Multiple Object Detection on COCO 2017 Dataset

Areeb Asad Syed, Wonsuk Jang, Jiaxi Lei,

Spencer Sheen, Rohitkumar Arasanipalai, Zheng Yang, Tianyi Zhou

Abstract

In this work, we explore different approaches to the multiple object detection problem in image processing. Inspired by the architectures of YOLOv3 (You Look Only Once) and Mixture Density Object Detector (MDOD), we tried modeling two different convolutional neural networks which can generate labeled bounding boxes for all detected objects on the image on the COCO 2017 dataset. From our experiment with YOLOv3, we found out that although it was able to recognize each distinct image well, it sometimes suffered from being unable to classify multiple objects due to having bounding boxes with lower confidence disappear due to non-max suppression. In the case of MDOD, we find that although MDOD specifies well for background features, it struggles with class imbalance. For both YOLOv3 and MDOD, we were able to implement the models with satisfactory performance, but we believe we can further improve it in future works by training with more classes of data and training our model longer on more powerful platforms. The GitHub repository for our project can be found here.

1 Introduction

With the advent of computer vision, researchers increasingly try to come up with the best models to help with object detection in given images. One such task that researchers try to work on is multiple object detection. In multiple object detection, a model tries to detect as many images as it recognizes on the image and tries to also output a bounding box around the image that it recognizes and assigns it a label. Therefore, unlike standard object detection, an image can have multiple bounding boxes around different objects with different labels. In our project, we attempt to implement two simplified versions of the state-of-the-art models for multiple object detection called YOLOv3 and Mixture Density Object Detector, which are created through carefully designed convolutional neural networks. Unlike other models, these models require much more finely designed training processes and loss functions, not only accounting for the prediction scores and labels, but also the accuracy of the bounding box that captures the object we are attempting to detect. In our implementation, rather than focus on both images and videos to detect objects, we decided to focus on images in our implementation. However, these models can be used on both images and videos to detect multiple real-time moving objects in the real world, allowing for object detection in real-time.

2 Motivation

One of the most amazing facts about human brain is its image processing speed: it helps us localize distinct objects in the environment and further classify them into what we think they are all day long, based on nothing more than a few glances. Computer vision has already outperformed human vision in trivial tasks like image classification since the start of 21st century, but when it comes to object detection, data scientists struggled to beat human brain in terms of speed and accuracy, until the birth of YOLO in 2016 [1]. Real-time object detection has been growing rapidly since

then, with improved models like YOLOv2[2], YOLO9000[2] and YOLOv3[3], as well as other implementations such as Mixture Density Object Detector [4].

In contrast to image captioning for our previous project in which we generate captions based on the given image without a "real" ground truth, in object detection, we seek to distinguish each object in the image and assign a class label to it. Generally, most object detection models approach this problem by predicting bounding boxes around valid objects in the image. After generating multiple predictions of bounding boxes for an image, it uses non-max suppression in order to find the box that is most likely to capture the object the best and removes the other boxes in order to make its prediction, similar to how humans mind filters overlapping bounding boxes around an object to the bounding box that best captures the object.

In this work, we mainly explore two different object detection models: YOLOv3 and Mixture Density Object Detector. Using simplified versions of both models, we present how they manage to localize and classify multiple objects in a single image with high efficiency and accuracy.

3 Related Works

3.1 YOLO

In a 2016 paper "You Only Look Once: Unified, Real-Time Object Detection" [1], J. Redmon et al. presented the YOLO model, which is a novel approach to the object detection problem. Despite not being the most accurate algorithm, YOLO's single network architecture made object detection faster than ever before, and it became a state-of-the-art real-time detection system.

Later that year, J. Redmon and A. Farhadi introduced the improved model YOLOv2 [2]. They utilized many tricks in the computer vision field, including batch normalization, anchor boxes, etc., making YOLOv2 more accurate and even faster than the original YOLO. They also proposed YOLO9000, a model trained using joint training techniques which can perform object detection for more than 9000 different classes.

In 2018, J. Redmon and A. Farhadi made significant changes to their model structure, applying a deeper backbone network with residual connections to perform cross-scale predictions [3]. The new YOLOv3 model has further improvements in detection accuracy with the cost of slightly lower speed, and it outperforms other detection methods in terms of speed/accuracy tradeoff.

3.2 Mixture Density Object Detector

The first research paper we looked at when approaching the task of object detection using a Mixture Density Object Detector, was "Training Multi-Object Detector by Estimating Bounding Box Distribution for Input Image", Jaeyoung Yoo et al. [4]. The problem addressed in this paper is of how we can get a network to learn multiple configurations of bounding boxes for object in multiple input images. It discusses past related works that are too complex when they train the detection network by directly assigning the ground truth bounding boxes to specific locations in the networks output. Here, they re-frame the object detection problem as a task of density estimation of the bounding boxes. A mixture model is used to estimate probability density of bounding boxes in the input images to train the network. The research proposes a new architecture for this task: Mixture Density Object Detector (MDOD), along with a function for training using density estimation.

Another research we looked at before starting our implementation was "Deep Mixture Density Network for Probabilistic Object Detection" by Yihui He and Jianren Wang [5]. This research addresses common issues that come up in object detection tasks. Specifically, bounding boxes of occluded objects in images can have many, sensible configurations. This paper proposes probabilistic object detection in images using a deep, multivariate mixture of Gaussian models. This model is able to learn border relationships and different configurations of occluded objects in images.

4 Dataset

For our dataset, we are using COCO 2017 [6]. The COCO dataset is a rich dataset composed of 91 different objects ranging from cats to refrigerators. For our project, as we were doing a supervised

learning task of multiple object detection, we decided to use images that were pre-labeled by COCO. We decided to use COCO for our dataset because out of the other open dataset with bounding boxes online, we found COCO to be of most variety and have the best bounding boxes to work with. In order to load the dataset, we used FiftyOne's Dataloader on COCO to build our dataloaders and trained the model on images of our selected labels rather than train on the entire data due to the limitations of our computing power. For the sake of our project, we decided to primarily focus on images of dimensions of 640 X 480 consisting of primarily dogs, cats, giraffes, and horses along with other objects to perform our object detection in order to make our computation feasible with our limited computing power.

5 Methods

5.1 YOLOv3 Model

5.1.1 Architecture

Instead of previous methods using re-purposed classifiers or sliding windows, YOLO approached the object detection problem by creating a singular neural network that trains on the full images and making predictions directly from the images to create class probabilities and bounding boxes of objects in the images [1]. As shown in Figure 1, the system first resizes the image, takes it through the convolutional neural network, and outputs bounding boxes on the image based on thresholds of the models confidence and class probabilities.

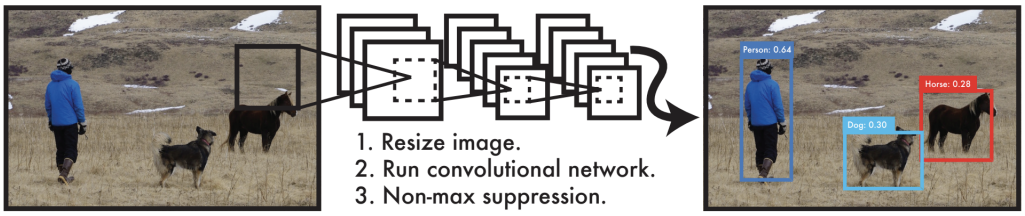

Figure 1: Overview of YOLO Detection System [1]

The YOLO model architecture is shown in Figure 2. It is made up of 24 convolutional layers, followed by 2 fully connected layers. This model uses 1×1 reduction layers followed by 3×3 covolutional layers. The network outputs predictions as a $7 \times 7 \times 30$ tensor.

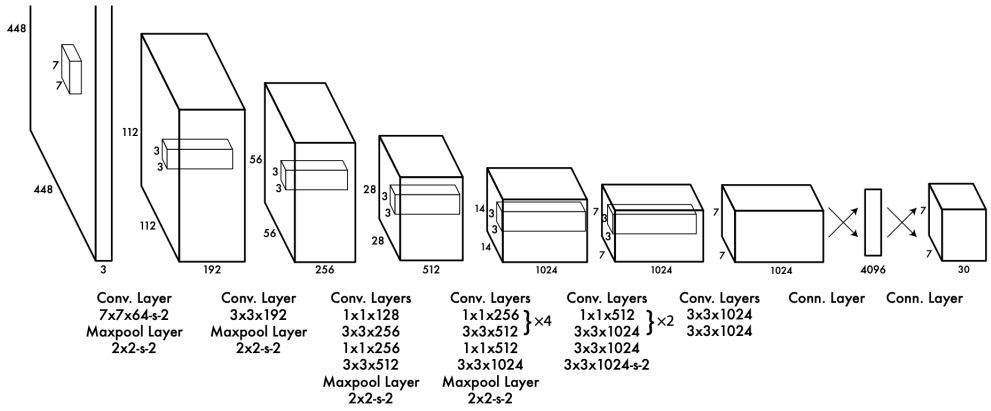

Figure 2: YOLO Network Architecture [1]

The architecture we implement and use in this project is an improved version of the YOLO model known as YOLOv3 [3]. YOLOv3 is a bigger network, but improves significantly on accuracy from

previous versions. The architecture of the YOLOv3 network used for feature extraction is a combination of the networks from YOLOv2 [2], Darknet-19, and residual networks. This architecture has a total of 53 convolutional layers, and is thus known as Darknet-53. This network alternates between 3×3 and 1×1 convolutional layers and has some shortcut connections. The network architecture is detailed further in the Figure 3 below.

	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
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

Figure 3: YOLOv3 Darknet-53 Network Architecture [3]

5.1.2 Bounding Box

Each predicted bounding box can be described by 5 parameters: b_x, b_y, b_w, b_h, and confidence. (b_x, b_y) represents the position of the box center in the image, b_w and b_h are the box width and height, and confidence accounts for the probability that the predicted box contains an object and the accuracy of the box, which is defined as follows.

$$\text{confidence} = P(\text{object}) \cdot \text{IOU}$$

Using these parameters, it becomes possible for neural networks to learn to predict a certain number of bounding boxes from an input image. However, learning to predict the box position and geometry directly turns out to be impractical, due to the large variance of their distributions. To solve this problem, YOLOv3 uses an alternative approach. It divides each image into an $N \times N$ grid and generates 3 sets of prior box geometries (p_w, p_h) for each grid, which are determined using k-means clustering. The network then learns to predict 4 relative coordinates t_x, t_y, t_w and t_h for each bounding box, which are related to the original parameters as follows:

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

Where (c_x, c_y) is the position of the top-left cell corner as shown in Figure 4, and $\sigma(z) = [1 + \exp(-z)]^{-1}$ is the sigmoid activation function.

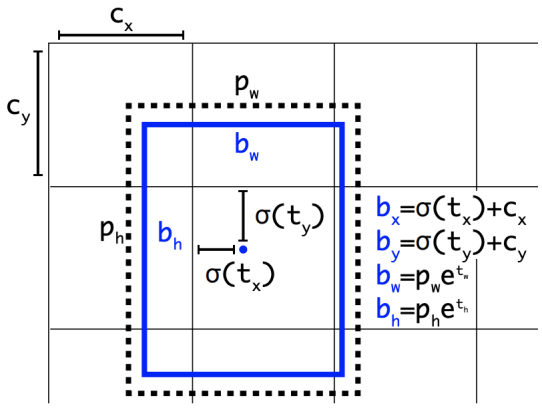

Figure 4: Bounding boxes with geometry priors and position prediction. The position of the box center is predicted using a sigmoid function, according to the location of filter application. The width and height of bounding boxes are predicted as adjusted by the cell offsets [2].

The procedure described above constraints each grid to predict local bounding boxes, and the change of variables rescales all parameters, which stabilizes training and makes learning easier.

For the confidence term, YOLOv3 also uses a different expression t_o, which is related to the original confidence by the following equation:

$$\text{confidence} = P(\text{object}) \cdot \text{IOU} = \sigma(t_o)$$

To determine the class label for each bounding box, the network also needs to generate a class probability map, i.e. predict all class probabilities for each grid, and then computes class confidence for each predicted bounding box as follows:

$$\text{class confidence} = P(\text{class}|\text{object}) \cdot P(\text{object}) \cdot \text{IOU} = P(\text{class}|\text{object}) \cdot \text{confidence}$$

Finally, the box coordinates, confidence and class confidences are used to generate the detection outputs. The process of bounding box prediction and generation is illustrated in Figure 5.

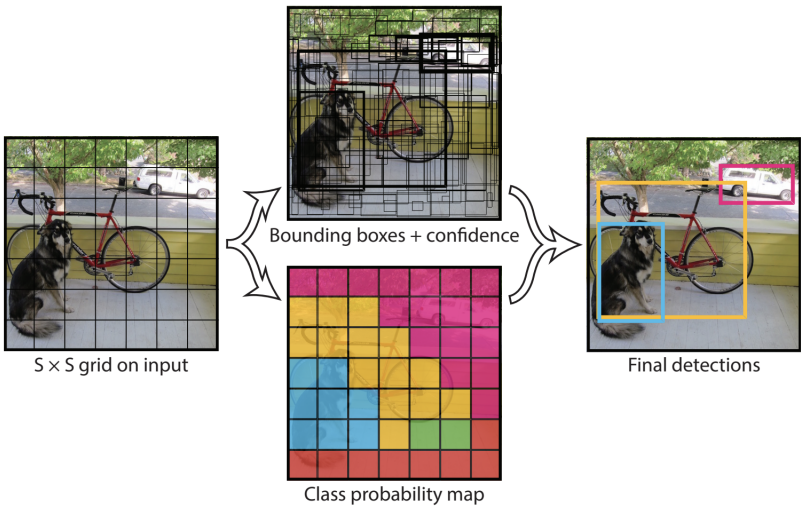

Figure 5: Bounding Box Mechanism [1]

5.1.3 Loss

The loss for our YOLOv3 model is separated into 3 parts: the bounding box loss, the objectness loss, and the class loss. The bounding box loss is the average of one minus Pixel IoU of all predicted bounding box/ all ground-true bounding box regardless of classified class. The objectness loss is applying focal loss/ cross entropy loss on model output confidence score and the Pixel IoU for each pair of model prediction and ground truth. The class loss is applying focal loss/ cross entropy loss on model output label and the one hot encoded ground truth label.

For each prediction, there will be a few offset bounding boxes generated from the ground truth bounding box by YOLO anchoring, let these be t_{bbox}. The losses are defined as follows:

$$L_{bbox} = \sum_{i=1}^{N} \text{mean}(1 - \text{IOU}(p^i_{bbox}, t^i_{bbox}))$$

$$L_{obj} = \sum_{i=1}^{N} \text{loss}(p^i_{\text{confidence}}, \text{IOU}(p^i_{bbox}, t^i_{bbox}))$$

$$L_{cls} = \sum_{i=1}^{N} \text{loss}(p^i_{cls}, t^i_{cls})$$

The final loss we use for optimizing our model will be:

$$L = N(g_{bbox} L_{bbox} + g_{obj} L_{obj} + g_{cls} L_{cls})$$

Where N is the batch-size, $g_{bbox}, g_{obj}, g_{cls}$ are the gains of each loss type.

For baseline model:

g_{bbox}	g_{obj}	g_{cls}
0.05	1	0.5

This implementation of loss is adopted from Ultralytic's implementation of YOLOv3 object detector. [8] Some key differences here between this implementation and the original loss discussed in YOLO paper includes the available option of focal loss and some slight implementation difference on determining the bounding box and objectness.

Following the YOLOv3 paper, the cross entropy loss we use here is binary cross entropy loss for two reasons: one obvious reason being classification datasets may have overlapping labels (for example a dog can also be labeled as an animal). Using softmax paired with multi-class cross entropy results in a false assumption that each prediction has only one ground truth label, which is not the case. [3] Further more, this enables the implementation of focal loss to be directly wrapped around the existing binary cross entropy loss.

Focal Loss The focal loss is defined as follows:

$$\mathrm{FL}\left(p_{\mathrm{t}}\right)=-\alpha_{\mathrm{t}}\left(1-p_{\mathrm{t}}\right)^{\gamma}\log\left(p_{\mathrm{t}}\right)$$

For object detection, there will be some hard examples and easy examples. In that case, even if we use the balanced cross entropy loss

$$\mathrm{CE}\left(p_{\mathrm{t}}\right)=-\alpha_{\mathrm{t}}\log\left(p_{\mathrm{t}}\right)$$

The easily classified data will occupy the loss and the model will focus on these easily classified data and ignore the hard classified examples, which are more crucial when we training the model.

5.2 Mixture Density Object Detector

One of the main challenges in multi-object detection is learning a variable number of bounding boxes on each image. Certain images will have many features that need to be detected while some only have a couple notable features in it. Previous methods (including YOLO) use anchor boxes compare ground truth boxes with predicted boxes which is then put through an objective function for training (left side of the figure).

Mixture Density Object Detection aims to simplify multi-object detection by predicting the bounding box as probability distribution for the predicted area and the individual category (right side of the figure).

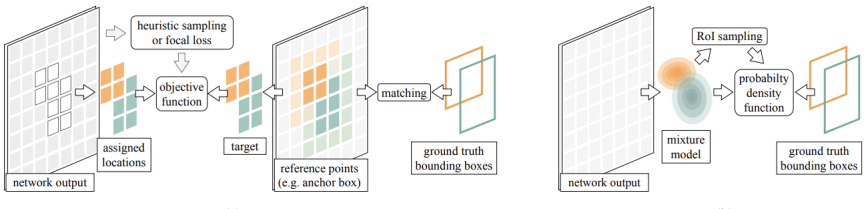

Figure 6: Detection Process for Mixture Density Object Detector[4]

5.2.1 Mixture Model

The Mixture Model is split into 2 parts. The probability for the bounding box and the probability the particular box belongs to a certain class. The bounding box probabilities are expressed using a Cauchy distribution, and the class probabilities are expressed using a categorical distribution.

5.2.2 Network Architecture

The image convolution layer or feature pyramid generates output sizes based on different scales using Resnet-50 as a pretrianed model. These features are then combined to form network outputs with width and height values based on the convolution sizes.

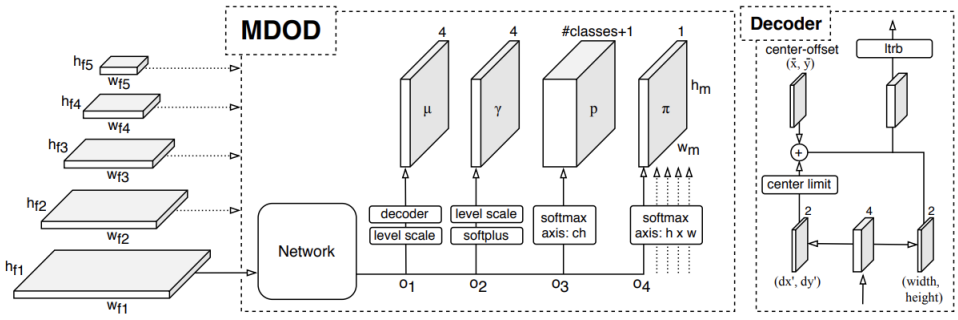

Figure 7: MDOD Architecture [4]

The location (μ), scale(γ), class probability (p), and mixing coefficient (π) maps are formed based on their own activation and scaling functions.

We altered the model structure slightly to reduce the sizes generated by the feature pyramid. To fit that corresponding change, the 3x3 convolution layer in MDOD model was switched to a 4x4 convolution layer with the stride value increased to 2. We named the smaller structured model Mini-MDOD.

5.2.3 Loss

The loss function for MDOD is separated into two sections: bounding box loss and class detection loss. To begin with, the bounding box loss is measured as as the negative likelihood of the Cauchy mixture.

$$\mathcal{L}_{MoC} = -\frac{1}{N_{gt}} \sum_{i=1}^{N_{gt}} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{F}(b_{gt,p}^{i}; \mu_k, \gamma_k) \right)$$

On the other hand, the class detection loss is measured as a negative log likelihood of sampled background bounding boxes. The sampled background bounding boxes are generated to combat the forward-backward problem.

$$\mathcal{L}_{MM} = -\frac{1}{N_{roi}} \sum_{j=1}^{N_{roi}} \log p(b_{roi}^{j} | \text{image})$$

The final loss function then becomes the sum of these two loss values along with a scaled constant α which scales the losses. For the implementation of the model, we followed the original paper's value of $\alpha = 2$.

$$\mathcal{L} = \mathcal{L}_{MoC} + \alpha \mathcal{L}_{MM}$$

5.2.4 Code Citation

To try recreating the implementation of Mixture Density Object Detector, we based our implementation on the following Github repository in https://github.com/yoojy31/MDOD[9].

6 Results

6.1 YOLOv3 Results

6.1.1 Initial Training

For our best model, we initially trained our model for 1000 epochs with a batch size of 8, image size of 640 X 480, initial learning rate of 0.01, momentum of 0.937, bounding box loss gain of 0.05, and set the classification loss gain to 0.5. Additionally, for non-max suppression, we use 0.7 for the confidence threshold and 0.1 for the IoU threshold.

Figure 8: Positive results generated from YOLOv3

Figure 8 shows the good results generated from the Yolov3 Model. Figure 9 shows one of the weak results. The right image shows the bounding boxes predicted before the non-maximum suppression and the left image is the output of the non-max suppression (NMS).

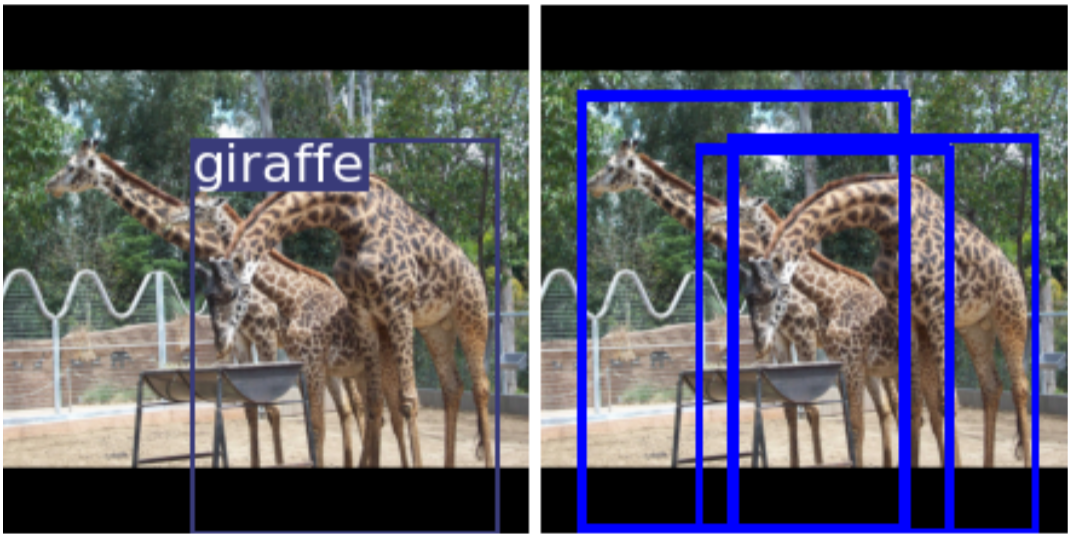

Figure 9: Negative results generated from YOLOv3. Bounding box before NMS (right). Bounding box after NMS (left)

6.1.2 Further Training with More Images

In order to get better results, we tried training longer with more images given our computational limitations and were able to train our YOLOv3 model with 256 images for 1000 epochs and with the same hyperparameters as our initial training, which roughly took about 8 hours on Datahub. Some samples of positive and negative results can be seen below.

Figure 10: Positive results with Further Training

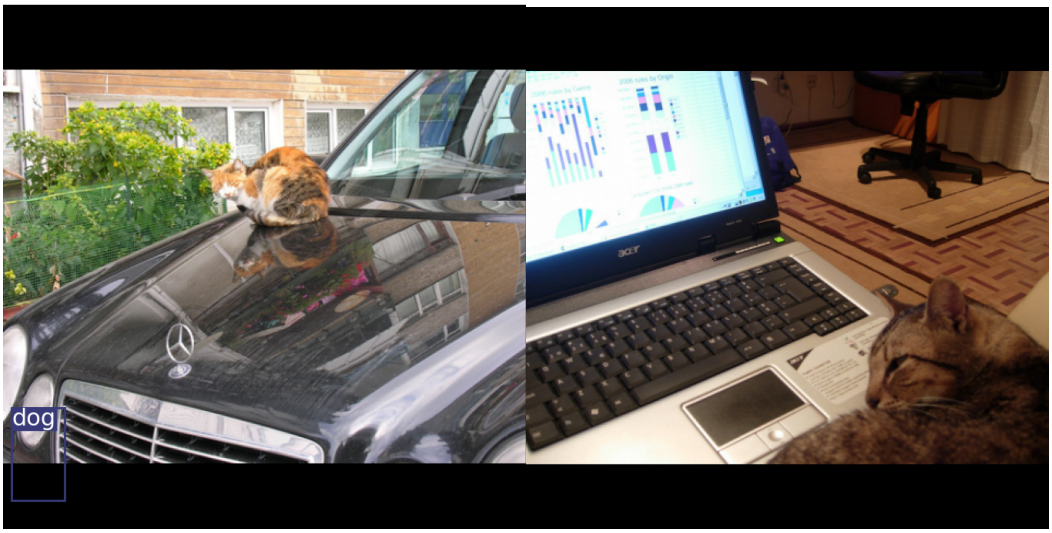

Figure 11: Negative results with Further Training

6.2 MDOD Results

6.2.1 2 Class Results

To begin with, we tried generating 2 Class Results by comparing cats and dogs. There were only 500 training images and 12 testing images, so we were able to train for 160 epochs which matched the default parameters. The model was trained on a learning rate of 0.005, momentum of 0.9, and weight decay of 0.00005. There were 2 workers used with a batch size of 8. Some sample outputs are shown in Figure 11.

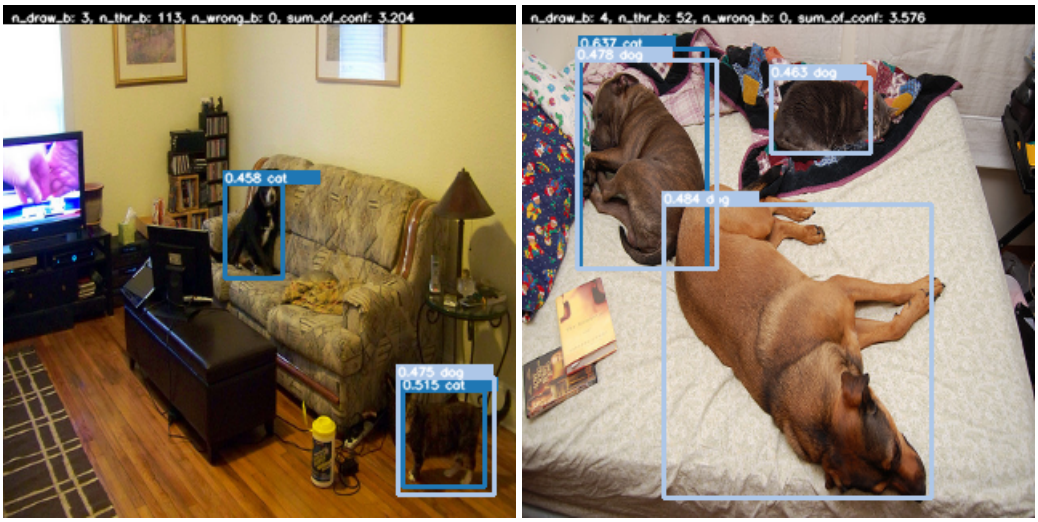

Figure 12: Mini-MDOD Test Outputs over 2 Classes

6.2.2 80 Class Results

Mini-MDOD was trained on 5 epochs with a learning rate of 0.005 and momentum of 0.9. It was trained with 2 workers and a batch size of 8. We were not able to run with more workers or a higher batch size on Datahub due to its limitations. Some strong test outputs are shown in Figure 8 and some weak outputs are showin in Figure 9.

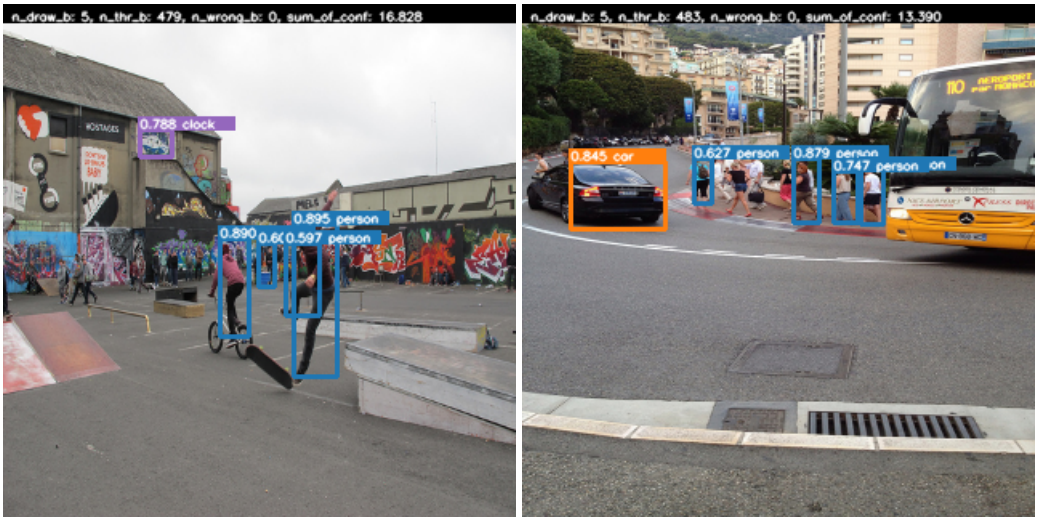

Figure 13: Mini-MDOD Strong Test Outputs over 80 Classes

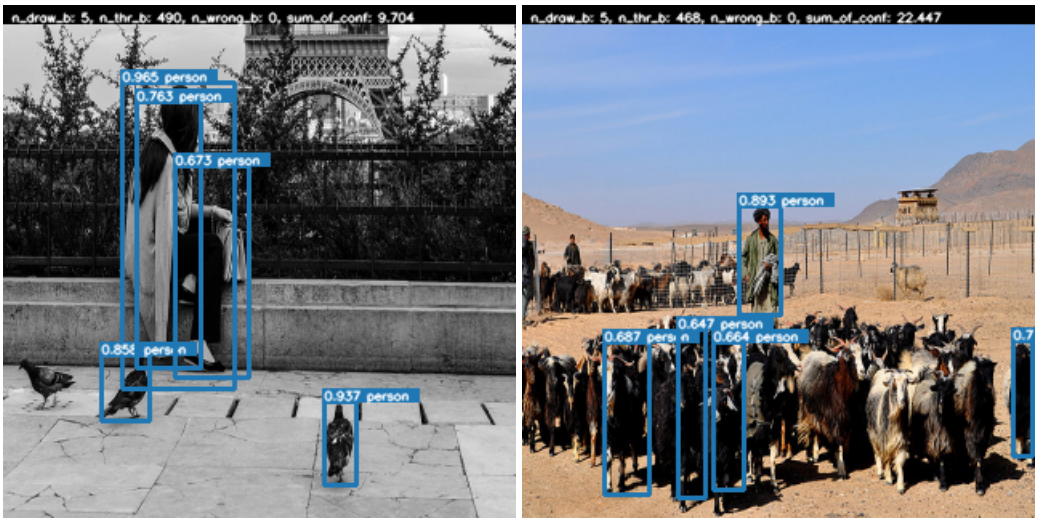

Figure 14: Mini-MDOD Weak Test Outputs over 80 Classes

Likewise, the precision graphs are shown below in Figure 10.

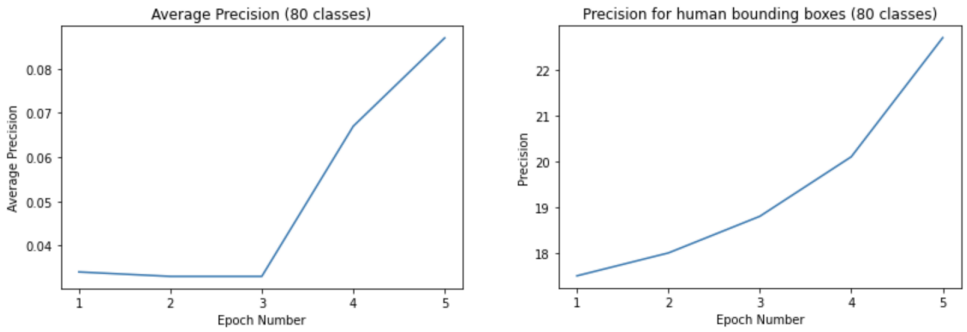

Figure 15: Mini-MDOD average precision and human precision (precision of only human bounding boxes)

7 Discussion

7.1 YOLOv3 Discussion

7.1.1 Initial Training

After plotting the bounding boxes for positive results, we can see that YOLOv3 does well for the objects which have large size and not overlapped by others. We see that the model is able to recognize and correctly identify the objects from the images.

On the other hand, from figure 9, we see that one instance of a bad result is when not all the objects of the same type are detected. This is due to the fact that the bounding boxes predicted from the model are merged after the non-max-suppression (NMS). In non-max-suppression, the bounding box with the highest confidence dominates overlapping bounding boxes with lower confidence. In our experiment, we used a high confidence threshold of .7. As only one bounding box of the giraffe was able to exceed the confidence threshold of .7, the remaining bounding boxes for other giraffes are removed. However, simply decreasing the confidence interval will not work because it will simply allow other bounding boxes to remain, potentially creating more objects to be detected than there are image at incorrect locations. Therefore, it is important for us to set a high confidence and low IoU threshold to the NMS function in order to prevent the bounding box redundance for

the same object. Therefore, an example of our model making a mistake can be when there are some overlapping between the same-class bounding boxes with low confidence, causing the NMS to potentially merge multiple objects into a single object.

7.1.2 Further Training with More Images

We attempted to train with more images for our further testing. Although we tried training with 80 classes also, we noticed that our model was not able to generalize well because we could only train for smaller sets of images due to the computing power of Datahub. Therefore, we decided to train for more epochs with smaller number of classes but with more samples. In fact, the original YOLOv3 model took couple days with many GPUs in order to achieve its state-of-the-art performance. In our case, if we increased the number of classes or even the number of samples, our model was not able to train for the required number of epochs in the given time, thus making the model harder to generalize. Therefore, we tested with a smaller subset of classes and samples and found out that although our model was able to still get positive results despite having a bit of trouble generalizing due to us not being able to train enough samples for enough epochs. As shown in Figure 11, we see that our model was sometimes not able to generate bounding boxes due to having low confidence and even incorrectly predicting odd areas in the image as one of the selected labels. These are issues that are caused due to our model not yet being ready to generalize, which is again attributed to our limited training. Thus, even though our model was not able to achieve the state-of-the-art results by the paper due to lack of training time, computational power, and bigger dataset, our model was able to reasonably perform well, showing us how it can achieve better performance by simply having a platform where it can run more samples for more epochs to allow for better generalization.

7.2 MDOD Discussion

7.2.1 2 Class Results

Despite training for 160 epochs, the 2 Class results share the same strengths and weaknesses as the 80 Class results as we will detail later below. Although the bounding boxes for each cat and dog are often found successfully, the bounding boxes do not distinguish between dogs and cats very well. This can be explained by the lack of training data (only 500 images of cats or dogs) and that the loss function prioritizes finding bounding boxes over classifying bounding boxes, which is again illustrated in our results with 80 classes.

7.2.2 80 Class Results

The strong sample outputs show that bounding boxes can be generated accurately, even if features are within close proximity to each other. The people features can be identified clearly, even if cropped by another object. Our simplified MDOD implementation also excels at identifying background features. The sampling background features in MDOD allows the model to detect key features in the background when testing.

However, MDOD does seem to have a significant class-imbalance problem. This is partially because the training and validation have a large number of "person" features compared to the other 79 classes. The weak images show that other animals (bird, goat, etc) still get identified as a person. This implies that MDOD can do well identifying features in general, but the classification itself can be improved. The class-imbalance is further shown through the average precision of each class. Frequently trained classes such as "person", "truck", and "giraffe" have high precision rates, while other classes are significantly lower.

The precision graphs over 5 epochs also tell a similar story. The overall average precision is significantly lower than human bounding box precision. This is largely because there are more human bounding boxes (both in foreground and background) to train compared to other classes such as a cow or goat. It is promising that both average precision and human class precision increase over 5 epochs, and with more computing power to train more epochs the precision will improve. Again, in order to improve our our MDOD model even further, we believe that being able to train for more epochs on bigger datasets in a stronger platform will help us achieve results close to the paper.

8 Conclusion and Future Work

In our project, we have successfully implemented the simplified versions of two state-of-the-art models for multiple object detection for a single image called YOLOv3 and MDOD. In our implementation, we simplified the existing architectures and ran the data on our selected labels of dogs and cats along with other objects in the COCO 2017 dataset in order to run our model. Although our results were not as good as the state-of-the-art models as expected due to us running on simplified versions of the models, we were still able to perform multiple object image detection on our images as shown in our results with the bounding boxes with the correct labels on the images for many of our training data. From our experiments, we found out that in order to achieve state-of-the-art performances like the papers were able to, we would need to have a much stronger platform than Datahub with multiple GPUs in order to train for more epochs with more samples in order for our models to better generalize.

For future work, we would like to not only have the opportunity to run our code on platforms with stronger computation, but we would also like to add more depth to the models rather than simplify it to allow for better performance. One idea we had in mind was changing the first feed forward to a layer with attention units to better pay attention to each distinct object. In addition, we also want to try running our model on video data in order to tackle the problem of real-time detection. Ultimately, we were able to successfully implement the simplified versions of YOLOv3 and MDOD to perform multi object detection given a single image.

References

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

[2] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[3] Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." arXiv preprint arXiv: 1804.02767 (2018).

[4] Yoo, Jaeyoung, et al. "Training Multi-Object Detector by Estimating Bounding Box Distribution for Input Image." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021.

[5] He, Yihui, and Jianren Wang. "Deep mixture density network for probabilistic object detection." 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020.

[6] Lin, Tsung-Yi, et al. "Microsoft coco: Common objects in context." European conference on computer vision. Springer, Cham, 2014.

[7] Kathuria, Ayoosh. "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] "Ultralytics YOLOv3." `https://github.com/ultralytics/yolov3`.

[9] "yoojy31 MDOD." `https://github.com/yoojy31/MDOD`.