Blood cells detection and classification using Faster R-CNN model

Capstone 2 Final report

Table of contents:

Introduction.

A complete blood cell count (CBC) is a vital medical test that provides valuable insights into a patient's overall health and can help diagnose a wide range of medical conditions. The CBC provides a detailed analysis of the different types of cells present in the blood, including red blood cells (RBCs), white blood cells (WBCs), and platelets. By examining the levels of these cells, doctors can detect various medical conditions such as anemia, infections, and blood disorders.

RBCs carry oxygen throughout the body, and low levels of RBCs may indicate anemia, while high levels may indicate dehydration. WBCs are the body's immune system cells that help fight off infections, and high levels of WBCs may indicate an infection or inflammation. Platelets are essential for blood clotting, and low levels of platelets may lead to excessive bleeding. The CBC can provide vital information to healthcare professionals that can help them diagnose and treat medical conditions early, which can improve the patient's prognosis and overall quality of life.

Traditionally, counting blood cells manually using a haemocytometer is time-consuming, prone to errors, and relies heavily on the expertise of the laboratory analyst. This is because blood samples typically contain a large number of blood cells.

However, the development of machine learning techniques, particularly deep learning, has led to more accurate and robust image classification and object detection applications. As a result, deep learning-based methods are now being used in various medical applications, including the detection of abnormalities in chest X-rays, automatic segmentation of the left ventricle in cardiac MRI, and identification of diabetic retinopathy in retinal fundus photographs.

Therefore, it is worth exploring the potential of deep learning-based methods to identify and count blood cells in smear images, which could greatly simplify and expedite the entire counting process. By implementing automated processes to count different types of blood cells from a smear image, we can overcome the limitations of traditional manual blood cell counting systems, and improve the accuracy and efficiency of blood cell analysis in clinical settings.

Problem statement:

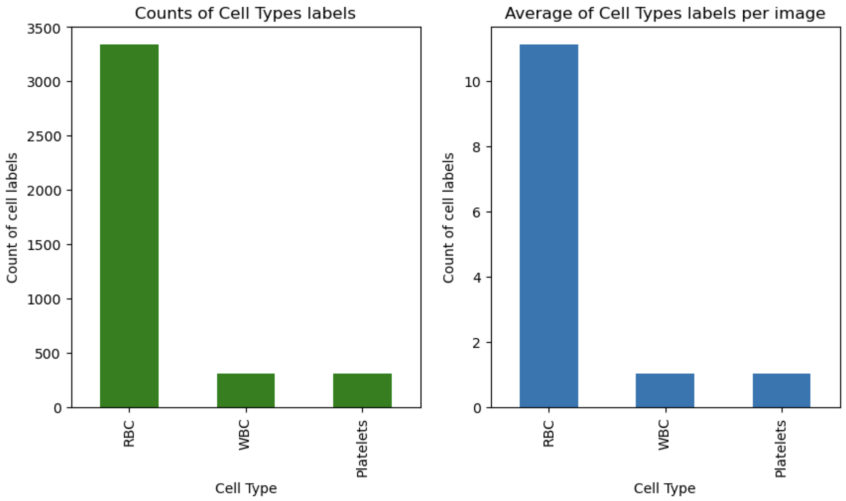
There is a need to develop an automated system that can predict the cell type and box coordinates of RBCs, WBCs, and platelets in complete blood cell (CBC) images. The existing methods of manual counting and classification are time-consuming, labor-intensive, and prone to errors. A more efficient and accurate solution is required to improve the speed and accuracy of CBC analysis.

Dataset description:

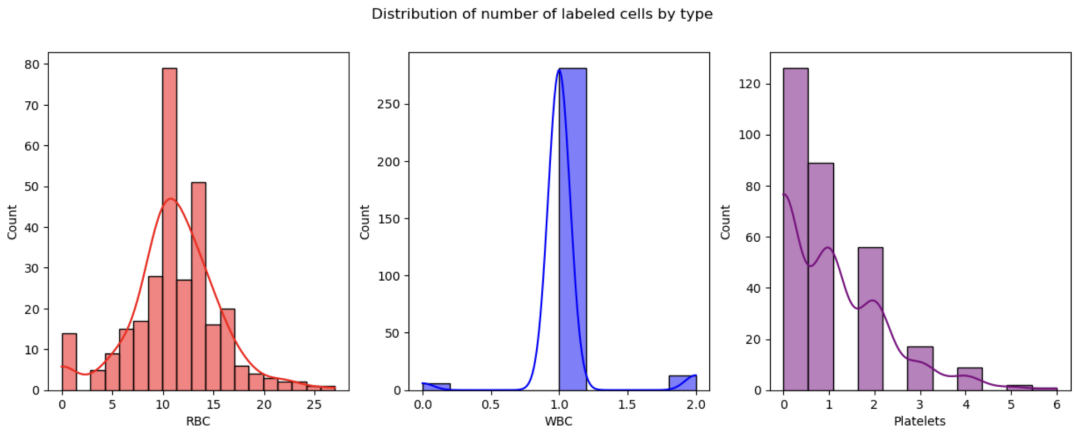
The CBC dataset contains 360 blood smear images with annotations. The training folder contains 300 images with annotations. The testing folder contains 60 images with annotations. It is a modification of the MIT Licence dataset with removal of fallacious images and labels.

Data wrangling and EDA.

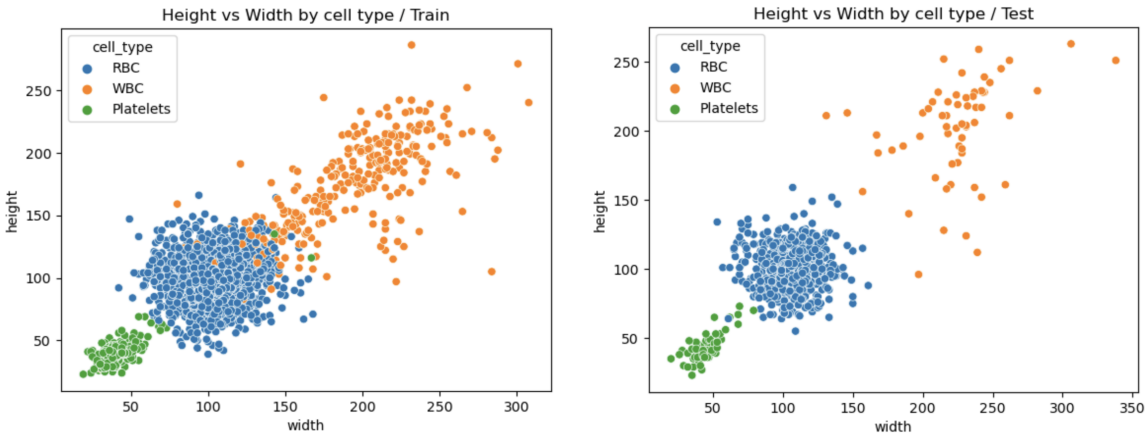
Dataset consists of .jpg images of size 640*480 px and cells bounding boxes and labels stored in corresponding .xml files for each image.
All these .xml files were parsed and new data frames were created for train and test datasets.
The structure of them is - ‘filename’, ‘cell_type’, ‘xmin’, ‘ymin’, ‘xmax’, ‘ymax’.
Size of train dataframe is (3951, 6) with no missing values,
Size of test - (907, 6)
Distribution of labels by cell types:



As we can see RBC cells number is much higher than numbers for WBC and Platelets. On average there is 1 WBC and 1 Platelet for blood smear. Our case represents a classification task with an unbalanced dataset.

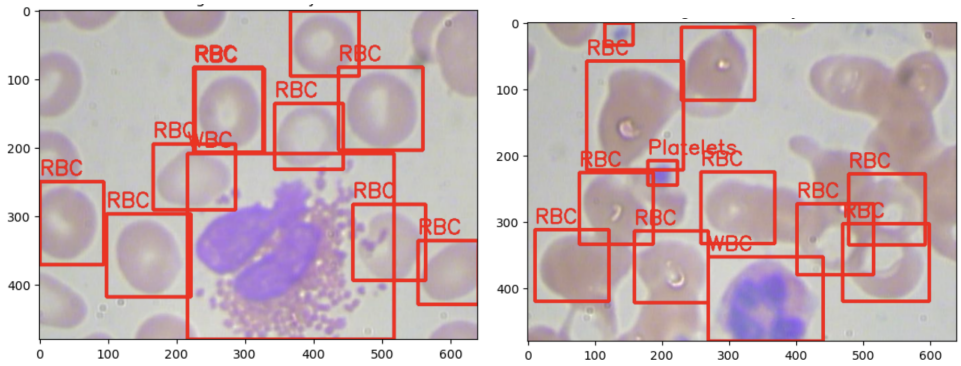


Out of 300 images there are 13 images with 2 labeled WBC and 6 with 0 WBC, more than 3 Platelets are labeled on 12 images. There are 14 images without RBC labeled.



From this plot we can see that in most cases we can separate cell classes by their size. Platelets are small cells in 95% cases. Only several images have platelets of height and width more than 56 pixels and height higher 52.85px. White blood cells are bigger than Red blood cells but their height and width can overlap and lead to misclassification. Test dataset has a more distinguished separation between classes by their size.

Let's see an example of images with bounding boxes:



Object detection approaches.

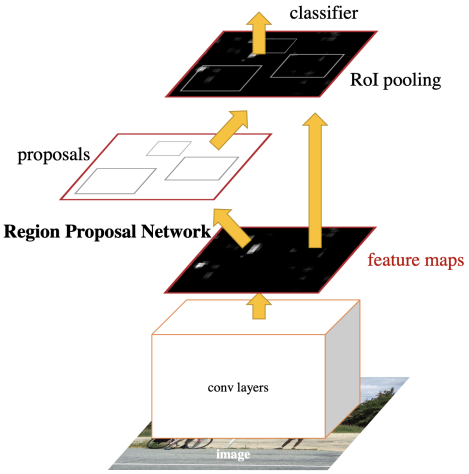
Object detection algorithms have evolved significantly over the years. One of the earliest approaches to object detection was the R-CNN (Region-based Convolutional Neural Network) algorithm. It works by generating a large number of region proposals, then classifying each region proposal as containing an object or not. R-CNN was groundbreaking in its time, but it had some drawbacks, such as being computationally expensive and slow.

To address these limitations, the Faster R-CNN algorithm was developed. This algorithm uses a region proposal network to generate region proposals, which are then classified and refined by a detection network. The region proposal network greatly reduces the number of proposals generated, making Faster R-CNN much faster than its predecessor. Faster R-CNN is still widely used today and is one of the most popular object detection algorithms.

Another widely used object detection algorithm is YOLO (You Only Look Once). YOLO takes a different approach from R-CNN and Faster R-CNN. It treats object detection as a regression problem, where the algorithm directly predicts the bounding boxes and class probabilities for each object in the image. This makes YOLO extremely fast and efficient, as it only needs to process the image once to detect all objects. YOLO also has high accuracy and can detect small objects, making it a popular choice for real-time apps.

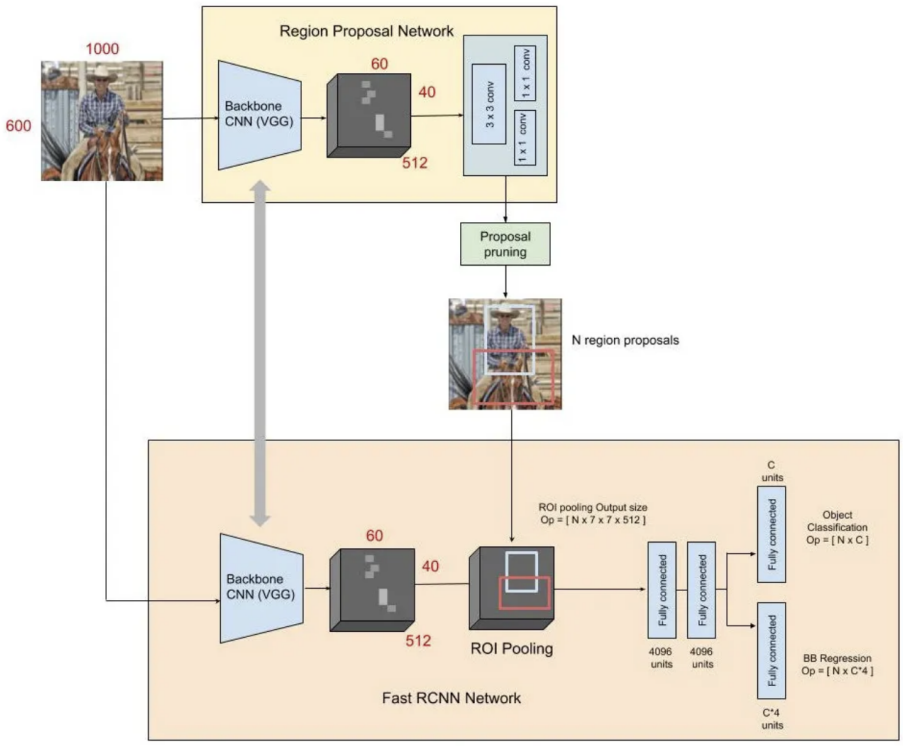
In our case we will work with the ***Faster R-CNN model*** to predict bounding boxes and labels for cell types.

The Faster R-CNN architecture consists of two main components: a region proposal network (RPN) and a Fast R-CNN network. These components work together to generate region proposals and perform object detection and classification.



Architecture of Faster R-CNN. Faster R-CNN is a single, unified network for object detection. The RPN module serves as the ‘attention’ of this unified network.

The first module is a deep fully convolutional network that extracts features from the input image and proposes regions called Region Proposal Network (RPN), and the second module is the Fast R-CNN detector that uses the proposed regions. The entire system is a single, unified network for object detection. Using the recently popular terminology of neural networks with ‘attention’ mechanisms, the RPN module tells the Fast R-CNN module where to look.



The RPN takes an input image and produces a set of object proposals. It operates on a feature map that is generated by a convolutional neural network (CNN) such as ResNet or VGG16. The RPN takes an image as input and outputs a set of object proposals, which are candidate bounding boxes that may contain objects of interest. The RPN slides a small convolutional filter over the feature map, generating a set of anchors at each spatial location. Each anchor is associated with a set of objectness scores, which indicate the likelihood that the anchor contains an object, as well as a set of bounding box regressors that are used to refine the anchor's position and size.

The RPN uses a loss function that encourages high objectness scores for positive anchors (those that overlap with ground-truth objects) and low scores for negative anchors (those that overlap only with background). The RPN also learns to predict the bounding box offsets for positive anchors, so that the anchors can be refined to better match the ground-truth objects.

By generating a set of candidate object proposals in an efficient manner, the RPN allows the Faster R-CNN model to focus on a small subset of the image regions that are most likely to contain objects, while ignoring the vast majority of the image that does not.

The Fast R-CNN network shares CNN with RPN and takes the output of the RPN and performs object detection and classification. It consists of a series of fully connected and convolutional layers that take the region proposals generated by the RPN and classify them into specific object categories. The Fast R-CNN network also performs bounding box regression to refine the location of the objects in the image.

The entire Faster R-CNN architecture is trained end-to-end using a multi-task loss function that combines the classification and bounding box regression losses from both the RPN and Fast R-CNN network. During training, the RPN and Fast R-CNN networks are optimized jointly to improve the accuracy of the object detection system.

Overall, the Faster R-CNN architecture is highly effective for object detection tasks due to its ability to generate high-quality region proposals and accurately classify and locate objects in images.
Advantages:

1. Faster R-CNN is more accurate than its predecessor, R-CNN, and Fast R-CNN. This is because it uses a two-stage detection process that generates region proposals before performing object classification and bounding box regression. This approach significantly reduces false positives and false negatives, resulting in more accurate object detection.
2. Faster R-CNN is faster than Fast R-CNN. This is because it uses a region proposal network to generate region proposals, reducing the number of proposals and speeding up the detection process.
3. Faster R-CNN is highly flexible and can be customized for specific applications. This is because it uses a modular architecture that allows for easy integration of different components, such as different backbone networks or feature extractors. This flexibility makes it easy to adapt to different datasets and environments, making it a versatile tool for object detection.

Faster R-CNN also has some disadvantages that should be considered when choosing an object detection system:

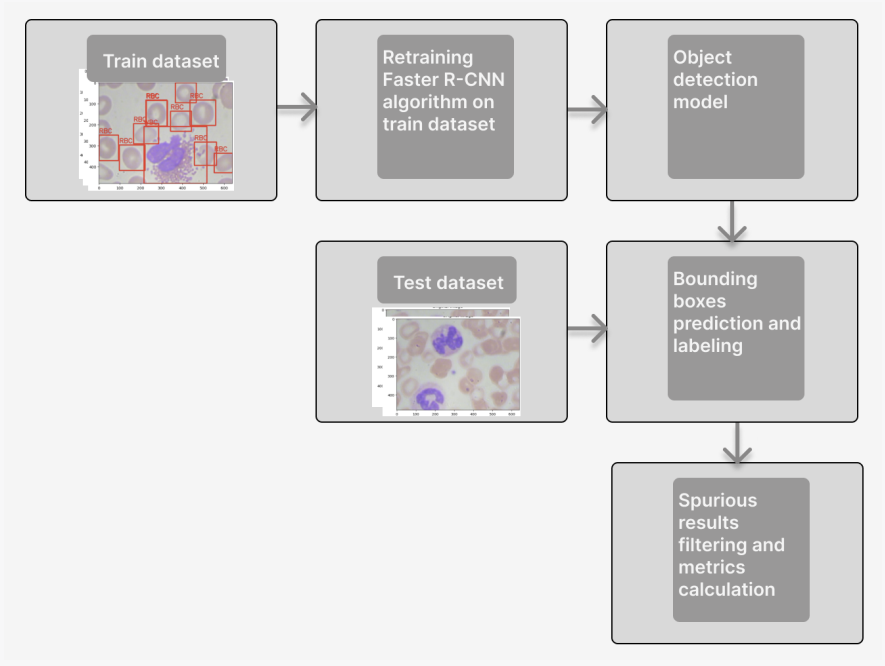
1. Computationally Expensive: Faster R-CNN requires significant computation resources, particularly during the training phase. This can make it difficult to train models on large datasets or with limited computing resources.
2. Slow Inference: Although Faster R-CNN is faster than previous object detection algorithms such as R-CNN and Fast R-CNN, it can still be relatively slow during inference. This can be a disadvantage when real-time object detection is required.
3. Limited Object Scale Detection: While Faster R-CNN is effective at detecting objects of different sizes, it can struggle with very small or very large objects. This can be a disadvantage in applications where small or large objects are common.

4. Sensitivity to Hyperparameters: The performance of Faster R-CNN can be sensitive to the choice of hyperparameters, particularly during the training phase. This can require significant experimentation and tuning to achieve optimal performance.
5. Limited Robustness to Adversarial Examples: Like many deep learning algorithms, Faster R-CNN can be vulnerable to adversarial examples, which are images that have been carefully crafted to fool the algorithm into misclassifying objects. This can be a disadvantage in applications where security is a concern.

Process overview.

For this project we'll use the Faster R-CNN model from the torchvision module. The pretrained Faster R-CNN model provided by PyTorch's torchvision package is pre-trained on the COCO dataset using ResNet50-FPN (Feature Pyramid Network) as its backbone architecture. COCO dataset consists of more than 330,000 images of 80 different object categories, making it one of the largest and most widely used datasets for object detection tasks.

1. Dataset preparations. In our case we need to create a custom dataset class: The annotated dataset needs to be wrapped in a custom PyTorch dataset class. This class should define how to load the images and annotations, apply any necessary transformations (in our case converting the image to a tensor ensures that the image data is normalized and scaled appropriately for use with the neural network), and return the data as PyTorch tensors. For the test dataset we won't create PyTorch tensors for annotations.
2. In a pre-trained model we will need to change the number of classes we are predicting for. In our case it is 3 cell-types plus additional class for background detection with index 0. Pre-trained model should be fine-tuned on the dataset using the PyTorch training pipeline. During fine-tuning, the weights of the pre-trained model are adjusted to better fit the new dataset, resulting in a model that can accurately detect objects in the new dataset.
3. Save fine-tuned model for further usage with all trained parameters.
4. Run PyTorch evaluation step of the model on the prepared test dataset. The output is a list of dictionaries with PyTorch tensors for bounding boxes, labels and confidence scores for each of them per image.
5. Define metric for model evaluation and ways to eliminate wrong predictions. mAP will be the main metric we'd like to maximize. For results filtering we will use different levels of score thresholding depending on cell type and also we will use non-maximum suppression function to remove redundant detections of the same object.



Model was retrained during 5 epochs and next optimization parameters: lr=0.001, momentum=0.9. The lr (learning rate) argument specifies the step size at each iteration of the optimization algorithm. It determines how quickly the optimizer moves in the direction of the gradient, and controls the speed of convergence during training. A higher learning rate allows for faster convergence but may lead to overshooting the optimal weights, while a lower learning rate may result in slower convergence or getting stuck in local minima.

The momentum argument specifies the momentum factor for the optimizer, which helps to speed up convergence by damping oscillations in the gradient descent path. It allows the optimizer to keep moving in the previous direction of the gradient, which can be useful when the gradient changes direction frequently. A higher momentum helps to smooth out the path and reach the optimal solution faster, but too high of a momentum can cause overshooting.

Model training phase compute next losses of the model:

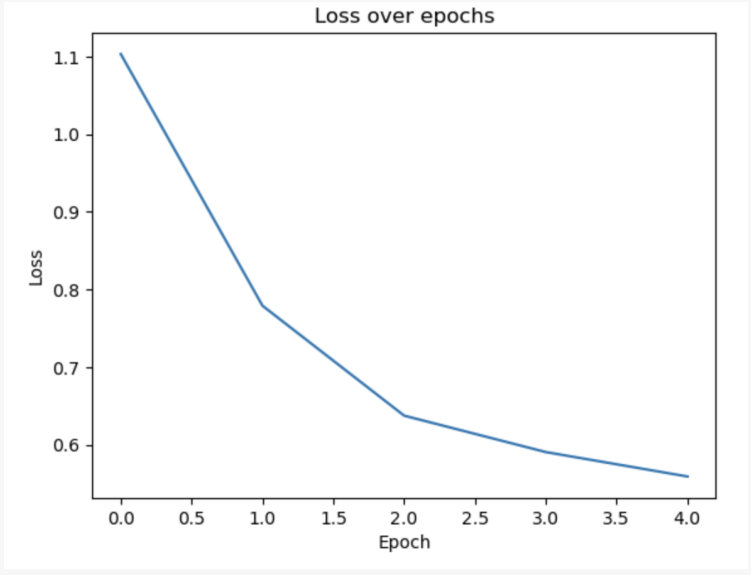
1. loss_classifier: This is the classification loss, which is a measure of how well the model predicts the class of each object in the image. This loss is calculated using cross-entropy loss function and is only applied to the positive anchors in the RPN (Region Proposal Network). The loss penalizes the difference between the predicted class probabilities and the true class labels for each object.
2. loss_box_reg: This is the regression loss, which is a measure of how well the model predicts the bounding box coordinates for each object in the image. This loss is calculated using smooth L1 loss function and is only applied to the positive anchors in the RPN. The loss penalizes the difference between the predicted bounding box coordinates and the true bounding box coordinates for each object.
3. loss_objectness: This is the objectness loss, which is a measure of how well the RPN predicts the presence or absence of an object in each anchor box. This loss is calculated

using binary cross-entropy loss function and is applied to both positive and negative anchors in the RPN. The loss penalizes the difference between the predicted objectness scores and the true objectness labels for each anchor.

4. loss_rpn_box_reg: This is the regression loss for the RPN bounding box coordinates, which is a measure of how well the RPN predicts the bounding box coordinates for each anchor. This loss is calculated using smooth L1 loss function and is only applied to the positive anchors in the RPN. The loss penalizes the difference between the predicted RPN bounding box coordinates and the true RPN bounding box coordinates for each anchor.

Due to the small size of the dataset and computational limitations of the working computer, the DataLoader object was set to process 4 images in one iteration before back propagation and model parameters update.

Each epoch loss is the sum of 4 losses described above. During training, the model uses stochastic gradient descent (SGD) to minimize this total loss and update the model parameters.



We can see that loss continues to drop over epochs and the model could be trained further, but the number of trained parameters in default architecture is 42.7 millions and it required 676.31 min to train the model over 5 epochs in total.
Model results were analyzed on a test dataset which contains 60 images.

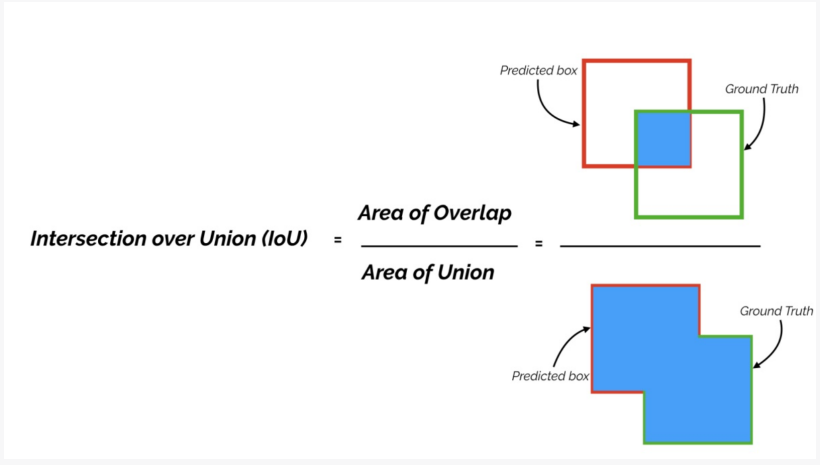
Model performance metrics.

mAP - mean average precision was chosen as the main metric for the model performance analysis. In order to understand mAP, we need to first understand some of the key concepts that it builds upon:

- Confusion Matrix: A confusion matrix is a table that summarizes the performance of a classification model by showing the number of true positive (TP), true negative (TN),

false positive (FP), and false negative (FN) predictions made by the model on a set of test data.

- IoU: IoU, or Intersection over Union, is a measure of overlap between two bounding boxes. It is calculated as the ratio of the area of the intersection of two boxes to the area of their union. IoU is commonly used to determine whether a predicted bounding box is a true positive or false positive.



- Precision: Precision is a measure of how many of the positive predictions made by the model are actually correct. It is calculated as the ratio of true positives to the sum of true positives and false positives.

Precision=$TP/(TP + FP)$

- Recall: Recall is a measure of how many of the actual positive examples in the test data are correctly identified by the model. It is calculated as the ratio of true positives to the sum of true positives and false negatives.

Recall=$TP/(TP + FN)$

TP+FN is a number of actual positive examples.

To calculate mAP, we first calculate the precision and recall values for each cell type at IoU threshold of 0.5. We then calculate the average precision for each cell type by computing the area under the precision-recall curve. In our case we will use the trapezoid rule for approximation of calculation of the area under the precision-recall curve. The trapezoid rule approximates the area under a curve by dividing the curve into a series of trapezoids and summing their areas. In the context of mAP, we can use the trapezoid rule to compute the area under the precision-recall curve for each class by dividing the curve into a series of segments and summing their areas.

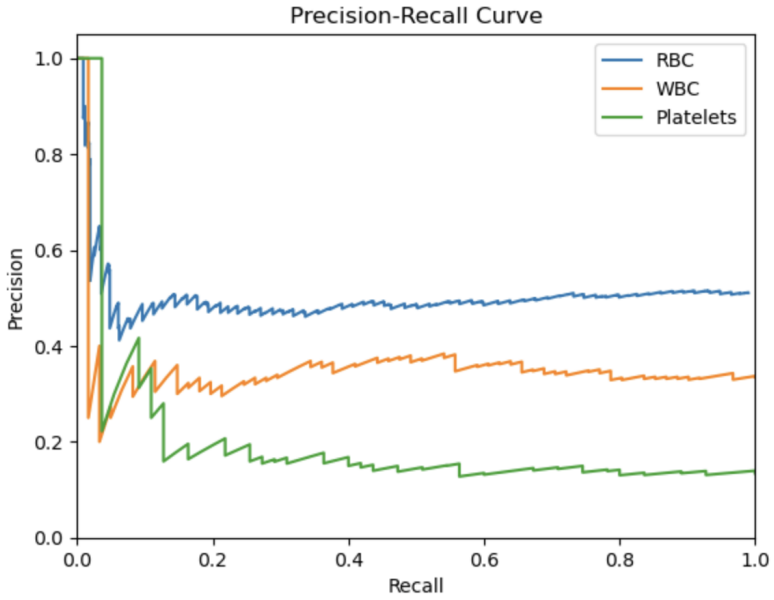
Finally, we compute the mean average precision by taking the mean of the average precision values across all classes.

In order to calculate precision and recall values, we use a set of bounding box predictions and ground truth bounding boxes for each class. We compute the IoU between each predicted box and each ground truth box, and assign the predicted box to the ground truth box for the same image with the highest IoU. We then consider a predicted box a true positive if its IoU with the assigned ground truth box is above a 0.5 threshold, and a false positive otherwise.

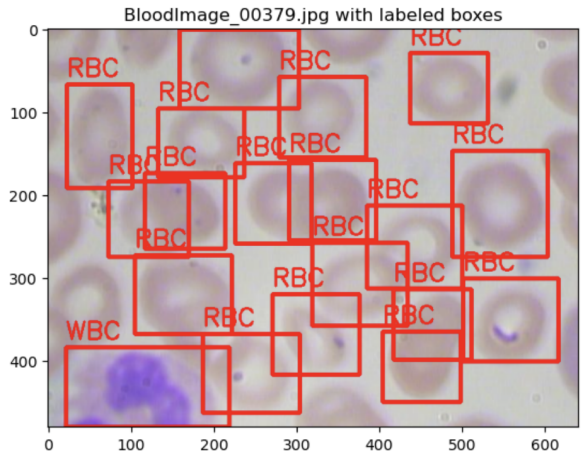
By computing precision and recall values and averaging the results, mAP provides a more comprehensive measure of model performance than simply looking at precision or recall alone. It takes into account the tradeoff between precision and recall and provides a more robust evaluation of object detection models.

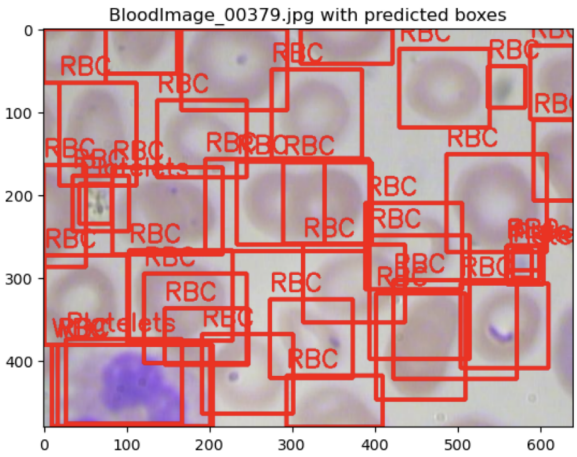
Unfiltered results

mAP=0.35
AP 'RBC' = 0.5,
AP 'WBC’ = 0.35,
AP 'Platelets' = 0.2

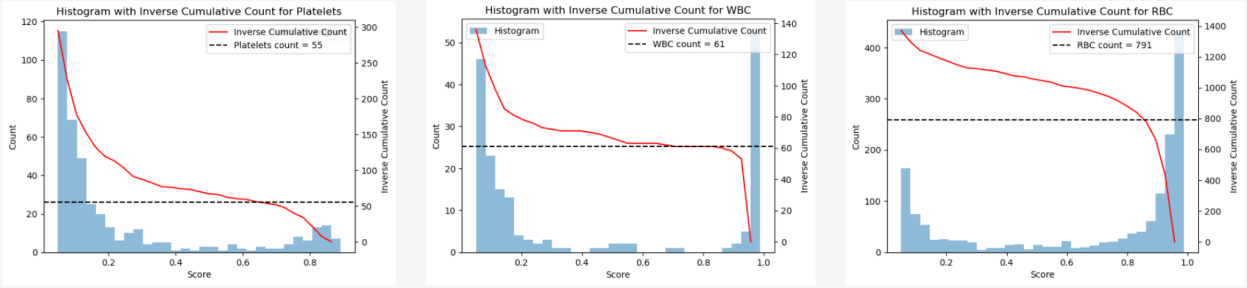


Let’s check how predicted bounding boxes look like:





We can check histogram plots for prediction results for each cell type:



As we can see, the model predicts different labeled boxes with different levels of confidence (score) for the same image area. We need to set thresholds for scores and intersected boxes to maximize evaluation metric mAP.

As we can see our predictions contain much more labeled boxes for all 3 classes. We need to filter out predictions in order to obtain more accurate results. We can use next approaches:

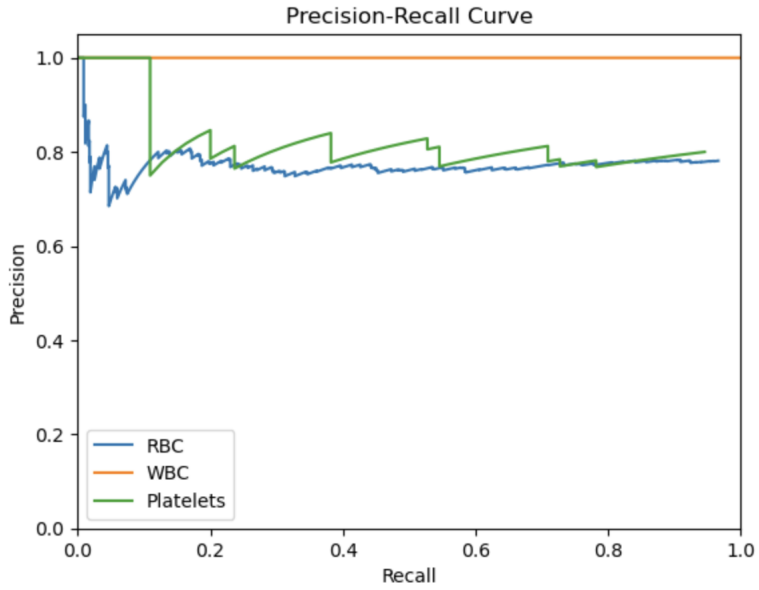
1. Set thresholds for scores to filter out boxes for each class:
    - For 'RBC' it can be in range of [0.5,0.7] - it will give around 1000 predicted boxes of this class
    - For 'WBC' it can be in range of [0.7,0.8] - it will give around 61 predicted boxes of this class
    - For 'Platelets' it can be in range of [0.4, 0.6] - it will give around 60 predicted boxes of this class
2. Use NMS approach to filtered boxes: NMS (Non-Maximum Suppression) is a technique used in object detection tasks to eliminate overlapping bounding boxes for the same object in an image. NMS Threshold will be set to remove boxes with lower score if IoU is higher than NMS Threshold

Filtered results

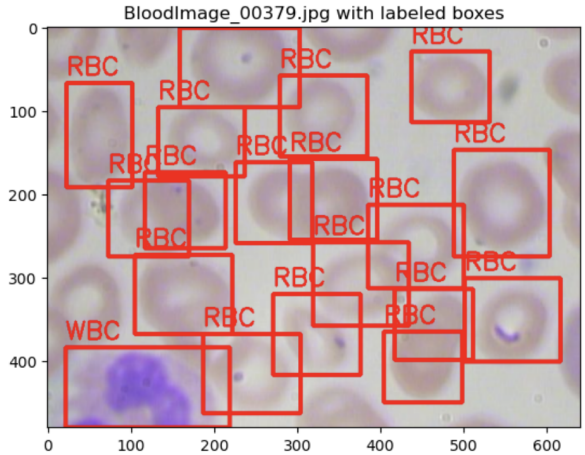
The best results were found with next parameters:

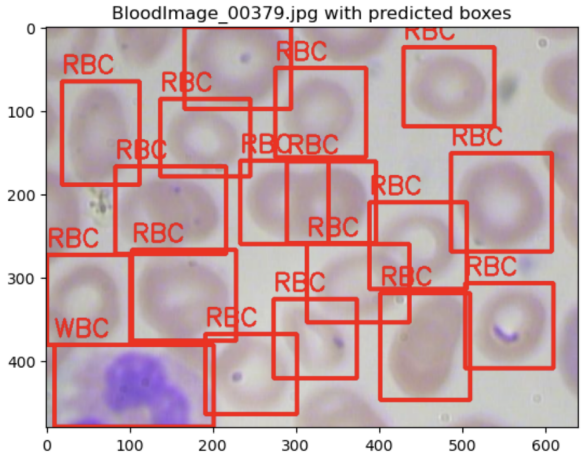
- Thresholds to filter predictions by score:
  Platelets: 0.55
  RBC: 0.70
  WBC: 0.80
- NMS threshold: 0.45

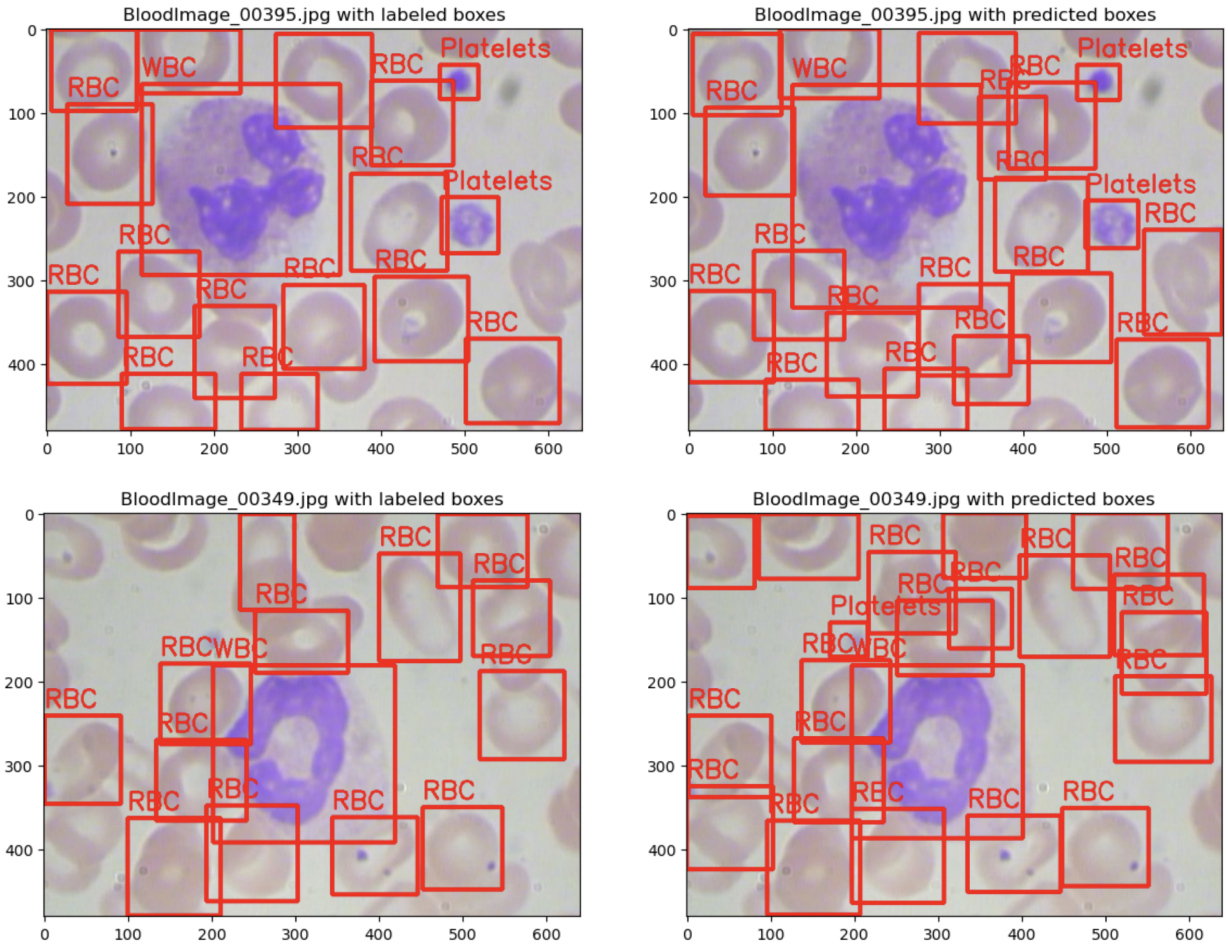
mAP = 0.84
AP 'RBC' = 0.7,
AP 'WBC' = 1.0,
AP 'Platelets' = 0.78



Let's check how predicted bounding boxes look like:







Filtering results with different levels of threshold significantly improved average precision for each class and mAP in general.

Model with frozen backbone CNN part.

As we could see training the whole model can be really time consuming, let's fine-tune only parts of the Faster R-CNN model that are related to bounding box prediction and labeling without training the backbone part that contains convolutional layers. Total time training for 5 epochs took around 100 min that significantly reduced time for training. But let's check the results.

Results without predictions filtering are:
mAP = 0.19
AP 'RBC' = 0.34,
AP 'WBC' = 0.1,
AP 'Platelets' = 0.13

After filtering:

- Thresholds by cell type:
  Platelets: 0.55
  RBC: 0.70
  WBC: 0.75
- NMS threshold: 0.45

mAP = 0.7
AP 'RBC' = 0.73,
AP 'WBC’ = 0.81,
AP 'Platelets' = 0.55

Summary

The Faster R-CNN model was trained on a dataset of 300 blood smear images, and the model achieved an mAP of 0.84 on a test set. This result suggests that the model is performing reasonably well in detecting blood cells and their classification in the images.
We could see that the fully fine-tuned model performed better than the model with frozen weights for the backbone CNN part. In that case mAP decreased to 0.7

Metric	Fully retrained Faster R-CNN		Frozen parameters for backbone CNN part	
Time to train	676.31 min		100 min	
Results filtering	Unfiltered results	Thresholds by cell type: Platelets: 0.55 RBC: 0.70 WBC: 0.8 NMS threshold: 0.45	Unfiltered results	Thresholds by cell type: Platelets: 0.55 RBC: 0.70 WBC: 0.75 NMS threshold: 0.45
mAP	0.35	0.84	0.19	0.7
AP ‘RBC’	0.5	0.7	0.34	0.73
AP 'WBC’	0.35	1.0	0.1	0.81
AP 'Platelets'	0.2	0.78	0.13	0.55

The model's performance can be further analyzed by looking at the AP (average precision) values for each class of blood cell. The AP for the 'WBC' class was 1.0, indicating that the model is very accurate at detecting white blood cells. As we remember from EDA most images have 1 white blood cell and it is bigger in size than other cells.

The AP for the 'RBC' class was 0.7, suggesting that the model is less accurate at detecting red blood cells. Red blood cells have the most number in each image and lots of them are overlapping that makes it harder to recognize all of them. Possible analysis can be related to different IoU thresholding. It may also be beneficial to perform an analysis of the false positives and false negatives generated by the model as the model recognised a higher number of red blood cells than was labeled manually.. By examining the images where the model incorrectly identified or missed blood cells, it may be possible to identify areas where the model needs further improvement. For example, if the model consistently misses red blood cells in certain parts of the image, it may be necessary to adjust the model's architecture or training data to better capture those features.
Finally, the AP for the 'Platelets' class was 0.78, indicating that the model is reasonably accurate at detecting platelets. Platelets are small blood cells and appear in images from 1 to 4 times. Additional analysis for prediction filtering by threshold can be done. For example, we can use K-nearest neighbor (KNN) and a small threshold of intersection over union (IOU) in each platelet case to avoid their double counting in some cases or mistaken filtration out of the predictions..
To improve the performance of the model, several steps can be taken.

- One approach is to increase the size of the training dataset, as more data may help the model learn to detect blood cells more accurately.
- Another option is to increase the number of the epochs the model is trained over as we could see potential decrease of loss. Additional GPUs can be used from external services.
- Additionally, data augmentation techniques, such as rotation and flipping, can be applied to increase the diversity of the training data and improve the model's generalization performance.

Overall, the performance of the Faster R-CNN model on the blood smear dataset is promising, but there is still room for improvement. By analyzing the model's performance and fine-tuning the training process, it may be possible to achieve even higher accuracy in detecting blood cells.

References:

1. https://arxiv.org/pdf/1506.01497.pdf
2. https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/object_localization_and_detection#faster-rcnn
3. https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46
4. https://towardsdatascience.com/evaluating-performance-of-an-object-detection-model-137a349c517b