNN to pretrained models refined through fine-tuning. The **custom CNN** initially achieved only **24%** test accuracy, probably because it was not able to generalize well on images different from those in the dataset. At the very end, we re-trained that same model applying some light RandAugment and AugMix augmentation. This improved its performance significantly, reaching **44%** test accuracy (as shown in the Table 1).
The usage of a pretrained model and of strong augmentations significantly improved accuracy, with the fine-tuned model based on **EfficientNetV2S** performing our best scoring **89%** test accuracy (as shown in the Table 1).
At the very end we also tried to train that same model using the bigger **ConvNeXtLarge [5]** instead of EfficientNetV2S, but it only achieved a test accuracy of 85% (as shown in the Table 1).

6 Conclusion

This work demonstrates the effectiveness of combining pretrained models with fine-tuning, and the importance of augmentation to build a strong and robust model. In fact, as shown in Table 1, there is a significant gap in test accuracy between the Custom CNN and the other models.
While the results are encouraging, further improvements could include:

- Increase the train set size by duplicating images and applying different transformations to them with RandAugment and AugMix.
- Implement a Learning Rate Scheduler to improve convergence.

7 Authors' Contributions

- **Luca Bordin**: Implemented basic augmentation techniques and tested pretrained architectures.
- **Cosimo Giovanni Negri**: Developed and tested advanced augmentation strategies, such as AugMix and RandAugment.
- **Mattia Menegale**: Cleaned the dataset and tested various normalization techniques (BatchNorm, GroupNorm, etc.).

All authors contributed to the overall analysis, development, and optimization of the models.

Table 1: Final models. Best results are highlighted in **bold**.

Model	Validation Accuracy (%)	Augmented Validation Accuracy (%)	Test Accuracy (%)
Custom CNN	90.05	37.54	44
ConvNeXtLarge + fine tuning	96.91	81.61	85
EfficientNetV2S + fine tuning	**97.91**	**90.38**	**89**

Note: The *Augmented Validation Accuracy* is computed on a strongly augmented validation set. We noticed that this provided a more accurate estimate of the test accuracy.

References

[1] S. learn Team. Class weight computation documentation. https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html.

[2] J. C. Olamendy. Understanding relu, leakyrelu, and prelu: A comprehensive guide. https://medium.com/@juanc.olamendy/understanding-relu-leakyrelu-and-prelu-a-comprehensive-guide-20f2775d3d64, 2023.

[3] K. Team. Augmentation - keras cv. https://keras.io/guides/keras_cv/cut_mix_mix_up_and_rand_augment/.

[4] K. Team. Augmix - keras cv. https://keras.io/api/keras_cv/layers/augmentation/aug_mix/.

[5] K. Team. Convnext - keras cv. https://keras.io/api/applications/convnext/#convnextsmall-function.

[6] K. Team. Dropout layer documentation. https://keras.io/api/layers/regularization_layers/dropout/.

[7] K. Team. Efficientnetv2 - keras cv. https://keras.io/api/applications/efficientnet_v2/#efficientnetv2s-function.

[8] K. Team. Globalaveragepooling - keras cv. https://keras.io/api/layers/pooling_layers/global_average_pooling2d/.

[9] K. Team. Image augmentation layers documentation. https://keras.io/api/layers/preprocessing_layers/image_augmentation/.

[10] K. Team. Lion - keras cv. https://keras.io/api/optimizers/lion/.

[11] K. Team. Randombrightness - keras cv. https://keras.io/api/layers/preprocessing_layers/image_augmentation/random_brightness/.

[12] K. Team. Randomcontrast - keras cv. https://keras.io/2.17/api/layers/preprocessing_layers/image_augmentation/random_contrast/.

[13] L. Zhu. Groupnorm, then batchnorm, instancenorm, layernorm. https://medium.com/@zljdanceholic/groupnorm-then-batchnorm-instancenorm-layernorm-e2b2a1d350a0, 2023.