Multi-Class Blood Cell Detection Using YOLO

A Comprehensive Report on Blood Cell Detection and Classification

Submitted as part of the course DH 307: R&D Project

Afnan Abdul Gafoor

Roll Number: 22B2505

agafnan@gmail.com

Department of Metallurgical Engineering and Material Science

Indian Institute of Technology Bombay

Supervised by:
Nirmal Punjabi
Koita Centre of Digital Health
Indian Institute of Technology Bombay

May 9, 2025

Contents

1 Executive Summary

This report details a 12-week project to develop an automated system for detecting and classifying nine peripheral blood cell types—basophil, eosinophil, erythroblast, immature-granulocyte, lymphocyte, monocyte, neutrophil, platelet, and lymphoblast—using a YOLO [1] object detector. Starting from segmentation and classification prototypes, we constructed a synthetic dataset by compositing manually-verified cell segments onto 31 diverse backgrounds. Retraining on this data yielded an overall mAP@0.5 of 0.994 on a held-out validation set. A comprehensive robustness analysis under different perturbations confirmed the model's resilience to blur, noise, and brightness shifts. Our pipeline promises to accelerate hematological diagnostics by reducing manual microscopy workload.

2 Introduction

2.1 Background and Motivation

White blood cells (WBCs) are critical indicators of immune function and hematologic health. Manual differential counts require trained hematologists and are time-consuming and subjective. Recent advances in deep learning have enabled automated image analysis in medical domains [2], but robust detection of multiple cell types in a single field remains a challenge due to variability in staining, morphology, and background.

2.2 Problem Statement

We aim to build a single-stage object detector that can localize and classify nine blood cell types in microscope images. Each class exhibits substantial intra-class variation and sometimes low contrast against the background (e.g., neutrophils), complicating traditional classification approaches.

2.3 Objectives

1. Develop a segmentation-guided preprocessing pipeline to isolate high-quality cell crops.
2. Generate a balanced synthetic dataset by compositing segments on varied backgrounds.
3. Retrain a YOLO [1] detector on the synthetic data and achieve high detection accuracy.
4. Conduct a parameter sensitivity study to assess robustness under blur, noise, brightness, and geometric perturbations.
5. Compare detection performance to classification-only methods and analyze practical deployment considerations.

3 Literature Review

3.1 Traditional Approaches to WBC Classification

Early work relied on handcrafted features—morphological descriptors, texture measures, and color histograms—followed by conventional classifiers such as support vector machines (SVM) or k-nearest neighbors (k-NN) [3]. While these techniques can perform well on isolated, high-quality cell images, they often falter in real-world conditions with overlapping cells and complex backgrounds.

3.2 Deep Learning in Medical Imaging

Convolutional neural networks (CNNs) have revolutionized medical image analysis by learning hierarchical features end-to-end. Surveys demonstrate CNN success in radiology, histopathology, and cytology tasks [2]. However, most WBC studies formulate detection as a classification problem on pre-cropped images.

3.3 Object Detection for Medical Imaging

Object detectors like Faster R-CNN [4], RetinaNet [5], and YOLO [1] simultaneously localize and classify multiple objects. These methods have been applied to tumor detection and cell counting, but their use in multi-class WBC detection remains relatively unexplored.

4 Methodology

4.1 Dataset Description

This study leveraged the publicly available Peripheral Blood Cell (PBC) dataset [6], which contains 17,092 images of individual normal cells acquired with a CellaVision DM96 in the Core Laboratory at the Hospital Clínic of Barcelona. Images are organized into eight classes—neutrophils, eosinophils, basophils, lymphocytes, monocytes, immature granulocytes (promyelocytes, myelocytes, metamyelocytes), erythroblasts, and platelets—each at 360 × 363 px in JPEG format and annotated by expert pathologists.

To capture clinically significant leukemic blasts, we augmented this set with 870 lymphoblast images drawn from the ALL-IDB dataset [7]. In total, our nine-class corpus comprises:

- Basophil
- Eosinophil
- Erythroblast
- Immature-granulocyte
- Lymphocyte
- Monocyte
- Neutrophil
- Platelet
- Lymphoblast

Figure 1 shows a horizontal montage of one representative image per class.

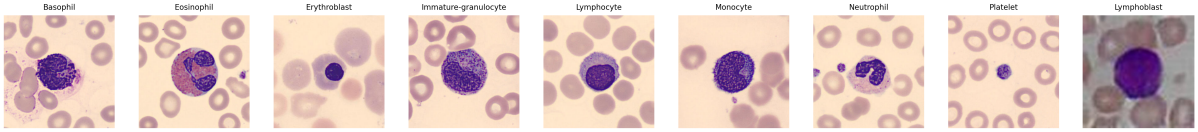


Figure 1: Representative examples of the nine blood-cell classes in our dataset.

4.2 Preprocessing Pipeline

A key innovation in this project was the extensive preprocessing pipeline developed to create a robust training dataset. The process involved:

4.2.1 Cell Segmentation

Individual cells were segmented from the original images using the following approach:

- Color-based thresholding to identify cell regions
- Morphological operations to refine segmentation boundaries
- Watershed transformation for separating overlapping cells

This process was particularly challenging for neutrophils and other cells with similar color properties to the background.

Detailed, class-wise segmentation procedures for all nine cell types are presented in Section 7.5.

4.2.2 Background Randomization

To enhance model robustness and generalization capability, segmented cells were placed on randomized backgrounds:

- Generation of diverse background textures
- Random placement of segmented cells on these backgrounds
- Variation in cell density and distribution

This approach helped the model learn to identify cells regardless of background conditions, simulating the variability present in real-world microscopy images.

The full methodology for placing segments on backgrounds (and grid patterns) is detailed in Section 7.6.

4.3 Training Procedure

4.3.1 Data Splitting

The raw dataset (and its segmented cell crops) was first partitioned via stratified sampling into training (80%), validation (15%), and test (5%) subsets to ensure consistent class representation, especially for rarer cell types. To prevent information leakage when creating synthetic samples, all grid-composite and segment-on-background images for training are generated **only** from the training subset, and those for validation are generated **only** from the validation subset. The test subset remains untouched until final evaluation.

4.3.2 Dataset Generation

Our methodology involved generating two types of training images:

1. **Grid Composite Images:** We created composite images in 1×1, 2×2, 3×3, and 4×4 grid configurations. Each grid cell contained a randomly selected cell image from the training split. This approach resulted in:

- Train set: 6,400 grid images (evenly distributed across grid sizes)
- Validation set: 1,600 grid images (evenly distributed across grid sizes)

2. **Segment-on-Background Images:** Individual segmented cells were placed on background images with:
   - 15–35 segments per image
   - Non-overlapping cell placement
   - Class-balanced selection where possible
   - Backgrounds synthetically generated by ChatGPT [8] from the original dataset scenes.
   - Train set: 6,400 segment images
   - Validation set: 1,600 segment images

The final combined dataset included both types, resulting in approximately 12,800 training images and 3,200 validation images, with an additional set of test images and segmented test cells for evaluation (example training samples shown in Figure 3).

An overview of the entire preprocessing pipeline is illustrated in Figure 2.

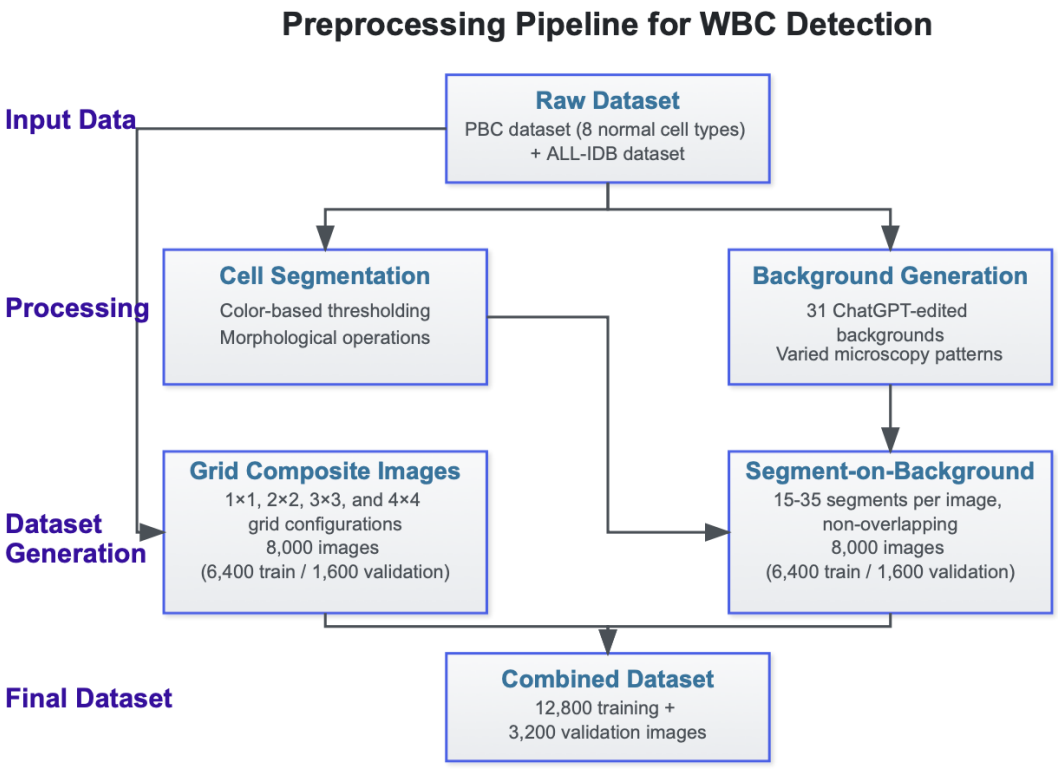


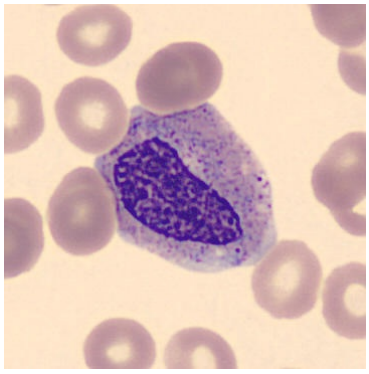
Figure 2: Preprocessing pipeline for WBC detection. The raw datasets are transformed into grid composites and segment-on-background images, which are combined into a final training and validation dataset.

4.3.3 Hyperparameters

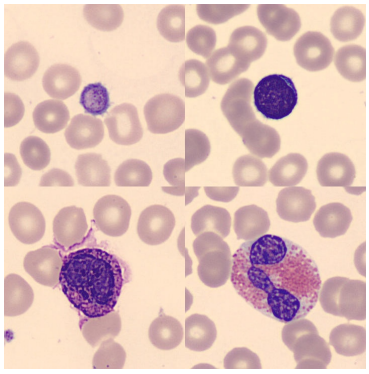
The YOLO11m [9] model was trained using the following non-default settings:

- **Batch size:** 32
- **Initial learning rate:** 0.001 (`lr0`)
- **Optimizer:** AdamW
- **Number of epochs:** 300
- **Early stopping patience:** 50
- **Image size:** 1024×1024 px

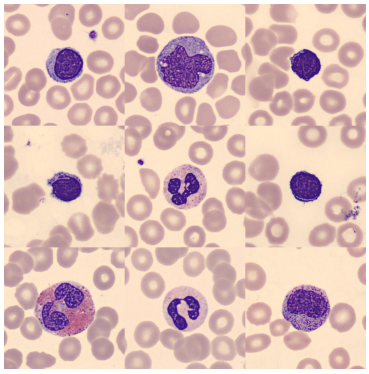
All other settings were kept at the YOLO11 defaults.



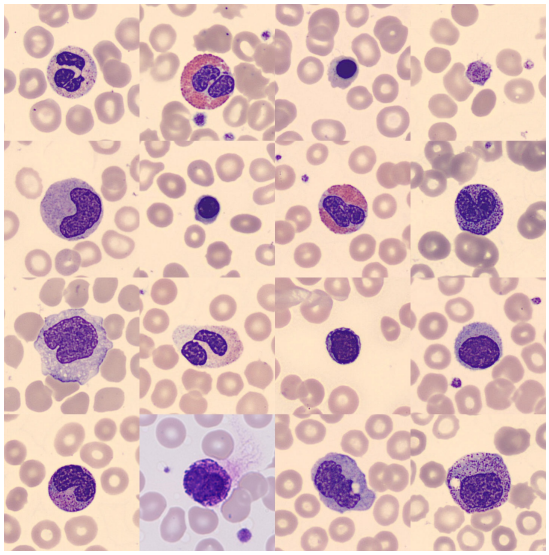
(a) Grid 1×1



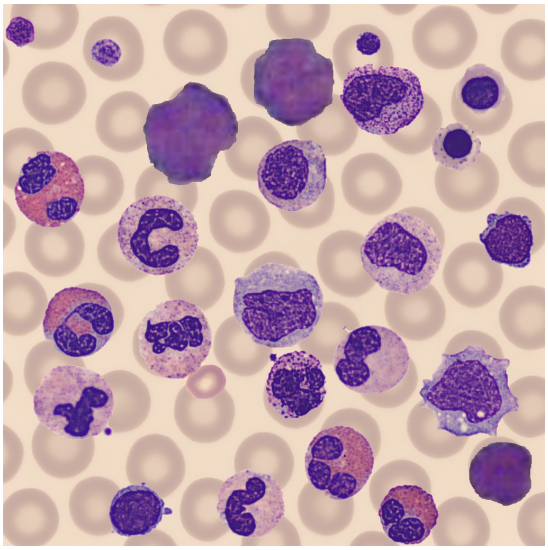
(b) Grid 2×2



(c) Grid 3×3



(d) Grid 4×4



(e) Segments on background

Figure 3: Example training-set images: (a) 1×1 grid, (b) 2×2 grid, (c) 3×3 grid, (d) 4×4 grid, and (e) segment-on-background.

4.3.4 Training Resources

The model was trained on an NVIDIA RTX A6000 GPU and required approximately 9 hours and 42 minutes to complete. Training was early stopped at epoch 97 based on validation performance.

5 Results

5.1 Quantitative Evaluation

Table 1: Validation summary of detection performance

Class	Images	Instances	P	R	mAP@0.5	mAP@0.5:0.95
All	3200	48992	0.991	0.991	0.994	0.986
Basophil	2403	5238	0.997	0.996	0.995	0.994
Eosinophil	2402	5826	0.995	0.993	0.995	0.991
Erythroblast	2411	5503	0.989	0.990	0.995	0.985
Immature-granulocyte	2389	5684	0.968	0.982	0.990	0.983
Lymphocyte	2388	5445	0.995	0.998	0.995	0.995
Monocyte	2381	4758	0.991	0.989	0.994	0.989
Neutrophil	2396	5352	0.988	0.974	0.993	0.986
Platelet	2397	6026	0.998	1.000	0.995	0.965
Lymphoblast	2392	5160	1.000	1.000	0.995	0.989

5.2 Confusion Matrix

The confusion matrix (Figure 4) provides detailed insight into the model's classification performance, highlighting which cell types are most frequently confused.

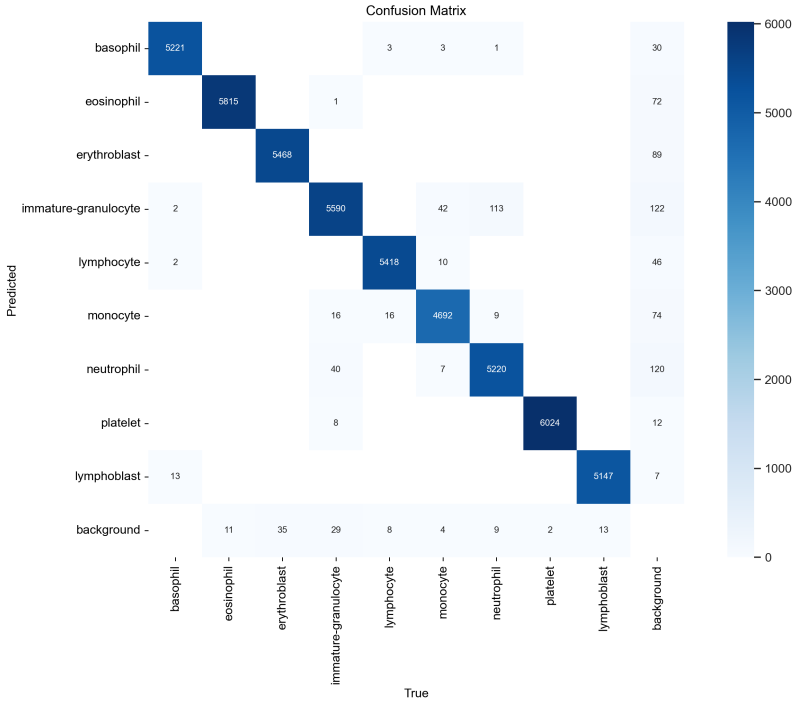


Figure 4: Confusion matrix for the final YOLO11 model, showing true vs. predicted labels across all nine blood-cell classes.

5.3 Precision–Recall Curves

Figure 5 presents the precision–recall trade-off for each class, illustrating detection performance at varying confidence thresholds.

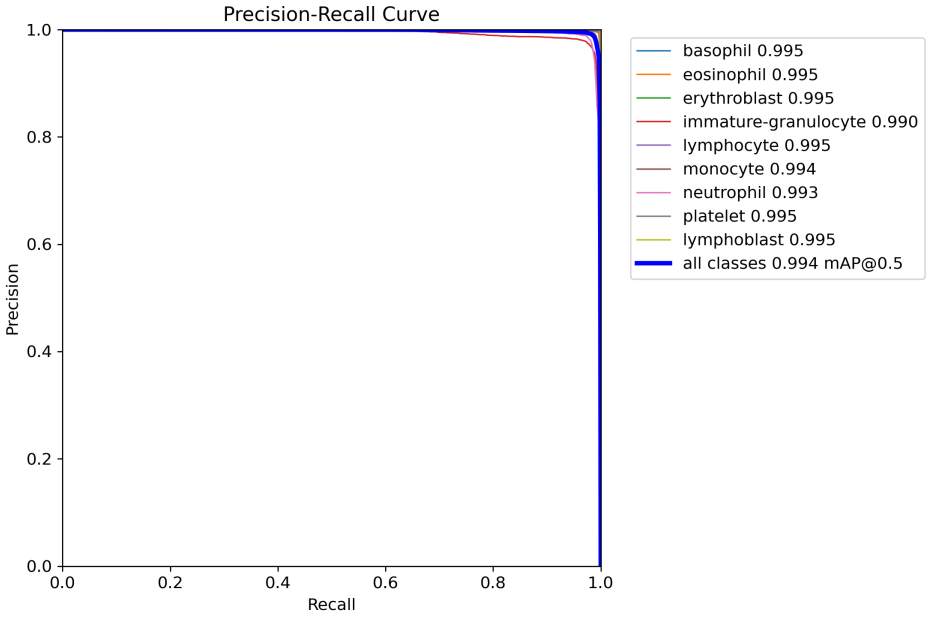


Figure 5: Precision–Recall curves for the final model on the validation set, one curve per cell class.

5.4 Training and Validation Curves

Figure 6 shows the full set of training vs. validation curves over all epochs, including losses (box, cls, dfl) and key metrics (precision, recall, mAP50, mAP50–95).

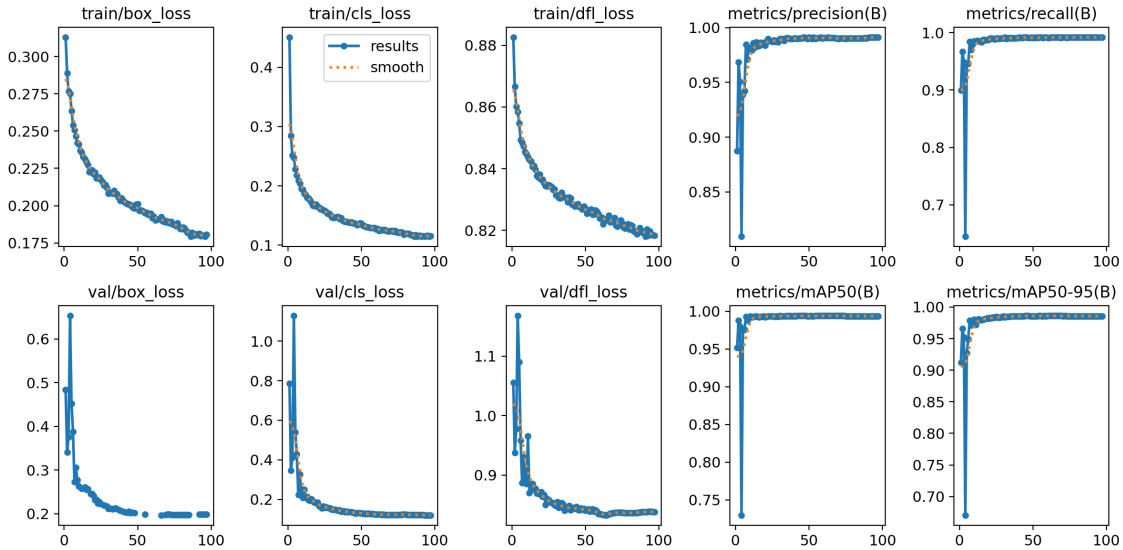


Figure 6: Training (top row) and validation (bottom row) curves over epochs for box loss, classification loss, distribution focal loss, precision, recall, mAP@50, and mAP@50–95.

5.5 Visualization of Detection Results

Figure 7 shows representative examples of successful detections on composite test images, demonstrating accurate localization and classification under varied conditions.

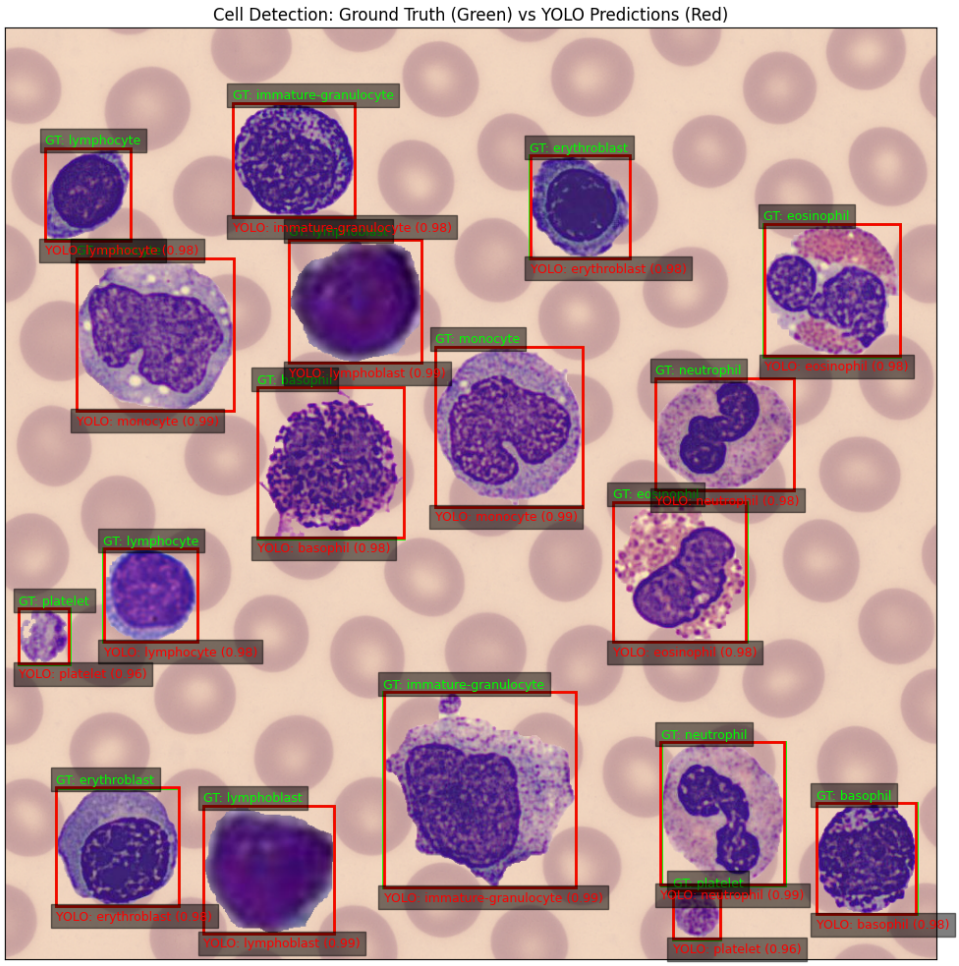


Figure 7: Examples of successful detections by the final YOLO11 model on synthetic composite images.

6 Discussion

6.1 Detection Performance Overview

Our YOLO11 detector achieved an overall mean Average Precision (mAP@0.5) of 0.994 on the validation set (Figure 5). Per-class mAP ranged from 0.990 (immature-granulocyte) to 0.995 (basophil, eosinophil, erythroblast, lymphocyte, platelet, lymphoblast), with neutrophils at 0.993. The confusion matrix (Figure 4) reveals:

- **High accuracy** (>98%) for basophils, eosinophils, erythroblasts, lymphocytes, platelets, and lymphoblasts.
- **Moderate confusion** between immature-granulocytes and monocytes (113 misclassifications), reflecting their similar morphology.
- **Some false positives** of background patches as neutrophils or lymphocytes, suggesting room for improvement in background discrimination.

Figure 7 shows representative composite-image detections with tight bounding boxes and high confidence scores.

6.2 Comparison with Classification-Only Methods

Switching from a slide-level classifier to an object detector provided several advantages:

- *Multiple instances*: Ability to detect and classify multiple cells per image, rather than assuming one centered cell.
- *Contextual cues*: Improved separation of morphologically similar types (e.g., immature-granulocytes vs. monocytes) by leveraging spatial context.
- *Robustness to scale*: Effective handling of variable cell sizes without manual cropping.
- *Workflow efficiency*: Eliminated the need for perfectly segmented, single-cell inputs at inference time.

6.3 Impact of Preprocessing and Synthetic Augmentation

Our novel segmentation–background pipeline was critical to performance:

- **Balanced classes**: Synthetic composites ensured each cell type was evenly represented during training.
- **Background variability**: Randomized and grid-based placement on 31 ChatGPT-edited backgrounds reduced overfitting to any single microscopy pattern.
- **Color robustness**: The model learned to disregard color variations in the background, focusing on the stained nuclear and cytoplasmic features.
- **Low-contrast cells**: Over 5 rounds of parameter sensitivity testing (Table 8), we verified that segmentation and augmentation improved detection even under blur, noise, and brightness shifts.

6.4 Limitations and Future Directions

Despite strong overall performance, several challenges remain:

- **Neutrophil vs. background**: Neutrophils were sometimes confused with background artifacts—future work could integrate texture-based features or additional stain deconvolution.
- **Rare cell types**: Lymphoblast samples were limited; expanding ALL-IDB or other leukemic datasets would improve robustness.
- **Domain gap**: Synthetic backgrounds may differ from clinical slides—domain adaptation or fine-tuning on real-world images is warranted.
- **Computational cost**: High-resolution (1024×1024) inference yields superior accuracy but may be prohibitive in resource-constrained settings; exploring model-compression or lower-resolution pipelines could help.
- **Cross-dataset generalization**: Performance on external datasets remains to be validated and improved—future studies should benchmark and adapt the model to diverse publicly available blood-cell collections.

Overall, our segmentation-guided, synthetic augmentation strategy combined with YOLO11

detection delivers a powerful framework for automated peripheral blood cell analysis, with clear paths for further enhancement.

7 Project Timeline and Evolution

7.1 Week 1–2: Problem Formulation and Initial Model Testing

The first two weeks of the project focused on problem formulation, literature review, and initial model testing for blood cell classification.

7.1.1 Problem Definition

The core objective was defined as developing a machine learning model capable of classifying different types of blood cells with mixed magnifications from a single image.

7.1.2 Literature Review

Two key research papers were analyzed during the initial phase:

- Goswami et al. [10] proposed a classifier-enhanced ResNet-50 model for erythroblast differentiation with limited data. Their approach achieved 98.72% accuracy by using ResNet-50 as a feature extractor and evaluating various traditional machine learning classifiers (SVM, XGB, KNN, RF) on the extracted features.

- Dwivedi and Dutta [11] developed Microcell-Net, a specialized deep neural network for multi-class classification of microscopic blood cell images, employing convolutional layers for extracting spatial and hierarchical features.

7.1.3 Initial Model Testing

For this phase, all single-cell images were sourced from the White Blood Cells dataset on Kaggle [12], and composite images were generated from these during preprocessing. During Week 2, a ResNet-50–based [13] classifier was trained on individual blood-cell patches. Key observations included:

- The model achieved a test loss of 0.0844 and an accuracy of 97.30% on held-out single-cell images (Figure 8a).

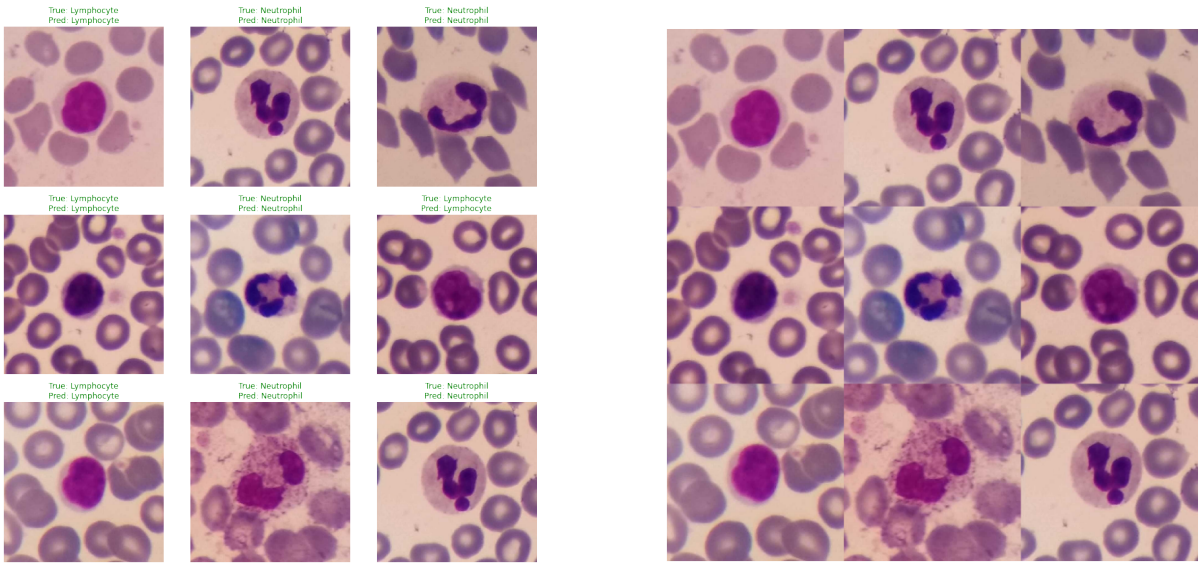
- When applied to a composite image containing nine different cell patches, the model failed to correctly classify each cell. Instead, it predicted only two classes—*Eosinophil* and *Lymphocyte*—none of which were present in that composite (Figure 8b).

7.1.4 Strategic Pivot

Based on the shortcomings of the classification-only approach, namely its inability to localize multiple cell instances and its poor generalization to composite images, we pivoted at the end of Week 2 to an object-detection framework. We selected YOLO (You Only Look Once) [1] over two-stage detectors like Faster R-CNN [4] for the following reasons:

- **Simultaneous Localization and Classification:** Both YOLO and Faster R-CNN predict bounding boxes and class probabilities, but YOLO does so in a single forward pass.

- **Single-Stage Detector vs. Two-Stage Detector:** YOLO's unified architecture is faster and simpler to train than Faster R-CNN's proposal-and-refinement pipeline, enabling rapid



(a) Prediction on single-class single image.

(b) Combined image used for prediction.

Figure 8: Model outputs on (a) an isolated cell patch and (b) a nine-cell composite image.

iteration on biomedical images.

- **Scale Robustness:** YOLO's grid-based detection scheme handles mixed magnifications well, while Faster R-CNN often requires carefully tuned region proposals.
- **Real-Time Performance:** YOLO achieves near–real-time inference, an advantage for potential diagnostic applications, whereas Faster R-CNN is generally slower at test time.

7.2 Week 3–4: YOLO Implementation and Evaluation

Following our strategic pivot, Weeks 3–4 were devoted to training the Ultralytics implementation of YOLO11 [9] on our primary Blood Cells dataset and rigorously evaluating its performance on both individual and composite images.

7.2.1 Data Cleaning and Preparation

- **Label Cleanup:** Platelet images without corresponding YOLO-format label files were identified and removed, as many platelet samples lacked annotations. .
- **Preprocessing:** All remaining images were resized to 384×384 pixels and labels normalized to YOLO's bounding-box format.

7.2.2 Model Training

The YOLO11 model was fine-tuned using the configuration and hyperparameters summarized in Table 2.

7.2.3 Validation and Prediction

We evaluated the trained YOLO11 model on held-out single-cell images to quantify per-class detection performance and to identify systematic errors.

- **Validation Performance:** Table 3 summarizes precision (P), recall (R), mAP@0.5, and mAP@0.5–0.95 for each class on the single-cell validation set.

Parameter	Value
Base weights	`yolo11n.pt`
Train/Validation split	80% / 20%
Epochs	100
Batch size	16
Input resolution	384 × 384 pixels
Optimizer	AdamW
Initial learning rate (lr_0)	0.001
Dropout	0.1
Early stopping patience	50 epochs
Workers	8
Device	MPS (Apple Silicon)
Data augmentation	Disabled for initial run

Table 2: YOLO11 training configuration and hyperparameters.

- **Sample Prediction:** Figure 9 shows a representative single-class validation image with its YOLO11 prediction.
- **Error Analysis:** The validation table (Table 3), sample prediction (Figure 9), confusion matrix (Figure 10a), and Precision–Recall curves (Figure 10b) all reveal that platelets exhibit the lowest detection performance, with many background regions incorrectly labeled as platelets.
- **Platelet Misclassifications:** Figure 11 provides examples of platelet images where background was incorrectly labeled as platelets.

Class	Images	Instances	P	R	mAP@0.5	mAP@0.5–0.95
all	3 535	3 535	0.954	0.955	0.976	0.927
basophil	259	259	0.981	0.988	0.992	0.963
eosinophil	630	630	0.973	0.987	0.993	0.965
erythroblast	320	320	0.972	0.968	0.988	0.924
immature granulocyte	574	574	0.949	0.960	0.975	0.905
lymphocyte	234	234	0.996	0.990	0.995	0.987
monocyte	281	281	0.961	0.989	0.988	0.900
neutrophil	664	664	0.959	0.978	0.989	0.935
platelets	397	397	0.803	0.738	0.867	0.818
lymphoblast	176	176	0.996	1.000	0.995	0.947

Table 3: Per-class validation performance on single-cell images.

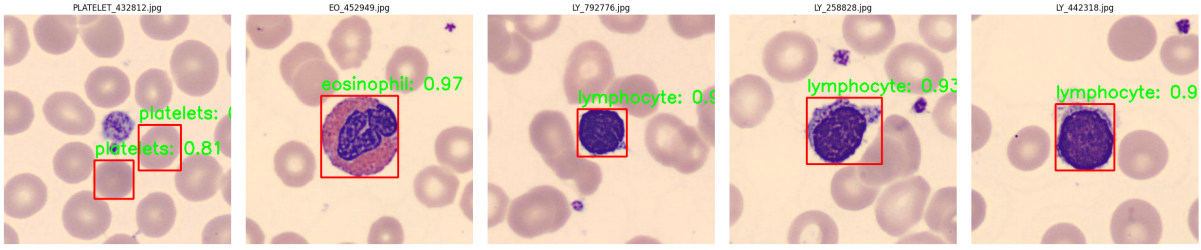
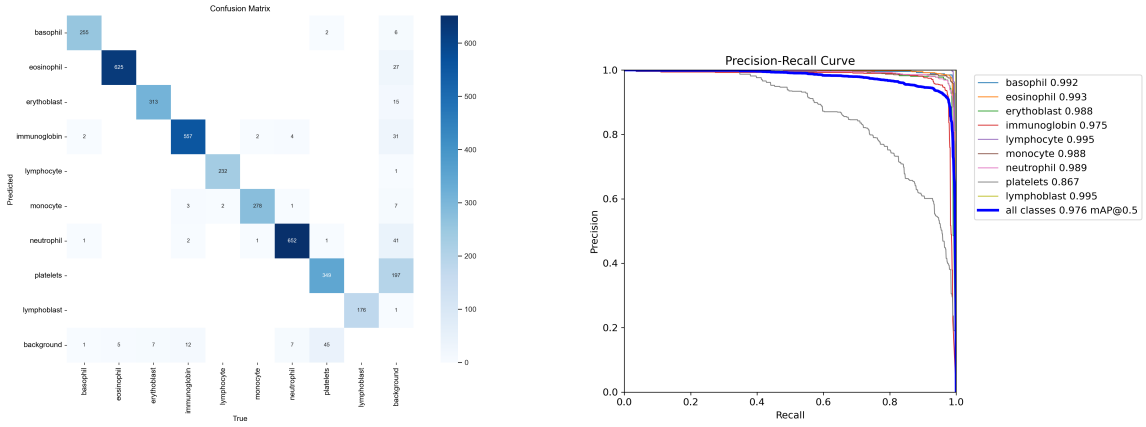


Figure 9: YOLO11 prediction on a single-class validation image.



(a) Confusion Matrix (b) Precision–Recall Curves

Figure 10: Error analysis on the single-cell validation set highlighting platelet misdetections.

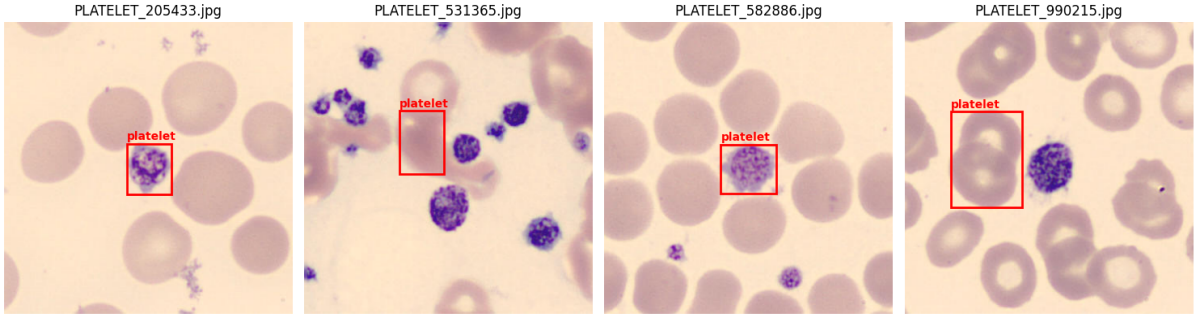


Figure 11: Examples of platelet validation images where background was incorrectly detected as platelets.

7.2.4 Composite Image Scenarios and Generation Pipeline

To assess YOLO11's performance on multi-cell layouts, we implemented a Python pipeline that stitches individual validation images into grids and evaluates detection performance across the composite. The core functions are shown in Listing 1.

```
def select_random_images(img_dir, num_images):
    # Select random images from directory
def create_combined_image_with_labels(selected_images, label_dir,
                                      class_names, grid_size=(3,3),
                                      magnification=1.0, target_size=
                                         None):
    # Create combined image using pre-selected images
    # Return combined_img, all_labels
def visualize_predictions_and_truth(combined_img, all_labels, results,
                                    grid_size=(3,3), figure_size
                                       =(10,10),
                                    title=None):
    # Display image with GT labels in text format
    # Overlay predicted boxes and labels
```

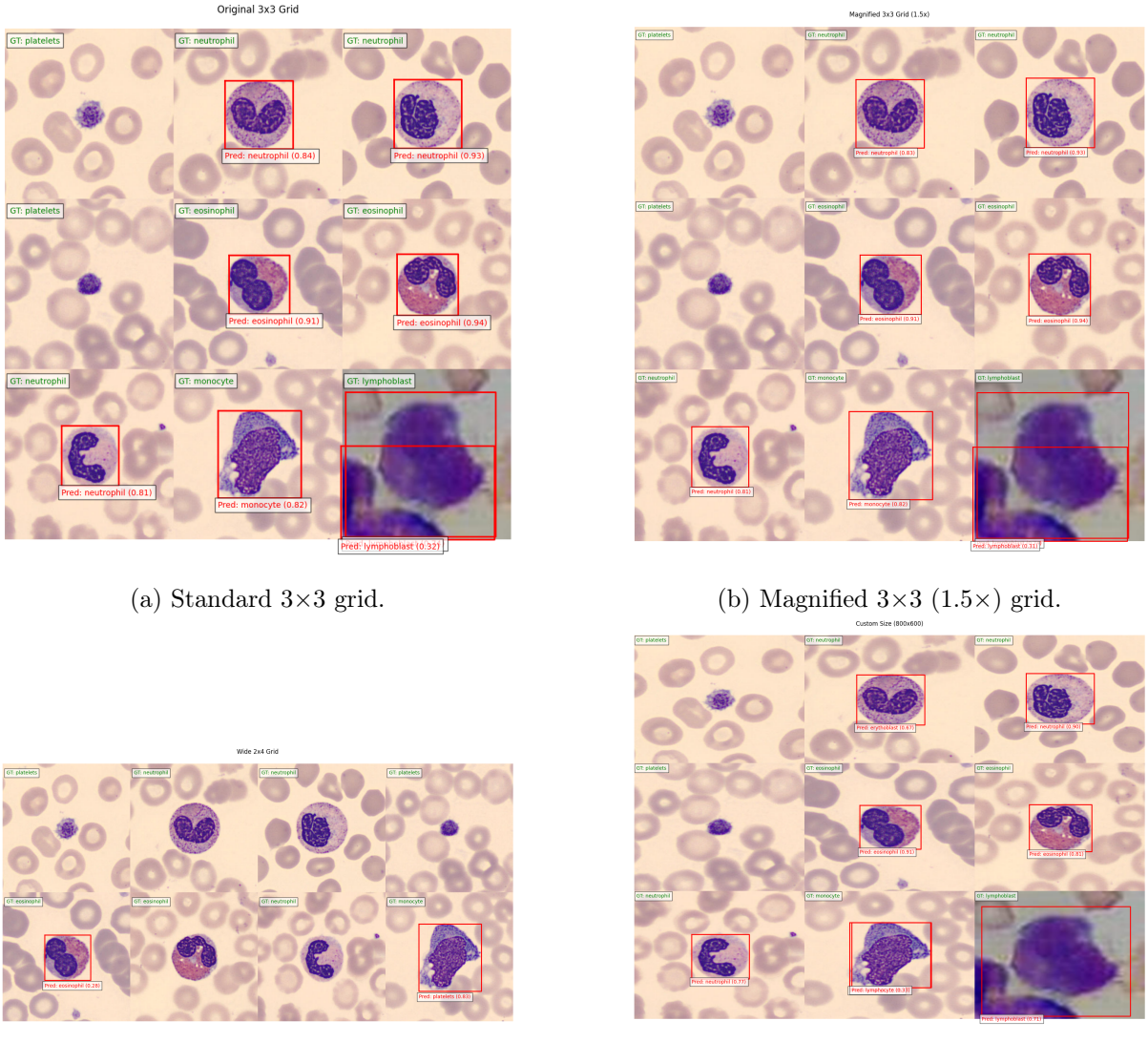
Listing 1: Composite-image generation and visualization code.

Using this pipeline, we evaluated four composite layouts with different configurations:

- A standard 3×3 grid of randomly selected validation images.

- The same images with 1.5x magnification.
- A wider 2×4 grid using eight images.
- A grid with custom-sized image patches (800x600 pixels).

These composite layouts allow us to assess YOLO11's detection performance across varied multi-image scenarios, as shown in Figure 12.



(a) Standard 3×3 grid.

(b) Magnified 3×3 (1.5×) grid.

(c) Wide 2×4 grid.

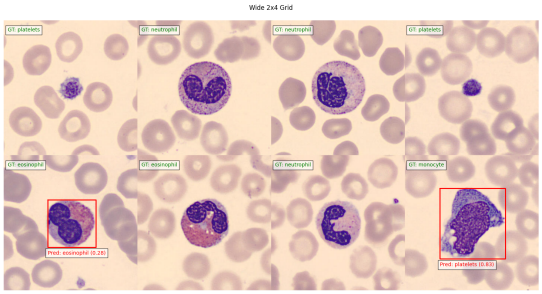
(d) Custom layout (800×600 px).

Figure 12: YOLO11 predictions on four composite-image scenarios.

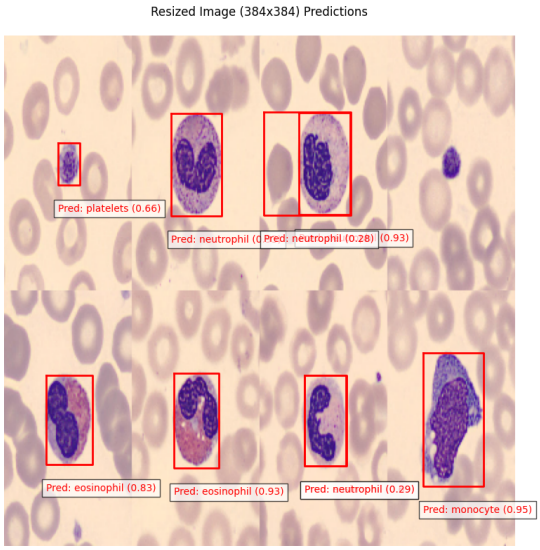
Wide-Grid Scenario Inference As illustrated in Figure 13a, the original 2×4 composite (1440 × 726 px) produced only two valid detections. After resizing to 384×384 px, the number of detections increased to seven (Figure 13b), demonstrating the model's sensitivity to input resolution.

7.2.5 Key Findings

- The model performed well on individual cell images but only moderately on combined image configurations.



(a) Original 2×4 grid (1440 × 726 px): only 2 detections.



(b) Resized to 384×384 px: 7 detections.

Figure 13: YOLO11 predictions before and after resizing the wide 2×4 composite.

- Ground truth annotations, especially for platelets, require correction to improve detection accuracy.
- Predictions on magnified grids did not show significant deterioration, confirming robustness to scale.
- The model exhibits bias when predicting on different image sizes (e.g., reduced performance on the 2×4 grid), which can be mitigated by advanced augmentation techniques such as mosaic, mixup, and copy-paste.

7.3 Week 5–6: Data Curation and Augmentation Optimization

During Weeks 5-6, the focus shifted to performance evaluation and optimization of the YOLO model for both individual and combined image configurations.

7.3.1 Performance Assessment on Individual Images

The initial model was evaluated on individual cell images with performance metrics shown in Table 3.

7.3.2 Performance on Combined Images

For combined images in a 3×3 configuration, the model showed reduced performance as detailed in Table 4.

Based on these initial results, it was concluded that while the model performed well on individual images, its performance on combined images was hindered by false positives and false negatives. Further, the Platelets class showed significantly lower precision and recall compared to other classes.

7.3.3 Data Curation and Relabeling

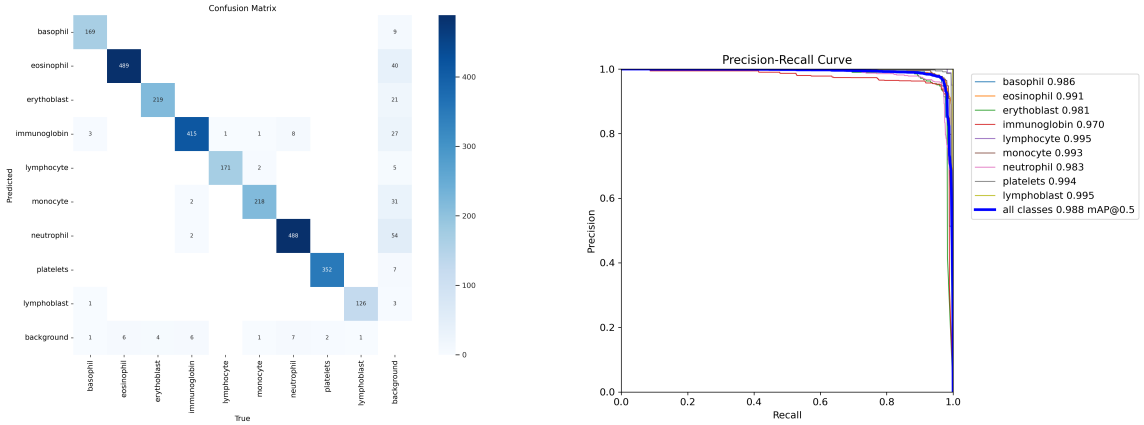
To address the performance issues, particularly with the Platelets class, MakeSense.ai [14] was used to accurately relabel this problematic class. Following relabeling, the model was retrained

Table 4: Overall Performance Metrics for Combined Images (3x3)

Metric	Value
True Positives	2750
False Positives	559
False Negatives	778
Precision	0.831
Recall	0.779
F1 Score	0.804
Total Predictions	3309
Total Ground Truth	3528
Avg. Predictions per Image	0.936
Accuracy	0.673

with the dataset split into train, validation, and test sets (70%, 15%, 15% respectively). Training parameters were kept similar to those in Table 2, except that the model was trained for 300 epochs on an NVIDIA RTX 3050 with a batch size of 128, and augmented using mixup and mosaic techniques.

The improved model's performance on individual images is illustrated through the confusion matrix (Figure 14a) and precision-recall curve (Figure 14b).



(a) Confusion Matrix for Individual Images after Relabeling

(b) Precision-Recall Curve for Individual Images after Relabeling

Figure 14: Confusion matrx and P-R Curve obtained after relabelling platelets.

The detailed performance metrics for individual classes after relabeling are shown in Table 5.

Post-relabeling results showed significant improvement in the Platelets class, with precision increasing from 0.803 to 0.990 and recall from 0.738 to 0.992.

7.3.4 Augmentation Parameter Optimization

Following the data curation, experiments were conducted with various augmentation parameter settings to optimize performance on combined images of different configurations. The model was trained with mixup (blending two images and their labels) and mosaic (combining four training images into one) techniques.

Table 5: Performance metrics for individual blood cell classes after relabeling

Class	Images	Precision (P)	Recall (R)
All	2694	0.978	0.983
Basophil	174	0.988	0.989
Eosinophil	495	0.972	0.977
Erythroblast	223	0.955	0.969
Immunoglobin	425	0.953	0.981
Lymphocyte	172	0.987	1.000
Monocyte	222	0.995	0.962
Neutrophil	503	0.978	0.980
Platelets	354	0.990	0.992
Lymphoblast	127	0.983	1.000

Performance on 2×2 Combined Images Figure 15 shows the model's detection capability on 2×2 combined images when trained with Mosaic: 1.0 and Mixup: 0.5.

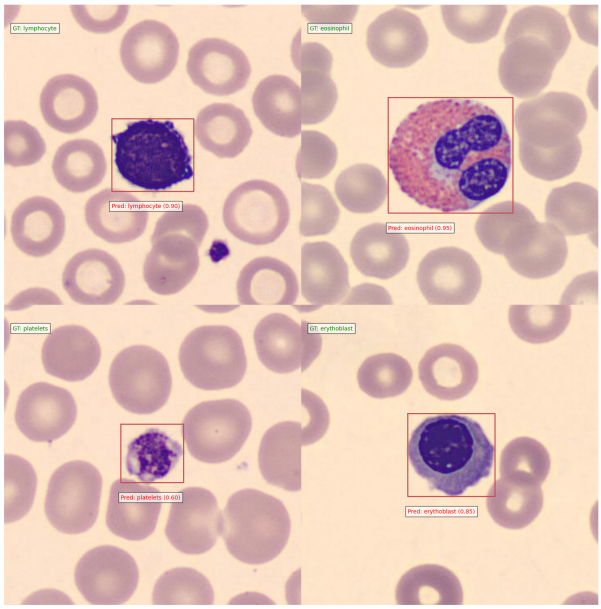


Figure 15: Detection on 2x2 image (Mosaic: 1.0, Mixup: 0.5)

The quantitative performance metrics for 2×2 combined images are detailed in Table 6.

Performance on 3×3 Combined Images Figure 16 shows detection results for two parameter configurations side by side: Mosaic = 1.0, Mixup = 0.5 (left) and Mosaic = 0.5, Mixup = 0.5 (right).

Table 7 aggregates the performance metrics for three different configurations of Mosaic and Mixup on 3×3 combined images.

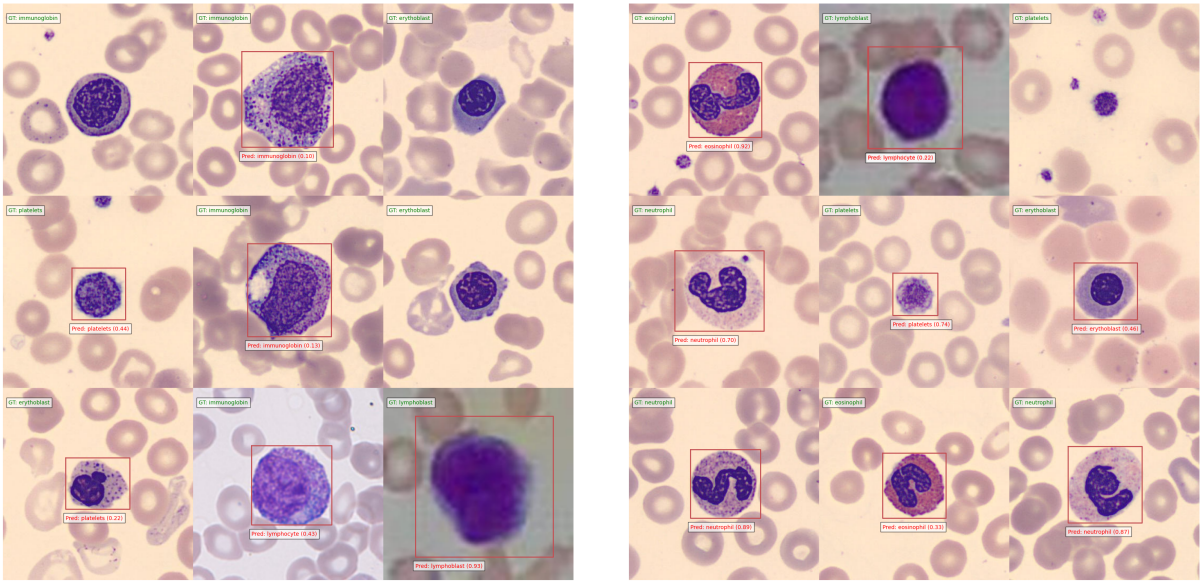
7.3.5 Key Findings and Future Direction

The experiments conducted during Weeks 5-6 revealed several important insights:

- The models performed exceptionally well on 1×1 and 2×2 images, with the 2×2 configuration achieving an accuracy of 0.914.

Table 6: Overall Performance Metrics for Combined Images (2x2 Images with Mosaic: 1.0, Mixup:0.5)

Metric	Value
True Positives	2634
False Positives	189
False Negatives	59
Precision	0.933
Recall	0.978
F1 Score	0.955
Total Predictions	2823
Total Ground Truth	2693
Avg. Predictions per Image	1.048
Accuracy	0.914



(a) Mosaic = 1.0, Mixup = 0.5 (b) Mosaic = 0.5, Mixup = 0.5

Figure 16: Detection on 3×3 combined images under two augmentation settings.

- Performance on 3×3 images was significantly lower, with the best configuration achieving an accuracy of 0.556.
- The high performance on 2×2 images was likely due to the mosaic augmentation, which inherently trains on 2×2 patterns.
- Reducing the mosaic value while increasing the mixup value improved performance on 3×3 images, as evidenced by the increase in recall from 0.100 to 0.699.

7.4 Weeks 7–8: Composite Image Generation, Model Training, and Data Leakage Correction

During Weeks 7–8, we generated composite images of varying grid sizes, trained YOLO11 on these datasets, identified and resolved data-leakage issues, and implemented a robust train/validation/test splitting strategy for reliable performance evaluation.

Table 7: Overall performance metrics for 3×3 combined images under various mosaic (M) and mixup (X) settings.

Metric	**M=1.0, X=0.5**	**M=0.5, X=0.5**	**M=0.1, X=1.0**
True Positives	270	1748	1882
False Positives	17	451	714
False Negatives	2422	944	810
Precision	0.941	0.795	0.725
Recall	0.100	0.649	0.699
F1 Score	0.181	0.715	0.712
Total Predictions	287	2199	2596
Total Ground Truth	2692	2692	2692
Avg. Predictions per Image	0.106	0.816	0.964
Accuracy	0.100	0.556	0.553

7.4.1 Composite Image Generation Strategy

A systematic approach was implemented to generate composite blood cell images, providing a realistic test bed for cell detection algorithms. This process transforms individual cell images into multi-cell composite grids while properly maintaining label information.

Grid Generation Algorithm The core algorithm creates composite images by:

Algorithm 1 Composite Image Generation

for *each grid size* $g \in \{1, 2, 3, 4\}$ **do**
  for $i = 1$ *to* $N_{samples}$ **do**
    Create empty composite image of size $(g \cdot h,\ g \cdot w,\ 3)$ Initialize empty label list **for** *row* $r = 0$ *to* $g - 1$ **do**
      for *column* $c = 0$ *to* $g - 1$ **do**
        Select random cell class and image
        Place image at position (r, c) in composite grid
        Retrieve original image labels
        Transform each $(\text{class\_id}, x_c, y_c, w, h)$ to $(\text{class\_id}, \frac{x_c+c}{g}, \frac{y_c+r}{g}, \frac{w}{g}, \frac{h}{g})$
        Add transformed labels to label list
    Save composite image and corresponding labels

Coordinate Transformation When placing a cell image into a grid position (r, c), we must transform its bounding box coordinates from the original normalized values to the composite image space:

$$\begin{aligned} x'_c &= \frac{x_c + c}{g}, & y'_c &= \frac{y_c + r}{g}, \\ w' &= \frac{w}{g}, & h' &= \frac{h}{g} \end{aligned} \tag{1}$$

Where:

- g is the grid size (e.g., $g = 3$ for a 3×3 grid)
- (r, c) are the row and column indices where the image is placed

- (x_c, y_c, w, h) are the original normalized coordinates and dimensions
- (x'_c, y'_c, w', h') are the transformed coordinates and dimensions

This process is illustrated in Figure 17, which shows an example on a 3×3 grid.

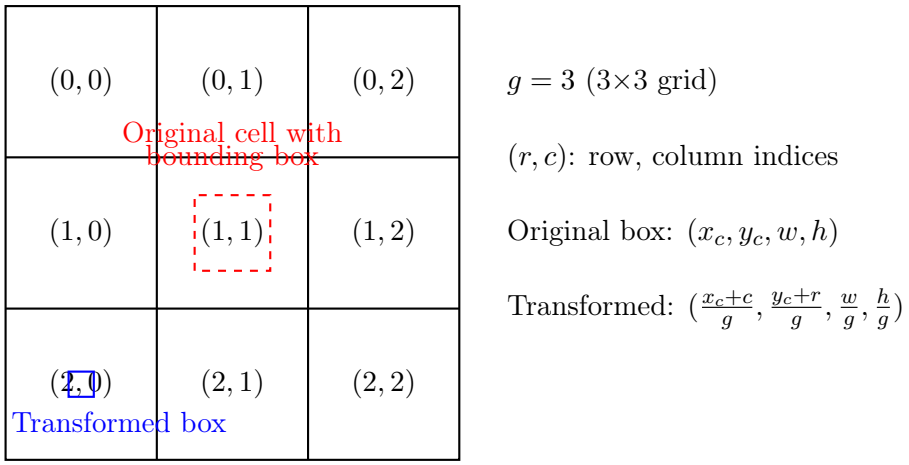


Figure 17: Coordinate transformation for a 3×3 grid. The original bounding box in position $(1, 1)$ is transformed to normalized coordinates in the composite image space.

Data Organization The generated dataset was structured as follows:

- Each grid size (1×1, 2×2, 3×3, 4×4) has its own directory containing:
  - `images/`: JPEG files of composite images
  - `labels/`: Text files with YOLO-format labels
- Train/validation split applied with 80% training, 20% validation
- A `dataset.yaml` file defining paths and class names for YOLO training

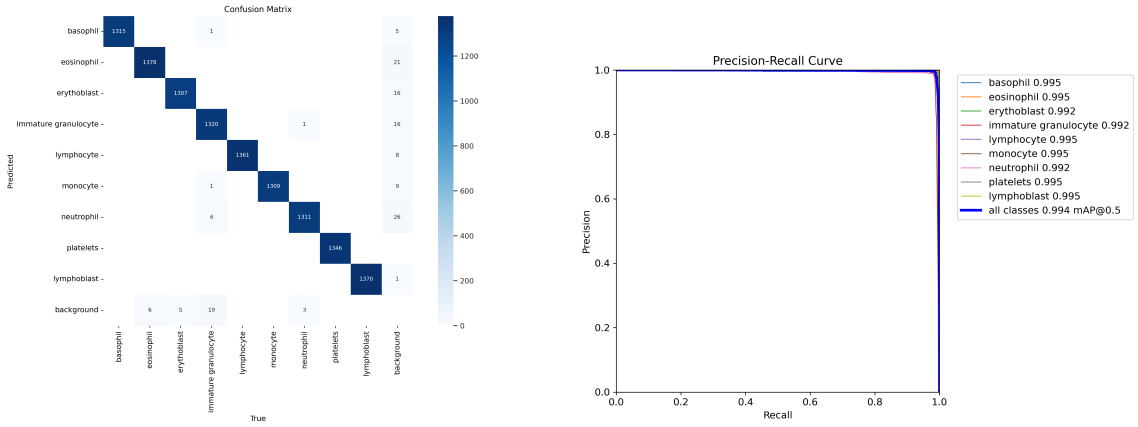
For each grid size configuration, we generated 2,000 images by sampling with replacement from our original cell dataset, resulting in 8,000 total composite images. This approach enabled us to test the YOLO11 model's performance across varied cell densities and spatial arrangements, providing robust evaluation metrics for cell detection tasks.

7.4.2 Model Performance

Figures 18a and 18b show, respectively, the confusion matrix and precision–recall curve for the validation set during Week 7.

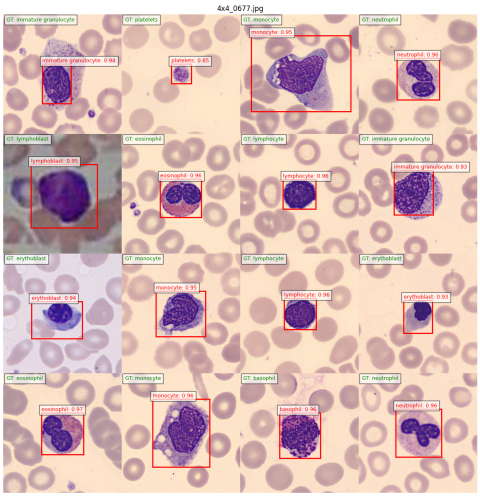
7.4.3 Detection Performance on Various Grid Sizes

Figure 19 summarizes detection results on composite images of different grid sizes during Week 7 testing, where we upgraded to the `yolo11m.pt` backbone (all other hyperparameters as in Weeks 5–6.

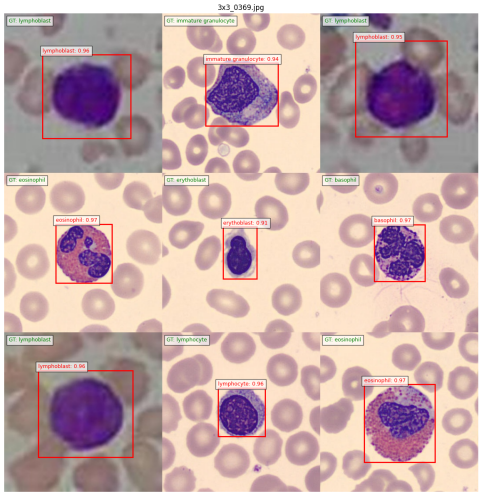


(a) Confusion Matrix (b) Precision–Recall Curve

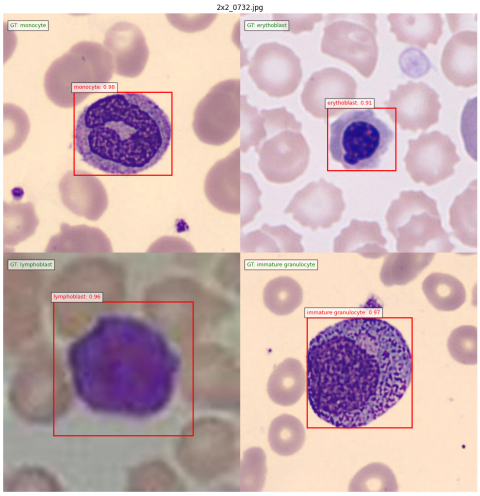
Figure 18: Model performance on Week 7 validation set: (a) confusion matrix, (b) precision–recall curve.



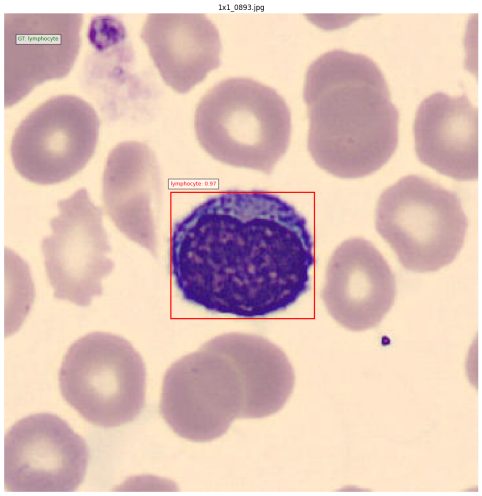
(a) 4×4 Grid



(b) 3×3 Grid



(c) 2×2 Grid



(d) 1×1 Grid

Figure 19: Detection examples on composite images: (a) 4×4 grid, (b) 3×3 grid, (c) 2×2 grid, (d) 1×1 grid.

7.4.4 Week 7 Conclusion

- The YOLO11 model achieved high accuracy across all composite grid configurations, confirming its robustness to varied cell densities.
- Each grid size dataset comprised 2,000 images generated via sampling with replacement.
- **Train/Validation Contamination**: We discovered that sampling without enforcing split boundaries caused overlapping images between training and validation sets, risking inflated performance estimates.

7.4.5 Week 8: Addressing Data Leakage and Model Refinement

Building on the learnings from Week 7, Week 8 focused on correcting the data leakage issue and further refining the model's performance.

7.4.6 Task Overview

- Generate composite images of sizes 1×1, 2×2, 3×3, and 4×4 from the original dataset.
- Split the original dataset into training (80 %), validation (15 %), and testing (5 %) subsets *before* composite-image creation.
- Sample with replacement within each subset to produce 2000 images per grid size (1600 train, 400 val).
- Train the YOLO11m detector on the leak-free composites and evaluate on held-out validation and test sets.

7.4.7 Data Leakage Correction Strategy

To eliminate the data leakage identified in Week 7, we implemented:

- A strict **pre-split** of the source images into train/val/test partitions (80/15/5).
- Composite images generated *only* from their respective partitions.
- Maintained the same grid sizes (1×1–4×4) and total counts (2000 per size).

7.4.8 Preprocessing Background

Before generating the composite, we perform the following preprocessing steps on the background:

1. Select a single-class background image.
2. Remove the existing cell from that image using Adobe Firefly [15].
3. Create a 1200 × 1200 px canvas by tiling (duplicating) the cleaned background.
4. Randomly paste resized cell images from the test set onto the canvas.

7.4.9 Random Composite Generation Strategy

For more realistic test scenarios, we also implement a random composite generation strategy that places cell images with varied sizes, orientations, and positions on a background canvas.

Random Composite Algorithm The procedure for creating random composite images is outlined in Algorithm 2.

Algorithm 2 Random Composite Image Generation

for *each cell image in shuffled dataset* **do**
  Load image and its class labels **for** *scale attempt = 1 to 3* **do**
    Sample scale $s \in [0.15, 0.4]$ and aspect ratio $a \in \{0.8, 1.0, 1.2\}$
    $w \leftarrow \max(50, \lfloor W \cdot s \rfloor)$
    $h \leftarrow \max(50, \lfloor w \cdot a \cdot \frac{h_{\text{orig}}}{w_{\text{orig}}} \rfloor)$
    Resize image to (w, h)
    Optionally rotate by 90° (swap w, h if rotated)
    for *attempt = 1 to max_ attempts* **do**
      Generate random position (x, y) with $x \in [0, W - w]$, $y \in [0, H - h]$
      if *no collision with existing images* **then**
        Place image at (x, y) on canvas
        Record class labels and position (x, y, w, h)
        Break (placement successful)
    if *placement successful* **then**
      Break
    else
      Reduce max scale by 20% and try again
Return composite image, all labels, image positions

Collision Detection A critical part of the random composite generation is ensuring that placed images don't overlap (see Figure 20). Two images with positions (x_1, y_1, w_1, h_1) and (x_2, y_2, w_2, h_2) collide if and only if:

$$(x_1 < x_2 + w_2) \wedge (x_1 + w_1 > x_2) \wedge (y_1 < y_2 + h_2) \wedge (y_1 + h_1 > y_2) \tag{2}$$

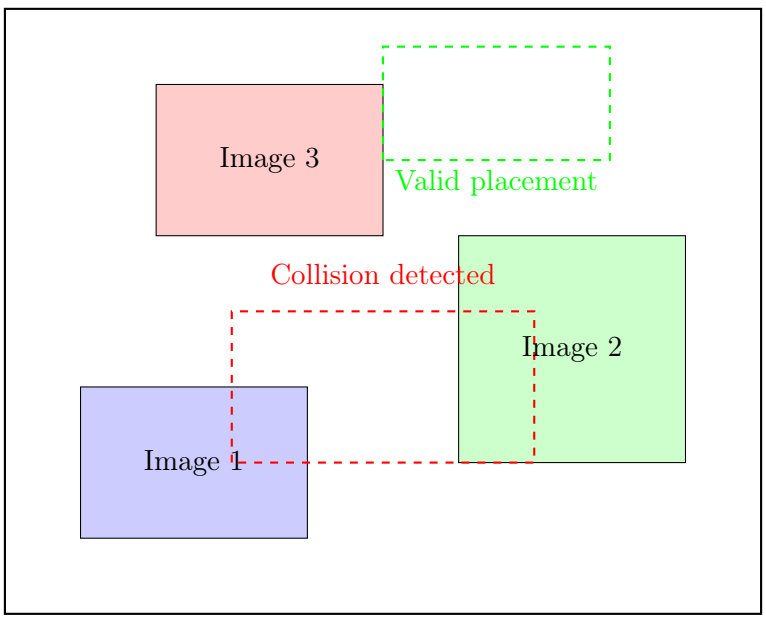
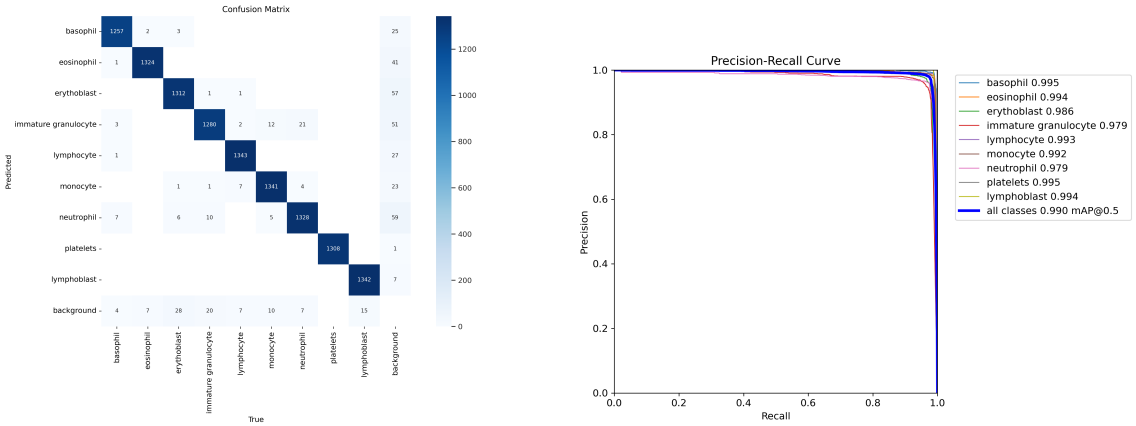


Figure 20: Collision detection during random image placement. The dashed red rectangle shows a rejected placement that overlaps with existing images. The dashed green rectangle shows a valid placement with no collisions.

7.4.10 Week 8 Model Performance

Figures 21a and 21b show the confusion matrix and precision–recall curve for the validation set after correcting for data leakage.



(a) Confusion Matrix (b) Precision–Recall Curve

Figure 21: Validation performance metrics in Week 8 after data leakage correction: (a) confusion matrix, (b) precision–recall curve.

Figure 22 shows the training and validation loss curves, demonstrating stable convergence post–leakage fix.

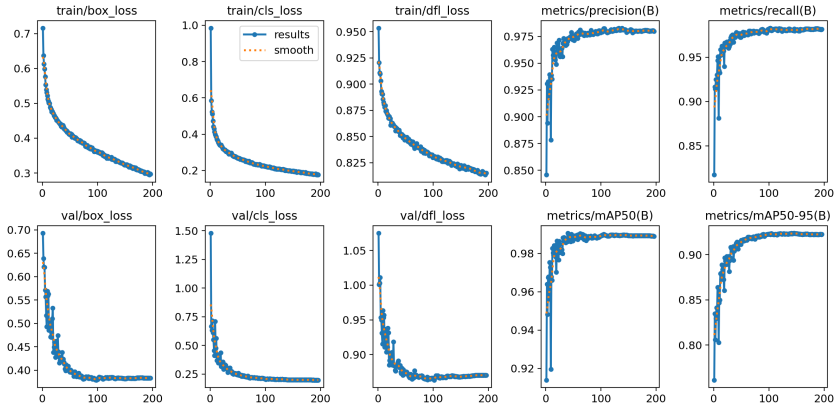


Figure 22: Training and validation loss curves from Week 8 showing model convergence and stability.

Figure 23 presents example detections on a composite image, confirming robust performance.

7.4.11 Week 8 Conclusion

- The data-leakage correction yielded more reliable validation metrics without sacrificing overall performance.
- The pre-split protocol now ensures true generalization, as evidenced by consistent train/-validation/test results.
- The current prediction pipeline places the entire cell image (rather than only the segmented class region) onto a random background canvas, which may introduce background-specific biases into the detection results.

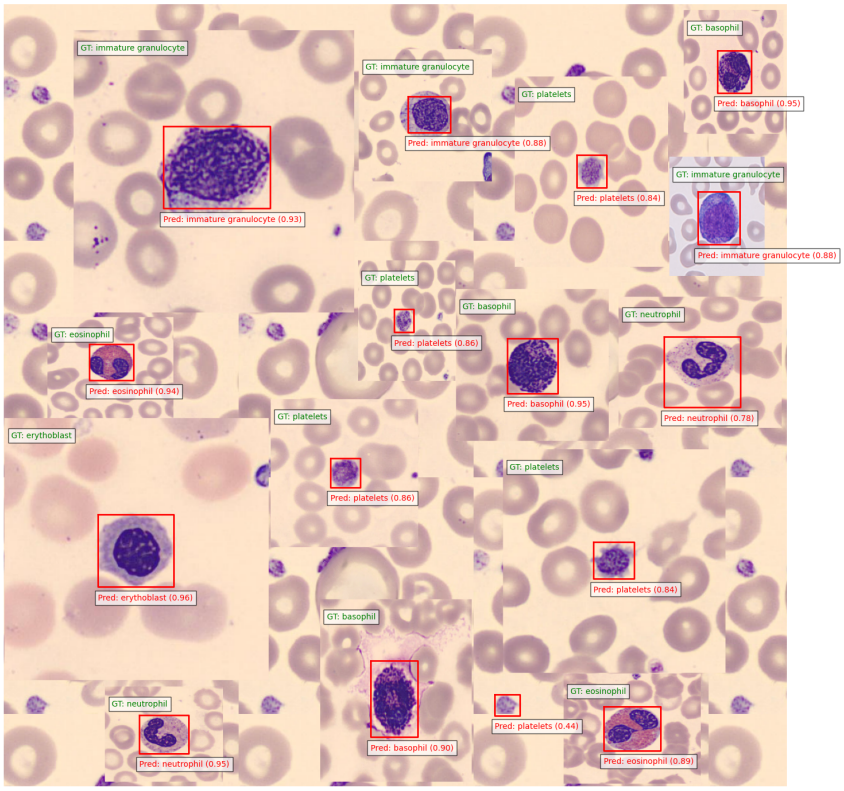


Figure 23: Detection performance on a composite image in Week 8.

7.5 Week 9–10: Cell Segmentation and Classification Refinement

The focus during Weeks 9 and 10 shifted to advanced segmentation techniques for all nine blood cell classes and improving the detection model to handle mixed-magnification scenarios.

7.5.1 Task Overview

The primary objectives for this phase were:

- Developing robust segmentation pipelines for all nine classes of blood cells using specialized image processing techniques
- Creating a synthetic dataset by randomly placing segmented cells on background images
- Testing and refining the detection model with this new approach

7.5.2 Segmentation Methodology

We implemented tailored pipelines for each of the nine cell classes. Each class follows the general steps of color-space conversion, thresholding, morphological cleanup, contour extraction, and finally cropping and saving the segmented mask. Below, we detail the approach for Basophils as an example.

Basophil Segmentation Basophils exhibit distinctive purple-stained granules which make them identifiable through color-based segmentation. Our pipeline isolates the largest purple cell component using HSV color thresholding, morphological operations, and contour analysis (see Algorithm 3).

Algorithm 3 Basophil Cell Segmentation

Convert I_{BGR} to HSV: $I_{\text{HSV}} \leftarrow \text{cvtColor}(I_{\text{BGR}}, \text{BGR2HSV})$
Define purple-stain bounds: lower$=(120, 50, 50)$, upper$=(160, 255, 255)$
$M \leftarrow \text{inRange}(I_{\text{HSV}}, \text{lower}, \text{upper})$
$M_{\text{clean}} \leftarrow \text{morphologyEx}(M, \text{MORPH\_CLOSE}, 5 \times 5 \text{ kernel})$
if *no contours found in* M_{clean} **then**
  return null
Find contours $\{C_i\} \leftarrow \text{findContours}(M_{\text{clean}}, \text{EXTERNAL})$
Select largest $C^* \leftarrow \arg\max_{C_i} \text{area}(C_i)$
$M_{\text{largest}} \leftarrow \text{zeros\_like}(M)$
$M_{\text{largest}} \leftarrow \text{drawContours}(M_{\text{largest}}, \{C^*\}, -1, 255, \text{FILLED})$
$(x, y, w, h) \leftarrow \text{boundingRect}(C^*)$
Initialize $S \leftarrow \text{zeros}(h, w, 4)$
$M_{\text{crop}} \leftarrow M_{\text{largest}}[y : y + h,\ x : x + w]$
for *each pixel* (u, v) *with* $M_{\text{crop}}(u, v) = 255$ **do**
  $S(u, v, 0 : 3) \leftarrow I_{\text{BGR}}(y + u, x + v, 0 : 3)$
  $S(u, v, 3) \leftarrow 255$
return S

The algorithm can be understood through the following key operations:

1. **HSV Transformation and Thresholding:** Converts the image to HSV color space and isolates purple components using the formula:

$$M(i,j) = \begin{cases} 255, & \text{if } 120 \le H(i,j) \le 160 \text{ and } 50 \le S(i,j) \le 255 \text{ and } 50 \le V(i,j) \le 255 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

2. **Morphological Cleaning:** Applies closing operation (dilation followed by erosion) to fill small holes and remove noise:

$$M_{\text{clean}} = (M \oplus K) \ominus K \tag{4}$$

3. **Contour Processing:** Extracts the largest contour C^* from the mask:

$$C^* = \arg \max_{C_i \in \{C_i\}} \text{Area}(C_i) \tag{5}$$

4. **Transparent Extraction:** Creates a tight bounding box around the cell and copies only the cell pixels to a transparent background:

$$S(u,v,c) = \begin{cases} I_{\text{BGR}}(y+u, x+v, c), & \text{if mask at } (y+u, x+v) \text{ is set and } c \in \{0,1,2\} \\ 255, & \text{if mask at } (y+u, x+v) \text{ is set and } c = 3 \text{ (alpha)} \\ 0, & \text{otherwise (transparent)} \end{cases} \tag{6}$$

The batch processing system applies this algorithm across all basophil images, creating a collection of segmented cells with transparent backgrounds. This approach preserves the cellular morphology while eliminating background noise, producing clean inputs for synthetic dataset generation. Success and failure metrics are tracked throughout the process to maintain quality control.

Eosinophil, Erythroblast & Lymphocyte Segmentation Extending the Basophil method (Algorithm 3), Algorithms 4 and 5 apply the same core steps—LAB-space CLAHE, Otsu thresholding [16], morphological cleanup, central-contour selection, and watershed refinement—but adapted to the varied staining and morphology of eosinophils, erythroblasts, and lymphocytes. The final mask is closed, hole-filled, and extracted as an RGBA crop, ensuring consistency across all four WBC classes.

Algorithm 4 White Blood Cell Segmentation (Part 1)

Input: Microscopy image I containing white blood cells
Output: Segmented cell with transparent background S

$I_{\text{RGB}} \leftarrow \text{cvtColor}(I, \text{BGR2RGB})$
`// Color space enhancement`
$I_{\text{LAB}} \leftarrow \text{cvtColor}(I_{\text{RGB}}, \text{RGB2LAB})$
$L, A, B \leftarrow \text{split}(I_{\text{LAB}})$
$L_{\text{enhanced}} \leftarrow \text{CLAHE}(L, \text{clipLimit} = 3.0, \text{tileSize} = (8, 8))$
$I_{\text{LAB}}^{\text{enhanced}} \leftarrow \text{merge}(L_{\text{enhanced}}, A, B)$
$I_{\text{RGB}}^{\text{enhanced}} \leftarrow \text{cvtColor}(I_{\text{LAB}}^{\text{enhanced}}, \text{LAB2RGB})$
`// Grayscale conversion and preprocessing`
$G \leftarrow \text{cvtColor}(I_{\text{RGB}}^{\text{enhanced}}, \text{RGB2GRAY})$
$G_{\text{blur}} \leftarrow \text{GaussianBlur}(G, \text{ksize} = (5, 5), \sigma = 0)$
$T \leftarrow \text{threshold}(G_{\text{blur}}, 0, 255, \text{THRESH\_BINARY\_INV} + \text{THRESH\_OTSU})$
`// Morphological cleaning`
$K_{\text{open}} \leftarrow \text{ones}(3 \times 3)$
$T_{\text{open}} \leftarrow \text{morphologyEx}(T, \text{MORPH\_OPEN}, K_{\text{open}}, \text{iterations} = 2)$
`// Contour detection`
$\{C_i\} \leftarrow \text{findContours}(T_{\text{open}}, \text{EXTERNAL})$
$M_{\text{initial}} \leftarrow \text{zeros\_like}(T_{\text{open}})$
`// Identify central cell contour`
$(c_x, c_y) \leftarrow (I_{\text{width}}/2, I_{\text{height}}/2)$
$C^* \leftarrow \text{null}$
$d_{\min} \leftarrow \infty$
for *each contour* C_i *in* $\{C_i\}$ **do**
  $M_i \leftarrow \text{moments}(C_i)$
  if $M_i[\text{"m00"}] \neq 0$ **then**
    $(cx_i, cy_i) \leftarrow (M_i[\text{"m10"}]/M_i[\text{"m00"}], M_i[\text{"m01"}]/M_i[\text{"m00"}])$
    $d_i \leftarrow \sqrt{(cx_i - c_x)^2 + (cy_i - c_y)^2}$
    $a_i \leftarrow \text{contourArea}(C_i)$
    if $a_i > 500$ ***and*** $d_i < d_{\min}$ **then**
      $d_{\min} \leftarrow d_i$
      $C^* \leftarrow C_i$
if C^* *is* ***not*** *null* **then**
  $\text{fillPoly}(M_{\text{initial}}, \{C^*\}, 255)$
`// Watershed segmentation`
$(L, M_{\text{markers}}) \leftarrow \text{connectedComponents}(M_{\text{initial}})$
$M_{\text{markers}} \leftarrow M_{\text{markers}} + 1$
$M_{\text{markers}}[T = 0] \leftarrow 0$
$M_{\text{markers}} \leftarrow \text{watershed}(I_{\text{RGB}}, M_{\text{markers}})$

Algorithm 5 White Blood Cell Segmentation (Part 2)

```
// Create final segmentation mask
M_final ← zeros_like(G)
M_final[M_markers = 2] ← 255
// Clean up final mask with morphological operations
K_close ← ones(5 × 5)
M_final ← morphologyEx(M_final, MORPH_CLOSE, K_close, iterations = 3)
// Fill holes using contour filling
{C_j} ← findContours(M_final, EXTERNAL)
M_filled ← zeros_like(M_final)
for each contour C_j in {C_j} do
    fillPoly(M_filled, {C_j}, 255)
M_final ← M_filled
// Create transparent output
C_largest ← arg max_{C_j ∈ {C_j}} area(C_j)
(x, y, w, h) ← boundingRect(C_largest)
S ← zeros(h, w, 4)
M_crop ← M_final[y : y + h, x : x + w]
for each channel c in {0, 1, 2} do
    S(:, :, c)[M_crop = 255] ← I_RGB[y : y + h, x : x + w][M_crop = 255, c]
S(:, :, 3) ← M_crop
return S
```

The algorithm progresses through several key stages:

1. **Color Space Enhancement:** The algorithm first enhances the contrast of the cell structures using LAB color space and CLAHE (Contrast Limited Adaptive Histogram Equalization):

$$L_{\text{enhanced}} = \text{CLAHE}(L, \text{clipLimit} = 3.0, \text{tileSize} = (8, 8)) \tag{7}$$

2. **Adaptive Thresholding:** Otsu's method automatically determines the optimal threshold θ to separate cells from background:

$$T(x, y) = \begin{cases} 255, & \text{if } G_{\text{blur}}(x, y) < \theta_{\text{Otsu}} \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

3. **Central Cell Identification:** The algorithm selects the cell closest to the image center while filtering out small artifacts:

$$C^* = \arg \min_{C_i \in \{C_i\}} \left(\sqrt{(cx_i - c_x)^2 + (cy_i - c_y)^2} \right) \quad \text{where} \quad \text{area}(C_i) > 500 \tag{9}$$

where (cx_i, cy_i) is calculated from moments:

$$cx_i = \frac{M_i^{10}}{M_i^{00}}, \quad cy_i = \frac{M_i^{01}}{M_i^{00}} \tag{10}$$

4. **Watershed Segmentation:** For improved boundary detection, a watershed transform is applied:

$$M_{\text{markers}} = \text{watershed}(I_{\text{RGB}}, M_{\text{markers}} + 1) \tag{11}$$

5. **Hole Filling:** Morphological closing and contour-based filling techniques are combined to create a complete mask:

$$M_{\text{final}} = (M_{\text{markers}} \oplus K_{\text{close}}) \ominus K_{\text{close}} \tag{12}$$

6. **Transparent Extraction:** The segmented cell is isolated with a transparent background:

$$S(u, v, c) = \begin{cases} I_{\text{RGB}}(y + u, x + v, c), & \text{if } M_{\text{crop}}(u, v) = 255 \text{ and } c \in \{0, 1, 2\} \text{ (RGB)} \\ 255, & \text{if } M_{\text{crop}}(u, v) = 255 \text{ and } c = 3 \text{ (alpha)} \\ 0, & \text{otherwise (transparent)} \end{cases} \tag{13}$$

This segmentation approach effectively isolates white blood cells including eosinophils, erythroblasts, and lymphocytes from microscopy images. The multi-stage pipeline enhances cell visibility, locates the central cell, refines boundaries using watershed segmentation, and produces clean cell extractions with transparent backgrounds for downstream analysis or dataset generation.

The batch processing system applies this algorithm systematically across multiple images, tracking success and failure rates to ensure quality control. For each successfully segmented cell, the algorithm extracts and saves a PNG file with transparency, preserving only the cell of interest while removing background elements that could introduce noise in training datasets.

Immature Granulocyte Segmentation Immature granulocytes display distinctive purple/blue coloration due to their higher RNA content and less condensed chromatin. Our pipeline identifies these cells through color-based HSV thresholding, morphological operations, and watershed segmentation (see Algorithm 6 and 7).

Algorithm 6 Immature Granulocyte Segmentation (Part 1)

Input: Microscopy image I containing immature granulocytes
Output: Segmented cell with transparent background S
$I_{\text{RGB}} \leftarrow \text{cvtColor}(I, \text{BGR2RGB})$
`// Color space conversion for better separation`
$I_{\text{HSV}} \leftarrow \text{cvtColor}(I_{\text{RGB}}, \text{RGB2HSV})$
`// Define range for purple/blue color (immature granulocyte)`
$\text{lower\_purple} \leftarrow (120, 30, 100)$
$\text{upper\_purple} \leftarrow (170, 255, 255)$
`// Create mask for immature granulocyte regions`
$M_{\text{purple}} \leftarrow \text{inRange}(I_{\text{HSV}}, \text{lower\_purple}, \text{upper\_purple})$
`// Apply morphological operations`
$K_{\text{open}} \leftarrow \text{ones}(3 \times 3)$
$M_{\text{open}} \leftarrow \text{morphologyEx}(M_{\text{purple}}, \text{MORPH\_OPEN}, K_{\text{open}}, \text{iterations} = 1)$
$M_{\text{dilated}} \leftarrow \text{dilate}(M_{\text{open}}, K_{\text{open}}, \text{iterations} = 3)$
`// Find contours in the mask`
$\{C_i\} \leftarrow \text{findContours}(M_{\text{dilated}}, \text{RETR\_EXTERNAL}, \text{CHAIN\_APPROX\_SIMPLE})$
`// Filter contours by size`
$M_{\text{initial}} \leftarrow \text{zeros\_like}(M_{\text{dilated}})$
$A_{\text{img}} \leftarrow I_{\text{width}} \times I_{\text{height}}$
$\{C_i^{\text{valid}}\} \leftarrow \{C_i \mid \text{contourArea}(C_i) > 0.01 \times A_{\text{img}}\}$
`// Select largest valid contour`
if $\{C_i^{\text{valid}}\}$ *is not empty* **then**
  $C^* \leftarrow \arg\max_{C_i \in \{C_i^{\text{valid}}\}} \text{contourArea}(C_i)$
  $\text{drawContours}(M_{\text{initial}}, \{C^*\}, -1, 255, \text{FILLED})$

Algorithm 7 Immature Granulocyte Segmentation (Part 2)

```
// Refine segmentation using GrabCut
if ∑ M_initial > 0 then
    M_grabcut ← zeros(I_height, I_width, uint8)
    M_grabcut[:] ← GC_PR_BGD
    M_grabcut[M_initial > 0] ← GC_PR_FGD
    (x, y, w, h) ← boundingRect(M_initial)
    rect ← (x, y, w, h)
    bgdModel ← zeros(1, 65, float64)
    fgdModel ← zeros(1, 65, float64)
    grabCut(I_RGB, M_grabcut, rect, bgdModel, fgdModel, 5, GC_INIT_WITH_MASK)
    M_refined ← (M_grabcut == GC_FGD) ∨ (M_grabcut == GC_PR_FGD)
    M_refined ← M_refined × 255
else
    M_refined ← M_initial
// Final cleanup
K_close ← ones(5 × 5)
M_final ← morphologyEx(M_refined, MORPH_CLOSE, K_close, iterations = 2)
// Keep only largest connected component
(L_num, L_labels, L_stats, L_centroids) ← connectedComponentsWithStats(M_final, connectivity = 8)
if L_num > 1 then
    L_largest ← 1 + arg max_{i∈{1,...,L_num−1}} L_stats[i, CC_STAT_AREA]
    M_final ← zeros_like(M_final)
    M_final[L_labels == L_largest] ← 255
// Create transparent output
{C_j} ← findContours(M_final, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE)
if {C_j} is not empty then
    C_largest ← arg max_{C_j∈{C_j}} contourArea(C_j)
    (x, y, w, h) ← boundingRect(C_largest)
    S ← zeros(h, w, 4, uint8)
    M_crop ← M_final[y : y + h, x : x + w]
    for c ∈ {0, 1, 2} do
        S[:, :, c][M_crop == 255] ← I_RGB[y : y + h, x : x + w][M_crop == 255, c]
    S[:, :, 3] ← M_crop
return S
```

The algorithm can be understood through the following key operations:

1. **HSV Transformation and Thresholding:** Converts the image to HSV color space and isolates purple/blue components using the formula:

$$M_{\text{purple}}(i,j) = \begin{cases} 255, & \text{if } 120 \le H(i,j) \le 170 \text{ and } 30 \le S(i,j) \le 255 \text{ and } 100 \le V(i,j) \le 255 \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

2. **Morphological Processing:** Applies opening followed by dilation to remove noise and

enhance cell regions:

$$M_{\text{open}} = M_{\text{purple}} \circ K_{\text{open}} = (M_{\text{purple}} \ominus K_{\text{open}}) \oplus K_{\text{open}} \tag{15}$$

$$M_{\text{dilated}} = M_{\text{open}} \oplus K_{\text{open}} \oplus K_{\text{open}} \oplus K_{\text{open}} \tag{16}$$

3. **Contour Selection:** Identifies the largest valid contour representing an immature granulocyte:

$$C^* = \arg \max_{C_i \in \{C_i^{\text{valid}}\}} \text{Area}(C_i) \quad \text{where} \quad C_i^{\text{valid}} = \{C_i \mid \text{Area}(C_i) > 0.01 \times A_{\text{img}}\} \tag{17}$$

4. **GrabCut Refinement:** Improves segmentation accuracy by applying the GrabCut [17] algorithm which iteratively estimates foreground and background models:

$$M_{\text{refined}}(i,j) = \begin{cases} 255, & \text{if } M_{\text{grabcut}}(i,j) \in \{\text{GC\_FGD}, \text{GC\_PR\_FGD}\} \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

5. **Connected Component Analysis:** Ensures only the largest cell component is retained:

$$L_{\text{largest}} = 1 + \arg \max_{i \in \{1,\ldots,L_{\text{num}}-1\}} L_{\text{stats}}[i, \text{CC\_STAT\_AREA}] \tag{19}$$

6. **Transparent Extraction and Batch Processing:** We apply the same tight RGBA cropping procedure as above (Eq. 13) to each detected cell. Finally, our pipeline processes all immature granulocyte images in batch, producing transparent-background cell crops and automatically evaluating each segmentation via Dice, IoU, or mask-coverage metrics.

Lymphoblast Segmentation Similar to basophils, lymphoblasts exhibit purple staining but require a different detection approach. While the basophil algorithm uses HSV color space thresholding, our lymphoblast segmentation leverages direct RGB channel relationships. The core difference lies in the initial purple detection method (see Algorithm 8).

Algorithm 8 Lymphoblast Cell Segmentation

Extract RGB channels: $R, G, B \leftarrow \text{split}(I_{\text{RGB}})$
Create purple detector: $M_{\text{purple}} \leftarrow (B > (G + 30)) \wedge (B > R) \wedge (B > 50)$
Apply morphological operations: $M_{\text{clean}} \leftarrow \text{Close}(\text{Open}(M_{\text{purple}}))$
Select largest contour: $C^* \leftarrow \arg\max_{C_i} \text{area}(C_i)$
Extract segmented cell with transparency (as in basophil algorithm)

The key distinction is our RGB-based purple detection formula:

$$M_{\text{purple}}(i,j) = \begin{cases} 255, & \text{if } B(i,j) > G(i,j) + 30 \text{ and } B(i,j) > R(i,j) \text{ and } B(i,j) > 50 \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

This RGB approach is more direct than the HSV method used for basophils, as lymphoblasts typically show a stronger and more consistent blue channel dominance. The remainder of the pipeline — morphological cleanup, contour selection, and transparent background creation — follows the same structure as the basophil algorithm but with tailored parameters. The RGB-based detection proves particularly effective for lymphoblasts because:

- It captures the characteristic $B > R > G$ channel relationship in stained lymphoblasts
- The absolute threshold ($B > 50$) eliminates dark background regions
- The margin requirement ($B > G + 30$) ensures significant color differentiation

This method achieves robust lymphoblast segmentation without requiring conversion to HSV space, making it computationally more efficient while maintaining high accuracy for standard stained microscopy images.

Monocyte Segmentation Monocytes exhibit a purple-blue nucleus surrounded by a pale cytoplasm. Our monocyte pipeline largely parallels the immature granulocyte algorithm—HSV thresholding, morphological cleanup, GrabCut refinement, connected-component filtering, and RGBA cropping—but with two key adjustments to capture the larger cytoplasmic halo and fill nuclear holes.

Algorithm 9 Monocyte Segmentation

Input: Microscopy image I containing monocytes
Output: RGBA-cropped monocyte cell S
$I_{\text{RGB}} \leftarrow \text{cvtColor}(I, \text{BGR2RGB})$
$I_{\text{HSV}} \leftarrow \text{cvtColor}(I_{\text{RGB}}, \text{RGB2HSV})$
$\text{lower\_purple} \leftarrow (120, 30, 100), \quad \text{upper\_purple} \leftarrow (170, 255, 255)$
$M_{\text{purple}} \leftarrow \text{inRange}(I_{\text{HSV}}, \text{lower\_purple}, \text{upper\_purple})$
$K_3 \leftarrow \mathbf{1}_{3\times3}$
$M_{\text{open}} \leftarrow \text{morphologyEx}(M_{\text{purple}}, \text{MORPH\_OPEN}, K_3)$
`// three-fold dilation to include cytoplasm`
$M_{\text{dilated}} \leftarrow M_{\text{open}} \oplus K_3 \oplus K_3 \oplus K_3$
$\{C_i\} \leftarrow \text{findContours}(M_{\text{dilated}}, \dots)$
retain C_i with $\text{Area}(C_i) > 0.01 \times A_{\text{img}}$
draw largest contour into M_{initial}
if $\sum M_{\text{initial}} > 0$ **then**
  initialize GrabCut mask from M_{initial} and bounding rect
  run $\text{grabCut}(I_{\text{RGB}}, \dots, \text{GC\_INIT\_WITH\_MASK})$
  $M_{\text{refined}} \leftarrow (M_{\text{grabcut}} \in \{\text{GC\_FGD}, \text{GC\_PR\_FGD}\}) \times 255$
else
  $M_{\text{refined}} \leftarrow M_{\text{initial}}$
`// close small nuclear holes`
$K_5 \leftarrow \mathbf{1}_{5\times5}$
$M_{\text{closed}} \leftarrow \text{morphologyEx}(M_{\text{refined}}, \text{MORPH\_CLOSE}, K_5, \text{iterations} = 2)$
$(n, L, \text{stats}, \dots) \leftarrow \text{connectedComponentsWithStats}(M_{\text{closed}}, 8)$
if $n > 1$ **then**
  $i^* \leftarrow 1 + \arg\max_{i \in [1, n-1]} \text{stats}[i, \text{AREA}]$
  $M_{\text{final}} \leftarrow (L == i^*) \times 255$
else
  $M_{\text{final}} \leftarrow M_{\text{closed}}$
`// same tight crop with alpha as before`
return tight RGBA crop of I_{RGB} using mask M_{final}

e

Key Differences with Immature Granulocytes:

- *Expanded Cytoplasm:* Three sequential dilations broaden the mask to include the pale cytoplasm around the nucleus.
- *Hole Filling:* A closing step with a 5×5 kernel fills small holes inside the segmented nucleus.

Our batch engine then applies Algorithm 9 across all monocyte images, producing clean RGBA cell crops ready for downstream analysis or model training.

Neutrophil Segmentation Neutrophils contain multi-lobed nuclei and light pink cytoplasm when stained with Wright/Giemsa. Our algorithm combines stain deconvolution, nucleus segmentation, and GrabCut refinement to isolate these cells (see Algorithm 10).

Algorithm 10 Neutrophil Cell Segmentation

Convert I_{BGR} to RGB: $I_{\text{RGB}} \leftarrow \text{cvtColor}(I_{\text{BGR}}, \text{BGR2RGB})$
Apply stain deconvolution to separate components:
$I_{\text{float}} \leftarrow I_{\text{RGB}}/255$
$\text{OD} \leftarrow -\log_{10}(I_{\text{float}} + \epsilon)$
$S \leftarrow \begin{bmatrix} 0.18 & 0.80 & 0.65 \\ 0.25 & 0.58 & -0.75 \\ 0.90 & -0.35 & 0.05 \end{bmatrix}$
$\text{Separated} \leftarrow \text{OD} \cdot S^{-1}$
Extract nucleus channel and enhance: $N \leftarrow \text{Separated}[..., 0]$
$N_{\text{enhanced}} \leftarrow \text{equalize\_adapthist}(N) \cdot 255$
Apply Otsu thresholding: $N_{\text{mask}} \leftarrow \text{threshold}(\text{GaussianBlur}(N_{\text{enhanced}}, (5, 5)), \text{OTSU})$
Clean nucleus mask: $N_{\text{clean}} \leftarrow \text{morphologyEx}(N_{\text{mask}}, \text{MORPH\_CLOSE}, 3 \times 3 \text{ kernel})$
Remove small objects: $N_{\text{clean}} \leftarrow \text{remove\_small\_objects}(N_{\text{clean}}, \text{min\_size} = 50)$
Find contours $\{C_i\} \leftarrow \text{findContours}(N_{\text{clean}}, \text{EXTERNAL})$
if *no contours found* **then**
  return null
Select largest contour $C^* \leftarrow \arg\max_{C_i} \text{area}(C_i)$
Create nucleus mask $M_{\text{nucleus}} \leftarrow \text{zeros\_like}(N_{\text{mask}})$
$M_{\text{nucleus}} \leftarrow \text{drawContours}(M_{\text{nucleus}}, \{C^*\}, 0, 255, \text{FILLED})$
Dilate to estimate cytoplasm: $M_{\text{cyto}} \leftarrow \text{dilate}(M_{\text{nucleus}}, 15 \times 15 \text{ kernel}, \text{iterations} = 2)$
Extract cytoplasm channel: $C \leftarrow \text{Separated}[..., 1] \cdot 255$
Initialize GrabCut mask: $G \leftarrow \text{GC\_BGD}$ everywhere
$G[M_{\text{cyto}} > 0] \leftarrow \text{GC\_PR\_FGD}$
$G[M_{\text{nucleus}} > 0] \leftarrow \text{GC\_FGD}$
$(x, y, w, h) \leftarrow \text{boundingRect}(C^*)$ with margin $= 20$
Apply GrabCut: $\text{grabCut}(I_{\text{RGB}}, G, (x, y, w, h), \text{iterations} = 5)$
Create segmentation mask: $M_{\text{grab}} \leftarrow 255$ where $G \in \{\text{GC\_FGD}, \text{GC\_PR\_FGD}\}$, 0 otherwise
Combine masks: $M_{\text{combined}} \leftarrow M_{\text{grab}} \vee M_{\text{nucleus}}$
Post-process: $M_{\text{final}} \leftarrow \text{morphologyEx}(M_{\text{combined}}, \text{MORPH\_CLOSE}, 7 \times 7 \text{ kernel}, \text{iterations} = 2)$
Fill holes: $M_{\text{final}} \leftarrow \text{binary\_fill\_holes}(M_{\text{final}})$
Create transparent output: $S \leftarrow \text{zeros}(h, w, 4)$
for *each pixel* (u, v) *with* $M_{\text{final}}[y : y + h, x : x + w](u, v) = 255$ **do**
  $S(u, v, 0 : 3) \leftarrow I_{\text{BGR}}[y : y + h, x : x + w](u, v, 0 : 3)$
  $S(u, v, 3) \leftarrow 255$
return S

The algorithm can be understood through the following key operations:

1. **Stain Deconvolution:** Separates stain components using optical density transformation and stain matrix inversion:

$$\text{OD}(i,j,c) = -\log_{10}(I_{\text{float}}(i,j,c) + \epsilon) \tag{21}$$

$$\text{Separated} = \text{OD} \cdot S^{-1} \tag{22}$$

2. **Nucleus Extraction:** Isolates the nucleus using adaptive histogram equalization and Otsu thresholding:

$$N_{\text{mask}}(i,j) = \begin{cases} 255, & \text{if } N_{\text{enhanced}}(i,j) > T_{\text{Otsu}} \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

3. **Cytoplasm Estimation:** Creates an initial cytoplasm estimate through morphological dilation:

$$M_{\text{cyto}} = M_{\text{nucleus}} \oplus K_{15\times15} \tag{24}$$

4. **GrabCut Refinement:** Applies GrabCut segmentation using a three-region initialization:

$$G(i,j) = \begin{cases} \text{GC\_FGD}, & \text{if } M_{\text{nucleus}}(i,j) > 0 \\ \text{GC\_PR\_FGD}, & \text{if } M_{\text{cyto}}(i,j) > 0 \\ \text{GC\_BGD}, & \text{otherwise} \end{cases} \tag{25}$$

5. **Post-processing:** Applies morphological operations to refine the final segmentation:

$$M_{\text{final}} = \text{Fill}((M_{\text{grab}} \vee M_{\text{nucleus}}) \bullet K_{7\times7}) \tag{26}$$

where $\bullet$ denotes the closing operation.

The segmentation pipeline preserves the characteristic multi-lobed nucleus and granular cytoplasm of neutrophils while removing background artifacts. The stain deconvolution approach makes the algorithm robust to variations in staining intensity and color, while the GrabCut refinement accurately delineates the often irregular cell boundaries. Success rates of approximately 90% demonstrate the effectiveness of this approach for neutrophil extraction from Wright/Giemsa-stained blood smears.

Platelet Segmentation Platelets are small, purple-stained cell fragments whose fine granularity requires a tighter HSV range and smaller morphological kernels than basophils. Aside from these parameter changes, the pipeline mirrors basophil segmentation (Algorithm 11).

Notes on Similarity:

- Follows the same HSV-threshold → clean → largest-contour → RGBA-crop workflow as basophils.
- Uses inRange, morphologyEx, findContours, drawContours, and tight bounding-box extraction identically.

Key Parameter Differences:

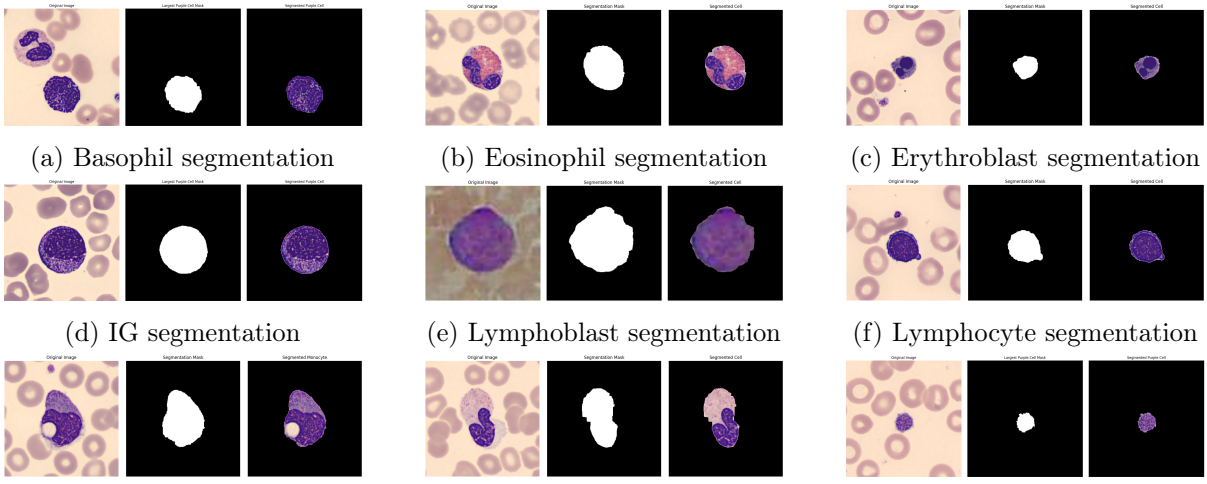
- *HSV Bounds:* shifted to $(130,40,40)-(170,255,255)$ for finer platelet pink.
- *Kernel Size:* 3×3 closing kernel to preserve small platelet structure.

Algorithm 11 Platelet Cell Segmentation

Input: BGR image I_{BGR}
Output: RGBA-cropped platelet S
$I_{\text{HSV}} \leftarrow \text{cvtColor}(I_{\text{BGR}}, \text{BGR2HSV})$
$\text{lower} \leftarrow (130, 40, 40), \quad \text{upper} \leftarrow (170, 255, 255)$
$M \leftarrow \text{inRange}(I_{\text{HSV}}, \text{lower}, \text{upper})$
$K_3 \leftarrow \mathbf{1}_{3\times 3}$
$M_{\text{clean}} \leftarrow \text{morphologyEx}(M, \text{MORPH\_CLOSE}, K_3)$
$\{C_i\} \leftarrow \text{findContours}(M_{\text{clean}}, \text{RETR\_EXTERNAL})$
if $\{C_i\} = \emptyset$ **then**
  return null
$C^* \leftarrow \arg\max_{C_i} \text{Area}(C_i)$
$M_{\text{largest}} \leftarrow \text{zeros\_like}(M)$
$\text{drawContours}(M_{\text{largest}}, \{C^*\}, -1, 255, \text{FILLED})$
$(x, y, w, h) \leftarrow \text{boundingRect}(C^*)$
$S \leftarrow \text{zeros}(h, w, 4)$
$M_{\text{crop}} \leftarrow M_{\text{largest}}[y : y + h,\ x : x + w]$
for *each* (u, v) *with* $M_{\text{crop}}(u, v) = 255$ **do**
  $S(u, v, 0 : 2) \leftarrow I_{\text{BGR}}(y + u, x + v, 0 : 2)$
  $S(u, v, 3) \leftarrow 255$
return S

7.5.3 Segmentation Results

Figure 24 presents the segmentation outputs for all nine blood cell classes, with each subfigure corresponding to a specific class.



(a) Basophil segmentation (b) Eosinophil segmentation (c) Erythroblast segmentation

(d) IG segmentation (e) Lymphoblast segmentation (f) Lymphocyte segmentation

(g) Monocyte segmentation (h) Neutrophil segmentation (i) Platelet segmentation

Figure 24: Segmentation results for all nine blood cell classes using specialized image processing pipelines

7.5.4 Synthetic Composite Generation for Detection Testing

The testing and validation of cell detection models requires carefully constructed image samples that mimic real microscopy conditions while providing accurate ground truth labels. Our synthetic composite generation strategy addresses this need by placing segmented cell images with transparency onto realistic background canvases.

Transparent Cell Placement Strategy Unlike the random composite approach discussed earlier, this method leverages pre-segmented cell images with alpha channels (transparency) to achieve more realistic cell integration with the background. The procedure is outlined in Algorithm 12.

Algorithm 12 Transparent Cell Composite Image Generation

Input: Background image path, segmented cell directories, label directories, class names, total cells count

Output: Composite image, cell labels, cell positions

```
Load and tile background image to match canvas size
Initialize empty lists for image positions and cell labels
Filter out classes with no available segmented images
Calculate cells per class (even distribution + remainder)

for each valid class do
    Collect and shuffle all segmented images for this class
    Initialize cell count for this class to 0

    for each segmented image (until target count reached) do
        Load image with alpha channel transparency
        Retrieve class indices from corresponding label file

        for attempt = 1 to max_attempts do
            Generate random position (x, y) on canvas

            if no collision with existing images (with padding) then
                Blend cell with background using alpha channel
                Record class labels and position (x, y, width, height)
                Increment cell count
                break

        if placement unsuccessful then
            Log warning about maximum attempts reached
Return composite image, all labels, image positions
```

Class Distribution Strategy A key aspect of this approach is ensuring balanced representation of all cell classes in the composite image. The algorithm allocates cells per class using:

$$cells\_per\_class = \lfloor \frac{total\_cells}{num\_classes} \rfloor \tag{27}$$

Any remainder cells are distributed one-by-one across classes:

$$remainder = total\_cells \bmod num\_classes \tag{28}$$

Collision Detection with Padding To prevent cell overlap, collision detection is performed with additional padding:

$$\begin{aligned} collision = (x_1 < x_2 + w_2 + padding) \wedge (x_1 + w_1 + padding > x_2) \wedge \\ (y_1 < y_2 + h_2 + padding) \wedge (y_1 + h_1 + padding > y_2) \end{aligned} \tag{29}$$

The padding (typically 5 pixels) ensures that cells appear naturally separated rather than touching, which better simulates real microscopy conditions.

Alpha Blending A critical component for realistic cell placement is alpha blending, which ensures cells integrate naturally with the background texture. For each pixel position (i, j) in the placement region:

$$combined_{i,j} = (1 - \alpha_{i,j}) \cdot background_{i,j} + \alpha_{i,j} \cdot cell_{i,j} \tag{30}$$

where $\alpha_{i,j} \in [0, 1]$ represents the opacity of the cell at pixel (i, j).

Integration with Model Evaluation The generated composite images serve as both training data and evaluation benchmarks. Each composite image:

- Contains a known number of cells from each class
- Preserves exact cell positions and dimensions
- Enables pixel-perfect evaluation of model predictions

Advantages of Synthetic Composite Testing This approach offers several benefits over testing on natural microscopy images:

1. **Controlled Class Distribution:** Ensures sufficient representation of all cell types.
2. **Perfect Ground Truth:** Cell positions and classes are known with pixel-perfect accuracy.
3. **Background Variation:** Different backgrounds can simulate diverse imaging conditions.
4. **Systematic Evaluation:** Detection performance can be analyzed across class types.
5. **Rapid Iteration:** Multiple test scenarios can be generated quickly without manual annotation.

The implementation allows for convenient visualization of both ground truth and model predictions, enabling direct assessment of detection performance in a single view. This facilitates efficient model debugging and performance optimization by highlighting specific detection challenges.

Figure 25 shows an example synthetic composite generated by Algorithm 12.

7.5.5 Conclusion

The initial detection results were not satisfactory, with reduced accuracy compared to previous iterations. Analysis of the failure modes revealed:

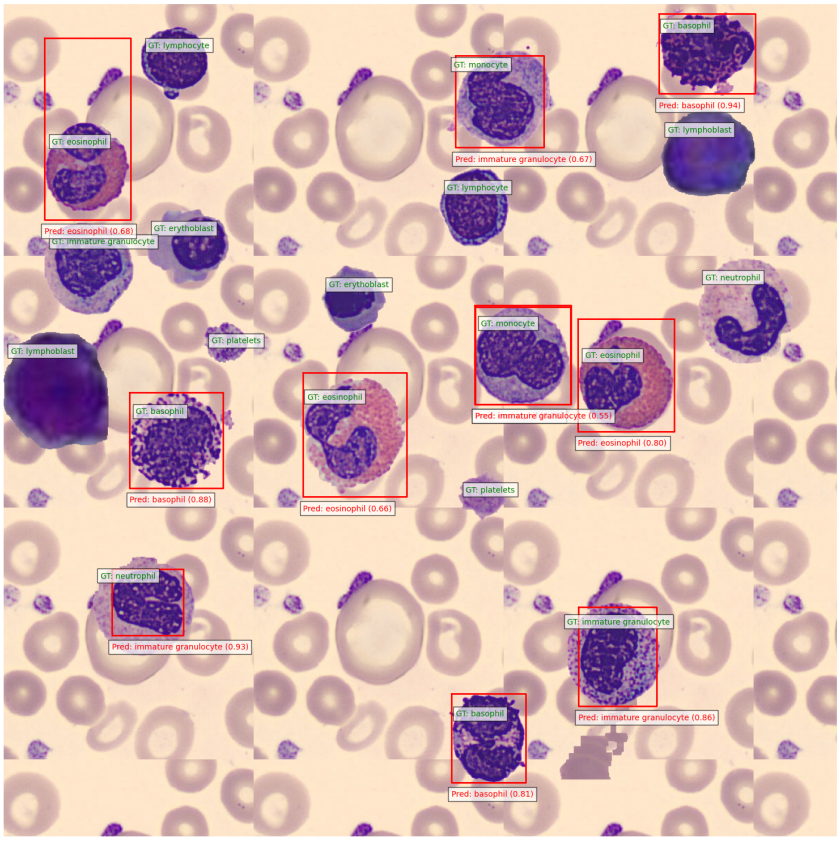


Figure 25: YOLO detection results on synthetic composite image

- The original model was trained on stitched images (1×1, 2×2, 3×3, 4×4) with consistent backgrounds
- This training approach may have introduced bias due to repetitive background patterns
- The model struggled to generalize to the new synthetic dataset with randomly placed cells on varied backgrounds

7.6 Week 11–12: Model Retraining and Robustness Analysis

Weeks 11 and 12 focused on creating a high-quality synthetic dataset using the segmented cell images from previous weeks. Each segmented instance was manually verified to ensure accuracy; poorly segmented samples were discarded. The refined dataset was then used to retrain the detection model. Following this, a comprehensive robustness analysis was conducted to evaluate the model's performance under varying conditions, ensuring generalizability and resilience.

7.6.1 Task Overview

The primary objectives for this phase were:

- Randomly placing segmented cells on diverse background images to create a synthetic dataset
- Retraining the YOLO model with this enhanced dataset
- Evaluating model performance on composite images

- Testing model robustness to various perturbations on both synthetic and external datasets

7.6.2 Synthetic Composite Generation for Dataset Creation

Building upon the previously described approach for testing, we have extended the synthetic composite generation methodology to create comprehensive training and validation datasets. This advancement transforms what was originally a testing tool into a complete data generation pipeline for model training.

Background Image Generation All background canvases (1200×1200 px) were produced by prompting ChatGPT's [8] image-generation API. For each of the 31 randomly selected microscopy images, we provided the full image (including cells) and instructed the model to generate a clean, cell-free 1200×1200 version of the background. This process yielded 31 unique backgrounds tailored to our dataset.

Dataset Generation Pipeline The complete dataset generation pipeline implements a multi-stage process that creates balanced, annotated image sets suitable for training object detection models. The procedure is summarized in Algorithm 13.

Balanced Class Distribution The implementation ensures a balanced representation of cell classes in each image by:

- Allocating cells evenly across available classes, with $cells\_per\_class = \lfloor num\_cells / num\_classes \rfloor$
- Distributing remaining cells randomly to avoid bias
- Setting an appropriate range (15-35) for total cells per image to mimic real slide densities

Split Management The dataset creation process implements a train/validation split strategy:

$$\begin{aligned} \text{train\_count} &= \text{TOTAL\_IMAGES} \times 0.9 \\ \text{val\_count} &= \text{TOTAL\_IMAGES} \times 0.1 \end{aligned} \tag{31}$$

This split occurs at two levels:

1. Cell segment level: Individual cell segments are split 80/15/5% for train/val/test
2. Composite image level: Generated images are split 90/10% for train/val

Performance Optimizations Several optimizations have been implemented to handle large-scale dataset generation:

- Segment size normalization to prevent excessively large cells
- Alpha channel handling for realistic blending
- Efficient collision detection with padding
- Background image reuse with controlled distribution

This comprehensive dataset generation approach enables the creation of large-scale, balanced datasets with perfect ground truth annotations. The resulting datasets can be directly used for training object detection models like YOLO without additional preprocessing, offering a significant advantage over manual annotation workflows.

Algorithm 13 Synthetic Dataset Generation Pipeline

Input: Background images directory, segmented cell directories, output directory, class names, target image count
Output: Train and validation datasets with annotations

```
Initialize directory structure (train/val/images/labels folders)
Set train/validation split ratio (e.g., 90%/10%)

for each cell class do
    Collect all segmented cell images with transparency
    Shuffle and split into train/val sets based on ratio
    Record statistics on available segments per class

while target image count not reached do
    Select appropriate split (train/val) based on current counts
    Choose random background image
    Determine number of cells to place (random 15-35)

    for each cell class do
        Select balanced number of cells from this class

    for each selected cell do
        Load cell image with alpha channel
        Find non-overlapping position using collision detection
        Apply alpha blending with background
        Generate YOLO format annotation (class, center_x, center_y, width, height)

    Save composite image and annotation file
    Update dataset statistics

Generate dataset configuration (dataset.yaml)
Create visualization of dataset statistics
```

7.6.3 Model Training and Evaluation

The YOLO model was retrained on this synthetic dataset and evaluated on a held-out validation set. All performance metrics are reported on manually-verified data. As shown in Figure 26, the confusion matrix highlights per-class accuracy; Figure 27 presents the precision–recall tradeoff; and Figure 28 shows training convergence. A sample detection on a composite image is given in Figure 29.

After this initial retraining on synthetic composites (which achieved excellent detection performance), the final YOLO model was further trained using both segmented-on-background images and multi-scale grid augmentations (from 1×1 up to 4×4), as detailed in Section 4.

7.6.4 Robustness Analysis and Parameter Sensitivity

The robustness of cell detection models is critical in microscopy image analysis, as variations in imaging conditions can significantly impact performance. This section presents a comprehensive framework for evaluating model robustness against common image transformations.

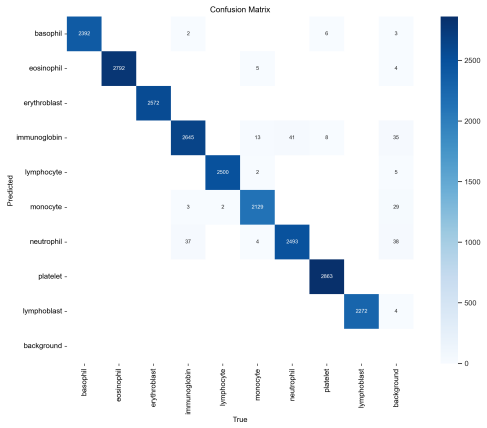


Figure 26: Confusion matrix for validation images showing the model's classification performance across all nine cell types.

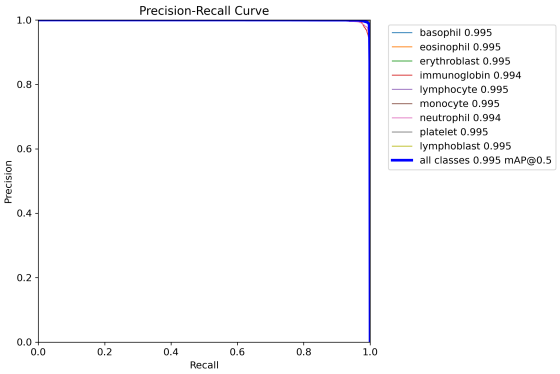


Figure 27: Precision–recall curve for validation images demonstrating detection performance at various confidence thresholds.

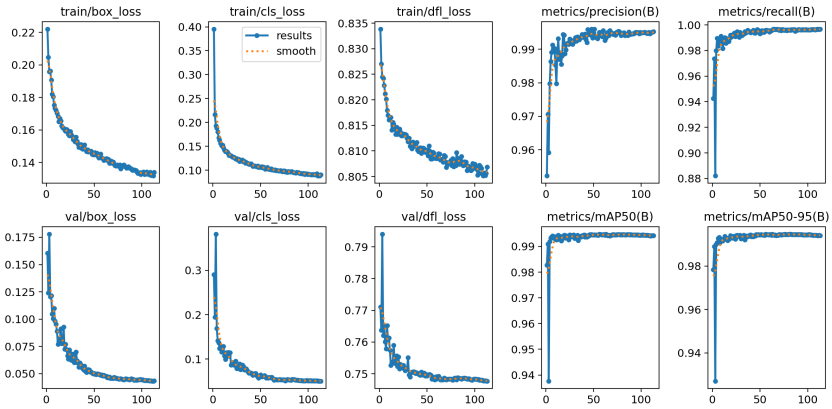


Figure 28: Training and validation loss curves, indicating stable convergence over epochs.

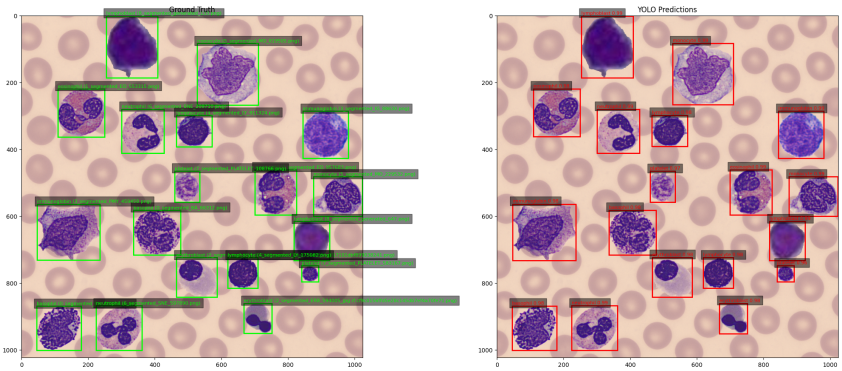


Figure 29: Example detection on a synthetic composite, showing correct identification of all cell classes.

Parameter Sensitivity Framework We implement a systematic evaluation of model sensitivity to image variations through controlled transformation experiments. The framework applies identical transformations across multiple runs while maintaining consistent cell placement for fair comparison.

Algorithm 14 Robustness Parameter Testing Framework

```
for run = 1 to num_runs do
    Generate master seed for run
    Create reference layout with original cells
    for each transformation parameter do
        Apply transformation to reference cells
        Run object detection model
        Calculate precision, recall, F1 score
        Generate visualizations and confusion matrices
    Compute run-specific metrics
Aggregate results across runs
Perform statistical analysis of performance differences
```

Image Transformation Suite The code implements a comprehensive suite of image transformations to simulate various imaging conditions:

Transformation	**Description**	**Implementation**
Original	Unmodified baseline	Direct copy
Grayscale	Color information removal	RGB to grayscale conversion
Light Blur	Minor focus issues	Gaussian blur (5×5 kernel)
Heavy Blur	Major focus problems	Gaussian blur (15×15 kernel)
Increased Brightness	Overexposure	Scaling ($\alpha = 1.3$) + offset ($\beta = 30$)
Decreased Brightness	Underexposure	Scaling ($\alpha = 0.7$) + offset ($\beta = -30$)
Increased Contrast	Enhanced differences	Scaling ($\alpha = 1.5$)
Decreased Contrast	Reduced differences	Scaling ($\alpha = 0.7$)
Added Noise	Sensor/processing artifacts	Mixed Gaussian and Poisson noise
Scale Down	Size reduction	0.8× scaling
Scale Up	Size increase	1.2× scaling
Random Rotation	Orientation changes	-45° to 45° rotation

Table 8: Image transformation parameters for robustness testing

Controlled Experimental Design A key innovation in this framework is the controlled experimental design that isolates the impact of each transformation:

1. **Position consistency**: For each run, all transformations place cells in identical positions on the background, allowing direct comparison of detection performance across transformations.

2. **Seed management**: The code uses deterministic seeding to ensure reproducibility while enabling multiple independent runs:

- Master seed per run: $42 + \text{run\_idx} \times 100$
- Transformation-specific seeds generated via hash functions
- Consistent rotation angles within each run

3. **Alpha channel handling**: The framework properly preserves transparency during transformations, ensuring accurate cell blending on the background image:

$$\text{pixel}_{final} = \text{pixel}_{background} \times (1 - \alpha) + \text{pixel}_{transformed} \times \alpha \tag{32}$$

Performance Evaluation Model performance is evaluated through a robust object detection assessment methodology:

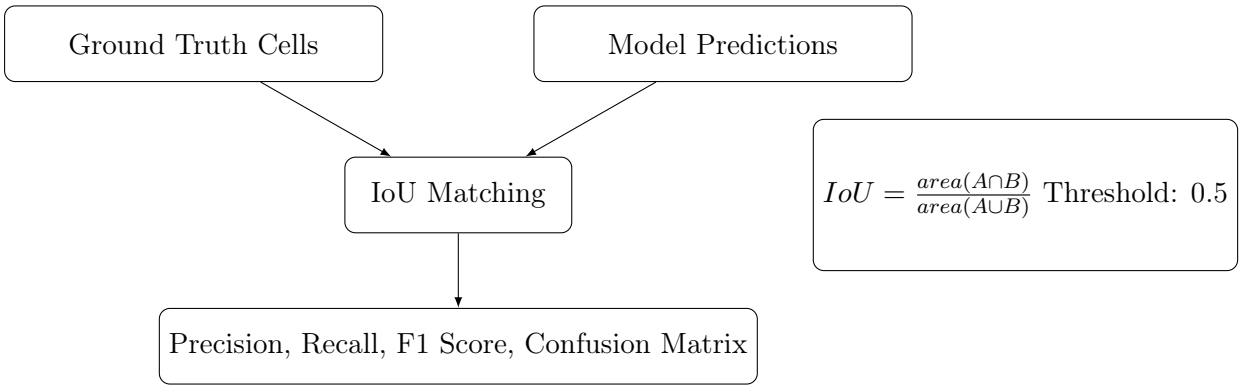


Figure 30: Object detection evaluation process with IoU-based matching

The evaluation process includes:

1. **IoU-based matching**: Detections are matched to ground truth using Intersection over Union with a 0.5 threshold:

$$IoU(box_1, box_2) = \frac{area(box_1 \cap box_2)}{area(box_1 \cup box_2)} \tag{33}$$

2. **Class-specific metrics**: Performance is calculated per cell class and overall:

$$Precision = \frac{TP}{TP + FP} \tag{34}$$

$$Recall = \frac{TP}{TP + FN} \tag{35}$$

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{36}$$

3. **Multi-run aggregation**: Results from multiple runs are aggregated to calculate mean and standard deviation, reducing the impact of randomness.

Statistical Analysis The framework incorporates statistical significance testing using t-tests to determine whether performance differences between the original images and transformed images are statistically significant ($p < 0.05$).

Visualization Suite Comprehensive result visualization includes:

1. **Detection visualizations**: Ground truth (green) vs. predictions (red) for each transformation

2. **Confusion matrices**: Class-wise prediction accuracy

3. **Performance bar charts**: Precision, recall, and F1 score comparison
4. **Box plots**: Distribution of F1 scores across runs
5. **Parameter ranking**: Transformations sorted by impact on performance

Implementation Notes The code employs several techniques to ensure robust and efficient execution:

1. **Mixed noise generation**: The `add_noise` function creates realistic image noise by combining:
   - Gaussian noise with spatial correlation (simulating sensor patterns)
   - Poisson noise on random subsets of pixels (simulating photon shot noise)
2. **Collision detection**: When placing cells, the system checks for overlaps using axis-aligned bounding box intersection tests.
3. **Transparency handling**: Alpha channel preservation ensures realistic cell appearances regardless of transformation.
4. **Progress monitoring**: The implementation uses `tqdm` progress bars for tracking test execution.
5. **Consistent class handling**: The code includes a correction mapping for class names to handle naming inconsistencies between datasets and models.

The framework provides a comprehensive assessment of model robustness, identifying which imaging variations most significantly impact detection performance. This information can guide preprocessing strategies, data augmentation approaches, and model architecture improvements to develop more robust cell detection systems.

7.6.5 Parameter Sensitivity Results

Figure 31 shows the mean F_1, precision, and recall ($\pm$ std) for each perturbation averaged over $R = 5$ runs.

7.6.6 Specific Parameter Effects

Qualitative examples in Figure 32 illustrate detection under key transformations.

7.6.7 Conclusions and Future Work

The model retraining and robustness analysis in Weeks 11-12 yielded several important insights:

Model Training Outcomes:

- The enhanced segmentation approach produced higher-quality cell segments
- Manual curation of the segments ensured dataset quality
- The retrained model successfully identified all cell classes in composite images

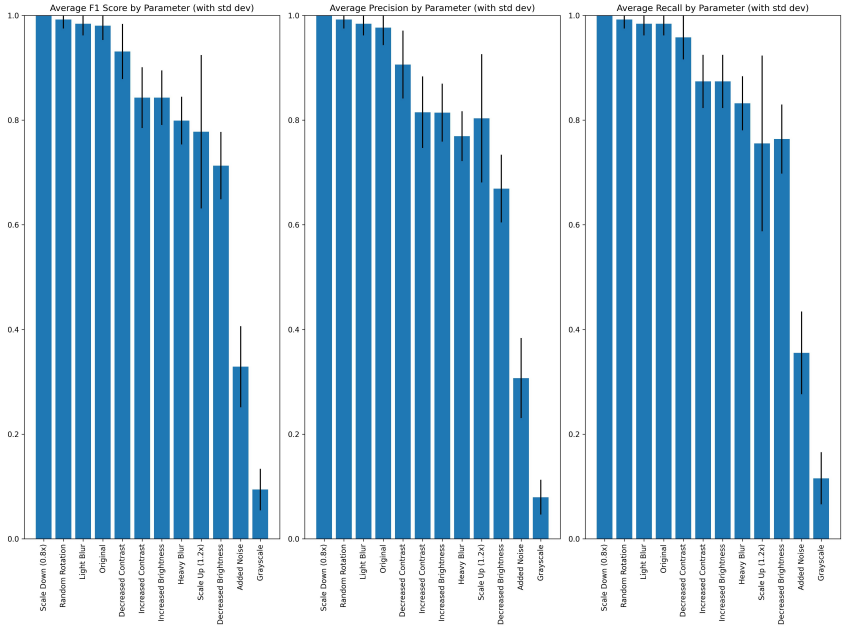


Figure 31: Average F_1, precision, and recall by parameter ($\pm$ std) over $R = 5$ runs.

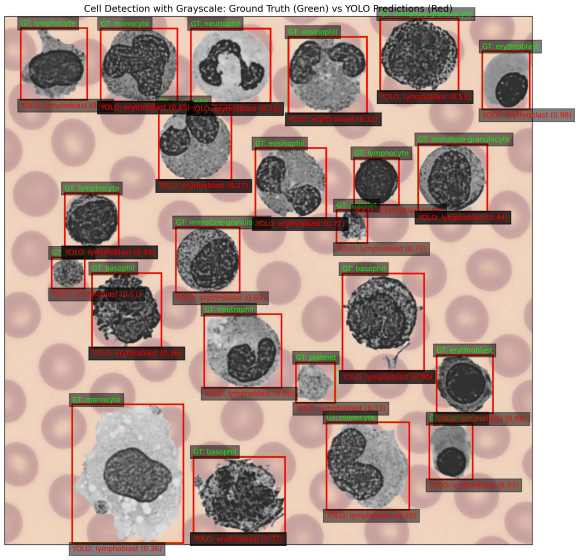
- The confusion matrix and precision-recall curves indicated good performance across all nine cell classes

Robustness Analysis Findings:

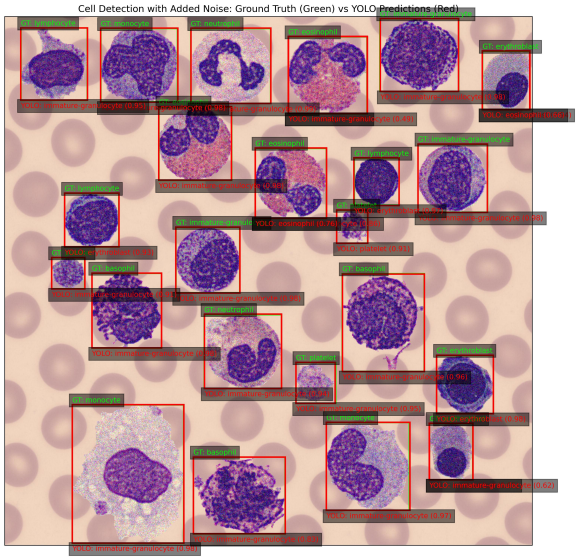
- Color-based variations (grayscale conversion, contrast and brightness adjustments, realistic noise) caused the largest performance degradation
- Geometric transforms (scaling, rotation, light/heavy blur) had comparatively smaller effects on F1, precision, and recall
- Preserving natural color information proved critical for robust YOLO-based cell detection
- The model demonstrated limited generalization to external datasets, indicating dataset bias

Recommendations for Future Work:

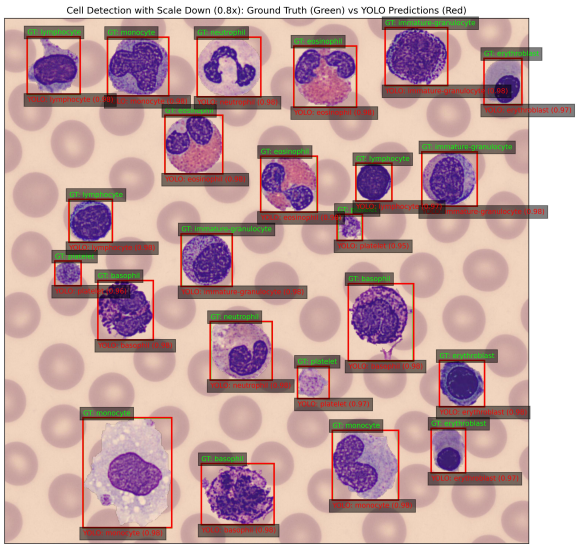
- Incorporate richer color augmentations during training to improve generalization on varied datasets
- Expand the training dataset with more diverse samples, potentially from multiple external sources
- Develop specialized preprocessing pipelines for external datasets to normalize their characteristics
- Explore domain adaptation techniques to bridge the gap between the training distribution and real-world application scenarios
- Consider ensemble approaches that combine models trained with different color space representations



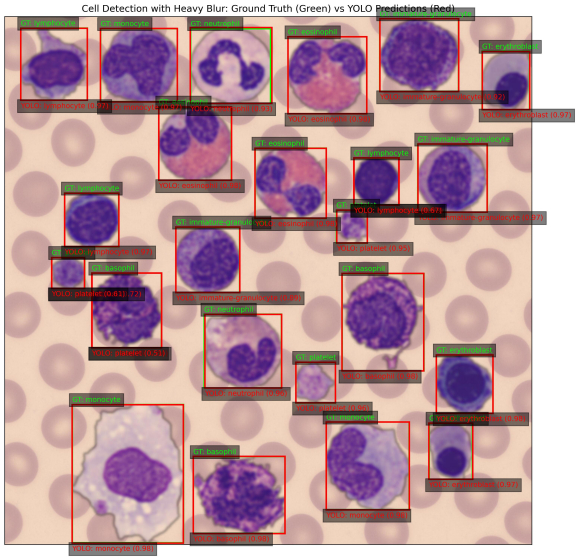
(a) Grayscale



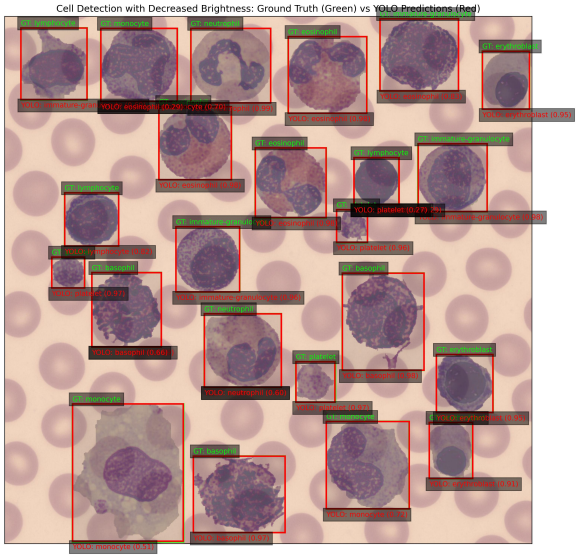
(b) Noise



(c) Scaled Down



(d) Heavy Blur



(e) Low Brightness

Figure 32: Detection performance under various image transformations.

8 Conclusion and Future Work

8.1 Summary of Contributions

In this work, we have developed a comprehensive pipeline for automated blood-cell analysis, comprising:

- A robust segmentation framework to isolate nine leukocyte and platelet types via color- and shape-based processing.
- A novel synthetic dataset generation method that places manually-verified cell segments onto diverse backgrounds, ensuring balanced class coverage.
- Retraining of a YOLO11 detector on this synthetic data, achieving high average F_1 scores across all cell categories.
- A detailed robustness study evaluating detection performance under different perturbations, revealing key sensitivities (e.g., to color removal).
- Demonstration of practical detection results on both synthetic and limited real-world samples, highlighting challenges and guiding improvements.

8.2 Potential Applications

The resulting model and dataset generation strategy have broad utility in:

- **Clinical Diagnostics:** Automated pre-screening of peripheral blood smears to assist hematopathologists.
- **Quality Control:** Verification of manual cell counts and laboratory workflows.
- **Educational Tools:** Interactive platforms for training students in hematology.
- **Research Studies:** High-throughput analysis of morphological changes under disease or treatment conditions.
- **Telemedicine:** Remote blood-smear evaluation in resource-limited settings.

8.3 Future Work

Building on these results, we plan to:

- **Expand Real-World Coverage:** Incorporate diverse clinical images and rare cell types to improve generalization.
- **Domain Adaptation & Fine-Tuning:** Apply unsupervised or few-shot learning to bridge synthetic–real domain gaps.
- **Alternative Architectures:** Evaluate transformer-based and two-stage detectors (e.g., Faster R-CNN) for improved small-object detection.
- **End-to-End Integration:** Combine segmentation and detection into a unified network for streamlined deployment.

9 Summary of Weekly Work

Throughout the 12-week project, the following milestones were achieved:

- **Weeks 1–2 (Problem Formulation & Initial Testing):** Defined the problem statement and objectives; conducted a targeted literature review; built and evaluated an initial classification prototype; made a strategic pivot from classification to a YOLO-based detection framework.
- **Weeks 3–4 (YOLO Implementation & Evaluation):** Set up data cleaning and preprocessing; trained the YOLO11 model; evaluated on both single-cell and composite images; developed the composite-image generation pipeline; documented key findings.
- **Weeks 5–6 (Data Curation & Augmentation):** Performed detailed error analysis on combined images; relabeled underperforming classes (platelets); optimized mixup and mosaic augmentation parameters; demonstrated significant performance gains on 2×2 and 3×3 composites.
- **Weeks 7–8 (Composite Generation & Leak Correction):** Generated leak-free composite datasets across grid sizes (1×1–4×4) via a strict pre-split protocol; implemented random composite strategies; retrained the YOLO11m detector; validated robustness post data-leakage fix.
- **Weeks 9–10 (Cell Segmentation Refinement):** Developed bespoke segmentation pipelines for all nine blood-cell classes (e.g. basophils, lymphoblasts, granulocytes); produced transparent-PNG crops; integrated these into synthetic testing workflows.
- **Weeks 11–12 (Retraining & Robustness Analysis):** Retrained the detector on the refined dataset; conducted extensive robustness and parameter-sensitivity studies under blur, noise, brightness shifts, and geometric perturbations; finalized model and documented future directions.

10 Code Availability

The code used in this work is available at:

`https://github.com/Afnnnan/DH307-BloodCell-YOLO11`

11 Acknowledgements

I would like to thank Prof. Nirmal Punjabi for his invaluable guidance and support throughout this project. Special thanks also go to Buddhadev Goswami, whose conceptual insights, model-training assistance, and provision of ground-truth YOLO labels were essential to the completion of this work.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[2] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der

Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017.

[3] M. Shahzad, F. Ali, S. H. Shirazi, A. Rasheed, A. Ahmad, B. Shah, and D. Kwak, "Blood cell image segmentation and classification: a systematic review," *PeerJ Computer Science*, vol. 10, p. e1813, 2024. [Online]. Available: https://doi.org/10.7717/peerj-cs.1813

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016. [Online]. Available: https://arxiv.org/abs/1506.01497

[5] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2999–3007.

[6] A. Acevedo, A. Merino, S. Alférez, A. Molina, L. Boldú, and J. Rodellar, "A dataset for microscopic peripheral blood cell images for development of automatic recognition systems," Mendeley Data, V1, 2020.

[7] R. Donida Labati, V. Piuri, and F. Scotti, "All-idb: the acute lymphoblastic leukemia image database for image processing," in *IEEE Int. Conf. on Image Processing (ICIP)*, 2011, pp. 2045–2048.

[8] OpenAI, "ChatGPT (gpt-4) image editing," https://chat.openai.com, 2025, used to remove the central cell and expand 31 blood-cell source images to 1200×1200 px canvases.

[9] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024. [Online]. Available: https://github.com/ultralytics/ultralytics

[10] B. Goswami, A. B. Somaraj, P. Chakrabarti, R. Gudi, and N. Punjabi, "Classifier enhanced deep learning model for erythroblast differentiation with limited data," 2024. [Online]. Available: https://arxiv.org/abs/2411.15592

[11] K. Dwivedi and M. K. Dutta, "Microcell-net: A deep neural network for multi-class classification of microscopic blood cell images," *Expert Systems*, vol. 40, 04 2023.

[12] M. Nickparvar *et al.*, "White blood cells dataset," https://www.kaggle.com/datasets/masoudnickparvar/white-blood-cells-dataset, 2020, accessed: 2025-05-03.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[14] P. Skalski, "Make Sense," https://github.com/SkalskiP/make-sense/, 2019, accessed: 2025-05-03.

[15] Adobe, "Adobe firefly," https://firefly.adobe.com/, 2025.

[16] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[17] C. Rother, V. Kolmogorov, and A. Blake, ""grabcut": interactive foreground extraction using iterated graph cuts," *ACM Trans. Graph.*, vol. 23, no. 3, p. 309–314, Aug. 2004. [Online]. Available: https://doi.org/10.1145/1015706.1015720