Final Report

Chao Zhou, Danlei Qian, Ya Liu

1. Introduction

Object detection is a computer vision technique in aim to detect objects of different classes, which locates the presence of an object in an image and draw a bounding box around that object. It has been applied widely in video surveillance, self-driving cars, object or people tracking and many other applications. And it is now well established from a variety of studies, including two stage methods developed from R-CNN, Fast R-CNN, Faster R-CNN to Mask R-CNN, and one stage methods developed from YOLO, YOLOv2, YOLOv3 to YOLOv4.

In this project, we would like to apply one of those well-known algorithm-- Faster R-CNN, which was first proposed in 2015 by Shaoqing Ren et al., to medical industry, with the object of detecting the state of malaria infection from the blood cells images.

2. Data Description

The dataset consists of 929 blood cells image and corresponding information, which indicate different state of malaria infection including "red blood cell", "difficult", "gametocyte", "trophozoite", "ring", "schizont" and "leukocyte". Each image(.png) includes several such cells and is labeled by a text file(.txt) and a bounding box (.json). Figure 1 shows the image without labels(left) and with labels(right).

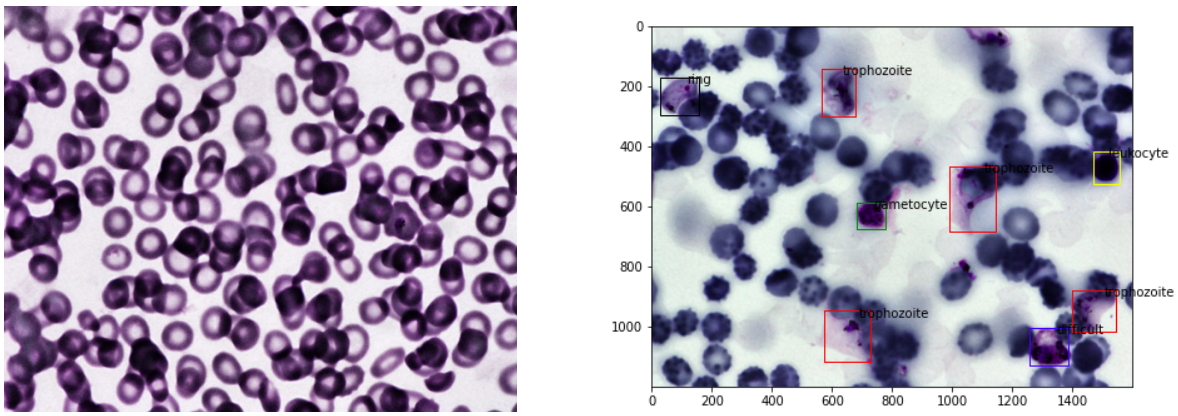


Figure 1. Sample image

3. Deep Learning Network(Faster R-CNN)

Faster R-CNN is designed to detect objects in the images. The data needed to train the mode is a bunch of images, bounding boxes and labels. Figure 2 shows the architecture of Faster R-CNN and it can be divided into four parts: convolutional layers, Region Proposal Networks(RPN), RoIPooling, and Classification. Convolutional layers generate feature maps from images; RPN generates anchors first, and then classify anchor boxes into positive and negative by utilizing Softmax. After that, it uses regression to modify bounding boxes; RoIPooling receives the

feature map from convolutional layers and proposals from RPN and transfers the feature maps of proposals into the same size; In the end, a classifier is used to classify the proposals and a regression is utilized again to modify the bounding box to get precise location.

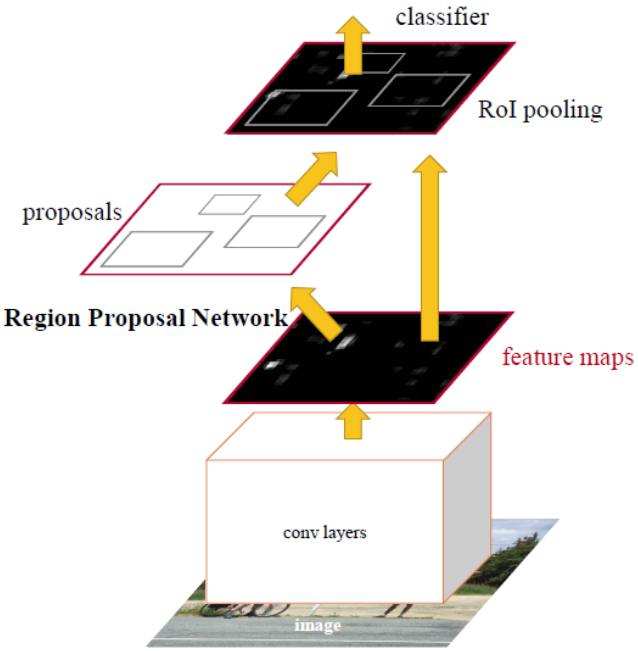


Figure 2. Faster R-CNN Architecture

Figure 3 is the architecture of Faster R-CNN based on VGG-16. We mainly use this example to illustrate the function of Faster R-CNN. On the other hand, we use ResNet as pre-trained model when training because ResNet is bigger than VGG, hence it has more capacity to actually learn what is needed.

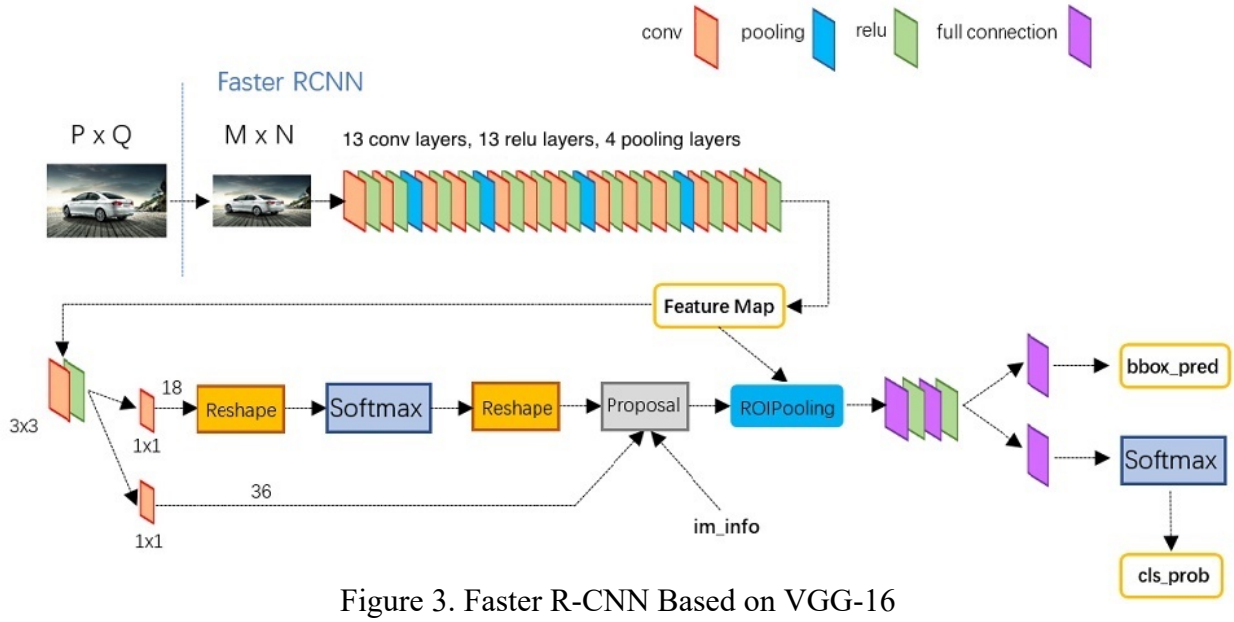


Figure 3. Faster R-CNN Based on VGG-16

3.1 Convolutional layers

As Figure 3 shows, the size of input image is $P \times Q$, and then we resize it to $M \times N$. Then we use convolutional layers to get feature maps. There are many networks we can choose including VGG, ZF, MobileNet, ResNet and so on. One point worth noticing is that we do not use the whole network, and that we get output from the intermediate layer. For example, when using VGG-16, we use the output of conv5/conv5_1 layer. So the network would have 13 convolutional layers, 13 relu layers and 4 pooling layers. Below is one block of VGG-16. We can see that in the Conv2D layer, padding is 1, so the size becomes $(M + 2) \times (N + 2)$. Since the kernel size is 3, the size would still be $M \times N$. However, because of the MaxPooling layer, the size turns to $\frac{M}{2} \times \frac{N}{2}$. Therefore, after the whole convolutional layers, the size of feature maps is $\frac{M}{16} \times \frac{N}{16}$.

$$x = Conv2D(64, (3,3), activation = 'relu', padding = 1, name = 'block1\_conv1')(img\_input)$$
$$x = Conv2D(64, (3,3), activation = 'relu', padding = 1, name = 'block1\_conv2')(x)$$
$$x = MaxPooling2D((2,2), strides = (2,2), name = 'block1\_pool')(x)$$

3.2 RPN

Region Proposal Networks(RPN) is the most crucial part of Faster R-CNN, which makes it outstand other object detection algorithms which uses selective search at that time, and influences the R-CNN till now.

The purpose of RPN is to find the bounding boxes which locate the objects. The input of RPN is feature maps, and the output is a set of rectangular object proposals, each with an objectness score. It mainly does two things. First, it classify proposals into positive and negative. Then it performs a regression to modify the coordinates of bounding boxes.

3.2.1 Generating Anchors

According to the Faster R-CNN paper, it defines anchors as 'an anchor is centered at the sliding window in question and is associated with a scale and aspect ratio'. By default we use 3 scales (8, 16, 32)and 3 aspect ratios($width: height \in \{1:1, 1:2, 2:1\}$), yielding $k = 9$ anchors at each sliding position of feature map. Therefore, if the size of image is $M \times N = 800 * 600$, the total number of anchors generated is $ceil\left(\frac{800}{16}\right) * ceil\left(\frac{600}{16}\right) * 9 = 17100$. It basically covers all shapes and sizes of objects.

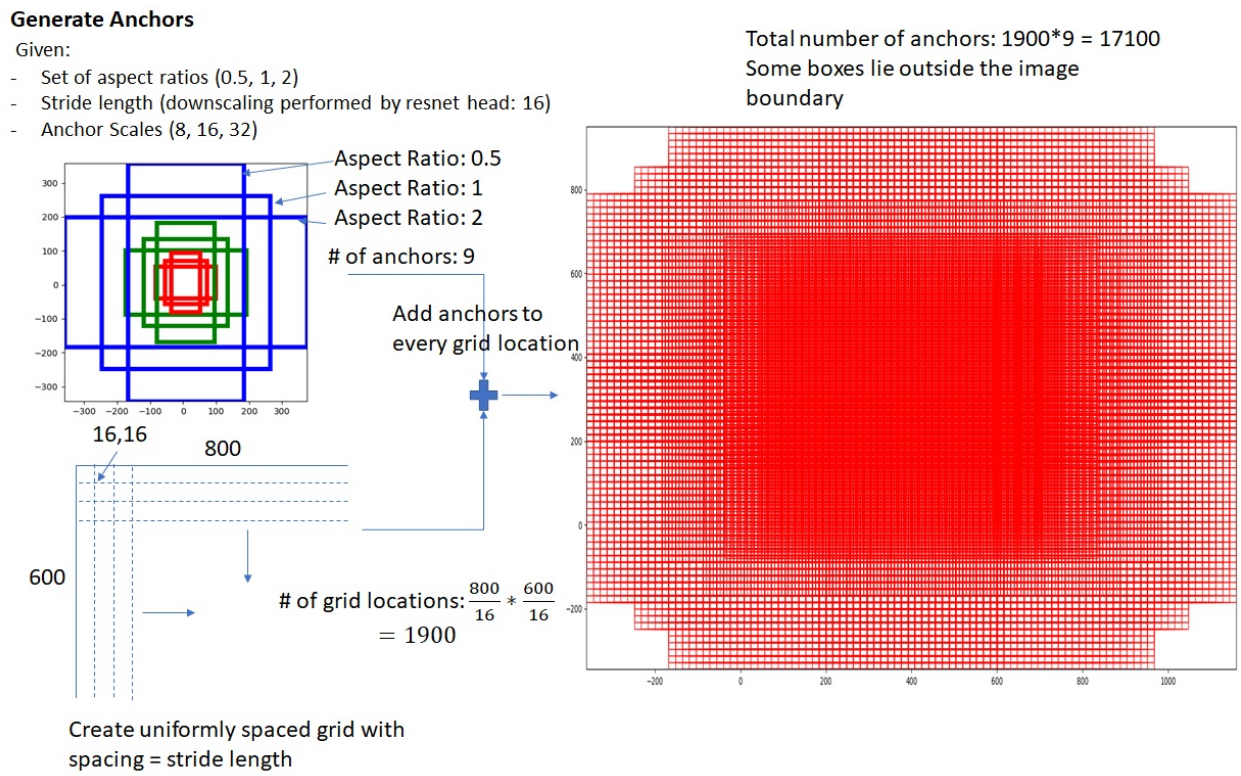


Figure 4. Generating Anchors

3.2.2 Classify Anchors

After generating anchors, we use a classifier to classify anchors into positive or negative. Positive means there is an object in the bounding box, and vice versa. As we define before, the image size is $M \times N = 800 * 600$, the size of feature map is $\frac{M}{16} \times \frac{N}{16}$. Let $W = \frac{M}{16}, H = \frac{N}{16}$. As Figure X shows, we will go through a 1×1 convolutional layers first, and since the $output\_number = 18$, the output size would be $W \times H \times 18$.

To do classification, we need to assign each candidate bounding boxes a positive or negative label to tell if they contain an object. We use a metrics called Intersection over Union(IoU) to measure this. IoU is based on the overlapped area between ground truth and candidate bounding boxes. The formula is demonstrated in Figure 5. A bounding box would be labeled positive if it meets one of the following two criteria: 1) IoU > 0.7, or 2) for a ground truth bounding box, the IoU of a bounding box is the highest one. We would label a bounding box as negative if IoU < 0.3. We will ignore those between 0.3 and 0.7, which would not influence our results since we would have enough candidates.

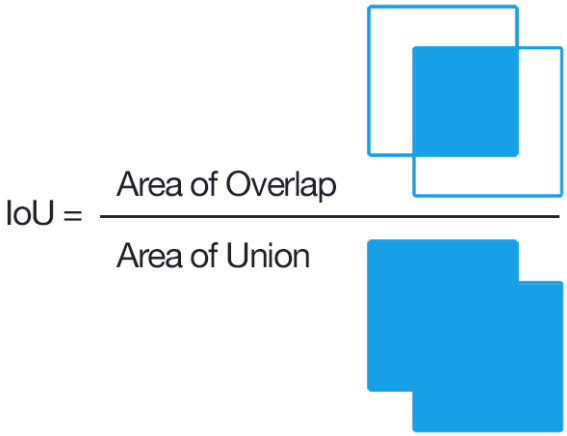


Figure 5. Calculation of IoU

3.2.3 Bounding Box Regression

We use regression in order to transfer bounding box near the ground truth. We use x, y, w, and h denote the box's center coordinates and its width and height, and then we want to find four transformation to change x, y, w and h near the ground truth. The loss function is:

$$Loss = \sum_{i}^{n} \left| t_*{}^{i} - W_*{}^{i} \cdot \phi(A^i) \right|$$

Where $*$ means four transformation, t_* is ground truth, W_* is the weight of transformation that we want to get, and $\phi(A)$ is eigenvector of corresponding feature map.

3.2.4 Loss Function

The loss function of RPN is:

$$L(\{P_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(P_i, P_i{}^*) + \lambda \frac{1}{N_{reg}} \sum_i P_i{}^* L_{reg}(t_i, t_i{}^*)$$

where i is the index of an anchor, P_i is the predicted probability of anchor i being an object(objectness score), $P_i{}^*$ is the corresponding ground truth, 1 for positive, 0 for negative. λ is weight with default value 10, which means the classification and regression are equally important. t_i is predicted bounding box and $t_i{}^*$ is the ground truth. $P_i{}^*$ in the second part means we only do regression to positive anchors.

3.2.5 Proposals

Given the image size is $M \times N = 800 * 600$, after the process we discuss above, we will get a matrix after classification with size $50 \times 38 \times 2k$ since it is binary classification, and another matrix after regression with size $50 \times 38 \times 4k$ since we need four transformation.

The proposal layer is used to filter bounding boxes. Some bounding box would cross boundary of the image, so we would clip to the image boundary. Besides, some bounding box highly overlaps with each other, we would adopt non-maximum suppression (NMS) on proposals, substantially reducing the number of proposals. After NMS, we use the top-N ranked proposal

regions for detection. So this is the end of RPN, and we complete the process of generating proposal regions.

3.3 RoI pooling

Since the sizes of proposal regions are different, but the size of input image of CNN is required to be the same, we use Region of Interest Pooling(RoI pooling) to fix this problem.

For each proposal region, we find its projection in the feature map we get after convolutional layer. If the size we want is 7×7, we will divide the projection area into 7×7, and do maxpooling as Figure 6 shows. In this way, we will get proposal regions with the same size.

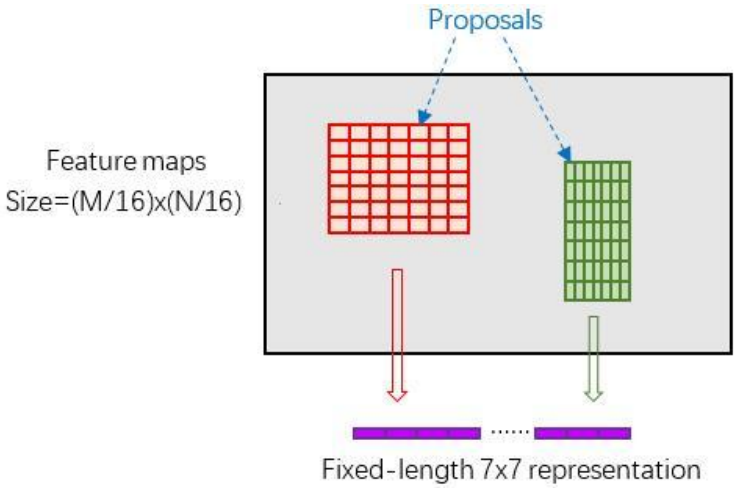


Figure 6. RoI Pooling

3.4 Classification & Regression

Before the final step of Faster R-CNN, we have used convolutional layer to abstract feature maps, RPN to generate proposal regions, and RoI pooling to resize the proposal regions to the same size. In this final stage, we receive the feature maps from RoI pooling, and then classify proposals into one of the classes, and adjust the bounding box by regression. At the end, we will get coordinates of bounding box and probabilities of a bounding box being in a class.

4. Experimental Setup

There are some implementations of Faster R-CNN online, and we choose one of them. To train the model, we need to first transfer our data into the required form of input. Then we fix the bugs during running the code.

4.1 Data Preprocessing

The required form of input is <filepath,x1,y1,x2,y2,class_name>, where filepath is the path of the image, x1 is the xmin coordinate for bounding box, y1 is the ymin coordinate for bounding box, x2 is the xmax coordinate for bounding box, y2 is the ymax coordinate for bounding box,

class_name is the name of the class in that bounding box. Figure 7 shows a piece of our input data.

```
/cells_1017.png,711,1105,844,1225,trophozoite
/cells_1017.png,761,972,891,1101,trophozoite
/cells_1017.png,907,53,1004,179,trophozoite
/cells_1017.png,195,176,315,292,trophozoite
/cells_1003.png,384,988,547,1153,difficult
/cells_218.png,463,621,603,750,trophozoite
/cells_218.png,320,1236,475,1374,trophozoite
/cells_230.png,172,941,311,1101,difficult
/cells_224.png,882,873,965,958,trophozoite
/cells_224.png,218,282,321,373,trophozoite
/cells_224.png,229,761,326,857,trophozoite
/cells_595.png,293,501,380,635,trophozoite
```

Figure 7. Input

4.2 Model training

4.2.1 Technical Challenge

After completing preprocessing blood cell image data, faster-rcnn model was applied to this dataset for accomplish object detection task. The reason why we choose faster-rcnn as our object detection is that it's predecessor r-cnn and fast-rcnn would take 50 times more time to train which each epoch of the training would take hours to finish. The time constraint of training other model can be avoid by using faster-rcnn. It originally took 3 hours to train each epoch. After some research, we discover the model was trained on CPU instead of utilizing GPU. We did experiment with different tensorflow version and Keras version in order to make use of the GPU computation power. Training time was drastically reduced to 1 hour per epoch. The time is relatively short compared with previous 3 hours per epoch but the whole training process was still not reasonable for our project due to limit time and computation power. We visualize the number of training samples by their categories(different stages malaria of red blood cell). We discover the data is extremely imbalanced. Since healthy red blood cell are very different from unhealthy one, we decided treat red blood cell as background in our object detection task which not only reduce numbers of input for training the network but also make the remaining training image data more balanced. Now each training epoch only took 15 minutes to finish which is more feasible than before. After finished solving these technical difficulties, we are ready to really train our model by adjusting hyper-parameters.

4.2.2 Hyperparameter tuning

4.2.2.1 RPN

Due to faster-rcnn containing two sub networks (pre-trained CNN and RPN), the variety of hyper-parameter increased a lot. We divided all hyper-parameters from faster-rcnn into two categories, one is common parameter and another is uncommon parameter. The common hyper-parameter we were referring to are the same parameters involved when a convolutional neural network is being trained. The uncommon hyper-parameters appear because of the existence of region proposal network. The uncommon hyper-parameter includes input image size, anchor size, anchor scale, and threshold value which determines if a proposal is positive.

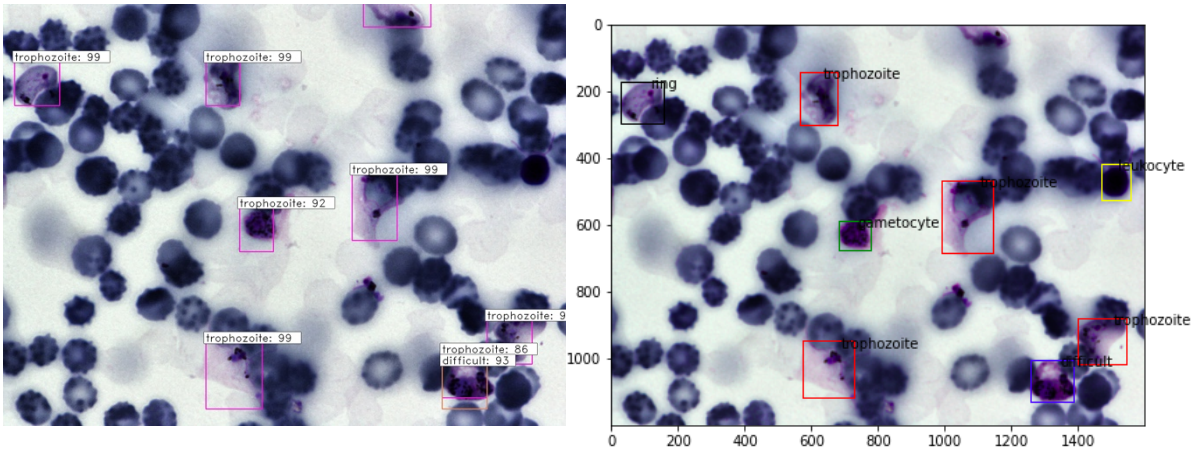
The first three uncommon parameters will be discussed together. As we know, generally object would have different sizes in a image, they even have different width and height ratio in order to produce accurate bounding boxes for variety sizes we need to choose the correct anchor size and aspect ratio for the specific task. In our case, since all the blood cell approximately have the same size and similar aspect ratio, we do not need the variety sizes of boxes therefore we do not need to worry about the anchor size and scale too much. As long as we choose the relatively right number for this task, the region proposal network will work well so the bounding box would be close to the object. We used the default parameters settings for producing anchor according to faster-rcnn paper. We find out the bounding box works well when using suggested number in the paper. We also keep our image size the same to theirs since anchor size is relative to image size. Since the limit of GWU VPN connection time, we can only train 10 to 20 epochs before automatically disconnection of VPN in order to access AWS, so overfitting is not yet encountered. We also tried image augmentation for our tasks, it does not work really well, we suspect that that's caused by not enough epoch trained on it.

4.2.2.2 CNN

As we discussed before, we used pre-trained convolutional neural network to accomplish the feature extraction before RPN. In our case, we used resnet50 as our option. We used the resnet50 as both feature extractor(freeze the weights and bias during training) and also fine-tuning the model using pre-trained weights and biases as initialization.

5. Results

Figure 8 shows the comparison of the sample prediction on test image(left) and original test image(right). It's exciting that although some false positive and false negative happen, most of cells are classified accurately with very high possibility shown on the top of the bounding boxes.



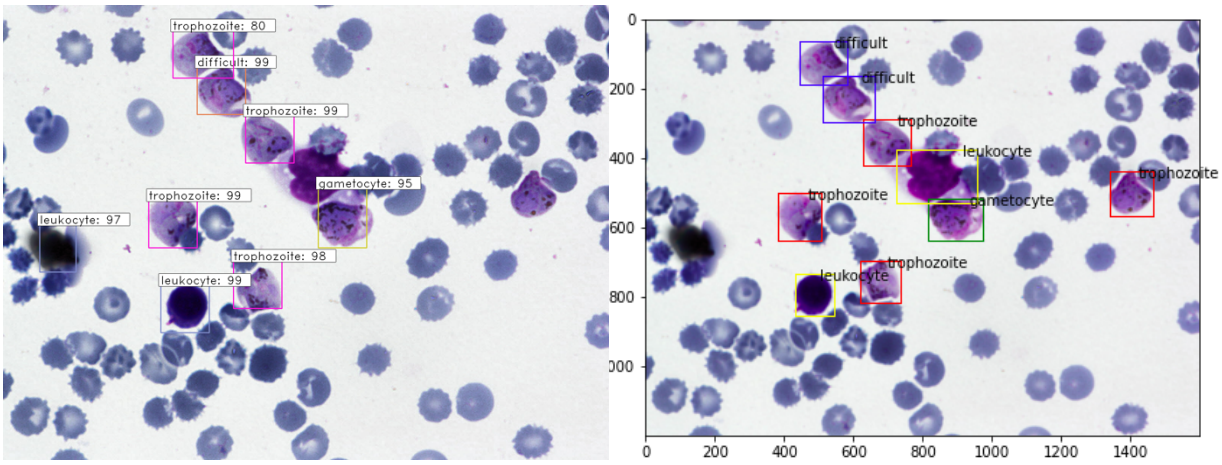


Figure 8. Comparisons of Prediction and Test Image

Table 1 shows the final Average Precision (AP) value of each categories and the mean Average Precision (mAP) values. As we can see, the AP values of 'trophozoite', 'difficult' and 'schizont' are very high, indicating that the features of these three categories are well captured by our model. In contrast, the detection ability on 'ring' is poor. A possible explanation for this may be the lack of adequate data since 'ring' cell only occupies 16.9% of the all the cells. Besides, the ring cell may have similar features with other cells, which may need additional techniques to be explored.

cells	AP
ring	0.4595
difficult	0.9888
trophozoite	0.9640
leukocyte	0.8202
schizont	0.9734
gametocyte	1
mAP = 0.8677	

Table 1. Average Precision

Besides, AP is calculated by the following formula, which is generally defined by the area under the precision-recall curve.

$$\mathrm{AP} = \int_0^1 p(r)dr$$

6. Summary and Future Work

The majority of our time contributes to understand the faster-rcnn and make the code work. In this process, we realized that to be able to fine tune a model, we really need to understand how it

works at first, then we can have more creative idea to deal with the problem. And we also learned that we can have creative way to make use of the existing model. Like in the faster-rcnn model, it is essential CNNs + Fully connected layer.Besides, we learned that transfer learning is very important in computer vision tasks, we can really build new work on previous people's work so we can solve variety of problems.

In the future, we know plotting loss function is very important for helping us to see how the training process goes. There are so many hyper-parameters in our model, after understanding the model we could pick to tune some important parameters first. And finally, the more data does not necessarily mean better model, but a better model definitely need more data!

Last but not least, just in the time when we are writing this report, we find out there are built in library in Pytorch to get the same faster-rcnn model in just one line of code. Next time, we definitely need to do some brain storming research before diving right into it. Although we also learned a ton from implement the model ourselves, we could do more one hyper-parameter tuning if we know there is already built in library support for this.

7. References

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in NIPS, 2015.
[2] R. Girshick, "Fast R-CNN," in IEEE International Conference on Computer Vision (ICCV), 2015.
[3] https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/
[4] https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52#f9ce