Machine Learning Engineer Nanodegree

Capstone Project

Moinuddin Syed

Project Overview

Humans use their eyes and their brains to see and visually sense the world around them. Computer vision is the science that aims to give a similar, if not better, capability to a machine or computer. Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding [1]. Researchers realized that it was necessary to tackle images from the real world. Thus, much research was needed in the so called ``low-level" vision tasks such as edge detection and segmentation. Low-level image processing algorithms are applied to 2D images to obtain the 'primal sketch' (directed edge segments, etc.), from which a 2.5 D sketch of the scene is obtained using binocular stereo. Finally, high-level (structural analysis, a priori knowledge) techniques are used to get 3D model representations of the objects in the scene. This is probably the single most influential work in computer vision ever [2].

Object detection, is an interesting challenge to identify the object in an image or series of frames of images, and has seen a lot of its application in cutting edge technologies like self driving car, security systems etc.

In the following section we will see, how object detection can be used to identify some products in an image.

Problem Statement

In this project I will be applying deep learning technology (CNN), to identify an object in an image as a product for ecommerce , image search purposes. For example, given an

image with multiple objects and if there is a *laptop* object in the image, the object identification model should identify the object in the image as a *laptop*, with its bounding box. Right now for this project the scope of product categories is electronic and electrical devices, I wish to later expand this to all categories.

Metrics

The primary evaluation metric used widely for object detection is **mAP** (mean Average Precision).

To calculate it for Object Detection, you calculate the average precision for each class in your data based on your model predictions. Average precision is related to the area under the precision-recall curve for a class. Then taking the mean of these average individual-class-precision gives you the Mean Average Precision.

Precision is the percentage true positives in the retrieved results. That is:

$$\text{precision} = \frac{tp}{tp + fp} = \frac{tp}{n}$$

where *n* is equal to the total number of images retrieved (*tp* + *fp*).

Recall is the percentage of the airplanes that the system retrieves. That is:

$$\text{recall} = \frac{tp}{tp + fn}$$

Average Precision is the area under the Precision-Recall curve.

sum over the precisions at every possible threshold value, multiplied by the change in recall:

$$\sum_{k=1}^{N} P(k)\Delta r(k)$$

where *N* is the total number of images in the collection, *P(k)* is the precision at a cutoff of *k*images, and *delta r(k)* is the change in recall that happened between cutoff *k-1* and cutoff *k*.[3]

Data Exploration

The dataset I use for this project is **openimages** dataset.

Open Images is a dataset of ~9 million URLs to images that have been annotated with image-level labels and bounding boxes spanning thousands of classes.Overall, there are 19,995 distinct classes with image-level labels (19,693 have at least one human-verified sample and 7870 have a sample in the machine-generated pool). Of these, 5000 classes are considered trainable.Overall, there are 600 distinct classes with a bounding box attached to at least one image. Of these, 545 classes are considered trainable (the intersection of the 600 boxable classes with the 5000 image-level trainable classes).[4]

The structure of the dataset is following,

1. **annotations-human-bbox.csv**, Human provided labels with bounding box coordinates (one file each for train, validation, and test

```
ImageID,Source,LabelName,Confidence,XMin,XMax,YMin,YMax
000002b66c9c498e,activemil,/m/0284d,1,0.560250,0.951487,0.696401,1.000000
000002b66c9c498e,activemil,/m/052lwg6,1,0.543036,0.907668,0.699531,0.995305
000002b66c9c498e,activemil,/m/0fszt,1,0.510172,0.979656,0.641628,0.987480
000002b66c9c498e,verification,/m/01mzpv,1,0.018750,0.098438,0.767187,0.892187
```

2. **class-descriptions.csv,** File with class label with their corresponding label names.

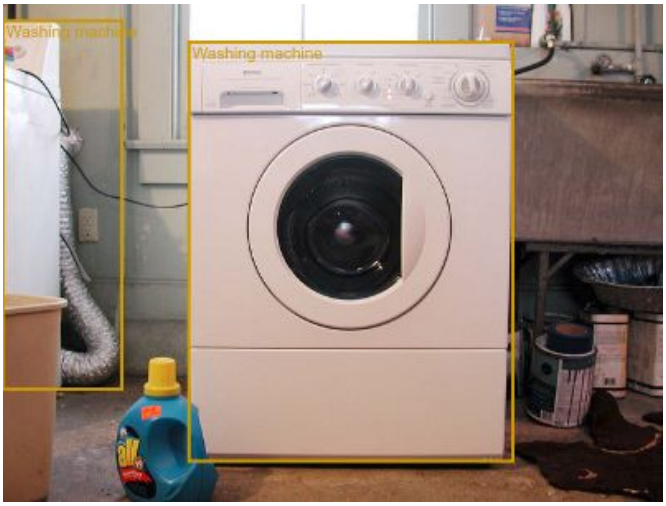
```
/m/025dyy,Box
/m/025f_6,Dussehra
/m/025fh,Professor x
/m/025fnn,Savannah Sparrow
```

3. **images.csv,** It has image URLs, their OpenImages IDs, titles, authors and license information (one for each Train, Validation and Test sets). Following are the fields it contains.

```
ImageID,Subset,OriginalURL,OriginalLandingURL,License,AuthorProfileURL,Author,
Title,OriginalSize,OriginalMD5,Thumbnail300KURL
```

Data Visualization

The dataset we use is in the form of images. Images contain the objects to be identified, the object bounding boxes are given for each object in an image.Below are some images with their human-verified bounding boxes drawn.
The bounding box is what help us to train the image with corresponding objects.



Algorithms and Techniques

CNN being the gold standard for image classification [5], something more than just classification is required for object detection.Object detection is the task of finding the different objects in an image and classifying them, this where Region based CNN (R-CNN) came into picture, to create boundary boxes or region proposals, for which a CNN generates features and then is classified and regressed to particular object class. But R-CNN also has been evolved along the time to more better and efficient versions, that is to Fast R-CNN and then to Faster R-CNN.[6]

In this project , Faster R-CNN is chosen as algorithm to solve this problem, Faster R-CNN uses a Region proposal Network which is solely based on CNN.

FASTER R-CNN

Faster R-CNN, is composed of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector [7] that uses the proposed regions. The entire system is a single, unified network for object detection . Using the recently popular terminology of neural networks with ‘attention’ [8] mechanisms, the RPN (Region Proposal Network) module tells the Fast R-CNN module where to look.

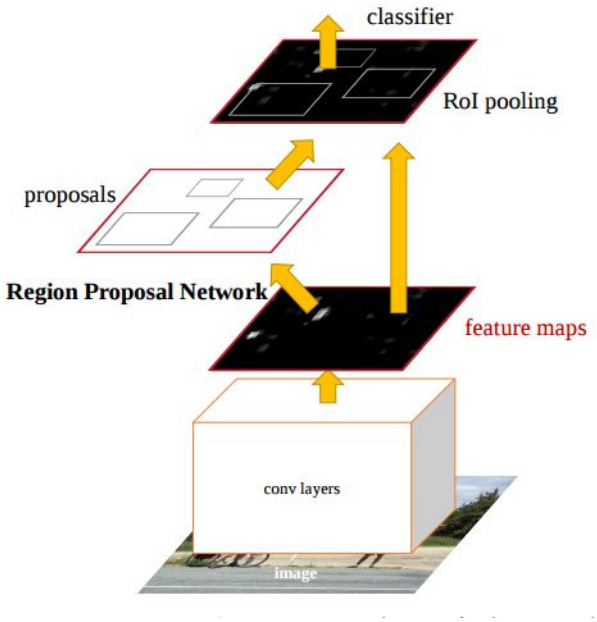


Figure1. Faster R-CNN is a single, unified network for object detection. The RPN module serves as the ‘attention’ of this unified network

A Region Proposal Network (RPN) takes an image as input and outputs a set of rectangular object proposals, each with an objectness score. This process is modeled with a fully convolutional network [9]. Because the ultimate goal is to share computation with a Fast R-CNN object detection network [7], we assume that both nets share a common set of convolutional layers. To generate region proposals, we slide a small network over the convolutional feature map output by the last shared convolutional layer. This small network takes as input an n × n spatial window of the input convolutional feature map. Each sliding window is mapped to a lower-dimensional feature. This feature is fed into two sibling fully connected layers—a box-regression layer (reg) and a box-classification layer (cls).. This mini-network is illustrated at a single position in Figure 2. Note that because the mini-network operates in a sliding-window fashion, the fully-connected layers are shared across all spatial locations. This architecture is naturally implemented with an n×n convolutional layer followed by two sibling 1 × 1 convolutional layers (for reg and cls, respectively)[10].

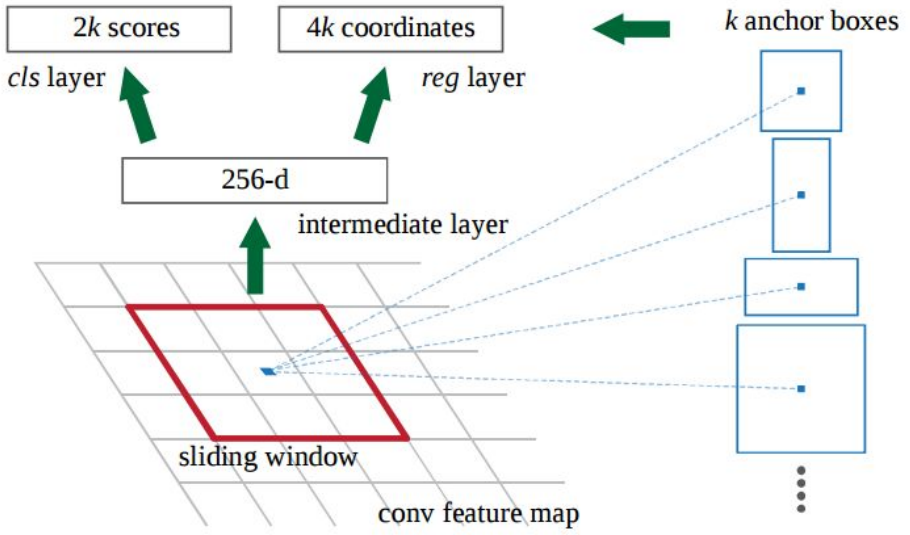


Figure 2. Region Proposal Network (RPN).

Transfer Learning

Object detection is a complex problem which using deep learning techniques requires that we train a model with hundreds of layers and millions of parameters. This is a tough task and requires a lot of data and compute resources. Transfer learning provides a framework to leverage the already existing model (based on some training

data) in a related domain. We can transfer the knowledge gained in the previous model to the new domain (and data).[11]

A pre-trained model is a model created by someone else to solve a similar problem. Instead of building a model from scratch to solve a similar problem, you use the model trained on other problem as a starting point.[12]

Benchmark

I will be using the results obtained by tensorflow model for openimages dataset using Faster R-CNN as benchmark[8]. The results obtained by tensorflow was **37 mAP** (mean Average Precision). As the data I'm using for electronic and electrical products, will be much smaller than the data the above model is trained, there will be variations in the results.[13]

Data Preprocessing

As the scope of this project is limited to detection of electronics and electrical products, only those images needs to be downloaded, from the class_description.csv, I shortlisted 17 classes.

```
id,labelid,labelname,n_imgs
1,/m/0174k2,Washing machine,319
2,/m/01b7fy,Headphones,913
3,/m/01c648,Laptop,5056
4,/m/01m2v,Computer keyboard,2692
5,/m/024d2,Calculator,199
6,/m/02522,Computer monitor,2527
7,/m/029bxz,Oven,466
8,/m/02pjr4,Blender,149
9,/m/040b_t,Refrigerator,506
10,/m/050k8,Mobile phone,3570
11,/m/063rgb,Mixer,198
12,/m/07c52,Television,2280
13,/m/07xyvk,Coffeemaker,255
14,/m/0dv5r,Camera,4185
15,/m/0fx9l,Microwave oven,431
16,/m/0h8mzrc,Wall clock,798
17,/m/0llzx,Sewing machine,365
```

I used *pandas* to import the annotation_human_bbox.csv files, then filter out the images which contain the classes, then fetched image_url of the desired images from images.csv.

Images are downloaded, they are verified to exclude the corrupt images and rescaled to a smaller size, for faster computation. After images are prepared, as we are using tensorflow's object detection api, the input of the images have to be in TFRecord format.

Tensorflow records are used nearly universally across Tensoflow objects as a dataset storage medium, and harbour a bunch of complexity. To create TFRecord, we need to pass the image metadata and the bbox coordinates.

Tensorflow requires a label_map file for evaluation, this object essentially just maps a label index with a label keyword as shown below.

```
item {
        id: 1
        name: /m/0174k2
        display_name: Washing machine
}
item {
        id: 2
        name: /m/01b7fy
        display_name: Headphones
}
```

Implementation

As we have now prepared the data, we need to now train it to achieve to attain our objective. We use tensorflow object detection API, and train the data on a pre trained model. The pre trained model we use is **faster_rcnn_resnet101_coco** using the **transfer learning** technology.

The machine we use for training has following specifications,

- 8 CPU
- Quadro M4000 with 8GB memory
- 30 GB RAM

We feed the image data with bounding box to the object detection API by first creating the TFRecords,

```
def create_tf_example(image_det, image_path, pdt):
   with tf.gfile.Open(image_path,'rb') as image_file:
       encoded_image_data = image_file.read()
   with Image.open(image_path) as img:
       width, height = img.size
   image_format = b'jpeg'
    filename = os.path.basename(image_path).encode("utf-8") # Filename of the
image. Empty if image is not from file
    xmins = [] # List of normalized left x coordinates in bounding box (1 per
box)
    xmaxs = [] # List of normalized right x coordinates in bounding box  (1 per
box)
    ymins = [] # List of normalized top y coordinates in bounding box (1 per
box)
   ymaxs = [] # List of normalized bottom y coordinates in bounding boxz
            # (1 per box)
   classes_text = [] # List of string class name of bounding box (1 per box)
   classes = [] # List of integer class id of bounding box (1 per box)

   for row in image_det.iterrows():
       xmin = row[1]['XMin']
       xmax = row[1]['XMax']
       ymin = row[1]['YMin']
       ymax = row[1]['YMax']
       labelid = row[1]['LabelName']
```

```
        class_text = labelid.encode("utf-8")
        class_ = pdt[pdt['labelid']==labelid].id.values[0]
        xmins.append(xmin)
        xmaxs.append(xmax)
        ymins.append(ymin)
        ymaxs.append(ymax)

        classes_text.append(class_text)
        classes.append(class_)

    print ("\nimage : {}".format(image_path))
    print ("classes : {}".format(classes_text))
    print ('classes_num : {}\n'.format(classes))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
      'image/height': dataset_util.int64_feature(height),
      'image/width': dataset_util.int64_feature(width),
      'image/filename': dataset_util.bytes_feature(filename),
      'image/source_id': dataset_util.bytes_feature(filename),
      'image/encoded': dataset_util.bytes_feature(encoded_image_data),
      'image/format': dataset_util.bytes_feature(image_format),
      'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
      'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
      'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
      'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
      'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
      'image/object/class/label': dataset_util.int64_list_feature(classes),
  }))
    return tf_example
```

Before we start training we need to create a configuration file, as we are using **faster_rcnn_resnet101_coco** as our pre-trained model, I use the same config with minute differences.

```
model {
  faster_rcnn {
    num_classes: 17
    image_resizer {
      fixed_shape_resizer {
        height: 350
        width: 350
```

```
      }
    }
```

Here *num_classes* is the number of classes / objects we are training with, The *image_resizer* is important, and there are two main types of resizing, *fixed_shape_resizer* and *keep_aspect_ratio_resizer*. Image dimensionality is important for object detection. It should be noted that fixed_shape_resizerwill pad the minor dimension instead of skewing or warping, which greatly improves stability in the face of natural web images []

We also mention our training configurations, where the *batch_size* is 1, as there is single GPU, and also we need to run evaluation parallely. We start with learning_rate at 0.00001

```
train_config: {
  batch_size: 1
    optimizer {
        adam_optimizer: {
            learning_rate {
            exponential_decay_learning_rate: {initial_learning_rate:0.00001}
            }
        }
    }
```

As I have used tensorflow object detection api, training was started by running a train.py file with its appropriate arguments.

```
python object_detection/train.py --logtostderr \
--train_dir=/home/paperspace/capstone_OD/train_dir2 \
--pipeline_config_path=/home/paperspace/capstone_OD/objectdetection_oid/data/pi
peline.config
```

The training starts and the below graph shows the drop in total loss, we can see that there is drastical drop in loss in initial stage, as we are training on the pre trained model. The total loss kept fluctuating between 0.5 and 1 after 10000 steps.

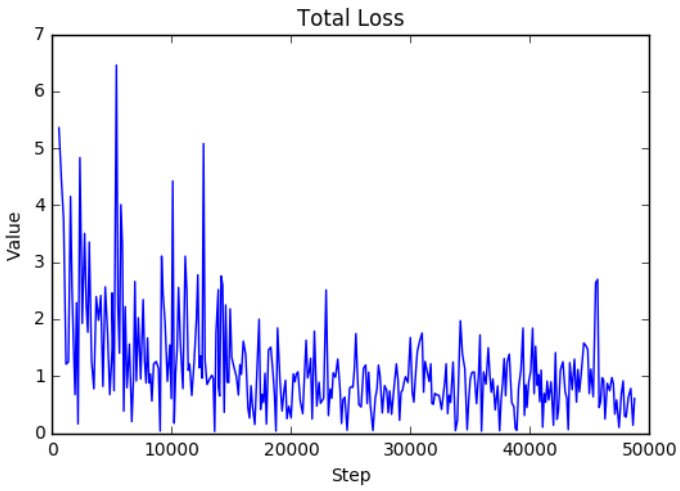


Figure 3. Total Loss

Evaluation of the model was also started on the validation set, parallelly to training to evaluate the model as it trains.

```
python object_detection/eval.py --logtostderr \
--eval_dir=/home/paperspace/capstone_OD/eval_dir \
--pipeline_config_path=/home/paperspace/capstone_OD/objectdetection_oid/data/pi
peline.config \
--checkpoint_dir=/home/paperspace/capstone_OD/train_dir
```

From evaluation, **mAP@0.5IOU** was calculated throughout the training steps. We can see that the mAP score increases to **60%** till **16K** steps and remains there until training finishes.

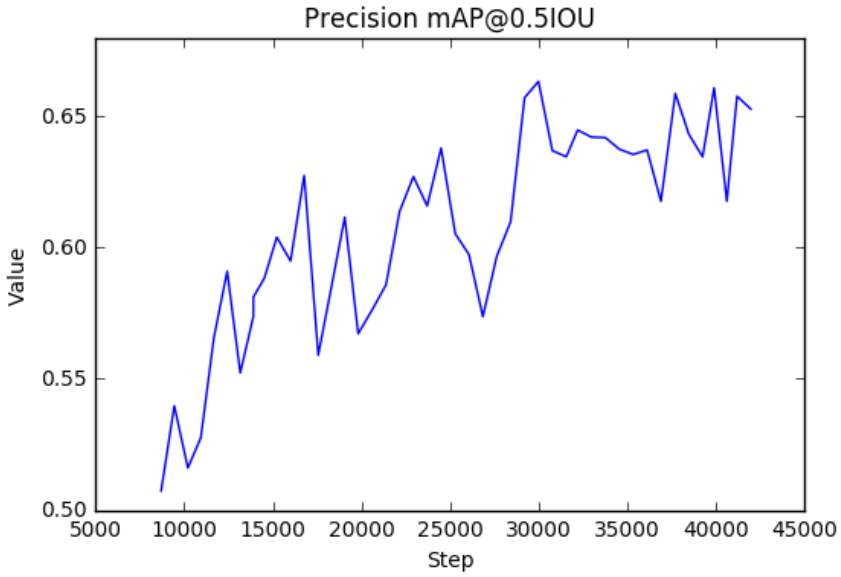


Figure 4. Validation mAP@0.5IOU

After training the model, we freeze the model graph, which allows us to combine the model structure (the configuration file) along with the weight and gradient data into a single binary protobuffer file.

```
python object_detection/export_inference_graph.py --input_type image_tensor \
--pipeline_config_path
/home/paperspace/capstone_OD/objectdetection_oid/data/pipeline.config \
--trained_checkpoint_prefix
/home/paperspace/capstone_OD/train_dir/model.ckpt-49830 \
--output_directory /home/paperspace/capstone_OD/output_dir
```

For the evaluation of the testing set, the Mean Average Precision (mAP) had to be calculated for the predicted results of the all testing images. The mAP is calculated by taking mean of Average Precision found at all the images. Average Precision is calculated by taking product of Precision at each image and rate of recall through all the images.

```
tp,fp,fn,precision,recall,p_recall,ap = 0,0,0,0,0,0,0

#images for which the testing has to be done against predicted
u_imgs = test_df.ImageID.unique().tolist()
mAP = 0
for i,img in enumerate(u_imgs):

        #annotations for objects in an image
        annon = test_df[test_df.ImageID.isin([img])]
        pred_res = get_predicted_box(img)
        if not pred_res:
            continue
        #iterating through each object and evaluating it
        for row in annon.iterrows():
            label = row[1]['LabelName']
            box = [row[1]['YMin'],row[1]['XMin'],row[1]['YMax'],row[1]['XMax']]
            label_check = False

            for key in pred_res:

                boxp = pred_res[key]
                iou = bb_intersection_over_union(box,boxp)
```

```
                if iou > 0.5:
                    if label == key:
                        tp += 1
                    else:
                        fp +=1
                    label_check = True
            if not label_check:
                fn += 1

        #to avoid divide by zero

        if tp+fp == 0 and tp+fn!=0:
            precision = 0
            recall = float(tp)/(tp+fn)

        elif tp+fn == 0 and tp+fp!=0:
            recall = 0
            precision = float(tp)/(tp+fp)

        else:
            precision = float(tp)/(tp+fp)
            recall = float(tp)/(tp+fn)


        rate_recall = recall-p_recall

        #cumulative average precision is calculated
        ap += float(precision) * float(rate_recall)

        #mAP is calculated dividing the ap by total images
        mAP = float(ap)/(i+1)

print ('The mean Average Precision (mAP) for test images is : {}'.format(mAP))
```

The *pipeline.config* file is where we change the parameters to alter the specifications in training process. I have done two training sessions, to train the model. As I was low on computation power with only one GPU, I reduced my batch size to 1, which was 5 initially, also in the first session I was unable to run the evaluation parallely as the training process took up the who GPU memory, therefore I had to reserve some memory in the second session to run evaluation parallely. With reduced batch size I could see reduce in step/sec while training and I could train the model for more number of steps. The results of second session were slightly better than the first one,

from 66.7mAP@0.5IOU to 66.8mAP@0.5IOU, which hardly makes any difference.

As we have used the transfer learning technique, default config of the trained model were used, and no refinements were made to algorithmic level.

Results

Results achieved during evaluation process on the validation set, showed that the **mAP** reached the peak of **65%** and remained around that until the training was stopped**.**

Evaluation run on validation set of images gave following results.

From the above images we see the model has performed fairly accurate by, detecting coffeemaker with **97%** confidence , headphones with **92%** confidence, mixer with **94%** confidence and refrigerator and oven with **91%** and **82%** correspondingly. But we also notice that predict box of refrigerator includes the microwave oven too, this is an error , but with *threshold* as **0.5**, the *IOU* being more than **0.5**, this won't much affect the **mAP**

There were also misclassifications, as we can see below,

In the above image we can see that the television was identified as a coffeemaker, this is a **false-positive**, and will affect the **mAP** score.

In the above image, both the iPhone and iPod were identified as mobile phones, whereas identifying an iPhone as mobile phone is true-positive, but identifying an iPod as a mobile phone is a false-positive, and will affect the **mAP** score.

When the evaluation was done using the **testing** set, the results calculated gave, a **66.8% of mAP@0.5IOU.**

Conclusion

We have seen a model trained on 17 product classes, and giving a validation score of **65 mAP@0.5IOU** and a testing score of **66 mAP@0.5IOU.** Both greater than our benchmark metrics which was **37 mAP@0.5IOU** trained on Open Images dataset. With this we can say that our model has performed fairly well with the same dataset, but fewer classes.

To summarise the whole project we can go through following points,

- At first we collected the data in different *csv* files, for later to be processed and to download required data, then verified and rescaled the images.
- We processed the images with the bounding boxes of the objects in the image to form a TFRecord file, which the tensorflow api takes in.
- We created the config file with training and evaluation configuration, along with the label_map file
- As we were using transfer learning technique, we downloaded the appropriate pre-trained model to create a checkpoint to train from.
- Using the tensorflow's object detection api, we trained the model, while evaluating it. When it seemed that the model had performed well enough using the evaluation from the validation set, we stopped the training process.
- The model was finally tested using the testing set, and to calculate the evaluation metric, Mean Average Precision (mAP)

From the above mentioned steps, I found two aspects to be of some challenge,

1. I was not clear when to stop training, as while evaluation I saw consistent variation in total loss, finally I took the mAP score as factor to know when to stop training, as it increased to a consistent level, and also looking at the model performance on the validation images.
2. Calculating the mAP metric on testing set was also challenging as, I was unclear of how to compare the ground truth boxes and the predicted boxes, but with good research I was able to accomplish this.

As the results seem promising, still there needs some improvement in decrease of false positives, in many cases of similar looking objects, this may seem to be solved with more data of unique characteristics in an object. We also saw cases where the object wasn't detected at all, these are the false negative, these issues can be solved by training the model for increased number of steps, with a higher compute capable machine.

With these results our model seems to be ready for real world applications. One of the main application where this model can be applied is e-commerce, to detect a product in an image and pass it through the product search API.

References

1. http://www.bmva.org/visionoverview
2. Huang, T. (1996-11-19). Vandoni, Carlo, E, ed. *Computer Vision : Evolution And Promise* (PDF).
3. Mean Average Precision, https://sanchom.wordpress.com/2011/09/01/precision-recall/
4. openimages github, https://github.com/openimages/dataset
5. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton ,ImageNet Classification with Deep Convolutional Neural Networks
6. Object Localization and Detection, https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html
7. R. Girshick, "Fast R-CNN," in IEEE International Conference on Computer Vision (ICCV), 2015.
8. J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in Neural Information Processing Systems (NIPS), 2015
9. J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
10. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun
11. Transfer Learning, https://www.packtpub.com/books/content/what-transfer-learning
12. Pre-trained model, https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/
13. Open Images-trained models, https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md#open-images-models