Precise Segmentation on Poly-Yolo, YoloV5 and FCN

Xinyuan Cai[1(✉)], Yangchenchen Jin[2], Yunfei Liao[3], Jiawen Tian[4], and Yancong Deng[5]

[1] Purdue University, West Lafayette, USA
cai282@purdue.edu
[2] Sun Yat-sen University, Guangzhou, China
jinychch@mail2.sysu.edu.cn
[3] University of Electronic Science, and Technology of China, Chengdu, China
pwlykfp@std.uestc.edu.cn
[4] University of New South Wales, Sydney, Australia
[5] University of California, San Diego, San Diego, USA
yad002@eng.ucsd.edu

Abstract. Nowadays, computer vision is becoming more and more popular and is applied to lots of fields. It can be used to detect safe and unsafe behavior happening in construction area [6]. It can also used for auto-vehicle driving, object detection, and so on. Currently, there are several ways to realize this like FCN and YOLO. However, they all exist some limitation. For example, the bounding box of YOLO is always being castigated by the user. There are a lot of versions of YOLO with different solutions to this kind of issues aiming to be used in different circumstances. Poly-YOLO is one of them. It decently solved the bounding box issue by using polygons rather than rectangles. In the paper, we focused on several techniques to achieve object detection. The YOLO approach will be mainly talked about for application, since it can be easily used combined with other algorithms like semantic segmentation, FCN, and etc. to enhance its performance. What's more, the computation speed of Poly-YOLO is the main advantage why people choose YOLO.

1 Introduction

Object detection is one of the most popular topic, since it can be used in wild areas, like self-driving cars, detecting safe and unsafe behavior during the lab, police department, face detection. It is a computer technology involved in computer vision, image processing, machine learning, artificial intelligence, and so on. At the same it is also challenging, especially during the time when there still exists a lot of unsolved problems involving in detecting the subjects overlapping with each other.

Every object has its own features which is useful for us to do the classifications. For instance, all cellphones are rectangular, which is the feature for classification and, at the same time, for computer to learn. Today, we have several techniques that can solve most of the problems when doing the object detection. Basically, it can be divided into two techniques: non-neural approaches and neural network approaches. In most cases, neural network approaches can be more useful, since it can improve itself from

Z. Xiao et al. (Eds.): ICECI 2022, LNICST 478, pp. 54–68, 2023.
https://doi.org/10.1007/978-3-031-28990-3_5

the result of its accuracy. The most six popular tools are: Region Proposals, Single Shot Multi-Box Detector, YOLO, Single-shot Refinement Neural Network, Retina-Net, and Deformable convolutional networks. There is no doubt that YOLO is the most popular one, since it is relatively easy to use and has the smallest size. Therefore, YOLO will also be our main topic in the paper.

Under some circumstances, these existing tools are not enough to satisfy our requirement. For example, the original YOLO algorithm is not precise enough to indicate the bounding objects. Therefore, we can propose a new algorithm from the original version or combine the existing tool to compromise the shortage of each other. Therefore, in the following sections, we will go through the background of some important tools, and then introduce poly-YOLO which is based on existing tools but with huge improvements (Fig. 1).

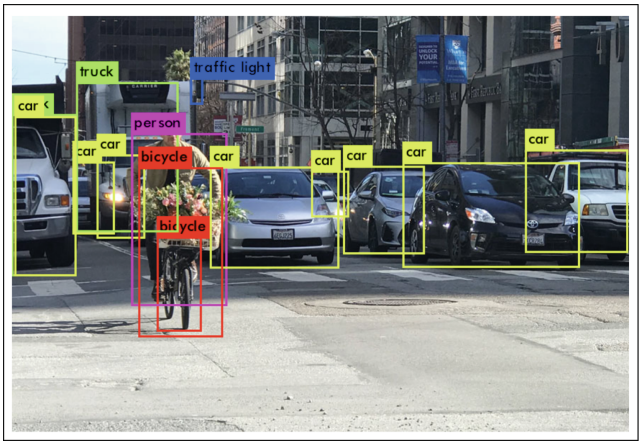

Fig. 1. How YOLO v3 looks like [5].

2 Related Work

2.1 Introduce of FCN

FCN stands for fully convolutional network. It is modified from the original convolutional network. Thus, before moving on to FCN, we focus on the convolutional network first.

The original convolutional network, which is also called ConvNet, is a deep learning algorithm. It can take in some images and then learn the feature among these images. These images need to be pre-processed, and assigned the weight of the feature for ConvNet to learn. The whole structure of convolutional network is analogous to human's brain; each node can be analogous to the neuron inside human's brain. Each neuron stores important information about the features of the images which will be used to detect the category of upcoming images. Although convolutional networks are powerful visual models that yield hierarchies of features, there still exists some limitation. One of the shortages is that it can also take fixed size of images, which is the reason why fully convolutional network shows up.

One step further, for the fully convolutional network, the basic idea is to do the normal convolutional network first and shrink the size of the image. After that, it will do the up sampling which I think is the key point in fully convolutional network. The special difference of this method is that it can take arbitrary size of image instead of fixed size of image [2].

By using the arbitrary size of image as input, we do the normal convolution first, using ReLU, shrink the pool, and repeat these steps. At the end, it uses up sampling to convert the image to specific sizes. Finally we do the classification on the images [2].

In sum, the key point for this FCN will be the upsampling, and with the different combination of up sampling, there are FCN32, FCN16, and FCN8 separately (Fig. 2).

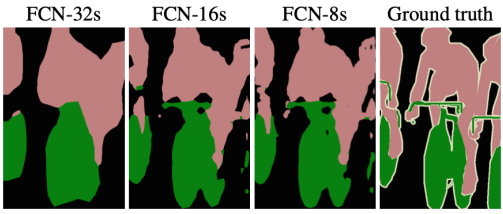

Fig. 2. Difference between FCN32, 16, 8 and Ground truth.

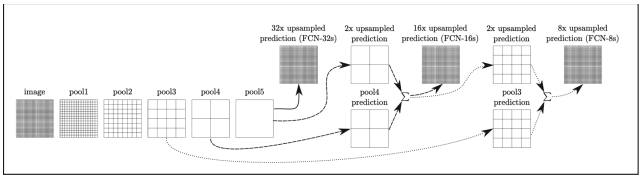

Fig. 3. The architecture of FCN [2]

2.2 Evolution of YOLO

YOLO stands for "You only look once", which is a viral and widely used algorithm. It is famous for its object detection characteristic. The main advantage of YOLO is model's small size and fast calculation speed. The structure of YOLO is straightforward. It can directly output the position and category of the bounding box through the neural network. The speed of YOLO is fast because YOLO only need to put the picture into the network to get the final detection result. YOLO's test results are poor for objects that are very close to each other and in groups. This poor performance is because only two boxes in the grid are predicted and only belong to a new class of objects of the same category.

Except the YOLO v1-5, it also has a few revised-limited versions such as YOLO-LITE and Poly-YOLO, which will be used in the later task. There are five main YOLO versions.

The original YOLO architecture consists of 24 convolution layers, followed by two fully connected layers. However, it has two defects: one is inaccurate positioning, and the other is the lower recall rate.

YOLO V2 improves these two defects from the V1, so it makes YOLO better and faster. The better is in four aspects: batch normalization, high-resolution classier, fine features, and multi-scale training. The faster is in three aspects: Darknet-19, training for clarification, and training for detection [1].

The difference between YOLO V2 and V3 is that YOLO V3 uses multi-scale features for object detection and adjusts the basic network structure. YOLO V3 also adopts feature graphs of three scales. YOLO V3 uses three prior boxes for each position, so K-means is used to get nine prior boxes and divide them into three scale feature maps. Feature maps with larger-scale use smaller prior boxes. What's more, YOLO V3 features extraction network used the residual model. Compared with Darknet- 19 used by YOLO V2, it contained 53 convolution layers, so it was called Darknet-53 [1].

The style of YOLO V4 has huge difference compared to the previous versions, more focus on comparing data, and has a substantial improvement. The integrator characterizes it and finally achieves very high performance. We can summarize it like this: YOLO V4 = CSP Darknet53+SPP+Pan+YOLO V3 [1].

Due to the multiple network architectures used by YOLO V5, it becomes more flexible to be used. At the same time, the model size of YOLO V5 is fairly small without losing the accuracy compared to the YOLO V4. However, people still keep reserved opinion towards YOLO V5 because it is less innovative than YOLO V4, but it has some performance improvements [1].

2.3 Limitations and Improvements

The main target of object detection is to distinguish the objects from the various background. However, sometimes the background contains other distracted objects which are not the objects we are intended to detect but look similar. Here comes the disadvantage of original YOLO algorithm. The original YOLO algorithm uses bounding boxes to indicate the detected objects. Nevertheless, some objects have extremely complex shapes. As a result, the bounding boxes cannot wrap the object tightly so that the distracted items at background will take up most of the areas inside the bounding boxes. To some extents, it reduced the performance of a classifier which will be applied over the bounding box. Shortly, it cannot satisfy the requirement when it comes to precise object detection. Here is the reason why we need poly-YOLO.

As mentioned before, besides these five main versions of YOLO, there also exists modified version of YOLO. One of the most useful one is POLY-YOLO. Poly-YOLO extends from the original architecture of YOLO V3, and, at the same time, it adds the feature of segmentation and improves the performance.

Since using a rectangle to show the boundary of objects is not precise enough, POLY-YOLO appeals to instance segmentation. Instead of rectangle, polygons are used to bound tightly enough around the detected objects. POLY-YOLO can learn itself in size-independent shapes according to the bounding box. Also, to bound tightly, POLY-YOLO can create a dynamic number of vertices per polygon. The number of vertices depends on how complex the shape is. Intuitively, if the number of vertices are large enough, it will be similar as curves by the human-eyes.

What's more, it removes two of its weaknesses: a lot of repeating labels and inefficient distribution of anchors. To solve the label rewriting issue, they adjust the value

of scale multiplicator so that the output and input resolutions are equal. To address the inefficient distribution of anchors, POLY-YOLO will create an architecture with a single output to aggregate information from various scales. Such improvements make POLY-YOLO easily to be integrated into various scenarios [3].

3 Comparative Methodology

3.1 Mathematics of FCN

Fully Convolutional Network is used to generate multi-dimension array rather than one-dimension array which is used by CNN. The basic architecture is shown in the Fig. 3. The FCN structure is divided into 5 blocks and 2 convolutions. We can divide 5 blocks into two groups: block 1–2 and block 3–5. For block 1–2, the input data will be convoluted twice, followed by a pooling process. The number of process of convolution is different between two groups. For block 3–5, they have 3 convolutional layers, followed by the same pooling process. Since FCN is based on VGG16, the basic information about each block is the same as the block in VGG16. The basic procedure of VGG16 can be summarized by the Fig. 4.

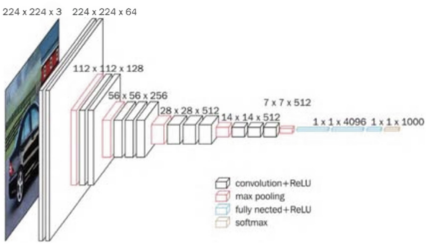

Fig. 4. Structure of VGG16 [4]

From the Fig. 4, it is clear that there are five pools in the model. Therefore, the size of final block five will be 2^5 times smaller than the size of input image. That's the reason why up-sampling is critical in FCN so that the different layers can match with each other. Then they will be added together. In short, up-sampling is to make different layers have same shapes to add with each other. The key point is that up-sampling won't change the dimension, instead it will only change the size - height and width. The following is the steps of up-sampling: [4] (Fig. 5)

- unpooling
- interpolation
- deconvolution
- dilated convolution

There also exists FCN version which is based on ResNet and Vanilla instead of VGG16.

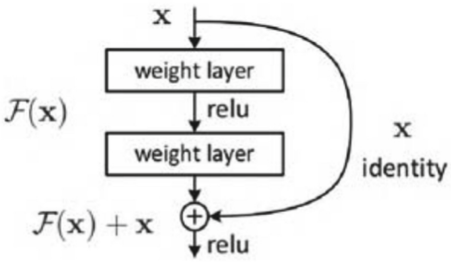

Fig. 5. Principle of residual learning [4]

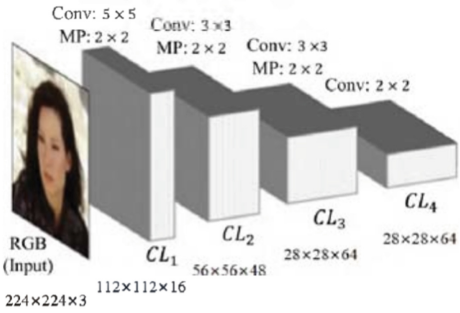

Fig. 6. Structure of Vanilla CNN [4]

For ResNet, instead of 5 blocks with fully connected layer, it uses principle of residual learning to combine different information together, which finally increase the accuracy.

For Vanilla, it replaces 5 blocks in VGG16 with structure in Fig. 6. The Vanilla algorithm won't fuse any layers together. It is just the structure of Fig. 4 and followed by conv6 and conv7.

3.2 Mathematics of YOLO

As mentioned on the related work, there are 5 versions of YOLO: v1 to v5. V1 to V3 were created by Joseph Redmon and Ali Farhadi. Since poly-YOLO is developed upon YOLOv3, we focus on the version 3 of YOLO in this paper which is more accurate compared to the original YOLO.

The general idea of YOLO v3 [7] algorithm is that the input image will be processed by a neural network with a result of bounding boxes and the prediction labels. In detail, one frame will be extracted from a video. Then this frame will be resized to 416×416. The resized frame will be given to the neural network which is similar to CNN, to do the prediction.

There are different neural network can be used by YOLO such as Darknet-19, ResNet-101, ResNet-152, and Darknet-53. The differences between these neural net-

Backbone	Top-1	Top-5	Ops	BFLOP/s	FPS
Darknet-19	74.1	91.8	7.29	1246	**171**
ResNet-101	77.1	93.7	19.7	1039	53
ResNet-152	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Fig. 7. Comparison between different kernels

work are the accuracy and the efficiency. For example, the YOLOv2 uses Darknet-19 and YOLO v3 uses Darknet-53 instead. It is true that Darknet-19 can process more frames per second than Darknet-53, but Darknet-53 has 53 convolutional layers instead of 19, which means that Darknet-53 are more powerful (shows in Fig. 7). The basic architecture of Darknet-53 consists of the following elements [5]: convolutional layers, residual layers, upsampling layers, and skip connections. After processed by Darknet-53, the output vector consisting of the following parameters is return: [5]

- The input image is divided into grid cells. There is a one cell corresponding to an object on the image to predict which object it is.
- The grid will predict the bounding boxes with classification information (Figs. 8 and 9).

There are five components in the bounding boxes (x, y, a, b, c). x, y is the coordinates of the center of the boxes. a, b is the width and the height of the box. c is the confidence value.

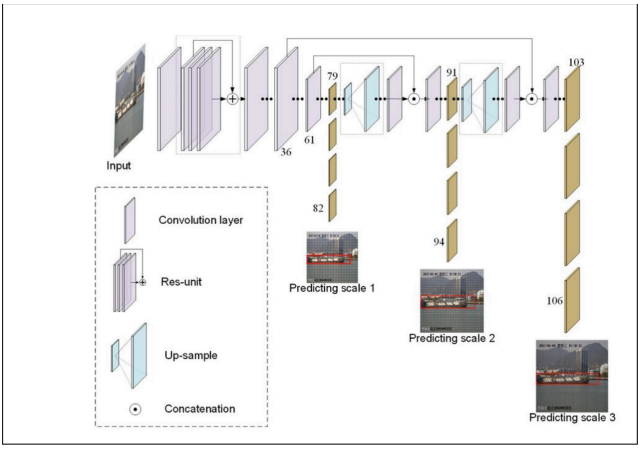

Fig. 8. The architecture of Darknet-53 [5]

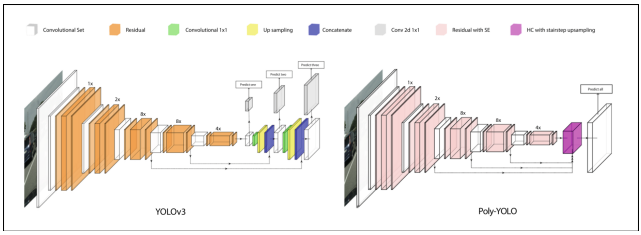

Fig. 9. Comparison between YOLOv3 and Poly-YOLO [3].

It also uses statistical method called "Intersection over Union" to determine whether the object detection algorithm is working or not. The Intersection over Union can be simplified as IOU:

$$IOU = \frac{Area\ of\ Intersection}{Area\ of\ Union}$$

If the result is greater or equal to 0.5, we can label it as "correct".

What's more, to avoid bounding boxes multiple times for each object, it also uses a technique called Non-max Suppression (NMS). The NMS is to guarantee that this algorithm will only detect each subject only once. The basic algorithm about how NMS works is that it first searches for a particular object with largest probabilities, which means it detects the object with the most confidence. Then NMS will also search for the remaining bounding boxes. With the help of IOU mentioned above, the bounding boxes can be suppressed into only one bounding box for each object.

3.3 Mathematics of Poly-YOLO

Poly-YOLO is based on YOLOv3. It solves the label rewriting problem in YOLOv3. There is a scale multiplicator to express the ratio of the output resolution to input resolution r. They donate this ratio as s_k [3]. It is obvious that the ideal goal is to make $r = rs_k$ and we can get that $s_k = 1$. This result means that the output resolutions will be equal to the input resolutions. Under this circumstance, there will be no label rewriting [3]. The U-Net is also using such condition [8]. To provide more accuracy, the Poly-YOLO architecture is using "squeeze-and-excitation (SE) blocks" in the backbone [9], since Darknet-53 has repetitive blocks and each block consist of coupled convolutions with residual connection [3]. Instead, the squeeze-and-excitation blocks uses spatial and channel-wise information to improve the accuracy [3]. However, the limitation of this is that it will slow down the computation speed. The reason why the series of YOLO is popular is that it has high computation speed. Therefore, in the feature extraction phase, Poly-YOLO reduced the number of convolutional filters. It reduces to $3/4$ of the original convolutional filters [3]. At the same time, the precision of Poly-YOLO is still higher than YOLOv3. For the processing speed consideration, they also propose Poly-YOLO lite whose number of parameters is only 16.5M, comparing with YOLOv3 which is 61.5M (Figs. 10 and 11).

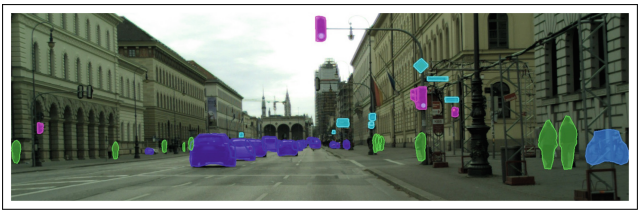

Fig. 10. How Poly-YOLO looks like [3].

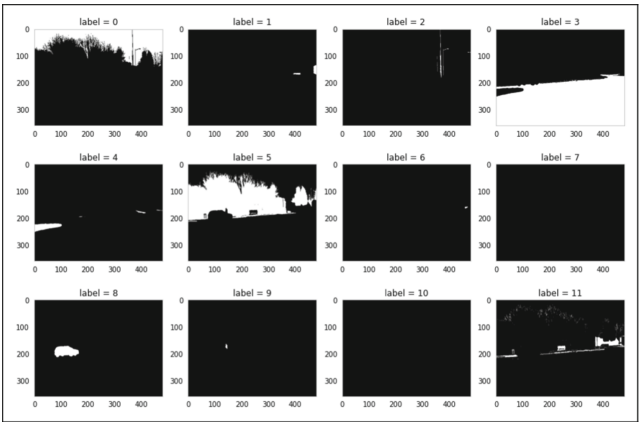

Fig. 11. FCN8 result without color assigned.

There is a sentence at the end of the paper of YOLOv3 saying that "Boxes are stupid anyway though, I'm probably a true believer in masks except I can't get YOLO to learn them" [7]. Thus, Poly-YOLO fixes this issue with bounding polygons instead of rectangle. In YOLOv3, they use Cartesian coordinate system to determine the bounding boxes. In poly-YOLO, they switch to polar coordinate system so that the network can learn more general and size-independent shapes instead of some particular instances with sizes [3]. In more understandable way, for instance, the same objects like cars will have different sizes corresponding to the distance from the cameras, smaller one and bigger one. The polar coordinate system model to train the images by using the angles, relative distance from the center of bounding box, confidence. Clearly, these values will be the same for both objects in the images, no matter the distance from the camera. After getting these values, the distances will be multiplied with these values to get particular values of the cars with different values. Since the values before being multiplied with distance can be shared with the same objects, the learning procedure can be simplified and speed up.

4 Experiments

4.1 FCN8

In an image for the semantic segmentation, each pixel is labeled with the class of its enclosing object. The semantic segmentation problem requires to make a classification at every pixel, which means that the dataset is needed to train the data.

FCN8 first trained the data and did the segmentation with black and white result. We then assigned color to it. In this experiment, we use sns color palette to assign color to the pattern. To simplify the problem, we reshaped all the images to the same size: (224, 224). Since VGG and FCN model both use this image shape. This blog uses a network which takes advantage of VGG structure. As a result, the FCN module becomes much easier to explain when the shape of the image is (224, 224) (Fig. 12).

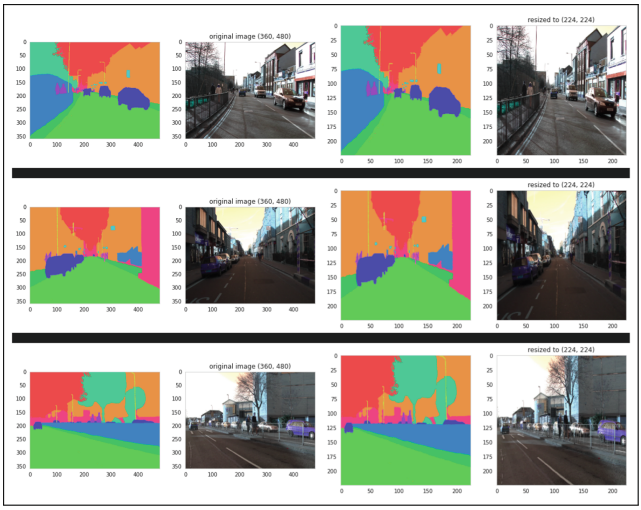

Fig. 12. FCN8 result with sns color palette assigned

VGG and some other successful deep learning models is original designed for performing classification. After combining its convolution layers with max-pooling and then stacking fully connected layers, it becomes able to learn something using global information where the spatial arrangement of the input falls away. For the segmentation task, however, spatial information should be stored to make a pixel-wise classification. FCN allows this by making all the layers of VGG to convolutional layers. Fully convolutional indicates that the neural network is composed of convolutional layers without any fully-connected layers usually found at the end of the network. Fully Convolutional Networks for Semantic Segmentation motivates the use of fully convolutional networks by "convolutionalizing" popular CNN architectures e.g. VGG can also be viewed as FCN. The following method is FCN8 from Fully Convolutional Networks for Semantic Segmentation. It duplicates VGG16 net by discarding the final classifier layer and

convert all fully connected layers to convolutions. Output image size is (output-height, output-width) = (224, 224).

The upsampling layer brings low resolution image to high resolution. There are various upsampling methods. This presentation gives a good overview. For example, one may double the image resolution by duplicating each pixel twice. This is so-called nearest neighbor approach and implemented in Keras's UpSampling2D. These upsampling layers do not have weights/parameters so the model is not flexible. Instead, FCN8 uses upsampling procedure called backwards convolution (sometimes called deconvolution) with output stride. This method simply reverses the forward and backward passes of convolution and implemented in Keras's Conv2DTranspose. In FCN8, the upsampling layer is followed by several skip connections. See details at Fully Convolutional Networks for Semantic Segmentation [2]. The following is the general procedure:

- input-height and width must be divisible by 32 because max-pooling with filter size = (2, 2) is operated 5 times
- Block 1 - 1st Convolutional layer with pool
- Block 2 - 2nd Convolutional layer with pool
- Block 3 - 3rd Convolutional layer with pool (save a copy for fusing upsampling)
- Block 4 - 4th Convolutional layer with pool (save a copy for fusing upsampling)
- Block 5 - 5th Convolutional layer with pool (save a copy for fusing upsampling)
- Set up VGG for the encoder parts of FCN8
- Conv6–7: (FCN-32s) and (2x up-sampled prediction based on block 5)
- 4 times upsampling for conv7: (FCN-32s) and (2x up-sampled prediction based on block 5)
- 2 times upsampling for pool3 and 4
- combine the upsampling and softmax
- create model and load weight.

4.2 YOLO

I choose YOLO v5 for this experiment since it is extremely to get and use. I use it through the torch module included in python. It is an amazingly easy module to use. The result image will have several box with the recognized name at the top left corner. The probability will also show up at the top right corner.

PyTorch supports both pre-trained dataset and customized dataset. It is simple to use pre-trained dataset. All you need to do is to set the parameter: pretrain = true. However, there are more steps if you would like to use customized dataset. The following is the procedure to use the customized dataset:

- create annotations
- create the dataset file
- train the dataset
- use the trained model to make predictions.

4.3 Poly-YOLO

I used pre-trained module inside the Poly-YOLO repository. There are two types of pertained module: poly-yolo and poly-yolo-lite. I used poly-yolo for this experiment. The result of the experiment for Poly-yolo is shown in Figureh. Since I used pertained model, the procedure is fairly simple. All I did is to load the model, setting the parameter $iou = 0.8$ and $score = 0.5$. The final thing is to set the color class. This is necessary because the color will be filled inside the bounding polygon, which can make the detected object stand out. From the Fig. 16, you can see that Poly-YOLO indeed fits the issue of bounding boxes, enabling the objects to be distinguished more easily from each other. However, not every object can be detected in this Fig. 16, which means that the accuracy of poly-YOLO are lower comparing to the FCN8 and YOLO.

5 Conclusion

From the experiment, we can see the advantage and disadvantage of FCN8, YOLOv5, and Poly-YOLO (Figs. 13, 14 and 15).

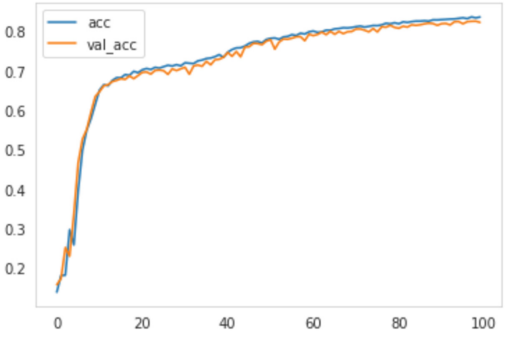

Fig. 13. Plot accuracy and value accuracy vs epoch (epochs = 100)

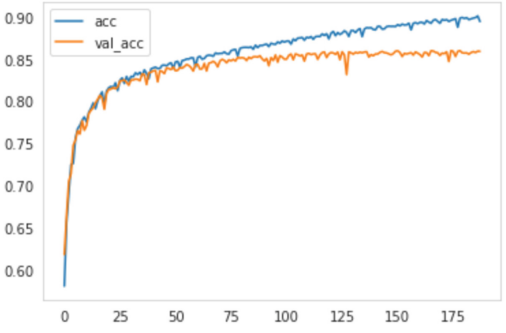

Fig. 14. Plot accuracy and value accuracy vs epoch (epochs = 188)

Fig. 15. The input image and output result from YOLOv5

FCN did a great work in doing the semantic segmentation. It is efficient asymptotically and absolutely. It eliminates a lot of complications needs in other work [2]. The best feature is that it doesn't specify the size of images. It also reduces the quantity of parameters for network because of the multiple convolution and pooling layers. As a result, it can simplify and improve the efficiency of learning. However, the shortcoming is also obvious. It will change the background into the same color, since FCN8 performs the semantic segmentation first, and then uses different color to present each part. Sometimes, it will be hard to recognize the detected object. This method will lose a bunch of information comparing to the original image.

Fig. 16. Experiment result by using Poly-YOLO.

The YOLO method avoids this shortcoming by using a really smart way. It create a bounding box around the detected object. It will show the detected label on the upper-left corner and the probability on the upper-right corner. Therefore, basically it didn't lose any information and that's also the reason why YOLO is extremely appropriate for the real-time detection, since it keeps all the things on the image. Its computation speed is also fast enough for the real-time detection. Therefore, this technique has already been use in a lot of areas, e.g.: using YOLO to detect unsafe behavior happening in the construction area [6]. However, the disadvantage for YOLO is also clear to see. Like the last sentence on the YOLOv3 paper: "Boxes are stupid anyway though, I'm probably a true believer in masks except I can't get YOLO to learn them" [7]. The box will cause a lot of confusions especially when two objects are really close to each other. Sometimes, it will be hard to distinguish which the object the box is bounding to. Poly-YOLO solves the issue of bounding box by using polygon instead of rectangle. By using the mathematical limit thinking, it is understandable that the polygon can be recognized as curved if there are infinite sides. That's how it works. By using multiple sides of polygon, the bounding polygon can be as close as enough to the detected object, which means that it won't bound unnecessary objects to increase the chance of confusion. What's more, in this way, the color can be filled into the bounding polygon to make the detected object stand out and easy to distinguish. Since it won't bound other objects into the bounding polygon, filling in color won't make the image lose any information comparing to the original image. However, due to the excellent the performance of Poly-YOLO, the computation is indeed slower that the other two. But, it is not slow too much. It removes few convolutional layers to increase the efficiency. What's more, the accuracy of Poly-YOLO is also lower than the other two (Fig. 17).

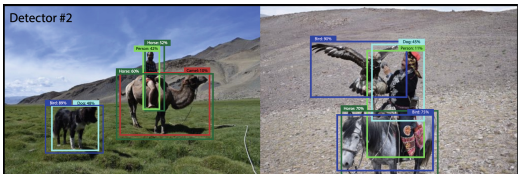

Fig. 17. Difficult to distinguish when objects are too close to each other [7]

In conclusion, each one has its own advantage. From my perspective, I prefer to give Poly-YOLO highest credits among those three. In practical usage, we can choose different tool according to various situations.

References

1. Jiang, P., Ergu, D., Liu, F., Cai, Y., Ma, B.: A review of yolo algorithm developments. Proc. Comput. Sci. **199**, 1066–1073 (2022). https://doi.org/10.1016/j.procs.2022.01.135. ISSN 1877-0509
2. Shelhamer, E., Long, J., Darrell, T.: Fully convolutional networks for semantic segmentation (2016)
3. Hurtík, P., Molek, V., Hula, J., Vajgl, M., Vlasánek, P., Nejezchleba, T.: Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3. CoRR, abs/2005.13243 (2020)
4. Hu, T., Deng, Y., Deng, Y., Ge, A.: Fully convolutional network variations and method on small dataset. In: 2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE), pp. 40–46 (2021). https://doi.org/10.1109/ICCECE51280.2021.9342059
5. Kumar, S., Yadav, D., Gupta, H., Verma, O.P., Ansari, I.A., Ahn, C.W.: A novel YOLOv3 algorithm-based deep learning approach for waste segregation: towards smart waste management. Electronics **10**(1), 14 (2021). https://doi.org/10.3390/electronics10010014
6. Yu, Y., Guo, H., Ding, Q., Li, H., Skitmore, M.: An experimental study of real-time identification of construction workers' unsafe behaviors. Autom. Constr. **82**, 193–206 (2017). https://doi.org/10.1016/j.autcon.2017.05.002. ISSN 0926-5805
7. Redmon, J., Farhadi, A.: YOLOv3: an incremental improvement. CoRR, abs/1804.02767 (2018)
8. Ronneberger, O., Fischer, P., Brox, T.: U-net: convolutional networks for biomedical image segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) MICCAI 2015. LNCS, vol. 9351, pp. 234–241. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24574-4_28
9. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7132–7141 (2018)