Blood Cell Count and Detection

Yassin Abdulmahdi

November 13, 2024

Abstract

Purpose The BCCD (Blood Cell Count and Detection) dataset consists of images containing three types of blood cells: Red Blood Cells (RBC), White Blood Cells (WBC), and Platelets. The dataset includes images stored in the `JPEGImages` folder and corresponding annotations in XML format stored in the `Annotations` folder. Each annotation provides bounding boxes around the cells.

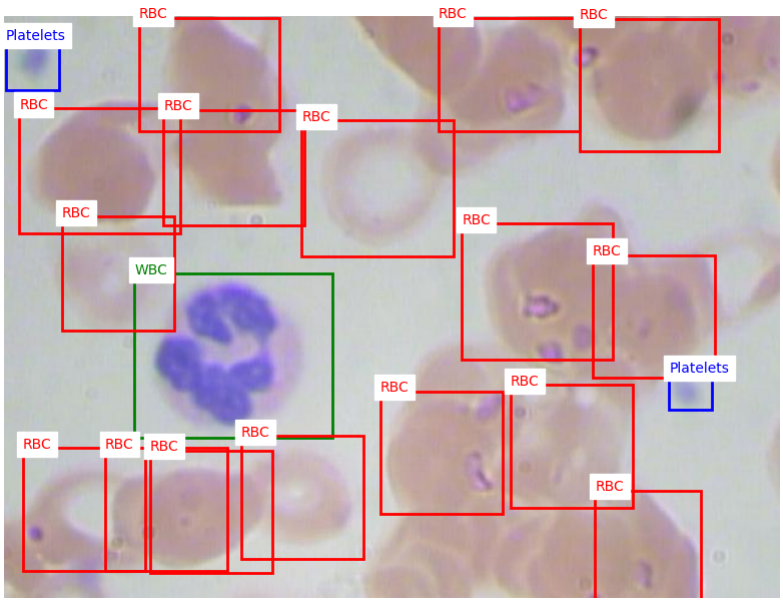


Figure 1: Sample Image with Bounding Boxes

1 Data Augmentation Techniques

To improve the robustness and generalization of the model, several data augmentation techniques were applied to the images and their corresponding bounding box annotations. The augmentations are performed randomly with specified probabilities, including horizontal flipping, color jitter, rotation, and cropping. Each technique is outlined below with the respective algorithm.

1.1 Random Horizontal Flip

Objective: This augmentation flips the image horizontally with a given probability. The bounding boxes are adjusted accordingly by swapping the x-min and x-max values.

Algorithm 1 Random Horizontal Flip

function RANDOM_HORIZONTAL_FLIP(image, target, prob)
    Input: image, target (bounding boxes), prob (flip probability)
    if random() ¡ prob **then**
        Flip the image horizontally
        for each bounding box in target[”boxes”] **do**
            Swap the x-min and x-max values of the bounding box
    return image, target

1.1.1 Sample Image:

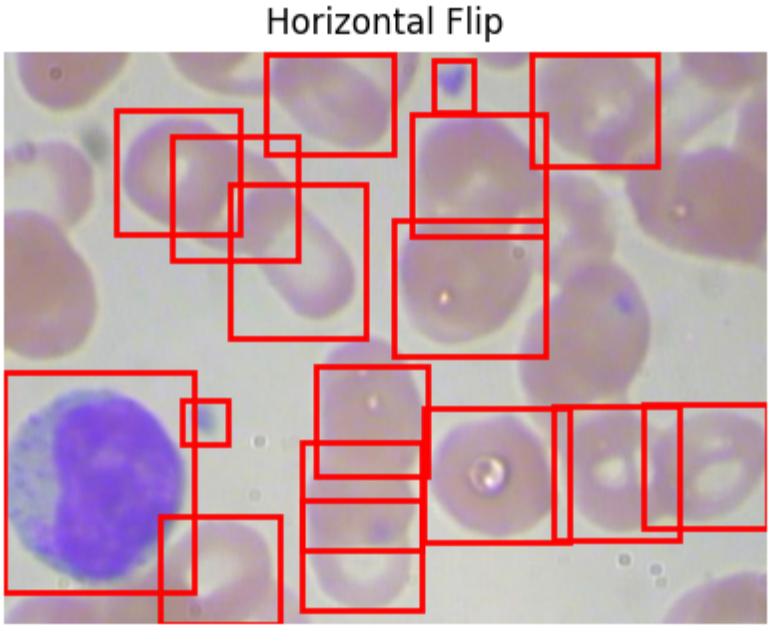


Figure 2: Sample Image with Random Horizontal Flip

1.2 Color Jitter

Objective: This augmentation adjusts the brightness, contrast, saturation, and hue of the image to simulate different lighting conditions.

Algorithm 2 Color Jitter

function COLOR_JITTER(image, target)
    Input: image, target
    Apply random adjustments to brightness, contrast, saturation, and hue of the image
    return image, target

1.2.1 Sample Image:

1.3 Random Rotation

Objective: This augmentation rotates the image by a random angle within a specified range. The bounding boxes are also rotated to maintain correct object localization.

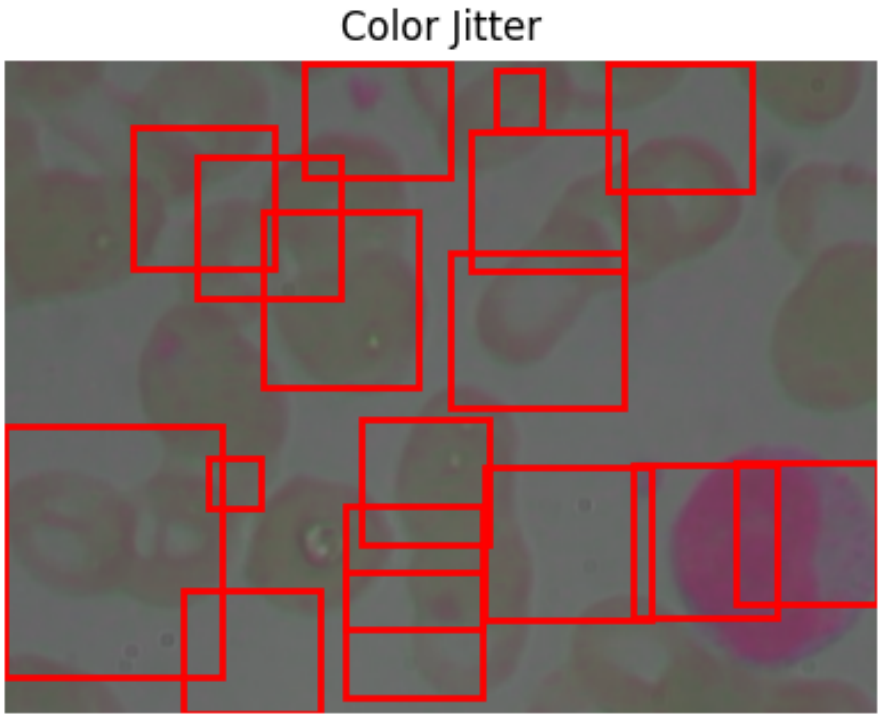


Figure 3: Sample Image with Color Jitter

Algorithm 3 Random Rotation

```
function RANDOM_ROTATION(image, target, degrees)
    Input: image, target (bounding boxes), degrees (angle range)
    Randomly choose an angle within the range [−degrees, degrees]
    Rotate the image by the chosen angle
    for each bounding box in target["boxes"] do
        Convert bounding box coordinates to corner points
        Rotate corner points using the chosen angle
        Convert rotated points back to bounding box coordinates
    return rotated image, updated target
```

1.3.1 Sample Image:

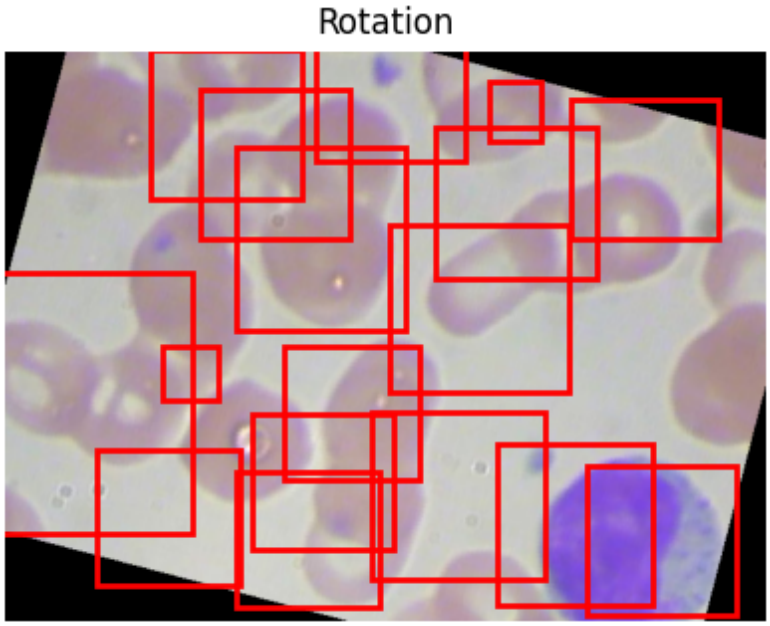


Figure 4: Sample Image with Random Rotation

1.4 Random Crop

Objective: This augmentation randomly crops a region of the image, ensuring that the bounding boxes are within the cropped area. Bounding boxes outside the cropped region are removed.

Algorithm 4 Random Crop

```
function RANDOM_CROP(image, target, min_size)
    Input: image, target (bounding boxes), min_size (minimum crop size)
    Randomly select a crop size and position
    Crop the image at the selected position
    for each bounding box in target["boxes"] do
        Adjust the bounding box coordinates based on crop position
        Clip the bounding box coordinates to remain within the cropped region
        Remove bounding boxes that no longer fit within the cropped image
    return cropped image, updated target
```

1.4.1 Sample Image:

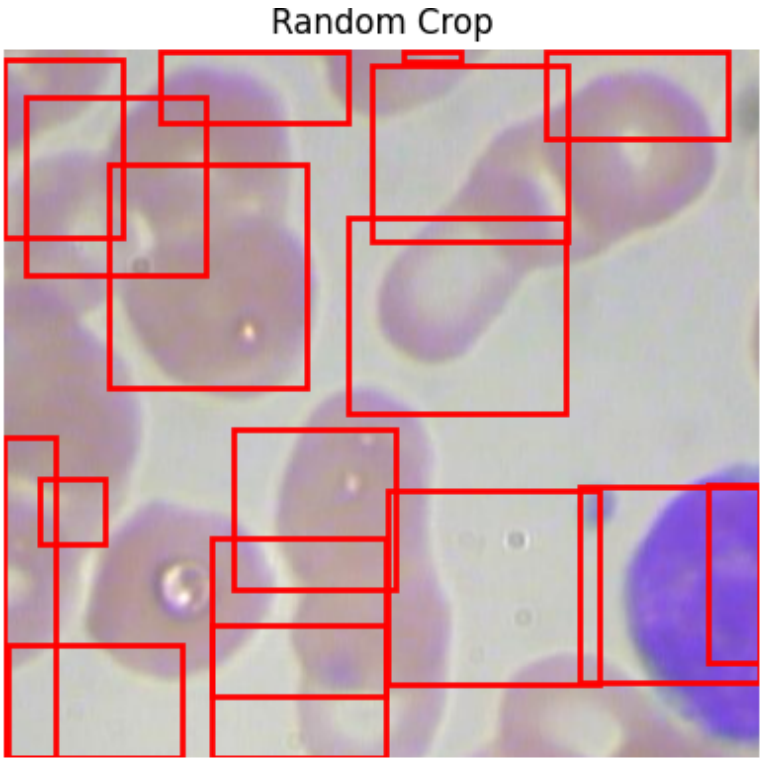


Figure 5: Sample Image with Random Crop

2 Data Augmentation and Metrics Calculation

The following subsections describe the process of preparing the transformations and calculating the evaluation metrics using the COCO format.

2.1 Data Augmentation Transformation Pipeline

The function `get_transform` is responsible for applying a set of transformations to the images during training or validation. The transformations include converting the image to a tensor, performing random horizontal flips, rotations, color jitter, and random cropping.

Algorithm 5 Data Augmentation Pipeline

function GET_TRANSFORM(train)
    Input: train (boolean indicating whether to apply transformations for training)
    Output: transforms (list of augmentation transformations)
    Initialize an empty list `transforms`
    Append `T.ToTensor()` to `transforms`
    if train **then**
        Append RandomHorizontalFlip(0.5) to `transforms`
        Append RandomRotation(10) to `transforms`
        Append ColorJitter(0.2, 0.2, 0.2, 0.1) to `transforms`
        Append RandomCrop(400) to `transforms` ▷ Adjusted default crop size
    return Compose(transforms)

2.2 Metrics Calculation

The function `calculate_metrics` is responsible for calculating the performance of the model using the COCO evaluation metric. It evaluates precision and recall for object detection tasks by comparing ground truth annotations with predicted results.

Algorithm 6 Metrics Calculation

function CALCULATE_METRICS(model, data_loader, device, iou_threshold)
    Input: model (trained model), data_loader (data for evaluation), device (device to use for computation), iou_threshold (IoU threshold for evaluation)
    Output: metrics dictionary containing AP, AP50, AP75, APs, APm, APl
    Set model to evaluation mode: `model.eval()`
    Initialize `coco_gt` and `coco_dt` for storing ground truth and predicted annotations
    for each batch of images and targets in `data_loader` **do**
        Transfer images to the specified device
        for each target in `targets` **do**
            Add image info (ID, width, height) to `coco_gt`
            for each box, label in target["boxes"], target["labels"] **do**
                Add bounding box and label annotation to `coco_gt`
        Get predictions from the model
        for each predicted box, score, label in the model predictions **do**
            Add predicted bounding box and score to `coco_dt`
    Create COCO ground truth object `gt_coco`
    Create COCO detection object `dt_coco`
    Create a COCO evaluator `cocoEval`
    Set IoU threshold for evaluation: `cocoEval.params.iouThrs = [iou_threshold]`
    Run evaluation and accumulate results
    Summarize the evaluation results
    return dictionary with the metrics: AP, AP50, AP75, APs, APm, APl

2.3 Model Setup Function

The `get_model` function initializes a Faster R-CNN model with a ResNet50 backbone and replaces the classifier head to match the number of classes.

2.3.1 `get_model` Function

1. Initialize the Faster R-CNN model with a ResNet50 backbone and FPN (Feature Pyramid Networks) .
2. Get the number of input features for the classification layer from the model.
3. Replace the box predictor head in the ROI heads with a new `FastRCNNPredictor` that matches the number of classes.
4. Return the modified model.

2.4 Training Loop

The function `train_one_epoch` performs one epoch of training by iterating over the data loader, computing losses, and updating the model.

Evaluation Results

Training and Validation Losses

During the training of the model over 10 epochs, we recorded the following training and validation losses:

- **Training Losses: [0.972, 0.904, 0.905, 0.956]**
- **Validation Losses: [1.033, 0.867, 0.916, 0.950]**

Evaluation Metrics

The evaluation of the model's performance on the validation dataset provides the following Average Precision (AP) and Average Recall (AR) metrics:

Average Precision (AP)

- **AP@[IoU=0.50:0.50 — area=all]**: 0.092
- **AP@[IoU=0.50 — area=all]**: -1.000
- **AP@[IoU=0.75 — area=all]**: -1.000
- **AP@[IoU=0.50:0.50 — area=small]**: 0.000
- **AP@[IoU=0.50:0.50 — area=medium]**: 0.085
- **AP@[IoU=0.50:0.50 — area=large]**: 0.120

Average Recall (AR)

- **AR@[IoU=0.50:0.50 — area=all — maxDets=1]**: 0.099
- **AR@[IoU=0.50:0.50 — area=all — maxDets=10]**: 0.197
- **AR@[IoU=0.50:0.50 — area=all — maxDets=100]**: 0.334
- **AR@[IoU=0.50:0.50 — area=small — maxDets=100]**: 0.000
- **AR@[IoU=0.50:0.50 — area=medium — maxDets=100]**: 0.325
- **AR@[IoU=0.50:0.50 — area=large — maxDets=100]**: 0.456

Final Evaluation Metrics

The following metrics were calculated after the final evaluation:

- **AP**: 0.092
- **AP50**: -1.000
- **AP75**: -1.000
- **APs** (small objects): 0.000
- **APm** (medium objects): 0.085
- **APl** (large objects): 0.120

Visualization of Predictions

The following results were obtained by visualizing the model's predictions alongside the ground truth for a set of test images:

- The ground truth bounding boxes were drawn in **green**, while the predicted bounding boxes were shown in **red** if their confidence score was above the threshold of 0.5.
- Images were displayed with the corresponding ground truth on the left and predictions on the right.

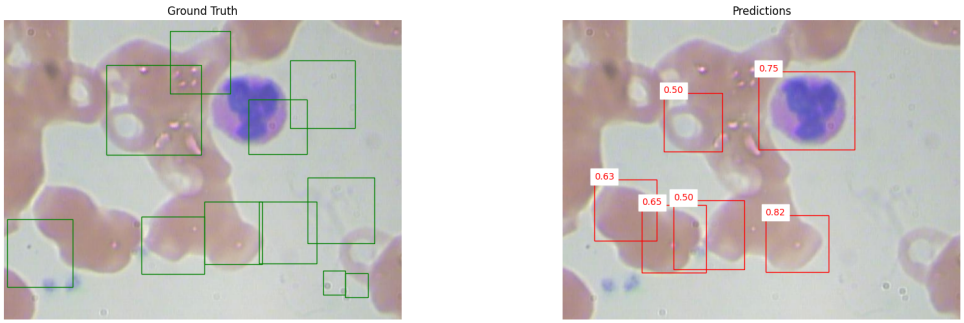


Figure 6: Sample Image with Predictions

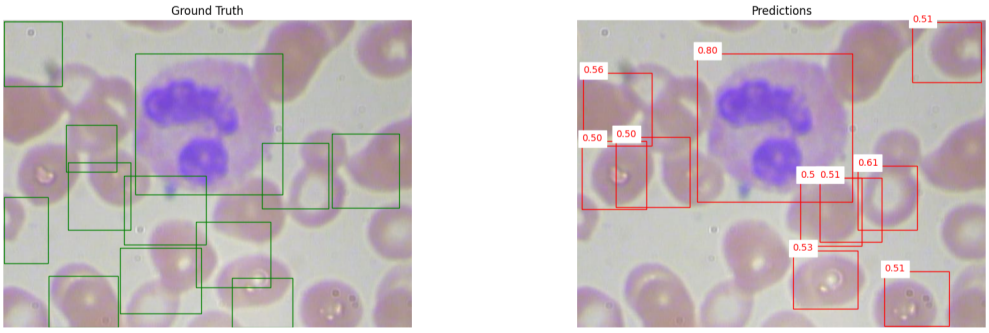


Figure 7: Sample Image with Predictions

Algorithm for YOLOv8 Blood Cell Detection

Step 1: Dataset Conversion to YOLO Format

1. Create necessary directories for the training and validation images and labels:
2. Define the class mapping for blood cells:
   - `RBC` → 0, `WBC` → 1, `Platelets` → 2.
3. Split the image dataset into training and validation sets (80% for training, 20% for validation).
4. For each image in the training and validation sets:
   (a) Copy the image to the appropriate directory.
   (b) Parse the corresponding XML annotation file to extract bounding box coordinates and object class.
   (c) Convert the bounding box from absolute coordinates to relative coordinates (center, width, height) as required by YOLO.

Step 2: Prepare Dataset YAML Configuration

1. Create a YAML file to define the paths to the dataset:
   - `path` → base directory of the dataset.
   - `train` → path to training images directory.
   - `val` → path to validation images directory.
   - `nc` → number of classes (3: RBC, WBC, Platelets).
   - `names` → list of class names: `['RBC', 'WBC', 'Platelets']`.

Step 3: Train YOLOv8 Model

1. Load a pretrained YOLOv8 model (YOLOv8n) as the starting point.
2. Set the training parameters:
   - `epochs` → 50 (number of training epochs).
   - `imgsz` → 640 (image size).
   - `batch` → 16 (batch size).
   - `patience` → 10 (early stopping if the model does not improve).
   - `device` → CUDA (GPU) if available, else use CPU.
3. Start the training process on the dataset defined by the YAML configuration.
4. Save the model weights after each epoch.

Step 4: Evaluate Model Performance

1. After training, validate the model on the validation set using the `val` function of YOLOv8.
2. Evaluate the following metrics:
   - `mAP50` → Mean Average Precision at IoU 0.50.
   - `mAP50-95` → Mean Average Precision at IoU from 0.50 to 0.95.
   - `precision` → the ratio of true positive detections to all predicted positive detections.
   - `recall` → the ratio of true positive detections to all actual positive samples.

YOLOv8 Model Evaluation Results

After training the YOLOv8 model on the BCCD dataset, the following results and evaluation metrics were obtained.

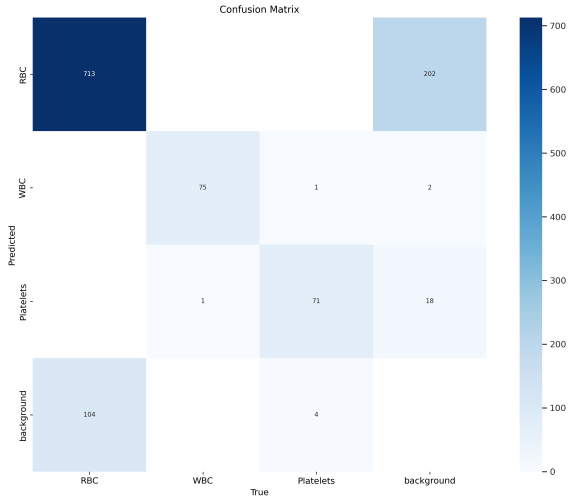
Evaluation Metrics

The performance of the trained YOLOv8 model was evaluated using several metrics, as shown below:

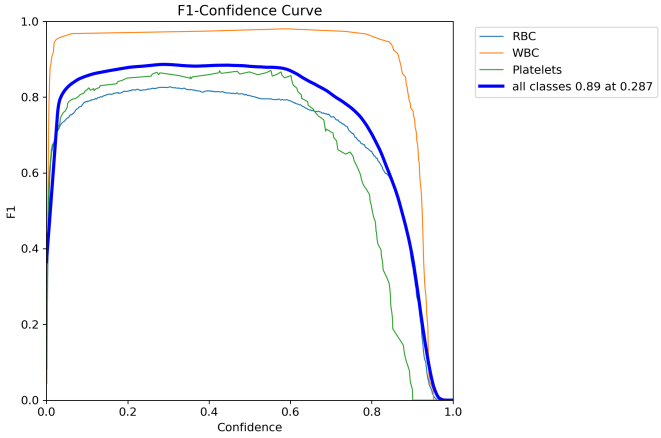
- **mAP50**: 0.9234 (Mean Average Precision at IoU 0.50)
- **mAP50-95**: 0.6530 (Mean Average Precision at IoU 0.50:0.95)
- **Precision**: 0.8556 (Precision of the model)
- **Recall**: 0.9201 (Recall of the model)

Evaluation Plots and Images

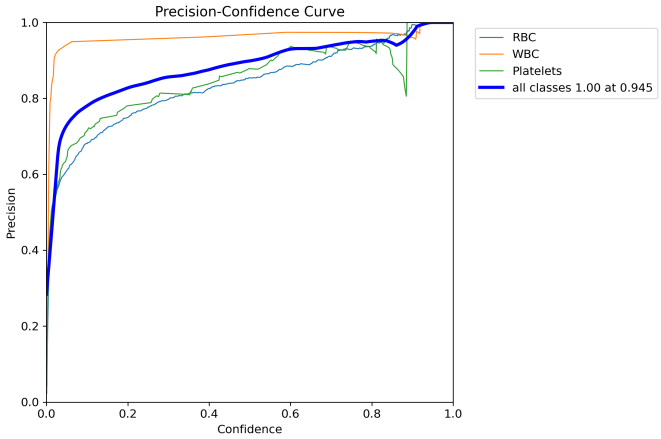
1. **Confusion Matrix**: This plot provides an overview of the model's performance across different classes.



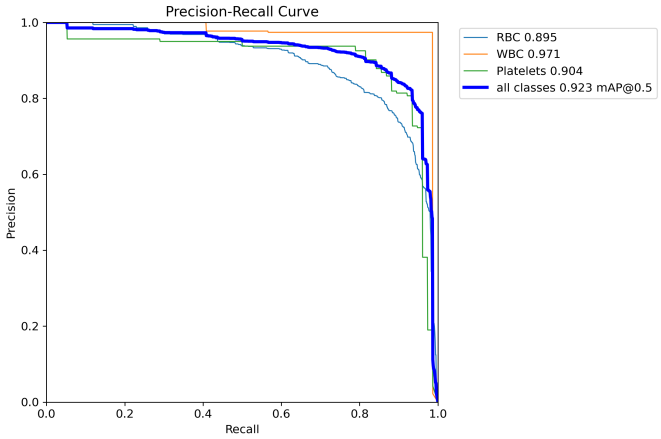
2. **F1-Score Curve**: This curve shows the F1-score across different thresholds, illustrating the model's trade-off between precision and recall.



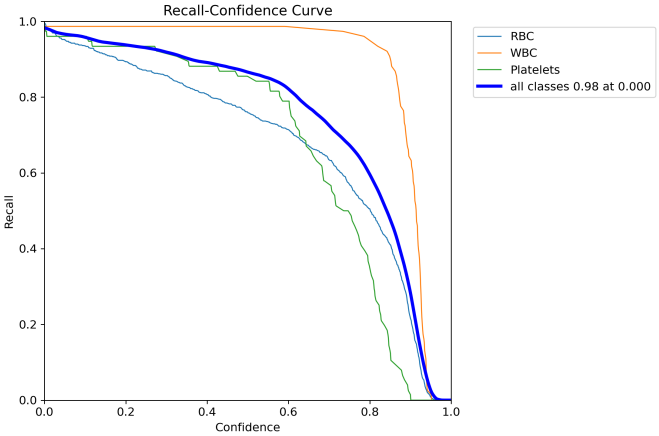
3. **Precision vs. Confidence**: This plot illustrates the relationship between the confidence score and precision.



4. **Precision-Recall Curve**: This curve shows the model's precision-recall trade-off.

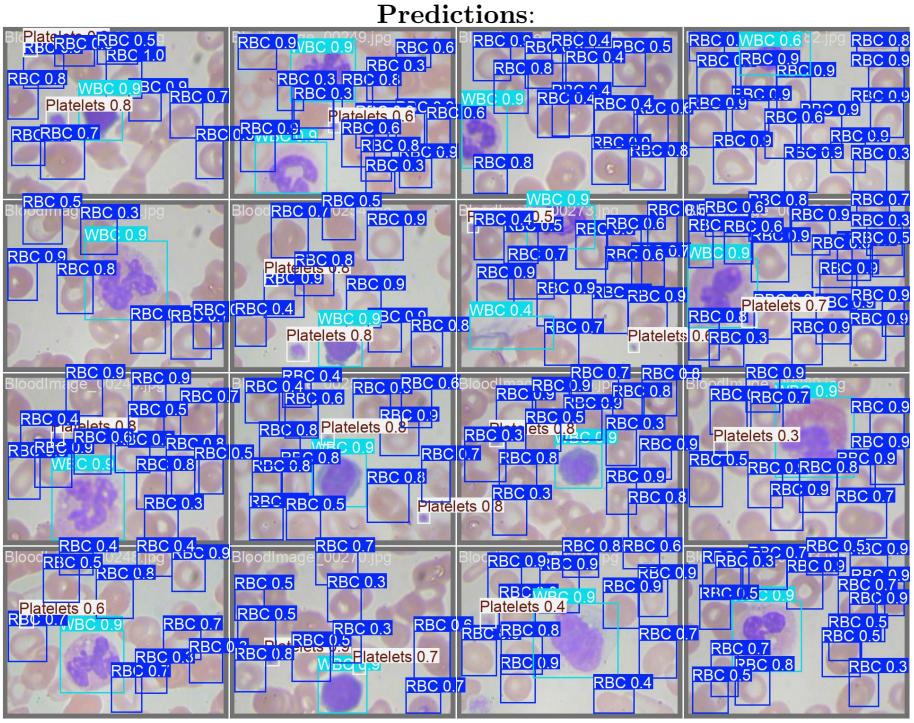


5. **Recall vs. Confidence**: This plot illustrates the relationship between the confidence score and recall.

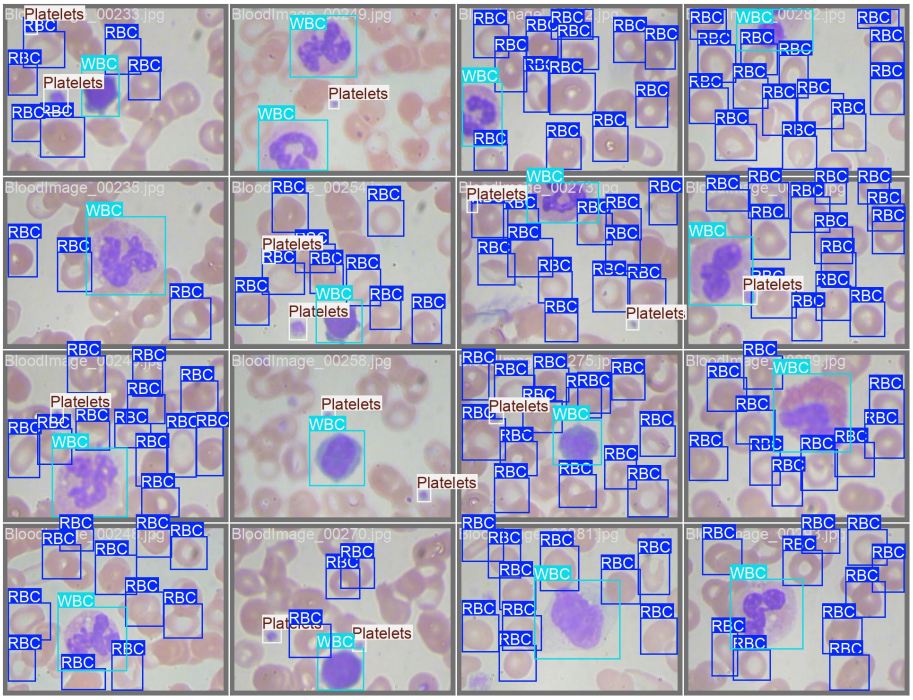


6. **Validation Predictions and Labels**: The following images show the predictions and ground truth labels for a sample batch from the validation set.

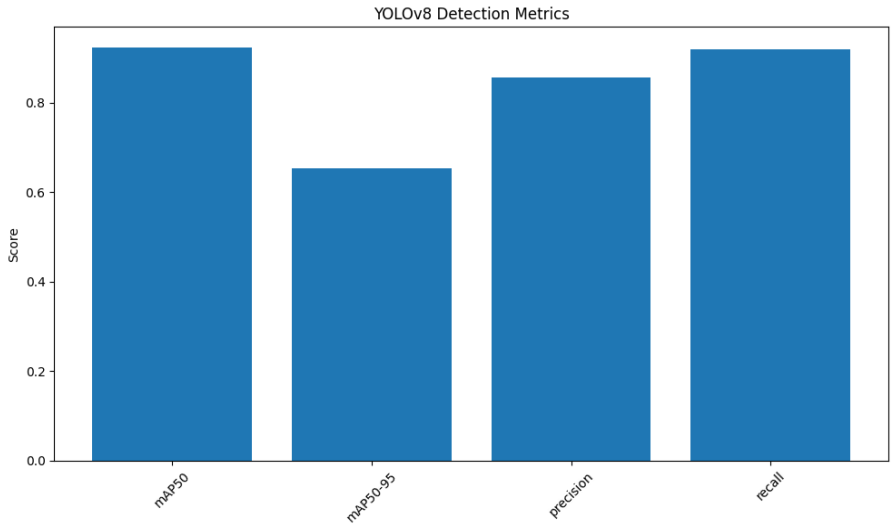
Predictions:



Ground Truth Labels:



7. **Metrics Plot**: This plot shows the model's performance metrics (precision, recall, mAP50, etc.).



Conclusion

The YOLOv8 model demonstrates strong performance on the BCCD dataset, with an impressive mAP50 of 0.923, indicating good accuracy in detecting blood cells. The precision, recall, and other metrics further highlight the model's ability to make reliable predictions across the different classes (RBC, WBC, Platelets).