## Document 1

Following is a report for a freelance project that I undertook in **January 2019** for the implementation of a people counter by utilizing overhead cameras placed in malls and shopping complexes. The code for the following project is available on <https://github.com/darpan-jain/crowd-counting-using-tensorflow>

**PEOPLE COUNTER**

horizontal line

### INTRODUCTION

This report summarizes my attempt at performing the task of implementing a people counter on the Open Source mall dataset.

### 

### USING PRE-TRAINED MODELS

Tensorflow’s Object Detection API provides pre-trained models for object detection, which are capable of detecting around 90 classes (objects) with ‘person’ being one of the classes.

On giving test images to a pre-trained model, the inference results were not as per requirements. In some instances the model detected the entire image as a person and also missing out on some fairly obvious ones.

*Figure 1. Results by the pre-trained model*

### TRAINING CUSTOM MODEL

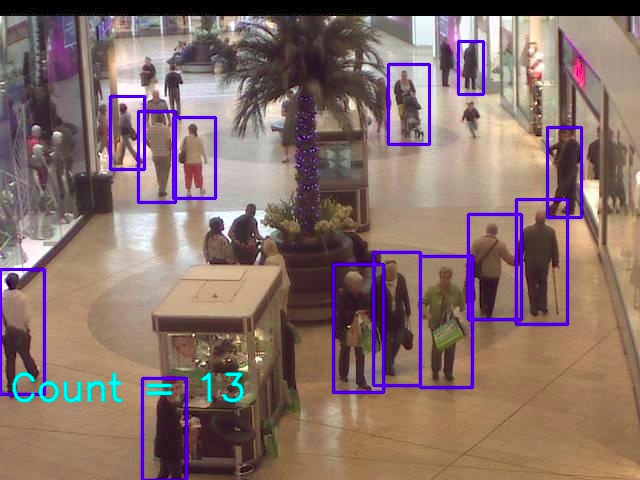
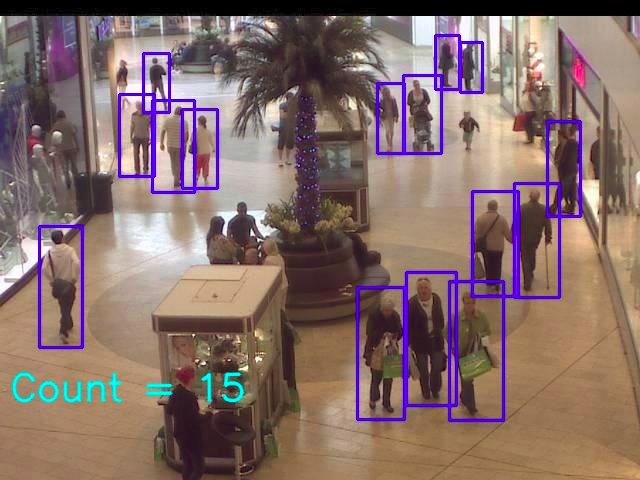
A custom model had to be trained for accurate implementation. The following steps were taken for the same.

1. Annotated training data had to be prepared before being given to the model.
2. 'LabelImg' was used for this purpose, to draw bounding boxes around objects of interest in the training images.
3. LabelImg gives the output as XML files containing coordinates of each of the bounding boxes in an image and the associated label of the object.
4. All the XML files were converted to a 'train.csv' and then into a 'train.record' format. TFRecord format is required by Tensorflow to perform the training of a custom model.
5. Similarly a ‘val.record’ was created for validation data.
6. The architecture of the model is based on the Faster RCNN algorithm, which is an efficient and popular object detection algorithm that uses deep convolutional networks.
7. The model config file was modified for the purpose of this assignment.
8. The last 90 neuron classification layer of the network was removed and replaced with a new layer that gives output for only one class i.e. person.
9. The config file for the same can be found in './data/utils/faster\_rcnn.config'
10. After training the model, the checkpoint model is saved as .pb file.
11. This model can now be deployed and used for obtaining inferences
12. The test images were the first 10 frames of the mall dataset.

The model can be found on this drive link: [Custom Model Link](https://drive.google.com/open?id=1IBgEyaASf10KUFTCbky9mtruUpyoqDWR)Please download and place the model in ./data/utils folder before executing main.py.

### RESULTS

Upon running 'main.py', the results are saved as shown below. (Refer ‘./results’ folder)

*Figure 2. Results of main.py*

### PREREQUISITES

All the required dependencies can be installed by running the command ‘pip install -r requirements.txt’

## Document 2

The following is a report for a technical assignment performed for a startup in Bangalore name Infilect Technologies in November 2018. The task was to use a combination of image and text data to identify duplicate products from a database of ~500K fashion products using Machine Learning.

Duplicate Detection

INTRODUCTION

This report briefly summarizes my attempt at performing the task of detecting Duplicate product listings from the provided dataset. The task was challenging but definitely interesting. Now while writing this report, I can identify certain aspects where I could have delivered better results. Nonetheless, I put my best effort into this assignment and have learned a great deal from it.

DEFINING WHAT “DUPLICATE” MEANS

The first thing that comes to mind when you hear the word duplicate is that the color and size would be the same. The seller being just a middle man would be just purchasing the products from some brand that manufactures it. So I decided to sort (cluster) the products according to the brand name. As the problem statement mentioned, the seller either listing the same product across different sites or on the same site; they would still belong to the same brand. Later on, the other attributes were taken for comparison.

HANDLING THE DATA FILE

The first most obvious thing to notice was obviously the sheer size of the dataset.. Glancing at the data files given, I gained some insights about the kind of data I’ll be working on. I noted the columns with the most relevant information; these were the only columns to be used. They were: Title, ProductID, Product brand, mrp, seller name, color, size and image url. Columns such as special price, seller reviews etc. were not useful in the given assignment and were dropped.

VECTOR REPRESENTATION

For clustering the brands (the feature here), feature vector extraction was needed. Since the brand names could be more than one word, doc2vec was used instead of word2vec. A model trained on Wikipedia data was used. The model returned 300-dimensional vectors for each brand name.

DIMENSIONALITY REDUCTION

The 300 dimensions had to reduced before being fed to the clustering algorithm. Principal Component Analysis (PCA) was used for this purpose. The dimensions were brought down to 3 using this approach; a relatively smaller dataset to perform clustering on.

CLUSTERING

The main obstacle here was to define the number of clusters. Hierarchical clustering was the first method of clustering I tried but soon I encountered a ‘memory error’. As it turns out, agglomerative clustering is not very scalable as it requires O(n^2) memory. Another approach was Birch Clustering which did not require the no. of clusters to be defined. Although quick the results were not very convincing. Later the execution times for MiniBatchKMeans and KMeans clustering were compared, the difference not being significant.

IMAGE HASHING

The images of each product were downloaded using the URLs provided and stored. The Python Imaging Library was used for image processing. All the images of products belonging to one cluster were converted to Hash values using ‘dHash’ algorithm which essentially compares the difference between pixels. It first converts the image to grayscale and resize it, then takes the relative gradient of the two adjacent pixels and generates hash values. The similarity between the hash values of two images would help us in determining the duplicates.

FINAL COMPARISON

The final step was to compare the other columns’ data within the clusters. The attributes were taken into consideration and comparisons were made with respect to each product. The product ids of duplicate products were then written in the JSON file. A duplication parameter was used to determine whether two products are indeed similar after comparisons with the remaining features (seller name color, size, mrp). The parameter was increment according to the relevance of the feature being used.