Assignment 3: Intelligent Analysis of Biomedical Images

Mahtab Bigverdi (96105604)

December 10, 2020

Goal

In this task we are trying to identify malaria infected cells in brightfield microscopy images of blood smears. There are two classes of uninfected cells (RBCs and leukocytes) and four classes of infected cells (gametocytes, rings, trophozoites, and schizonts).
Some details are discussed briefly.

- **Malaria** Malaria is an infectious disease caused by Plasmodium parasites, transmitted through mosquito bites. Symptoms include fever, headache, and vomiting, and in severe cases,seizures and coma. The World Health Organization reports that there were 228 million cases and 405,000 deaths in 2018, with Africa representing 93% of total cases and94% of total deaths. Rapid diagnosis and subsequent treatment are the most effective means to mitigate the progression into serious symptoms. However, many fatal cases have been attributed to poor access to healthcare resources for malaria screenings.
- For malaria as well as other microbial infections, manual inspection of thick and thin blood smears by trained microscopists remains the gold standard for parasite detection and stage determination because of its low reagent and instrument cost and high flexibility. Despite manual inspection being extremely low throughput and susceptible to human bias, automatic counting software remains largely unused because of the wide range of variations in brightfield microscopy images. However, a robust automatic counting and cell classification solution would provide enormous benefits due to faster and more accurate quantitative results without human variability; researchers and medical professionals could better characterize stage-specific drug targets and better quantify patient reactions to drugs.

Figure 1: Red Blood Cell (RBC)

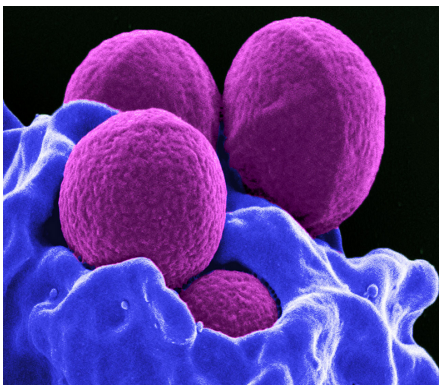


Figure 2: Leukocytes

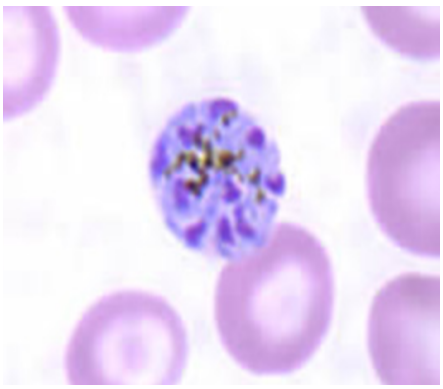


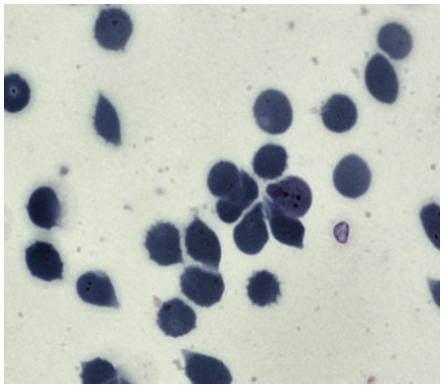
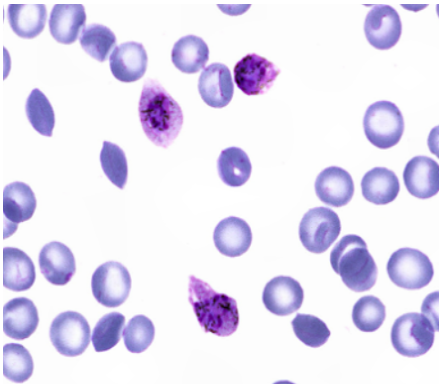
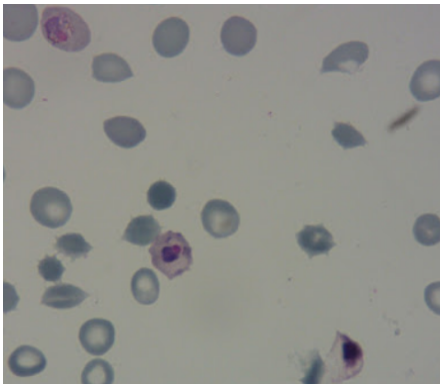
Figure 3: Schizonts

Dataset that is used in this task is from Broad Bioimage Benchmark Collection [1].

[1] https://bbbc.broadinstitute.org/BBBC041

1 Preprocess

I converted images to grayscale because the color of the images was different and then I used **unsharp masking** to enhance edges.
As we see below the colorof the images is not same.

1.1 Unsharp Masking

```
def unsharp (self, image):
        kernel = np.array([[-1,-1,-1],
                           [-1, 9,-1],
                           [-1,-1,-1]])
        sharpened = cv2.filter2D(image, -1, kernel)
        return sharpened
```

2 Dataset

To use implemented functions in Pytorch, the dataset should have a special pattern.

- image: a PIL Image of size (H, W)
- target: a dict containing the following fields
  - boxes (FloatTensor[N, 4]): the coordinates of the N bounding boxes in [x0, y0, x1, y1] format, ranging from 0 to W and 0 to H labels (Int64Tensor[N]): the label for each bounding box. 0 represents always the background class.
  - image_id (Int64Tensor[1]): an image identifier. It should be unique between all the images in the dataset, and is used during evaluation
  - area (Tensor[N]): The area of the bounding box. This is used during evaluation with the COCO metric, to separate the metric scores between small, medium and large boxes.
  - iscrowd (UInt8Tensor[N]): instances with iscrowd=True will be ignored during evaluation.

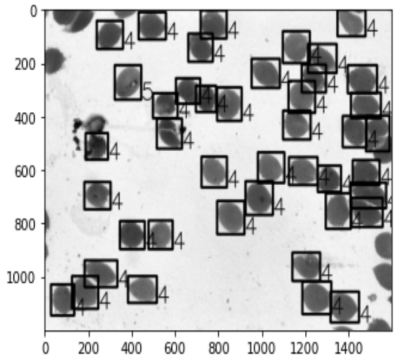
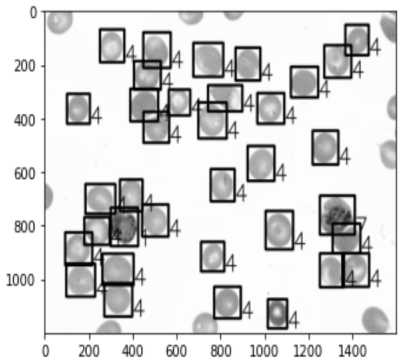
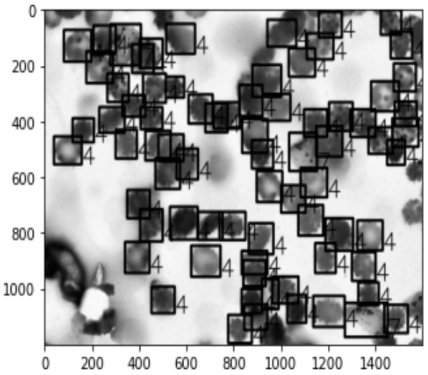
We have 8 classes for objects. 0 is for background.

```
category = {    1: 'difficult',
                2 : 'gametocyte',
                3: 'leukocyte' ,
                4: 'red blood cell',
                5: 'ring',
                6: 'schizont',
                7: 'trophozoite' }
```

In the dataset we have two json files. We construct our custom dataset using these files.

```
        for j in range(len(labeldict[i]['objects'])):
            x0 = labeldict[i]['objects'][j]['bounding_box']['minimum']['c']
            y0 = labeldict[i]['objects'][j]['bounding_box']['minimum']['r']
            x1 = labeldict[i]['objects'][j]['bounding_box']['maximum']['c']
            y1 = labeldict[i]['objects'][j]['bounding_box']['maximum']['r']
            boxes.append([int(x0), int(y0), int(x1), int(y1)])
            label = labeldict[i]['objects'][j]['category']
            labels.append(inverse_cat[label])

        boxes = torch.as_tensor(boxes, dtype=torch.float32)

        labels = torch.as_tensor(labels, dtype=torch.int64)

        image_id = torch.tensor([i])

        area =  (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])

        iscrowd = torch.zeros((len(boxes),), dtype=torch.int64)

        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd
```

Three images from train dataset and their bounding boxes.

3 Model

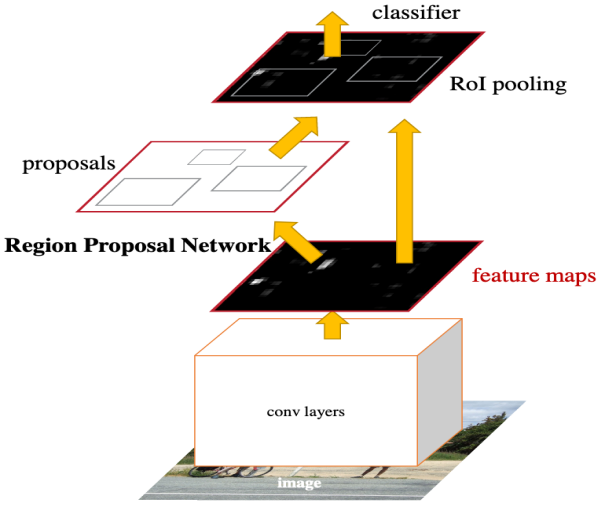
I used Faster R-CNN from torchvision models with **Resnet-50-FPN** backbone.[2].
I modified the model a bit.

```
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.rpn import AnchorGenerator


model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained= False)

num_classes = 8  # 7 class  + background
# get number of input features for the classifier
in_features = model.roi_heads.box_predictor.cls_score.in_features
# replace the pre-trained head with a new one
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

### input is grayscale with one channel
model.backbone.body.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3

anchor_generator = AnchorGenerator(sizes=((8, 16, 32),),
                                   aspect_ratios=((0.5, 1.0, 2.0),))

# model.rpn.anchor_generator = anchor_generator

model.transform = torchvision.models.detection.transform.GeneralizedRCNNTransform
(min_size=(800,), max_size=1333, image_mean=[0.485], image_std= [0.229])
```

- Images are in range 0 and 1.
- As we see above, I changed num_classes to 8 (7 cells and background).
- I changed input channels of first convolutional layer (grayscale images)
- I changed number of channels in transformations.

Some Terms are discussed below.

3.1 Faster R-CNN



[2] https://pytorch.org/docs/stable/torchvision/models.html

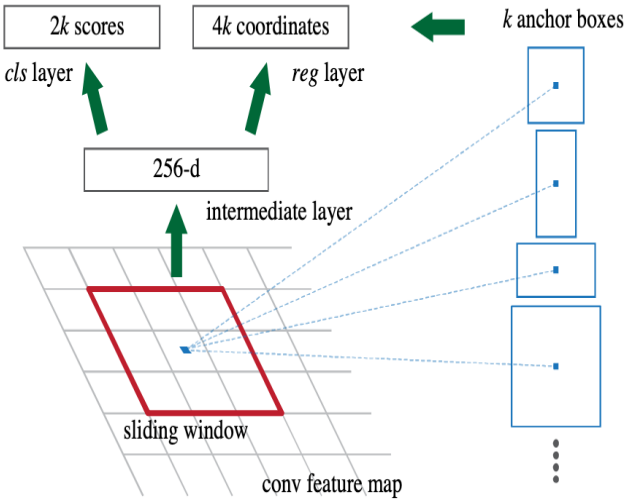
Faster R-CNN is composed of two modules. The first module is a deep fully convolutional network that proposes regions,and the second module is the Fast R-CNN detector that uses the proposed regions. The RPN module tells the Fast R-CNN module where to look.

3.1.1 Region Proposal Network (RPN)

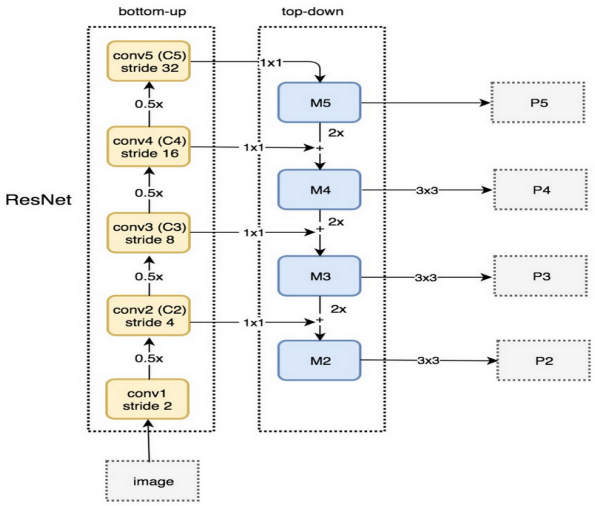
Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an **objectness** score. To generate region proposals, we slide a small network over the convolutional feature map output by the last shared convolutional layer. This small network takes as input an n×n spatial window of the input convolutional feature map.
At each sliding-window location, we simultaneously predict multiple region proposals, where the number of maximum possible proposals for each location is denoted ask. So the reg layer has 4k outputs encoding the coordinates of k boxes, and the cls layer outputs 2k scores that estimate probability of object or not object for each proposal.
The k proposals are parameterized relative to k reference boxes, which we call **anchors**.

3.1.2 Trainning RPN

The RPN can be trained end-to-end by back-propagation and stochastic gradient descent (SGD). Each mini-batch arises from a single image that contains many positive and negative example anchors.



3.2 Feature Map Network (FPN)



The bottom-up pathway uses ResNet to construct the bottom-up pathway. It composes of many convolution modules (convi for i equals 1 to 5) each has many convolution layers. As we move up, the spatial dimension is reduced

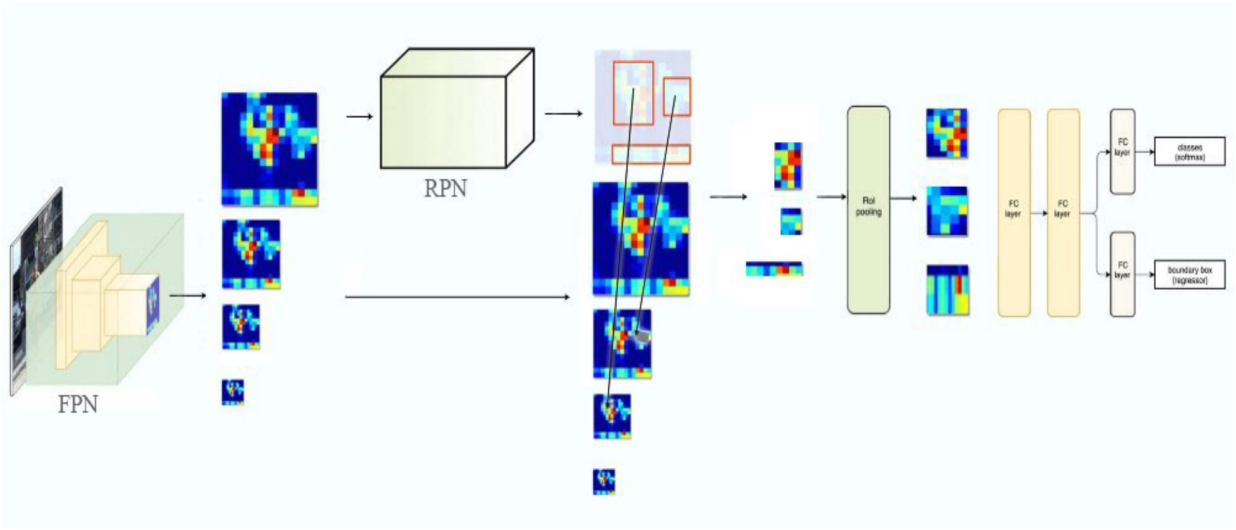
by 1/2 . We apply a 1 × 1 convolution filter to reduce C5 channel depth to 256-d to create M5. This becomes the first feature map layer used for object prediction.

As we go down the top-down path, we upsample the previous layer by 2 using nearest neighbors upsampling. We again apply a 1 × 1 convolution to the corresponding feature maps in the bottom-up pathway. Then we add them element-wise. We apply a 3 × 3 convolution to all merged layers. This filter reduces the aliasing effect when merged with the upsampled layer.

3.2.1 FPN with Faster R-CNN

In FPN, we generate a pyramid of feature maps. We apply the RPN to generate ROIs. Based on the size of the ROI, we select the feature map layer in the most proper scale to extract the feature patches.
The formula to pick the feature maps is based on the width w and height h of the ROI.

$$k = k_0 + log_2(\sqrt{wh}/224)$$



4 Training

Pytorch has implemented several functions to help developers with object detection. I used these:

```
!pip install cython
!pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
```

```
from engine import train_one_epoch, evaluate
import utils
```

Details of training:

- number of train datapoints = 300 (Due to lack of RAM and Storage)
- number of test datapoints = 120
- train data loader : batch_size = 2
- test data loader : batch_size = 1
- optimizer:

```
    optimizer = torch.optim.SGD(params, lr=0.005,
                                momentum=0.9, weight_decay=0.0005)
```

- scheduler:

```
    lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                                   step_size=3,
                                                   gamma=0.1)
```

- num of epoches = 10

```
def train():

    # construct an optimizer
    params = [p for p in model.parameters() if p.requires_grad]
    optimizer = torch.optim.SGD(params, lr=0.005,
                                momentum=0.9, weight_decay=0.0005)
    # and a learning rate scheduler
    lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                                   step_size=3,
                                                   gamma=0.1)


    num_epochs = 10

    for epoch in range(num_epochs):
        # train for one epoch, printing every 10 iterations
        train_one_epoch(model, optimizer, data_loader_train, device, epoch, print_freq=10)
        # update the learning rate
        lr_scheduler.step()
        # evaluate on the test dataset
        evaluate(model, data_loader_test, device=device)
```

5 Metrics

Object detection metrics serve as a measure to assess how well the model performs on an object detection task. It also enables us to compare multiple detection systems objectively or compare them to a benchmark. In most competitions, the average precision (AP) and its derivations are the metrics adopted to assess the detections and thus rank the teams.

The general definition for the Average Precision(AP) is finding the area under the precision-recall curve.
The mAP for object detection is the average of the AP calculated for all the classes. mAP@0.5 means that it is the mAP calculated at IOU threshold 0.5.
The mAP is a good measure of the sensitivity of the neural network. So good mAP indicates a model that's stable and consistent across different confidence thresholds. Precision, Recall and F1 score are computed for given confidence threshold.

After one epoch of training:

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.058
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.134
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.030
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.026
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.059
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.002
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.018
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.078
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.096
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.079
```

After ten epoch of training:

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.158
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.220
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.188
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.161
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.162
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.085
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.141
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.230
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.170
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.242
```

More epoches causes overfitting.

6 Results

Some Images from test set:

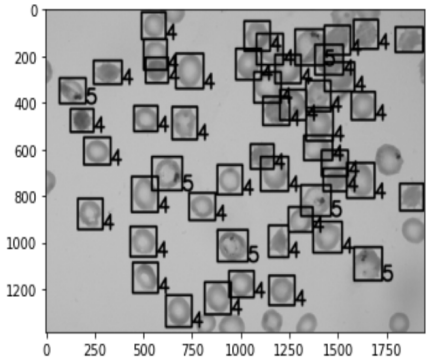


Figure 4: True boxes

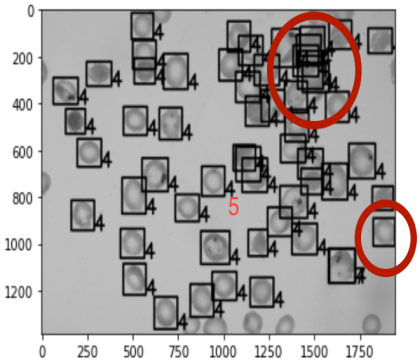


Figure 5: Prediction boxes

As we see label 5 is predicted 4 and in crowded parts more objects are detected.

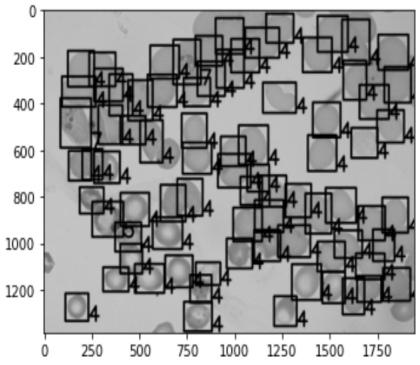


Figure 6: True boxes

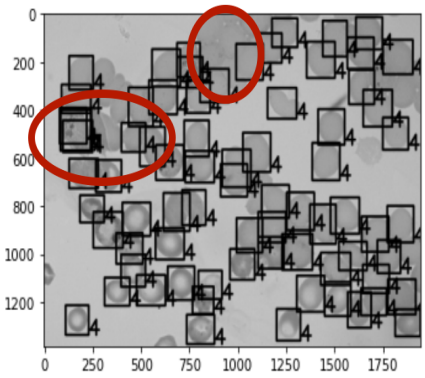


Figure 7: Prediction boxes

As we see in fade parts less objects are detected.

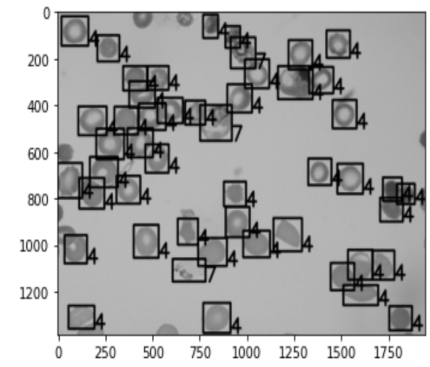


Figure 8: True boxes

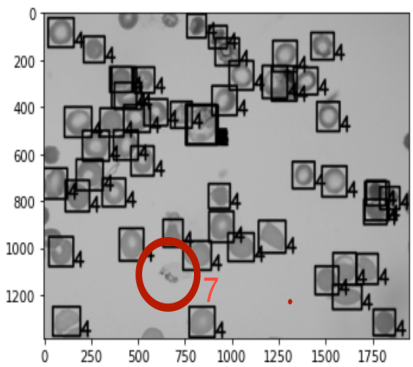


Figure 9: Prediction boxes

Object with class of 7 is not detected.

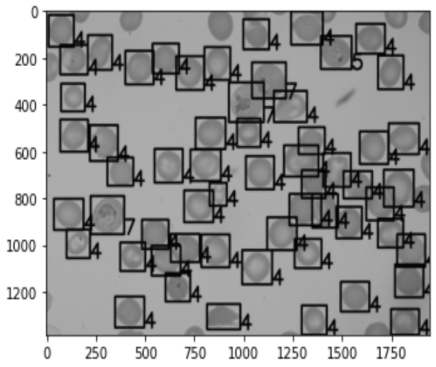


Figure 10: True boxes

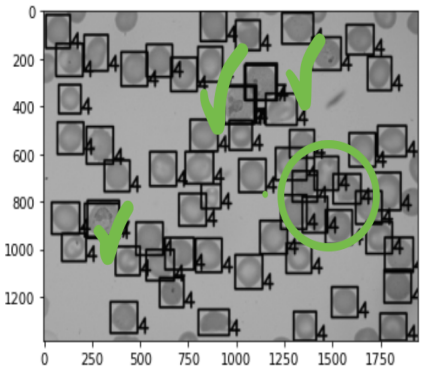


Figure 11: Prediction boxes

Objects with class 7 are truely detected and crowded parts are OK too.

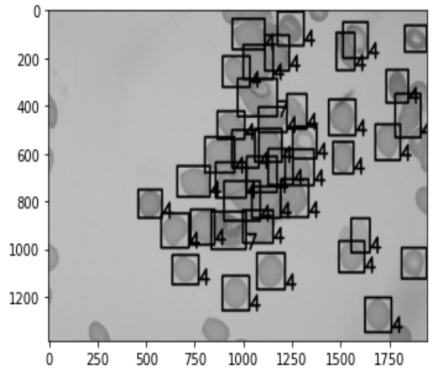


Figure 12: True boxes

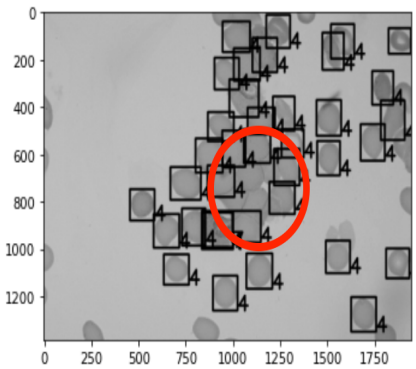


Figure 13: Prediction boxes

Again problem with crowded parts.

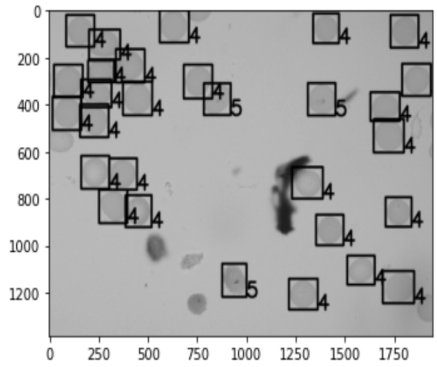


Figure 14: True boxes

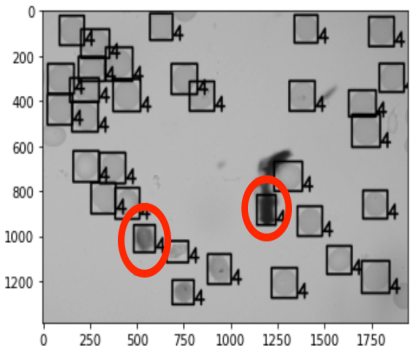


Figure 15: Prediction boxes

Some stains are not objects but they are detected by the model.

7 Reference

1. https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html