Real-time multiple object detection and tracking

Sahar Siddiqui ss12414@nyu.edu
Priyank Pathak pp1953@nyu.edu
Ilya Kostrikov kostrikov@cs.nyu.edu
Rob Fergus fergus@cs.nyu.edu

Computer Vision Course Project
Courant Institute of Mathematics
New York University

Abstract

The project deals with a real time object detector and tracker, which estimates the object at a distance using a kinect-based camera feed. We aimed to create a real-time model since it is supposed to be deployed to a robot so that it could move in the direction of the object and based on the distance, adjust its position to pick up the object. We used the *RetinaNet model* for object detection which in turn makes use of ResNet50 in the backend. Furthermore, tracking is deployed to ensure object tracking while computations are being done in the background. The camera feed is supposed to be around 30 frames and our pipeline using object detection single-handedly can produce **38 frames/sec**. After training the model on CLEVR dataset, we fine-tuned it on real life mobile videos to remove the dependencies on the camera angle and increase robustness against background noise and dynamically changing camera angles.

1 Introduction

Our problem deals with the detection and tracking of simple geometrical shapes that can be found and controlled in real life. We propose an object detector and tracker for a robotic arm to track and pick up the object of desired shape which would further use reinforcement learning to adjust its position via distance calculation based on the detection and tracker results obtained on kinect feed. To accomplish this, we made use of the CLEVR [1] dataset which closely resembles with the geometric shaped toys available to us. The major advantage of this datatset is that it synthetically creates realistic 3D shapes along with their shadows using Blender, hence mimicking a realistic setting. The model is trained on common shapes like cylinder, cube and sphere. However, in regards to our problem statement, there some fundamental issues with the dataset that we address in future sections (**4.1**).

Since the proposed method intends to provide distance to the robot, we are dependent on the kinect camera feed given by its 4th channel to take depth map as an input. Once we have the object detected, we can apply segmentation on the bounding box patch and compute average distance for the same. Apart from all these computations, the model is supposed to be real-time, hence all the computations need to be done in the background in nearly constant time. Hence, we include a tracker in our pipeline, to balance the time taken by segmentation process. The tracker deployed is CSRT tracker [2] which is one of the fastest tracking algorithms. The tracker is variant of the initial bounding boxes, so it's not possible to completely depend upon the tracker. Hence, we propose a two-stage pipeline which includes simultaneous object detection and tracking on the video frames resulting in real time object tracker.

After experimenting with all the known fast detectors, we settled on RetinaNet [3], a model which we found to be the easiest among all to generalize for custom datasets apart from VOC Pascal [4] and COCO Dataset [5].

We sub-divided our task into the following subparts :

1. Using blender to create CLEVR dataset along with bounding boxes.
2. Object Detection using fast object detectors
3. Making the system robust to background noise and different camera angles
4. Including object tracker in the pipeline to increase the speed for the output computation.
5. Using the tracker to create new dataset taken from mobile to finetune the model in real life environment.

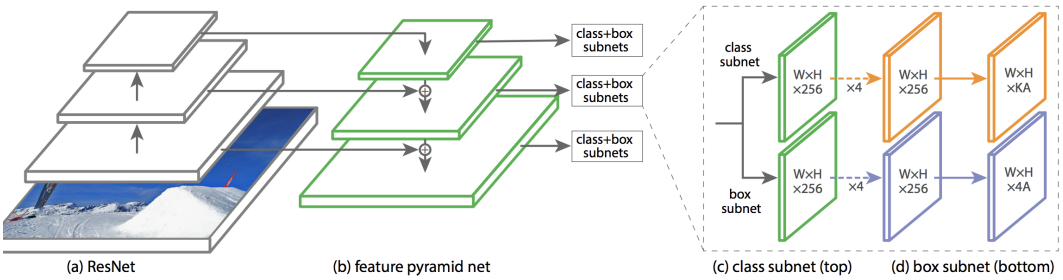


Figure 1: The one-stage RetinaNet network architecture taken from [3]

2 Related Work

As the name suggests, multiple object tracking consists of keeping track of objects in a video as they move around, i.e. we need to localise and identify objects in each frame in the video and these identified objects should be consistently labelled across all the frames. Challenges with the consistent labelling arise when the objects get temporarily occluded or move out of the frame for a brief moment.

Different types of models exist for different scenarios. One basic distinction is that of *offline* vs *online* models. An online model receives a video on a frame to frame basis and has to output the result for each frame on the go. On the other hand, an offline model has access to the entire video beforehand which means that frames from past as well as future can be used to devise the result. This ultimately boils down to an optimization problem which has been solved using linear programming and k-shortest path optimization methods. For real-time tracking, however, only online models can be used as we don't have access to future frames at any point of time.

Researches conducted on object tracking systems until 2006 [6] focused on finding good features, object representations and motion models for improvement of tracking ability. However, these approaches were highly restricted by the constraints like static camera [7], high contrast objects with little occlusion in videos [6]. Yilmaz *et al.* [6] identified these assumptions to be greatly limiting when it comes to real-world application, stating that while said assumptions increased the performance, they made the models too specific.

Convolutional Neural Networks (CNNs) have been greatly outperforming in the task of image classification. Three datasets commonly used for benchmarking are ImageNet [8], Pascal VOC [4] and MS COCO [5]. The best performing models on these datasets make use of CNNs indicating that the detection capability today has far surpassed that of 2006. This in turn leads us to the idea of two-stage tracking wherein we make use of an object detector coupled with algorithmic tracker to achieve the task of real-time object detection and tracking.

The dominant paradigm in modern object detection is based on a two-stage approach which is used in R-CNN [9]. R-CNN was further improved over the years in terms of speed and learning capabilities which has given rise to Faster R-CNN [10]. Another paradigm is that of one-stage detectors like SSD [11] and YOLO [12]. These detectors have been tuned for speed but their accuracy trails that of two-stage methods.

Lin *et al.* introduced the RetinaNet which is able to match the speed of previous one-stage detectors while surpassing the accuracy of all existing state-of-the-art two-stage detectors.

3 Model

3.1 RetinaNet

RetinaNet is a one stage real time detector where it surpasses all the state-of-the-art object detectors. We opted for the keras pre-existing library[1] for our implementation, where we used ResNet-50 pre-trained weights on COCO dataset as initials and further trained on CLEVR dataset. The architecture of RetinaNet is heavily inspired from the architectures of YOLO, SSD and tends to surpass them by making use of Focal Loss which helps eliminate class imbalance.

Cross entropy (CE) loss for binary classifica-

[1] https://github.com/fizyr/keras-retinanet

tion= $log(p_t)$, where

$$p_t = \begin{cases} p, & \text{if } y = 1 \\ 1 - p, & \text{otherwise} \end{cases} \quad (1)$$

The equation for Focal Loss is given as follows:

$$FL(p_t) = -\alpha^t (1 - p_t)^\gamma log(p_t). \quad (2)$$

3.1.1 Architecture

RetinaNet is a single, unified network composed of a backbone network and two task-specific subnetworks. The architecture is shown in the figure 1.

1. **Feature Pyramid Network Backbone**: Convolutional network with a top-down pathway and lateral connections so the network efficiently constructs a rich, multi-scale feature pyramid from a single resolution. Each level of the pyramid can be used for detecting objects at a different scale
2. **Translation-invariant anchor boxes**: On each pyramid level. As each anchor is assigned to at most one object box, we set the corresponding entry.
3. **Classification Subnet**: The classification subnet predicts the probability of object presence at each spatial position for each of the A anchors and K object classes.
4. **Box Regression Subnet**: In parallel with the object classification subnet, another small FCN is attached to each pyramid level for the purpose of regressing the offset from each anchor box to a nearby ground-truth object.

3.2 CSRT Tracker

Short-term tracking is an open and challenging problem for which discriminative correlation filters (DCF) have shown excellent performance. Lukežič *et al.* [2] introduced the channel and spatial reliability concepts to DCF tracking. It uses only 2 standard features (HoGs and Colornames). Although, it operates at a comparatively lower fps (25 fps) but gives higher accuracy for object tracking. The implementation has been taken from OpenCV[2]. The CSRT tracker alone at work can be seen in figure 2.

4 Datasets

In order to train and fine-tune the model for object detection, we have used two datasets:

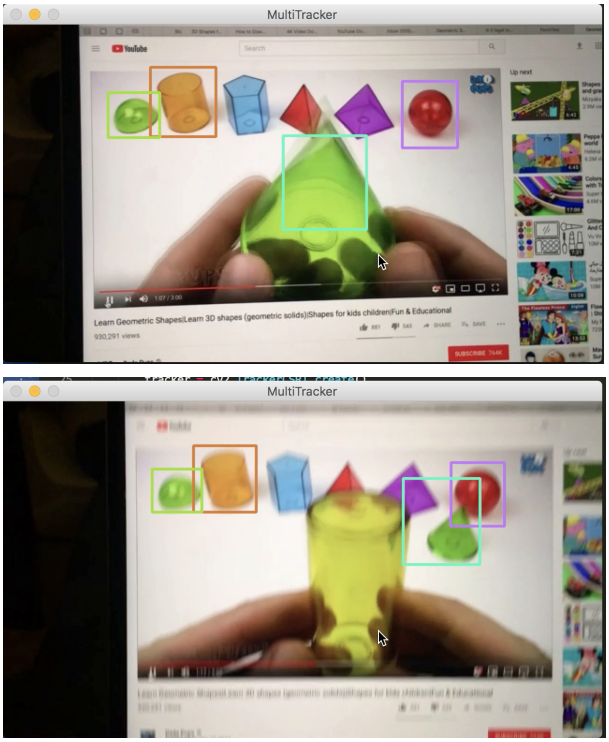


Figure 2: CSRT working on a YouTube video (https://tinyurl.com/y7kxdsys).

– **CLEVR dataset**: Johnson *et al.* introduced the CLEVR Dataset, which consists of synthetically rendered images (using Blender) containing multiple objects of 3 different shapes (cube, sphere, and cylinder) that come in two absolute sizes (small and large), two materials (shiny metal and matte rubber), and eight colors. Since the original dataset does not provide any information on the bounding boxes of shapes, *we tweaked the code*[3] to generate the 12753 images from scratch to get the 2D bounding boxes for every object in the frame.

– **Real-time mobile videos**: In order to further fine-tune the model on the actual 3D objects to be picked up by the kinect, we curated a dataset of 1075 frames from two mobile video recordings of the objects. We further performed a semi-automated annotation to get the bounding boxes of the geometrical shaped objects in every frame of the videos. This was done by manually annotating every tenth frame and the deploying the tracker to get the bounding boxes for the next 10 frames.

[2]https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/

[3]https://tinyurl.com/ybn9l2zd

4.1 Problems with the CLEVR dataset

– Since the dataset is a 3D generated dataset created from a fixed camera angle, the model trained is highly camera angle dependent. A snapshot of the dataset is shown in the figure 3.

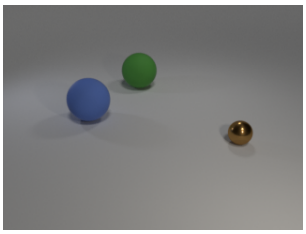


Figure 3: CLEVR Dataset snapshot.

– As one can notice in figure 4, the mere absence of background noise makes the model vulnerable to even the slightest noise and the model starts detecting bounding boxes on almost every noise it comes across. Even the plain background to infinity makes the dataset highly simulated, impractical to the real world challenges. An easy replication of this problem would be having an almost high angle vertical camera angle covering the entire table.

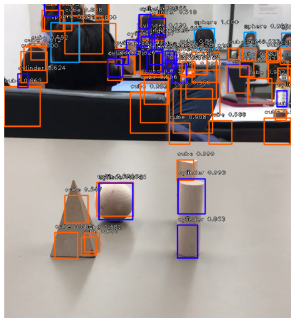


Figure 4: Cluttered output from the model because of the unseen background noise.

– The high camera angle forces the model to have a dependency on it, which can't account for the robot being far away from the object or close to it. While conducting practical experiments we found that, once the camera angle deviates from the proposed camera angle, the results distort badly and consequently the detector can't seem to recognize even the most obvious shapes.
– Absence of negative sampling is another problem. Since the CLEVR dataset consists of only 3 objects, our initially trained model never saw any object which its not supposed to detect and draw a bounding box for, resulting in an almost 100% accuracy on regression task. So even though the classification task is suffering the overall accuracy is very high.

5 Experimental setup

5.1 Negative Sampling on Dataset

1. Reduce the problem to a sub problem of detecting only 2 classes: Sphere and Cylinder, and treating the cube as a background noise.
2. The tracker tends to grow the number of bounding boxes over a period of time, hence the threshold count on the tracker is used to maintain the initial count. If the tracker exceeds initial count, object detection is invoked, else it is automatically invoked after every 10 frames.
3. The model is further fine-tuned over the mobile videos to improve the accuracy further. Note, the video recorded with an iPhone has a resolution quite different from what the model is trained on. Hence fine tuning doesnt improve the accuracy by quite an margin. Although the model seems to over-fit the fine-tune data set badly.
4. Fine tuning on real life videos also helps model dissolve the camera angle constraint, as the model is now able to detect (atleast) sphere correctly on most camera angles, although it still suffers from not being able to detect cylinder.

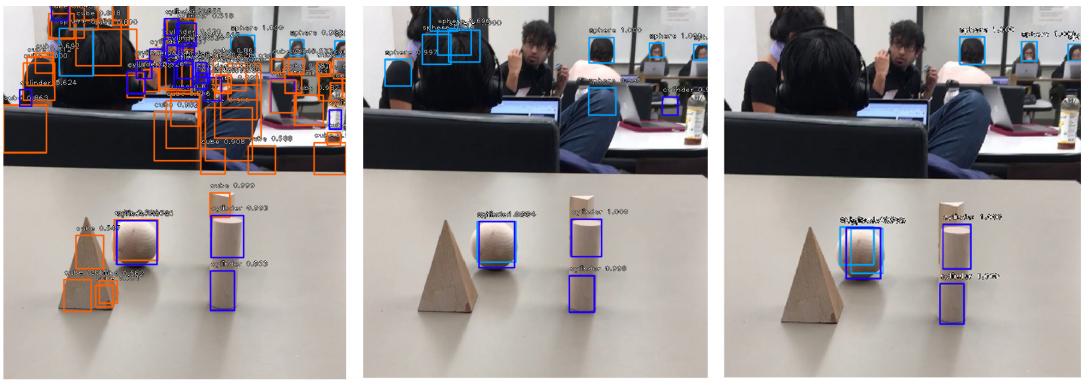
5.2 Specifications

Dataset is made compatible with **Keras dataset generator** script, which makes it compatible with the standard dataset like VOC Pascal and COCO Dataset.
The Model is trained on CLEVR Dataset for 50 epochs, and fine-tuned on the Mobile Video dataset for 100 epochs with a learning rate divided by a factor of 10 as used for training.

See table 1, for further details.

6 Results and Analysis

The model completely learns the fine-tuned mobile videos set, proving it overfits the training data. This proves that more data needs to be added. See the figure 5 for comparing the results of negative sampling and fine tuning.



a) b) c)

Figure 5: **(a)** shows the output of the model without fine tuned or negative sampling **(b)** is after the model has negative sampling, but without finetuning **(c)** After the model is trained with negative sampling and fine-tuned on manually crafted 1075 frames

Optimizer	Adam
Learning Rate	1e-4 (Train) 1e-5 (Fine-Tune)
Classification Loss	focal loss $(\alpha = 0.25, \gamma = 2.0)$
Regression Loss	smooth L1()

Table 1: Training specs

The figure 6 shows that the model over fits the fine tune data set and more fine tune dataset needs to be added.

Since the model has never seen a noisy background and shaky camera frames along with varying camera angles (which seems more appropriate for robustness in the real scenario), our fine-tuned model works far better than the model trained on only CLEVR dataset.

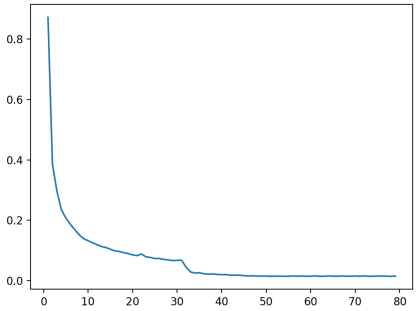


Figure 6: Graph of Training loss vs Number of epochs

The model has an approximate accuracy of almost 100% on the CLEVR dataset, confirming the lack of diversity in the dataset. While training on the dataset, the model saturates in about 6 epcohs, but accuracy increases further on validation set by a small margin. Our best model for CLEVR dataset comes out to be on **50th epoch**.

The model without negative sampling saturates on fine-tuning on about 7th epoch. Out of 100 epochs, the best model was reached at **7th epoch**.

The model with negative sampling never saturates and accuracy grows even after 100 epochs. For this experiment we ran it till **100 epochs** which is our best model.

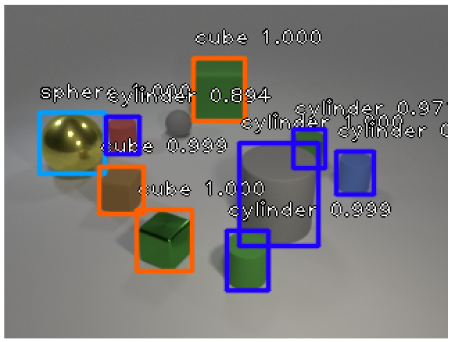


Figure 7: Almost 100% accuracy on CLEVR dataset

While validating it on the actual mobile videos, given feed at the rate 30 frames per second, our object detector was able to process 415 frames in 15.52, i.e. a reading rate of **26.73 frames/sec**. While reading the video stored on computer the

object detector read 628 frames in 16.4783 seconds, i.e. **38.1107 frames /sec**.

The above calculations are based on the object detector reading the frames, re-sizing it, prepossessing it and drawing all the bounding boxes on the image.

7 Conclusion

The system still lacks Kinect feed, thus we are at this point still awaiting a working kinect model. Once access is granted, we can combine all our pipeline based on data feed rate and computation time. Along with it, we need to incorporate pixel based segmentation or pixel-clustering to calculate the mean distance of bounding box from the camera. *(Since we are already using 50 layer DeepNet for calculating bounding boxes, we can't deploy more deep nets and will be prefer to use an algorithm based tracking.)*

As shown, the model completely learns the mobile video dataset completely. Thus, we would need to create more bounding boxes to further fine-tune the model. In order to completely remove the dependency from the camera angle constraint we are trying to modify the code for the Blender to re-generate the CLEVR dataset in order to produce/simulate data from various camera angle. Additionally, to train the model in more realistic setting, we are planning to incorporate the pybullet repository[4]. See figure 8.

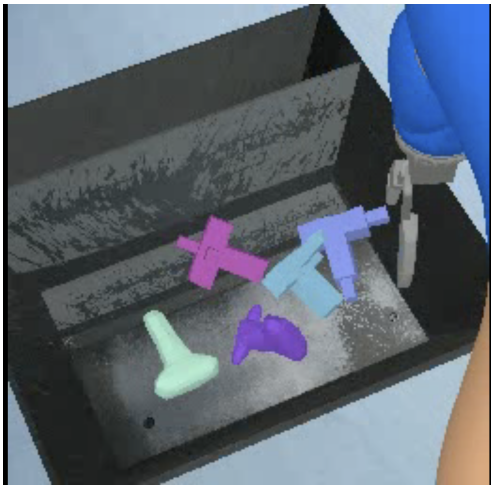


Figure 8: A snapshot what pybullet model data looks like

[4]https://tinyurl.com/y8ce45ly

References

[1] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Computer Vision and Pattern Recognition (CVPR), 2017*. IEEE.

[2] Alan Lukežič, Tom'aš Voj'iř, Luka Čehovin Zajc, Jiř'i Matas, and Matej Kristan. Discriminative correlation filter with channel and spatial reliability. In *CVPR*, 2017.

[3] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[4] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 2010.

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755, 2014.

[6] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006.

[7] Massimo Piccardi. Background subtraction techniques: a review. IEEE, 2004.

[8] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 2014.

[10] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. 2015.

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. 2016.

[12] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint*, 2017.