SIT 789

Task 5.2: Minor research project

1. Introduction

Artificial Intelligence has been changing the way of software development across different verticals. Better deep learning models and algorithms are coming up owing to better hardware capabilities and increase in organically generated data. In [1] we can also see this impact on computer vision, where impact of deep learning in the domain of computer vision has been discussed at length. Object detection is a specific subproblem within the domain of computer vision where the objective is to find the exact location as bounding box coordinates on a given image frame. Over last few decades, several algorithms have been proposed for performing object detection. Earlier, classical machine learning methods based on hand crafted features like Haar Cascade as proposed by Viola-Jones in [2] and Histogram of Oriented Gradients (HOG) [3] were used across tasks related to object detection. The impact of deep learning can be seen in object detection scenarios as well, recent deep learning architectures like Faster RCNN [4], SSD [5] and YOLO [6] constitute some of the state-of-the-art object detection deep learning networks. This report focusses on summarising the working of YOLO9000 [7] and finally demonstrates the working using Road Sign Detection dataset [8].

2. The road to YOLO9000

At the time when YOLO [6] was introduced, detection systems used classifiers which used techniques like sliding window or region proposal to classify various regions of image at different scales. YOLO proposed a novel approach of reframing object detection problem as a regression problem, the algorithm uses a unified model to predict object coordinates and class probabilities using raw image pixels. The algorithm outputs multiple bounding boxes and probabilities representing associated class labels and then uses non-maximum suppression to choose dominating object bounding location and label (Shown in figure 1). An added advantage of this approach is that because the network looks at the complete image frame during the training stage, it also learns the contextual information associated with the object of interest through object's neighbourhood.

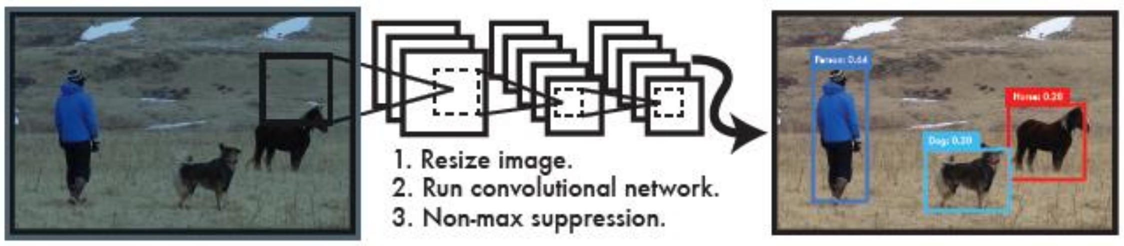


Fig 1: Image showing the working of YOLO [6].

While YOLO had its novelties, it suffered from multiple shortcomings, a significant error when compared to region proposal methods was the low recall of YOLO. YOLO9000 [7] or also known as YOLOv2 was introduced to address these shortcomings and take the YOLO architecture to at par or surpass the state-of-the-art approaches.

3. Why YOLO works?

This section discusses the additions and novel concepts introduced by YOLO9000, which make it the state-of-the-art object detection framework of the time.

YOLO9000 introduced **Batch Normalisation**, this helps the model to improve convergence and tackle regularisation issues. [7] reports 2% increase in mAP (mean Average Precision) after adding Batch Normalisation on all the convolution layers of the original YOLO architecture. Inspired from the use of priors in Faster R-CNN, YOLO9000 introduces the concept of **Anchor Boxes** to the YOLO architecture. To perform this modification, fully connected layer of the original YOLO architecture is removed, and the network learns to predicts offsets with respect to hard coded anchor boxes, this simplifies the learning process. Another key modification while introducing anchor boxes was the reducing the image resolution from 448 x 448 to 416 x416, this forces the network to output odd number of bounding box coordinates for the feature map and the network has only one centre cell. This also means that the output feature map is now of the dimension 13 x 13 as YOLO9000 has convolution layers which down sample the input image by a factor of 32.

With these modifications, another change to the YOLO architecture was needed, the original architecture predicted class probabilities based on actual locations of the object in the image, instead of this, the architecture now predicts class probabilities and *objectness score* for every anchor box. The concept of objectness score is preserved in the modified architecture as well, where objectness score denotes the IOU of the proposed bounding box and the ground truth bounding box and given there is an object, class probabilities denote the conditional probability that of the class labels associated with the object. These changes contribute to a significant increase in recall, taking the initial recall figure of 81% to 88%. While there is an increase in recall, the accuracy of the architecture takes a slight hit going down to 69.2 mAP from 69.5 mAP.

While the introduction of anchor boxes increased recall, it had some shortcomings, the first major shortcoming was the hand picked or hard coded anchors used, this makes it difficult for the network to predict better detections for different objects. To tackle this issue, authors use KMeans to select the anchor boxes (with K value of 5). This strategy results in much higher IOU as opposed to using hard coded anchor boxes.

Because the anchor boxes are predicted as offsets, they might be shifted to an inaccurate location in the image. This unconstrained nature of the anchor boxes prohibits the network to become stable, this is tackled by introducing logistic activation. This is like the approach used in YOLOv1 and the predicted bounding box coordinates are relative to the grid cell location. This squashes the input between 0 and 1 and makes the learning process easier as the offsets are localised the given grid cells and introduces additional stability on the network. Another key experiment was the introduction of **passthrough layer**, this allows the features from previous layers to be used deeper in the network. This brings higher resolution features and stacks them with lower resolution features, this provides the detector access to an elaborated feature map induced with fine grained features to perform detection.

Another powerful addition to the training process of the improved YOLO9000 architecture is the introduction of **multi-scale training**, during the training process the network chooses an image resolution randomly after every 10 batches. This is possible because the network only consists of convolution and pooling layer, this makes the network elastic enough to be resized during the training process. This also makes the network more robust to handle various input sizes, this makes YOLO9000 ideal for real time applications where such high frame rate is preferred. The range for these resizes values varies from 320 x 320 to 608 x 608 (multiples of 32, this makes the down sampling convenient). The result of this multi scale training is that the network achieves very **high mAP of 78.6** for high resolution of **544 x 544 at an FPS of 40** whereas it performs reasonably well at low resolutions of **288 x 288** where it achieves **a decent mAP of 69.0 with very high 91 FPS**. This provides the end user a flexibility to use the network depending on the use case (high latency vs high accuracy).

4. Possible improvements on YOLO9000

The previous section describes how and why the YOLO9000 detector came to be known as one of the state-of-the-art detectors. While the modifications to YOLOv1 architecture made several improvements to design and performance, there is still a considerable room for future improvements to the architecture of YOLO9000. This section explores the possible improvements which can make this architecture even more efficient. Section 4.1 highlight some of the work that has already been done to improve YOLO9000 and section 4.2 highlights some proposals which can lay foundations to increase the performance even more.

4.1. The YOLO family

Over the years several implementations have contributed to adding features and improvements to the YOLO9000 architecture. Some of these are from the original authors (Yolov3) while others (YOLOv4, v5, v6, v7) are from different contributors. This is just a brief overview of the other architectures in this family.

YOLOv3 [9]: YOLOv3 aims to improve the accuracy of YOLOv2 and increases the complexity of the features extracted by the convolution layer. To achieve this, the convolution layers are increased from 19 to 53. The framework for training and testing YOLO is developed in C++ is called darknet, for YOLOv3 this framework is called as Darknet-53 and for YOLOv2 Darknet-19, the suffixed number denotes the number of convolution layers in each YOLO architecture. Because YOLOv3 is deeper, it can extract higher order features and thus there is an increase in the accuracy, this is coupled with shortcut connections which enable fine grained features.

YOLOv4 [10]: YOLOv4 builds on top of YOLOv3 and introduces universal features which improve a CNN networks performance, YOLOv4 uses **Weighted-Residual-Connections, Self-adversarial-training, Mish-activation, Cross mini-Batch Normalization, Cross-Stage-Partial-connections** to improve the performance. This also uses data augmentation techniques like CutMix, MixUp, Blurring, Mossaic to improve the robustness of the model.

YOLOv5 [11]: YOLOv5 is not a published work rather a PyTorch implementation of existing concepts with minor changes. This work is more focussed on improving integration of the YOLO architecture and is compatible with popular cloud platforms like AWS, GCP and containers like Docker. The focus of this work is to provide wide support for usage of YOLO architecture apart from Darknet framework.

Note: Due to the easy integration, PyTorch compatibility and most resemblance with YOLO9000, this repository is used to demonstrate the working of YOLO on Road Sign Detection Dataset in this report.

YOLOv6 [12]: Building on top of YOLOv4, YOLOv6 improves and modifies the architecture to improve both accuracy and speed using novel quantization method which is responsible for faster FLOPs.

YOLOv7 [13]: YOLOv7 introduces trainable bag of freebies methods, re-parameterization of original model, *compound scaling* and *extend* methods and reducing parameters to increase inference speed and accuracy.

4.2 Other improvements

While the above works suggest that there is ongoing research to increase the efficiency of YOLO architectures, it is important to evaluate these architectures with the AI bias viewpoint so that we can move forward towards a *fair* object detector (YOLO architectures are significantly used in pedestrian detection scenarios, hence it is important to evaluate them on terms of *AI bias*). This will help uncover any prevalent biases and introduce training methods which can help mitigate these.

Another important area of improvement for these architectures is focus on inference using lightweight hardware. As we move towards democratization of AI more and more mobile applications and other lightweight devices (FPGA based devices) will rely on AI models. YOLO has lightweight variants such as tiny-YOLO, but even these modifications are too heavy to run in real time on low power devices, YOLO can be stripped down to a lighter network architecture like [14].

5. Experiment on Road Sign Detection Dataset

This section demonstrates how YOLO can be used to set up a training pipeline and inference pipeline using a custom dataset.

5.1 Dataset Details: The dataset [8] contains 877 samples which are divided as shown in table 1.

The dataset has these class labels:

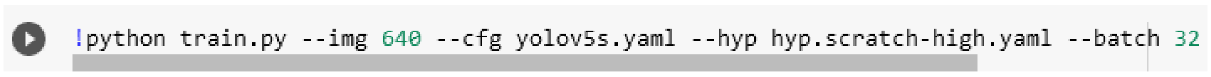
- traffic light
- stop
- speed limit
- crosswalk

Train	701
Validation	88
Test	88

Table 1: Dataset split

5.2 Training: To train the YOLO model we use the train.py script from the official YOLOv5 repository and save the trained weights to the specified directory. It is important to note that because this is just a demonstration of the training process, we have trained the model till 100 epochs, whereas in real world scenarios the training process can go up in order of thousands of epochs.

This also includes changes to the training hyperparameter config file, where the number of classes parameter must be changed to 4 to reflect updated number of classes.

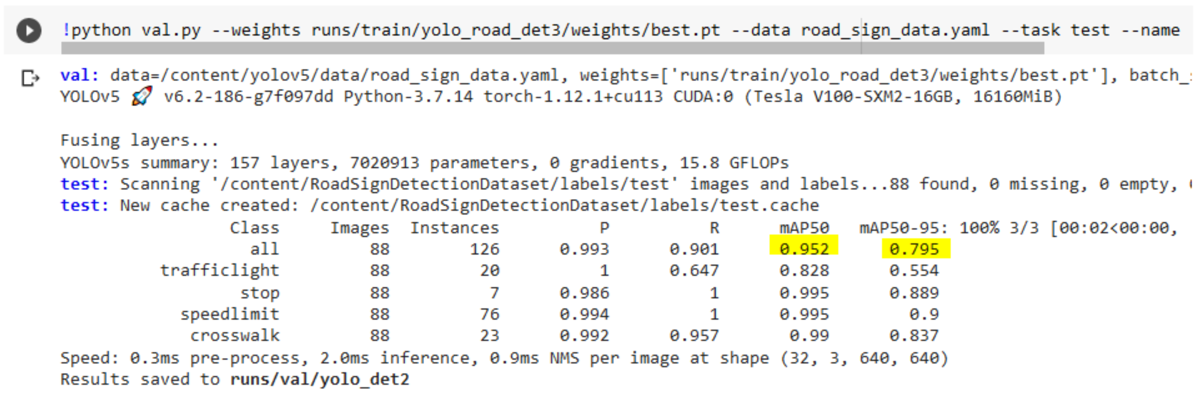


```
!python train.py --img 640 --cfg yolov5s.yaml --hyp hyp.scratch-high.yaml --batch 32
```

Fig 2: Training command for YOLO

5.3 Testing: There are several ways to test the trained model, the attached notebook shows using the test script to test on a given test set as well as using the API to test which can be used in real world applications and integrated into inference pipelines.

5.4 Results: Although the network is trained on a small dataset and only for 100 epochs, we can see that the network achieves high mAP of 95.2%, 99.3% Precision and 90.1% Recall. It is important to note that the training uses pretrained weights of YOLOv5 which contributes to such high evaluation metrics. This also presents the case of how easily YOLO can be used for real world dataset and thus is fit for real world applications with minimal finetuning.



```
!python val.py --weights runs/train/yolo_road_det3/weights/best.pt --data road_sign_data.yaml --task test --name
```

```
val: data=/content/yolov5/data/road_sign_data.yaml, weights=['runs/train/yolo_road_det3/weights/best.pt'], batch_
YOLOv5 🚀 v6.2-186-g7f097dd Python-3.7.14 torch-1.12.1+cu113 CUDA:0 (Tesla V100-SXM2-16GB, 16160MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7020913 parameters, 0 gradients, 15.8 GFLOPs
test: Scanning '/content/RoadSignDetectionDataset/labels/test' images and labels...88 found, 0 missing, 0 empty,
test: New cache created: /content/RoadSignDetectionDataset/labels/test.cache
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:02<00:00,
                   all         88        126      0.993      0.901      0.952      0.795
          trafficlight         88         20          1      0.647      0.828      0.554
                  stop         88          7      0.986          1      0.995      0.889
            speedlimit         88         76      0.994          1      0.995        0.9
             crosswalk         88         23      0.992      0.957       0.99      0.837
Speed: 0.3ms pre-process, 2.0ms inference, 0.9ms NMS per image at shape (32, 3, 640, 640)
Results saved to runs/val/yolo_det2
```

Fig 3: Result showing the overall mAP, Precision and Recall as well as class wise evaluation metrics.

Note: For entire training and testing process please refer to the attached notebook in Appendix – 1.

6. Conclusion

This report summarises the YOLO architecture and highlights the evolution of algorithm over different versions. This report also indicates future areas of improvement in YOLO9000 and other YOLO architectures. Finally, this report showcases the algorithm in practice using a real-world dataset, a similar model can be used in self driving car use-case to identify Road Signs.

7. References

[1] P. Patel and A. Thakkar, “The upsurge of deep learning for computer vision applications,” *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 10, no. 1, p. 538, 2020.

[2] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2005, vol. 1, p. I–I.

[3] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, 2005, vol. 1, pp. 886–893 vol. 1.

[4] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” *arXiv [cs.CV]*, 2015.

[5] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” *arXiv [cs.CV]*, 2015.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *arXiv [cs.CV]*, 2015.

[7] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[8] Larxel, “Road sign detection.” 24-May-2020.

[9] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” *arXiv [cs.CV]*, 2018.

[10] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” *arXiv [cs.CV]*, 2020.

[11] *Yolov5*. .

[12] C. Li *et al.*, “YOLOv6: A single-stage object detection framework for industrial applications,” *arXiv [cs.CV]*, 2022.

[13] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” *arXiv [cs.CV]*, 2022.

[14] J. Pedoeem and R. Huang, “YOLO-LITE: A real-time object detection algorithm optimized for non-GPU computers,” *arXiv [cs.CV]*, 2018.

APPENDIX - I

The following cells require the dataset to be in its respective location.