YOLO Object Detector for Onboard Driving Images

Albert Soto i Serrano

Abstract– With the evolution of artificial intelligence and, specially, machine learning, tech and car manufacturing companies are in research of the car of the future. Along with the arrival of new powerful hardware, deep learning is expected to be one of the most outstanding fields in the automotive sector. In this paper, we will be developing an object detection system with neural networks using the You Only Look Once (YOLO) network architecture. We will train and evaluate the model using various datasets and extract conclusions on its feasibility for autonomous driving or other driving assistance applications.

Keywords– Deep Learning, Convolutional Neural Networks, Object Detection, YOLO, ADAS

1 Introduction

ADVANCED Driving Assistance Systems, commonly known as ADAS, is one of the most outstanding fields nowadays. The evolution of artificial intelligence and, specially, machine learning, has lead tech and car manufacturing companies to the research of the car of the future. Conventional ADAS technology could perform basic object detection and classification, driver alerting of hazardous road conditions and eventually vehicle speed regulation. The new hardware and software platforms brought the world of deep learning to ADAS, providing a spectrum of new capabilities such as advanced object recognition systems capable of distinguishing a tourism from a police car from an ambulance or interpreting speed limit signals.

1.1 Objectives

The objectives of this project are:

- Develop an object detection system using YOLO neural-network-based model, capable of recognizing typical objects present on a driving session, such as cars, trucks, pedestrians, cyclists or even animals.
- Evaluate the convenience of the YOLO detection system for autonomous driving and other ADAS applications.

The object detection system will be developed using the You Only Look Once neural network architecture, which claims to be one of the most promising deep learning detection systems out in the wild.

We will have to evaluate the performance using the original YOLO weights against fine tuning with other datasets. So another tasks will be finding and adjusting proper datasets to fine-tune the model to increase its performance for our specific model.

1.2 Methodology

During the development of the project, we have worked following a methodology based on SCRUM. SCRUM is an agile development methodology based on iterations (commonly called sprints) of short duration. Each sprint is assigned with a list of small tasks that is tracked upon completion. Due to the reduced number of members, the role policy of SCRUM has not been followed to the letter.

For each sprint, we defined the tasks to be developed based on the project planning. At the end of each sprint, we made a retrospective look of the amount of work that was completed and rescheduled the uncompleted work for the next sprints.

To keep the software under control, we worked with a GitHub repository for code and script development.

1.3 Document structure

This document begins with a brief introduction to the project, its objectives and scope, emphasizing on the new horizons of deep learning in the automotive field. Following the brief introduction, the problem of developing an object detection system will be detailed. Later, we will focus on the YOLO detection system. Finally we will expose and analyze the obtained results and extract conclusions.

• Contact e-mail: albert.sotoi@e-campus.uab.cat
• Major: Computer Science
• Tutored by: Dr. Antonio M. López (CVC)
• Course 2016/17

2 Developing an Object Detector with Neural Networks

The following sections will describe the task of creating an object detection system using neural networks and other elements that are involved, such as the used datasets.

2.1 Classification and detection

Object classification is the task of determining the presence or absence of a certain object in an image. Results are binary: object is present or object is absent.

Object detection is the task of determining the presence or absence of a certain object in an image and the location of this object in the image. Location is usually represented using bounding boxes (see Ground Truth section below).

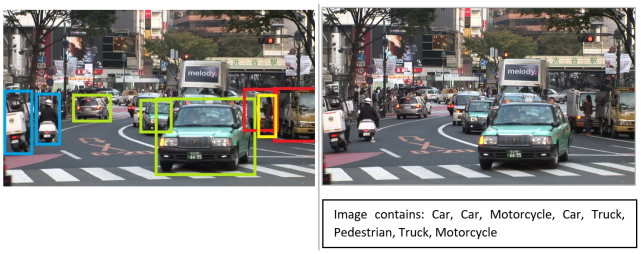


Fig. 1: Obj. detection output (left) vs Obj. classification output (right)

Thus, object detection is a more complex procedure, since it includes object classification plus further computations to determinate the location of the objects in the image.

2.2 Convolutional Neural Networks (CNN) against traditional Neural Networks

We will not be digging into neural networks in this article but it is important to understand the basis of their operation to understand how YOLO works. Especially, we will focus on what convolutional neural networks are, how do they differentiate from traditional neural networks and what are they composed of.

Neural networks can be described as a biologically-inspired programming paradigm which enables a computer to learn from observational data [12].

Traditional neural networks consist of an input and output layer and a collection of fully-connected hidden layers of artificial neurons. They take a vector as input, and transform it through the hidden layers up to the output layer where class scores are presented.

Traditional networks do not scale properly with large images. The fully connected layers involve a real problem with parameter scalability and bad generalization of the model, also known as overfitting.

On the other hand, convolutional neural networks take as premise that we will be working with images to constraint its architecture in a way to solve the weaknesses of traditional neural networks.

Neurons on the layers of a CNN are arranged in three dimensions: width height and depth. At the output of the network full images are reduced into a single vector of class scores.

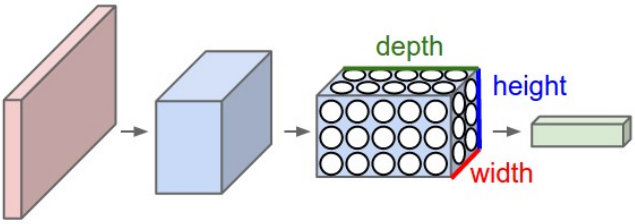


Fig. 2: Example of a Convolutional Neural Network (CNN) [13]

Every layer of a CNN transforms the previous volume of activations to the next using a differentiable function. Network architects are free to design the network the way they wish, but usually a set of operations/layers are common in all of the networks:

- Input layer: holds the raw values of the image, having a volume equal to the image dimensions.
- Convolutional layer: computes the output of the locally connected regions in the input of the layer applying the dot product between their weights and the region to which they are connected. After a convolutional layer, height and width dimensions are usually reduced in favor to a grown of the depth of the activation volumes, that will later represent the classification scores.
- Rectified Linear Unit (ReLU) layer: applies an element-wise activation function, such as max(0,x) thresholding at zero, where x is the input to a neuron. ReLU layers do not alter the size of the volume as convolutional layers do.
- Pooling layer: performs down-sampling operations on width and height dimensions altering the activations volume.
- Fully-Connected layers: the output layer where the class scores will be computed, resulting in a vector with a depth equal to the number of classes. All the neurons of the layer are connected to all the neurons of the previous layer.

All of the previous explanation corresponds to the task of classification but not detection. When performing the task of classification in neural networks, we are limited to a class per image and we can not straightly get the location of the object in the image.

For object detection we can apply several techniques. The simplest is applying a regression, usually by attaching another fully-connected layer to the last convolutional layer to compute the bounding box. This method will only work with one object per image, letting us with what is known as

an object localizer.

To achieve multiple object detection, we can divide the input image into regions. For example, by creating a region proposal network to get regions to be treated individually. The region division can be performed in different steps of the process, changing the behavior and performance of the system. Region based detectors are slow, since they might require lots of extra steps.

2.3 Datasets

In this project we will be using several datasets, directly or indirectly, for training or evaluating. The Data Information Specialists Committee (DISC) defines a dataset as a group of data files along with the documentations files (such as codebook, technical or methodology report, data dictionary) which explain their production or use [1]. They are generally unusable for sound analysis by a second party unless it is well documented.

2.3.1 Ground Truth

Ground Truth is factual data that has been observed or measured, can be analyzed objectively and has not been inferred [2]. Specifically, in image recognition technologies, ground truth is information obtained by direct visual examination, especially as used to check or calibrate an automated recognition system [3].

In our case, for object detection, the Ground Truth is a set of data describing the position, size and class of the objects present in the dataset images. With position and size information we can generate what is called a Bounding Box.
A Bounding Box, in a bi-dimensional image scenario, is an area defined by two points plus a width and height or four points generating a rectangle.
The rectangle shape is somehow a limitation for detection systems, since most real objects do not usually have a rectangle shape, but it simplifies computation and ground truth definition.

TABLE 1: EXAMPLE OF A BOUNDING BOX DEFINED IN THE GROUND TRUTH

Class	X	Y	Width	Height
Car	1	1	4	3

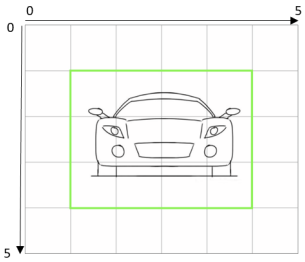


Fig. 3: representation of Bounding Box from Table 1

2.3.2 Udacity Dataset

The Udacity dataset for object detection [4] was created together by Google's Self-Driving Car project and Udacity. The aim of the project is to develop an open source self-driving car with user contributions.

It includes images of driving in Mountain View (California) and neighboring cities during daylight conditions. It contains over 65.000 labels across 9.423 frames collected from a Point Grey research cameras running at full resolution of 1920x1200 at 2Hz. The dataset Ground Truth was annotated using a combination of machine learning and humans [5].

Fig. 4: samples from the Udacity dataset

The ground truth has annotations for 3 different classes: cars, trucks and pedestrians. The number of classes may seem limited for a complete autonomous driving object detector but it is interesting to see which results we can achieve.

Also, the Udacity dataset lacks from a complete documentation, so no proper analysis can be done. Because of these two reasons, the Udacity dataset will not be our reference dataset for this project but it will be tested anyway to asses the performance of YOLO in different datasets.

The dataset has been modified to be compatible with YOLO: images were cropped to square images of 320 x 320 pixels (Figure 5) and the ground truth format was changed from the original four point coordinates to the initial horizontal and vertical coordinates plus height and width of the boxes.

Fig. 5: samples from Udacity dataset once adapted

2.3.3 KITTI Benchmark Dataset

KITTI Vision Benchmark Suite datasets [7] are captured from driving sessions around the city of Karlsruhe (Germany), in rural areas and on highways. Up to 15 cars and 30 pedestrians are visible per image. The dataset consists of 7481 training images with a resolution of 1240 x 375 pixels.

The number of classes notated in the dataset ground truth is 11. These classes include:

- Background
- Car
- Van
- Truck
- Pedestrian
- Person sitting
- Cyclist
- Tram
- Misc.
- Don't care

Compared to the three classes of the Udacity dataset, with the KITTI dataset we expect to be able to get a more complete detection system. Classes like tram or cyclist are crucial for an autonomous driving system since car behavior will dramatically vary depending on the type of vehicle ahead.

Fig. 6: samples from the KITTI benchmark dataset

The main classes present in the dataset are: cars, with 28.742 samples, followed by pedestrians, with 4.487 samples and cyclists, with 1.627 samples.

KITTI will be our reference dataset since it is a properly documented dataset and it has annotations for more classes. Also, the images quality seems to be higher than the Udacity dataset one's, especially when speaking of exposure control and dynamic ranges, being the images on the KITTI dataset more exposure balanced and with better contrast.

There are, though, many inconveniences with the KITTI dataset: the first and most important inconvenience is that no testing set is provided with the dataset, so part of the training set had to be taken for testing. The other main inconvenience is the size and aspect ratio of the images which lead to incompatibility and memory issues when training the model. This last issue was fixed by resizing the images and readapting the ground truth.

2.3.4 VOC 2007 Dataset

The PASCAL VOC 2007 dataset [8] consists of 9.963 images, containing 24.640 annotated objects from 20 different classes. Data is split into 50% for training/validation and 50% for testing. The distributions of images and objects by class are approximately equal across the training/validation and test sets.
In this project, we did not work with this dataset directly, but we used original YOLO pre-trained weights on this dataset [6]. Hence, since we used the pre-trained weights in our experiments, we need to know what the contents of the dataset to determinate if the weights could improve our results.

The 20 classes present in the dataset are:

- Airplanes
- Bicycles
- Birds
- Boats
- Bottles
- Buses
- Cars
- Cats
- Chairs
- Cows
- Dinning tables
- Dogs
- Horses
- Motorbikes
- People
- Potted plants
- Sheep
- Sofas
- Trains
- TV/Monitors

At a first glance, there are a lot of classes that are not of our interest, since these objects will not be typically present in a driving environment, but there are some classes that could help us on tuning our detection system such as people, cars, buses, motorbikes, trains, bicycles and, also important, animals.

Looking in detail to these classes, we can see the number of annotated objects and the number of images in which the objects appear (Table 2).

TABLE 2: NUMBER OF IMAGES AND LABELS PRESENT IN TRAIN AND VALIDATION SETS OF THE PASCAL VOC 2007 DATASET

	Train		**Validation**		**Train + Validation**	
Class	Images	Labels	Images	Labels	Images	Labels
Train	127	145	134	152	261	297
Bicycle	116	176	127	177	243	353
Bus	97	115	89	114	186	229
Person	1025	2358	983	2332	2008	4690
Motorbike	120	167	125	172	245	339
Car	376	625	337	625	713	1250
Animals [1]	622	887	660	877	1282	1764

The number of objects of the classes in which we are interested is acceptable and could help improving the results of our model fine-tuning with other datasets, but we shall see how the pre-trained weights perform by themselves.

In Figure 7, we can see that the samples from the PASCAL VOC 2007 dataset are varied. We should evaluate how this level of variety affects to our results in a driving environment.

3 YOLO: You Only Look Once

You Only Look Once (YOLO from now on) is a state-of-the-art, real-time object detection system [6] that uses neural networks to detect objects in images.

Prior detection systems re-purpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

YOLO uses a different approach applying a single neural network to the full image. The network divides the image into regions and predicts bounding boxes and

[1] Animals include categories: horse, dog, cat, sheep and cow.

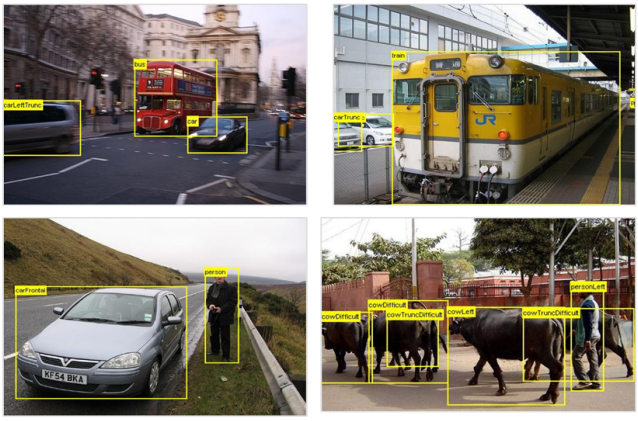


Fig. 7: samples from the PASCAL VOC 2007 dataset

probabilities for each region. These bounding boxes are weighted by the predicted probabilities (Figure 8).

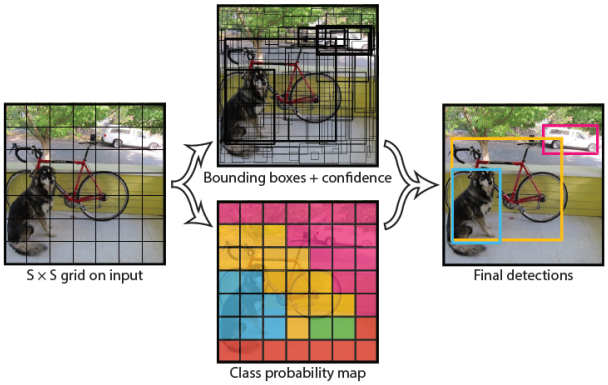


Fig. 8: YOLO image processing overview

3.1 Architecture

Now lets take a deeper look: YOLO detection network has 24 convolutional layers, followed by 2 fully connected layers. It alternates 1 x 1 convolutional layers to reduce the features space from the preceding layers [6]. Initial convolutional layers extract features from the image and the fully-connected layers predict the output probabilities and coordinates of the object.

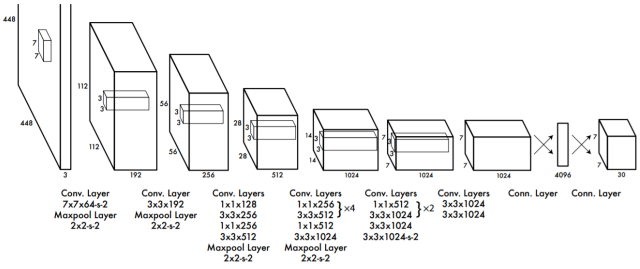


Fig. 9: YOLO network architecture

3.2 What to expect

From its authors, YOLO still lags behind state-of-the-art detection systems in accuracy. While it can quickly identify objects in images it struggles to precisely localize some objects, especially small ones. We will find out how big are those limitations and how do they affect to us in the specific field of autonomous driving.

3.3 Implementation

YOLO was originally implemented using Darknet [9], an open-source neural network framework written in C and CUDA. For our implementation of YOLO, we used Tensorflow [10], an open-source software library for Machine Intelligence developed by Google and Keras [11], a high-level neural networks API, written in Python and capable of running on top of either TensorFlow or Theano, as front-end.

4 Results

During the development of the project, lots of experiments were made, but most of them did not introduce any improvement or even lead us to unwanted results. We selected three experiments that gave us the best results on Udacity and KITTI datasets to resume the experimental phase of the project.

4.1 Metrics

The metrics used to quantitative evaluate the results can be divided into classification and detection metrics. We need to consider both since object detection includes object classification as part of its procedure.

4.1.1 Classification metrics

As stated in previous sections, the final classification output is a binary result: the object is present or is absent. From the result we can compute several metrics that can give us an idea of how good our classification system is.

Using the ground truth and predicted information, we can create a confusion matrix (Table 3).

TABLE 3: Confusion Matrix

		Predicted condition	
		True	**False**
True condition	**True**	True positive (TP)	False negative (FN)
	False	False positive (FP)	True negative (TN)

The most relevant metrics for classification that we are going to take as reference are:

- Precision: gives us an idea of how many selected items are relevant. We can compute it by dividing the number of true positive predictions between the sum of the true positive predictions and the false positive predictions.

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

- Recall: how many relevant items are selected. We can compute it by dividing the number of true positive predictions between the sum of the true positive predictions and the false negative predictions.

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

- F-Score: reference value. It can give us a general idea of how good the system performs. We compute it using the previous precision and recall metrics as seen in equation 3.

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (3)$$

4.1.2 Detection metrics

When it comes to detection metrics, we will take as reference Intersection over Union (IoU). Intersection over Union is the most common metric in object detectors as it is computable independently from the type of detector being used (SVM, CNN, ..). The only things that are required are the ground truth and predicted bounding boxes.

To compute the metric, we divide the area of intersection between the predicted box and the ground truth box by their union area (Equation 4).

$$IoU = \frac{Area\_of\_Intersection}{Area\_of\_Union} \quad (4)$$

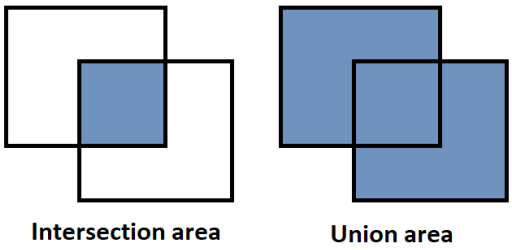


Fig. 10: visual representation of intersection and union areas between two bounding boxes.

We will not start considering a predicted box as good if its intersection over union value is below 40%. In Figure 11, we can see three examples of IoU values.

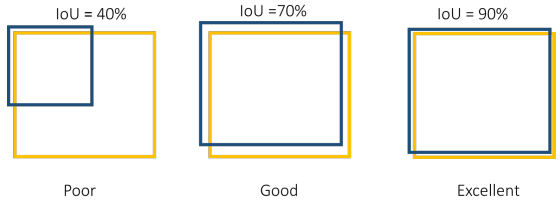


Fig. 11: Examples of IoU

4.2 Detection threshold

Since the output detections of our model are the probabilities of a detection being reliable, we can establish a threshold to discard predictions with low probability. This way we can adjust the level of reliability of our model and decide weather to be more or less restrictive.

4.3 Non-maximum Suppression

In object detection, it is common to face situations in which the system detects multiple overlapping bounding boxes for the same object. This shall not be considered as a bad indicator per se, because if the detection is right, it means that the system is working well. But obviously this is not the expected output of a good object detector and we need ways to fix these situations.

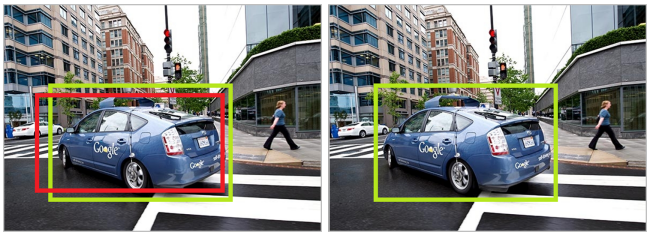


Fig. 12: Result of applying a non-maximum suppression (right)

It is usual to use non-maximum suppression algorithms to prune them. Non-maximum suppression algorithms compute the degree of overlapping between bounding boxes and, if the overlapping is greater than an specified threshold, the bounding box is removed.

4.4 Experiment 1: VOC pre-trained weights test on KITTI dataset

In this experiment, we tested YOLO pre-trained weights on the KITTI dataset. The weights were trained using the PASCAL VOC 2007 dataset as in the original implementation. Dataset has tags for 20 classes of very varied nature. Not all the classes are of our interest, but it was important to see how good they behaved in a driving environment.

TABLE 4: EXPERIMENT 1 TEST PARAMETERS

Parameter	Value
Test dataset	KITTI
Samples	481
Detection threshold	50%
Non-Maximum Suppression threshold	20%
Test image size in pixels (height x width)	320 x 1060

TABLE 5: EXPERIMENT 1 TEST RESULTS

Metric	Value
Intersection over Union	40.10%[2]
Throughput	2.25127 fps[3]

Before stepping into visual results, we can see that the throughput of the system is very low. This may be because of the resolution of the images (1060 x 320px) but it is still very far from a desirable performance.

The average intersection over union is 40%, which is

[2] Average IoU computed before applying detection threshold
[3] Throughput measures on Nvidia GTX Titan X 12GB

relatively low, but considering that we are running YOLO with pre-trained weights on a dataset not meant specifically for autonomous driving is not that far from what we would initially expect.

Fig. 13: results of experiment number one. Images numerated from left-right, top-bottom, 1-6.

Having a look into the visual results: overall, the performance of the system seems pretty poor, especially on vehicles, where it struggles the most. Bounding boxes do not cover the object properly, and sometimes they have random sizes, which are crucial for an autonomous driving system as they could be an indicator of the object distance to the car.

Sometimes, though, it performs quite well, as seen on Figure 13, images 3 and 5. It works especially well on persons, which, if we remember, was the class with more labeled samples in the training set.

On very large objects as the train on Figure 13, image 4, YOLO struggles to detect the whole object.

Also, it has trouble detecting far objects in group, as seen on Figure 13, image 3, where it missed the group of pedestrians on the background.

4.5 Experiment 2: Fine tuning the model with the KITTI dataset

In this experiment we will be fine tuning the pre-trained weights on the KITTI dataset. We did test training the model from scratch and compared it to train loading the pre-trained weights. In the second case, the performance improvement was clearly visible, so further experiments were made loading the pre-trained weights.

The first thing we can see is that precision and recall are very good. The intersection over union seems to be similar to the previous experiment. The performance is low, mainly because of the KITTI large images, with an average of only 2.35 fps.

Once again, having a look into the visual results we can see that the system still lacks from the same issues as in the previous experiments. For example, the truck in Figure 15, image 3, which bounding box is too small and is un-properly labeled as a car or in Figure 15, image 1,

[6]Average IoU computed before applying detection threshold
[5]Throughput measures on Nvidia GTX Titan X 12GB

TABLE 6: EXPERIMENT 2 TRAINING PARAMETERS

Parameter	Value
Dataset	KITTI
Training samples	6519
Validation samples	481
Learning rate	0.00001
Optimizer	rmsprop
L2 regularization	0.000005
Pretrained weights	VOC 2007
Epochs	10
Learning rate decay	None
Train image size in pixels (height x width)	320 x 448

Fig. 14: experiment 2 model training history

TABLE 7: EXPERIMENT 2 TRAINING RESULTS

Metric	Value
Average recall (train)	49.7%
Average Intersection over Union (train)	49.1%
Loss (train)	6.923
Average recall (validation)	53.7%
Average Intersection over Union (validation)	51.3%
Loss (validation)	6.6666

TABLE 8: EXPERIMENT 2 TEST PARAMETERS

Parameter	Value
Test dataset	KITTI
Samples	481
Detection threshold	65%
Non-Maximum Suppression threshold	20%
Test image size in pixels (height x width)	320 x 1060

TABLE 9: EXPERIMENT 2 TEST RESULTS

Metric	Result
Precision	58.93%
Recall	40.92%
F-Score	48.30%
Intersection over Union	40.96%[4]
Throughput	2.35578 fps[5]

Fig. 15: results of experiment number two. Images numerated from left-right, top-bottom, 1-6.

where YOLO has troubles detecting very large objects like the tram.

As in previous experiments, there are cases in which YOLO performs well, as in Figure 15, images 4, 5 and 6, in which detects almost all the objects present in the image and the bounding boxes size is the proper one.

4.6 Experiment 3: Fine tuning the model with the Udacity dataset

For this experiment, we trained the model on the Udacity dataset loading YOLO pre-trained weights.

TABLE 10: EXPERIMENT 3 TRAINING PARAMETERS

Parameter	Value
Dataset	Udacity
Training samples	8679
Validation samples	501
Learning rate	0.00001
Optimizer	rmsprop
L2 regularization	None
Pretrained weights	VOC 2007
Epochs	30
Learning rate decay	None
Train image size in pixels (height x width)	320 x 320

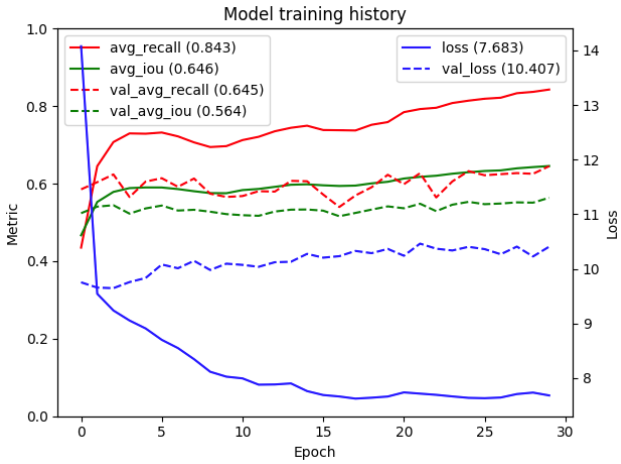


Fig. 16: Experiment 3 model training history

TABLE 11: EXPERIMENT 3 TRAINING RESULTS

Metric	Value
Average recall (train)	84.30%
Average Intersection over Union (train)	64.60%
Loss (train)	7.683
Average recall (validation)	64.50%
Avg. Intersection over Union (validation)	56.40%
Loss (validation)	10.407

TABLE 12: EXPERIMENT 3 TEST PARAMETERS

Parameter	Value
Test dataset	Udacity
Samples	2500
Detection threshold	65%
Non-Maximum Suppression threshold	20%
Test image size in pixels (height x width)	320 x 320

TABLE 13: EXPERIMENT 3 TEST RESULTS

Metric	Result
Precision	37.28%
Recall	20.63%
F-Score	27.10%
Intersection over Union	51.12%[6]
Throughput	9.58 fps[7]

From the values on Table 13, we can see how the throughput of the system has increased due to the lower size of the images compared to the KITTI dataset ones. It is still too low for a real-time detection system with a frequency of only 9.58 Hz.

Intersection over union is better than in experiment number one and it is above 50% so the performance of the system in terms of object detection is acceptable.

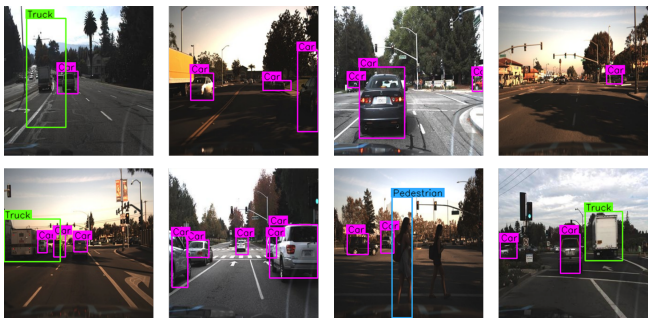


Fig. 17: results of experiment number three. Images numerated from left-right, top-bottom, 1-8.

Having a look into the third experiment visual results, we can see that the system performs a lot better than in the first experiment, especially in aspects such as the size of the bounding boxes.

In this case, bounding boxes have proper sizes, sur-

[4] Average IoU computed before applying detection threshold
[7] Throughput measures on Nvidia GTX Titan X 12GB

rounding the whole object in most of the cases, but sometimes taking too much size as seen with trucks on Figure 17, images 1 and 5. The Udacity dataset has only labels for 3 classes, which makes it very limited for a complete detection system, but still, results are overall decent.

5 Conclusions

YOLO is announced as one of the most promising detection systems using neural networks but, from the results we have obtained, we can say it is yet still far from being a perfect model, as it is not too far from a traditional object detector. Even if results can be improved, the detector behavior is still unpredictable in some situations.

Generally speaking, the model works, but not for our needs. For autonomous driving, the wrong size of bounding boxes and the miss rate is what really limits its usage. The size of bounding boxes could be used to determine the proximity of objects. On a real-world situation, we could not trust a system based on YOLO for autonomous driving, but it could help in other tasks, like warning alerts over high probability detections or augmented reality applications on the car's head unit.

Acknowledgements

This project would have not been possible without the guidance of our tutor, Dr. Antonio M. López and the help from David Vázquez. Special thanks to all the members of the Computer Vision Center (CVC) of Barcelona who helped us in one way or another and to the Engineering School members of the Autonomous University of Barcelona (UAB).

References

[1] Data Information Specialists Committee UK. Questions and Answers, 2010. [Online] Available: http://www.disc-uk.org/qanda.html [Accessed: 15- May- 2017].

[2] TopCoder.com, The Importance of Ground Truth in Data Science. [Online] Available: http://crowdsourcing.topcoder.com/the-importance-of-ground-truth-in-data-science/ [Accessed: 15- May- 2017].

[3] Oxford Dictionary, Ground Truth. [Online] Available: https://en.oxforddictionaries.com/definition/ground_truth [Accessed: 15- May- 2017].

[4] Udacity, We're Building an Open Source Self-Driving Car. [Online] Available: https://github.com/udacity/self-driving-car [Accessed: 15- May- 2017].

[5] Udacity, Annotated Driving Dataset. [Online] Available: https://github.com/udacity/self-driving-car/tree/master/annotations [Accessed: 15- May- 2017].

[6] J. Redmon and A. Farhadi, YOLO9000. Faster, Better, Stronger, 2016.

[7] A. Geiger, P. Lenz, R. Urtasun, Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite, 2012.

[8] Everingham, M. and Van Gool, L. and Williams, C. K. I. and Winn, J. and Zisserman, A., The PASCAL Visual Object Classes Challenge 2007 (VOC2007), 2007.

[9] Joseph Redmon, Darknet: Open Source Neural Networks in C, 2013–2016 [Online] Available: https://pjreddie.com/darknet/ [Accessed: 08- June- 2017]

[10] M. Abadi, A.Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015.

[11] Chollet, François and others, Keras, GitHub, 2015. [Online] Available: https://github.com/fchollet/keras [Accessed: 10- June- 2017]

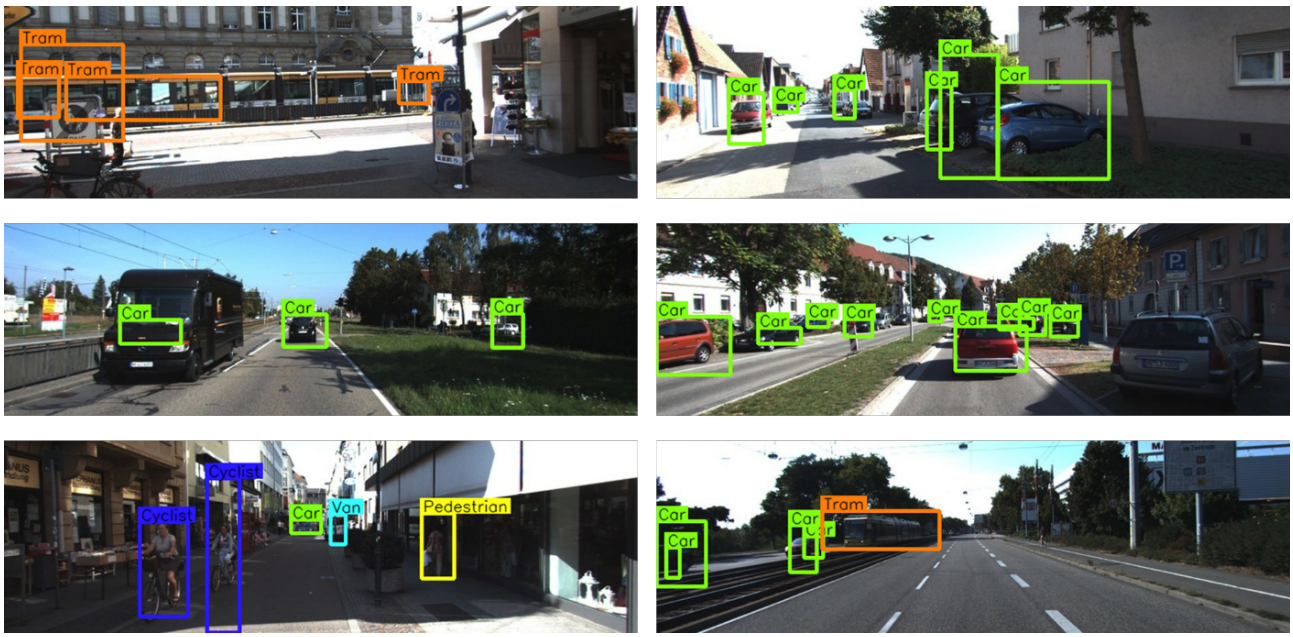
[12] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015.

[13] Stanford, Class CS231n: Convolutional Neural Networks for Visual Recognition [Online] Available: http://cs231n.github.io/convolutional-networks [Accessed: 10- June- 2017]

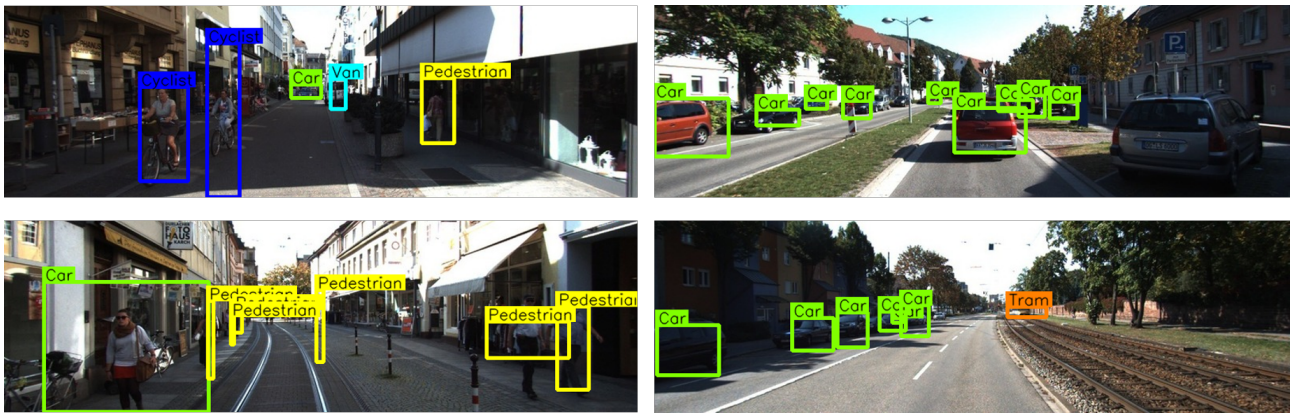
APPENDIX

A.1 Experiment 1 results

A.2 Experiment 2 results



A.3 Experiment 2 additional results



A.4 Experiment 3 results