Blood Cell Detection and Classification

Varun Shaji

1. Models chosen

Two models – *Faster R-CNN* and *YOLO* was chosen to detect and classify blood cells. Both models are considered state of the art and used regularly in object detection tasks.

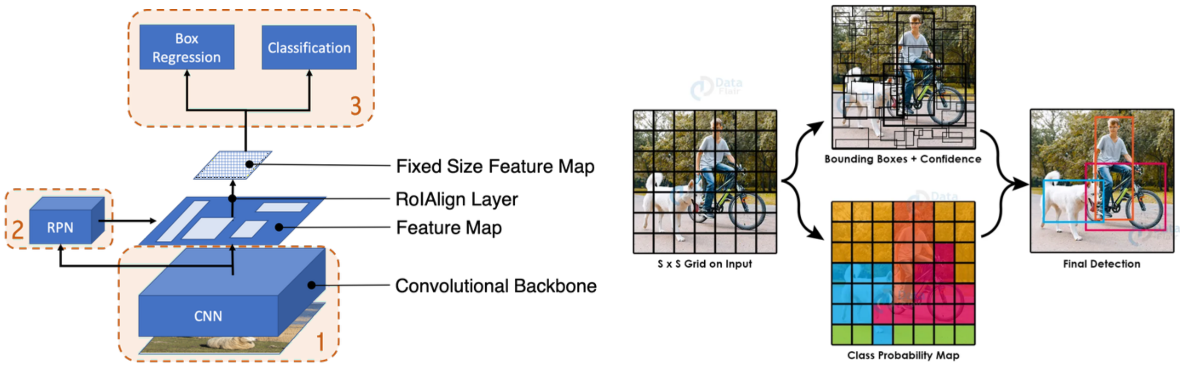

Fig 1. [Left] Architecture of Faster RCNN showing Region Proposing Network. **[Right]** Illustration of how YOLO network detects objects

Faster RCNN is an improvement on region based convolutional neural networks – RCNNs (2013) and Fast RCNNs (2014), introducing a Region Proposal Network (RPN) that applies the concept of attention for object detection; it provide regions of interest within the image to the CNN backbone to focus on [1]. The original paper uses VGG16 CNN backbone, but a ResNet50 backbone was chosen to train in this case. Faster RCNN uses cross-entropy for foreground and background loss, and L1 regression for bounding box coordinates.

You only look once (YOLO), first introduced in 2015, tries to overcome certain drawbacks with existing object detection models. It is a single end-to-end deep convolutional network that predicts the bounding boxes and as well as the class probabilities for these boxes [2]. Targeted for real-time processing, the model architecture is simpler and quicker than Faster RCNNs. YOLO uses L2 loss for bounding box regression and classification. A limitation with this model is that it struggles with small objects close to each other within the image, due to spatial constraints of the algorithm. YOLOv3, a version of the original YOLO model has been implemented [3].

In this study, both models use weights pre-trained on the COCO dataset [4] rather than training from scratch to significantly shorten the training time.

2. Dataset preparation and Code summary

The Blood Cell Detection Dataset contains 100 image files of *png* format in the *images* folder and a list of locations and labels of objects within *annotation.csv*. *visualise.py* can be used to visualise image samples in the folder. The number of RBCs and WBCs within the dataset are highly imbalanced with 2237 RBCs and 103 WBC. The images and their corresponding annotations are split into train, valid and test data in the ratio 70:20:10. *Table 1* illustrates the distribution of labels.

Table 1. Distribution of images and labels in the train, valid and test split

		Train	Valid	Test
images		68	22	10
labels	RBC	1534	485	218
	WBC	70	24	9

2.1. Faster RCNN

Faster RCNN model accepts annotations in the *Pascal VOC xml* format, with each image having one xml file containing locations *[xmin, ymin, xmax, ymax]* and labels of all objects within the image. The model hyperparameters and data path are declared in *config.py*. The Faster RCNN model is loaded from the torchvision model hub with pre-trained weights in *model.py*. Custom Dataloader defined in *datasets.py* loads the image and annotations, performs random augmentations and feeds it to the training loop defined in *train.py*. *inference.py* and *val.py* detects objects in the test data using the trained model and computes evaluation metrics respectively.

2.2. YOLO

YOLO model accepts annotations in txt format instead of xml files with annotation locations in the format *[x-centre, y-centre, width, height]*. The model configuration is defined in *models/yolo.py, model/yolov3.yml* with the train/test data split and other parameters defined in dataset.yml. Custom DataLoader defined in *utils/dataset.py* loads the image and annotations and feeds it to the training loop in *train.py*. The *utils* folder also contains the implementations of losses, metrics and augmentations. *detect.py* and *val.py* detects objects in the test data using the trained model and computes evaluation metrics respectively.

3. Evaluations metrics

Several different metrics are available in literature to evaluate how well a model performs when trained on a dataset. The most commonly used metric is precision and recall – precision reflects how reliable the model is in classifying samples as positive while recall measures the model's ability to detect positive samples. The higher the recall, the more positive samples detected. If TP, FP and FN are the true positive, false positive and false negative in the detected samples, then:

$$precision = \frac{TP}{TP+FP} \qquad recall = \frac{TP}{TP+FN}$$

It is to be noted that the definition of a true positive sample for a given trained model depends on the choice of confidence threshold and IoU threshold for an object detection task. Confidence score is the probability that a predicted bounding box contains an object; usually predicted by a classifier. IoU (Intersection over Union) metric evaluates the degree of overlap between the ground truth and prediction – higher the IOU, higher the closeness of the predicted bounding box to the ground truth. If B_{gt} and B_p are the ground truth bounding box and predicted bounding box, then:

$$IOU = \frac{area(B_{gt} \cap B_p)}{area(B_{gt} \cup B_p)}$$

With regards to this dataset, it would be useful to benchmark our models with metrics used to evaluate COCO dataset [5] (a well-known object detection dataset) - Average Precision (AP) and Mean Average Precision (mAP). AP conveniently summarises the precision-recall curve into one single metric; it is defined as the area under the interpolated precision-recall curve for each class. Since there are multiple classes in an object detection task, the mean of AP for all classes is computed as the mean-Average Precision (mAP). The mAP can be calculated for various confidence thresholds and IOU thresholds to understand the models' performance on the dataset.

4. Experiment and Results

2.1. Data pre-processing

The training data was augmented during the training loop to improve generalisability and increase the available training data. Random transforms of horizontal flip, and 90 degree rotations were applied to the training data with 0.5 probability using the *albumentations* library – this allows for bounding box and labels to be modified automatically to reflect the image transformations. Additional augmentations such as random crop, blurring, changing to grayscale was considered, but was not applied for simplicity. These can be added in the training process to improve the model if the evaluation metrics is low.

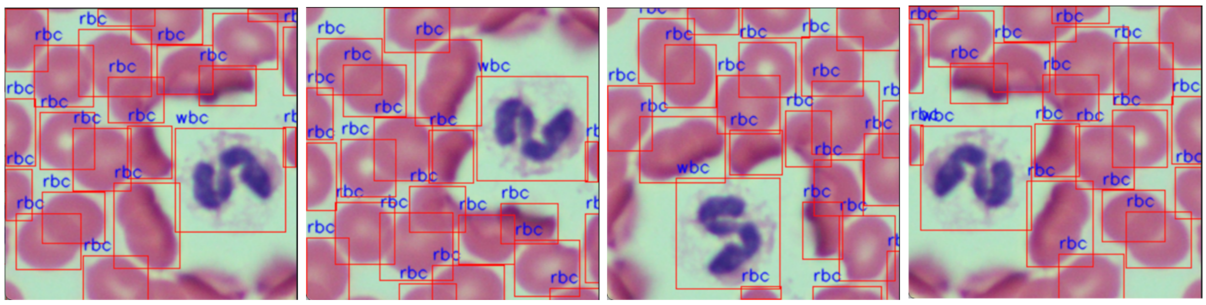
Fig 2. Horizontal flip and 90 deg. rotations of *image-12.png*

2.2. Model settings

Table 2. Model Hyperparameters used for training

	Faster R-CNN	**YOLO**
Initial LR	0.001	0.01
Momentum	0.9	0.937
Weight Decay	0.0005	0.0005
Optimiser	SGD	SGD

Default values used to train the COCO dataset for both Faster RCNN and YOLO was used to train the blood cells dataset. SGD optimiser was preferred over Adam as it is known to perform better in image classification tasks [6]. A batch size of 4 was used for training and validation.

2.3. Experiment Results and Observation

The Faster RCNN model was trained for 25 epochs, and halted when the training and validation loss plateaued. YOLO model was trained for 300 epochs. Both took similar times to train despite the difference in epochs.

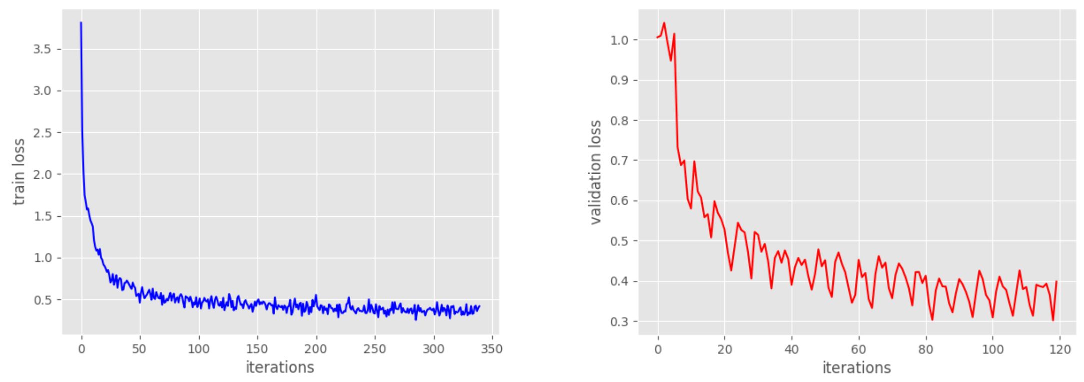

Fig 3. Training and validation loss during Faster RCNN model training

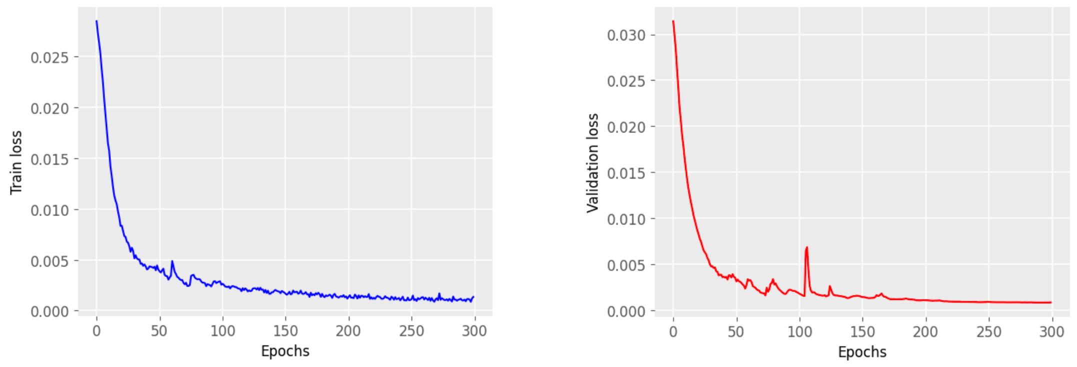

Fig 4. Training and validation loss during YOLO model training

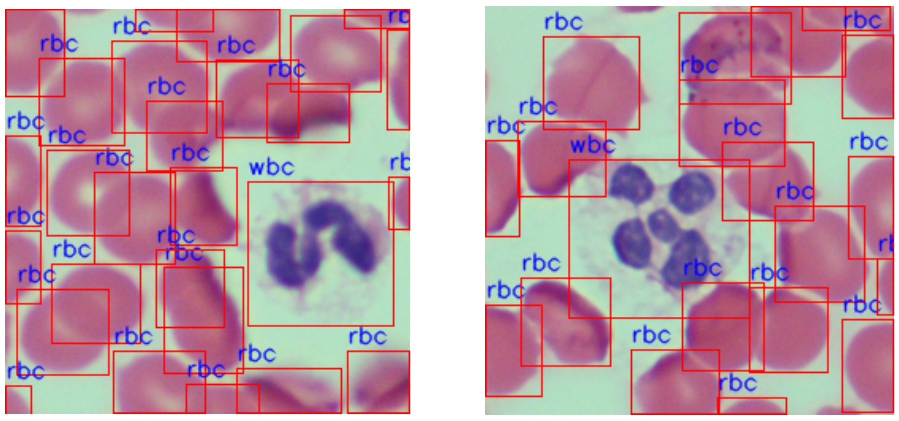

Fig 5. Detections from trained Faster RCNN model

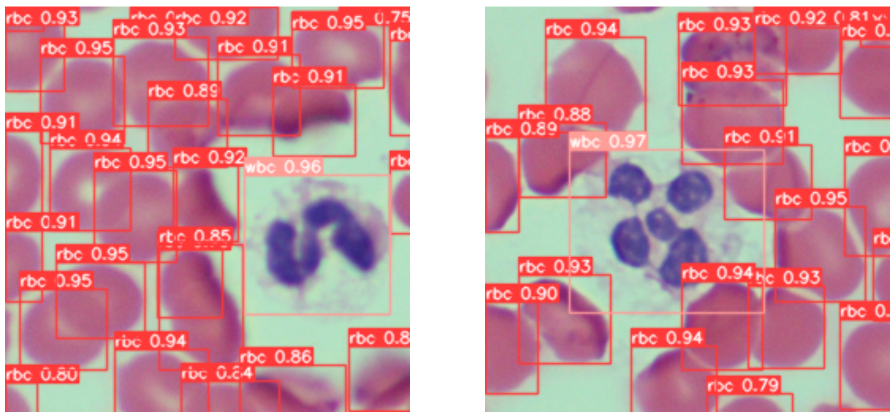

Fig 6. Detections from trained YOLO model

Table 3. Results of the validation and test data of all models when *confidence score threshold is* ***0.50***

		All			Red Blood Cells			White Blood Cells		
Type	**Split**	**P**	**R**	**mAP@0.5**	**P**	**R**	**mAP@0.5**	**P**	**R**	**mAP@0.5**
Faster R-CNN	Test	0.920	0.965	0.978	0.921	0.963	0.957	0.900	1.000	1.000
	Valid	0.943	0.998	0.965	0.942	0.975	0.972	0.958	0.958	0.958
YOLO	Test	0.975	0.984	0.987	0.950	0.968	0.979	1.000	1.000	0.995
	Valid	0.984	0.960	0.987	0.969	0.961	0.980	1.000	0.958	0.979

Table 4. Results of the validation and test data of all models when *confidence score threshold is* ***0.75***

		All			Red Blood Cells			White Blood Cells		
Type	**Split**	**P**	**R**	**mAP@0.5**	**P**	**R**	**mAP@0.5**	**P**	**R**	**mAP@0.5**
Faster R-CNN	Test	0.927	0.960	0.977	0.929	0.959	0.953	0.900	1.000	1.000
	Valid	0.965	0.971	0.963	0.963	0.971	0.968	1.000	0.958	0.958
YOLO	Test	0.979	0.975	0.981	0.958	0.949	0.968	1.000	1.000	0.995
	Valid	0.985	0.954	0.975	0.971	0.951	0.971	1.000	0.958	0.979

Table 5. Results of the validation and test data of all models when *confidence score threshold is* ***0.90***

		All			Red Blood Cells			White Blood Cells		
Type	**Split**	**P**	**R**	**mAP@0.5**	**P**	**R**	**mAP@0.5**	**P**	**R**	**mAP@0.5**
Faster R-CNN	Test	0.960	0.956	0.978	0.958	0.954	0.949	1.000	1.000	1.000
	Valid	0.974	0.954	0.916	0.972	0.958	0.956	1.000	0.857	0.857
YOLO	Test	0.994	0.853	0.922	0.987	0.706	0.850	1.000	1.000	0.995
	Valid	0.997	0.835	0.916	0.994	0.711	0.854	1.000	0.958	0.979

Both models performs well in detection and classification of blood cells, with a high precision (>0.92), recall (>0.95) and mAP (>0.95) at the minimum confidence threshold of 0.5. RBCs are properly detected and identified, including overlapping and partially visible ones. Even with the high class imbalance, the model detects WBCs as accurately as RBCs. This could be explained due to the easily identifiable characteristics of the WBCs such as the larger size and different cell colour. As the confidence threshold increases, the recall for both classes decreases as expected, since the proportion of positive samples detected with higher confidence scores decreases. YOLO performs slightly better than Faster RCNN (~1-2%) in all metrics for the confidence threshold of 0.5 and 0.75. But YOLO is able to detect WBCs better with higher confidence, while Faster RCNN performs better with RBCs – this is seen in Recall and mAP of *Table 5.* Models perform better on valid dataset than test dataset as expected since we referred to validation metrics during training to decide when to stop.

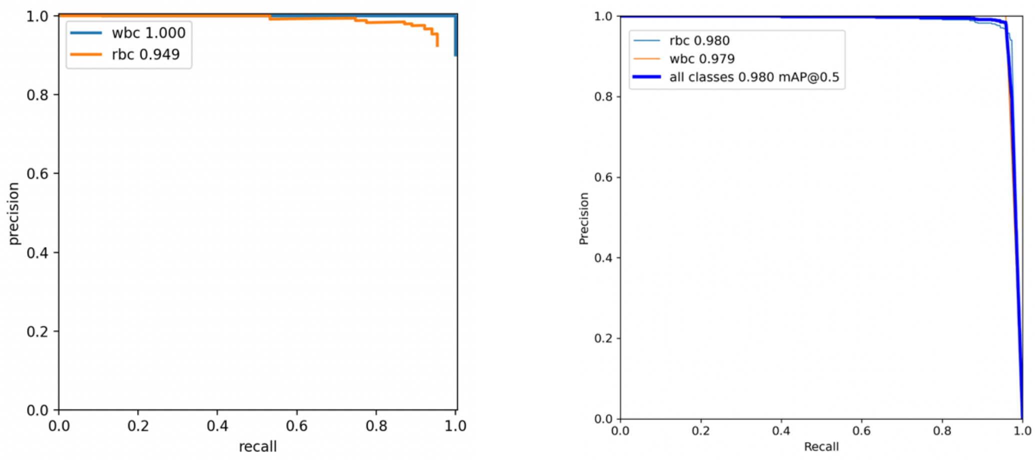

Fig 7. Precision-Recall curves at conf. threshold 0.5 IOU 0.7 for **[Left]** Faster RCNN **[Right]** YOLO

References

[1] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” arXiv.org, 06-Jan-2016. [Online]. Available: https://arxiv.org/abs/1506.01497.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *arXiv.org*, 09-May-2016. [Online]. Available: https://arxiv.org/abs/1506.02640.

[3] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv.org*, 08-Apr-2018. [Online]. Available: https://arxiv.org/abs/1804.02767.

[4] “Common objects in context,” *COCO*. [Online]. Available: https://cocodataset.org/#home.

[5] “COCO: Detection Evaluation,” *COCO*. [Online]. Available: https://cocodataset.org/#detection-eval.

[6] A. Gupta, R. Ramanath, J. Shi, and S. S. Keerthi , “Adam vs. SGD: Closing the generalization gap on Image Classification.” 13th Annual Workshop on Optimization for Machine Learning [Online]. Available: https://www.opt-ml.org/papers/2021/paper53.pdf.

Code References

S.-J. Lee, P.-Y. Chen, and J.-W. Lin, “Complete Blood Cell Detection and counting based on Deep Neural Networks,” MDPI, 14-Aug-2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/16/8140/htm.

“Torchvision object detection finetuning tutorial,” PyTorch Tutorials. [Online]. Available: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html.

R. Padilla, “Object detection metrics,” GitHub. [Online]. Available: https://github.com/rafaelpadilla/review_object_detection_metrics.

Faster RCNN model

S. R. Rath, “Custom object detection using Pytorch Faster RCNN,” DebuggerCafe, 25-Sep-2022. [Online]. Available: https://debuggercafe.com/custom-object-detection-using-pytorch-faster-rcnn/.

YOLO

G. Tanner, “YOLO object detection,” GitHub. [Online]. Available: https://github.com/TannerGilbert/YOLO-Tutorials/tree/master/YOLO-Object-Detection-in-PyTorch.