YOHO (You Only Hear Once): object detection in street view with audible output

Reza Mansouri
Georgia State University
rmansouri1@student.gsu.edu

Abstract

Object detection is a prominent discipline in computer vision, where its job is to recognize the objects in a scene, classify, and localize them. There are many algorithms for doing so, such as YOLO, R-CNN, Semantic Segmentation and so on. In line with that, assisting impaired people, specifically visually impaired ones, is a cause that technology has been focusing on for the past few years. In this project, I train a YOLOv2 model (via transfer learning and fine-tuning) to detect 9 object classes in street view, and further, develop an inference engine called *Tell a Vision* to obtain audible output from the model's predictions. With *YOHO*, a visually impaired person would be able to take a walk outside with their smart-phone in their hand while it tells them about their surroundings. The source code is available in jupyter notebook format at `http://github.com/rezmansouri/YOHO`.

Keywords: Object detection, YOLO, Transfer learning, Inference engine

1 Introduction

Convolutional neural networks are the heart of computer vision systems. The convolution operator perfectly grasps learning features from volumes of data. Images are 3-channel volumes of data, and thus, CNNs are the perfect choice for processing them.

Object detection refers to taking in an input image, and providing results such as where and what kind of objects are present in the scene. Many algorithms and architectures were proposed for object detection, namely, R-CNN [1], YOLO [2], and Semantic Segmentation with U-Net [3].

U-Net provides the most detailed results, where instead of predicting the coordinates and dimensions of objects, as done by R-CNN and YOLO, it can discern the scene down to the pixel level, i.e., classifying each pixel in the input image. The downside of semantic segmentation in real-time is that it's neither fast enough, nor predicting the output in a simple, able-to-infer format. YOLO on the other hand is fast, accurate enough, and gives the results in the form of bounding boxes, which are easy to infer. Many versions of YOLO were proposed and researchers improved its accuracy and speed.

In this project, YOLOv2's [4] architecture is used to create a model to perform transfer learning on, using the Bdd100K dataset [5]. This model can perform object detection in the streets with 9 object classes with adequate performance. Moreover, an inference engine called *Tell a Vision* [6] is developed to obtain scene descriptions from the model's predictions. This engine will take the predicted bounding box coordinates and classes from the model and give results such as: "There are two cars middle close,

one bike right far, two pedestrians left near." This architecture, as a whole, is called *YOHO (You Only Hear Once)*. It can have industrial applications, i.e., in production lines for robots, or in helping visually impaired people as a virtual assistant.

2 Problem definition

The goal is to train an object detection model and create a tool to infer the predictions such that descriptions of scenes in the streets can be later obtained. Hence, the first challenge is training.

The Berkeley DeepDrive dataset contains images taken from diverse locations in the United States. With ∼70,000 training image and annotation pairs, and ∼30,000 validation pairs, this dataset is a perfect choice for our application. The process of training won't be from scratch, instead, our model will be loaded with pre-trained weights [7] on the COCO [8] dataset, and we'll perform transfer learning on the model's last layer and fine-tuning the one before.

The second challenge is designing a fast procedure to summarize the model's predictions into a description that can be later transformed into an audible format. *Tell a Vision* is the answer to take the predicted bounding box coordinates and classes to do so.

3 Implementation

This section explains the architecture of the system, the pre-processing and post-processing, the issues I faced while training the model and the solutions I found, and finally the development of the inference engine, *Tell a Vision*.

3.1 Architecture

The YOLOv2 architecture roughly contains 6 consecutive blocks, where each block consists of a 3×3 convolution layer, the Leaky ReLU activation layer, and a 2×2 max-pooling layer. This will down-sample our $416 \times 416 \times 3$ input volume (the image) to a $13 \times 13 \times 1024$ volume. Finally, using a 1×1 convolution layer, we'll be able to get our final $13 \times 13 \times 70$ output volume. The reason for this output shape being so will be discussed later.

After acquiring the predictions in this shape and rescaling it to obtain the inferable bounding boxes and classes, *Tell a Vision* comes into play to first, locate the objects (i.e. specify the objects' horizontal and vertical locations) and second, specify the distance of the objects from the observer and finally, infer the description of the scene which can be transformed to audio format using text to speech. Figure 1 shows the architecture of our system.

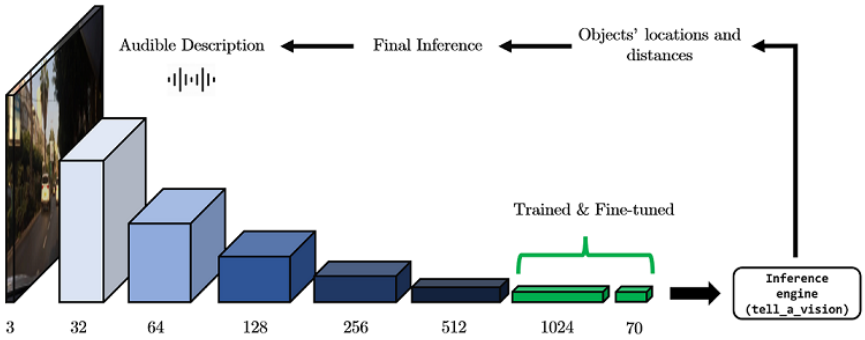

Figure 1: *YOHO's* architecture

3.2 Pre-processing

YOLO uses anchor boxes to alleviate the problem of overlapping objects. YOLO requires pre-defined anchor box dimensions for the model. Hence, I performed k-means clustering on the training set object dimensions with $2 \leqslant k \leqslant 10$ to obtain the anchor box dimensions, where the distance function was IoU (Intersection over Union) of the boxes.

To find how many bounding boxes were required, I used the elbow curve after plotting the clustering results. In my case, 5 clusters (anchor boxes) were the answer to the trade-off between accuracy and performance.

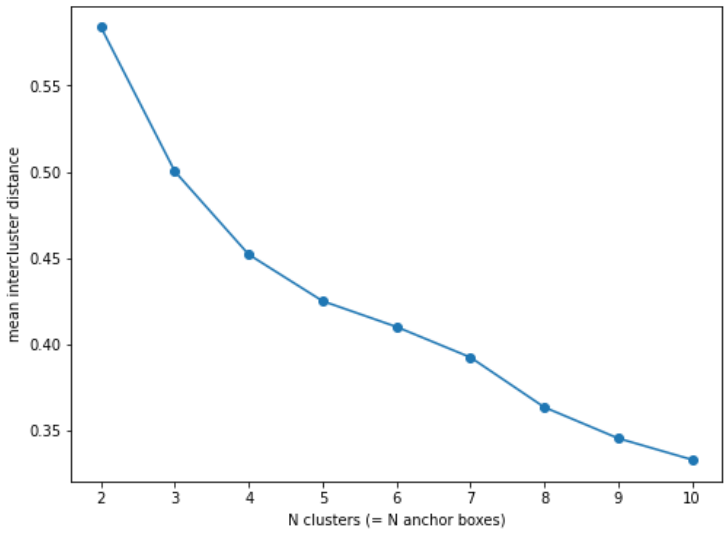

Figure 2: k-means result for acquiring the anchor boxes

These anchor boxes were used to create the corresponding label for each image in the dataset, where an object belongs to the anchor box with the highest IoU compared to the other anchor boxes, regarding only the dimensions.

$$IoU = \frac{Area\ at\ intersection}{Sum\ of\ areas}$$

Figure 3: IoU (Intersection over Union)

3.3 Post-processing

The reason behind the shape of the final output being 13 × 13 × 70 is that in each grid cell, there will be 14 prediction values for each anchor box (i.e., 13 × 13 × 5 × 14). The figure below depicts this.

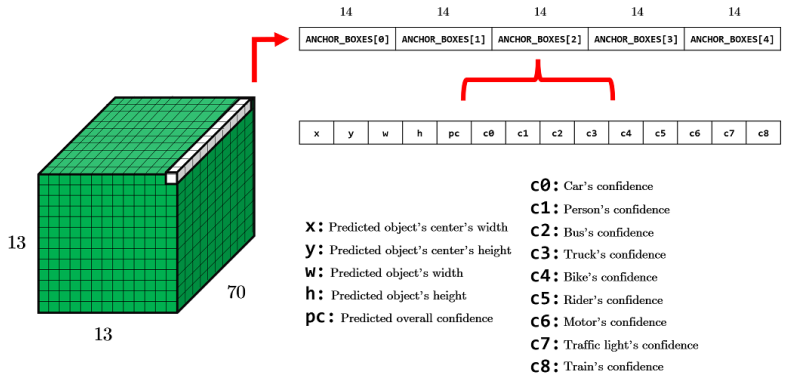

Figure 4: Model's output

Rescaling the predictions

Each element in a 1 × 14 vector here, has its own meaning and boundaries. These numbers can be any real number. Therefore, we cannot apply a single activation function on the whole volume. Instead, we need to rescale it according to each element's purpose. For example, confidence scores should be between 0 and 1, whereas

predicted widths and heights should be in grid scale (i.e., between 0 and 13). The following calculations must be done to acquire the correct predictions:

$$\hat{x}_i = \sigma(\tilde{x}_i) + i \tag{1}$$

$$\hat{y}_j = \sigma(\tilde{y}_j) + j \tag{2}$$

$$\hat{w} = e^{\tilde{w}} \times corresponding\ bounding\ box's\ width \tag{3}$$

$$\hat{h} = e^{\tilde{h}} \times corresponding\ bounding\ box's\ height \tag{4}$$

$$\hat{pc} = \sigma(\tilde{pc}) \tag{5}$$

$$\hat{c}\ [0...8] = softmax(\tilde{c}\ [0...8]) \tag{6}$$

Where in equations (1) and (2), i and j are the predicted object's center's width and height grids indices, and σ is the sigmoid function. In equations (3) and (4), width and height cannot be negative; hence, we use the exponential of the predicted values. The rescaled predictions are mentioned with a hat ($\hat{}$) and the raw predictions are shown with a tilde ($\tilde{}$) on top.

Acquring final scores

Further, we need to calculate final class scores, meaning: let pc and c $[0 \ldots 8]$ show their purpose as a whole:

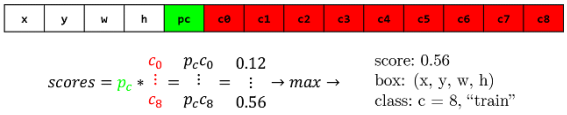

Figure 5: Final scores to distinguish objects' classes

After calculating these scores, by taking the maximum of them we'll be able to distinguish the predicted object's class.

Non-Max Suppression (NMS)

Our object detection system will output region proposals for the objects of interest. These proposals are usually redundant as may many refer to a single object. Non-max suppression [9] is a technique to filter out these proposals. I used the implementation from TensorFlow's image module. This is what NMS results in.

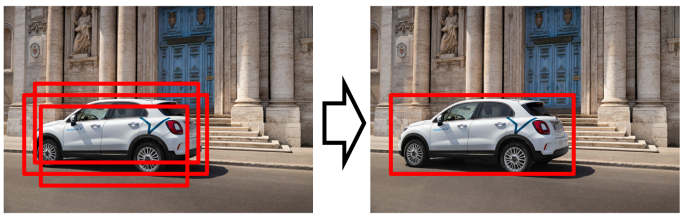

Figure 6: Non-Max Suppression

3.4 Training

At first, the model is loaded with pre-trained weights on the COCO dataset while the last layer's weights are initialized, and only the last two layers are set to be trainable. The training is done with the Adam optimizer [10], and according to the loss function described next. The training is done using an Nvidia Tesla T4 GPU, and TensorFlow.

Loss function

The loss function is similar to a Mean Squared Error (MSE) calculation. It encompasses three losses, one for the bounding boxes (x, y, w, and h), one for the overall confidence score (pc), and another for class confidence scores (c [0...8]).

$$loss_{xywh} = \sum pc \times [(x - \hat{x})^2 + (y - \hat{y})^2 + (\sqrt{w} - \sqrt{\hat{w}})^2 + (\sqrt{h} - \sqrt{\hat{h}})^2] \tag{7}$$

$$loss_{pc} = \sum pc \times (pc - \hat{pc})^2 + \sum (1 - pc) \times (pc - \hat{pc})^2 \tag{8}$$

$$loss_c = \sum \sum_{i=0} pc \times (c_i - \hat{c_i})^2 \tag{9}$$

$$loss = \frac{loss_{xywh} + loss_{pc} + loss_c}{batch\ size} \tag{10}$$

Figure 7: Example results

Issues: exploding gradients

Exploding gradients are a problem when large error gradients accumulate and result in very large updates to neural network model weights during training. Gradients are used during training to update the network weights, but typically this process works best when these updates are small and controlled.

As mentioned, the last layer's output could contain any real number, and to alleviate the problem of having negative predicted widths and heights of the objects, I applied the exponential function on them. This is something that we should be careful about, because if the last layer's weights are not initialized with small enough values, the optimizer will receive `+inf` values during forward propagation which cannot be back-propagated.

In order to avoid this issue, I tried many things. At first, the weights were initialized using Xavier Glorot's uniform method [11] by default. These weights were high enough to halt the training. Then, unaware of what was causing this, I trained the model only

on 2000 images, reduced the optimizer's learning rate, tried AMSGrad [12], and was not successful.

After dealing with this conundrum for some time, I realized that the weight initialization on the last layer was the problem. Therefore, I initialized the weights using a normal distribution scaled with `1e-4`. Fortunately, these were the weights that the training could be started with.

3.5 Tell a Vision

Tell a Vision (`tv`) is an inference engine in the form of a Python package that can provide explanatory analysis on the output of object detection algorithms. These algorithms' results include bounding box coordinates, objects' classes, confidence scores, etc., just like YOLO.

Through processing the bounding box coordinates and objects' classes, *tv* can infer the distance from the objects (or their relative size), their position in a scene (right, middle, left, above, or bottom), and finally output an audible description of the scenes. `tv` was implemented using NumPy for fast vectorized calculations and gTTS (Google Text To Speech) for obtaining the narration of the scenes in audio format. The rest of this section will explain how this inference engine works.

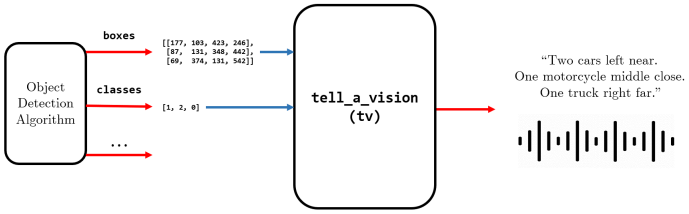

Figure 8: *Tell a Vision*

Step one: specify the location

`tv` decides if an object is on the right, the left, or in the middle (and above, midst, or at the bottom) by comparing its coordinates with an imaginary vertical (or horizontal) line in the middle of the scene. If a predefined portion of an object's width (or height) is placed on the left/right of this line, it is considered in that side's direction, otherwise, it is taken as in the middle. This process is implemented in a vectorized way for efficiency, and not in a greedy, for-loop approach. Figure 9 depicts these boundaries.

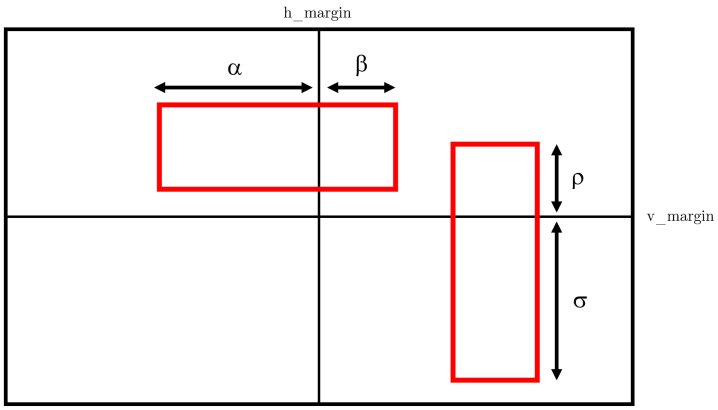

Figure 9: The boundaries to discern objects' locations

With two parameters, `h_point` and `v_point` which specify the predefined horizontal and vertical portion of an object's dimension as a threshold to discern its location, the task is to cover the following conditions:

- If $\alpha >$ `h_point` $\times$ object's width, it is considered on the left.
- If $\beta >$ `h_point` $\times$ object's width, it is considered on the right.
- Otherwise, it is considered in the middle.

And vertically,

- If $\rho >$ `v_point` $\times$ object's height, it is considered above.
- If $\sigma >$ `v_point` $\times$ object's height, it is considered at the bottom.
- Otherwise, it is considered in the midst.

Step two: estimate the distance

In order to tell if an object is close or far (small or big) `tv` needs to be fit on a dataset to obtain the quartile ranges of a class's area in advance. Accordingly, during runtime, the area of the objects found will be calculated to figure out their quartile, and finally, be assigned a size/distance label.

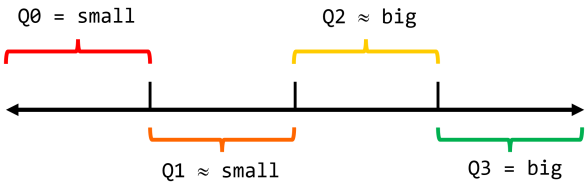

Figure 10: Area quartiles and their corresponding sizes

After these steps, the engine will accumulate the results to obtain the final inference. For example, for the objects in the scene below, the following results will be produced.

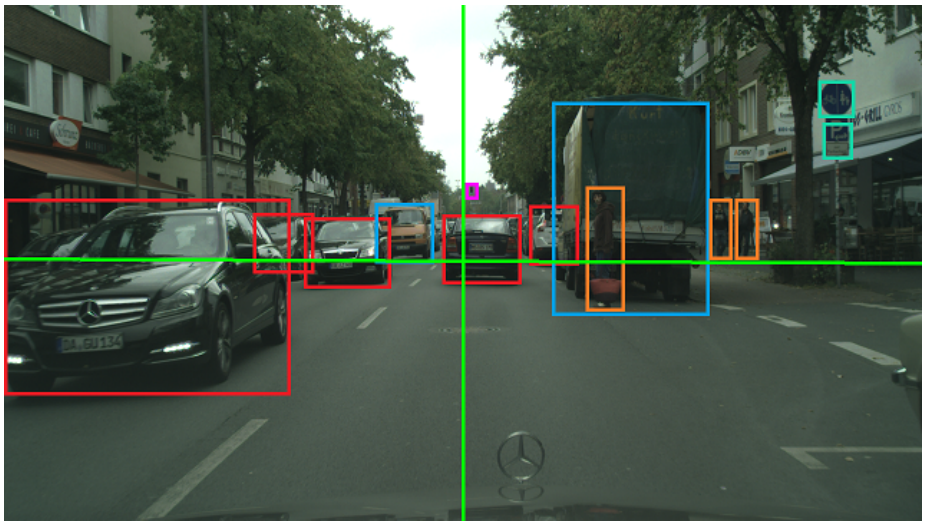

Figure 11: Example scene

- One car left close
- Two cars left near
- One truck left near
- One car middle near
- One traffic light right far
- One car right near
- One truck right close,
- One pedestrian right close,
- Two pedestrians right near,
- Two traffic signs right near

Afterwards, these inferences are ready to be narrated using text-to-speech.

4 Results and discussion

The model was trained for 77 epochs. Inspecting the losses, while the training loss kept decreasing, the validation loss stopped decreasing after the 22^{nd} epoch. Therefore, the weights put in use were from the 22^{nd} epoch. This is an approach known as early stopping, where if there has not been an improvement during training for some epochs, in our case, the validation loss, the training is stopped. Figure 12 plots the losses based on the number of epochs.

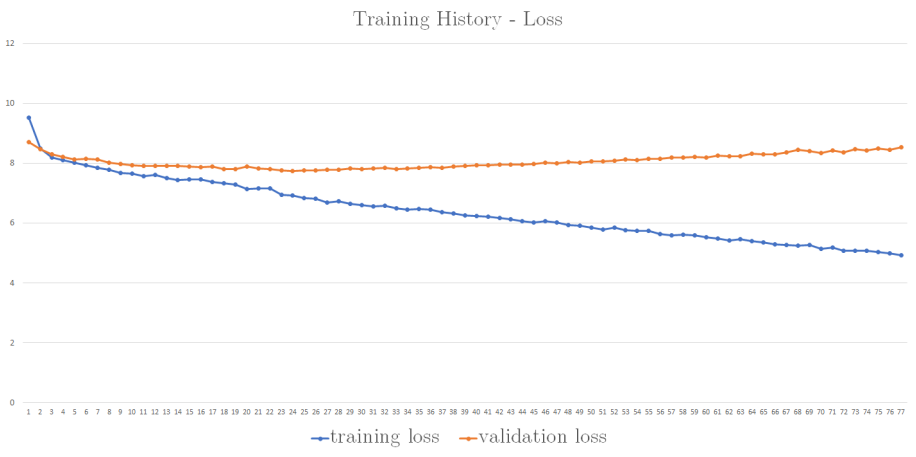

Figure 12: Training history

5 Conclusion

In this project, I created an object detection system with an inference engine to infer the results in audible format. With its adequate accuracy and efficiency, *YOHO* can be applied in assisting the visually impaired while commuting in the city. It may also be applied in the industry, such as robots in production lines in factories that may require succinct inference for object detection.

While only the idea of *YOHO* was implemented, there are lots of unexplored obstacles on the path to put it into production, such as observing the performance on mobile devices, adding a short-term memory to avoid redundant inferences, and etc.. Future works can include the improvement of the model with a better loss function, or modifications to the inference engine for more intuitive results.

References

[1] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[2] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[4] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[5] Fisher Yu et al. "Bdd100k: A diverse driving dataset for heterogeneous multitask learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2636–2645.

[6] URL: https://github.com/rezmansouri/tell_a_vision.

[7] URL: https://pjreddie.com/darknet/yolo/.

[8] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 740–755. ISBN: 978-3-319-10602-1.

[9] Navaneeth Bodla et al. "Soft-NMS — Improving Object Detection with One Line of Code". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 5562–5570. DOI: 10.1109/ICCV.2017.593.

[10] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[11] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[12] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. "On the convergence of adam and beyond". In: *arXiv preprint arXiv:1904.09237* (2019).