A Deep learning Based YOLOv3 Best selective search on CCTV videos

Nagendra Nambaru, BJM RAVI KUMAR M.tech (Assistant professor)

Department of Computer Science and Engineering , Raghu Institute Of Technology , Visakhapatnam , AP , India .

Abstract

In this paper, we are proposed an best YOLOv3-based neural network for De- identification technology. The existing YOLOv3 is a network with fast speed and performance recently. Most surveillance system using CCD cameras simultaneously store images from cameras installed in multiple locations. In such an environment, the use of deep learning requires a method of detecting objects through a single inference engine in a plurality of image. If the inference engine hardware is used for each camera channel, the cost of building a surveillance system increases significantly. Therefore, in the field of surveillance systems, a network structure with a high detection speed is required even if the detection performance is slightly degraded. This paper proposes a method to increase the detection speed by reducing the existing YOLOv3 network Architecture. 53 feature extractors, Darknet-53, are reduced to 24 layers. Therefore, a total of 106 layers is reduced to 39. And 53 YOLOv3 box detection parts are reduced to 15 layers. . In order to verify the efficiency of the proposed algorithm, the WIDER FACE dataset and its own collected dataset, we compared the performance with the existing YOLOv3-tiny and YOLOv3. As a result, the result was 87.48% mAP improved by 19.55% compared to the conventional YOLOv3-tiny. And I got a slow result of 100.5 FPS speed than the existing speed. And Object De-identification technology has been applied according to the results of the detection box. Therefore, it is faster than YOLOv3, and it is similar to YOLOv3 detection accuracy, proving that it is better than YOLOv3-tiny in real time detection.

1) INTRODUCTION

In Recently, object detection technology using deep learning has attracted attention. Deep learning technology is used in various applications, especially in the field of video. These deep- running techniques have been actively studied for more efficient detection, among which Darknet's YOLO [1] is a simple process and has high accuracy performance compared to other real-time detection systems. However, there are some drawbacks of low accuracy for small objects. In order to improve the performance of existing YOLO, several small changes were applied to improve the performance of YOLOv2 [2] and YOLO9000 [2] Improved low detection performance for small objects. However, in the field of surveillance systems, in order to apply object non- identification techniques to images input from multiple channels, performance is required to reduce costs through multiprocessing rather than high-speed and single processing rather than low-accuracy performance. therefore, a deep- running model with faster and more accurate balance

performance than YOLOv3-tiny is needed. The proposed model reduces the number of layers in the existing YOLOv3 network model to shorten the flops and feature extraction time. We will use the proposed YOLOv3-39 model to compare the performance of existing YOLOv3 and YOLOv3- tiny and apply de-identification technology to the detected boxes.

[1] IMPROVED YOLOV3-39 NETWORK MODEL

The existing YOLOv3 network model extracts features through Darknet-53. Darknet-53 is a residual network-type feature extractor that is composed of Convolutional layer and Shortcut layer, which can reduce computation while extracting high-performance features.

	Darknet-53				Proposed Feature Extractor			
	Type	Filters	Size	Output	Type	Filters	Size	Output
	Convolutional	32	3x3	416 x 416	Convolutional	16	3x3	416 x 416
	Convolutional	64	3x3/2	208 x 208	Convolutional	32	3x3/2	208 x 208
1x	Convolutional	32	1x1		Convolutional	16	1x1	
	Convolutional	64	3x3		Convolutional	32	3x3	
	Residual			208 x 208	Residual			208 x 208
	Convolutional	128	3x3/2	104 x 104	Convolutional	128	3x3/2	104 x 104
2x	Convolutional	64	1x1		Convolutional	64	1x1	
	Convolutional	128	3x3		Convolutional	128	3x3	
	Residual			140 x 104	Residual			140 x 104
	Convolutional	256	3x3/2	52 x 52	Convolutional	128	3x3/2	52 x 52
8x	Convolutional	128	1x1		Convolutional	256	1x1	
	Convolutional	256	3x3		Convolutional	128	3x3	
	Residual			52 x 52	Residual			52 x 52
	Convolutional	512	3x3/2	26 x 26	Convolutional	256	3x3/2	26 x 26
8x	Convolutional	256	1x1		Convolutional	128	1x1	
	Convolutional	512	3x3		Convolutional	256	3x3	
	Residual			26 x 26	Residual			26 x 26
	Convolutional	1024	3x3/2	13 x 13	Convolutional	512	3x3/2	13 x 13
4x	Convolutional	512	1x1		Convolutional	256	1x1	
	Convolutional	1024	3x3		Convolutional	512	3x3	
	Residual			13 x 13	Residual			13 x 13

Fig. 1. YOLOv3 network Architecture

High-performance feature extraction through Darknet-53 enables high-performance object detection. However, in the field of surveillance system, it shows performance degradation in terms of speed at which objects can be simultaneously detected from images input from multi- channels. In order to solve the problem in terms of speed, the feature extraction layer is modified as shown in Fig 1. Darknet-53 is a feature extractor less than 21 layer. YOLOv3 YOLOv3 predicts boxes of three different scales. YOLO v3 extracts features from various scales in a similar way to the concept of the feature pyramid network. You only need to add a few convolutional layers in the basic feature extractor. Finally, we finally get the 3-d tensor as output, which contains information about box, objectness, and class.

Fig. 2. The proposed modified YOLO detection layers

2.EXPERIMENTAL

RESULTS

In order to verify the performance of the proposed algorithm, we compared the accuracy and speed with the existing YOLOv3 and YOLOv3-tiny network models using WIDER FACE

[4] and our own collected datasets. We used image data with ground-truth values for 4 objects (face, person, license plate, ID). In addition, a total of 23,565 pieces of image data were trained 500,000 times, including 21,209 pieces of learning data and 2,356 pieces of verification data. As a result, as shown in Table 1, the accuracy performance comparison results were 88.99% for YOLOv3, 67.93% for YOLOv3-tiny, and 87.48% for YOLOv3-tiny. This result shows that the accuracy performance is improved by 19.55% compared to the existing YOLOv3-tiny. This result shows that the accuracy performance is improved by 19.55% compared to the existing YOLOv3-tiny. And compared with YOLOv3, the result was 1.51% lower. Also, the speed performance was 54.7 FPS lower than YOLOv3-tiny. On the other hand, we obtained

70.2 FPS faster than YOLOv3.

Table. 1 Deep learning model comparison result

Network Model	**Accuracy(mAP)**	**Speed(FPS)**
YOLOv3	88.99%	30.3
YOLOv3-tiny	67.93%	155.2
Proposed YOLOv3-tiny	87.48%	100.5

From the experimental results, similar restoration angles of about 85% or more were confirmed in one frame of shaken images. Fig. 3 is the proposed image of the object detection

result using YOLOV3-tiny.

Fig. 3. Object detection results using the proposed YOLOv3

In addition, the box of the object detected by using YOLOv3-39 applied De-identification technology as shown in Fig.

1. Training&Validation Data preparation

Download the Raccoon dataset from from

Organize the dataset into 4 folders:

- train_image_folder <= the folder that contains the train images.
- train_annot_folder <= the folder that contains the train annotations in VOC format.
- valid_image_folder <= the folder that contains the validation images.
- valid_annot_folder <= the folder that contains the validation annotations in VOC format.

There is a one-to-one correspondence by file name between images and annotations. If the validation set is empty, the training set will be automatically splitted into the training set and validation set using the ratio of 0.8.

Also, if you've got the dataset split into 2 folders such as one for images and the other one for annotations and you need to set a custom size for the validation set, use create_validation_set.sh script to that. The script expects the following parameters in the following order:

```
./create_validation_set.sh $param1 $param2 $param3 $param4
# 1st param - folder where the images are found
# 2nd param - folder where the annotations are found
# 3rd param - number of random choices (aka the size of the validation set in absolute value)
# 4th param - folder where validation images/annots end up (must have images/annots folders inside the given directory as the 4th param)
```

2. Edit the configuration file

The configuration file is a json file, which looks like this:

```
{
    "model" :
        { "min_input_size":    352,
        "max_input_size":      448,
        "anchors":             [10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198,  373,326],
        "labels":              ["raccoon"]
    },

    "train": {
        "train_image_folder": "/home/andy/data/raccoon_dataset/images/",
        "train_annot_folder": "/home/andy/data/raccoon_dataset/anns/",

        "train_times":         10,           # the number of time to cycle through the training set, useful for small datasets
        "pretrained_weights":  "",           # specify the path of the pretrained weights, but it's fine to start from
        scratch "batch_size":              16,# the number of images to read in each batch
        "learning_rate":       1e-4,         # the base learning rate of the default Adam rate scheduler
        "nb_epoch":            50,           # number of epoches
        "warmup_epochs":       3,            # the number of initial epochs during which the sizes of the 5 boxes in
each cell is forced to match the sizes of the 5 anchors, this trick seems to improve precision emperically
        "ignore_thresh": 0.5,
```

```
        "gpus":                 "0,1",

        "saved_weights_name": "raccoon.h5",
        "debug":                true            # turn on/off the line that prints current confidence, position, size, class losses
and recall
    },

    "valid":
        { "valid_image_folder":
        "", "valid_annot_folder":
        "",


        "valid_times":          1
    }
}
```

The labels setting lists the labels to be trained on. Only images, which has labels being listed, are fed to the network. The rest images are simply ignored. By this way, a Dog Detector can easily be trained using VOC or COCO dataset by setting labels to ['dog'].

Download pretrained weights for backend at:

This weights must be put in the root folder of the repository. They are the pretrained weights for the backend only and will be loaded during model creation. The code does not work without this weights.

3. Generate anchors for your dataset (optional)

python gen_anchors.py -c config.json

Copy the generated anchors printed on the terminal to the anchors setting in config.json.

4. Start the training process

python train.py -c config.json

By the end of this process, the code will write the weights of the best model to file best_weights.h5 (or whatever name specified in the setting "saved_weights_name" in the config.json file). The training process stops when the loss on the validation set is not improved in 3 consecutive epoches.

5. Perform detection using trained weights on image, set of images, video, or webcam

python predict.py -c config.json -i /path/to/image/or/video

It carries out detection on the image and write the image with detected bounding boxes to the same folder.
If you wish to change the object threshold or IOU threshold, you can do it by altering obj_thresh and nms_thresh variables. By default, they are set to 0.5 and 0.45 respectively.

Evaluation

python evaluate.py -c config.json

Compute the mAP performance of the model defined in saved_weights_name on the validation dataset defined in valid_image_folder and valid_annot_folder.

CONCLUSIONS

This paper proposed an improved YOLOv3-tiny for real- time object detection. The existing YOLOv3-tiny is a with higher speed than accuracy. However, it has a disadvantage of low accuracy in detection of small objects. The proposed YOLOv3-tiny consists of a Fully Convolutional Network(FCN) and two upsample processes. The enhanced YOLOv3-tiny consists of a Fully convolutional network and two Upsample processes. FCN can prevent information distortion by upsample process by maintaining location and spatial information that is reduced by pooling process. Through two upsampling procedures, we can infer three scales and improve the accuracy of small objects. In order to verify the performance of the proposed algorithm, we compared the accuracy and speed with the existing YOLOv3 and YOLOv3- tiny network models using WIDER FACE datasets and our own collected datasets. We used image data with ground-truth values for 4 objects (face, person, license plate, ID). In addition, a total of 18,951 pieces of image data were trained 500,000 times, including 17,056 pieces of learning data and 1,895 pieces of verification data. As a result, as shown in Table 1, the accuracy performance comparison results were 88.99% mAP for YOLOv3, 67.93% mAP for YOLOv3- tiny, and 87.48% mAP for YOLOv3-tiny. This result shows that the accuracy performance is improved by 19.55% mAP compared to the existing YOLOv3-tiny. And compared with YOLOv3, the result was 1.51% lower. Also, the speed performance was 54.7 FPS lower than YOLOv3-tiny. On the other hand, we obtained 70.2 FPS faster than YOLOv3. it is similar to YOLOv3 detection accuracy, proving that it is better than YOLOv3- tiny in real time detection.

REFERENCES

1.J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640, 2015. 5

2.J. Redmon and A. Farhadi, “YOLO9000: Better, Faster,Stronger,” arXiv preprint arXiv:1612.08242, 2016

3.J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” arXiv preprint arXiv:1804.02767, 2018

4.J. Redmon and A. Farhadi, “YOLO9000: Better, Faster,Stronger,” arXiv preprint arXiv:1612.08242, 2016 [4] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang, " WIDER FACE: A Face Detection Benchmark", arXiv preprint arXiv:1511.06523, 2015.

5.S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, June 2016.
Show in Context View Article Full Text: PDF (1177KB) Google Scholar

6.J. Dai, Y. Li, K. He and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks", *Proceeding of International Conference on Neural Information Processing Systems*, 2016.
Show in Context Google Scholar

7.T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature pyramid networks for object detection", *Proceeding of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.
Show in Context View Article Full Text: PDF (262KB) Google Scholar

8.T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollar, "Focal loss for dense object detection", *Proceeding of IEEE International Conference on Computer Vision*, October. 2017.
Show in Context View Article Full Text: PDF (262KB) Google Scholar

9.J. Redmon and A. Farhadi, "Yolov3: An incremental improvement", 2018. Show in Context Google Scholar

10.W. Liu, D. Anguelov, D. Erhan, C. Szegedy and S. Reed, "SSD: Single shot multibox detector", *Proceedding of European Conference on Computer Vision*, 2016.
Show in Context Google Scholar