

VRDL-Homework2

Name: Kuok-Tong Ng

Student ID: 309652030

Project repository: [K-T-Ng/VRDL-Homework2 \(github.com\)](https://github.com/K-T-Ng/VRDL-Homework2)

1. Introduction

This homework is about **object detection**. We are dealing with **SVHN dataset** which contains 33402 training images and 13068 testing image. The images in SVHN dataset contains digits (see the figure below). Our goal is to train a **fast** and **accurate** digit detector.



Figure 1: SVHN dataset

In this homework, we use YOLOv3 architecture to train a digit detector. I use the code from [1], which implements YOLOv3, but with slightly different from the original paper [3]. I will explain the details in the next section.

2. Methodology

2.1 Model architecture

2.1.1 *Backbone* (Darknet-53)

The goal of backbone network is to extract **features**. The backbone in YOLO keep improving from V1 to V3, so I want to describe the evolution of the backbone here.

The reason that the author use Darknet-53 as backbone rather than Darknet-19 or ResNet-101 because Darknet-53 is more powerful than Darknet-19. Moreover, with comparable accuracy, Darknet-53 is more efficient than ResNet-101.

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	171
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	77.6	93.8	29.4	1090	37
Darknet-53	77.2	93.8	18.7	1457	78

Table 2. Comparison of backbones. Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

Figure 5 Comparison between Darknet and ResNet, from YOLOV3 paper [4]

There are one more thing that V1 and V2 doesn't have. YOLOV3 predicts boxes at 3 different scales (similar concept of FPN, which is in the next section), so there will be 3 outputs with different scale that is shown in Figure 4.

2.1.2 **Prediction** (Feature Pyramid Networks + YOLO head)

There is a lots of changes from YOLOV1 to YOLOV3.

In YOLOV1, the image is divided into an $S \times S$ grid. Each grid cell predicts

B bounding boxes

B confidence scores for these boxes $\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$

C conditional class probabilities $\Pr(\text{Class}_i | \text{Object})$.

As shown in the figure below, bounding boxes are described by

(x, y): centered of the bounding box (offset of a particular grid so that they are bounded between 0 and 1)

(w, h): width, height of the bounding box (divided by the image width and image height so that they are bounded between 0 and 1)

conf.: the confidence score of that box

Limitation: Since the model learns to predict bounding boxes coordinates from data, it struggles to generalize to objects in new aspect ratios.

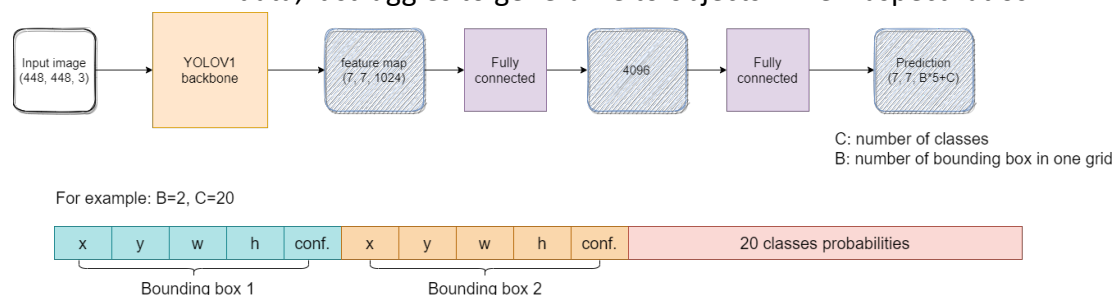


Figure 6 YOLOV1 prediction

In YOLOV2, author adopted the concept in Faster-RCNN. They use anchor boxes

to predict the bounding boxes. The network predicts the offsets for anchor boxes instead of the coordinates of the bounding boxes.

Different from Faster-RCNN, the anchor boxes in YOLOV2 is chosen by k-means (Faster-RCNN's anchor boxes are chosen by hand) on the training dataset with the metric $d(A, B) = 1 - \text{IOU}(A, B)$. The author do this because if we pick a better prior, it is easier for the network to learn. In YOLOV2, the author use 5 anchor boxes.

Although using anchor boxes will slightly decrease mAP, but there is a significant improvement of Recall (result from YOLOV2 paper). Which means the false positive are decreased and there has more space to improve.

Figure7 shows the procedure of YOLOV2 predicting bounding boxes. For each bounding boxes, there are 5+C elements. (t_x, t_y, t_w, t_h) are the offset of bounding box and $\sigma(t_o)$ is the confidence score.

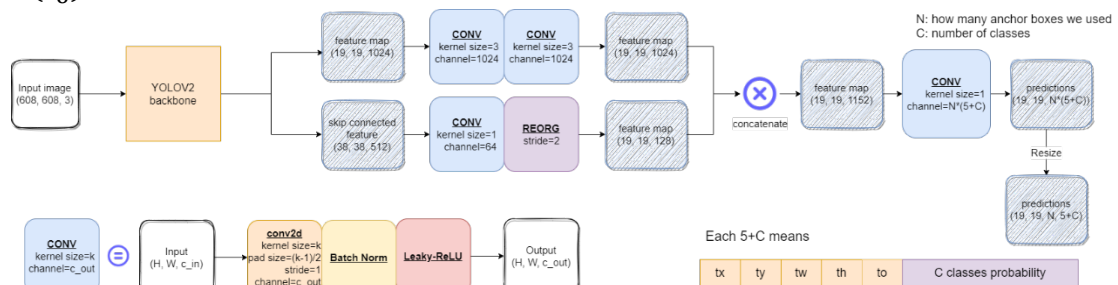


Figure 7 YOLOV2 prediction

Figure8 shows how to recover the true position for the bounding box. For example, Figure8 shows the (1, 1)th grid, so $(c_x, c_y)=(1, 1)$. By adding the sigmoid of offset, then multiply by 32 (since YOLOV2 network down-sample by a factor of 32), then we get the exact coordinate of the bounding box. On the other hand, suppose the anchor box with $(\text{width}, \text{height})=(p_w, p_h)$, then the exact width and height will be multiply by exponential of (t_w, t_h) (since there is no negative width and height).

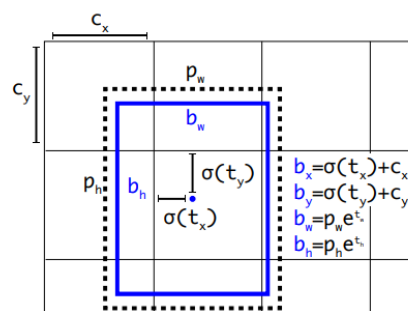


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

Figure 8 Bounding box recovery (from [3])

In YOLOV3, the author adopted the concept of Feature Pyramid Networks, which is a method that detecting objects at different scales. Which can be done by using shortcut tricks as the following figure. The smaller feature map has larger

receptive field, which can be detect the larger object. The bigger feature map is merged with upsampled features using concatenation, which can get more meaningful semantic information and finer-grained information, which is suitable for detecting small object.

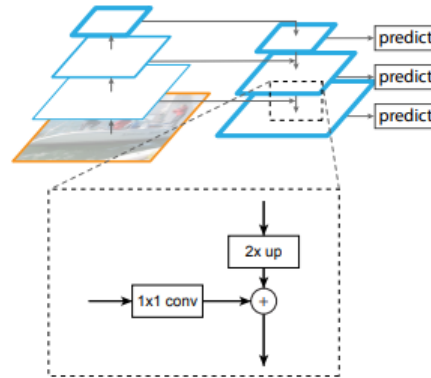


Figure 3. A building block illustrating the lateral connection and the top-down pathway, merged by addition.

Figure 9 FPN (from [5])

Figure10 shows how to predict bounding boxes. It use the concept of FPN (the feature extraction process does not shown in this figure but in Figure4). YOLOV3 also use anchor boxes chosen by k-means. YOLOV3 uses 9 anchors boxes and divide equally into 3 different scales prediction. Hence, each different scales has 3 bounding boxes ($N=3$ in Figure10). The recovery of the bounding boxes is the same as YOLOV2, which is shown in Figure8.

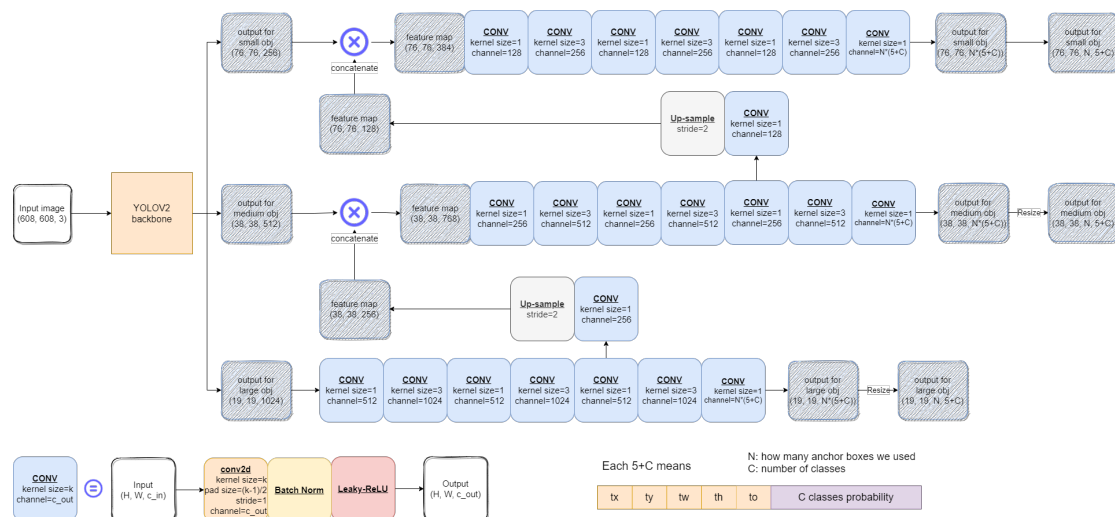


Figure 10 YOLOV3 prediction

2.2 Data preprocess

Before feeding the images to the model, we first resize (keeping the axis ratio and padding with gray value) the image to $(3, N, N)$, where N will be determined

later.

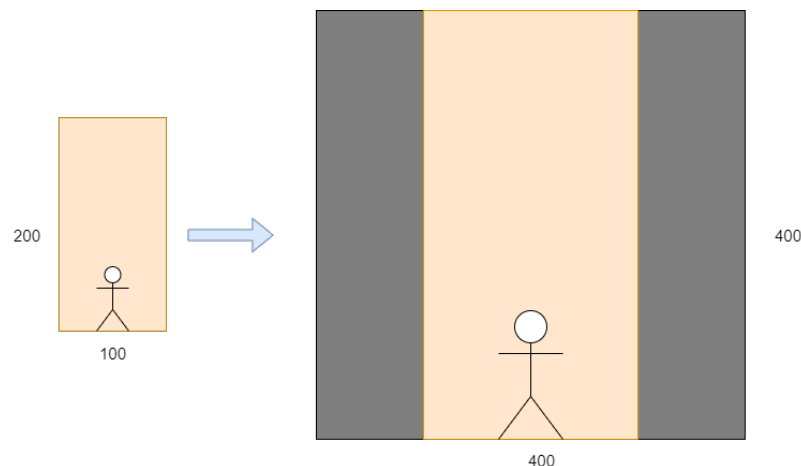


Figure 11 Resize image

For data augmentation, we apply random horizontal flip with probability 0.5.

Note that the model architecture of YOLOV3 is down sampled of a factor 32 and it only uses the layers that can adopt various input size. Therefore, for the image with both width and height divided by 32, it can be the input of YOLOV3. Hence, multi-scale training will adopt in this homework. For every 10 iterations, we random the value of N from {320, 352, 384, 416, 448, 480, 512, 544, 576, 608}.

2.3 Anchor K-means

As we mention in 2.1, YOLOV3 use anchor boxes that obtained from K-means on the training data. In this homework, we split the training dataset into (30000 for training, 3402 for validation), we perform K-means on 30000 images after resize the image to 416 by 416 (of course, bounding boxes coordinates also). We obtain the following result

(22.61, 49.62), (35.47, 71.36), (33.95, 115.04) for small object
 (48.97, 91.93), (48.52, 134.27), (64.33, 111.38) for medium object
 (62.71, 150.82), (80.02, 140.86), (102.62, 166.35) for large object.

2.4 Loss function

According to the YOLOV3 paper, for each scale prediction, its loss composed by 3 elements and it seems like compute as follows. Consider a scenario in Figure10

- We obtained $76*76*3+38*38*3+19*19*3=22743$ from the model
- For each ground truth bounding box, find a predict bounding box from the remaining bounding box, compute
 1. **coordinate loss by MSE** $\sum_{* \in \{x,y,w,h\}} (t_* - \hat{t}_*)^2$, where t_* is the output of the model, \hat{t}_* comes from the ground truth bounding box (need to computed by inverting the equation in Figure8)
 2. **objectness score loss** $-\log \sigma(t_o)$

3. **class prediction loss** $\sum_{i=1}^C \hat{c}_i \log(\sigma(c_i)) + (1 - \hat{c}_i) \log(1 - \sigma(c_i))$,
 where c_i are the outputs of the model (C classes probability in Figure8)
 and \hat{c}_i are the ground truth class label.

- Cancel those predicted bounding boxes that has high IOU (>0.5) with this ground truth bounding box.
- When all ground truth bounding boxes are done, the remaining predicted boxes should not exist. Therefore, calculate objectness score loss for those remaining bounding box $-\log(1 - \sigma(t_o))$.
- Sum them up.

The loss function that implemented in [1] has slightly difference with the description above.

- For the coordinate loss, [1] use GIOU loss.

In order to improve the accuracy in mAP@0.5:0.95 (contains a lot's of IOU calculation), it is reasonable to use a loss function that contains IOU.

$\mathcal{L}_{iou} = 1 - IOU$ is a metric and it is scale invariant. However it has a major weakness, when PD and GT bounding box does not intersect, $\mathcal{L}_{iou} = 1$ by definition. It does not reflect how far between PD and GT.

GIOU loss can mitigate this problem. For 2 convex sets $A, B \subseteq \mathbb{R}^n$, find the smallest convex set $C \subseteq \mathbb{R}^n$ that enclosing A, B , then

$$GIOU(A, B) = IOU(A, B) - \frac{|C - (A \cup B)|}{|C|}$$

Clearly, GIOU attain maxima when A and B are coincide. If $A \cap B = \emptyset$ and A and B are far away from each other, $GIOU(A, B) \approx -1$ (See Figure12). GIOU loss is defined by $\mathcal{L}_{GIOU} = 1 - GIOU$.

Since GIOU loss is also scale invariant, we may convert the predicted bounding box into the original scale and calculate with the ground truth bounding box. I.e. Given (t_x, t_y, t_w, t_h) and $(\hat{x}, \hat{y}, \hat{w}, \hat{h})$, calculate

- $x = (\sigma(t_x) + c_x) * \text{stride}, y = (\sigma(t_y) + c_y) * \text{stride}$ (There are 3 different scales in YOLOV3, stride can be 8, 16 or 32)
- $w = p_w e^{t_w}, h = p_h e^{t_h}$
- $\mathcal{L}_{GIOU} = 1 - GIOU((x, y, w, h), (\hat{x}, \hat{y}, \hat{w}, \hat{h}))$

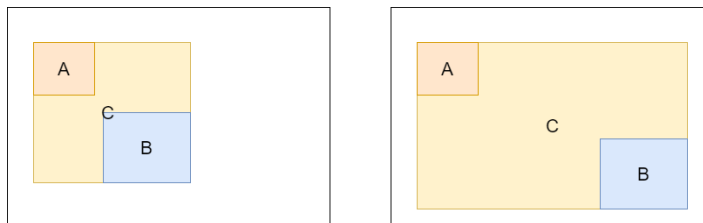


Figure 12 GIOU loss

- For the objectness score loss, [1] use focal loss with $\gamma = 2, \alpha = 1$

2.5 Mish activation

Mish activation is defined by $\text{Mish}(x) = x * \tanh(\ln(1 + e^x))$. Comparing with Leaky ReLU

Advantages: More mathematical background (Figure13) than Leaky ReLU, and performing well than Leaky ReLU (Figure14, which is from YOLOV4 paper, comparing the backbone network accuracy as a classifier).

Table 1 provides a detailed summary of the properties of Mish Activation Function.

Table 1. Properties Summary of Mish

Order of Continuity	C^∞
Monotonic	No
Monotonic Derivative	No
Saturated	No
Approximates Identity Near Origin	Yes

Figure 13 (from [7])

Table 3: Influence of BoF and Mish on the CSPDarknet-53 classifier accuracy.

MixUp	CutMix	Mosaic	Blurring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.2%	93.6%
	✓	✓		✓			77.8%	94.4%
	✓	✓		✓		✓	78.7%	94.8%

Figure 14 Comparison between Leaky ReLU and Mish(from [8])

Disadvantages: Higher computational cost than Leaky ReLU.

I found that the author in [1] has implemented Mish activation function but didn't try to plug in the model, so I have tried to put Mish activation into the layers that is not in the backbone network (since we use the pretrained weights for the backbone network). The result will be state in Summary section.

2.6 Hyper parameter

- Mini batch size = 6 (The max size that can avoid OOM)
- Epochs = 30
- Optimizer: SGD with momentum 0.9, weight decay 0.0005
- LR scheduler: Cosine decay LR with warm up.
2 epoch for warm up, linear increase from 0 to $1e-4$
Then the others epochs cosine decay to $1e-6$.

2.7 Inference procedure

For the inference procedure, we predict bounding boxes for an images by

performing the following steps:

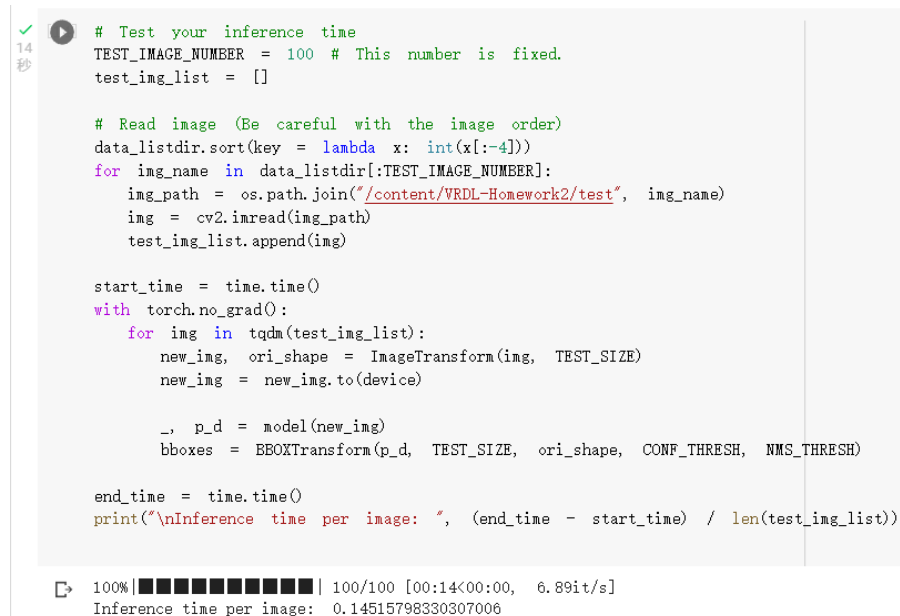
1. Resize the image to 608 by 608.
2. Feed the image to the network and get 22743 bounding boxes
3. Scale the coordinate to the original image size (Inverse Resize)
4. Filter out those invalid boxes and confidences scores less than 0.05
5. Reduce the number of bounding boxes by NMS using 0.5 as threshold.

The code will be pasted in Summary section.

3. Summary

In this homework, I have only trained 2 models for 30 epochs (around 10 hours). The only difference of this two models are whether using Mish activation on prediction layers. I have got 0.388359 and 0.395243 mAP@0.5:0.95 on the testing data by using the same setting in 2.7, respectively. By tuning confidence score threshold and NMS threshold, I have received 0.395872 mAP@0.5:0.95 finally.

For the speed-test on colab. The inference time result is as follows



```
# Test your inference time
TEST_IMAGE_NUMBER = 100 # This number is fixed.
test_img_list = []

# Read image (Be careful with the image order)
data_listdir.sort(key = lambda x: int(x[:-4]))
for img_name in data_listdir[:TEST_IMAGE_NUMBER]:
    img_path = os.path.join("/content/VRDL-Homework2/test", img_name)
    img = cv2.imread(img_path)
    test_img_list.append(img)

start_time = time.time()
with torch.no_grad():
    for img in tqdm(test_img_list):
        new_img, ori_shape = ImageTransform(img, TEST_SIZE)
        new_img = new_img.to(device)

        _, p_d = model(new_img)
        bboxes = BBOXTransform(p_d, TEST_SIZE, ori_shape, CONF_THRESH, NMS_THRESH)

end_time = time.time()
print("\nInference time per image: ", (end_time - start_time) / len(test_img_list))
```

100%|████████████████████| 100/100 [00:14<00:00, 6.89it/s]
Inference time per image: 0.14515798330307006

Figure 15 Speed test on colab

The functions **ImageTransform** and **BBOXTransform** in this screenshot plays a role in **preprocessing on image** and **post processing on bounding box**. The code of these two functions are also in the inference.ipynb.

Combining with the paper and the code, I have a better understanding how YOLO works.

Reference:

- [1] [Peterisfar/YOLOV3: yolov3 by pytorch \(github.com\)](#)
- [2] YOLOV1 paper: [\[1506.02640\] You Only Look Once: Unified, Real-Time Object Detection \(arxiv.org\)](#)
- [3] YOLOV2 paper: [\[1612.08242\] YOLO9000: Better, Faster, Stronger \(arxiv.org\)](#)
- [4] YOLOV3 paper: [\[1804.02767\] YOLOv3: An Incremental Improvement \(arxiv.org\)](#)
- [5] FPN paper: [\[1612.03144\] Feature Pyramid Networks for Object Detection \(arxiv.org\)](#)
- [6] GIoU loss paper: [\[1902.09630\] Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression \(arxiv.org\)](#)
- [7] Mish activation: [\[1908.08681\] Mish: A Self Regularized Non-Monotonic Activation Function \(arxiv.org\)](#)
- [8] YOLOV4 paper: [\[2004.10934\] YOLOv4: Optimal Speed and Accuracy of Object Detection \(arxiv.org\)](#)