# Retail Store Usage Analysis

Roshan Sridhar
Dept of Electrical and
Computer Engineering
NYU Tandon School of Engineering
N18434577
rs5788 at nyu dot edu

Abhishek Vasu
Dept of Electrical and
Computer Engineering
NYU Tandon School of Engineering
N19532416
av1912 at nyu dot edu

***Abstract*—Predicting people and their actions are valuable for a multitude of reasons. Analysis of customer behavior is key to a company's progress. The data obtained is of vital importance and is analyzed and for benefits, needs, convenience, demands and profits. A deep learning system is efficient in detecting trends from their actions and movement. Our project aims to create an application that detects and records people in a surveillance videos. The recorded data is visualized to observe movement patterns and recognize trends of people. To successfully implement this, we proceed with a well known deep learning system called You Only Look Once: Unified, Real-Time Object Detection and modified it to our application.**

***Index Terms*—Person Detection, Convolutional Neural Network**

## I. Overview

The project requires a solid and accurate object detection system to perform effectively. The main component of the project is the detection system called YOLO. It stands for You Only Look Once. The YOLO: Real-Time Object Detection is an extremely fast real-time object detection framework[1]. It is used for object detection for object classes found in the Pascal VOC 2012 dataset[2]. This is originally built on C and CUDA. Using this prebuilt implementation, we lacked the ability to view and learn from the code. We also could not edit the code to add our features or optimize the code to suit our applications needs. To overcome this problem, we use a Tensorflow implementation of the same system created by Choi[3]. Both these systems described above perform object detection for image files. Changes were implemented to this application to perform detection on videos and hence, to cater to our project. The detected people are tracked across the video space and recorded. This data is then visualized to effectively depict human activity. The visualization is generated in the form of a heat map using the Seaborn library for Python. This provides an innovative way to represent patterns and trends thus help analyze retail store usage.

## II. Concept

Older systems run classifiers on images to perform detection. YOLO, compared to these detection schemes, executes the complete detection pipeline by looking only once at an image. In YOLO, the detection is posed as a regression problem. It contains a single neural network that concurrently predicts both bounding boxes and class probabilities for these boxes.

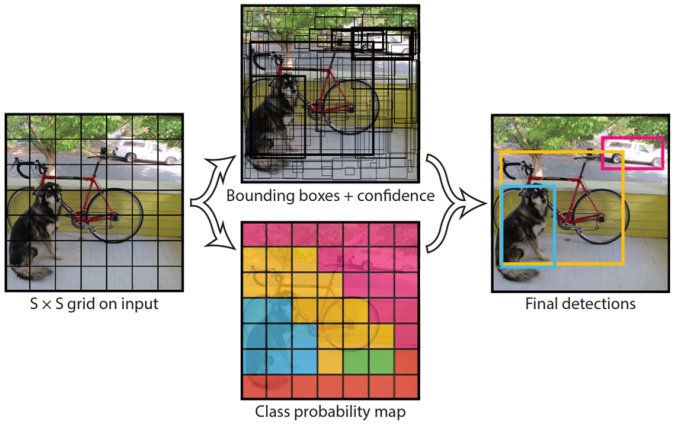


Fig. 1: The model: Image is divided into an SxS grid. Each grid cell predicts bounding boxes with confidence values and class probabilities.[1]

See Figure 1. To perform this, a new parameterization scheme was introduced to detect objects. A grid is imagined to be overlaid over the target image. The cells in this grid are useful for multiple reasons.
First, each cell predicts bounding boxes and it's confidence values for each box, which contains the probability that the cell contains the detected object. Note that if a cell doesn't contain any object, it still predicts bounding boxes but the confidence values are very low. This creates a map of all objects detected in the image along with their confidence values. Note that this only returns the location where the objects are located, and not what the object is. The confidence is defined as

$$Pr_{object} * IOU_{pred}^{truth}[1] \quad (1)$$

the product of the probability of the object and the intersection over union(IOU) between the prediction and ground truth.

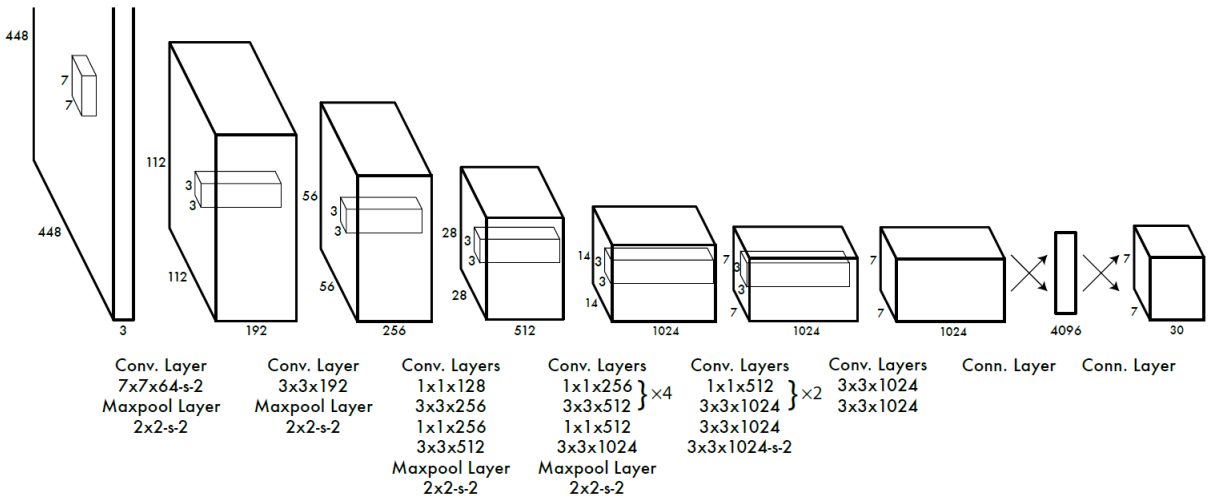


Fig. 2: The network architecture.[1]

Next, class probabilities are also predicted by each cell. This generates a segmented map of the classes predicted in the image by their conditional probability values.

$$Pr(Class_i|Object)[1] \tag{2}$$

In simple words, the segmented map denotes that if an object is present in a particular segment, then by probability, it belongs to the predicted class of that segment.
Finally, the product of confidence values computer earlier and the class probability values passed through a threshold function generates the bounding boxes of the highly probable objects present in the image.

$$Pr(Class_{ij}|Object) * Pr_{object} * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}[1]$$

For Pascal VOC data, the suitable grid size S x S chosen is 7 x 7.

See figure 2 for the complete network. This network contains 24 convolutional layers and 2 fully connected layers.

## III. Project Accomplishment

Since the original program is a pre-built application, the code is not transparent. This doesn't allow us to modify and change the code with ease. To overcome this, we use Jinyoung Choi's implementation of the YOLO object detection called YOLO_tensorflow. YOLO_tensorflow is created using the Python 2.7 programming language. The Tensorflow library for Python is employed to build and emulate the convolutional neural network. Using this implementation provides an open source solution to modify and optimize the code to suit our application. The Opencv2 library for Python is used to read media files.

### A. Video Implementation and Detection

YOLO_tensorflow, similar to the original YOLO, is built to detect objects in image files. First step was to make it video compatible. To perform this, the read function is first changed to read a video file. The detection function process is now executed frame-by-frame for every frame of this video. A video codec is specified to cater to our suitable platform. The bounding boxes are then applied on all the frames before all the frames are recompiled and saved.

Fig. 3: Frame Input

### B. Matrix Representation

A matrix with the same size as the frame pixel resolution used by YOLO for computation is declared to record and store the tracking information of a person. YOLO re-sizes the frame to a 448 x 448 pixel resolution and the same dimension is used to define the tracking matrix. This tracking matrix is first initialized to zero and is updated at every iteration of the detection performed on the frame. During detection, the elements of the matrix at the corresponding pixel area

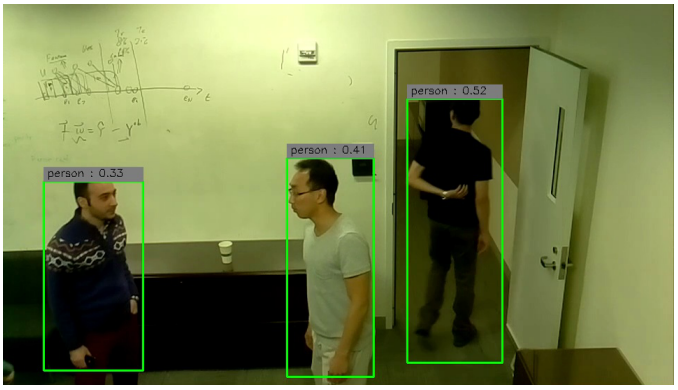


Fig. 4: Frame Output

occupied by the bounding box is incremented in an iterative fashion.

### C. Heat Map Generation

After the detection process is complete, the matrix now contains high values at co-ordinates where there is highest human activity and vice-versa. To generate a heat map from the tracking matrix now containing gathered data, we use a data visualizing library called Seaborn. First, the tracking matrix is L2 normalized to generate a smooth graph. It is then passed into the heatmap function and plotted.
For convenience, the heat map is superimposed over an empty frame of the video to effectively visualize and interpret human activity.

Fig. 5: Generated heatmap from matrix

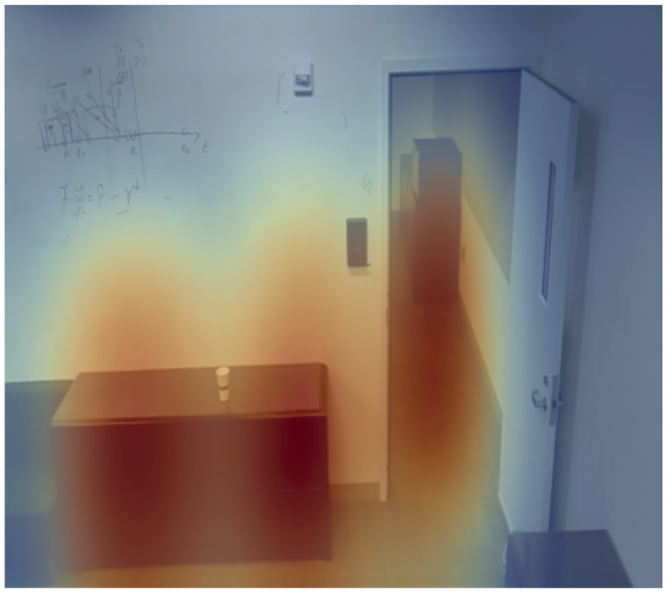

Fig. 6: Heatmap superimposed to frame

## IV. Summary & Future Work

We effectively use a robust object detection deep learning system for images to work with videos and use this application to track, record and visualize customer usage in a retail store setting.
In the future, this project can be expanded to work with real-time video feed to provide instant results. With this feature implemented, we can further improve its effectiveness in real-life scenarios by building the application on a portable development board. This would allow us to mount the same on a surveillance camera to produce and provide real time usage data and statistics.
**Acknowledgements:** Special thanks to Prof. Yao Wang and advisor Yilin Song for providing guidance and support.

## 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] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98136, Jan. 2015.
[3] J. Choi YOLO_Tensorflow: Tensorflow implementation of 'YOLO : Real-Time Object Detection'. 2016-2017. https://github.com/gliese581gg/YOLO_tensorflow/

## V. Appendix

The attached depository contains
1. Python code 'YOLO_small_tf.py'[3]
2. Weights file pre-trained on Pascal VOC[3]
3. 'own_dataset.mp4' Input file by advisor Yilin Song
3. 'Output.avi' Output video after detection
4. 'heatmap.png' Generated heatmap