Efficient Deep Learning Architectures for Blood Cell Classification

Aryan Gupta[†], Nils van Es Ostos[‡]

***Abstract*—Accurate blood cell classification is essential for diagnosing a range of medical conditions, from anemia to hematological cancers such as leukemia and lymphoma. While AI-powered medical imaging has made significant advancements, the challenge remains to develop models that balance accuracy with computational efficiency and interpretability, especially in resource-constrained environments. This study proposes a systematic approach to blood cell classification by iterating through different convolutional neural network (CNN) architectures, starting with simple models; and refining through advanced architectures like VGG, ResNet, and various attention mechanisms. The methodology focuses on optimizing performance while ensuring efficiency and interpretability. The results show that the optimized models achieve around 98% accuracy, offering a practical solution with minimal computational cost for clinical and remote applications. This work supports the development of reliable, cost-effective AI solutions for blood cell analysis and provides valuable insights for future research and integration into automated diagnostic systems. The code implementation is available on GitHub.**

***Index Terms*—Deep Learning, Blood Cell classification, Medical Imaging, Image Classification, Convolutional Neural Networks, Attention Mechanisms.**

I. Introduction

Classifying blood cells is essential for diagnosing and monitoring various diseases, as abnormalities in cell count, shape, or behavior often indicate underlying health issues. For instance, low platelet counts can signal clotting disorders, while abnormal levels of specific white blood cells, such as B cells or T cells, may point to cancers like leukemia or lymphoma. Traditionally, blood cell analysis was performed manually by domain experts, which, although accurate, was time-consuming and prone to human error. AI models now offer a faster, scalable alternative, achieving high accuracy in detecting and classifying blood cells. However, ensuring that these models genuinely learn meaningful features, such as cell morphology or texture, is crucial. If a model inadvertently relies on spurious correlations, such as image artifacts or dataset biases, it risks producing misleading results on unseen data, which can lead to incorrect clinical decision-making.

In this study, we focus on the task of blood cell classification, a foundational step in automating hematological analysis. Despite significant advancements in AI-powered medical imaging, the challenge remains to develop models that are not only accurate but also computationally efficient and interpretable. These qualities are particularly important for clinical and resource-constrained settings, where reliable and cost-effective solutions are vital. Addressing this need, our study explores innovative methods to enhance the accuracy and applicability of AI-based blood cell analysis, with the goal of improving diagnostic speed and reliability in diverse environments.

To tackle this challenge, we adopt a systematic and iterative approach to blood cell classification, combining both experimental and guided model design. Starting with basic convolutional neural networks (CNNs), we progressively refine our models, incorporating advanced architectures such as VGG, Autoencoders, InceptionNet, ResNet, and attention-based mechanisms like Channel and Spatial Attention to improve feature extraction and performance. Our approach emphasizes balancing accuracy, computational efficiency, and interpretability; factors critical for clinical deployment. Additionally, we conduct error analysis to address issues like class imbalance and mislabeled annotations, ensuring our models learn meaningful biological features rather than relying on non-relevant patterns. This contribution provides a robust framework for advancing AI-driven blood cell classification and lays the foundation for further integration into medical and diagnostic workflows.

The results of this study can guide researchers and practitioners in developing efficient and accurate models for blood cell classification and related tasks, with potential applications in automated diagnostics, personalized medicine, and integration into clinical decision-support systems.

This report is organized as follows: Section II provides an overview of the state-of-the-art advancements in blood cell classification. Section III delves into the dataset's features, exploratory data analysis, and the overall model pipelining and evaluation workflow. In Section IV, we detail the iterative experiments conducted to understand key CNN components and their influence on performance. Section V explores various model architectures, followed by the incorporation of attention mechanisms to enhance performance in Section VI. Finally, in VII, we compare the performance of all models, discussing interpretability and explainability, in the results section. Concluding remarks and future directions are presented in Section VIII.

II. Related Work

The evolution of deep learning architectures for medical imaging has seen remarkable advancements, with early convolutional neural networks (CNNs) [1] paving the way for

[†]Erasmus Mundus BDMA, University of Padova, email: {aryan.gupta}@studenti.unipd.it
[‡]Erasmus Mundus BDMA, University of Padova, email: {nils.vanesostos}@studenti.unipd.it

more sophisticated models. Initially, CNN based architectures like VGG-16 [2] demonstrated the potential of automatically learning spatial features from peripheral blood cell images, providing a strong baseline despite their large number of parameters. The introduction of InceptionV3 [3] helped to address computational inefficiencies by incorporating inception modules for multi-scale feature extraction, which improved speed and accuracy. ResNet [4] further revolutionized the field by addressing the vanishing gradient problem through residual connections, enabling deeper networks without sacrificing performance.

The dataset used in this study consists of peripheral blood cell images from the BloodMNIST [5] collection, part of the MedMNIST dataset [6]; a large-scale MNIST-like repository of standardized biomedical images. A related work [7] based on the BloodMNIST dataset, presented an automated system for classifying eight types of peripheral blood cells using VGG-16 and InceptionV3. More recent works like AIMIC [8] provide automated methods for deep learning models on these datasets, to benchmark their performance.

More recent innovations have seen the integration of transformers and hybrid models that combine CNNs with attention mechanisms for enhanced performance. These early models were augmented by techniques such as the Convolutional Block Attention Module (CBAM) [9], which allowed the networks to focus on important features in blood cell images, enhancing interpretability and accuracy. Vision Transformers (ViTs), which process images as patches and utilize self-attention mechanisms, have shown great promise in capturing long-range dependencies crucial for biological classification tasks. The MedViT model [10], a hybrid approach combining the strengths of CNNs and ViTs, exemplifies this shift, achieving robust performance on blood cell classification.

Furthermore, the increasing focus on interpretability, as seen in tools like AIMIC, shows the growing emphasis on making AI tools accessible and understandable, fostering their adoption in real-world medical scenarios. Together, these developments reflect a shift from simpler architectures to more complex, hybrid models that are not only more accurate but also more adaptable to real-world challenges in blood cell classification.

III. Processing Pipeline

This section begins by outlining the processes of data acquisition and preprocessing. It then provides a high-level overview of the methods employed, followed by an evaluation of their performance using specific metrics.

A. Dataset Features

The BloodMNIST dataset is based on individual normal cells obtained from individuals without any infections, hematologic or oncologic diseases, and free of pharmacologic treatments at the time of blood collection. The dataset comprises 8 distinct classes representing different blood cell types, shown in Table 1.

TABLE 1: Cell Type Distribution

Cell Class	Images	Distribution %
neutrophils	3329	19.48
eosinophils	3117	18.24
basophils	1218	7.13
lymphocytes	1214	7.10
monocytes	1420	8.31
immature granulocytes	2895	16.94
erythroblasts	1551	9.07
platelets	2348	13.74
Total	**17,092**	**100**

The dataset includes 17,092 images, in multiple resolutions: 28 * 28, 64 * 64, 128 * 128, and 224 * 224 pixels. The dataset is pre-stratified (Fig. 1) and split into training, validation, and test sets in a 7:1:2 ratio. The experiments were conducted using a T4 GPU on Google Colab, which, due to limited runtime and low RAM, presented computational constraints. As a result, this study focuses on 64 * 64 RGB images, striking an optimal balance between maintaining sufficient image resolution for effective classification and ensuring computational efficiency.

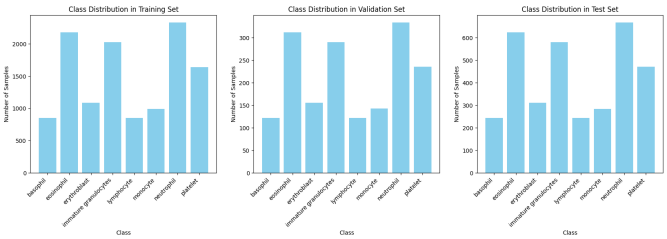
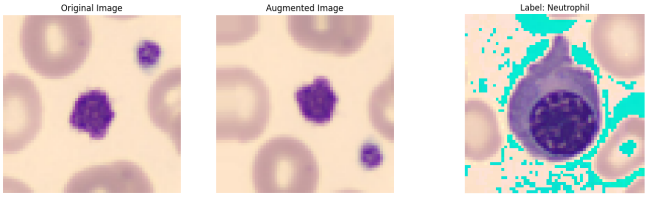


Fig. 1: Stratification across Training, Validation and Test Set

B. Data Processing

Fig. 1 shows a class imbalance across the 8 blood cell categories, which could potentially bias the model towards the majority class. To mitigate this bias, oversampling techniques were applied. Data augmentation was used to oversample the minority classes in the training data by applying transformations such as flipping, rotation and zooming (Fig. 2a).

Another technique to oversample minority class involved generating synthetic data using the SMOTE (Synthetic Minority Oversampling Technique) algorithm, which adds noise to image samples, as shown in Fig. 2b. Undersampling the majority class was avoided as it could lead to loss of critical information without the domain expertise to guide the removal of samples.



(a) Data Augmentation on Sample Image (b) SMOTE Resampling

Fig. 2: Oversampling Techniques

The images were then normalized for consistent input scaling; and the preprocessed data was converted to TensorFlow

datasets, each for train, validation and test sets and batched with a size of 64 for efficient training, to ensure there is no data leak. To ensure reproducibility, random seeds were fixed. The data pipeline integrated Keras functionalities with custom utility scripts consisting of models to streamline model preparation, training, and evaluation efficiently. For the Auto-Classifier model, a separate dataset was created consisting of images paired as both input and target for reconstruction tasks. These representations are then used to support downstream classification tasks.

C. Model Methods

This study focuses on classifying blood cells using a range of algorithms fine-tuned to their optimal configurations through a systematic and structured workflow. We start with simple CNNs, conducting iterative experiments (Section IV) to explore the impact of key components like convolutional layers and max-pooling, achieving strong baseline results. Building on these insights, we investigate established CNN-based architectures such as VGG, InceptionNet, and ResNet (Section V), and experiment with an innovative approach using Autoencoders for classification. The top-performing models are selected based on evaluation metrics emphasizing a high F1 score while minimizing training time and storage requirements. Finally, we enhance these models using attention mechanisms - channel attention emphasizes key features across channels, while spatial attention sharpens the focus on critical regions within the feature map. By combining these approaches, we optimize performance for the blood cell classification task.

D. Model Evaluation

To comprehensively evaluate the model's performance, we considered both training and test metrics to provide a holistic assessment. Training metrics included the total parameters, which estimate the model's learning complexity and memory requirements, the training time for the first epoch as a representation of computational complexity, and the model's memory usage to evaluate storage efficiency. For test metrics, we focused on the classification report, which provides precision, recall, and F1-scores for each class, alongside the weighted F1-score, which accounts for class imbalance to offer a robust measure of the model's overall classification performance. Together, these metrics offer valuable insights into the resource demands and predictive capabilities of the trained models.

IV. CNN Experiments

To systematically understand the influence of various hyperparameters within CNNs and CNN based architectures, we undertook an iterative study to identify effective configurations for the CNN models.

Fixed Configurations: The initial models were designed using standard hyperparameters, with the following assumptions:

- Image Size: Fixed to (64, 64, 3) for consistency.
- Batch Size: 64
- Epochs: 10
- Conv2D Layer: Standard configurations including default filters, kernel size, strides and padding.
- Activation Function: ReLu for introducing non-linearity in convolution layers.
- Pooling: MaxPooling Layer to reduce dimensionality.
- Kernel Initializer: Default Glorot Uniform Initializer.
- Optimizer: Adam, for its adaptive learning capabilities.

This baseline served as the foundation for testing various modifications to the CNN architecture.

Architectural Variations:

To enhance the baseline CNN models, the following aspects of the architecture were systematically modified:

- Number of Conv2D Layers to assess feature extraction performance.
- Effect of MaxPooling Layer after final Convolutional Layer to assess spatial information retention performance.
- Number of neurons in the fully connected layers.
- Variants with and without fully connected layers.

All experiments, presented in Table 2 were conducted without applying data augmentation, ensuring that the observed results reflected the raw capability of the CNN models rather than enhanced training data diversity.

TABLE 2: Performance Comparison of CNN Architectures

Model	Layers	Maxpool	Dense Units	Test Accuracy	Test Weighted F1 Score
1	3	0	0	0.91	0.91
2	3	1	0	0.91	0.91
3	4	0	0	0.93	0.93
4	4	1	0	0.87	0.87
5*	4	1	0	0.95	0.95
6	3	0	128	0.92	0.92
7	3	0	256	0.92	0.92
8	3	1	128	0.91	0.91
9	3	1	256	0.90	0.91
10	4	0	128	0.93	0.93
11	4	0	256	0.91	0.91
12	4	1	128	0.92	0.92
13	4	1	256	0.91	0.91

*Model configuration includes Batch normalization.

Inferences:

- **Batch Normalization** stabilizes training by normalizing layer inputs as a regularizer, especially for deep networks.
- **Convolutional Layers**: 4 layers seem substantial for the given dataset; additional layers introduce unnecessary complexity and risk overfitting.
- **Pooling Layer**: Presence of pooling layers reduce spatial dimensions, help extract dominant features, removing redundant information with low overhead. Without pooling, the model retains redundant information, leading to overfitting and slower training.
- **Dense Layers**: Adding dense layers improved accuracy marginally, but the improvement was not substantial, even with varied number of neurons, which hints towards overfitting of the model.

To train the optimal CNN model effectively while minimizing overfitting and ensuring convergence, specific strategies and parameters were employed. An exponential learning rate scheduler was utilized to dynamically adjust the learning

rate, starting at 0.001, with a decay rate of 0.1 over epochs to facilitate efficient convergence. Batch normalization was implemented to enhance generalization by stabilizing the training process and reducing the model's sensitivity to weight initializations. Using these parameters, the best CNN results were obtained at 96.43% F1-Score, whose training process is shown by the accuracy and loss curves in Fig. 3.

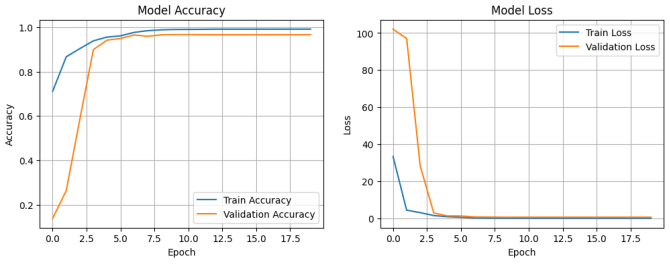


Fig. 3: CNN Model without data augmentation (96.43%)

The optimal model was rerun, this time incorporating oversampled data through the application of data augmentation techniques, with the epoch based accuracy and loss shown in Fig. 4, achieving an F1-Score of 95.04%.

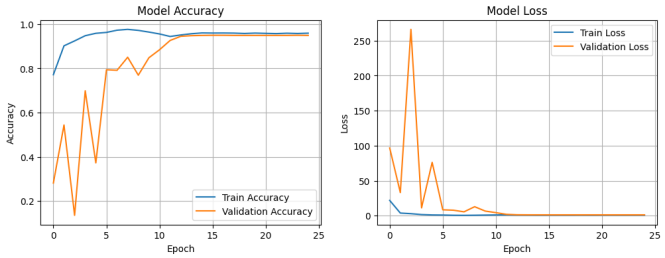


Fig. 4: CNN Model with data augmentation (95.04%)

The accuracy showed a slight decrease with the oversampled balanced data. Since it did not enhance model performance and could potentially lead to overfitting, we chose not to implement oversampling techniques. Furthermore, as noted in the original paper by Acevedo et al., the class imbalance is not significant enough to justify the use of oversampling methods.

V. Model Architectures

The inferences gained from Section IV guide the architectures in this section. The focus is on evaluating the independent predictive capabilities of each architecture. Therefore, fully connected layers and data oversampling were excluded, as they offered minimal improvement and increased the risk of overfitting. The number of epochs was increased to 20 for more complex models to allow sufficient learning time. Dropout layers were added when there were significant discrepancies between training and validation performance, helping to mitigate overfitting. Additionally, a custom convolutional block consisting of a Convolution layer, Batch Normalization, and ReLU Activation was used across all architectures.

A. VGG

VGG-16 consists of 16 learnable layers, which contribute to weight updates. For input images of size (64 * 64 * 3), instead of the original 224 * 224 * 3, the number of filters is adjusted to accommodate the smaller dimensions. Despite this reduction, the network maintains high accuracy even with minimal input sizes. Through combination of layers, we experimented with different VGG architectures (VGG-8, 12, 15 & 18), with VGG-15 giving the best result.

B. Inception Net

For Inception Net, the SGD optimizer was selected over Adam to better align with their training dynamics.

For our case, the Inception architecture has been modified to process input images of size 64 * 64 * 3, making it suitable for smaller datasets or lower-resolution data. This adaptation is a more conservative version of the original Inception model, which is typically designed for larger images (e.g., 128 * 128). The adjustments ensure that the data is not overly downsampled during processing, preserving important features. This makes it an effective choice for handling complex data while leveraging the strengths of Inception blocks for extracting multi-scale features.

C. ResNet

For ResNet, the SGD optimizer was selected over Adam to better align with their training dynamics, similar to Inception Net.

ResNet is adapted to learn residual mappings for input images of size 64 * 64 * 3, preserving its ability to handle complex data with large datasets efficiently. Residual mappings are defined as F(x) = H(x) - x, where H(x) is the desired mapping and x is the input. ResNet34 incorporates Residual Blocks with shortcut (or residual) connections, allowing the input to bypass certain layers, mitigating the vanishing gradient problem and enabling deeper networks.

The two key blocks in ResNet34 are the Identity Block, which consists of two 2D convolutional layers with the input directly added to the output, and the Convolution Block, which includes a convolutional layer with a stride of 2 to reduce spatial dimensions while increasing the number of channels. These blocks form the backbone of ResNet34. For our case, the network has been adjusted to handle smaller input sizes while preserving the hierarchical feature extraction capabilities of the original design, making it effective for complex tasks even with lower-resolution images.

D. Autoencoder based Classifier

For our case, the autoencoder is designed to encode the most important features from input images of size 64 * 64 * 3 into a latent representation of size 64 (`latent_dim=64`). The encoder extracts and compresses the 64 most relevant features of the image, while the decoder reconstructs the original image from the latent representation by minimizing the reconstruction error. This structure ensures that the latent

representation retains critical information about the input while reducing its dimensionality.

Once the autoencoder is trained, the decoder is removed, and the trained encoder is reused to build a classifier. By freezing the encoder's weights, a CNN is connected to the output of the encoder to perform classification tasks. This hybrid model, referred to as the AutoClassifier, combines the pre-trained encoder from the autoencoder with a CNN for classification purposes. The AutoClassifier leverages the compressed feature representations learned during autoencoder training, making it an efficient and effective approach for classification tasks while minimizing the need for additional training on large datasets.

Based on the results from various model architectures presented in Table 3, CNN and VGG emerged as the top-performing models and were selected for further improvement with attention, which we see in the next section.

VI. Adding Attention

Attention mechanisms improve the performance of convolutional neural networks by focusing on the most relevant features in an image, enhancing feature representation and task-specific accuracy.

Channel Attention particularly through **Squeeze-and-Excitation (SE)** Attention, enhances feature maps by adaptively recalibrating channel importance. The squeeze operation aggregates spatial information to create a global channel descriptor, while the excitation phase applies learned weights to emphasize relevant channels and suppress irrelevant ones. This improves the model's ability to focus on key features, boosting performance in convolutional networks.

Spatial Attention (SA) highlights important regions in an image by generating attention maps based on spatial dimensions, allowing the model to focus on relevant spatial features and improving performance in tasks like image classification.

Convolutional Block Attention Module (CBAM) integrates both Channel and Spatial Attention to further enhance feature extraction and potentially improve classification accuracy.

We explore the impact of these attention mechanisms on the following optimized CNN and VGG architectures, as seen in Fig. 5.

A custom convolutional block typically consists of a Convolution layer, followed by Batch Normalization, and a ReLU Activation function. In attention-based architectures, an attention block (such as channel, spatial, or combined attention) is added after each modular combination, which allows the model to focus on important features by assigning different attention weights to various regions or channels in the input data. Fig. 6 shows the changes in the proposed architectures. Dropout layers with a rate of 0.2 are applied after flattening the feature maps to improve regularization.

These combinations allow us to compose 3 distinct attention based CNN architectures, namely:

- SE-CNN: CNN with Squeeze Excitation Channel Attention Blocks

Layer	Output Size
Input Layer	$64 \times 64 \times 3$
1^{st} Convolution Block	$64 \times 64 \times 32$
Max Pooling	$32 \times 32 \times 32$
2^{nd} Convolution Block	$32 \times 32 \times 64$
Max Pooling	$16 \times 16 \times 64$
3^{rd} Convolution Block	$16 \times 16 \times 128$
Max Pooling	$8 \times 8 \times 128$
4^{th} Convolution Block	$8 \times 8 \times 256$
Max Pooling	$4 \times 4 \times 256$
Flatten	$1 \times 1 \times 4096$
Output	$1 \times 1 \times 8$

(a) CNN Architecture

Layer	Output Size
Input Layer	$64 \times 64 \times 3$
$2\times$ 1^{st} Convolution Block	$64 \times 64 \times 32$
Max Pooling	$32 \times 32 \times 32$
$2\times$ 2^{nd} Convolution Block	$32 \times 32 \times 64$
Max Pooling	$16 \times 16 \times 64$
$2\times$ 3^{rd} Convolution Block	$16 \times 16 \times 128$
Max Pooling	$8 \times 8 \times 128$
$2\times$ 4^{th} Convolution Block	$8 \times 8 \times 256$
Max Pooling	$4 \times 4 \times 256$
Flatten	$1 \times 1 \times 4096$
Output	$1 \times 1 \times 8$

(b) VGG Architecture

Fig. 5: Proposed CNN and VGG Architectures

Layer	Output Size
Input Layer	$64 \times 64 \times 3$
1^{st} Convolution Block	$64 \times 64 \times 32$
Max Pooling	$32 \times 32 \times 32$
Attention Block	$32 \times 32 \times 32$
2^{nd} Convolution Block	$32 \times 32 \times 64$
Max Pooling	$16 \times 16 \times 64$
Attention Block	$16 \times 16 \times 64$
3^{rd} Convolution Block	$16 \times 16 \times 128$
Max Pooling	$8 \times 8 \times 128$
Attention Block	$8 \times 8 \times 128$
4^{th} Convolution Block	$8 \times 8 \times 256$
Max Pooling	$4 \times 4 \times 256$
Attention Block	$4 \times 4 \times 256$
Flatten	$1 \times 1 \times 4096$
Drop Out (0.2)	$1 \times 1 \times 4096$
Output	$1 \times 1 \times 8$

(a) CNN Architecture

Layer	Output Size
Input Layer	$64 \times 64 \times 3$
$2\times$ 1^{st} Convolution Block	$64 \times 64 \times 32$
Max Pooling	$32 \times 32 \times 32$
Attention Block	$32 \times 32 \times 32$
$2\times$ 2^{nd} Convolution Block	$32 \times 32 \times 64$
Max Pooling	$16 \times 16 \times 64$
Attention Block	$16 \times 16 \times 64$
$2\times$ 3^{rd} Convolution Block	$16 \times 16 \times 128$
Max Pooling	$8 \times 8 \times 128$
Attention Block	$8 \times 8 \times 128$
$2\times$ 4^{th} Convolution Block	$8 \times 8 \times 256$
Max Pooling	$4 \times 4 \times 256$
Attention Block	$4 \times 4 \times 256$
$2\times$ 5^{th} Convolution Block	$4 \times 4 \times 512$
Max Pooling	$2 \times 2 \times 512$
Attention Block	$2 \times 2 \times 512$
Flatten	$1 \times 1 \times 2048$
Dropout (0.2)	$1 \times 1 \times 2048$
Output	$1 \times 1 \times 8$

(b) VGG Architecture

Fig. 6: Proposed CNN and VGG Attention Architectures

- SA-CNN: CNN with Spatial Attention Blocks
- CBAM-CNN: CNN with Convolutional Block Attention Modules (CBAM)

Similarily, VGG based attention architectures include:

- SE-VGG15: VGG-15 with Squeeze Excitation Channel Attention Blocks
- SA-VGG15: VGG-15 with Spatial Attention Blocks
- CBAM-VGG15: VGG-15 with Convolutional Block Attention Modules (CBAM)

VII. Comparison of Model Performances

Given the variety of models, their performance is evaluated based on accuracy, training time, and memory usage to identify the optimal model for blood cell classification.

A. Models without Attention

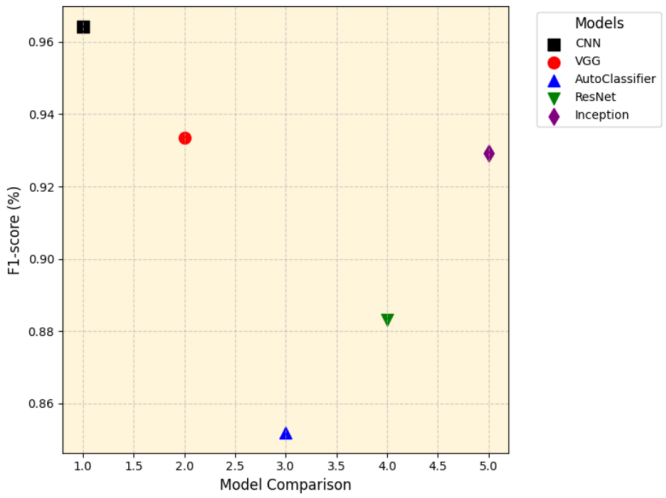


Fig. 7: Model Performance across Weighted F1 Score

1) F1 Score Analysis: We achieved the highest weighted F1 score of 96.43% using a simple CNN, seen in Fig. 7. However, as the model architecture became more complex with additional layers, performance began to decline. This can be attributed to the relatively small and less complex dataset, which may not support the increased capacity of larger models effectively. The AutoClassifier had the lowest performance, presumably because it depends heavily on how well the features are represented in the latent space. Even after fine - tuning the latent dimension, it's possible that not all important features can be captured effectively. Some features might be highly significant, while others, though less important, still play a role. When these features get compressed, valuable information might be lost, which could explain why the model struggled in comparison to others.

2) Analysis of Training Time and Memory Storage: As seen in Fig. 8, VGG models maintain a strong F1 score while requiring approximately three times less training time compared to the best-performing models, such as CNNs. Inception, due to its larger size, takes longer to train, especially in the first epoch, but performs on par with VGG. While ResNet and Autoclassifiers show poorer performance, the Autoclassifier is more efficient in terms of memory and training time, whereas ResNet demands significantly more memory - about double that of the CNN model.

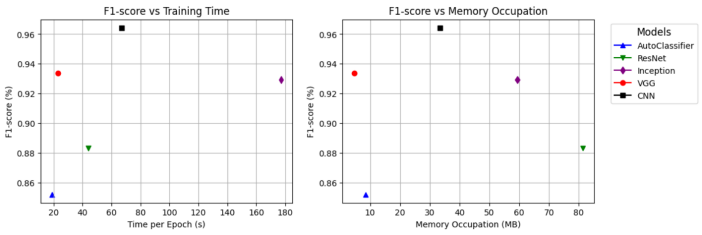


Fig. 8: Model Performance across Training Time and Memory Storage

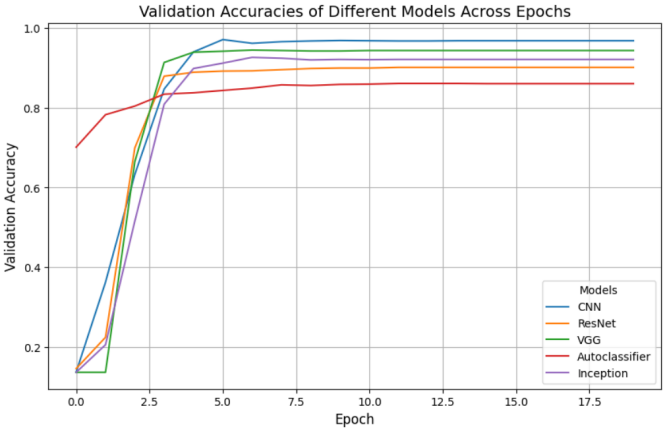


Fig. 9: Model Validation Accuracies across epochs

3) Analyzing Convergence across Epochs: Fig. 9 shows that all models achieve rapid initial learning, with validation accuracy stabilizing after around 5 epochs. CNN and VGG models perform the best, with CNN achieving the highest accuracy, while the Autoclassifier model converges quickly but to a lower accuracy. ResNet and Inception models perform well but do not surpass CNN or VGG.

B. Models with Attention

We now focus on CNN and VGG based attention models, which have emerged as the most suitable candidates for our task.

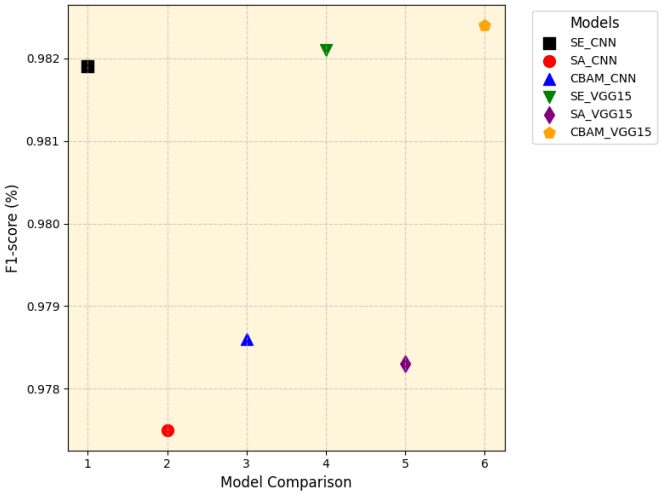


Fig. 10: F1 Score across Attention based Models

1) Analysis of F1 Score: Attention-based CNN architectures outperform other models, consistently achieving F1 scores above 97% (Fig. 10). VGG models benefit slightly more from attention mechanisms than CNN models, likely due to their ability to better utilize the added complexity. SE-based models, incorporating channel attention, outperform SA-based models, which rely on spatial attention. While combined attention mechanisms like CBAM may be overly complex for simpler CNN architectures, they perform slightly better with

VGG models, surpassing other attention-based approaches and even the best CNN models.

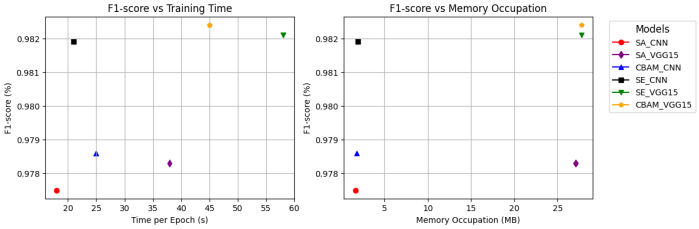


Fig. 11: F1 Score vs Training Time and Memory Storage

2) Analysis of Training Time and Memory Storage: Attention-based CNN models are significantly faster (Fig. 11), taking approximately 60% less time compared to VGG models, which is expected due to the additional computational layers in VGG. CNN attention models are also highly space-efficient, requiring only around 5 MB on average, while VGG attention models occupy approximately 25 MB. Both maintain an accuracy of over 97%, making them ideal for our use case. For example, a standard CNN achieved a 96% F1 score, occupying 30 MB and taking 60 seconds to train. In contrast, an attention-based CNN improved to a 98% F1 score while reducing memory usage to 5 MB and training time to just 20 seconds.

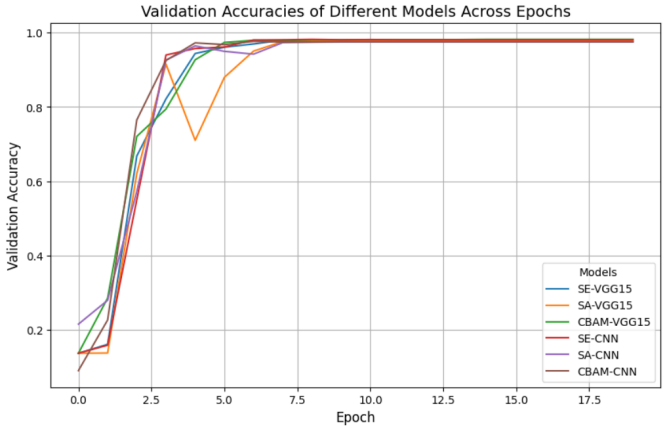


Fig. 12: Attention based Model Validation Accuracies across epochs

3) Analyzing Convergence across Epochs: Fig. 12 shows that all models achieve rapid improvements in validation accuracy within the first few epochs, with accuracy stabilizing after around 5 epochs. SE-CNN and CBAM-CNN models achieve the highest final accuracy, while SA-VGG15 shows some fluctuations before stabilizing. Most models reach near-perfect accuracy, and therefore are strong contenders as learning architectures.

Table 3 summarizes all the models and their comparative parameters. The highest accuracy (98.33%) and weighted F1 Score (98.24 %) is achieved by the CBAM-VGG15 model. However, the most efficient models in terms of training time and memory storage i.e. the amount of parameters are Attention Based CNNs, since their number of parameters are atleast 10 times less than Attention based VGGs.

C. Model Interpretability

Grad-CAM (Gradient-weighted Class Activation Mapping) is a visualization technique that highlights regions in an image that strongly influence a model's predictions. It computes the gradients of the target class score with respect to the feature maps of a chosen convolutional layer. These gradients are used to weight and combine the feature maps, producing a heatmap overlaid on the original image. The highlighted areas on the heatmap represent the regions where the model is 'focusing' to make its decision.

Fig. 13 shows how the features are captured in attention based models. As per the Grad-CAM heatmaps, attention is able to keep the gradient's focus to the dense part of the cell and highlights the influential area in the decision making process.

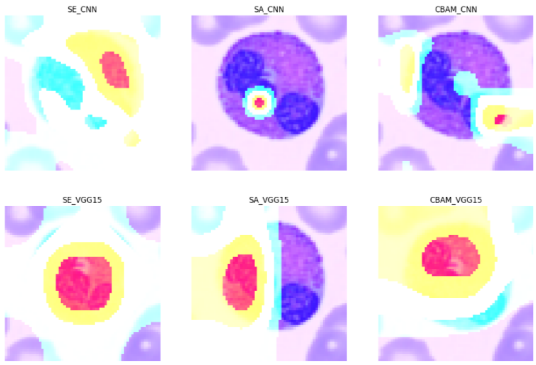


Fig. 13: Model Interpretability using Attention

D. Model Explainability

Beyond achieving high interpretability, it is essential for our models to be explainable. While our optimized models demonstrate impressive performance with over 98% accuracy, understanding how they make predictions is equally important. To this end, we examined various images to assess the classification challenges from a human perspective. As illustrated in Fig. 14, certain cell types appear visually similar, making it difficult to distinguish them even with the human eye.

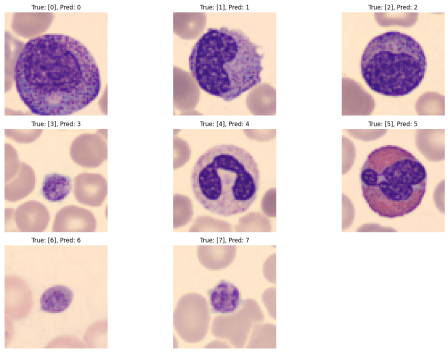


Fig. 14: Correct Predictions by the CNN Model.

TABLE 3: Performance comparison of CNN architectures.

Model	Accuracy	F1-Score	Precision	Recall	Params	Time (s)
CBAM-VGG15	0.9833	0.9824	0.9824	0.9825	7.27M	45
SE-VGG15	0.9828	0.9821	0.9822	0.9822	7.27M	58
SE-CNN	0.9792	0.9819	0.9819	0.9819	0.46M	21
CBAM-CNN	0.9792	0.9786	0.9786	0.9787	0.47M	25
SA-VGG15	0.9784	0.9763	0.9763	0.9763	7.10M	38
SA-CNN	0.9746	0.9775	0.9775	0.9775	0.42M	18
CNN	0.9597	0.9643	0.9644	0.9643	8.78M	67
VGG	0.9377	0.9336	0.9341	0.9336	1.21M	23
Inception	0.9290	0.9293	0.9299	0.9293	15.59M	177
ResNet	0.8851	0.8831	0.8836	0.8834	21.31M	44
AutoClassifier	0.8413	0.8519	0.8529	0.8515	2.25M	19

Fig. 15 shows examples of misclassified images. Upon reviewing them manually, they appear difficult to classify and may even be mislabeled. This highlights the potential for a human-in-the-loop approach, where domain knowledge can help improve the model and correct any data inconsistencies.

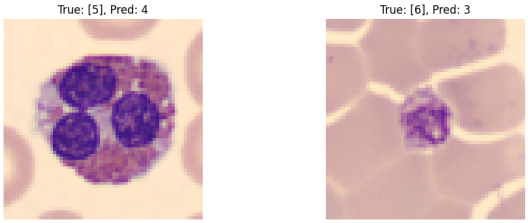


Fig. 15: Incorrect Predictions by CNN Model

VIII. Conclusion & further work

In this study, we explored various architectures for blood cell classification, achieving a peak accuracy of more than 98 % using attention-based CNNs. We developed a data pipeline incorporating techniques to enhance the dataset and fine-tuned models to strike a balance between accuracy, model simplicity, parameter efficiency, and training time, ensuring cost-effective computation. Additionally, we focused on model interpretability through gradient heatmaps to verify that the models learned meaningful patterns. While some errors could be attributed to image dimensionality or potential mislabeling by pathologists, we believe our model offers a strong performance in both efficiency and accuracy.

While the current model demonstrates strong performance, there are several avenues for further improvement. Transfer learning and fine-tuning with pretrained models such as EfficientNet and MobileNet could enhance accuracy, particularly for edge computing applications where efficiency is key. Looking forward, exploring advanced architectures like Spiking Neural Networks (SNNs) and Transformers (e.g. vision transformers such as MedViT) may provide new opportunities. Additionally, addressing class imbalance through targeted classification techniques could improve performance for underrepresented classes. Tackling challenges like mislabeled annotations and data diversity is crucial for enhancing model robustness and reliability. Finally, incorporating ensemble models with majority voting could improve both performance and interpretability.

In conclusion, this project has been a valuable learning experience in developing and fine-tuning model architectures through hands-on experimentation and informed decision-making. We successfully replicated and adapted existing models from research papers to address our specific problem and dataset. Through careful evaluation and comparison, we identified the best-performing models and fine-tuned them for optimal results. A key takeaway from this journey was learning to recognize when a model has reached its peak, balancing compute time, memory usage, and accuracy. Ultimately, we discovered that sometimes, the most effective approach is not to overcomplicate, but rather to focus attention on simpler architectures that deliver outstanding performance - because, as we found, attention truly is all you need.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] K. Simonyan, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[5] A. Acevedo, A. Merino, S. Alférez, Á. Molina, L. Boldú, and J. Rodellar, "A dataset of microscopic peripheral blood cell images for development of automatic recognition systems," *Data in brief*, vol. 30, p. 105474, 2020.

[6] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification," *Scientific Data*, vol. 10, no. 1, p. 41, 2023.

[7] A. Acevedo, S. Alférez, A. Merino, L. Puigví, and J. Rodellar, "Recognition of peripheral blood cell images using convolutional neural networks," *Computer methods and programs in biomedicine*, vol. 180, p. 105020, 2019.

[8] R. Liu, W. Dai, T. Wu, M. Wang, S. Wan, and J. Liu, "Aimic: Deep learning for microscopic image classification," *Computer Methods and Programs in Biomedicine*, vol. 226, p. 107162, 2022.

[9] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

[10] O. N. Manzari, H. Ahmadabadi, H. Kashiani, S. B. Shokouhi, and A. Ayatollahi, "Medvit: a robust vision transformer for generalized medical image classification," *Computers in Biology and Medicine*, vol. 157, p. 106791, 2023.