AN2DL - First Homework Report
aaa000

Kamil Hanna, Giorgio Negro, Enrico Tirri

kamilhanna, giorgionegro, enti243

10932895, 10766662, 10742316

July 4, 2025

1 Introduction

The objective of this homework is to develop a classifier using Convolutional Neural Networks:

- The classifier must accurately classify 96x96 RGB images of blood cells into one of eight categories, with each category corresponding to a specific cell state.
- This is a single-class classification problem, meaning the classifier must assign exactly one label to each image.

Throughout the project, the group's primary objectives have been:

- **Analyzing and Exploring** the provided dataset
- **Designing** a structured problem-solving pipeline
- **Testing and Evaluating** various approaches to address the problem
- **Enhancing** the performance of the most promising models

2 Problem Analysis

The initial dataset contains $13,759$ samples
Upon examination, two significant outliers were identified within the dataset as shown in figure 1

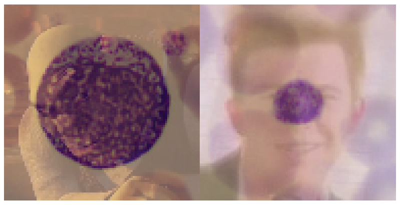

Figure 1: Outliers in the training set, with $1,600$ and 200 samples respectively.

After removing these outliers, the *cleaned dataset* consists of $11,959$ samples, which remain unevenly distributed across the classes as shown in figure 2

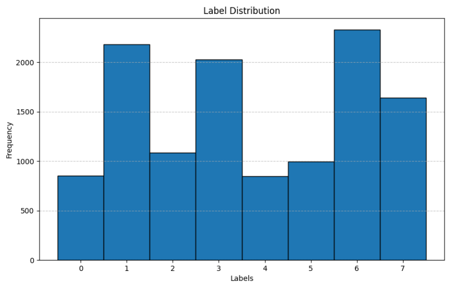

Figure 2: Label distribution in the cleaned training dataset.

We initially assumed that the test set would be similar to the cleaned training set, considering factors such as translations, rotations, zoom, and variations in colour saturation and contrast. However, we soon discovered that this approach was insufficient, as the results were poor until more aggressive image augmentations were considered.
Reference augmentation shown in image 3

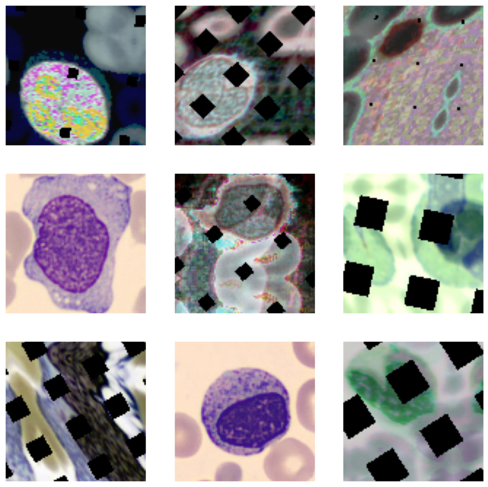

Figure 3: Example of training set

3 Development

Our initial development phase began with a simple CNN and low augmentation, which yielded unsatisfactory results. In response, we attempted to build more complex models with a large number of layers and extended training sessions. However, these approaches failed to produce meaningful improvements.

We then explored transfer learning by leveraging pre-trained ImageNet models. Each team member tested different architectures, and we compared the results. Among the models we evaluated were **VGG16**, **InceptionV3**, **EfficientNetB0V2/B3V2**, and **ConvNeXt**. Initially, **InceptionV3** appeared to be the best-performing model, achieving a Codabench score of 0.38. However, this result was still inadequate for our needs.

Next, we developed several data augmentation pipelines. While these pipelines were effective at increasing data diversity, we encountered significant performance issues due to the lazy loading mechanism in TensorFlow's dataset. Although lazy loading enabled dynamic random augmentations during each epoch, it severely slowed down the training process. To address this issue, we pre-applied all augmentations to the dataset, effectively "freezing" the augmented images before training. This adjustment improved efficiency and allowed us to achieve a score of 0.70 (before fine-tuning) and 0.75 after training with the frozen augmentations, as a side note with this more severe augmentation **ConvNeXt** came on top of our testing followed by **EfficientNetB3V2**.

To further enhance performance, we introduced several modifications:

- Enlarged the dense layer following the ImageNet feature extractor.
- Applied **L2 regularization** during coarse training and **L1L2 regularization** during fine-tuning for the dense layer.
- Added a **dropout layer** to reduce overfitting.
- Specified the `'avg'` pooling mode for the ConvNeXt model to improve feature aggregation.
- Added class weight to handle class imbalance in the train set [3]

While pursuing this development process, we explored various alternative architectures, but these were unable to deliver satisfactory results on the test platform. Among the approaches we tested were an **ensemble of models** [6], a **Siamese network** [12], and a **preprocessing technique** designed to isolate the cell [10]. Despite their potential, these methods did not outperform our primary model, and we ultimately focused our efforts on refining the ConvNeXt-based architecture.

These changes collectively improved our performance, achieving a final Codabench score of 0.86.

4 Method

Our approach leverages a Convolutional Neural Network (CNN) model to classify 96x96 RGB images of blood cells into one of eight categories. The pipeline consists of data preprocessing, augmentation, model design, training, and fine-tuning. To apply some augmentation and to handle class imbalance, labels were converted into categorical formats.

4.1 Augmentation

Augmentation was a crucial step in our approach to address data variability and mitigate overfitting. This process included the following techniques:

- **MixUp [14] & CutMix [13]:** Merging pairs of images and their corresponding labels with weighted combinations to enhance model robustness.
- **Channel Shuffling:** Randomly rearranging image channels to improve generalization.
- **RandAugment [5] & AugMix [7]:** Applying a range of geometric and colour transformations to enhance data diversity.

This step was critical not only for improving model generalization to real-world scenarios but also because the reference dataset is likely **highly augmented**.

4.2 Model Architecture

For the final submission we adopted the **ConvNeXtBase** architecture pre-trained on ImageNet as the backbone, excluding its top classifier layers. A custom classification head was designed with the following components:

- **Input Layer:** Processes 96x96 RGB images.
- **ImageNet Backbone:** Pre-trained on ImageNet, with the default classifier head removed.
- **Dense Layers with SELU Activation:** One or more fully connected (dense) layers using the **Scaled Exponential Linear Unit (SELU)** activation function [9].
- **Dropout Layers:** Placed after each dense layer to reduce overfitting [1].
- **Final Classification Layer:** Comprising 8 neurons with a **softmax** activation function to output class probabilities.

5 Training Pipeline

Figure 4 illustrates the training pipeline used in our machine learning process. The pipeline begins with a **Cleaned Dataset**, which is then split into two parts: 80% for training and 20% for testing. This split is performed while maintaining the class distribution.

The remaining 80% of the dataset is further divided into two subsets. 80% of this remaining data is used as the **Train Set**, and the other 20% is used as the **Validation Set**. The validation set is important for tuning model parameters and avoiding overfitting during training.

To augment the training data, we apply data augmentation techniques such as MixUp and CutMix at the `MixUp / CutMix` node. These techniques create new training samples by blending images, increasing the diversity of the training data. The augmented data is then added to the training set at

Model Name	**Train Accuracy**	**Test Accuracy**	**Train Loss**	**Validation Accuracy**	**Validation Loss**
Basic CNN	0.9659	0.9394	0.0981	0.9383	0.2024
EfficientNet	0.9003	0.9490	0.5084	0.9519	0.1624
ConvNext	**0.9715**	**0.9490**	**0.2173**	**0.9786**	**0.0953**

Table 1: Comparison of model performance metrics.

the `Add to Training Set` node. Additionally, other augmentation techniques like geometric transformations and color adjustments are applied to the training data in the `Apply Augmentation` node, further enhancing the model's robustness.

The training process begins with the `Train model` node, where the model is trained using the augmented training data. The training is further divide into two sub steps a coarse training and a fine tuning (While having a long coarse training is beneficial to the local tests we found little returns on the online test, so we progressively lowered the epochs and elevated the fine-tuning learning rate).

Coarse Training: During this phase, the ConvNeXt layers are frozen, and the focus is on training the newly added dense layers. Only **L2 regularization** is applied to the dense layer after the ImageNet `c_dense0`, with a kernel regularizer[4] of:

`tfk.regularizers.L2(1.3e-5)`

Fine-Tuning: In this phase, deeper layers of ConvNeXt are unfrozen, allowing the model to adapt its features to the domain-specific dataset. The regularization for `c_dense0` is enhanced to a combination of **L1L2 regularization**:
`tfk.regularizers.L1L2(l1=1.5e-4, l2=4e-4)`
The L1 component encourages sparsity, promoting feature selection, while the L2 component ensures weight stability. On output layer `c_output` we apply L2 regularization to stabilize predictions.

The addition of the regularization enhanced the trainability of the model enabling us to reach an improvement of roughly 2 percentage points During the training overfitting was controlled with the validation test and an Early Exit callback was provided; furthermore the leaning rate was controlled using a ReduceLRonPlateau callback. Finally, the model is evaluated using the test set.
For the main models that we tried to submit the local results are showed in table 1

6 Results

With The ConvNextSmall model and the training pipeline previously detailed we were able to achieve a **0.86** score on Codabench leader board. Unfortunately, due to our local hardware limitations and Google Colab free tier limitations we were unable to train bigger model with a moderate dataset. For instance, EfficientNet Large was achieving good results, but Colab was halting its training at around 25 epochs due to hardware resources limitations. Furthermore we were limited in our iteration development due to the impossibility to effectively tune our model with the Codabench test during the last days of the development phase.

Local result remained comparable with the accuracies of other method in the literature [10] [12]

7 Conclusions

While the model demonstrates good performance on reasonably distorted images, it struggles with strong magnitude distortions. This limitation could potentially be mitigated by employing a more diverse range of augmentations. Adopting an external library such as **Albumentations** [8] could provide access to advanced and varied augmentation techniques. Additionally, incorporating state-of-the-art augmentation methods, such as those designed to mitigate adversarial image modifications[11],, might further enhance the model's robustness. Another avenue for improvement could involve using a larger and more complex model to better handle these distortions.

Further work can also focus on hyperparameter tuning, particularly for the regularization layers. Optimizing these parameters could improve the model's generalization and performance. Moreover, additional testing could explore alternative activation functions for the classification head, as proposed in [2], and on the dense layer to assess their impact on final predictions.

By addressing these areas, the model's resilience to distortion and overall performance should improve sensibly

8 Contributons

We all worked in parallel on homework by employing a task-splitting strategy. In the first few days **Giorgio & Enrico** were focusing on data preprocessing and developing basic CNN's, while **Kamil**'s focus was on transfer learning, testing different types of pre-trained models such as Xception, MobileNet and Inception. From there we started having regular daily meetings to discuss the progress. We all had a contribution in data augmentation, **Giorgio** in particular was doing very well on that task. As we moved forward, **Kamil & Enrico** were exploring more pre-trained models and testing them on the augmented dataset. They then focused on fine-tuning these models to further improve performance. Subsequently, **Giorgio** was exploring modifications in the last network layers. Each one of us was running the models, locally and on Colab in order to exploit the limited resources we had.

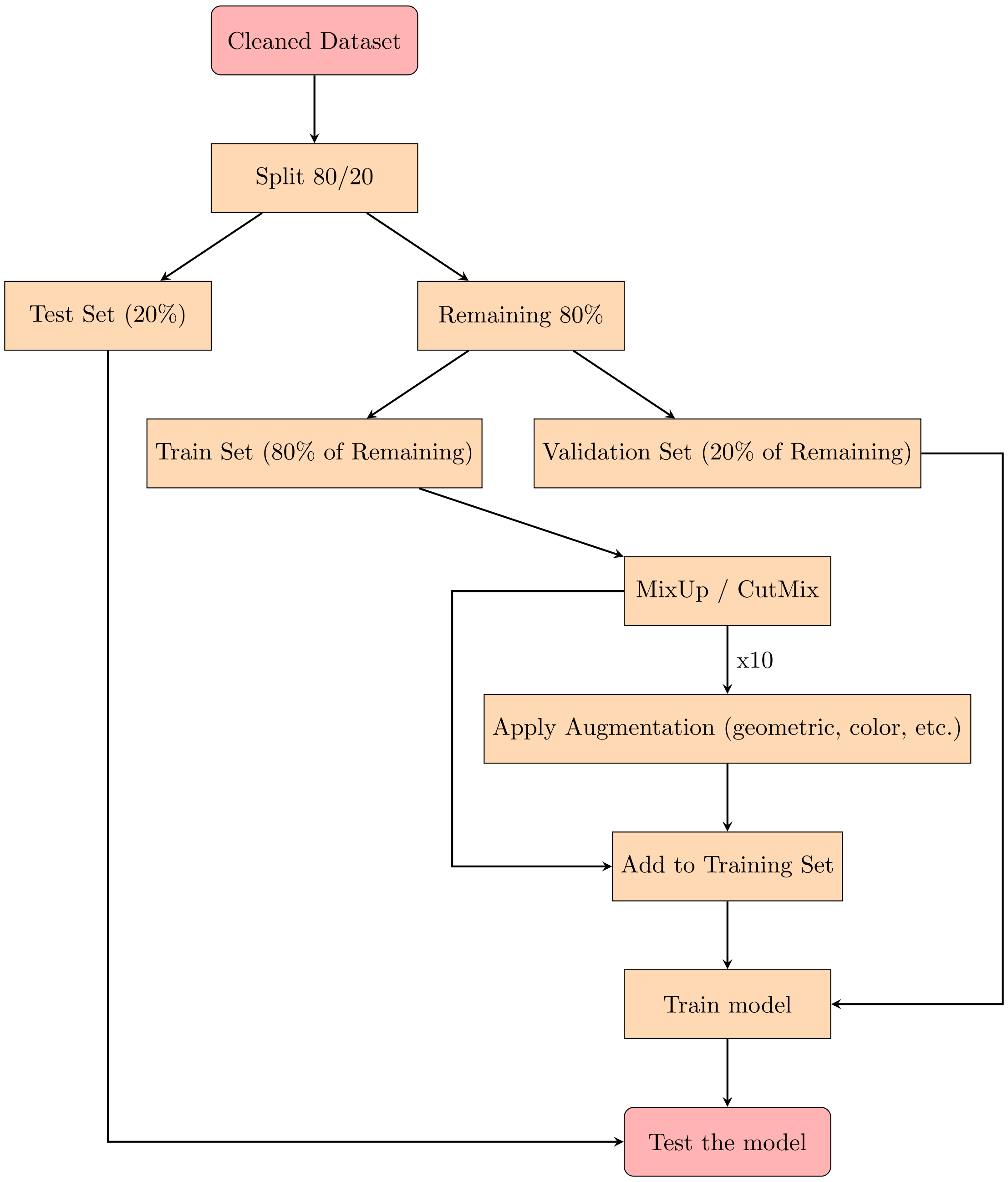

Figure 4: Training pipeline

References

[1] P. Baldi and P. J. Sadowski. Understanding dropout. *Advances in neural information processing systems*, 26, 2013.

[2] K. Banerjee, V. P. C, R. R. Gupta, K. Vyas, A. H, and B. Mishra. Exploring alternatives to softmax function, 2020.

[3] A. Buja, W. Stuetzle, and Y. Shen. Loss functions for binary class probability estimation and classification: Structure and applications. *Working draft, November*, 3:13, 2005.

[4] C. Cortes, M. Mohri, and A. Rostamizadeh. L2 regularization for learning kernels. *arXiv preprint arXiv:1205.2653*, 2012.

[5] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.

[6] N. Dong, Q. Feng, C. Jianfang, and X. Mai. White blood cell classification based on a novel ensemble convolutional neural network framework. *The Journal of Supercomputing*, 80:1–22, 06 2023.

[7] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*, 2019.

[8] G. Marques, D. Agarwal, and I. De la Torre Díez. Automated medical diagnosis of covid-19 through efficientnet convolutional neural network. *Applied soft computing*, 96:106691, 2020.

[9] A. D. Rasamoelina, F. Adjailia, and P. Sinčák. A review of activation function for artificial neural network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 281–286, 2020.

[10] 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*, 14(1):3570, 2024.

[11] R. Soklaski, M. Yee, and T. Tsiligkaridis. Fourier-based augmentations for improved robustness and uncertainty calibration. *arXiv preprint arXiv:2202.12412*, 2022.

[12] S. Tummala and A. K. Suresh. Few-shot learning using explainable siamese twin network for the automated classification of blood cells. *Medical & Biological Engineering & Computing*, 61(6):1549–1563, 2023.

[13] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features, 2019.

[14] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization, 2018.