Real Time Object Detection for Visually Impaired People

Tushar Chand Kapoor
Department of Computing Science
Simon Fraser University
tkapoor@sfu.ca

Kashish Kohli
Department of Computing Science
Simon Fraser University
kashishk@sfu.ca

Mehak Parashar
Department of Computing Science
Simon Fraser University
mparasha@sfu.ca

Kanksha Masrani
Department of Computing Science
Simon Fraser University
kmasrani@sfu.ca

Manan Parasher
Department of Computing Science
Simon Fraser University
mparashe@sfu.ca

Abstract

This project presents an approach for object detection in real time which aims at expanding possibilities for the visually impaired people to achieve their full potential. The proposed approach uses state of the art YoloV3 network trained with our own modifications using keras on the darknet53 weights. We have used machine learning techniques such as SSD and classifier network VGG16 detector. Our system is used to find objects in the real world such as person, chairs or tables that are common in the scene. Objects detected from the scene are represented by their names and are converted to speech. Our main target was to detect multiple objects & expand the recognition task in parallel to keeping the processing period under control. The experimental results reveal the performance of the proposed system in real time.

1 INTRODUCTION

Vision impairment is one of the most debilitating of all handicap conditions that exist. Affecting over 1.3 billion people worldwide with some form of distance or near vision impairment, it is the fastest growing medical conditions given our constant exposure to high energy light emitted from digital screens from childhood onwards. This has led to experts touting that a 'Global epidemic of blindness' is on the horizon. We have set forth to develop a model which can detect relevant obstructions in front of a person and convert it to speech to be transmitted to the user in an earpiece. This can serve as a guide to the world for a person with visual impairment. The challenge comes with making the model highly accurate while 'seeing' even just a part of the object. This required extensive training even with partial objects. The model also needs to fast enough to work in a real time. We achieved both of these criteria albeit limited computing power. Given higher computing power, the model can be used to assist visually impaired people with no observable lad. Our results on the object detection are presented and can be used to detect up to 45 objects in the frame at a time.

2 Approach

To assist visually impaired people with obstructions in front of them, we have prepared a custom dataset of objects that we come across in our daily life. The dataset we used contained images of both full objects as well as their partial counterparts to facilitate highly precise

image detection and training even with partial objects. The dataset also contains a huge variation of the images be it from angles or dimensions.

2.1 Dataset

For the dataset, we have coagulated several smaller datasets from Flickr and other sources. These images had their bounding boxes generated by Flickr after they were queried. For other sources we used third party softwares like Bbox Label tool to generate their bounding boxes manually. We obtained a dataset of 24 objects and a collection of total 50000+ images of 4.3 GB. Distribution of images of each class is shown in fig 2.1.
A complete list of these 24 objects includes human, two-wheeler, vehicle, traffic light, stop sign, bird, cat, dog, bag, umbrella, tie, bottle, cup, fork, knife, spoon, bowl, banana, apple, pizza, chair, laptop, mobile and book.

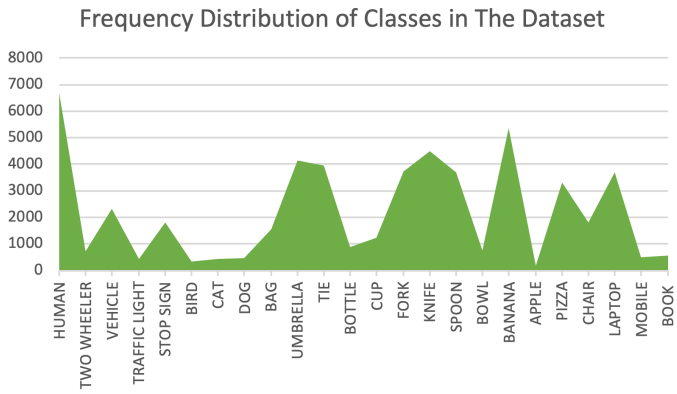

Fig 2.1 – Frequency Distribution of Classes in The Dataset

2.2 Preprocessing

The obvious issue that we faced, dealt with wrongly labelled images which would lead to inaccurate detection and lower accuracies. Most of the images which were not matching any class were manually removed during preprocessing of the dataset however a certain degree of unmatched images always remains be it with misclassified images or with inappropriate lighting of the image providing an inaccurate representation of the object. These images were removed during the manual drawing of their bounding boxes. A snippet of sample images in the dataset are shown in fig 2.2.

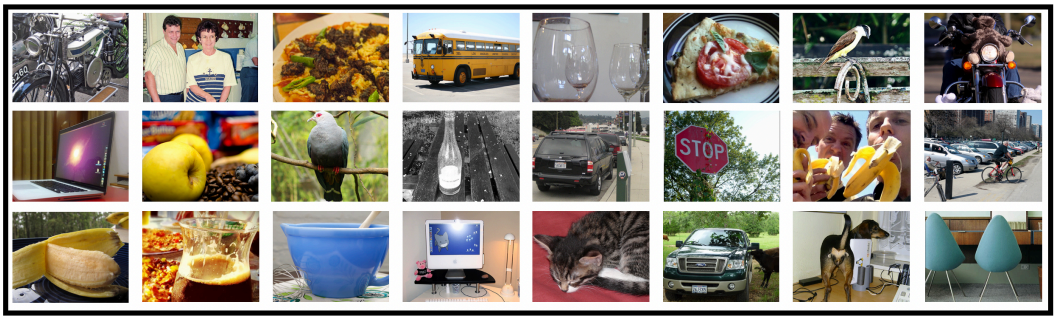

Fig 2.2 – Sample Images from the dataset

2.3 Bounding Boxes

In order to solve any supervised Machine Learning task, you need labelled data. The critical issue in training an object detection model deals with constructing their bounding boxes. The

bounding box or the anchor basically tells the model what part of the image is the object that it needs to recognize.
Most of the time bounding boxes are manually drawn and dimensions are fed along with the image in a txt file. However, doing the same for 50k images is not possible hence we queried the flickr images dataset which comes with predefined bounding boxes for each image. For the rest of the images that we extracted from several other sources we drew their bounding boxes using Bbox label tool, shown in fig 2.3.
We train our model using the bounding boxes coordinates and once the classifier is trained, we display the predicted confidence score on the final predicted bounding box with the above classifier's result.

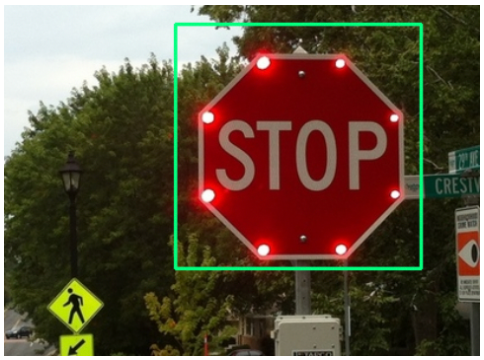

Fig 2.3 – Bounding Box

3 Experiments

3.1 YoloV3 Custom Configuration

We made changes in configuration file in-order to train our custom dataset since it uses darknet53 configuration files were already given but the configuration file is specific to a dataset and the hardware capabilities. First and foremost, we changed the filters in the configuration to 87 using equation 3.1 in which B=3 and C = number of classes which is 24 in our case.

$$\text{Filter} = (B \times (5 + C)) \quad \text{.....[3.1]}$$

Secondly, we changed the batch number which specifies how many iterations will be performed for each of the epoch since we had only 4GB GTX 1050 GPU we had to decrease the batch size which increases the number of iteration per epochs, furthermore we also decreased the subdivisions in order to match with the computing performance.
Then we set the layers attribute the under the [route] to -4 so it will output the feature map to fourth layer from back. The for the anchors we changed the mask to mask = 5,6,7 so it used these anchor numbers from the list of anchors. Then we created the configuration parser function in order to parse the '.cfg' file that we made for model processing.

3.2 Visualizing the YoloV3 Model Architecture

We visualized the architecture using the tensor flow graphs library in order to get an understanding of what is happening within the network or what is the flow of the network. This was an integral part of the experiment since we needed to understand the bounding box layers of the architecture so that we can make proper predictions. This was on the basis of the number of layers we needed to freeze and also, we can experiment with sigmoid or Leaky ReLU functions.

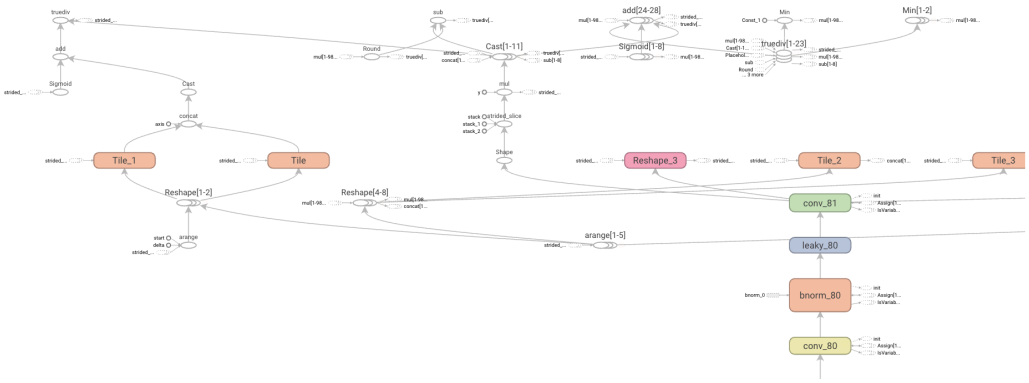

Fig 3.1 – Visualization of YOLOV3 Model Architecture

3.3 Training

It is the most time-consuming part of the project, in our code the model goes through two fit_generator steps each of which is compiled with 'Adam' optimizer. In the first fit_generator step we freeze the first 42 layers of the darknet53 body to train the model on the data set which runs 50 epochs and each of which runs 1406 iterations. In the second fit_generator we unfreeze all the layers of the model and continue the training to fine tune the model and it trains longer if the result is not good using the call back method of keras library, which runs another 50 epochs. Since more GPU memory is required after unfreezing the body so batch size for this was reduced to 16 so each epoch ran for 2600 iterations.

3.2.1 Comparison Between Our Custom Dataset and CIFAR10

We first trained out model on the CIFAR10 dataset then calculated mAP (Mean Average Precision) which is a three-step process, which is first we compute the precision and recall as given in equation 3.2, then average maximum precision across all recall levels in step sizes [1].

$$Precision = \frac{TP}{total\ positive\ result} \qquad \text{...[3.2]}$$

$$Recall = \frac{TP}{TP + FN}$$

Table 3.1 shows the mAP values of CIFAR10 and own 24 class it is clear that our dataset is has a better performance than cifar10 dataset, this is the reason of taking our data set over cifar 10 also it has more classes which are of use of a blind person.

	mAP value (on GTX 10150)
CIFAR10	50.9
Our 24 Class Dataset	55.1

Table 3.1 – Compassion of mAP values of ciaf10 and custom dataset

Then we used another performance metric of IoU (Intersection over Union) which is the measure of the overlap of the two regions; in our case which is the ground truth bounding box labeled by the Bbox tool, and prediction bounding box as shown in fig 3.2. The IoU values of our dataset were giving better values during iterations for each of the epochs.

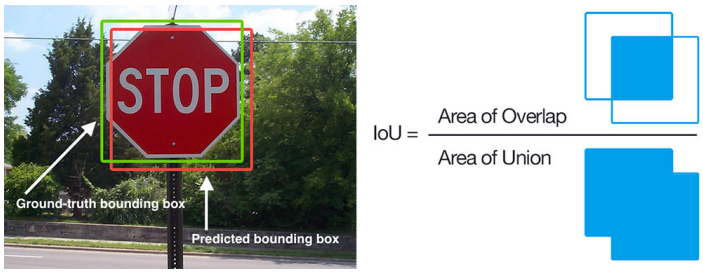

Fig 3.2 – Example Intersection of Union [1]

3.4 Calculating the loss

To calculate the loss, we use Binary Cross Entropy since the input is between zeros and ones it is used to measure the performance of the model, basically we need to optimize the weights in order to obtain the desirable outcome. So, to find optimal weights we used binary cross entropy. Using the keras binary cross entropy function it is a cross entropy between and output tensor and a target sensor [2], it is helpful to avoid experience overflow. However, the confidence and class prediction score is predicted thought logistic regression.

3.4 YoloV3 vs Resnet – 101

Each network is tested at 256x256 and is trained under similar setting. We calculated the run times on GTX 1050 GPU at 256x256 therefore it can be concluded that Darknet-53 performs on par with state-of-the-art classifiers but is fast and has less floating-point operations. On comparison of the performance of DarkNet-53 with ResNet-101, it can be said that it is 1.5 times faster. Darknet-53 has highest measured floating-point operations per second which implies that the network structure uses the GPU to the fullest thus making the evaluation more efficient and quicker. The reason behind such behavior is that ResNet-101 have a lot of layers which are not efficient as well.

3.5 Testing on darknet19 vs darknet53

YoloV3 uses the darknet53 which has 53 layers and then the YoloV3 uses another 53 in total making 106 layers giving fully conventional architecture, which makes this model slow but at the same time it can detect even smaller objects which yolov2 wasn't able to do. Yolov3 uses SSD (single shot detector) as shown in fig 3.3. It has no region proposal network, and Instead a set of default boxes with different sizes, aspect ratios and positions are used and then it performs classification on of the boxes.

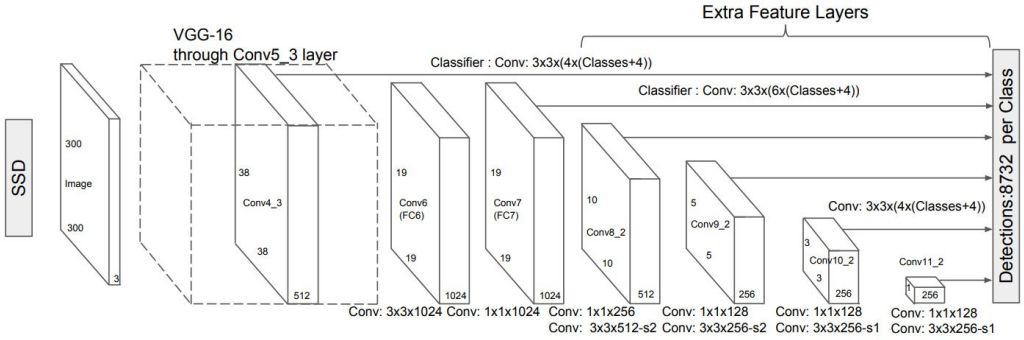

Fig 3.3 – Typical OD Network: Single Shot Detector (SSD) [2]

Where if we talk about darknet19 it is a 19-layer architecture with 11 more layers for detecting object making a total of 30 layers, but it was struggling with detecting small objects, so we decided to go with darknet53 weights [4], table 3.2 shows the comparison of the mAP values of darknet19 and darklnet53.

	mAP value (on GTX 10150)
darknet19	47.2
darnet53	55.1

Table 3.2 – Compassion of mAP values of darknet19 and darnet53

3.5 Demonstration of the System

Figure 3.3 shows the demonstration of the system in a real world scenario using the webcam of the laptop.

Fig 3.3 – Sample detection of Real Time Object Detection for Visually Impaired People

4 Conclusion and Future Work

4.1 Conclusion

We used the OpenCV library to detect objects with the laptop's front-camera. The detected objects were classified into different classes based on our trained model. The predicted classes were converted into an audio output along with a boundary box containing the object and objectness score for each boundary box.

4.2 Future Work

The possible future work can be used to make vast improvements to the model. These include adding a distance measurer in the module such that the audio input can also tell the distance to the person. This can be in the format, 'Human: 2m'. This can be made possible by addition of a distance sensor like an Ultrasonic Distance measurer (Parallax 28015) or Microsoft Kinect

etc. These distance sensors can accurately give the location of the first object in front of the person.

In addition to providing distance functionality, our device can also be trained on the family and friends of the visually impaired person. With an extended dataset of these people, we can implement facial recognition to help identify certain people by their names instead of giving the vague attribution of 'human' to every person. For immotile visually impaired people recognizing the few people they interact with on a daily basis instantly can be a boon.

Thirdly, the computation power for the model can also be increased if we shift the operations to Google Cloud platform or any of its counterparts. The crucial part of our model is real time object detection and a delay of even seconds is unaffordable. Hence shifting our model to the cloud with the availability of high-speed computation power can make a significant difference.

4.3 Contributions

Task	Member
Dataset ETL	Tushar Kapoor, Kashish Kohli, Mehak Parashar, Manan Parasher, Kanksha Masrani.
Bounding Box Labeling	Tushar Kapoor, Kashish Kohli, Mehak Parashar, Manan Parasher, Kanksha Masrani.
Training on Custom Dataset	Tushar Kapoor, Mehak Parashar
Training on CIFAR-10	Kashish Kohli, Kanksha Masrani
ResNet 101 and YOLOv3	Mehak Parashar, Manan Parasher, Kanksha Masrani
Darknet 19 vs Darknet 53	Kashish Kohli, Tushar Kapoor
Distance & Speech Module	Manan Parasher
Poster	Tushar Kapoor, Kashish Kohli, Mehak Parashar, Manan Parasher, Kanksha Masrani.
Report	Tushar Kapoor, Kashish Kohli, Mehak Parashar, Manan Parasher, Kanksha Masrani.

References

[1] A. Rosebrock, "A gentle guide to deep learning object detection," 14 May 2018. [Online]. Available: https://www.pyimagesearch.com/2018/05/14/a-gentle-guide-to-deep-learning-object-detection/.

[2] "tf.keras.backend.binary_crossentropy," 20 November 2018. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/backend/binary_crossentropy.

[3] C. Lee, "[Learning Note] Single Shot MultiBox Detector with Pytorch—Part 1," 3 July 2017. [Online]. Available: https://towardsdatascience.com/learning-note-single-shot-multibox-detector-with-pytorch-part-1-38185e84bd79.

[4] A. Kathuria, "What's new in YOLO v3?," 23 April 2017. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.