Technische Universität München | Fakultät für Informatik
Boltzmannstr. 3 | 85748 Garching

Masterpraktikum 2019

Vision-Based Navigation and Learning for Autonomous Vehicles

Task 2: Automatic Navigation System Based on Deep Learning Method.

Final Report

Name:	**Wu, Kuan-Hsun**
Praktikum Partner:	
Praktikum Tutor:	**Feng, Mingyue**
Praktikum Date:	

I. Introduction

Since the object detection technique has become mature and precise by convolutional neural network (CNN) recently, the autonomous driving task applies it as the complemental system of navigation. The system recognizes the traffic signs via dashboard cameras when the GPS signal is not available. There are many CNN object detection architectures suitable for this task. However, the accuracy and the time latency are always the tricky trade-offs in this topic.

Álvaro Arcos García in his paper (Arcos-Garcia, Alvarez-Garcia, & Soria-Morillo, 2018) analyses the state-of-the-art of several object-detection systems (Faster R-CNN, R-FCN, SSD, and YOLO V2) in 2018. The evaluation and comparison of these models include key metrics, such as the mean average precision (mAP), memory allocation, running time, number of floating-point operations, number of parameters of the model, and the effect of traffic sign image sizes. In praktikum task 1, I extended his paper comparing the latest YOLO V3 and real-time response YOLO tiny V3. After analyzing the YOLO tiny V3 architecture, I proposed a slimmer YOLO tiny V3 called YOLO tiny tiny which performs better than most of the models and responds in real-time under the CPU environment.

II. Materials

For the training data preparation, I followed the Yolo V3 (Redmon & Farhadi, 2018) training guide to install the Darknet, create the label txt file for every picture in GTSDB and transfer the label format from (Xmin, Xmax, Ymin, Ymax) to (X, Y, W, H).

Due to time restrictions and computational costs, for all experiments, we fine-tune these models with the GTSDB dataset and in order to compare with the statistic of Álvaro Arcos García's paper. We also detect and classify traffic sign superclasses based on their shapes and colors: mandatory, prohibitory, and danger.

The models are trained and evaluated on a computer built with an Intel Core i5 CPU, 8 GB RAM 2133 MHz LPDDR3 and the GPU is provided on the Google Colab platform with a Tesla K80 GPU, 12 GB RAM. The spec is shown in *Figure 1*.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.59       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   47C    P8    10W /  70W |      0MiB / 15079MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure 1 : Spec of Google Colab GPU which can only run 12 hours in a row and 2 Jupiter notebook at once, then it needs cool down for another 12 hours.

III. Procedures

The object detection models are divide into two architecture, two-stage, and one-stage detectors. Models in the R-CNN family are all two-stage detectors. The detection happens in two stages: (1) First, the model proposes a set of regions of interest by select search or regional proposal networks. The proposed regions are sparse as the potential bounding box candidates can be infinite. (2) Then a classifier only processes the region candidates. The advantage of R-CNN family is high accuracy which is around 95 %, but the drawback is time-consuming. One-stage detectors skip the region proposal stage and run detection directly over a dense sampling of possible locations. This is faster and simpler but might potentially drag down the performance a bit. Due to autonomous driving applications, the detectors must respond in real-time and even run in the CPU environment. I chose the latest one-stage detector YOLO V3 which improved the performance by adding multi-scale predictions, a better backbone classifier and the residual network stuff compared to the YOLO V2 in Álvaro Arcos García's paper.

Besides YOLO V3, I also used the smaller model YOLO tiny V3 accelerating the model prediction. Even though the mAP dropped 3% compared to YOLO V3, overall

performance still reached 90%. This result shows that the complexity of small models like YOLO tiny V3 is sufficient for traffic sign detection. In contrast to the Microsoft COCO dataset with 80 classes of many different objects, the traffic signs of GTSDB only have simple shapes such as circle, triangle and simple colors like blue, red and black. The model doesn't need so many layers for feature extraction and only need to classify the simple contours and colors. Based on this reason, I cut four layers of YOLO tiny V3 making the model slimmer and called it YOLO tiny tiny. In addition to cutting layers, I enlarged input images as well because we prefer to recognize the traffic sign as far as possible and bigger images enhance the detection of the small object which is also confirmed in Álvaro Arcos García's paper.

IV. **Data Recording**

The experiment of YOLO V3 performs better than any one-stage detector in Álvaro Arcos García's paper. The model structure, inference time in CPU and GPU is shown in *Figure 2*. The result is shown in *Figure 3*.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2 (a) : Structure of YOLO V3

```
Loading weights from yolov3.weights...Done!
data/dog.jpg: Predicted in 19.728576 seconds.
bicycle: 99%
truck: 92%
dog: 100%
```

Figure 2 (b) : Inference time on CPU environment

```
Total BFLOPS 139.512
 Allocate additional workspace_size = 52.43 MB
Loading weights from tiny/one_class/yolov3_last.weights...
 seen 64
Done!
tiny/img/data/00556.jpg: Predicted in 28.007000 milli-seconds.
```

Figure 2 (c) : Inference time on GPU environment

Figure 2 : Model structure, inference time of YOLO V3

```
Loading weights from tiny/one_class/yolov3_last.weights...
 seen 64
Done!

 calculation mAP (mean average precision)...
300
 detections_count = 588, unique_truth_count = 361
class_id = 0, name = prohibitory, ap = 99.26%            (TP = 159, FP = 5)
class_id = 1, name = mandatory, ap = 84.45%      (TP = 33, FP = 2)
class_id = 2, name = danger, ap = 93.54%         (TP = 59, FP = 4)
class_id = 3, name = other, ap = 94.74%          (TP = 77, FP = 7)

 for thresh = 0.25, precision = 0.95, recall = 0.91, F1-score = 0.93
 for thresh = 0.25, TP = 328, FP = 18, FN = 33, average IoU = 75.91 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.929997, or 93.00 %
```

Figure 3 : The prediction result of test dataset of YOLO V3

The experiment of YOLO tiny V3 also performs well under restricted model size. The model structure is shown in *Figure 4*. Inference time in CPU and GPU is shown in *Figure 5*. The result is shown in *Figure 6*.

```
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   608 x 608 x   3   ->   608 x 608 x  16 0.319 BF
    1 max          2 x 2 / 2   608 x 608 x  16   ->   304 x 304 x  16 0.006 BF
    2 conv     32  3 x 3 / 1   304 x 304 x  16   ->   304 x 304 x  32 0.852 BF
    3 max          2 x 2 / 2   304 x 304 x  32   ->   152 x 152 x  32 0.003 BF
    4 conv     64  3 x 3 / 1   152 x 152 x  32   ->   152 x 152 x  64 0.852 BF
    5 max          2 x 2 / 2   152 x 152 x  64   ->    76 x  76 x  64 0.001 BF
    6 conv    128  3 x 3 / 1    76 x  76 x  64   ->    76 x  76 x 128 0.852 BF
    7 max          2 x 2 / 2    76 x  76 x 128   ->    38 x  38 x 128 0.001 BF
    8 conv    256  3 x 3 / 1    38 x  38 x 128   ->    38 x  38 x 256 0.852 BF
    9 max          2 x 2 / 2    38 x  38 x 256   ->    19 x  19 x 256 0.000 BF
   10 conv    512  3 x 3 / 1    19 x  19 x 256   ->    19 x  19 x 512 0.852 BF
   11 max          2 x 2 / 1    19 x  19 x 512   ->    19 x  19 x 512 0.001 BF
   12 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024 3.407 BF
   13 conv    256  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 256 0.189 BF
   14 conv    512  3 x 3 / 1    19 x  19 x 256   ->    19 x  19 x 512 0.852 BF
   15 conv     27  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x  27 0.010 BF
   16 yolo
   17 route  13
   18 conv    128  1 x 1 / 1    19 x  19 x 256   ->    19 x  19 x 128 0.024 BF
   19 upsample            2x    19 x  19 x 128   ->    38 x  38 x 128
   20 route  19 8
   21 conv    256  3 x 3 / 1    38 x  38 x 384   ->    38 x  38 x 256 2.555 BF
   22 conv     27  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x  27 0.020 BF
   23 yolo
```

Figure 4 : The model structure of YOLO tiny V3

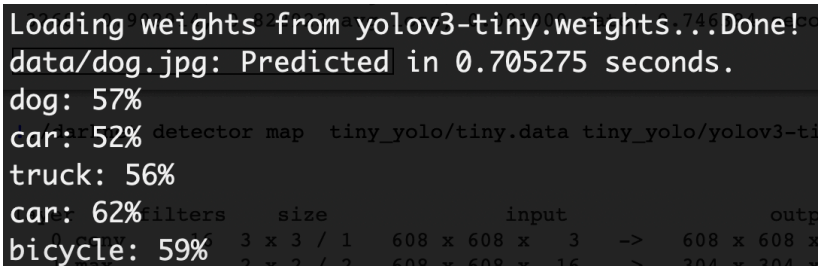

Figure 5 (a) : Inference time of YOLO tinyV3 on CPU

```
Total BFLOPS 11.647
 Allocate additional workspace_size = 52.43 MB
Loading weights from tiny/one_class/yolov3-tiny-voc_final.weights...
 seen 64
Done!
tiny/img/data/00556.jpg: Predicted in 8.940000 milli-seconds.
```

Figure 5 (b) : Inference time of YOLO tiny V3 on GPU

Figure 5 : Inference time of YOLO tiny V3

```
Loading weights from tiny/one_class/yolov3-tiny-voc_8000.weights...
 seen 64
Done!

 calculation mAP (mean average precision)...
300
 detections_count = 775, unique_truth_count = 361
class_id = 0, name = prohibitory, ap = 97.95%            (TP = 159, FP = 11)
class_id = 1, name = mandatory, ap = 85.50%      (TP = 41, FP = 7)
class_id = 2, name = danger, ap = 91.76%         (TP = 57, FP = 4)
class_id = 3, name = other, ap = 88.06%          (TP = 67, FP = 7)

 for thresh = 0.25, precision = 0.92, recall = 0.90, F1-score = 0.91
 for thresh = 0.25, TP = 324, FP = 29, FN = 37, average IoU = 75.38 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.908177, or 90.82 %
Total Detection Time: 120.000000 Seconds
```

Figure 6 : The prediction result of test dataset of YOLO tiny V3

The experiment of my proposed model YOLO tiny tiny even made the small progress compared to YOLO tiny V3. The model structure is shown in *Figure 7*. Inference time in CPU and GPU is shown in *Figure 8*. The result is shown in *Figure 9*.

```
layer     filters    size              input                output
   0 conv     16  3 x 3 / 1   608 x 608 x   3   ->   608 x 608 x  16 0.319 BF
   1 max          2 x 2 / 2   608 x 608 x  16   ->   304 x 304 x  16 0.006 BF
   2 conv     32  3 x 3 / 1   304 x 304 x  16   ->   304 x 304 x  32 0.852 BF
   3 max          2 x 2 / 2   304 x 304 x  32   ->   152 x 152 x  32 0.003 BF
   4 conv     64  3 x 3 / 1   152 x 152 x  32   ->   152 x 152 x  64 0.852 BF
   5 max          2 x 2 / 2   152 x 152 x  64   ->    76 x  76 x  64 0.001 BF
   6 conv    128  3 x 3 / 1    76 x  76 x  64   ->    76 x  76 x 128 0.852 BF
   7 max          2 x 2 / 2    76 x  76 x 128   ->    38 x  38 x 128 0.001 BF
   8 conv    256  3 x 3 / 1    38 x  38 x 128   ->    38 x  38 x 256 0.852 BF
   9 max          2 x 2 / 2    38 x  38 x 256   ->    19 x  19 x 256 0.000 BF
  10 conv    512  3 x 3 / 1    19 x  19 x 256   ->    19 x  19 x 512 0.852 BF
  11 conv     27  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x  27 0.010 BF
  12 yolo
  13 route  9
  14 conv    128  1 x 1 / 1    19 x  19 x 256   ->    19 x  19 x 128 0.024 BF
  15 upsample            2x    19 x  19 x 128   ->    38 x  38 x 128
  16 route  15 8
  17 conv    256  3 x 3 / 1    38 x  38 x 384   ->    38 x  38 x 256 2.555 BF
  18 conv     27  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x  27 0.020 BF
  19 yolo
```

Figure 7 : Model structure of YOLO tiny tiny

```
Loading weights from yolov3-tiny-tiny-416_last.weights...Done!
00002.jpg: Predicted in 0.566050 seconds.
```

Figure 8 (a) : Inference time of YOLO tiny tiny on CPU

```
 Total BFLOPS 7.198
  Allocate additional workspace_size = 31.79 MB
 Loading weights from tiny/one_class/yolov3-tiny-tiny_last.weights...
  seen 64
 Done!
 tiny/img/data/00556.jpg: Predicted in 8.461000 milli-seconds.
```

Figure 8 (b) : Inference time of YOLO tiny tiny on GPU

Figure 8 : Inference time of YOLO tiny tiny

```
Loading weights from tiny/one_class/yolov3-tiny-tiny_last.weights...
 seen 64
Done!

 calculation mAP (mean average precision)...
300
 detections_count = 1337, unique_truth_count = 361
class_id = 0, name = prohibitory, ap = 99.17%            (TP = 160, FP = 9)
class_id = 1, name = mandatory, ap = 86.74%      (TP = 41, FP = 16)
class_id = 2, name = danger, ap = 91.31%         (TP = 57, FP = 7)
class_id = 3, name = other, ap = 88.18%          (TP = 75, FP = 16)

 for thresh = 0.25, precision = 0.87, recall = 0.92, F1-score = 0.90
 for thresh = 0.25, TP = 333, FP = 48, FN = 28, average IoU = 71.18 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.913487, or 91.35 %
Total Detection Time: 72.000000 Seconds
```

Figure 9 : The prediction result of test dataset of YOLO tiny tiny

V. **Analysis**

I compared YOLO V3, YOLO tiny V3 and YOLO tiny tiny with the models which are analyzed in Álvaro Arcos García's paper respect to mAP and executive time. The *Table 1* illustrates the result.

Model	**mAP**	**exec_millis**
Faster R-CNN Resnet 50	91.52	104.0363553
Faster R-CNN Resnet 101	95.08	123.2729175
Faster R-CNN Inception V2	90.62	58.53338971
Faster R-CNN Inception Resnet V2	95.77	442.2206796
R-FCN Resnet 101	95.15	85.45207971
YOLO V2	78.83	21.4810122
YOLO V3	93	28 CPU —19.72 secs
YOLO tiny V3	90.82	8.94 CPU—-0.7052secs
YOLO tiny tiny	91.35	8.461 CPU—-0.566secs

Table 1 : Analysis of models

In Álvaro Arcos García's analysis, only R-CNN family has over 90% precision. However, in terms of executive time, even the lightest Faster R-CNN Inception V2 model still takes 58.533 milli-secs for inference. For the highest precision one-stage detector YOLO V2, 78.83% precision is too low to guarantee driving safety. YOLO V3, YOLO tiny V3, and YOLO tiny tiny progress a lot and meet the requirements of autonomous driving. To compared with YOLO V2, YOLO V3 improved precision to 93% by increasing 7 milli-secs, YOLO tiny V3 and YOLO tiny tiny outperform in the large margin in both accuracy and latency time. In comparison with RCNN family, YOLO tiny tiny only scarifies 4 ~ 5 % precision to make model speed up 443 milli-secs of inference time and it can even respond in real-time in the CPU environment.

VI. **Discussion**

For the YOLO tiny tiny modification, I removed the 4 layers at the last CNN plus max pool layer which is shown in *Figure 10*. Several CNN plus max pool layers before the removed layers represent the feature extraction process in different sizes. The model needs this process to recognize the multi-size pattern in the images. However, in the removed part, the last max pool layer does not shrink the size of input size and the rest CNN layers increase the filter number to 1024 then apply size 1*1 CNN. The intention behind this part is to detect the complex detailed pattern by attention mechanism.

```
10 conv    512  3 x 3 / 1    19 x  19 x 256   ->    19 x  19 x 512 0.852 BF
11 max          2 x 2 / 1    19 x  19 x 512   ->    19 x  19 x 512 0.001 BF
12 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024 3.407 BF
13 conv    256  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 256 0.189 BF
```

Figure 10 : The removed layers of YOLO tiny V3

Although these layers increase the performance of general cases, they are not necessary for relative simple detection like our task. See the analysis in Table 1, the model benefits the structure without these layers. YOLO tiny tiny not only doesn't deteriorate but also take advantage of it.

VII. **Conclusion**

In this praktikum, I applied the latest YOLO V3 and YOLO tiny V3 for the vision-based navigation system of autonomous driving. Due to the structure of residual layers and multi-scale bounding boxes, both of them achieved over 90% precision and lower inference time. For task 2, I proposed the slimmer model YOLO tiny tiny which based on YOLO tiny V3. The smaller model takes advantage of traffic signs' geometric shapes and elementary colors to reach 91% of accuracy and real-time response in the CPU environment. We always need to balance the model complexity and accuracy. Understanding the task's characteristics is the best way to achieve the goal rather than using large CNN models.

VIII. References

Arcos-Garcia, A., Alvarez-Garcia, J. A., & Soria-Morillo, L. M. (2018). Evaluation of deep neural networks for traffic sign detection systems. *Neurocomputing, 316*, 332-344.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

IX. Source codes

See the further information on my GitHub https://github.com/kuanhsunwu/Vision-Based-Navigation-and-Learning-for-Autonomous-Vehicles-

Firstly, download the dataset and the corresponding weights by me google drive https://drive.google.com/open?id=176XhrGdofwgOuHx2Klj1_h3xe645AHGK and save on your google drive as tiny then open the Colab notebook and mount your google drive.