Design of lightweight pedestrian detection network in railway scenes

Chuanyi Han¹, Jinlei Zhu¹, Feng Li², Shengkai Wan² and Houjin Chen¹*

¹School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, 100044, China
²Beijing Century Real Technology Co., Ltd., Beijing, 100085, China
*Corresponding author’s e-mail: hjchen@bjtu.edu.cn

Abstract. The deployment of many advanced pedestrian detection applications is largely hindered by the high computational cost of deep convolutional neural networks (CNNs). In this paper, we propose a two-step pruning method to design a lightweight pedestrian detection network in railway scenes. The first step is feature pyramid network (FPN) pruning, which utilizes the characteristic of pedestrian in railway scenes and the FPN structure in YOLOv3. The second step is regular channel pruning, which utilizes network slimming knowledge and is an accelerator-friendly pruning strategy. Our two-step pruning method gives about 88% reduction in parameters and about 74% reduction in computing complexity with comparable detection accuracy.

1. Introduction

In recent years, with the advancement of convolutional neural networks (CNNs), computer vision is developing rapidly. Object detection is an important branch of computer vision and pedestrian detection is an important branch of object detection. Pedestrian detection is widely used in video surveillance [1], autonomous driving [2], robots [3] and other fields. Our work focuses on pedestrian detection in railway monitoring scenarios, and the goal is to design a highly accurate and lightweight pedestrian detection model for railway security.

There are two ways to design pedestrian detection networks. One is dedicated object private network design, such as PCN [4], ExtFCF [5]. For specific detection tasks, their detection accuracy is usually high, but the design process is very complicated. The other is general object detection network, such as ECP Faster R-CNN [6], SAF R-CNN [7]. These networks usually have better generalization performance. Although their detection accuracy may be inferior to the private network, there is a lot of room for optimization. We choose the second way to design our network based on the principles of easy design, fine accuracy and lightweight.

The general object detection network can be divided into two-stage and one-stage networks. Regarding the two-stage network, Ross Girshick et al. [8] first apply CNN to object detection task. Subsequently more advanced Fast R-CNN [9] and Faster R-CNN [10] are proposed. Regarding the one-stage network, Redmon Joseph et al. [11] first design YOLO. Later SSD [12] and RetinaNet [13] are also developed. More advanced anchor-free target detectors appear recently, such as CornerNet [14] and CenterNet [15]. In contrast, the one-stage network is usually lighter than the two-stage network and the emerging one-stage network is also not inferior in detection accuracy. In our task, YOLOv3 [16] is chosen as our preferred network for its excellent balance between accuracy and speed.
Although YOLOv3 is lightweight enough, if the network is directly deployed on the embedded side, it will still cause a huge delay due to the heavy memory operations and the huge calculations, so the model needs to be further compressed. Common model compression methods include low-rank decomposition [17], model quantization [18] [19], weight pruning [18] [20], network structure pruning [21] [22] and neural network structure learning [23]. In this paper, we introduce a two-step pruning method which can be classified as network structure pruning. First, according to the characteristics of the pedestrian scale in the railway scene, the branches of the FPN structure in YOLOv3 are partly pruned. Then we follow the channel pruning strategy proposed by Zhuang Liu et al. [22] and apply it to YOLOv3. Furthermore, an accelerator-friendly pruning strategy is also utilized, which is faster and a bit more accurate than Liu’s pruning strategy.

2. Two-step pruning methods

2.1. YOLOv3 person detection network

YOLOv3 was proposed by Redmon Joseph et al. in 2018, and it was trained with the COCO dataset [24]. The backbone of YOLOv3 is Darknet-53, which is similar to ResNet [25] in structure but more efficient. YOLOv3 uses a multiscale prediction strategy similar to the feature pyramid network (FPN) [26]. Figure 1 shows a modified YOLOv3 network that only detects pedestrian. The channel numbers of YOLO output layer are changed from 255 to 18, and the input size of the network is 608*608. We call this network YOLOv3-PDN in the rest of the paper.

![Figure 1. YOLOv3-PDN. The blue dotted line represents the FPN structure in YOLOv3-PDN.](image)

2.2. FPN pruning

2.2.1. Statistics for the dataset. Our task is to detect pedestrian in the railway scene. The images in Figure 2 shows some actual scenes. We newly create a railway scene dataset containing 4000 images for training and 1000 images for testing. The height of 17407 pedestrians is calculated in 4000 training images, and a blue columnar distribution chart is drawn in Figure 3, in which the range of pedestrian height is the abscissa and the number of pedestrians is the ordinate. At the same time, we randomly select 10000 images of the COCO dataset [25], a total of 18229 pedestrians, and plot the orange columnar distribution in Figure 3. The blue bar chart in Figure 3 shows that in railway scenes pedestrians are concentrated in small scales. The yellow bar chart shows that the pedestrians are distributed on various scales in general scenes. In the statistical bar chart, the number of pedestrians with a height greater than 180 pixels only accounts for 1% of the total number of pedestrians in railway scenes. While the scale distribution span of pedestrians is large in COCO dataset and the number of pedestrians with
a height greater than 180 pixels accounts for 50% of the total number of pedestrians. We will make full use of the prior knowledge of small human scale in the railway scene for network pruning.

Figure 2. Pedestrian pictures in some railway scenes. The picture size is 1920 * 1080 and the scale of pedestrians is very small.

Figure 3. The pedestrian scale distribution in the railway scene dataset and the COCO dataset.

2.2.2. Process of FPN pruning. FPN pruning is especially designed for the FPN structure in YOLOv3-PDN. In Figure 4, two YOLO output branches are pruned. Specially, we retain 4 convolutional layers before each upsampling layer in order to fully fuse the features.

Figure 4. The process of FPN pruning.
YOLOv3-PDN has three YOLO output layers of different sizes which are 19 * 19, 38 * 38 and 76 * 76. Each grid in each Yolo output layer is responsible for predicting 3 bounding boxes. (1) shows the width and height prediction equation of each bounding box.

\[ \begin{align*}
    b_w &= p_w e^{\gamma} \\
    b_h &= p_h e^{\beta}
\end{align*} \]  

(1)

An exponent term with \( \epsilon \) as the base is introduced as a multiplication factor in the prediction of \( b_w \) and \( b_h \), which greatly increases the dynamic range of the predicted value of \( b_w \) and \( b_h \). The value of \( p_w \) and \( p_h \) corresponds to the prior value of the width and height of the bounding box to be predicted. In original YOLOv3, their values are \((10,13), (16,30), (33,23), (30,61), (62,45), (59,119), (116, 90), (156, 198) \) and \((373, 326)\). Among them, the \( 19 \times 19 \) YOLO output layer corresponds to the three large scales \((116, 90), (156, 198), (373, 326)\), and the prior height ranges from 90 to 326. The \( 38 \times 38 \) YOLO output layer corresponds to \((30, 61), (62, 45), (59, 119)\), and the prior height ranges from 45 to 119. The YOLO output layer of \( 76 \times 76 \) corresponds to \((10, 13), (16, 30), (33, 23)\), and the prior height ranges from 13 to 30. As shown in Figure 3, the pedestrian height of the railway scene is mainly distributed at \((20, 140)\). During the detection process, the image of \( 1920 \times 1080 \) is scaled to \( 608 \times 608 \), and the corresponding pedestrian height is adjusted to \((11, 79)\), which has little intersection with \((90, 326)\) and \((45,119)\). For the two YOLO output layers of \( 19 \times 19 \) and \( 38 \times 38 \), pruning should be carried out, because the large prior height scale has little help in detecting small targets but brings redundant calculations.

We perform 3-clustering on the railway dataset to obtain a new set of bounding box priors which are \((7, 11), (14, 30), (31, 66)\). The prior height ranges from 11 to 66. In addition, as shown in Figure 4, our pruning method reduces the redundant YOLO layer while retaining the main structure of the FPN. The final YOLO output layer has both deep semantic information and shallow fine-grained information, and it retains the detection accuracy for small targets.

2.3. Regular channel pruning

Zhuang Liu [22] utilizes the \( \gamma \) coefficients of Batch Normalization (BN) layer for channel pruning. We apply Liu’s pruning method to YOLOv3-PDN. The main steps include sparse training and channel pruning. Especially in the process of channel pruning, we improve the pruning strategy for accelerator deployment.

2.3.1. Theory of channel pruning. According to Figure 1, YOLOv3-PDN contains many CBL structures. Three layers of CONV, BN and RELU are connected in series. The core of our pruning strategy is the sparse training of \( \gamma \) scaling coefficients in the BN layer. The conversion equation of the BN layer is shown in (2).

\[ \hat{z} = \frac{z_{in} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad z_{out} = \gamma \hat{z} + \beta \]  

(2)

Where the \( z_{in} \) and \( z_{out} \) are the input and output of the BN layer. \( B \) means mini-batch. \( \mu_B \) and \( \sigma_B \) represent the mean and standard deviation on the mini-batch \( B \). \( \gamma \) and \( \beta \) represent the trainable scaling coefficients and offsets. The sparse training only needs to add the L1 regularization term of the \( \gamma \) coefficients to the original loss function of YOLOv3, as is shown in (3), and \( \alpha \) is the weight of the L1 regularization term.

\[ L = \text{loss}_{\text{yolo}} + \alpha \sum_{\gamma \in \Gamma} f(\gamma), \quad f(\gamma) = |\gamma| \]  

(3)

It can be known from (2) that the multiplication factor \( \gamma \) coefficient has a linear scaling effect on the input feature map of the BN layer. The value of the \( \gamma \) coefficient will directly determine the value
of the output feature map. The closer the value is to zero, the less important the feature map is. Therefore, we can directly decide whether to prune the current feature map (channel) according to the value of the $\gamma$ coefficient after sparse training. Figure 5 shows the details. After sparse training, many $\gamma$ coefficients approach zero and they are marked in red in Figure 5. When the numbers on the input layers multiply the red numbers, the values of output layers tend to zero.

![Figure 5](image)

**Figure 5.** Illustration of the principle of channel pruning.

### 2.3.2. From “extreme” to “regular”.

The original pruning strategy of Liu [22] is to cut as many channels as possible while retaining accuracy. We call this strategy extreme channel pruning. However, in the actual deployment of accelerator such as GPU and FPGA, it is necessary to send a regular number of channels which is usually $2^n$ ($n \geq 3$) to obtain the maximum parallel computing capacity according to the characteristics of the accelerator. If the number of channels is less than $2^n$, the computing resources for $2^n$ channels will also be occupied, which leads to a waste of computing resources. Therefore, we add the constraint of regular channel to the extreme channel pruning, shown in Figure 6. We increase 3 channels to 8 channels and 11 channels to 16 channels to meet the regular requirements. All the orange channels will be saved and all the blue channels will be cut. During the specific design process, we first obtain the number of remaining channels according to the pruning ratio, and then adjust the number of remaining channels to the closest and larger $2^n$. Finally, we sort the corresponding channels and cut off the excess channels with the increase of $\gamma$ coefficient.

![Figure 6](image)

**Figure 6.** Illustration of the principle of regular channel pruning

### 3. Experiment

We utilize a public version [27] of YOLOv3 implemented by PyTorch to complete the specific design and pruning process. The pruning result is tested on the self-create railway scene dataset. According to
the $AP_{50}$ standard of COCO dataset [25], the original YOLOv3-PDN is used as a benchmark to compare the AP values, the parameter values, and the calculations of FPN pruning with regular channel pruning. Specially, we first obtain the network after FPN pruning, and then perform regular channel pruning on the basis of the pruned network. The results are shown in Table 1. According to Table 1, when the AP is 82.5, we reduce the parameter amount by 88% and the calculation amount by 74%. It can be noted that the accuracy decrease is mainly concentrated on FPN pruning, and the regular channel pruning only brings about 0.4 percentage accuracy drop. Figure 7 shows the actual detection results. We can find that YOLOv3-PDN still performs well for the detection of small-scale pedestrian even after huge pruning.

Table 1. This table compares the accuracy, parameters and calculations of each pruning step with the original YOLOv3-PDN.

|                  | AP   | Parameters | Pruned | FLOPs  | Pruned |
|------------------|------|------------|--------|--------|--------|
| YOLOv3-PDN       | 84.6 | 62.57M     |        | 141.5G |        |
| FPN pruning      | 82.9 | 42.55M     | 32%    | 121.4G | 14%    |
| Regular channel pruning | 82.5 | 7.43M | 88% | 36.9G | 74% |

Figure 7. The actual detection results.

Original training of YOLOv3-PDN. We utilize darknet pretraining weights to make the model converge quickly and retain the excellent detection performance of the original YOLOv3. The traditional gradient descent method is used during training. We set the momentum to 0.9, the weight attenuation to 0.0005, the learning rate to 0.001, the batchsize to 64 and the iteration epochs to 100.

FPN pruning. We retrain the pruned model on the training set by 80 epochs due to the modified network structure, and the hyperparameters are the same as the original training.

Regular channel pruning. During the sparse training process, we set the $\alpha$ weight to 0.001, the pruning ratio to 70% and the iteration epochs to 300. After pruning, we finetune 100 epochs on the training set to restore accuracy, and the hyperparameters are the same as the original training.

Table 2. Comparison of the actual performance of the extreme channel pruning and regular channel pruning on 2080ti.

| Pruning method     | AP   | Inference Time(s) |
|--------------------|------|-------------------|
| YOLOv3-PDN-FPN pruning | 82.9 | 0.023             |
| Extreme channel pruning | 82.2 | 0.013             |
| Regular channel pruning | **82.5** | **0.011**         |
To verify the accelerator-friendliness of regular channel pruning, we use YOLOv3-PDN with FPN pruning as a control group and test the extreme channel pruning used by Liu [22] and our regular channel pruning on 2080ti, respectively. The results are shown in Table 2. It can be seen that the speed of the regular channel pruning has a slight advantage when applied to the accelerator. It is worth mentioning that the AP of the regular channel pruning is also slightly higher than the extreme channel pruning. We consider that regular channel pruning retains more channels which are still helpful for prediction.

4. Contribution
The main contribution of our work is to design a lightweight network for specific scenario. In many object detection applications, the FPN pruning is also suitable for the tasks when the target scale is small or big and the FPN structure is used for detection. In the channel pruning, we take the regular channels requirements for accelerator deployment into consideration. The regular channel pruning utilizes accelerator resources more efficiently, and it is instructive for other deployment-oriented channel pruning work.

5. Conclusion
In this paper, we design a pedestrian detection network with YOLOv3 as the main body for the task of railway security. A two-step pruning method is proposed for the demand of lightweight network deployment. Firstly, we take full advantage of the small human scale in the railway scenes and prune the FPN structure of the YOLOv3-PDN. Secondly, we utilize the network slimming knowledge and develop an accelerator-friendly channel pruning strategy. Finally, experimental results show that we reduce the amount of network parameters by 88%, the amount of calculations by 74%, and the pruned network only introduces a loss of accuracy of 2.1 percentage. Our work greatly reduces the runtime memory consumption and computing resource requirements of original YOLOv3-PDN, which lays the foundation for future terminal deployment of network.

Acknowledgments
This work was partly supported by Key Research Project of Shandong Province(2019TSLH0206) and Education Foundation Program of Beijing Jiaotong University under Grant 9907005501.

References
[1] M. Bilal, A. Khan, M. U. Karim Khan and C. Kyung. (2017) A low-complexity pedestrian detection framework for smart video surveillance systems. IEEE Transactions on Circuits and Systems for Video Technology, 27: 2260-2273.
[2] A. Broggi, A. Zelinsky, U. Ozguner, and C. Laugier. (2016) Intelligent vehicles. In: Springer Handbook of Robotics. New York, USA. pp. 1627-1656.
[3] B. Nam, S. I. Kang, and H. Hong. (2011) Pedestrian detection system based on stereo vision for mobile robot. In: FCV. Ulsan, Korea. pp. 1-7.
[4] S. Wang, J. Cheng, H. Liu, F. Wang and H. Zhou. (2018) Pedestrian detection via body part semantic and contextual information with dnn. IEEE Transactions on Multimedia, 20: 3148-3159.
[5] M. You, Y. Zhang, C. Shen and X. Zhang. (2018) An extended filtered channel framework for pedestrian detection. IEEE Transactions on Intelligent Transportation Systems, 19: 1640-1651.
[6] M. Braun, S. Krebs, F. Flohr, and D. M. Gavrila. (2018) The EuroCity persons dataset: A novel benchmark for object detection. https://arxiv.org/abs/1805.07193.
[7] J. Li, X. Liang, S. Shen, T. Xu, J. Feng and S. Yan. (2018) Scale-Aware Fast R-CNN for Pedestrian Detection. IEEE Transactions on Multimedia, 20: 985-996.
[8] R. Girshick, J. Donahue, T. Darrell, J. Malik. (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR. Columbus, USA. pp. 580-587.
[9] R. Girshick. Fast R-CNN. (2015) In: ICCV. Santiago, Chile. pp. 1440-1448.
[10] Ren, S., He, K., Girshick, R. and Sun, J. (2015) Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: NIPS. Montreal, Canada. pp. 1137-1149.

[11] Redmon J, Divvala S, Girshick R, et al. (2016) You only look once: Unified, real-time object detection[C]. In: CVPR. Las Vegas, USA. pp. 779-788.

[12] Liu W, Anguelov D, Erhan D, et al. (2016) SSD: Single Shot MultiBox Detector. In: ECCV. Amsterdam, Netherlands. pp. 2016:21-37.

[13] Lin T Y, Goyal P, Girshick R, et al. (2017) Focal Loss for Dense Object Detection. In: ICCV. Venice, Italy. 2017: 2999-3007.

[14] Hei Law and Jia Deng. (2018) CornerNet: Detecting objects as paired keypoints. In: ECCV. Munich, Germany. pp. 734-750.

[15] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang and Q. Tian. (2019) CenterNet: Keypoint Triplets for Object Detection. In: ICCV. Seoul, Korea (South). pp. 6568-6577.

[16] Redmon, J., and A. Farhadi. (2018) YOLOv3: an incremental improvement. arXiv preprint arXiv:1804.02767. https://pjreddie.com/darknet/yolo/.

[17] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. (2014) Exploiting linear structure within convolutional networks for efficient evaluation. In: NIPS. Montreal, Canada. pp. 1269-1277.

[18] S. Han, J. Pool, J. Tran, and W. Dally. (2015) Learning both weights and connections for efficient neural network. In NIPS. Montreal, Canada. pp. 1135–1143.

[19] M. Courbariaux and Y. Bengio. (2016) BinaryNet: Training deep neural networks with weights and activations constrained to +1 or -1. arXiv preprint arXiv:1602.02830.

[20] S. Srinivas, A. Subramanya, and R. V. Babu. (2016) Training sparse neural networks. CoRR, abs/1611.06694.

[21] H. Li, A. Kadav, I. Durbanovic, H. Samet, and H. P. Graf. (2016) Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710.

[22] Liu, Zhuang, et al. (2017) Learning efficient convolutional networks through network slimming. In: ICCV. Venice, Italy. pp. 2736-2744.

[23] B. Baker, O. Gupta, N. Naik, and R. Raskar. (2017) Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167.

[24] Lin, T.-Y. et al. (2014) Microsoft COCO: Common Objects in Context. In: ECCV. Zurich, Switzerland. pp. 740–755.

[25] He, K., Zhang, X., Ren, S. and Sun, J. (2016) Deep Residual Learning for Image Recognition. In: CVPR. Las Vegas, NV, USA. pp. 770–778.

[26] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. (2017) Feature pyramid networks for object detection. In: CVPR. Hawaii, USA. pp. 2117–2125.

[27] Glenn Jocher, guigarfr, perry0418, Ttayu, Josh Veitch-Michaelis, Gabriel Bianconi and IlyaOvodov. (2019) ultralytics/yolov3: Rectangular Inference, Conv2d + Batchnorm2d Layer Fusion (Version v6). Zenodo. http://doi.org/10.5281/zenodo.2672652.