AN2DL - First Homework Report
Spritzers

Federico Lombardo, Camilla Magnelli, Edoardo Margarini

federicozz, camillamagnelli, edomarg

252705, 226152, 252733

November 24, 2024

1 Introduction

This project aimed to classify blood cell images into eight classes using deep learning for multiclass classification. We adopted a stepwise approach, starting with simpler models, gradually progressing to more complex ones, evaluating performance at each step. Key actions included data preprocessing, model exploration, layer freezing and dropping to prevent overfitting, and performance evaluation with accuracy metrics. This approach resulted in a robust and accurate model.

2 Problem Analysis

2.1 Data inspection

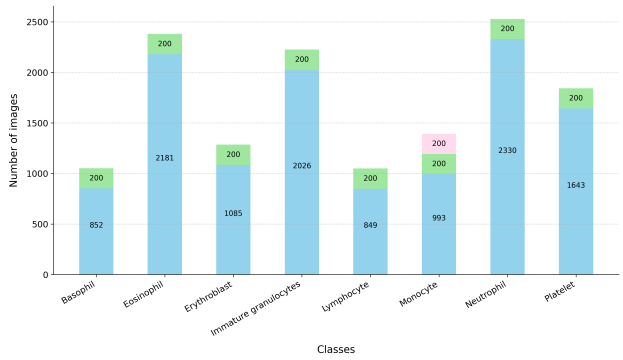

Figure 1: Data distribution across the classes.

The provided dataset consists of 13,759 images with dimensions 96×96 in RGB format. As illustrated in Figure 1, the dataset is not perfectly balanced and contains evident outliers. (Figure 2)

Figure 2: Outliers in the dataset.

After identifying and removing these obvious outliers, we decided to trust the integrity of the remaining dataset for our experiments.

2.2 Main challenges

The main challenges we encountered during the project included:

- **Avoiding overfitting:** Ensuring that the model generalizes well and does not memorize the training data.
- **Achieving competitive performance on Codabench:** This platform used a hidden dataset for evaluation, making generalization critical.
- **Tuning hyperparameters:** Experimenting with various hyperparameter configurations to improve model performance.

- **Efficient training:** Optimizing the training process to achieve desirable results within reasonable computational resources and time constraints.

3 Experiments & Results

We followed a systematic progression of models, improving accuracy and addressing practical limitations at each stage:

1. **Simple CNN:** Initially, we implemented a simple CNN with 4 convolutional layers, hypothesizing that it would suffice for the classification task. While it outperformed a linear classifier locally, it achieved poor results on *Codabench*, with an accuracy of **0.25**.

2. **Modified VGG:** Transitioning to a modified VGG architecture, we halved the number of filters in convolutional layers and reduced the neurons in fully connected layers. This improved accuracy to **0.45**, but the model's size (nearly 1GB) made further experimentation impractical.

3. **Transfer Learning with MobileNetV3Small:** Using the pre-trained **MobileNetV3Small** on ImageNet, we achieved **0.45** accuracy, showing transfer learning's potential for improvement.

4. **EfficientNetB4 with Fine-Tuning:** Experimentation with pre-trained models included **ResNet101** and **DenseNet169**, which showed moderate performance but did not meet our accuracy expectations. This led us to **EfficientNetB4**, where we froze the first 124 layers and applied basic augmentation techniques (rotations and translations), resulting in an improved accuracy of **0.65**.

5. **Advanced Augmentation:** We utilized **KerasCV** for advanced augmentation techniques, which contributed to improving performance. The combination of **RandAugment** and **RandMix** increased the accuracy to **0.86**, though training stability remained a challenge. We also explored an oversampling approach by balancing all class samples to the maximum class size, achieving the same accuracy.

6. **Optimizer and Learning Rate Adjustment:** During fine-tuning, we observed instability due to a high learning rate, leading to significant fluctuations in training accuracy. To address this, we reduced the learning rate to 10^{-5}, stabilizing the weight updates. Simultaneously, we transitioned from the Adam optimizer to the more robust **Lion** optimizer, which further enhanced stability and convergence. This combination proved effective, contributing to the final accuracy of **0.92**.

Table 1: Performance comparison of different models. Best results are highlighted in **bold**

Model	Accuracy	Precision	Recall	ROC AUC
Custom VGG	95.11	95.44	95.11	91.57
MobileNetV3Small	96.61	96.61	96.61	99.85
EfficientNetB4 (AugMix e RandAug)	96.82	96.90	96.82	99.87
EfficientNetB4	**99.33**	**99.33**	**99.33**	**99.99**

4 Method

This section details the approach used for the image classification task, which resulted in the best-performing model after experimentation with different architectures and training strategies. We adopted a *Transfer Learning* pipeline using **EfficientNetB4** as the pre-trained backbone due to its proven efficiency and accuracy on similar tasks. The methodology includes preprocessing, advanced data augmentation, feature extraction, and fine-tuning to achieve robust and efficient performance.

4.1 Model Architecture

EfficientNetB4 is a convolutional neural network architecture that employs compound scaling, optimizing depth, width, and input resolution for improved accuracy with fewer parameters and computations. [3] Key features include:

- Batch Normalization (BN): Enhances training stability and convergence by normalizing activations, especially when combined with the Swish activation function.

- Swish Activation: A differentiable function

$$\text{Swish}(x) = x \cdot \text{sigmoid}(x)$$

  allowing for efficient handling of negative values and improved non-linear modeling.

- Depthwise Separable Convolutions: Reduce computational cost by separating spatial and channel-wise operations.

- Squeeze-and-Excitation (SE) Blocks: Introduce channel-wise attention to emphasize relevant features dynamically.
- Global Average Pooling (GAP): Minimizes overfitting by aggregating spatial information before the final dense layer.

The final model architecture follows a modular design:

1. **Input Layer:** Images were resized to $(380 \times 380 \times 3)$, ensuring compatibility with *EfficientNetB4*.
2. **Preprocessing and Augmentation:** Data augmentation was applied using `KerasCV`, including `AugMix`, blended augmentations with severity 0.2, and `RandAugment`, with two transformations per image and magnitude 0.3. Inputs were normalized using `preprocess_input` for compatibility with pre-trained weights.
3. **Feature Extraction:** The *EfficientNetB4* backbone, pre-trained on ImageNet, employed global average pooling (`pooling='avg'`). Initially, all layers were frozen (`trainable=False`) to use it as a feature extractor.
4. **Regularization and Output:** A *Dropout* layer (rate 0.3) was added before a dense layer with `softmax` activation [1] to predict 8 target classes:
$$\sigma(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{N} \exp(x_j)} \quad (1)$$
where: x_i is the i-th element of the input vector x, N is the total number of elements in the vector x, $\exp(x)$ is the exponential function applied element-wise.

4.2 Dataset Preparation

The dataset pipeline was implemented using `TensorFlow Datasets`:

- **Data Splitting:** The challenge's structure enabled evaluation on the platform's test set, allowing more data for training and reserving the validation set for tuning and optimization.
- **Data Augmentation and Preprocessing:** Samples were dynamically augmented and preprocessed, including resizing and normalization, to ensure compatibility with *EfficientNetB4* pre-trained weights.

4.3 Fine-Tuning Strategy

After the transfer learning phase, fine-tuning was applied to further refine the model. The deeper layers of *EfficientNetB4*, particularly the `Conv2D` and `DepthwiseConv2D` layers, were selectively unfrozen, while the first 100 layers remained frozen to maintain low-level feature extraction. The *Lion* optimizer was used for its stability and fast convergence, with a learning rate of 10^{-5}.

4.4 Loss Function and Evaluation

The model was optimized using the *Categorical Crossentropy* loss function [2]:

$$\mathcal{L} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \quad (2)$$

where y_i is the ground truth label, $\hat{y}_i$ is the predicted probability and N is the number of classes. Accuracy was used as the primary evaluation metric to monitor performance.

5 Discussion

This project showed both strengths and limitations. On the positive side, using EfficientNetB4 as a pre-trained backbone significantly improved generalization, achieving competitive accuracy on the hidden *Codabench* dataset. The model also provided an effective trade-off between accuracy and computational cost. Furthermore, advanced augmentation techniques, such as `RandAugment` and `RandomMix`, enhanced robustness by reducing overfitting and improving resilience to input variability.
However, fine-tuning required careful adjustments to learning rate and optimizer parameters to address training instability. Additionally, performance was highly sensitive to augmentation hyperparameters, necessitating extensive experimentation to identify optimal configurations.

6 Conclusions

The project achieved promising results, but performance remained highly dependent on the quality of preprocessing and augmentation pipelines.
Throughout the process, each team member shared useful insights and ideas, helping to make important improvements and uncovering key breakthroughs that shaped the final approach.

6.1 Future Directions

To further enhance the model, future efforts could focus on alternative architectures, investigating state-of-the-art models, such as Vision Transformers (ViT) or hybrid architectures combining convolutional layers, or on exploring data-driven approaches, like AutoAugment, to minimize sensitivity to manual hyperparameter tuning.

References

[1] K. A. Documentation. Activations. https://keras.io/api/layers/activations/.

[2] GeeksforGeeks. Categorical cross-entropy in multi-class classification. https://www.geeksforgeeks.org/categorical-cross-entropy-in-multi-class-classifi 2024.

[3] TensorFlow. Efficientnet model. https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet, 2024.