Vehicle detection using YOLO in MATLAB

Shruti Shukla*, Georgios I. Orfanidis*, Gabriel Gilman*
*Department of Electrical Engineering and Computer Science Florida Atlantic University, Boca Raton, FL 33431
E-mail: {sshukla2020, gorfanidis2021, ggilman2017}@fau.edu

***Abstract*—One of the most challenging problems in the field of object detection, which is a field of image processing and artificial intelligence. Reason being is that it combines object location within a scene with object categorization. In recent years, with the development of smart systems, there is a substantial need for those smart systems to recognize and detect objects that people use. One particular algorithm is the You Only Look Once (YOLO) algorithm. There are many benefits to effective tools that can use this algorithm to detect objects. One particular case is the use of detect pathogens in an object that can potentially be harmful to us. Another case is for smart systems to distinguish objects so that it won't confuse one object to another, which can somewhat lead to negative consequences. The goal of this study is to develop an object identification model that can recognize a wide variety of things that may be located inside structures. It will leverage the You Only Look Once (YOLO) real-time object detection technology to make this happen.**

***Index Terms*—YOLO, Object Detection, Artificial Intelligence, Smart Systems, Deep Learning.**

I. Introduction

Neural networks provide many alternatives that specialize in object detection. One of the algorithms is the YOLO algorithm. Object detection is one of the fundamental aspects of machine learning, since it is required to do it's namesake, to detect objects . Many additional computer vision tasks, like instance segmentation [5], [6], [7], [8], picture captioning [9], [10], [11], and object tracking [12], are built on object detection. Recent years have seen a tremendous advancement in deep learning techniques [13], which has dramatically accelerated the advancement of object identification, producing amazing innovations and drawing unheard-of levels of interest to it as a research hotspot. In many current real-world applications, including autonomous driving, robot vision, and video surveillance, object detection is now widely employed.

Object detection has progressed in since its creation in 2001. Over the years, many algorithms pertaining to object detection have been instantiated. Though deep learning has split into two different implementations. Before 2014, there were traditional object detection. After 2014, it became based on deep learning principles. Deep learning is based on using multiple stages to train a network. Object detection became divided again because some object detection algorithms required two-stages to detect images while some required only one. In fact, partially the reason why YOLO is so popular is because that it is the first object detection algorithm that only used one stage to detect objects. Figure 1 shows how object detection have advanced over the years.

The concept of object detection in computer vision includes identifying different things in digital photos or movies. Among the things found are people, vehicles, chairs, stones, structures, and animals. The fundamental questions to ask is what is the object and where it is? As mentioned before, there are many object detection algorithms that perform this but most can not solve the challenges of data limitation and modeling in a single algorithm run. The YOLO algorithm however, does.

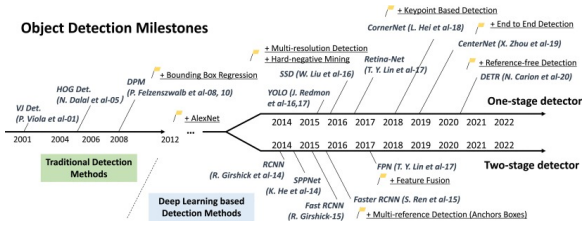


Fig. 1: Object detection timeline

A. Traditional object detection algorithms

Viola and Jones Detectors: In 2001, Viola and Jones [14], [15] successfully accomplished the first real-time detection of human faces without any limitations (such as skin color segmentation). The detector was tens or even hundreds of times quicker than other algorithms in its day with equivalent detection accuracy while running on a 700 MHz Pentium III CPU. The most basic detection method used by the VJ detector is sliding windows, which involves going through every conceivable position and size in an image to determine whether windows contain human faces. Although it sounds like a pretty straightforward procedure, the computation required was far more complex than the capabilities of computers at the time. Three crucial strategies have been incorporated into the VJ detector to significantly increase its detection speed: "integral image", "feature selection" and "detection cascades".

Histogram of Oriented Gradients (HOG) Detector: Dalal and Triggs [16] introduced the HOG feature descriptor in 2005. The scale-invariant feature transform [17], [18], and shape contexts [19] of its era may all be regarded significantly improved by HOG. The HOG descriptor is intended to be computed on a dense grid of evenly spaced cells and employ overlapping local contrast normalization (on "blocks") in order to balance the feature invariance (including translation, scale, illumination, and so on) with the nonlinearity. Although HOG may be used to identify a wide range of object types, the issue of detecting pedestrians served as its main inspiration. The HOG detector scales the input picture several times while maintaining the size of a detection window in order to identify objects of various sizes. For many years, the HOG detector has served as a crucial building block for several object

detectors [20], [21], [22] and a wide range of computer vision applications.

B. Two-Stage Detectors

He et al. (2014) [23] proposal for spatial pyramid pooling networks (SPPNet) is known as SPPNet. For example, AlexNet requires a 224 224 picture as its input in previous CNN models [24]. SPPNet's primary innovation is the addition of a spatial pyramid pooling (SPP) layer, which enables a CNN to provide a fixed-length representation regardless of the size of the picture or region of interest without rescaling it. In order to avoid repeating computing the convolutional features while using SPPNet for object identification, the feature maps may be calculated from the full picture once, and then fixed-length representations of any arbitrary sections can be created for training the detectors. Without compromising any detection accuracy, SPPNet is more than 20 times quicker than R-CNN (VOC07 mAP = 59.2%). Despite the fact that SPPNet has significantly sped up detection, it still has major limitations: first, training is still multistage; and second, SPPNet only fine-tunes its completely linked layers, disregarding all earlier layers. Fast RCNN [25], which was introduced later that year, offered a solution to these issues.

Fast RCNN: Girshick [25] suggested a Fast RCNN detector in 2015, which is an advancement over R-CNN and SPPNet [26], [27]. We can train a detector and a bounding box regressor simultaneously using fast RCNN and the same network parameters. Fast RCNN improved the mAP from 58.5% (RCNN) to 70.0% on the VOC07 dataset while having a detection speed over 200 times quicker than R-CNN. Even though Fast-RCNN effectively combines the benefits of R-CNN with SPPNet, its detection speed is still constrained by proposal detection (for more information, see Section II-C1). Next, the issue "can we generate object proposals with a CNN model" naturally emerges. Faster R-CNN [28] provided an answer to this query later.

C. One-Stage Detectors

A coarse-to-fine processing paradigm is used by the majority of two-stage detectors. While the fine refines the localisation based on the coarse detection and focuses more emphasis on the discriminate capacity, the coarse aims to increase memory ability. They can easily achieve great precision without using any bells and whistles, but because of their slow speed and immense complexity, they are rarely used in engineering. One-stage detectors, on the other hand, may obtain all objects in a single inference step. Their real-time and simple deployment capabilities make them popular with mobile devices, but their performance drops substantially when detecting dense and tiny objects.

Aforementioned, YOLO was a pioneer in one-stage detection. Since this paper is entirely based on the YOLO algorithm and implementing it, the researchers decided to mention it separately. Though the researchers would also like to take the time to mention another one-stage detection algorithm.

Single-Shot Multibox Detector (SSD): In 2015, Liu et al. [29] presented the SSD. The inclusion of multireference and multiresolution detection techniques (to be discussed in Section II-C1), which greatly increase a one-stage detector's detection accuracy, particularly for some tiny objects, is the major contribution of SSD. (COCO mAP@.5=46.5%, a fast version runs at 59 fps) SSD provides benefits in terms of both detection speed and accuracy. The primary distinction between SSD and earlier detectors is that SSD does detection on all levels of the network, whereas earlier ones only did so on the top layers.

II. YOLO ALGORITHM

You Only Look Once is known by the acronym YOLO. This algorithm (which operates in real-time) can find and identify different items in images. The class probabilities of the discovered photos are provided by the object identification process in YOLO, which is carried out as a regression problem.

A. Evolution of YOLO

Two fundamental neural network training algorithms that can be used for object detection are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks. RNNs and CNNs can both perform object detection by implemented by using multiple pretrained model layers.

You Only Look Once (YOLO): Joseph et al. [30] first suggested YOLO in 2015. In the era of deep learning, it was the first one-stage detector [31]. A quick version of YOLO runs at 155 frames per second with a VOC07 mAP of 52.7%, while its enhanced version runs at 45 frames per second with a VOC07 mAP of 63.4%. When compared to two-stage detectors, the YOLO approach uses a single neural network to process the whole image. The picture is divided into regions by this network, which concurrently predicts bounding boxes and probabilities for each region. Despite a significant increase in detection speed, YOLO has less accurate localisation than two-stage detectors, particularly for some tiny objects. This issue has received greater attention in YOLO's later versions [32], [33], [34], and the latest suggested SSD [35]. A follow-up project from the YOLOv4 team, YOLOv7 [52], has recently been suggested. It introduces improved structures, such as dynamic label assignment and model structure reparameterization, to exceed the majority of current object detectors in terms of speed and accuracy (ranging from 5 to 160 fps).

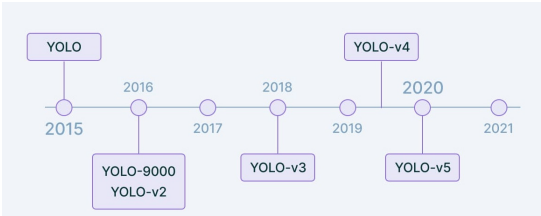


Fig. 2: Timeline of versions of YOLO

YOLO v1: This architecture was initially released in 2015. It predicts the class probabilities and bounding boxes of

several objects in an image using a single convolutional neural network (CNN), at the same time.

YOLO v2: This was an updated version of YOLO v1. It was released in 2016. A new network architecture with shortcut connections, batch normalization, anchor boxes, and other enhancements were incorporated into the previous version.

YOLO v3: Released in 2018, YOLO v3 offers an additional significant improvement over YOLO v2. It introduced functions like multi-scale prediction, feature pyramid networks (FPN), and a larger network with more convolutional layers.

YOLO v4: The latest and most powerful version of YOLO, v4 was released in 2020. It contains a number of novel methods, including Scaled-YOLO v4, Mish activation function, CSPDarknet53 backbone, and others.

YOLO v5: YOLO v5 is built on a new architecture than the previous YOLO iterations and was introduced in 2020 by a research team called Ultralytics.

The version that we have used here for this experiment is YOLO v2. The YOLO v2 algorithm, also referred to as YOLO-9000, was introduced in 2016 as a revision to the original YOLO algorithm. It was designed to detect a wider variety of object classes. It had an objetive to be more precise and fast than YOLO. This new version also incorporated Darknet-19, a CNN backbone that is distinct from CNN and is based on the VGGNet architecture with straightforward progressive convolution and pooling layers. The integration of anchor boxes was one of the main enhancements in YOLO v2. An assortment of bounding boxes that have various aspect ratios and scales are called ”anchor boxes.” YOLO v2 generates bounding boxes by combining the anchor boxes and expected offsets to come up with the final bounding box. This enabled algorithm to accommodate a wider range of item sizes and aspect ratios as a result. The inclusion of batch normalization, which helps to increase the model's accuracy and stability, is another development in YOLO v2. Another approach used by YOLO v2 is multi-scale training, which entails training the model on photos at various scales and averaging the predictions. This boosts the efficacy of small object detection.A new loss function that is more conducive to object detection tasks has been incorporated in YOLO v2. The sum of the squared errors between the predicted and actual bounding boxes and class probabilities serves as the basis for the loss function. [2]

B. YOLO Architecture

YOLO architecture resembles GoogleNet. It has overall 24 convolutional layers, four max-pooling layers, and two fully connected layers. [3], [4]

The architecture of YOLO functions by scaling down the input image to 448x448 before entering the convolutional network. A 3x3 convolution is used to create a cuboidal output, in succession of a 1x1 convolution which ensures the reduction of the number of channels. The final layer utilizes a linear activation function while ReLU is the activation function used in the background. The model is further regularized and prevented from overfitting using additional techniques like batch normalization and dropout.

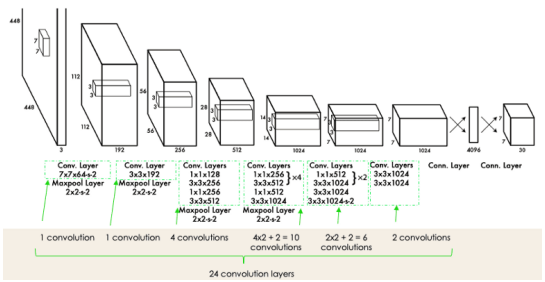
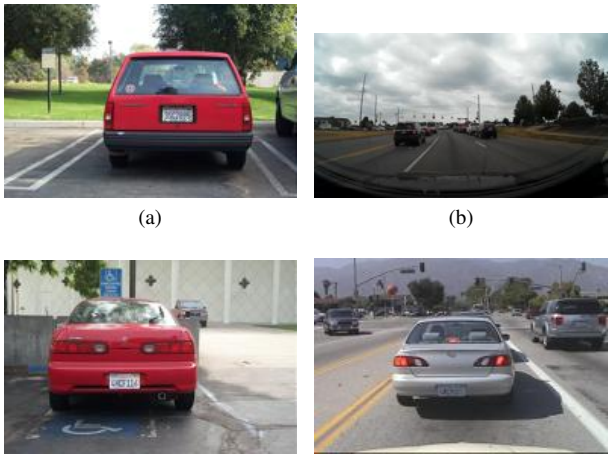


Fig. 3: Architecture of YOLO

III. Implementation Details

A. Dataset

To examine YOLO's potential for vehicle detection MATLAB's vehicle dataset was used. This dataset contains a collection of vehicle images of diverse types. The dataset is very well-suited for the problem on hand because the images of the vehicles were captured in diverse environments and conditions such as different lighting conditions, different angles, and different backgrounds. This diversity of the dataset makes it a great resource for implementing and evaluating machine learning models since the model can learn and improve from a wide range of real-world scenarios and achieve high testing performance. In Fig. 3 four arbitrary images from the dataset are shown. Evidently, the images are diverse in many different aspects. The environments where the vehicles are located are different (parking, highways, etc), the fidelity of the images varies, as well as the sizes of the vehicles changes. Some vehicles are captured from a very close distance while others are captured from far away.



(a) (b)

(c) (d)

Fig. 4: (a) to (d) are sample images from MATLAB's vehicle dataset.

The only limitation of the dataset we are considering herein is its size (the number of available images). In more detail,

in a vehicle detection problem implemented with YOLO, the size of the dataset is essential. Since YOLO, as described earlier in every detail, is a complex neural network architecture it requires a large amount of data to be trained effectively. The availability of additional data enables the model to learn more features and patterns that are responsible for detecting vehicles. Another reason, that YOLO specifically is a data-hungry model is that it divides the image under consideration into sub-images (in a grid fashion) and predicts bounding boxes for each sub-images. As a result, the spatial resolution of the predictions is negatively affected by a small dataset.

B. YOLO Implementation using MATLAB

For vehicle identification in Matlab, we have utilized a pre-trained YOLO v2 model as described in previous sections. In order to train and accurately evaluate the aforementioned model we partitioned the available dataset into two distinct sub-datasets: the training dataset and the testing dataset. In more detail, 80% was used for training the model while 20% was left out for testing purposes. For the training of the model, the following parameters were specified. We used stochastic gradient descent with momentum for the optimizer with an initial learning rate equal to $0,001$. Additionally, we set up the mini-batch size to be 16 and the maximum number of epochs to 30.

The basic implementation steps are outlined below:

1) Get the pre-trained YOLO v2 model: The weights and configuration files for the pre-trained YOLO v2 model are available for download from the official YOLO website.
2) Load Matlab vehicle dataset from vision toolbox and load the image dataset into Matlab and use it as your starting point for vehicle detection.
3) In Matlab, load the trained model: To load the pre-trained YOLO v2 model weights and configuration files, use an existing Matlab wrapper or create your own Matlab code.
4) Shuffle the training dataset and split to training (80%) and testing (20%) sets
5) Train the vehicle detector on training dataset
6) Evaluate the detector on testing dataset

IV. Experimental Results

In this section, we will demonstrate the performance of the YOLO architecture by means of precision and recall calculated on both the training and testing datasets.

Initially, during training, it can be seen in Fig. 4 that we achieve a mini-batch loss of 0.2 and a mini-batch RMSE of 0.4 at the last epoch. We could have increased the number of epochs to achieve a lower mini-batch loss and RMSE however, in that case, our model might have overfitted that data which in turn would have negatively affected the performance of the model on the testing dataset.

The training loss for the model which indicates how well the model is fitting is shown in the Fig. 5

```
*************************************************************************
Training a YOLO v2 Object Detector for the following object classes:

* vehicle

Training on single GPU.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     RMSE     |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:05 |         7.13 |         50.8 |          0.0010 |
|       3 |          30 |       00:00:10 |         1.38 |          1.9 |          0.0010 |
|       5 |          60 |       00:00:15 |         0.87 |          0.8 |          0.0010 |
|       7 |          90 |       00:00:20 |         0.74 |          0.5 |          0.0010 |
|       9 |         120 |       00:00:24 |         0.69 |          0.5 |          0.0010 |
|      11 |         150 |       00:00:30 |         0.68 |          0.5 |          0.0010 |
|      13 |         180 |       00:00:38 |         0.48 |          0.2 |          0.0010 |
|      15 |         210 |       00:00:45 |         0.54 |          0.3 |          0.0010 |
|      18 |         240 |       00:00:50 |         0.47 |          0.2 |          0.0010 |
|      20 |         270 |       00:00:54 |         0.44 |          0.2 |          0.0010 |
|      22 |         300 |       00:00:58 |         0.49 |          0.2 |          0.0010 |
|      24 |         330 |       00:01:02 |         0.47 |          0.2 |          0.0010 |
|      26 |         360 |       00:01:07 |         0.45 |          0.2 |          0.0010 |
|      28 |         390 |       00:01:11 |         0.35 |          0.1 |          0.0010 |
|      30 |         420 |       00:01:16 |         0.40 |          0.2 |          0.0010 |
|=========================================================================================|
Training finished: Max epochs completed.
Detector training complete.
*************************************************************************
```

Fig. 5: Training procedure

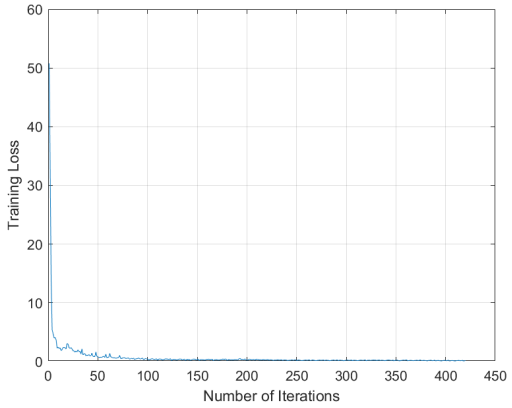


Fig. 6: Training Loss VS Number of Iterations

In Fig. 5 we can see the Training Loss decreases as the number of iterations increases. For this experiment three metrics Precision, Recall and Average precision are used to evaluate the performance of the detector. Precision is defined as the ratio of true positive instances over all positive instances of objects in the detector, based on the ground truth. Recall is the ratio of true positive instances over the sum of true positives and false negatives in the detector, based on the ground truth. The average precision provides a single number that incorporates the ability of the detector to make correct classifications (precision) and the ability of the detector to find all relevant objects (recall). Observing Fig. 6 we can conclude the model has been sufficiently trained on the training dataset. During the training phase the model achieves an average precision of 0.9 as it is illustrated in Fig. 6.

Similarly, the Fig. 7 depicts the Precision vs Recall graph for the testing phase. The average precision achieved here is of 0.73. As expected the testing average precision is lower than the training precision. The fact that is significantly lower, can be explained by the lack of enough training samples.

Next, Fig. 8 shows the performance of the vehicle detector on four arbitrary images from the testing dataset. The probability of detection is specified in the label box as well.

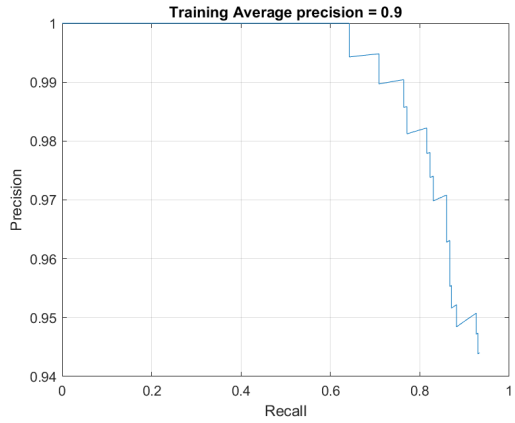


Fig. 7: Training Precision VS Recall curve

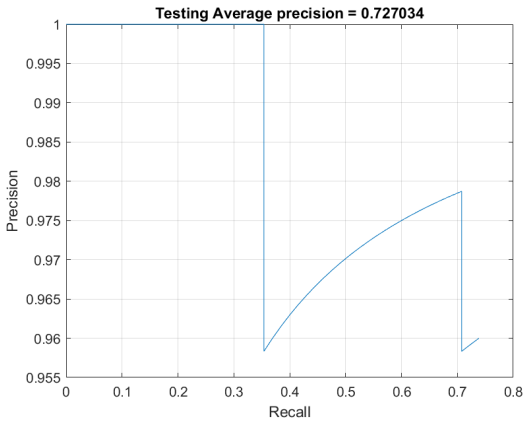
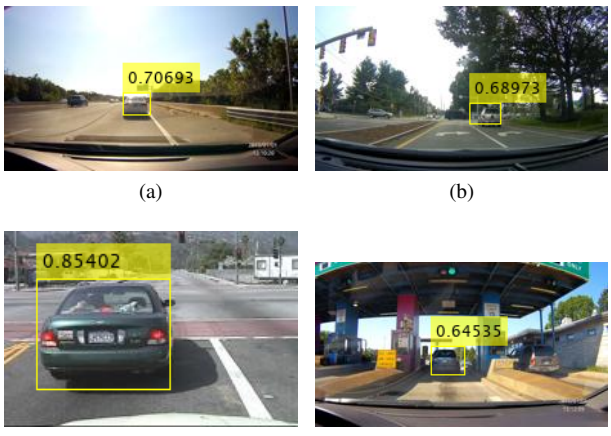


Fig. 8: Testing Precision VS Recall curve



(a) (b)

(c) (d)

Fig. 9: (a) - (d) show the performance of the model on the testing dataset. The probability that a vehicle is present in the image is also specified.

V. Conclusions

In this paper, we studied the problem of vehicle detection in diverse environments by the state-of-the-art model, YOLO. We used the publicly available vehicle dataset released by MATLAB in its Computer Vision Toolbox. We demonstrated via extensive simulation studies that YOLO can serve as a robust, reliable, and fast vehicle detector. Nevertheless, we observed that is heavily affected by the availability of data. The testing performance attained based on a small dataset was not sufficient to enable the use of the detector in real-life applications. We can conclude, that the training of the model using substantially more training samples can significantly increase the testing performance and enable real-time vehicle detection in a variety of environments.

VI. Future Work

To improve the testing performance of the model we will need to create a larger dataset. From the experiments we have performed, it looks like the number of available images in MATLAB's vehicle dataset is not sufficient to achieve performance that will enable the safe release of the model for real-world applications. Another direction would be to consider the newer version of the model, YOLO v3. However, since the newer version of the model is more computationally demanding it may affect the identification speed of vehicles which is of vital importance.

VII. Acknowledgments

This research was performed while the authors were enrolled in FAU's Digital Image Processing course taught by Professor Oge Marques. The authors were introduced to algorithms, tools, and principals for image processing which helped them invaluably in the preparation of this work. The authors would also like to thank Professor Oge Marques for his feedback throughout the process.

References

[1] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 6517-6525, doi: 10.1109/CVPR.2017.690.

[2] Rohit Kundu, "YOLO: Algorithm for Object Detection Explained [+Examples]." v7labs.com https://www.v7labs.com/blog/yolo-object-detection(accessed April, 14, 2023).

[3] Zoumana Keita, "YOLO Object Detection Explained." datacamp.com https://www.datacamp.com/blog/yolo-object-detection-explained (accessed April, 18, 2023).

[4] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[5] B. Hariharan, P. Arbeláez, R. Girshick and J. Malik, "Simultaneous detection and segmentation", Proc. ECCV, pp. 297-312, 2014.

[6] B. Hariharan, P. Arbelaez, R. Girshick and J. Malik, "Hypercolumns for object segmentation and fine-grained localization", Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), pp. 447-456, Jun. 2015.

[7] J. Dai, K. He and J. Sun, "Instance-aware semantic segmentation via multi-task network cascades", Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), pp. 3150-3158, Jun. 2016.

[8] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN", Proc. ICCV, pp. 2980-2988, Oct. 2017.

[9] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions", Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), pp. 3128-3137, Jun. 2015.

[10] K. Xu et al., "Show attend and tell: Neural image caption generation with visual attention", Proc. ICML, pp. 2048-2057, 2015.

[11] Q. Wu, C. Shen, P. Wang, A. Dick and A. van den Hengel, "Image captioning and visual question answering based on attributes and external knowledge", IEEE Trans. Pattern Anal. Mach. Intell., vol. 40, no. 6, pp. 1367-1381, Jun. 2018.

[12] K. Kang et al., "T-CNN: Tubelets with convolutional neural networks for object detection from videos", IEEE Trans. Circuits Syst. Video Technol., vol. 28, no. 10, pp. 2896-2907, Oct. 2018.

[13] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning", Nature, vol. 521, no. 7553, pp. 436, Feb. 2015.

[14] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR), pp. 1-9, Dec. 2001.

[15] P. Viola and M. J. Jones, "Robust real-time face detection", Int. J. Comput. Vis., vol. 57, no. 2, pp. 137-154, 2004.

[16] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection", Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., vol. 1, no. 1, pp. 886-893, Jun. 2005.

[17] D. G. Lowe, "Object recognition from local scale-invariant features", Proc. IEEE Int. Conf. Comput. Vis., vol. 2, pp. 1150-1157, Sep. 1999.

[18] D. G. Lowe, "Distinctive image features from scale-invariant keypoints", Int. J. Comput. Vis., vol. 60, pp. 91-110, Dec. 2004.

[19] S. Belongie, J. Malik and J. Puzicha, "Shape matching and object recognition using shape contexts", IEEE Trans. Pattern Anal. Mach. Intell., vol. 24, no. 4, pp. 509-522, Apr. 2002.

[20] P. Felzenszwalb, D. McAllester and D. Ramanan, "A discriminatively trained multiscale deformable part model", Proc. IEEE Conf. Comput. Vis. Pattern Recognit., pp. 1-8, Jun. 2008.

[21] P. F. Felzenszwalb, R. B. Girshick and D. McAllester, "Cascade object detection with deformable part models", Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., pp. 2241-2248, Jun. 2010.

[22] T. Malisiewicz, A. Gupta and A. A. Efros, "Ensemble of exemplar-SVMs for object detection and beyond", Proc. Int. Conf. Comput. Vis., pp. 89-96, Nov. 2011.

[23] K. He, X. Zhang, S. Ren and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition", Proc. ECCV, pp. 346-361, 2014.

[24] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet classification with deep convolutional neural networks", Proc. Adv. Neural Inf. Process. Syst., 2012.

[25] R. Girshick, "Fast R-CNN", Proc. IEEE Int. Conf. Comput. Vis. (ICCV), pp. 1440-1448, Dec. 2015.

[26] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", Proc. IEEE Conf. Comput. Vis. Pattern Recognit., pp. 580-587, Jun. 2014.

[27] K. He, X. Zhang, S. Ren and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition", Proc. ECCV, pp. 346-361, 2014.

[28] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks", Proc. Adv. Neural Inf. Process. Syst., pp. 91-99, 2015.

[29] W. Liu et al., "SSD: Single shot multibox detector", Proc. ECCV, pp. 21-37, 2016.

[30] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You only look once: Unified real-time object detection", Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), pp. 779-788, Jun. 2016.

[31] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement", arXiv:1804.02767, 2018.

[32] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection", arXiv:2004.10934, 2020.

[33] J. Redmon and A. Farhadi, "YOLO9000: Better faster stronger", arXiv:1612.08242, 2016.

[34] W. Liu et al., "SSD: Single shot multibox detector", Proc. ECCV, pp. 21-37, 2016.

[35] 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:2207.02696, 2022.