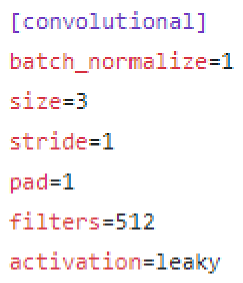
Abstract

This project is to implement an objection detection function which is deployed on dash cameras of vehicles. We apply Yolov3 as our object detection algorithm, since it is capable to achieve real-time image processing and outperforms other object detection algorithms (e.g., SSD, Fast-RCNN) in processing speed and accuracy.

Algorithms Description

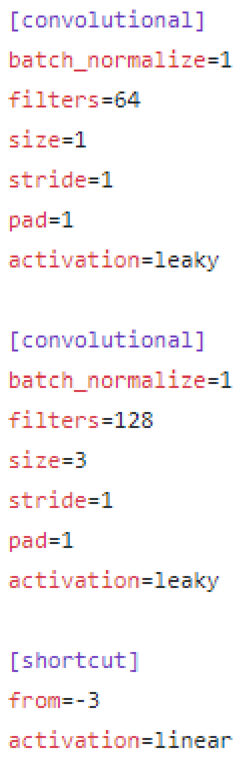
Yolov3

There are 252 layers in Yolov3 in total, including 75 Conv2D layers, 72 Batch Normalization layers, 72 Leaky ReLU activation layers, 23 Add layers, 5 2D Zero Padding layers, 2 Concatenate layers, 2 2D Up-sampling layers, and a very last Input layer. Yolov3 usually combine some layers as a unit. A Cov2D usually is followed by a BatchNormalization layer with a LeakyReLU layer behind. These three layers form a Darknet_Conv2D (DBL) unit. A Residual (Res) unit consist of 2 DBL units and an Add layer which adds the DBL output and the input of Res unit together. In addition, Yolov3 designs Resn units, which stack a Zero Padding layer and a DBL unit with n numbers of Res units together.



```
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky
```

Figure 1. DBL unit



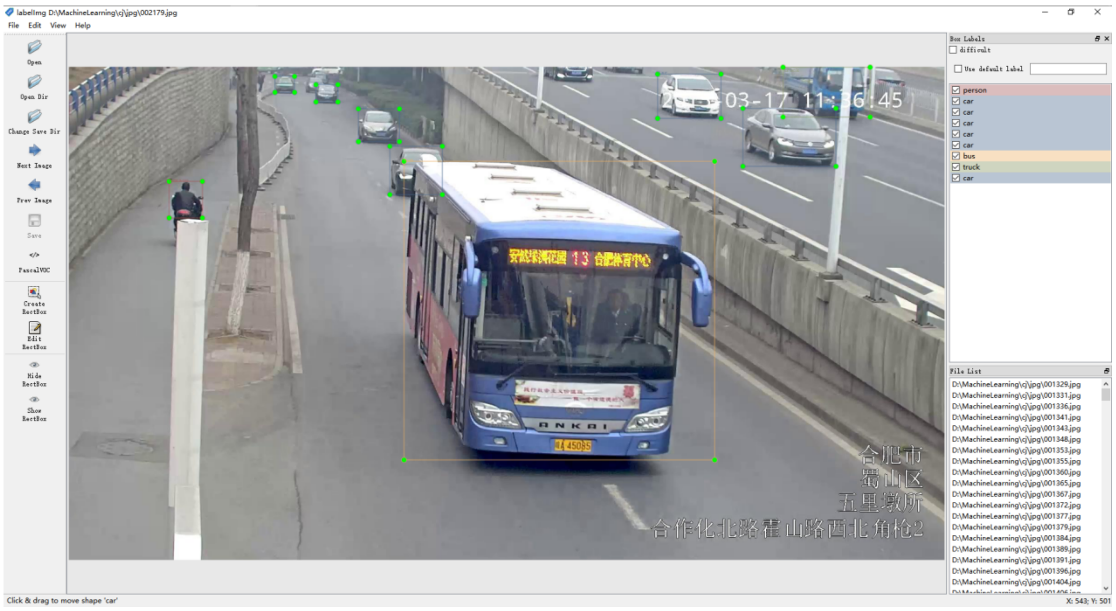
```
[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

Figure 2. Res Unit

Training process

Training dataset preparation

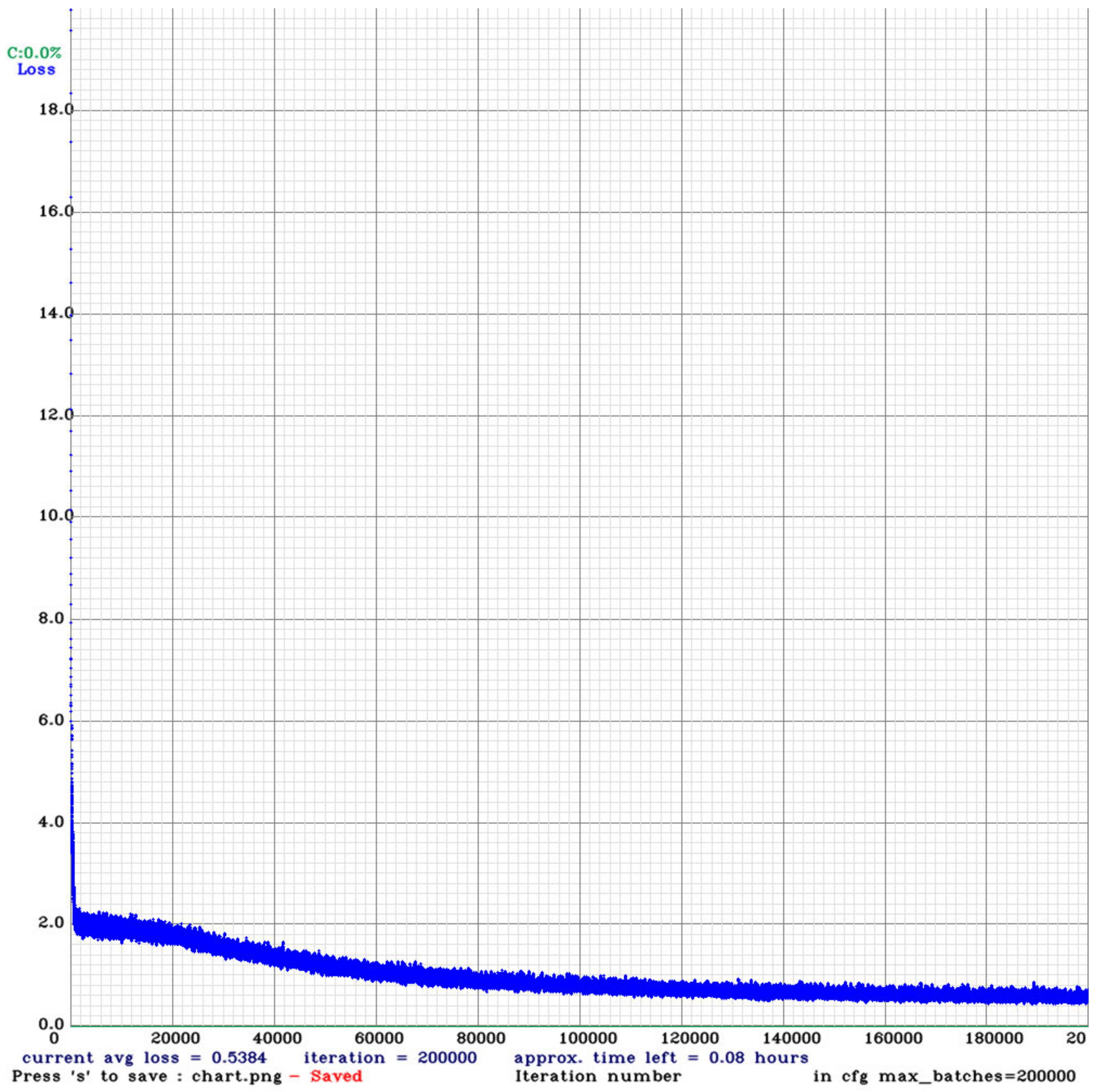
For the training data, we used some part of the VOC dataset which contains similar on road scene. For most part of our training data, we get some urban road monitoring videos and some dashcam video of from the Internet. First, we use ffmpeg to convert these videos into images, and then we label them using the LabelImg to generate the .xml file just like the VOC dataset. One of the image that we labeled is shown below.



D:\MachineLearning\cj\jpg\001329.jpg
D:\MachineLearning\cj\jpg\001331.jpg
D:\MachineLearning\cj\jpg\001336.jpg
D:\MachineLearning\cj\jpg\001341.jpg
D:\MachineLearning\cj\jpg\001343.jpg
D:\MachineLearning\cj\jpg\001348.jpg
D:\MachineLearning\cj\jpg\001353.jpg
D:\MachineLearning\cj\jpg\001355.jpg
D:\MachineLearning\cj\jpg\001360.jpg
D:\MachineLearning\cj\jpg\001365.jpg
D:\MachineLearning\cj\jpg\001367.jpg
D:\MachineLearning\cj\jpg\001372.jpg
D:\MachineLearning\cj\jpg\001377.jpg
D:\MachineLearning\cj\jpg\001379.jpg
D:\MachineLearning\cj\jpg\001384.jpg
D:\MachineLearning\cj\jpg\001389.jpg
D:\MachineLearning\cj\jpg\001391.jpg
D:\MachineLearning\cj\jpg\001396.jpg
D:\MachineLearning\cj\jpg\001404.jpg

Training YOLO v3

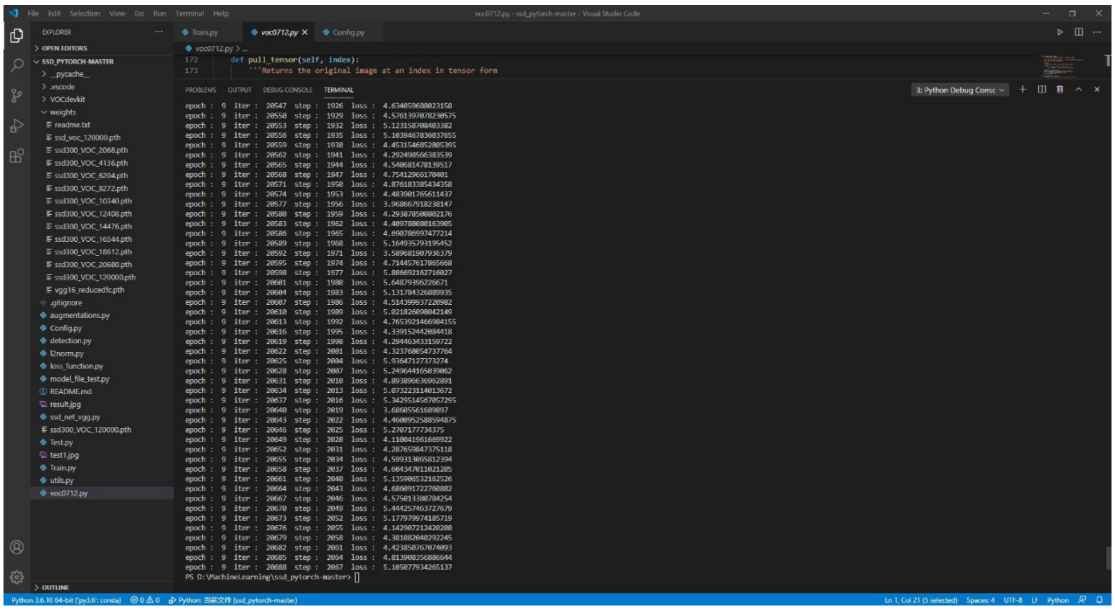
First, we need to compile the YOLO v3 from darknet . After modify the configuration file, which is .data and yolo.cfg to fit our training category. Then, we generate download the pretrain model and start training. We trained 200000 times and the final loss is below 0.5. The training process is shown below.



```
current avg loss = 0.5384    iteration = 200000    approx. time left = 0.08 hours
in cfg max_batches=200000
```

Training SSD

For SSD, we use the pytorch version of SSD (ssd.pytorch). After modify the config.py; VOC0721.py; train.py; eval.py and ssd.py, we started trainig.


```
172     def pull_tensor(self, index):
173         '''Returns the original image at an index in tensor form
```

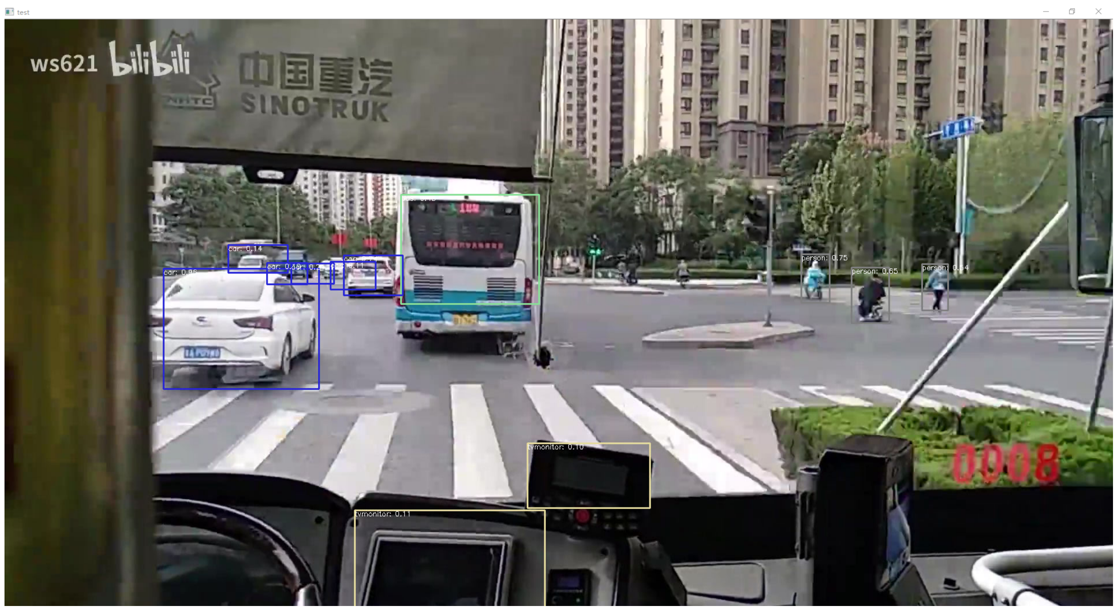
```
epoch :  9  iter :  20547  step :  1926  loss :  4.634059688023158
epoch :  9  iter :  20550  step :  1929  loss :  4.5761397078230575
epoch :  9  iter :  20553  step :  1932  loss :  5.123158708403382
epoch :  9  iter :  20556  step :  1935  loss :  5.1039467836037655
epoch :  9  iter :  20559  step :  1938  loss :  4.4531546852805395
epoch :  9  iter :  20562  step :  1941  loss :  4.292498566383539
epoch :  9  iter :  20565  step :  1944  loss :  4.540681478139517
epoch :  9  iter :  20568  step :  1947  loss :  4.75412966170401
epoch :  9  iter :  20571  step :  1950  loss :  4.876183385434358
epoch :  9  iter :  20574  step :  1953  loss :  4.483901765611437
epoch :  9  iter :  20577  step :  1956  loss :  3.968667918238147
epoch :  9  iter :  20580  step :  1959  loss :  4.293878500802176
epoch :  9  iter :  20583  step :  1962  loss :  4.409788080163905
epoch :  9  iter :  20586  step :  1965  loss :  4.690786997477214
epoch :  9  iter :  20589  step :  1968  loss :  5.164935793195452
epoch :  9  iter :  20592  step :  1971  loss :  3.589681907936379
epoch :  9  iter :  20595  step :  1974  loss :  4.714457617865668
epoch :  9  iter :  20598  step :  1977  loss :  5.886692162716027
epoch :  9  iter :  20601  step :  1980  loss :  5.64879396226671
epoch :  9  iter :  20604  step :  1983  loss :  5.131784326889935
epoch :  9  iter :  20607  step :  1986  loss :  4.514399937220982
epoch :  9  iter :  20610  step :  1989  loss :  5.021826098042149
epoch :  9  iter :  20613  step :  1992  loss :  4.7653921466984155
epoch :  9  iter :  20616  step :  1995  loss :  4.339152442084418
epoch :  9  iter :  20619  step :  1998  loss :  4.294463433159722
epoch :  9  iter :  20622  step :  2001  loss :  4.323768054737764
epoch :  9  iter :  20625  step :  2004  loss :  5.93647127373274
epoch :  9  iter :  20628  step :  2007  loss :  5.249644165039062
epoch :  9  iter :  20631  step :  2010  loss :  4.893896636962891
epoch :  9  iter :  20634  step :  2013  loss :  5.073223114013672
epoch :  9  iter :  20637  step :  2016  loss :  5.3429514567057295
epoch :  9  iter :  20640  step :  2019  loss :  3.68605561689897
epoch :  9  iter :  20643  step :  2022  loss :  4.4600952588594875
epoch :  9  iter :  20646  step :  2025  loss :  5.2707177734375
epoch :  9  iter :  20649  step :  2028  loss :  4.110041961669922
epoch :  9  iter :  20652  step :  2031  loss :  4.287659847375118
epoch :  9  iter :  20655  step :  2034  loss :  4.599313065812394
epoch :  9  iter :  20658  step :  2037  loss :  4.604347011021205
epoch :  9  iter :  20661  step :  2040  loss :  5.135906532162526
epoch :  9  iter :  20664  step :  2043  loss :  4.686091722760882
epoch :  9  iter :  20667  step :  2046  loss :  4.575013380784254
epoch :  9  iter :  20670  step :  2049  loss :  5.444257463727679
epoch :  9  iter :  20673  step :  2052  loss :  5.177979974185719
epoch :  9  iter :  20676  step :  2055  loss :  4.142907212420208
epoch :  9  iter :  20679  step :  2058  loss :  4.381082040292245
epoch :  9  iter :  20682  step :  2061  loss :  4.423850767074093
epoch :  9  iter :  20685  step :  2064  loss :  4.813908356886644
epoch :  9  iter :  20688  step :  2067  loss :  5.105077934265137
PS D:\MachineLearning\ssd_pytorch-master> []
```

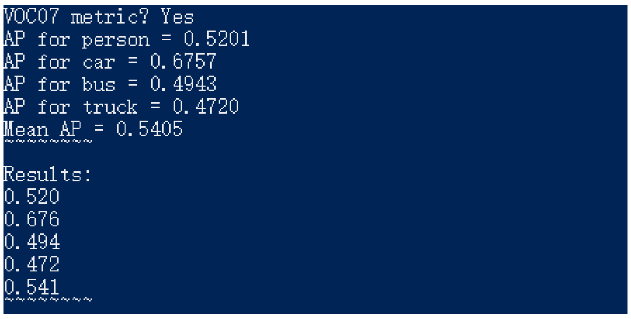
Evaluation

We used mAP to evaluate the training result.

In yolo v3, we get into some trouble that our rename file will mismatch the pictures and the annotation file. So we need to retrain the model.

In SSD, we successfully get the result. But, SSD does not perform well when there are multiple objects overlapping. One of the test result and the mAP are shown below.





```
VOC07 metric? Yes
AP for person = 0.5201
AP for car = 0.6757
AP for bus = 0.4943
AP for truck = 0.4720
Mean AP = 0.5405
~~~~~~~~
Results:
0.520
0.676
0.494
0.472
0.541
~~~~~~~~
```

Expectation

In order to run YOLO v3 on a dashcam, we need to quantize it. After finish the training, we believe that it will fit all the working scenarios.