Convolutional Neural Network for Acute Lymphoblastic Leukemia Classification

Instituto Tecnológico y de Estudios Superiores de Monterrey

Author: *Gamaliel Marines*, gamaliel_projects@hotmail.com

Abstract

Implementation of a *Convolutional Neural Network* (CNN) to detect different stages of *Acute Lymphoblastic Leukemia (ALL)* and *classify* them correspondingly. The data set consists of *3256* PBS *images* from 89 suspected of ALL patients. This dataset is divided into four classes: *benign, early, pre,* and *pro*. The result of the implementation is a *CNN model* using *transfer learning ResNet50*, reaching an *accuracy* of 97% in test cases, 99% in validation cases, and 98% in training cases.

Impact

Leukemia is one of the most lethal forms of cancer, particularly affecting children and young adults [1]. The implementation of a Convolutional Neural Network (CNN) for the detection and classification of Acute Lymphoblastic Leukemia (ALL) stages has the potential to significantly enhance diagnostic accuracy and speed. Early and accurate detection is crucial in improving patient outcomes, as timely intervention can lead to better treatment responses and increased survival rates.

Improved Diagnostic Accuracy: The CNN model demonstrated great accuracy in test cases and validation cases. By automating the classification process, healthcare providers can reduce misdiagnosis and ensure that patients receive appropriate treatment more swiftly.

Reduction in Diagnostic Time: The use of CNNs can drastically reduce the time required for diagnosis. Traditional methods may take longer, while a trained CNN can analyze and classify images in a matter of seconds [2]. This rapid processing capability is particularly beneficial in emergency situations where prompt diagnosis is essential for effective treatment.

Scalability and Accessibility: The deployment of CNNs in clinical settings can enhance the scalability of leukemia detection. With an increasing number of patients requiring evaluation, automated systems can handle larger volumes of data without compromising accuracy. This can be particularly advantageous in regions with limited access to specialized medical professionals, making advanced diagnostic tools more accessible to underserved populations.

Introduction

Acute Lymphoblastic Leukemia (ALL) is a rapidly progressing cancer of the blood and bone marrow that predominantly affects children but can also occur in adults [1]. Early and accurate diagnosis of ALL is critical, as timely intervention significantly improves survival rates and patient outcomes. Traditional diagnostic methods rely heavily on manual examination of Peripheral Blood Smear (PBS) images by hematologists [3], a process that is both time-consuming and prone to human error due to the visual similarities between leukemic and normal cells.

Advances in deep learning, particularly Convolutional Neural Networks (CNNs), have revolutionized image-based diagnostics by enabling automatic feature extraction and classification with high accuracy [3]. CNNs

are especially well-suited for medical imaging tasks, as they can identify subtle patterns and anomalies that may be overlooked by the human eye.

This study presents the implementation of a CNN model using transfer learning with ResNet50 to detect and classify different stages of ALL. The model is trained on a dataset comprising 3,256 PBS images from 89 patients suspected of having ALL, categorized into four classes: benign, early, pre, and pro.

By using deep learning, the proposed system aims to support medical professionals in diagnosing ALL more accurately and efficiently. The CNN model achieved 97% accuracy on the test set, 99% on validation set, and 98% on training set, demonstrating its potential as a reliable diagnostic tool in clinical practice.

Related Work and Research

The introduction of Convolutional Neural Networks (CNNs) marked a turning point in this field. CNNs are capable of learning hierarchical features directly from raw image data, thereby eliminating the need for manual feature engineering. Several studies have successfully applied CNN architectures to classify PBS images, demonstrating superior performance over traditional methods. For instance, Ghaderzadeh et al. proposed a fast and efficient CNN model for the diagnosis of B-ALL and its subtypes using PBS images, achieving high accuracy and significantly reducing processing time [6]. Their work also laid the foundation for the publicly available dataset used in this study.

Other notable contributions include works that employed transfer learning techniques using pre-trained models such as VGG16, InceptionV3, and ResNet50. These models, originally trained on large-scale datasets like ImageNet, can be fine-tuned on medical imaging data to leverage their robust feature extraction capabilities while requiring fewer training samples [7], [8]. Transfer learning has proven particularly effective in domains like medical diagnostics where acquiring large annotated datasets is often challenging.

Additionally, researchers have explored segmentation-based preprocessing to improve classification results. Techniques such as color thresholding in HSV color space, morphological operations, and contour extraction help isolate leukocytes from the background, thereby reducing noise and improving model focus. Some studies combine segmentation with attention mechanisms or ensemble learning to boost classification accuracy [9].

Despite these advancements, challenges remain in achieving consistent performance across diverse datasets and clinical settings. Variations in microscope type, staining techniques, and image acquisition parameters can affect model generalizability. Therefore, continuous validation with real-world clinical data and interpretability of model predictions remain crucial areas of ongoing research.

The current study builds upon this body of work by implementing a CNN model based on ResNet50 using transfer learning to classify benign and malignant PBS images, as well as the subtypes of B-ALL. By leveraging a high-quality, well-labeled dataset and focusing on the early identification of leukemic subtypes, this research contributes to the development of reliable and accessible AI-assisted diagnostic tools.

Understanding the Dataset

In the context of machine learning and deep learning, a dataset refers to a structured collection of data that is used to train, validate, and test models. In image classification tasks such as medical diagnostics, a dataset typically comprises labeled images that represent various classes or conditions to be recognized

by the model. The quality, diversity, and representativeness of the dataset are crucial for building a model that performs reliably in real-world scenarios.

A high-quality dataset is characterized by the following attributes:

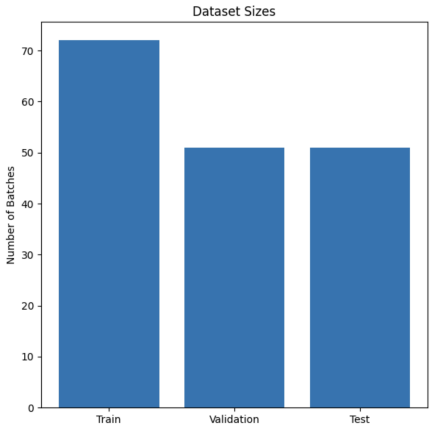
1. Clear labeling of samples by qualified experts.
2. Balanced representation of all target classes to prevent bias.
3. Sufficient sample size to allow the model to learn meaningful patterns.
4. High-resolution images that preserve important visual features.
5. Consistency in imaging conditions, such as lighting and magnification, to reduce noise.

For this study, I use the publicly available Acute Lymphoblastic Leukemia (ALL) image dataset prepared at the bone marrow laboratory of *Taleghani Hospital* in Tehran, Iran. This dataset includes 3,256 peripheral blood smear (PBS) images collected from 89 individuals, including 25 healthy (benign) cases and 64 patients diagnosed with ALL. The ALL class is further categorized into three malignant subtypes: *Early Pre-B*, *Pre-B*, and *Pro-B* ALL. All images were captured using a Zeiss microscope camera at 100x magnification and saved in JPG format to ensure visual quality.

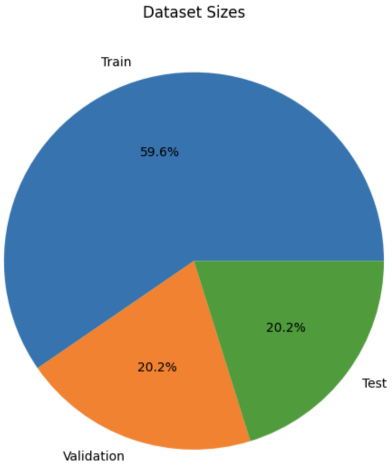
The labeling of cell types and subtypes was done by a specialist using flow cytometry, a gold standard in hematological diagnostics, which adds to the reliability of the dataset [4]. Furthermore, the dataset includes both original and segmented images, with segmentation performed using color thresholding in the HSV color space, providing additional data for model training or augmentation.

Given its professional preparation, expert labeling, and public availability, this dataset is well-suited for developing and evaluating a Convolutional Neural Network (CNN) model for the detection and classification of ALL and its subtypes.

To better understand the dataset, its distribution, and the subsets I use plotting and graphing tools to display categories and their proportions.

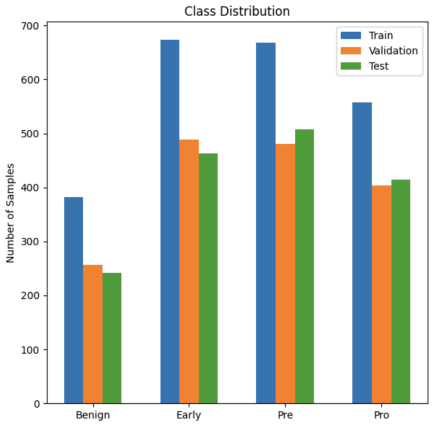


The first graph illustrates the division of the dataset into three subsets: training, validation, and test. Each subset serves a distinct purpose. The training set, being the largest, is used to train the model and helps it to learn patterns. The validation set is used during training to monitor the model's performance and adjust hyperparameters, helping to prevent overfitting. Finally, the test set is used after training is complete to evaluate the model's performance on unseen data.

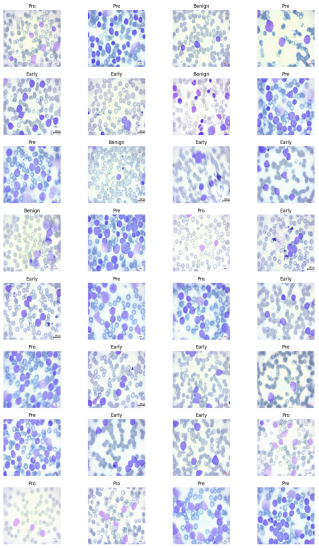


The pie chart displays the proportion of each subset within the dataset. Achieving an

appropriate balance among the training, validation, and test subsets is crucial to maximizing the effectiveness of the dataset. A sufficiently large training set is essential for building a robust and well-fitted model. However, if the validation and test subsets are too small, they may not provide reliable insights into the model's performance. Conversely, allocating too much data to validation and testing can compromise the training process by limiting the data available for learning. Therefore, careful distribution is necessary to ensure both reliable evaluation and effective training.



This graph is particularly important, as it provides insight into the distribution of classes across the training, validation, and test subsets. A balanced class distribution is essential to prevent model bias, which could lead to unreliable or skewed classifications. In cases of significant imbalance, the model may overfit to the majority class and underperform on minority classes. However, in this scenario, the graph indicates a well-balanced distribution across all subsets, supporting a fair and reliable training process. As such, the model implementation can proceed with confidence in the dataset's representativeness.



One last step before starting with the implementation of the model is to verify the quality and the labeling of the dataset. A quick validation of the content of the dataset will help understand how the model will work and what to expect as the result.

Dataset Source: *Mehrad Aria et al., "Acute Lymphoblastic Leukemia (ALL) image dataset." Kaggle (2021). DOI: 10.34740/KAGGLE/DSV/2175623*

Publication Reference: *Ghader Zadeh, M. et al. (2022). A fast and efficient CNN model for B-ALL diagnosis and its subtypes classification using peripheral blood smear images. Int J Intell Syst. 37: 5113–5133. doi:10.1002/int.22753*

Implementation of the solution

To implement the classification model, initial work was required on the dataset through a process known as ETL—Extract, Transform, and Load. In this context, extraction involved identifying and selecting a reliable dataset, validating its quality, and storing it locally. Once acquired, the data underwent transformation, which included splitting it into training, validation, and test subsets, as well as balancing the classes to ensure fair model training. Instead of loading the data into a data warehouse or data lake, the final load phase consisted of feeding the transformed data directly into the model for training and evaluation.

In the data preprocessing phase, two essential Python libraries—Pandas and NumPy—played a crucial role in managing and analyzing the image dataset before feeding it into the deep learning model.

Pandas was used to create and display a tabular representation of the dataset structure. Specifically, *pandas.DataFrame()* was applied to generate a quick view of the image filenames contained within the dataset directory. This enabled initial verification and exploration of the data files to ensure correctness and completeness before training.

NumPy was instrumental in performing numerical operations, particularly in computing the distribution of classes across different dataset splits – training, validation, and test. The flexibility and efficiency of NumPy's array-based operations made it an ideal choice for this type of computation, allowing for quick accumulation of label frequencies.

The dataset itself was organized into three subsets using the *image_dataset_from_directory* function from *TensorFlow's* preprocessing module. This function automatically labels images based on the directory structure and resizes them to a uniform dimension of 224 × 224 pixels to match the input size required by the model. The data was split using stratified validation and separate random seeds were used to generate reproducible and distinct training, validation, and testing sets.

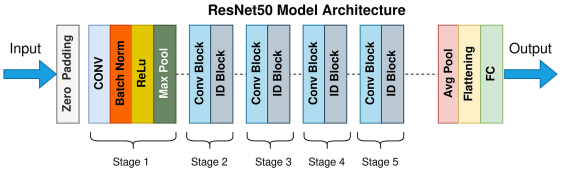
To better understand the data distribution and identify any potential class imbalance, visualizations were generated using *Matplotlib*:

A bar plot comparing the number of batches in the training, validation, and test sets.

A pie chart showing the relative dataset sizes.

A grouped bar chart illustrating the distribution of class labels across each subset.

To implement the classification model, a pre-trained *ResNet50* architecture was employed as the base model. The input shape was configured to *224 × 224 × 3* – consistent with the dimensions of the preprocessed images. To mitigate the risk of overfitting and to retain the robust feature extraction capabilities learned from a larger, diverse dataset, the weights of the base model were frozen during training. This means that the convolutional layers, which have already learned to detect low- to high-level visual features such as edges, textures, and patterns, were not updated during backpropagation. Freezing these layers allows the model to benefit from this prior knowledge and focus learning solely on the newly added dense layers tailored for the specific classification task.



A custom *classification head* was added on top of the *ResNet50* base to adapt the model for the specific task. This head included a *flattening layer* to convert the feature maps into a one-dimensional vector, followed by a *dropout layer* with a rate of 0.5 to mitigate overfitting by randomly deactivating neurons during training. The final layer was a *dense output layer* with four units and a *softmax activation function*, suitable for the multi-class classification. The model was compiled using the *Adam optimizer* along with *categorical cross-entropy* as the loss function and accuracy as the performance metric.

As part of the model architecture, a *Flattening layer* was included immediately after the base ResNet50 model. The purpose of this layer is to convert the multidimensional output of the convolutional base into a one-dimensional

vector – a required format for the fully connected (dense) layers that follow.

Convolutional Neural Networks (CNNs), such as ResNet50, typically produce feature maps with three dimensions: height, width, and number of channels. For example, the output of the convolutional base might be a tensor of shape *(7,7,2048)*. The Flatten layer transforms this into a vector of shape *(7×7×2048)* or *(100352)*, preserving the spatial structure in a linearized format.

Mathematically, if the output from the convolutional layer is a 3D tensor:

$$T \in \mathbb{R}^{h \times w \times c}$$

the Flatten operation reshapes it into a 1D tensor:

$$\mathbf{x} \in \mathbb{R}^{h \cdot w \cdot c}$$

This transformation is essential for connecting the output of convolutional operations with dense layers, which require 1D input vectors to perform classification.

The Flatten layer itself does not contain trainable parameters; instead, it serves as a bridge between the feature extraction phase of the CNN and the decision-making phase handled by the dense layers.

In the final stage of the neural network architecture, a *Dense layer* – also known as a fully connected layer – was added to perform classification. This layer takes the flattened feature vector produced by the previous layer and maps it to the output classes using learned weights and biases.

Mathematically, a dense layer performs a linear transformation followed by an activation function. If the input vector is $\mathbf{x} \in \mathbb{R}^n$, the dense layer computes:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\hat{\mathbf{y}} = f(\mathbf{z})$$

where:

- $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight matrix,
- $\mathbf{b} \in \mathbb{R}^m$ is the bias vector,
- f is the activation function (in this case, **softmax**), and
- $\widehat{y}$ is the predicted probability distribution over the output classes.

In this implementation, the dense layer has *4 output units*, corresponding to the four target classes in the classification problem. The *softmax activation function* is applied to ensure that the output values represent probabilities, summing up to 1. This allows the model to output a confidence score for each class, making it suitable for multiclass classification tasks.

The dense layer is trainable, meaning its weights and biases are updated during backpropagation to minimize the loss function and improve prediction accuracy over time.

To optimize the performance of the convolutional neural network, the Adam – Adaptive Moment Estimation – optimizer was employed. It combines the advantages of two other methods: *momentum*, which accelerates convergence by accumulating the exponentially decaying average of past gradients, and *RMSProp*, which adapts the learning rate based on the moving average of squared gradients.

Mathematically, at each training step t, Adam computes the first moment estimate (mean of gradients) as:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

and the second moment estimate (uncentered variance of gradients) as:

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

where gtg_tgt is the gradient at step t, and β1 and β2 are the decay rates, typically set to 0.9 and 0.999, respectively. To address initialization bias, the estimates are corrected as:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

The parameters are then updated using the following rule:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

This is the rule for updating the model parameters (θ). It takes a step in the direction of the corrected first moment estimate ($\widehat{m_t}$), scaled by the learning rate (α) and normalized by the square root of the corrected second moment estimate ($\sqrt{\widehat{v_t}}$) plus a small constant (ϵ) to prevent division by zero. This normalization effectively adapts the learning rate for each parameter.

To evaluate the performance of the classification model, the *Categorical Cross-Entropy* loss function was utilized. This loss function is commonly employed in multi-class classification problems, where the model outputs a probability distribution across multiple classes.

Categorical Cross-Entropy measures the dissimilarity between the predicted probability distribution and the true distribution (represented as a one-hot encoded vector). For a single sample, the loss is defined as:

$$L = -\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i)$$

where:

- C is the number of classes,
- y_i is the true label (1 if the class is correct, 0 otherwise),
- $\widehat{y_i}$ is the predicted probability for class i.

In this context, the true label vector y is one-hot encoded, meaning that only the correct class has a value of 1, and all others are 0. The loss function then penalizes the model more severely as the predicted probability for the correct class deviates from 1. Lower loss values indicate that the predicted probability distribution is closer to the true distribution.

The use of categorical cross-entropy in this model ensures that the training process prioritizes correct classification by minimizing the difference between predicted and actual class probabilities. This approach is especially effective when combined with a softmax activation function in the output layer, which ensures that the output values represent a valid probability distribution summing to one.

The *Softmax activation function* is used in the output layer of neural networks designed for multi-class classification problems. Its main purpose is to convert the raw output scores – known as logits – into normalized probability values that sum to one. This allows the model to express the likelihood of an input belonging to each class.

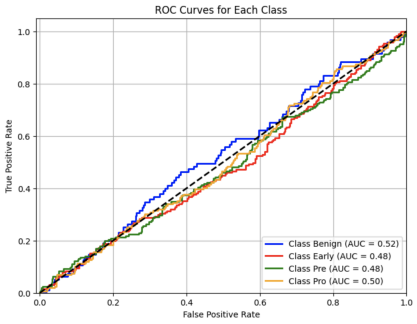
Mathematically, for a given output vector Z containing K logits, the Softmax function computes the probability of class i as follows:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Each component e^{z_i} is the exponential of the logit for class i, and the denominator is the sum of exponentials of all logits, ensuring that the output values form a valid probability distribution.

By emphasizing the largest values in the input vector while suppressing the smaller ones, Softmax enables the model to make a clear prediction by assigning higher probability to the most likely class.

I encounter the problem of class weights and skewness of the model. To address these issues I applied ROC curves to find the optimal weights for each class and improve the result of the model.



To optimize training performance, two callbacks were incorporated: ModelCheckpoint, which saved the best model during training, and EarlyStopping, which halted training early if the validation loss did not improve. The model was trained for three epochs using a defined training and validation dataset, and its performance was monitored throughout the training process.

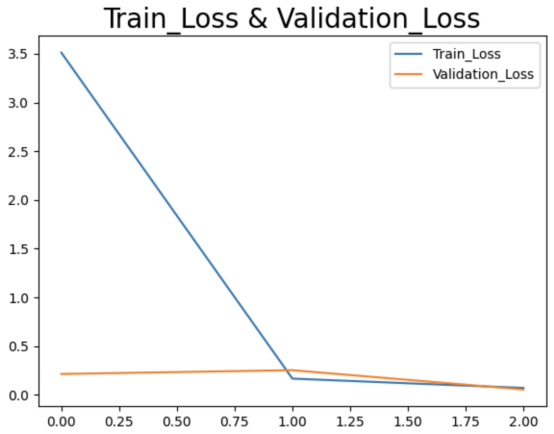
Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
flatten (Flatten)	(None, 100352)	0
dropout (Dropout)	(None, 100352)	0
dense (Dense)	(None, 4)	401,412

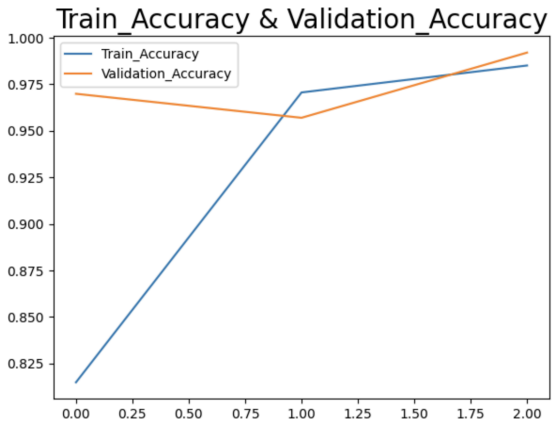
```
Epoch 1/3
72/72 ━━━━━━━━━━━━━━━━━━━━ 274s 4s/step - accuracy: 0.6686 - loss: 3.8103 - val_accuracy: 0.9343 - val_loss: 0.4046
Epoch 2/3
72/72 ━━━━━━━━━━━━━━━━━━━━ 239s 3s/step - accuracy: 0.9718 - loss: 0.2245 - val_accuracy: 0.9859 - val_loss: 0.0896
Epoch 3/3
72/72 ━━━━━━━━━━━━━━━━━━━━ 230s 3s/step - accuracy: 0.9911 - loss: 0.0534 - val_accuracy: 0.9785 - val_loss: 0.1388
```

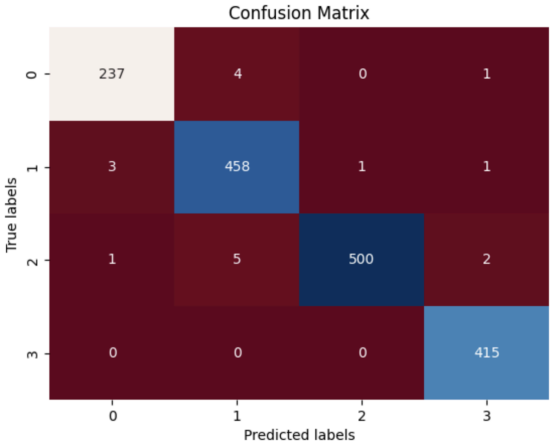
The code of the implementation of the solution can be found in GitHub under the following link: Gamaliel-Marines/CNN-ALL-Classification

Results

The first version of the model was overfitting, and was memorizing the features of the dataset. Great accuracy was reached but when tested, the predictions were incorrect.



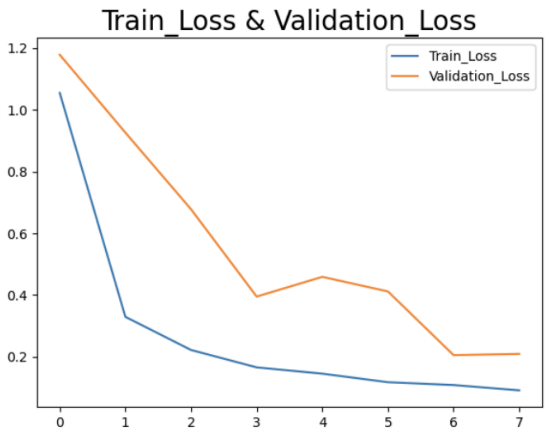


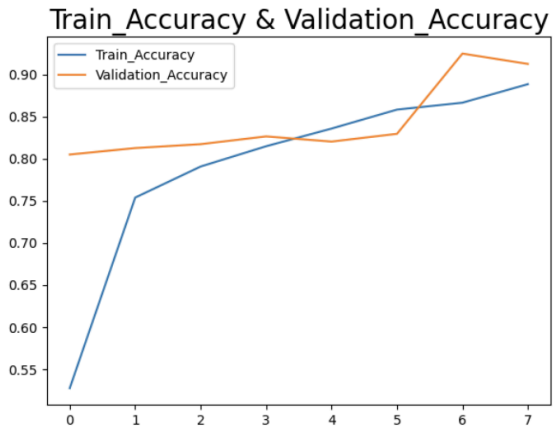


Classification Report is :

	precision	recall	f1-score	support
0	0.97	0.99	0.98	95
1	0.99	0.99	0.99	191
2	1.00	0.99	0.99	206
3	1.00	0.99	1.00	159
accuracy			0.99	651
macro avg	0.99	0.99	0.99	651
weighted avg	0.99	0.99	0.99	651

After doing modifications, the model actually started learning, and the single predictions increased in correctness.

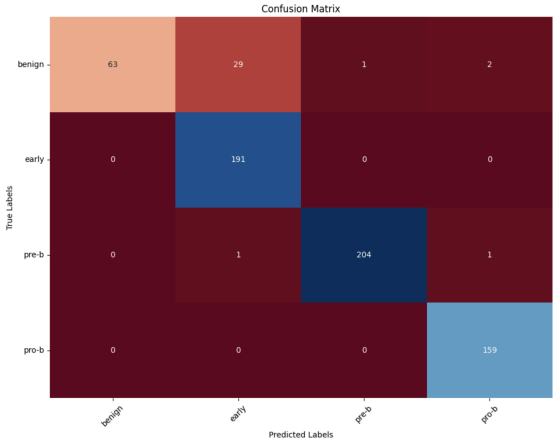


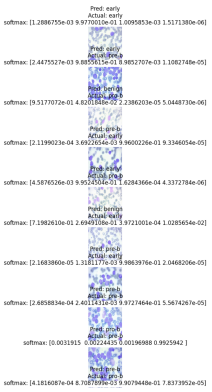


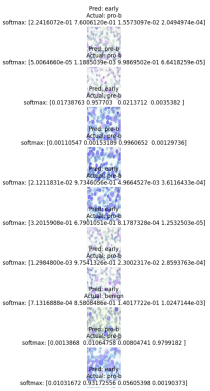
Classification Report is :

	precision	recall	f1-score	support
0	0.98	0.98	0.98	95
1	0.98	0.99	0.99	191
2	1.00	0.99	0.99	206
3	1.00	0.99	1.00	159
accuracy			0.99	651
macro avg	0.99	0.99	0.99	651
weighted avg	0.99	0.99	0.99	651

After doing the pertinent corrections for the weight classes, the final confusion matrix looks like this. The matrix shows an optimal proportion between true and false predictions. In this case it is better to have false positives for non-benign classes.







Future Work

A logical next step is to retrain the model using a more diverse dataset that includes images with varying characteristics. This enhancement would increase the robustness of the training data and is expected to improve the model's accuracy and generalization capability. Incorporating a wider range of samples would better equip the model to handle variability in real-world scenarios.

Following this retraining phase, the model should be evaluated in clinical settings using live laboratory images. These results can then be compared with diagnoses made by expert hematologists, providing a practical assessment of the model's performance and reliability.

Based on the outcomes of this validation, further refinement may be necessary—through fine-tuning or full retraining—to correct any discrepancies. If the model consistently demonstrates high accuracy and aligns well with expert evaluations, it could be considered for deployment in under-resourced or remote healthcare environments. In such settings, the model could serve as a valuable diagnostic support tool, assisting in the early detection of Acute Lymphoblastic Leukemia (ALL) and helping bridge the gap in medical expertise.

References

[1] National Cancer Institute, "Childhood Acute Lymphoblastic Leukemia Treatment (PDQ®)–Patient Version," *National Cancer Institute*, May 2023. [Online]. Available: https://www.cancer.gov/types/leukemia/patient/child-all-treatment-pdq

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[3] M. S. Islam, M. A. N. Baki, M. M. Rahman, and M. M. H. Khan, "A combined deep CNN-LSTM network for the detection of acute lymphoblastic leukemia using peripheral blood smear images," Computers in Biology and Medicine, vol. 145, p. 105405, 2022

[4] S. K. Shafique and S. Tehsin, "Acute lymphoblastic leukemia detection and classification of its subtypes using pretrained deep convolutional neural networks," *Technology in Cancer Research & Treatment*, vol. 20, pp. 1–17, 2021.

[5] S. Rehman, M. A. Khan, M. S. Anwar, and S. W. Baik, "A hybrid deep learning framework for the detection of acute lymphoblastic leukemia using blood smear images," *Microscopy Research and Technique*, vol. 84, no. 12, pp. 2902–2912, 2021. doi: 10.1002/jemt.23814

[6] M. Ghaderzadeh, M. Aria, A. Hosseini, F. Asadi, D. Bashash, and H. Abolghasemi, "A fast and efficient CNN model for B-ALL diagnosis and its subtypes classification using peripheral blood smear images," *International Journal of Intelligent Systems*, vol. 37, no. 7, pp. 5113–5133, 2022. doi: 10.1002/int.22753

[7] H. P. Nguyen, H. T. Le, D. D. Nguyen, and H. X. Nguyen, "Transfer learning with deep convolutional neural networks for cancer cell classification," *Proceedings of the 2019 IEEE-RIVF International Conference on Computing and Communication Technologies*, Danang, Vietnam, 2019. doi: 10.1109/RIVF46745.2019.8968010

[8] A. Spanhol, L. Oliveira, C. Petitjean, and L. Heutte, "Breast cancer histopathological image classification using Convolutional Neural Networks," *2016 International Joint Conference on Neural Networks (IJCNN)*, Vancouver, BC, Canada, 2016, pp. 2560–2567. doi: 10.1109/IJCNN.2016.7727519

[9] T. M. Abbas, A. M. Sarhan, and H. AlZubaidi, "Improved segmentation technique for leukemia detection using color thresholding and morphological operations," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 10, no. 2, pp. 540–546, 2019. doi: 10.14569/IJACSA.2019.0100268