Özgün Yargı

House Number Detection

Abstract

In this exercise, large version of YOLOv5 was the best model with mean average precision %83.3 (at 91th epoch). Colab was used in this exercise. For the best performance, I applied 100 epochs to 320x320 image size. The training process was very fast. It was completed in 1 hour 51 minutes with. It was seen that changing the image size has an impact on accuracy of the model.

1. Introduction

Recognizing digits inside of an image can be a challenging task that can be crucial for many different industries. One implementation could be recognizing house numbers inside of images. This is a challenging task, since real world taken images contains distortions and corruptions that are difficult to handle for a system based on hand-engineered representation (Yuval Netzer, et al. 2011). In this exercise, we try to detect the numbers inside on of an image.

In this task, SVHN dataset was used. This dataset includes 33,496 images which may contain more than one digit. Each digit covered by a bounding box which is defined by giving the height-width of the bounding box as well as the location of upper left corner.

2. Methods

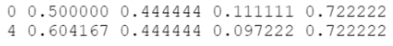
2.1 Preprocessing

The dataset used for this training contains 33,404 images with different widths and heights. Yolov5 requires a strict format for training. For instance, it expects bounding boxes to be normalized and expects same width and height for all images. Because of this, I resized all the images in two formats to check whether the image size influences detection precision or not. Because of this, I created two different datasets. Ones with 64x64 sizes and the other with 128x128.

Other task of preprocessing was normalizing the bounding boxes. To do this, each bounding box was divided to their image width and height. After bounding boxes are arranged, dataset was split as train and validation. However, since we have lots of images (33,404), due to hardware limitations, I decided to use only the 4000 images for training and 1000 images for validation.

As it was mentioned earlier, Yolov5 requires specific arrangement on data itself as well as their locations. It expects you to open a folder that contains two folders inside. One for images, and the other for labels. Each of them must contain two other folders for training and validation. The image names and corresponding label document's name must be same. Also, the label text must be in the following order:

- Each line => Class Label, x center, y center, width, height.



```
2 0.354167 0.444444 0.180556 0.722222
0 0.500000 0.444444 0.111111 0.722222
4 0.604167 0.444444 0.097222 0.722222
```

Figure 1: Inside of a text file

Figure 1. shows the inside of a text file. As it can be seen, all bounding box information is normalized and at the beginning of each line, we described related class value.

2.2 Network Architecture

For this exercise, I used two different Yolov5: Large model, small model. There is no big difference between small and large but only some additional layers and higher/lower number of parameters. When it comes to difference, small model is much faster than large model however, in return, large model is more accurate. It is separated into three as backbone, neck, and head.

For the backbone, CSP-Cross Partial Networks, are used to extract features from input image. Main property of this backbone is that it reduces the computation %20 without reducing the accuracy of the model which makes it a fast backbone (Chien-Yao Wang, et al. 2019).

For the neck, PANet is used to generate feature pyramids which helps models to perform well on unseen data.

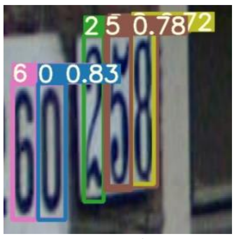
For the head, yolo is used. Leaky RELU and Sigmoid activation functions are used with Stochastic Gradient Descent (learning rate = 0.01, momentum = 0.937).

3. Results

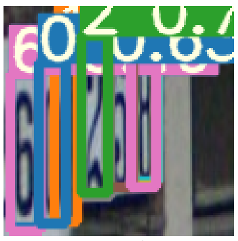
Some of the details regarding trained models can be seen from Table 2.

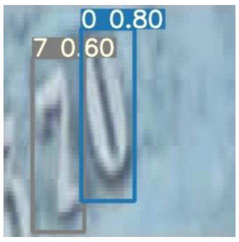
Models	Epochs	Image Size	Precision	Recall	**mAP@.5**
Yolov5 (small)	100	64	0.82	0.61	70.1%
Yolov5 (small)	1500	128	0.781	0.702	75.03%
Yolov5 (large)	100	320	0.843	0.779	83.01%

Table 1: Some Specifications regarding trained models and results

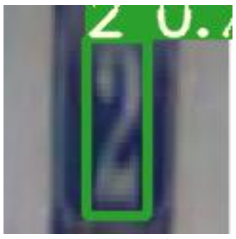


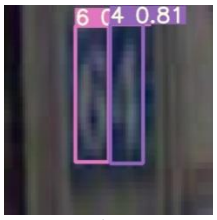
(a): Prediction from large model


(b): Prediction from small model

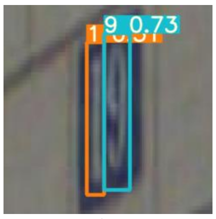


(c): Prediction from large model


(d): Prediction from small model



(e): Prediction from large model



(f): Prediction from large model

4. Discussion

As it can be seen from the predictions, both models do the detection well. One noticeable thing is that the training duration. For 1500 epochs, training with 4000 (128x128) images took 1 hour and 51 minutes. For the large model, it took about 1 hour (for 100 epochs) which is pretty fast in my opinion. One other thing that can be interesting to mention is that the mAP@.5 plot of the small model.

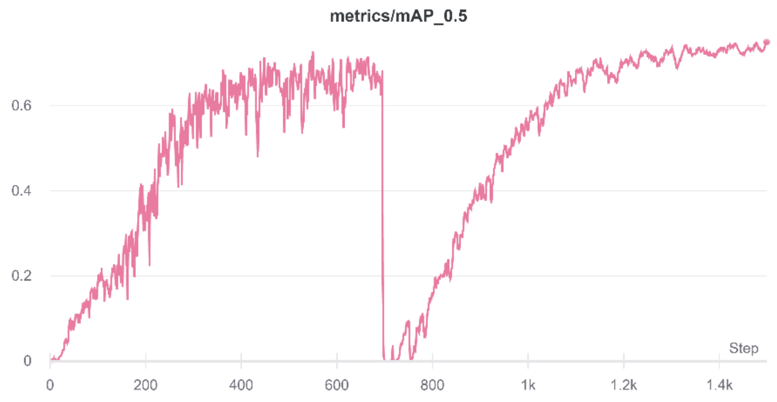


Figure 2: mAP@.5 accuracy versus epoch plot

As it can be seen from the figure 2, there is a high drop in the 700th epoch which makes not sense. Nothing had changed during the process. Because of this, I couldn't find any reason why there was a drop.

When both models' accuracies are compared, we can see the effect of the image size. As image size increases, it becomes easier for a model to detect objects inside of an image. This can also be seen from the Table 1. as well. The model that was trained with bigger image size has better accuracy than the other.

5. Appendix

- YOLOv5_ForImageSize_300.ipynb
- YOLOv5_ForImageSize_120.ipynb
- Yolov5_ForImageSize_64.ipynb
- Preprocessing_1.ipynb
- Preprocessing_2.ipynb
- Yolov5_Predict.ipynb

6. References

- Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, Jiaya Jia, 2018, *Path Aggregation Network for Instance Segmentation*
- Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, 2019, *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*