Traffic Multiple Target Detection on YOLOv2

Junhong Li, Huibin Ge, Ziyang Zhang, Weiqin Wang , Yi Yang
Taiyuan University of Technology, Shanxi, 030600, China
wangweiqin1609@link.tyut.edu.cn

Abstract

Background subtraction arithmetic is one of the practical and efficient moving objects detection algorithms based on still and complicated background, especially on the real-time detection. The paper represents several popular algorithms such as Faster R-CNN, YOLOv1 and YOLOv2 to achieve the detection on the traffic. The result shows YOLO present better than other algorithms, especially the improved version YOLOv2, this method can detect and track multiple targets even when they cross each other under complicated and bad background.

KEYWORDS：*Real-time detection; Faster R-CNN; YOLOv1; YOLOv2*

1. Introduction

In order to realize the purpose of real-time monitoring of road vehicles, we have tried several popular algorithms at the moment. Including YOLOv1, YOLOv2, fast R-CNN and SSD. Among them, of which the effect are prominent fast R-CNN and YOLOv2. YOLO is refreshingly simple: a single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection[1]. It has a very fast rate in detection . First the training on the YOLOv1 has a loss of 18 and the load speed is about 0. 285s/iter. YOLOv2 is better at a loss of 9 and the load speed is about 0.00007s/iter. The detection processing on the Tesla P100-SXM2-16GB in the YOLOv2 is more than 40 FPS. And has a mAP of 52% on the aic1080 and a mAP of 28% on the aic 480.

2. 1 Faster R-CNN

Faster R-CNN is a classic and effective way to solve the Object Detection problem in the field of machine vision. The first choice of Faster R-CNN is because Faster R-CNN is faster than Fast R-CNN, and with VGG net as feature extractor, the mAP can reach 73% on VOC2007. Compared to R-CNN to solve "why do not use CNN do classification" and Fast R-CNN to solve "why not output bounding box and label", Faster R-CNN solves the question: "why use selective search?". This is undoubtedly a historic initiative, precisely because the Faster R-CNN is relatively fas，to achieve real-time detection. First download the project Faster_R-CNN_TF of Fu-Hsiang Chan on github, and the training data is imported into tensorflow using the VOC dataset format. After 24 hours of training, the number of iterations reached 40,000 times. After testing, we found many problems, such as IOU is too small, mAP is relatively large, too few objects marked, and even some marked objects are not accurate. The algorithm performs poorly on the data set, see the following picture and we look for a new algorithm YOLO.

Figure1: the performance on the algorithm Faster R-CNN

2. 2 YOLOv1

YOLOv1 is a real-time object detection algorithm treated as a regression problem. By taking an entire picture resized to 448 × 448 as inputs to the neural network. Some coordinates of bounding box and the class and confidence intervals of objects in box are obtained through the network. Finally, the network selects the appropriate boxes above the threshold.

We used a pre-trained model in order to increase the speed and the efficiency of training. And adding both convolutional and connected layers to pre-trained networks can improve performance[2]. The network system divides the input image into an 7 × 7 grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts 2 bounding boxes and confidence scores for those boxes.

These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. we use Pr(Object)×IOU. If there is no object exists in that cell, the confidence scores would be zero[3].

Our first temp is using the network structure of 24 convolutional layers and 4 maxpool layers followed by 2 fully connected layers. And the final output of our network is the 7 × 7 × 30 tensor of predictions. Then we use a linear activation function for the final layer and other layers using the leaky relu activation. During the training we optimize the following, multi-part loss to as one of the standards to adjust our model.

$$\lambda_{coord}\sum_{i=0}^{s^2}\sum_{j=0}^{B} I_{ij}^{obj}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]$$

$$+\lambda_{coord}\sum_{i=0}^{s^2}\sum_{j=0}^{B} I_{ij}^{obj}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]$$

$$+\sum_{i=0}^{s^2}\sum_{j=0}^{B} I_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$

$$+\lambda_{noobj}\sum_{i=0}^{s^2}\sum_{j=0}^{B} I_{ij}^{noobj}\left(C_i-\hat{C}_i\right)^2$$

$$+\sum_{i=0}^{s^2} I_{j}^{obj}\sum_{c\in classes}\left(p_i(c)-\hat{p}_i(c)\right)^2$$

The loss is sum-squared because it is easy to optimize. But the loss function only penalizes classification error if an object is present in that grid cell. It also only penalizes bounding box coordinate error if that predictor is "responsible" for the ground truth box.

To deal with the overheat, train a convolutional neural network to perform localization and adapt the localizer to perform detection[4]. We use dropout and extensive data augmentation. A dropout layer with rate = . 5 after the first connected layer prevents coadaptation between layers[5]. We use random scaling and translations of up to 20% of the original image size. We also randomly adjust the exposure and saturation of the image by up to a factor of 1. 5 in the HSV color space.

We train the network for about 25 epochs on the training and validation data sets from aic480 and aic1080. Throughout training we use a batch size of 45. The first training begin with a learning rate 0.0001 and then learning rate decay every 30000 iterations . After 2 days training the loss is about 22. Then we try to adjust the learning rate to 0.1 and decay every 10000 iterations . This time the speed is much faster and the loss is down to 20. However the result isn't satisfactory. The number of network low-level parameters and the number of high-level network parameters affect the overall performance of the network to a large extent. When the number of network parameters is increased to a critical value, the number of parameters in the guarantee under the condition of constant low layer network parameters and increase the number of parameters can make the network level, to further enhance the performance[6]. So we decide to redefine the network. We reduce 2 convolutional layers each time, when we reduce it to 18 layers. As a result, the loss is also reduced to 12. As we continued to reduce the convolutional layers, loss began to rise. Then we try to reduce fully connected layers. But the performance is not good either. See the following picture.

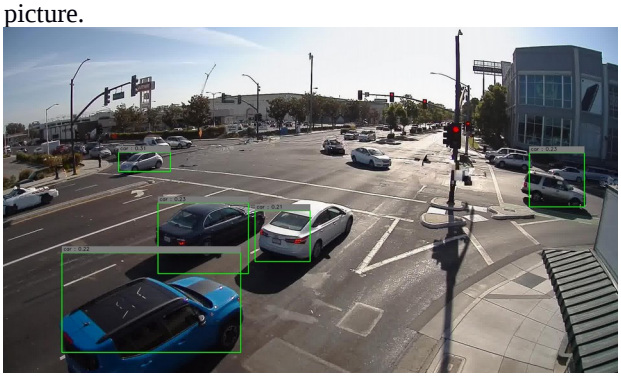

Figure2: the performance on the algorithm YOLOv1

2. 3 YOLO v2

Using a series of methods to improve the YOLO, while maintaining the original speed, improve the accuracy, YOLOv2[7].

By training aic480 and aic1080 of datasets including train and val, we obtain more accurate up to ten thousand weight files.

Using this method we've trained YOLOv2 simultaneously on the aic480 training and validating dataset and the aic1080 training and validating dataset by different configures. As a result, we've obtained more accurate up to ten thousand weight files. At the same time, we do not use only one network configuration for weight file training. After the iteration number is achieved, we also make some improvements on this configuration such as adjusting the learning rate, increase the number of convolution for accurate prediction again. After that, we also get the same iteration number of weight files.

Besides, we also keep the weight files with different iterations. By testing of the same picture using different weight files, we come to the conclusion that large iterations better than fewer ones. In the process, we find an inevitable issue that model instability especially during early iterations. Most of the instability comes from predicting the (x, y) locations for the box. In region proposal networks the network predicts values tx and ty and the (x, y) center coordinates are calculated as:

$$x = (t_x * w_a) - x_a \qquad (1)$$

$$y = (t_y * h_a) - y_a \quad (2)$$

For example, a prediction of tx > 0 would shift the box to the right , a prediction of tx < 0 would shift it to the left.

The network predicts 5 bounding boxes at each cell in the output feature map. The network predicts 5 coordinates for each bounding box, tx, ty, tw, th, and to. If the cell is offset from the top left corner of the image by (cx, cy) and the bounding box prior has width and height pw, ph, then the predictions correspond to:

$$b_x = \sigma(t_x) + c_x \quad (3)$$

$$b_y = \sigma(t_y) + c_y \quad (4)$$

$$b_w = p_w e^{t_w} \quad (5)$$

$$b_h = p_h e^{t_h} \quad (6)$$

$$P_r(object) * IOT(b, object) = \sigma(t_o) \quad (7)$$

Because we limit the location prediction, the parametrization is easier to learn, making the network more stable. By using dimension clusters along with directly predicting the bounding box center location, YOLO is improved by almost 5% over the version with anchor boxes.

This modified YOLO predicts detections on a 13 × 13 feature map. While this is sufficient for large objects, it may benefit from finer grained features for localizing smaller objects. Although Faster R-CNN and SSD both run their proposal networks at various feature maps in the network to get a range of resolutions, we take a different approach, simply adding a passthrough layer that brings features from an earlier layer at 26 × 26 resolution. And the passthrough layer links the higher and low resolution features together, by stacking adjacent features into different channels instead of spatial locations, similar to the identity mappings in ResNet. This turns the 26 × 26 × 512 feature map into a 13 × 13 × 2048 feature map, which can be concatenated with the original features. Our detector runs on top of this expanded feature map so that it has access to fine grained features.

The multi-scale training method is adopted to make the model adapt to different input sizes.

As we all known, the original YOLO uses an input resolution of 448 × 448. But with the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. In every 8 batches our network randomly chooses a new image dimension size with limits of 320 to 608. Thus the smallest option is 320 × 320 and the largest is 608 × 608. We resize the network to that dimension and continue training.

This measure forces the network to learn to predict well across a variety of input dimensions. It means the same network can predict detections at different resolutions. The network runs faster at smaller sizes so YOLOv2 offers an easy tradeoff between speed and accuracy. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy.

Most detection frameworks rely on VGG-16 as the base feature extractor[8]. VGG-16 is a powerful, accurate classification network but it is needlessly complex. The convolutional layers of VGG-16 require 30.69 billions floating point operations for a single pass over a single image at 224 × 224 resolution. The YOLO framework uses a custom network based on the Googlenet architecture[9]. We propose a classification model to be used as the base of YOLOv2. Similar to the VGG models we use mostly 3 × 3 filters and double the number of channels after every pooling step[7]. We use batch normalization to stabilize training, speed up convergence, and regularize the model[10].

During training we use standard data augmentation tricks including random crops, rotations, and hue, saturation, and exposure shifts. As discussed above, after our initial training on images at 224 × 224 we fine tune our network at a larger size, 448. For this fine tuning we train with the above parameters but for only 10 epochs and starting at a learning rate of 0.001.

We modify this network for detection by removing the last convolutional layer and instead adding on three 3 × 3 convolutional layers with 1024 filters each followed by a final 1 × 1 convolutional layer with the number of outputs we need for detection. For VOC we predict 5 boxes with 5 coordinates each and 15 classes per box so 100 filters. We also add a passthrough layer from the final 3 × 3 × 512 layer to the second to last convolutional layer so that our model can use fine grain features.

We train the network for 88 epochs with a starting learning rate of 0.001, dividing it by 10 at 30 and 45 epochs. We use a weight decay of 0.0005 and momentum of 0.9. We use a similar data augmentation to YOLO and SSD with random crops, color shifting, etc. We use the same training strategy on VOC.

The following conclusions are obtained:

We use YOLOv2, real-time detection systems. YOLOv2 is state-of-the-art and faster than other detection systems across a variety of detection datasets. Further-more, it can be run at a variety of image sizes to provide a smooth tradeoff between speed and accuracy.

Many of our techniques generalize outside of object detection. Dataset combination using hierarchical classification would be useful in the classification and segmentation domains. Training techniques like multi-scale training could provide benefit across a variety of visual tasks.

For future work we hope to use similar techniques for weakly supervised image segmentation. We also plan to improve our detection results using more powerful

matching strategies for assigning weak labels to classification data during training. Computer vision is blessed with an enormous amount of labelled data. We will continue looking for ways to bring different sources and structures of data together to make stronger models of the visual world.

Our joint training allows YOLOv2 to predict detections for object classes which is unlabeled detection data.

Our method leverages labeled detection images to learn to precisely localize objects while it uses classification images to increase its vocabulary and robustness.

Batch normalization helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

For YOLOv2 we first fine tune the classification network at the full 448 × 448 resolution for 10 epochs on ImageNet.

YOLO's convolutional layers downsample the image by a factor of 32, so by using an input image of 416 we get an output feature map of 13 × 13. And the performance is much better than the algorithms.

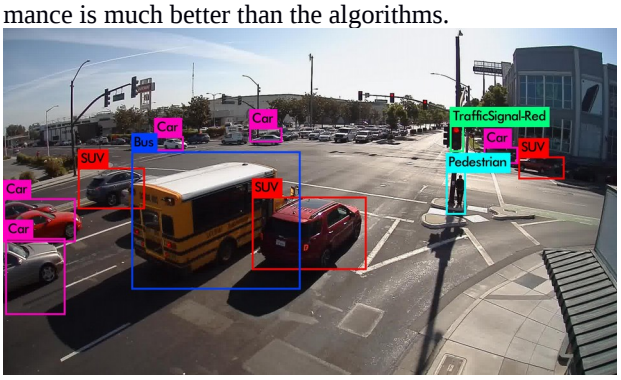

Figure3: the performance on the YOLO v2

2. Test

After comparing with three algorithms on Tesla P100, we trained 2 weight files on YOLOv2 to validate and test. First weight file is trained on the aic1080 with 80000 iterations, the second is trained on the aic480 with 50000 iterations, and we get the AP and F1-score of each object.

Table1: the mAP on the aic 480

Class	AP	F1-score
Bicycle	0. 000000	0. 000000
Bus	0. 242073	0. 413793
Car	0. 662296	0. 720890
LargeTruck	0. 216222	0. 413002
MediumTruck	0. 210589	0. 389215
Motorcycle	0. 272727	0. 500000
Pedestrian	0. 000000	0. 000000
SUV	0. 543668	0. 670706
SmallTruck	0. 447620	0. 605061
Van	0. 232815	0. 381829
Localization	0. 753812	0. 822263
Overall mAP	0. 325620	

From table1, the test APs on the Car, SUV and Localization are high (more than 0. 5) since these objects appear more frequently. But Van and Medium Truck are below 0. 5, we consider Van is similar with SUV, and it is bad that we can not predict the pedestrian.

Table2: the mAP on the aic1080

class	AP	F1-score
Bicycle	0. 202309	0. 372243
Bus	0. 332525	0. 472610
Car	0. 517973	0. 606611
GroupOfPeople	0. 092580	0. 210286
LargeTruck	0. 192787	0. 333152
MediumTruck	0. 312638	0. 499872
Motorcycle	0. 000000	0. 000000
Pedestrian	0. 061460	0. 179663
SUV	0. 431840	0. 581575
SmallTruck	0. 426725	0. 587486
TrafficSignal-Green	0. 099842	0. 241642
TrafficSignal-Red	0. 315179	0. 418002
TrafficSignal-Yellow	0. 117528	0. 242956
Van	0. 228392	0. 398169
Localization	0. 502400	0. 398169
Overall mAP	0. 255612	

From the table2 , we can see that Car , TrafficSignal-Red , and small truck have a good performance. While the others are relatively poor. We consider the reason is that we do not train them enough. We lack time and equipment to train it. Since we have two teams using only one Tesla P100.

3. References

[1] S. Ren, K. He, R. B. Girshick, X. Zhang, and J. Sun. Object Detection networks on convolutional feature maps. CoRR, abs/1504. 06066, 2015. 3, 7

[2] S. Ren, K. He, R. B. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. CoRR, abs/1504. 06066, 2015. 3, 7

[3] Redmon J, Divvala S, Girshick R, et al. You Only Look Once: Unified, Real-Time Object Detection[J]. 2015:779-788.

[4] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localiza tion and detection using convolutional networks. CoRR, abs/1312. 6229, 2013.

[5] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207. 0580, 2012.

[6] The Influence of the Amount of Parameters in Different Layers on the Performance of Deep Learning Models[J]. Computer science and Applications, 2015,

05(12): 445-453. http://dx. doi. org/10. 12677/CSA. 2015. 512056
[7] Redmon J, Farhadi A. YOLO9000: Better, Faster, Stronger[J]. 2016.
[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409. 1556, 2014. 2, 5
[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, abs/1409. 4842, 2014. 5
[10] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502. 03167, 2015. 2, 5