Object detection with YOLOv3

Lucas Lovén lucassl@kth.se

Samuel Wintzell wintz@kth.se

Devrat Singh devrat@kth.se

Gustav Lenart glenart@kth.se

Department of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, 114 28

Abstract

Multiple applications of deep neural networks (NN) exist within industry and research. One area that has found extensive implementation is object detection - an extension of image classification. Here the prediction of the model is class label and bounding box. Implementations include robotics, autonomous vehicles and radiology. It is essential that the model predicts with accuracy and speed specifically within the two firstly mentioned domains. The YOLO architecture is a deep NN that emphasizes inference speed over accuracy, and regarding real-time object detection current versions of YOLO achieves higher mean average precision (mAP) compared to its contemporaries. Initial version of YOLO were known for not being able to detect small objects relative to the image size. In YOLOv3 the architecture was upgraded in order to rectify this. In this project we implement YOLOv3 using PyTorch and investigate the trained models ability to detect details based on the characteristics of training data. Our results show that indeed bigger objects are easier to detect due to salient features. Training on a data set with more contextual information results in a model that can generalize better as well.

1 Introduction

The goal of the group members was to get an understanding, beyond the mathematical syntax, of neural networks (NN) conventional applications in industry and research. Object detection was decided when discussing what area to specify the problem in. Object detection is an extension of the image classification problem in such a way that the trained models goal, in addition to predicting class, is to predict a bounding box for the respective objects. The group members sought practical training on implementing a NN with established python packages. One popular model within object detection that prioritize inference speed is the YOLO architecture [17].

In section 1 we formulate our problem, why it is important and related work. In section 3 we present and discuss the data that was used when training and validating our models. The intrinsic structure and syntax of YOLOv3 is described in section 4. Finally in section 5 and 6 we detail the experiments that were performed and discuss conclusions.

1.1 Problem

The problem that the project aims to investigate is object detection. Specifically the YOLOv3 architecture performance on object detection. Here we formulate some key points related to implementation, training and testing:

- Implement the YOLOV3 model from "scratch" using `PyTorch`. This was done by following a tutorial by Alladin Persson [15]
- Investigate YOLOv3 performance on two different datasets of varying difficulty.

- What performance is achievable by training networks on a subset of classes and images.
- Investigate YOLOv3 performance on detecting small versus large objects.

1.2 Why is this important?

Object detection has found its way into several industrial domains. Examples include analysis of medical imagery or decision making by autonomous vehicles. Regarding the former, the limitations created by radiologist put under stress can be reduced by the help of a trained NN model detecting and segmenting the essential parts of the imagery that is required.

The automotive industry is moving towards implementing systems fully utilizing autonomous driving and decision making. While fully autonomous vehicles are not on the market, semi-autonomous are. Such vehicles are defined by having the human driver act as fallback performance of the driving task [12]. The decision making of autonomous vehicles relies heavily on the vehicle "knowing" it's surroundings and an important technical application is object detection. Moreover the camera input processing and decision making needs to be in real time. Thus the module that handles the object detection needs to be as fast and accurate as possible. As already mentioned, the Yolo architecture including and succeeding version 3 is a NN architecture that emphasizes speed [17]. In conclusion the work of this project is important because it aims to give insight in the inner workings of a fast NN model, and its potential limitations.

2 Related work

One way to compare and discuss performance of a trained NN model is to use established benchmarks in the form of mean average precision (mAP) or intersection over union (IOU). Moreover training and validation data are transparent. With this method of comparison there is a potential that causes of limitations for each model are veiled by the distinguishing numbers (mAP etc.). Excluding network architecture the source can be tracked to the data, for this case images. Reasons include occlusion, viewpoint and relative size of object within image. This project will focus on the latter.

Limitations due to data set characteristics are discussed in [5]. Specifically the authors analyze the effect that characteristics such as occlusion, aspect ratio and object size has on the models predicting ability. In conclusion false positives and localization errors are very dependent on such characteristics. The authors also recommend further research be put in analyzing the detection of small versus large objects, partly due to the difference in salient features in the image. In [4] further research is done on the detection of small objects. A data-set consisting of small objects (relative to image) was created and an extension of the R-CNN framework was created. Moreover the bias of the PASCAL VOC data-set is discussed - how the samples may be synthetic in the sense that viewpoints, occlusion and similar classes in the same image are not frequent. This may bias the model and result in performance limitations when implemented on a machine. This is briefly elaborated on in section 6.

3 Data

The two datasets used for object detection in this project are *The Pascal Visual Object Classes (VOC) Challenge* [9] and *Microsoft COCO: Common Objects in Context* [18]. The COCO dataset is downloaded from [14] and consists of 120k images including 80 classes. VOC dataset consists of 40k images including 20 classes, downloaded from [1].

Data in COCO and VOC have different characteristics. Distinguishing attributes are size and number of classes, but also in the amount of objects per image, relative size and location of objects in image [18]. The effect of these attributes on network performance and learning is investigated in this project. As the amount of time and resources are limited, we have restricted ourselves to subsets of the full datasets.

3.1 Preprocessing

Before training, input data is preprocessed to improve generalization of trained networks. To begin with, all images are resized to 416x416 with padding if needed. Additional preprocessing done on the training data with some randomness include: Rotating input image, added gaussian noise to input image, horizontal flip, convert image to grayscale and blur input image.

3.2 Filtering

To be able to train and compare performance between networks trained on different datasets they had to be filtered in several ways. These include

- Reducing the number of samples such that the training time is reasonable while mitigating under-fitting.
- Balancing the data-set, i.e reducing over-representation of classes in the training data.
- Create subsets categorized according to relative size of object in the image.
  - Specific for ex. 2&3: Not VOC, but only objects and labels that can also be found in VOC. Only contain either big or small objects.

In the end there was 3 different datasets for experiment 1, where the first of these three was based on VOC, and the latter two on COCO. Then there was 3 for experiment 2, and 6 for experiment 3, all based on COCO.

4 Method

4.1 What is YOLO?

Convolutional Neural Networks (CNNs) are the norm for classification and localization problems in computer vision. YOLO (**Y**ou **O**nly **L**ook **O**nce) is a CNN based method but what makes it stand out from its contemporaries such as R-CNN is its speed of making predictions and the accompanying accuracy. This being a key reason for its popularity in real time applications. Generally, CNN based methods requires input images to be processed multiple times with focus shifting from one region to another within the image. On the other hand, YOLO makes the predictions by simply passing the image once through its FCNN (Fully Convectional Neural Network) and the result is in the form of a bounding box around the detected class. Now, since YOLO considers the whole image while making the classifications, it keeps the global context of the image into account, which could be lost when detections are made by concentrating on one section of the image at a time [17].

4.2 YOLO Architecture

A significant change from its predecessors is that YOLOv3 employs Darknet-53 CNN. The network is built such that it consists of 53 layers made up of repeating consecutive 3×3 and 1×1 convolutions layers, along with skip connections and up-sampling. This is shown in Figure 1 in Appendix A.1.

The task for YOLO architecture is two fold, Feature extraction and Feature detection. To accomplish this, it calls for stacking of the previously mentioned 53 layers for extraction, with another 53 layers for detection. As a result, the final architecture consists of 106 layers. One of the reasons for having this many layers is in response to the problems associated with previous versions of YOLO in detecting and classifying small objects [7].

The full architecture of YOLOv3 is graphically described by the Figure in A.2. However, due to the limit on the number of pages it is not possible to delve into each and every layer of this structure.

4.2.1 Multi-Scale Detections

To make better predictions about a variety of object sizes in the image, the YOLOv3 model adopts a multi-scale detection scheme. This means that out of all layers, the detections are made three times at layer 82, 94 and 106 at different scales. However since the detections are considered for multiple scales the features should also be extracted at varying scales. To clarify, the goal of a feature extractor is to perform convolutions with a kernel in order to produce a feature map. These feature maps are generally a more concentrated representation of the input image. For example the extracted features could be curves or edges in the image.

Consider the network shown in Figure 1 in A.1, with an example input image of size 416×416. Here feature extraction and detection are made at different scales. From the Figure, the 52×52 feature vector will be in relation to small objects, 26×26 for medium size objects and finally, the most broad feature map 13×13 for large objects. How these different feature map sizes help in detecting objects of different sizes should be apparent. A higher resolution allows to capture finer details in the image and the reverse is applicable to a coarse grid size.

4.2.2 Predictions

With object detection, we are looking for outputs that contain the location of the object in the image and its class label. Therefore the predictions are in the form of bounding boxes represented by the entities in (1). Here (t_x, t_y) are not explicit locations of the detected items, rather these are values relative to the top left corner of the image (c_x, c_y). Similarly, the output (t_w, t_h) are not the exact width and height of the bounding box. Instead these dimensions need to be transformed by applying a log space transform and multiplied by the dimensions of a anchor box. Before we discuss what an anchor box is, it is convenient to show how we get the final bounding boxes of interest. The transformations are shown in (2). Here σ stands for the Sigmoid function, to map the values between 0 and 1 and p_w and p_h are dimensions of the anchor box.

$$bb = [(t_x, t_y, t_w, t_h), (p_o), (p_{c_1}, p_{c_2}, ...)] \quad (1)$$

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \quad (2)$$

Previously, it was mentioned that the feature maps at different scales were used to make predictions. How these translate to predictions is facilitated by the fact that, in YOLO the size of the prediction maps is the same as the before-mentioned feature maps. Therefore, one can imagine the prediction maps as a grid dividing the image into equal parts and the size of the grid varies. Consequently each grid cell is then responsible for providing three bounding boxes (predictions) with the attributes shown in (1). Accordingly, the total number of predictions after taking each scale into account is: $(52 \times 52 + 26 \times 26 + 13 \times 13) \times 3 = 10647$. As you can imagine, these are way too many predictions and therefore we need to use methods to filter the results to finally get the best classification and localization.

4.2.3 Anchor Boxes

The anchor boxes (also known as prior boxes) are utilized such that they have predefined dimensions even before training. The anchor boxes are usually made to suit the ground truth. In YOLOv3, each grid cell has 3 anchor boxes (equal to number of bounding boxes predicted per cell). It might seem counter intuitive to have our predictions relative to these predefined boxes, but direct prediction of bounding box dimensions leads to unstable gradients during training [7].

4.2.4 Filtering Predictions

When listing the attributes of a bounding box in (1), the entities p_o and $(p_{c_1}, p_{c_2}, ...)$ were left un-mentioned. These represent the objectness score and individual class probabilities respectively. According to [16] and [17], p_o is essentially the confidence we have if a bounding box contains an object and $(p_{c_1}, p_{c_2}, ...)$ represents the probabilities for certain classes being present in the bounding box.

The filtering is now explained in steps.

1. Out of the three bounding boxes predicted per cell, we select the ones which has highest $\text{IOU}_{\text{pred}}^{\text{truth}}$ (overlap between ground truth and prediction).
2. Calculate the objectness score with $Pr(\text{Object}) * IOU_{\text{pred}}^{\text{truth}}$.
   - With this, now we know how sure a box is about an object being inside it. Therefore, we can remove those boxes which are less confident. Subsequently, although reduced, we will still end up with multiple overlapping bounding boxes in the object region.
3. Non-maximum suppression is performed based on IOU to pick the best prediction.

The Figure in A.3, visually demonstrates the above filtering steps.

5 Experiments

The YOLOv3 network architecture is implemented in PyTorch [15]. To confirm the implementation, we overfitted the network on a few samples from the VOC dataset. After a successful implementation different experiments were conducted to investigate YOLOv3 performance and points listed in section 1.1.

5.1 YOLOv3's measures of precision

In this section we use 4 different measures of precision: mean average precision (mAP), object accuracy, class accuracy and no object accuracy. An intuitive understanding of these is presented below where every measure can take a value between 0-1:

- mAP: Is 1 if the predicted bounding box's location, width and height is exactly according to the label. In the report mAP refer to mAP_{50}, where 50 denotes the IOU threshold 50%.
- Object accuracy: Is 1 if the network identifies every labelled object in the picture.
- Class accuracy: Is 1 if every object that is detected is predicted to be of the correct class.
- No object accuracy: Is 1 if the network never predicts an object where there is none.

5.2 Experiment 1: YOLOv3 trained and tested on different datasets

To investigate performance of YOLOv3 we trained three networks, referred to as COCO-1, COCO-2 and VOC-1. The networks were trained to detect two types of objects: people and cars. Table 1 shows information about the subsets that each network was trained on.

Table 1: Information about the dataset for each trained network

Network	VOC-1	COCO-1	COCO-2
Set	1	2	3
Dataset	VOC	COCO	COCO
Num images	4302	4500	1200
Num objects: people	5764	23970	6515
Num objects: cars	4468	16461	4408

From now on, set 1 and 2 in table 1 is referred to as *filtered* VOC and *filtered* COCO respectively.

COCO-1 was trained for 710 epochs, corresponding to 100 hours. VOC-1 was trained for 400 epochs, 40 hours and COCO-2 was trained for 535 epochs, 20 hours. All three networks reached their individual peak in performance after 200-300 epochs of training, further training didn't yield any increase in performance.

5.2.1 Results

Appendix B.1 shows performance on filtered COCO by our trained networks. It also displays benchmark performances on the COCO dataset by pre-trained YOLOv3 networks and other popular object detection models. Appendix B.2 is similar but with performances on filtered VOC and full VOC dataset.

Best performance on filtered VOC is achieved by VOC-1 with mAP = 0.609. Just ahead of COCO-1 with mAP = 0.591. The benchmark performance on the VOC dataset used is mAP = 0.782, achieved by the pre-trained network VOC-78 in appendix B.2. Appendix C.2 shows an example of VOC-1 detecting people and cars in images from VOC dataset.

Best performance on filtered COCO dataset was COCO-1 with mAP = 0.383. Where a benchmark performance on the COCO dataset is mAP = 0.553 by YOLOv3-416 [2]. Appendix C shows COCO-1 detecting people and cars in COCO images.

5.3 Experiment 2: YOLOv3 ability to detect smaller sized objects

In section 3 in [17] it is mentioned that YOLOv3's performance on detecting smaller sized objects has gotten better in comparison to it's predecessors, but it is also mentioned that further investigation of this is needed as no quantitative data regarding this is presented. This experiment is a consequence of that. It was decided that the best representation of YOLOv3 would be a pre-trained network, and not a locally trained one. This was to make sure that the local implementation and training was not limiting the results that was going to represent YOLOv3 overall. With VOC-78 and this data a fair representation on YOLOv3's ability to detect smaller sized objects would be achieved. When these datasets were classified with the VOC-78 net it would hence give fair measures on how well it performed on the different sized objects. The results are shown in appendix E and is summarized in following section.

5.3.1 Results

The networks' ability to detect smaller objects is shown via the "object accuracy" measure, and from the results of the experiments one can deduce that the object accuracy goes up as object size goes up and that VOC-78 does not classify smaller objects as good as it classifies big ones. So does every other measure besides "No object accuracy" that remains close to 1 on all datasets.

5.4 Experiment 3: COCO-1 and VOC-1's ability to detect smaller sized objects

In experiment 3 we analyze and verify that the locally trained models COCO-1 and VOC-1 show a similar behaviour as the VOC-78 model. This also further expands on evaluating YOLOv3 ability to detect smaller sized objects. The same philosophy as in experiment 2 lies behind why the data separation is fair in this also, with the only difference that one of these networks had been trained on COCO data. This is taken into account when analysing the results. Measures from VOC-78 are added as a reference. The key results are shown in appendix E.2, and is summarized in following section.

5.4.1 Results

VOC-1's ability to detect smaller sized objects follows the same trend as VOC-78, that the object accuracy goes up as object size goes up and it does not classify smaller objects as good as it classifies big ones. When it comes to COCO-1, the performance drops as the object sizes go up.

6 Conclusions

Object detection is a fundamental problem in computer vision. A problem where the size of objects acts as a performance bottleneck [8], a notion that is strengthened by the results in section 5.

From appendix B.2 and appendix C.2, we can see that the YOLOv3 network VOC-1 achieves satisfactory performance on VOC images after having trained for only 20 hours. However, looking at object detection and it's applications in industry and real world, the COCO dataset acts as a better representative of real world applications out of the two. Quantitative comparisons between the two datasets is depicted in appendix D and D.2. A fundamental problem in object detection is detecting smaller objects, since smaller objects are highly dependent on contextual information [18]. Something COCO aims at rectifying by a collection of images with objects in their natural context, rich in contextual information. E.g when the object to detect is sunglasses. The contextual information "on a person" may be important for the model to accurately classify sunglasses. This is made even more difficult with multiple small objects in each scene, where the network may find it hard find a balance between capturing semantically strong features and retaining contextual information [8].

In the end, COCO-1 was overall the best model. In experiment 1 it performs just as well as VOC-1 on VOC images, whilst being far superior on COCO images. Likely because it has been trained on a richer COCO dataset, as seen in appendix D.2. VOC-1 has probably captured the features of people and cars well judging by it's results on filtered VOC, but misses much of the spatial and contextual information required to accurately classify COCO images. In experiment 3, appendix E.2, COCO-1 performance drops when tested on datasets only containing big objects. Just like the author mentions at the end of section 3 in the YOLOv3 paper [17]. This is assumed to be a consequence of the data COCO-1 was trained on, which consist mostly of small objects. Further results from experiment 2 and 3 solidifies that in general, YOLOv3 does perform better on big objects. But also that performance on small objects can be improved with appropriate training data.

Overall, our YOLOv3 network COCO-1 manages to classify people and cars in everyday situations at a pretty good level. But as expected, it's still a fair bit behind the top performing YOLOv3 networks out there due to less training data and restricted resources.

References

[1] https://www.kaggle.com/aladdinpersson/pascal-voc-dataset-used-in-yolov3-video.

[2] Yolo: Real-time object detection, https://pjreddie.com/darknet/yolo/.

[3] Yolov3 in pytorch, https://github.com/aladdinpersson/machine-learning-collection/tree/master/ml/pytorch/object_detection/yolov3.

[4] Chenyi Chen, Ming-Yu Liu, Oncel Tuzel, and Jianxiong Xiao. R-cnn for small object detection. In Shang-Hong Lai, Vincent Lepetit, Ko Nishino, and Yoichi Sato, editors, *Computer Vision – ACCV 2016*, pages 214–230, Cham, 2017. Springer International Publishing.

[5] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. Diagnosing error in object detectors. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 340–353, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[6] Ali Farhadi Joseph Redmon. Yolo9000: Better, faster, stronger. 2016.

[7] Ayoosh Kathuria. How to implement a yolo (v3) object detector from scratch in pytorch, https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/.

[8] Lin Ma Kai Mu Yonghong Tian Kui Fu, Jia Li. Intrinsic relationship reasoning for small object detection. 2020.

[9] C. K. I. Williams .J Winn A.Zisserman M. Everingham, S. M. Ali Eslami · L. Van Gool. The pascal visual object classes challenge: A retrospective. 2014.

[10] Manogna Mantripragada. Digging deep into yolo v3 - a hands-on guide part 1, https://towardsdatascience.com/digging-deep-into-yolo-v3-a-hands-on-guide-part-1-78681f2c7e29.

[11] Qi-Chao Mao, Hong-Mei Sun, Yan-Bo Liu, and Rui-Sheng Jia. Mini-yolov3: Real-time object detector for embedded applications. *IEEE Access*, 7:133529–133538, 2019.

[12] Society of Automotive Engineers (SAE). Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. 2021.

[13] Rosina De Palma. Yolov3 architecture: Best model in object detection, https://bestinau.com.au/yolov3-architecture-best-model-in-object-detection/.

[14] Aladdin Persson. https://www.kaggle.com/dataset/79abcc2659dc745fddfba1864438afb2fac3fabaa5f37daa8a51e36466db10

[15] Alladin Persson. Yolov3 from scratch.

[16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

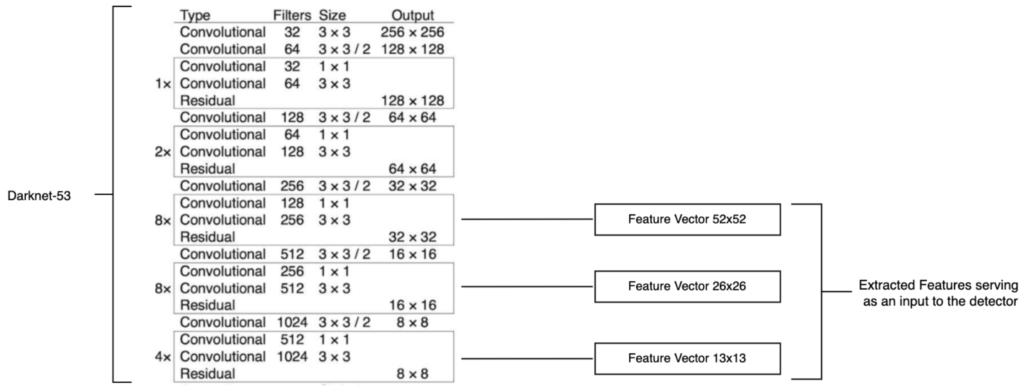
[17] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[18] S. Belongie L. Bourdev R. Girshick J. Hays P. Perona D. Ramanan C. L. Zitnick P. Dollar T-Y. Lin, M. Maire. Microsoft coco: Common objects in context. 2015.

Appendices

A

A.1 YOLOv3 Architecture



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8

Figure 1: Darknet-53 (taken from [10], [17])

A.2 YOLOv3 Architecture

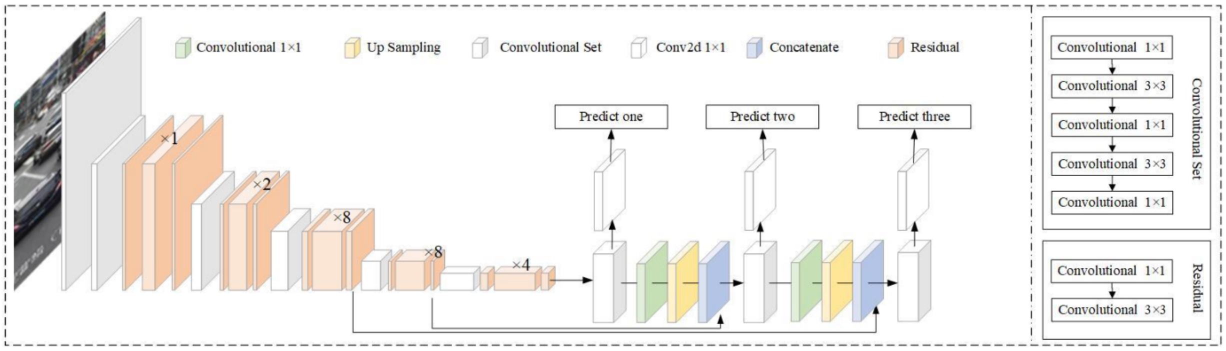


Figure 2: YOLOv3 Full Architecture (taken from [11])

A.3 Filtering Predictions

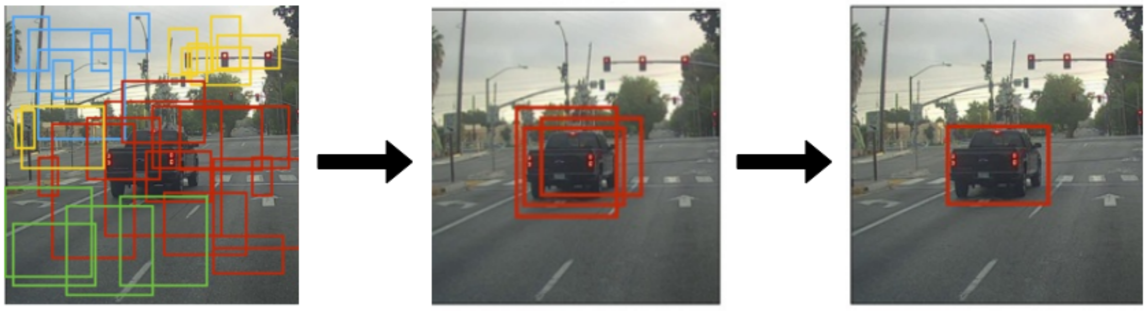


Figure 3: Filtering Bounding Box Predictions (adopted from [13]).The first image from the left, depicts the numerous bounding box predictions, the second image is a result of removing the boxes which do not contain an object, and finally, the third image is a result of non-maximum suppression on second image.

B Experiment 1: Performance on COCO and VOC datasets

B.1

Table 2: Performance on COCO dataset [2] [17]

Filtered COCO dataset				
Model	mAP_{50}	obj acc	class acc	no obj acc
COCO-1 @290epochs	**0.383**	**0.483**	**0.895**	0.993
COCO-1 @710epochs	0.372	0.432	0.888	0.996
VOC-1 @400epochs	0.244	0.332	0.799	0.996
VOC-1 @200epochs	0.238	0.318	0.801	0.995
COCO-2 @535epochs	0.232	0.259	0.827	0.998
COCO-2 @235epochs	0.226	0.289	0.837	0.997
COCO dataset				
Model	mAP_{50}	obj acc	class acc	no obj acc
YOLOv3-320	0.515	-	-	-
YOLOv3-416	0.553	-	-	-
YOLOv3-608	**0.579**	-	-	-
YOLOv2 608x608	0.481	-	-	-
Retinanet-101-800	0.575	-	-	-
Faster R-CNN+++	0.557	-	-	-

B.2

Table 3: Performance on Pascal VOC dataset [6] [3]

Filtered VOC dataset				
Model	mAP_{50}	obj acc	class acc	no obj acc
VOC-1 @200epochs	**0.609**	**0.601**	**0.970**	0.995
VOC-1 @400epochs	0.592	0.559	0.976	0.997
COCO-1 @290epochs	0.591	0.589	0.947	0.994
COCO-1 @710epochs	0.579	0.468	0.948	0.998
COCO-2 @535epochs	0.360	0.302	0.902	0.998
COCO-2 @235epochs	0.323	0.303	0.910	0.998
VOC dataset				
Model	mAP_{50}	obj acc	class acc	no obj acc
VOC-78	**0.781**	-	-	-
YOLOv2 544	0.738	-	-	-
Fast R-CNN	0.684	-	-	-
ResNet	0.738	-	-	-
Faster R-CNN	0.704	-	-	-
SSD512	0.749	-	-	-

C Trained network images

C.1 VOC-1

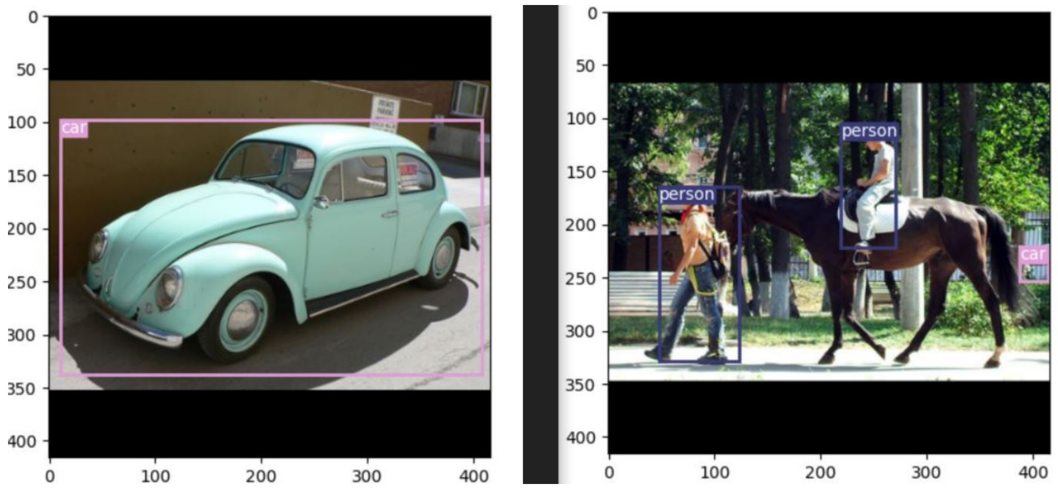


Figure 4: VOC-1 detecting people and cars in VOC images.

C.2 COCO-1

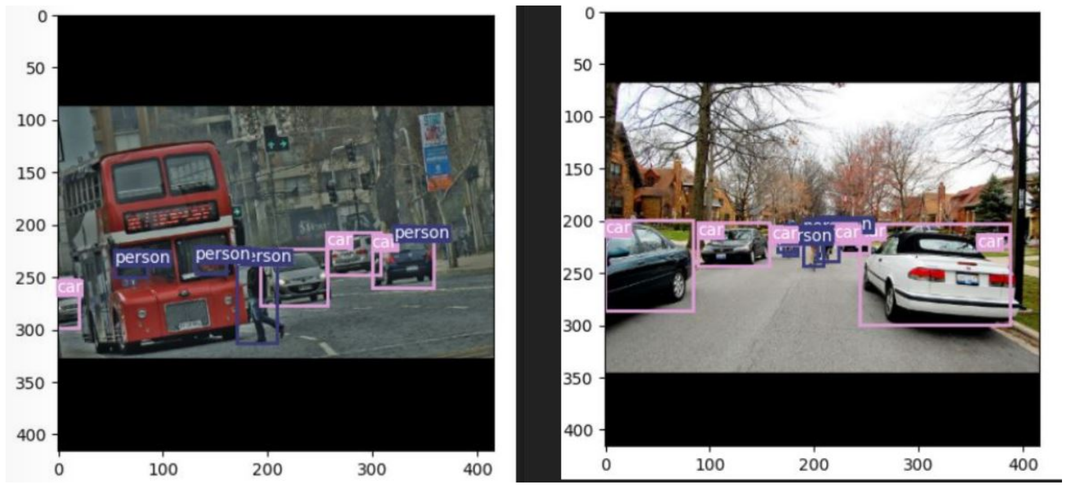


Figure 5: COCO-1 detecting people and cars in COCO images.

D Quantitative analysis of COCO and VOC

D.1

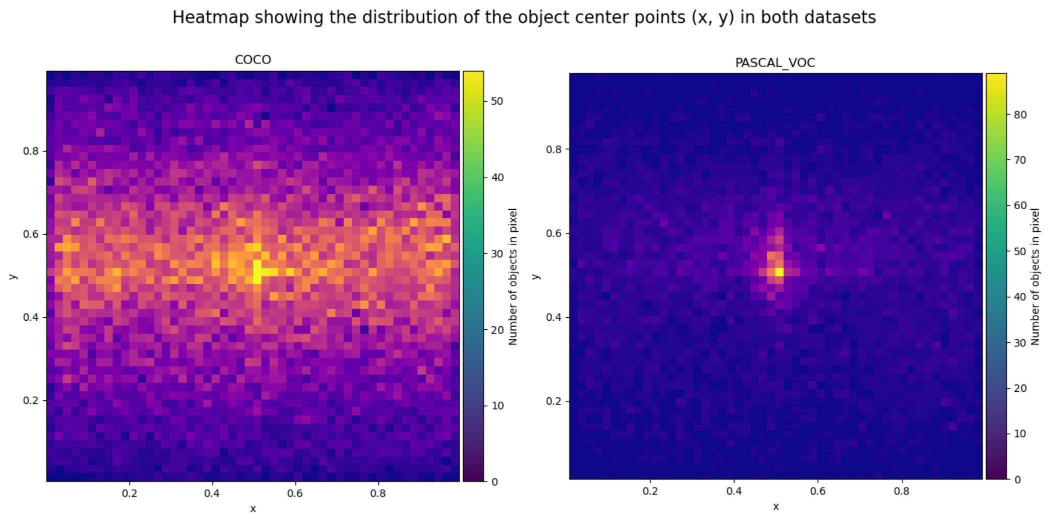


Figure 6: Distribution of object center points in *filtered* datasets.

D.2

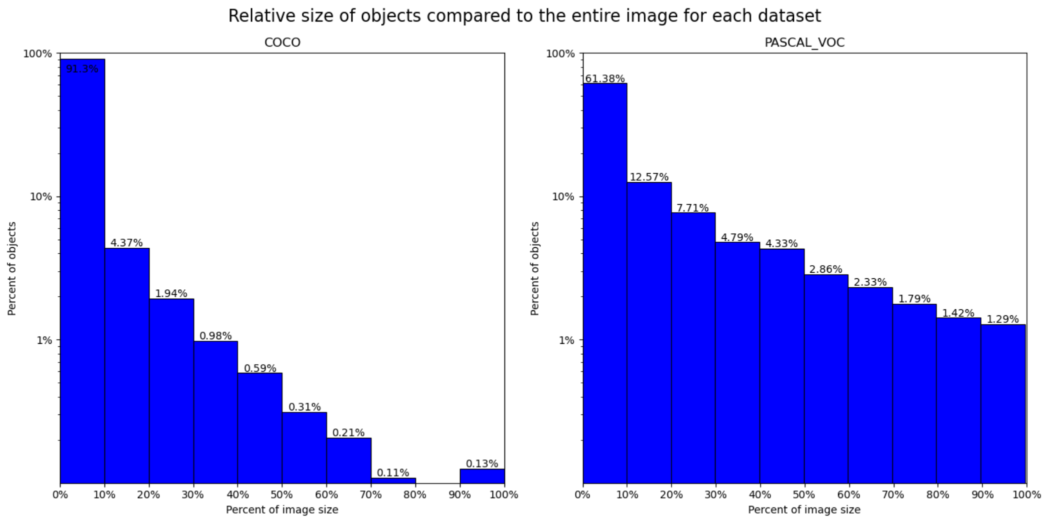


Figure 7: Relative size of objects compared to entire image for each *filtered* dataset.

E Experiment 2 and 3: Results

E.1

Table 4: Result from experiment 2

Dataset	Class accuracy	No object accuracy	Object accuracy	mAP_{50}
small1	0,848	0,998	0,453	0,332
small6	0,857	0,997	0,562	0,474
big50	0,909	0,997	0,858	0,728

E.2

Table 5: Result from experiment 3

VOC-1				
Dataset	Class accuracy	No object accuracy	Object accuracy	mAP_{50}
small1_pc	0,697	0,998	0,260	0,097
small6_pc	0,821	0,997	0,361	0,256
small10_pc	0,786	0,998	0,331	0,278
big30_pc	0,917	0,997	0,583	0,666
big40_pc	1	0,996	0,733	0,768
big50_pc	1	0,997	0,556	0,461
COCO-1				
Dataset	Class accuracy	No object accuracy	Object accuracy	mAP_{50}
small1_pc	0,697	0,998	0,437	0,253
small6_pc	0,866	0,997	0,494	0,400
small10_pc	0,855	0,997	0,441	0,374
big30_pc	0,875	0,997	0,417	0,370
big40_pc	0,8	0,996	0,533	0,280
big50_pc	0,667	0,996	0,333	0,111
VOC-78				
Dataset	Class accuracy	No object accuracy	Object accuracy	mAP_{50}
small1_pc	0,874	0,999	0,437	0,374
small6_pc	0,919	0,998	0,526	0,493
small10_pc	0,917	0,997	0,507	0,474
big30_pc	1	0,995	0,917	0,925
big40_pc	1	0,995	0,933	0,963
big50_pc	1	0,993	1	0,903

F Datasets used in experiments

Experiment 1				
Dataset	Number of images	Object size threshold	Classes	Distribution of classes
Filtered VOC	4302	None	Person, car	56/44
Filtered COCO 1	4500	None	Person, car	59/41
Filtered COCO 2	1200	None	Person, car	60/40
Experiment 2				
Dataset	Number of images	Object size threshold	Classes	
small1	676	<0,01	$\{VOC\} \bigcup \{COCO\}$	-
small6	2868	<0,06	$\{VOC\} \bigcup \{COCO\}$	-
big50	1005	>0,50	$\{VOC\} \bigcup \{COCO\}$	-
Experiment 3				
Dataset	Number of images	Object size threshold	Classes	
small1_pc	131	<0,01	Person, car	-
small6_pc	281	<0,06	Person, car	-
small10_pc	330	<0,10	Person, car	-
big30_pc	81	>0,30	Person, car	-
big40_pc	59	>0,40	Person, car	-
big50_pc	40	>0,50	Person, car	-