AN2DL - First Homework Report
Team: Machine Dumbening

Seyedehyekta Kamaneh, Prachi Kedar, Francesco Frontera

yektakamaneh, prachikedar27, frankman110

243997, 245581, 247174

November 24, 2024

1 Introduction

This report, prepared for the Artificial Neural Networks and Deep Learning course, focuses on solving an image classification problem using Convolutional Neural Networks (CNNs). The dataset contains 13,759 images, each 96x96 pixels with three RGB color channels, curated for classifying blood cell types. It includes eight labeled classes representing different blood cells, with each label corresponding to a specific type. Examples for each class are provided in Figure 1.

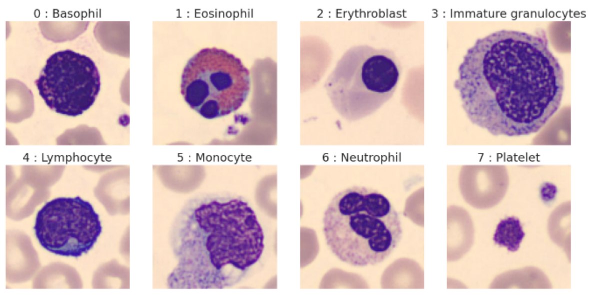


Figure 1: Example image for each class

2 Data Inspection

During dataset inspection, we identified two recurring image types unrelated to blood cell classes, acting as contaminants across most class labels. To automate their identification and separation from the rest of the dataset, we used a pre-trained VGG16 model as a feature extractor. Contaminants were manually selected at indices 13202 and 13690, and the images were resized to 224×224 pixels before passing through VGG16 to extract feature vectors from the 'fc1' layer. Cosine similarity was calculated between the feature vectors of all images and the contaminants, with matches having similarity scores above 80% flagged as contaminants.

This approach identified 200 instances of the first contaminant and 1,600 of the second. Contaminants were isolated into a separate subset with labels preserved, and a cleaned dataset was created by excluding them. Figure 2 shows the two types of mentioned contaminants.

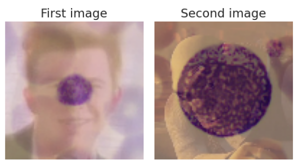
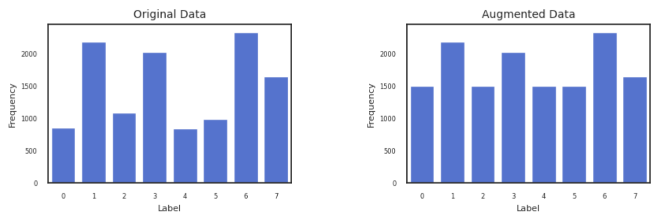


Figure 2: Data contamination samples

3 Data Balancing

After removing contaminant images, the cleaned dataset contained 11,959 samples across 8 classes. Analysis of the data distribution revealed significant imbalance, with classes 0, 2, 4, and 5 heavily underrepresented. To address this, data augmentation was applied to generate synthetic images for these classes, increasing their sample counts to a target of 1,500 samples each.

The augmentation process applies transformations such as random rotations, flips, scaling, brightness and contrast adjustments, and Gaussian noise. These ensured the synthetic images retained the characteristics of their original classes while introducing variability. Synthetic images were generated in batches for each underrepresented class until the target count was reached. The augmented data was then combined with the original dataset, resulting in a balanced dataset with improved representation for all classes.



(a) Original Dataset (b) Balanced Dataset

Figure 3: Dataset Distributions

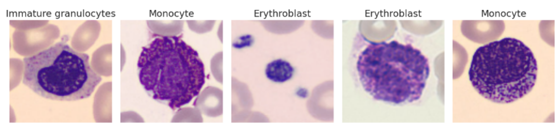


Figure 4: Synthetic data samples

4 Custom Model Design

The custom CNN model is inspired by the U-Net architecture, commonly used in medical image processing. Our U-Net model, with eight convolutional blocks, consists of a contracting path with convolutional layers (64 to 512 filters) and max-pooling for feature extraction, followed by an expanding path with transposed convolution layers for upsampling and skip connections to preserve image details. At the center, a bottleneck layer with 1024 filters captures the most abstract representation of the input. The final classification uses global average pooling and a dense softmax layer for output probabilities across eight blood cell classes. Figures 5 and 6 show the training vs validation accuracy and loss performed with cross-entropy.

We report the following benchmarks on the test-set using the model trained in this phase:

- Accuracy score over the test set: 97.83%
- Precision score over the test set: 97.86%
- Recall score over the test set: 97.83%
- F1 score over the test set: 97.83%

Even though this model performs good on the test set, on the Codabench platform didn't score above 30%, therefore we suspect over-fitting.

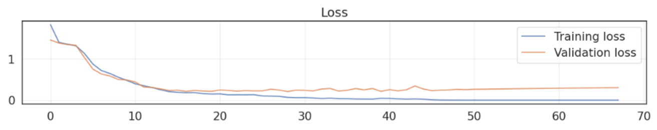


Figure 5: Training vs. Validation Loss

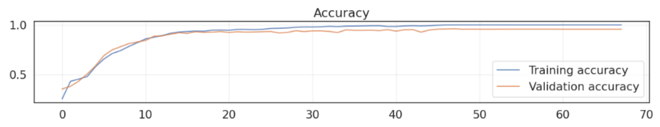


Figure 6: Training vs. Validation Accuracy

5 Using Pre-trained Models

For the transfer learning phase, we employed several pre-trained models, including InceptionV3, ResNet50, VGG16, VGG19, MobileNet, and EfficientNetB0. During this phase, we froze all layers of the selected pre-trained model and added two fully connected (dense) layers for classification. Once the model was trained, fine-tuning was performed by unfreezing a portion of the final layers and training them while keeping the other layers frozen. To enhance training, we reduced the learning rate from 1×10^{-4} to 1×10^{-5} and decreased the batch size from 128 to 64.

The number of neurons in the added dense layers was adjusted based on the architecture of each model. We aimed to ensure that the number of neurons in the first dense layer was at least as large as the number of neurons in the final pooling layer of the pre-trained model to preserve important information for classification. To prevent overfitting, a dropout rate of 0.3 was applied to each dense layer.

In the fine-tuning phase, the number of layers unfrozen depended on the size and complexity of the model. Larger models had more layers unfrozen, while smaller models had fewer. The results for each model are summarized in Table 1.

Table 1: Comparison of pre-trained models

Model name	Transfer Learning			Fine-tuning			Unfrozen layers
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	
MobileNet	95.32	95.43	95.32	97.07	97.07	97.07	40
VGG16	76.92	80.21	76.92	95.32	95.63	95.32	10
VGG19	75.42	77.63	75.42	93.97	93.64	93.64	15
InceptionNetv3	83.72	84.96	83.72	95.14	95.48	95.14	40
ResNet50	28.03	26.85	27.24	50.42	49.98	50.22	75
EfficientNetb0	26.71	25.82	25.24	48.33	48.67	48.33	75

6 Selected Model

As shown in the table, the model with the highest benchmarks is the one trained using MobileNet. Below, we present the plots of training versus validation for both loss and accuracy metrics. Additionally, the confusion matrix for the test set is provided in Figure 9.

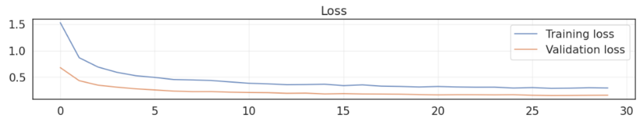


Figure 7: Training vs. Validation Loss

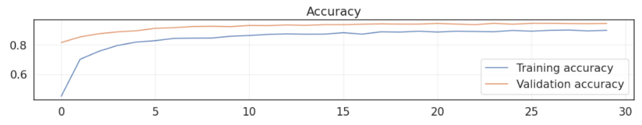


Figure 8: Training vs. Validation Accuracy

7 Hyperparameter Fine-Tuning

To optimize performance of the selected model, we applied hyperparameter tuning using random search, grid search, and Bayesian optimization techniques to explore different configurations. The results are listed here:

- Learning rate: 0.0001
- Number of layers: 2
- Number of neurons per layer: 832, 320
- Batch normalization of each layer: yes, yes
- Dropout rate of each layer: 0.3, 0.1

8 Discussion

Through this project, we arrived at two major conclusions:

1. For a classification problem of this nature, fine-tuning a pretrained model significantly outperforms training a custom model from scratch, even when the custom model has a sophisticated and intricate architecture.

2. When selecting a pretrained model to fine-tune, we experimented with various options, including models with highly complex architectures (e.g., ResNet and EfficientNet) and others with relatively simpler architectures (e.g., MobileNet, VGG, and InceptionNet). Interestingly, we observed that models with simpler architectures and fewer layers outperformed the more complex ones.

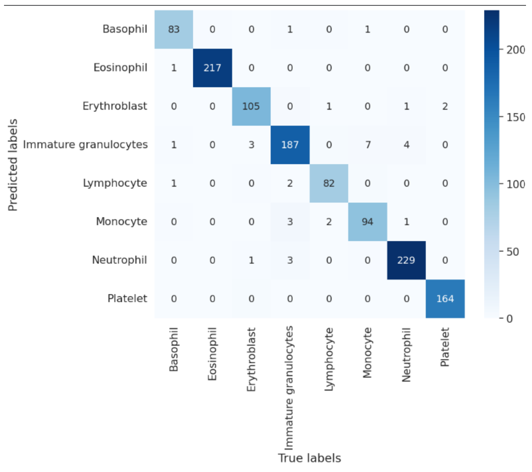


Figure 9: Confusion matrix of the selected model

This project reflects a collaborative effort, combining the contributions of all team members. Using a shared IPython notebook, we worked together through a mix of in-person and online sessions, exchanging ideas and integrating our work to achieve the final result.