Convolutional Neural Networks for Object Detection

Udacity Capstone Project

Matthew Lee
August 25th, 2018

I. Definition

Project Overview

In recent years, one of the hottest applications of AI in computer vision has been object detection. Object detection has wide variety of applications such as robotics, autonomous driving, facial recognition, security, and etc. Such advancements are transforming our society and helping us to achieve higher goals with unprecedented speed and accuracy.

Despite some of the hypes around AI's unlimited potentials, current AI algorithms and solutions are very specific to individual applications. However, I believe object detection algorithms can be very versatile and used across countless applications.

This capstone project wishes to expand object detection domain by developing an algorithm to successfully detect objects in Open Images Dataset, publicly released by Google AI, hosted by Google in Kaggle competition: Google AI Open Images – Object Detection Track. Dataset contains 1.7 million images as train dataset that contain 12 million bounding-box annotations for 500 object classes, making it the largest existing dataset with object location annotations. Bounding-box annotations are drawn by professional annotators to ensure accuracy and consistency, making a very optimal dataset for training an object detection algorithm. Competition also provides 33K images, which are annotated with bounding-boxes as well, for model validation. Finally, there are 100k images as test dataset for which trained object detection model has to detect objects.

Training and Validation Set -> Download #1 or Download #2
Testing Set -> Download

Problem Statement

As a goal of this project, object detection model will be implemented in order to detect 500 object classes within 100k test image dataset. Model will output its prediction for detected object, prediction confidence, location, and size of the object relative to image size. In order to achieve this, 1.6 million train image dataset will be used to train selected object detection model. 33k validation image dataset will also be used to evaluate the model throughout training process. Effective use of validation dataset will be vital in this particular project by

allowing efficient use of given resources since overwhelming amount of resources and times are expected to be required to train the model.

Currently, the most optimal way to detect objects in an image and/or classify an image as a whole is achieved by training Convolutional Neural Networks (CNN) with labeled images with bounding boxes. CNN is a perfect fit for problems dealing with images due to its usage of spatial information capturing f filters, which can detect lines, curves, complex shapes, colors, and so on.

There are many CNN architectures available as an open source project, and as a result, they evolve and transform very quickly. Depending on certain combination of elements in an architecture and hyper-parameter tuning, CNN model's performance can swing very easily. Therefore, choosing applicable architecture needs to be careful in order to achieve a model that will surpass human's performance at detecting and classifying objects in an image.

Evaluation Metrics

Benchmark model and the solution's performances will be evaluated by mean Average Precision (AP) (*mAP*) across 500 classes in 1.7 million images in the dataset.

Precision (*P*) will be calculated by dividing true positives (*Tp*) over the true positives plus false positives (*Fp*) (Precision-Recall)

$$P = \frac{T_p}{T_p + F_p}$$

Detections are considered true or false positives based on the area of overlap with ground truth bounding boxes. To be considered a correct detection, the area of overlap *ao* between the predicted bounding box *Bp* and ground truth bounding box *Bgt* must exceed 50% by the formula (PASCAL VOC 2010):

$$a_o > 0.5 \; where \; a_o = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

Recall (*R*) is calculated by dividing true positives (*Tp*) over the true positives plus the false negatives (*Fn*) (Precision-Recall)

$$R = \frac{T_p}{T_p + F_n}$$

Then, average precision (*AP*) will be calculated for each of 500 classes.

$$AP = \sum_{k=1}^{n} (R_k - R_{k-1}) P_k$$

Finally, final mean average precision (*mAP*) will be calculated by taking average of all APs over the 500 classes.

$$mAP = \frac{\sum_{k=1}^{500} AP_k}{500}$$

Note that, unlike previous standard challenges such as PASCAL VOC 2010, Google AI Open Images - Object Detection Track has three new metrics that affect the way True Positives and False Positives are accounted (Open Images Challenge 2018 - object detection track - evaluation metric)

- Due to the Open Images annotation process, image-level labeling is not exhaustive.
- The object classes are organized in a semantic hierarchy, meaning that some categories are more general than others (e.g. 'Animal' is more general than 'Cat', as 'Cat' is a subclass of 'Animal').
- Some of the ground-truth bounding-boxes capture a group of objects, rather than a single object.

II. Analysis

Data Exploration

Open Images Dataset V4 provides 1,743,042 train images, 41,620 validation images, and 125,436 test images which span 600 object classes. However, for Google AI Open Images – Object Detection Track which provided scope for this project, we are only interested in images spanning over 500 object classes. As a result, number of train images reduced to 1,674,979, validation images to 33,073, and test images to 99,999. This truncation of 100 object classes for the competition was performed to remove some of very broad or infrequent classes such as “clothing” and “paper cutter”. Difference in data between 600 class and 500 class set is summarized in below **Table 1**. Notice that there is no bounding box information for test set of 500 classes because they are being used to evaluate the competition.

	Images Count		Bounding Box Count	
# Classes	600	500	600	500
Train	1,734,042	1,674,979	14,610,229	12,195,144
Validation	41,620	33,073	204,621	166,905
Test	125,436	99,999	625,282	N/A

Table 1. Difference in data for 600 and 500 classes of Open Image Dataset

Images for training, validation, and testing are downloadable in JPG format, which is a very popular extension for CNN training. Total volume of three sets of images will be over 500 GB. These images have been rescaled to have at most 1024 pixels on their longest side, while preserving their original aspect-ratio. Below is an example image that contains multiple objects.

Image 1. Source

Besides images, another vital input data is annotation for train and validation images that contain class, location, and size of objects in an image. Annotation files for both train and validation dataset follow the same format. Below **Table 2** shows sample of first 5 rows of validation annotation file.

	ImageID	Source	LabelName	Confidence	XMin	XMax	YMin	YMax	IsOccluded	IsTruncated	IsGroupOf	IsDepiction	IsInside
0	0001eeaf4aed83f9	freeform	/m/0cmf2	1	0.02246	0.96418	0.07066	0.80016	0	0	0	0	0
1	000595fe6fee6369	freeform	/m/02xwb	1	0.14103	0.18028	0.67626	0.73246	0	0	0	0	0
2	000595fe6fee6369	freeform	/m/02xwb	1	0.21378	0.25303	0.29876	0.35496	1	0	0	0	0
3	000595fe6fee6369	freeform	/m/02xwb	1	0.23293	0.28845	0.48895	0.54515	1	0	0	0	0
4	000595fe6fee6369	freeform	/m/02xwb	1	0.24537	0.29036	0.66185	0.71661	1	0	0	0	0

Table 2. First 5 Rows of Validation Annotation Data

Refer to the following bullet points for explanation of column names of annotation files (source for below bullet points):

- ImageID: the image this box lives in.
- Source: indicates how the box was made:

    - freeform and xclick are manually drawn boxes.
    - activemil are boxes produced using an enhanced version of the method [1]. These are human verified to be accurate at IoU>0.7.
- LabelName: the MID of the object class this box belongs to.
- Confidence: a dummy value, always 1.
- XMin, XMax, YMin, YMax: coordinates of the box, in normalized image coordinates. XMin is in [0,1], where 0 is the leftmost pixel, and 1 is the rightmost pixel in the image. Y coordinates go from the top pixel (0) to the bottom pixel (1).
- Attributes. For each of them, value 1 indicates present, 0 not present, and -1 unknown.

    - IsOccluded: Indicates that the object is occluded by another object in the image.
    - IsTruncated: Indicates that the object extends beyond the boundary of the image.
    - IsGroupOf: Indicates that the box spans a group of objects (e.g., a bed of flowers or a crowd of people). We asked annotators to use this tag for cases with more than 5 instances which are heavily occluding each other and are physically touching.
    - IsDepiction: Indicates that the object is a depiction (e.g., a cartoon or drawing of the object, not a real physical instance).
    - IsInside: Indicates a picture taken from the inside of the object (e.g., a car interior or inside of a building).

Exploratory Visualization

Diagram 1 below shows semantic hierarchy of 500 classes. Notice that some categories include other more general categories. (for example, class “animal” includes “cat”,”dog”, and etc). Therefore, end result of detection for an image containing a cat should provide detection for classes “animal” and “dog” to reach 100% recall.

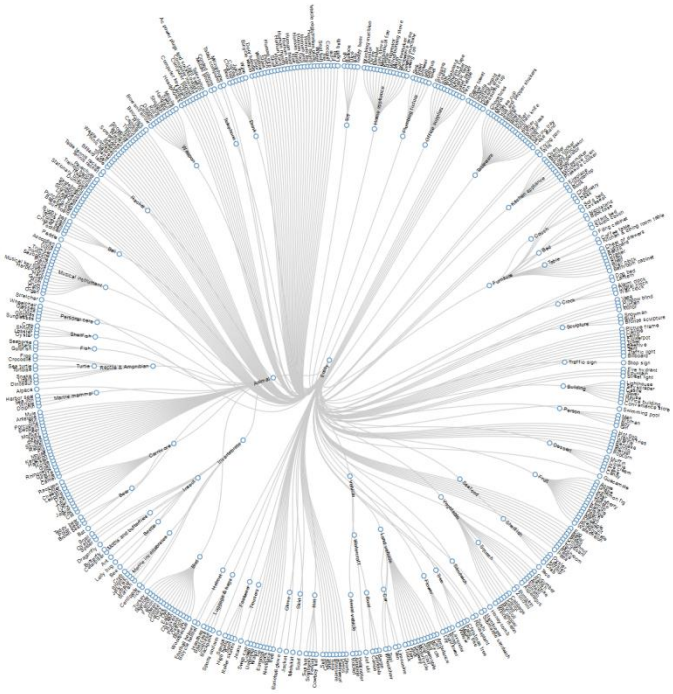

Diagram 1. Open Images Dataset V4 Flare Dendogram for 500 Classes (Link for full size)

Total number of bounding boxes for each of 500 classes in train images show a serious imbalance. This could slow down training process and potentially cause losses to diverge. This imbalance is nicely displayed **Diagram 2** below. Log scale is applied to y-axis which shows bounding box count of each class whiel x-axis represents 500 classes. Data preprocessing to balance training data in order to improve training result will be covered in later sections.

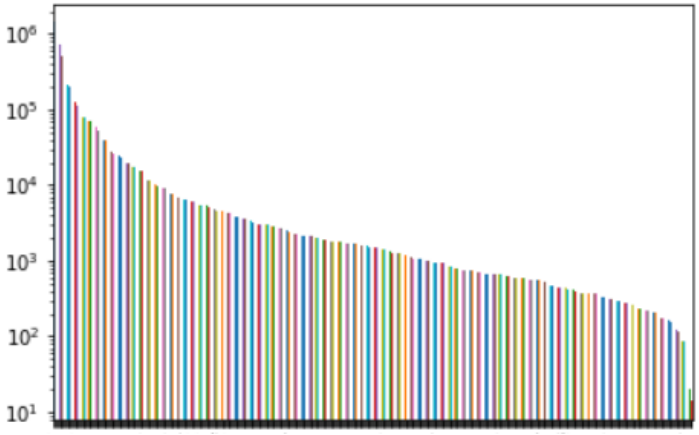

Diagram 2. Train Bounding Box Imbalance on Log Scale

Algorithms and Techniques

For this project, I chose YOLOv3 using Darknet-53 as a backbone to detect and classify objects. YOLOv3 has CNN architecture with many of current state-of-the-art features implemented such

as residual skip connections, multi-scale training, data augmentation(rescale, flips, and etc), batch normalization (YOLOv3, 2018). Overall, YOLOv3 has Convolutional layers that perform feature extraction. Below **Image 2** shows a snapshot of aforementioned architecture. You can see that residual skip connections are implemented. Also shown is 3 scale training and detection of objects in an image with fixed anchor boxes.

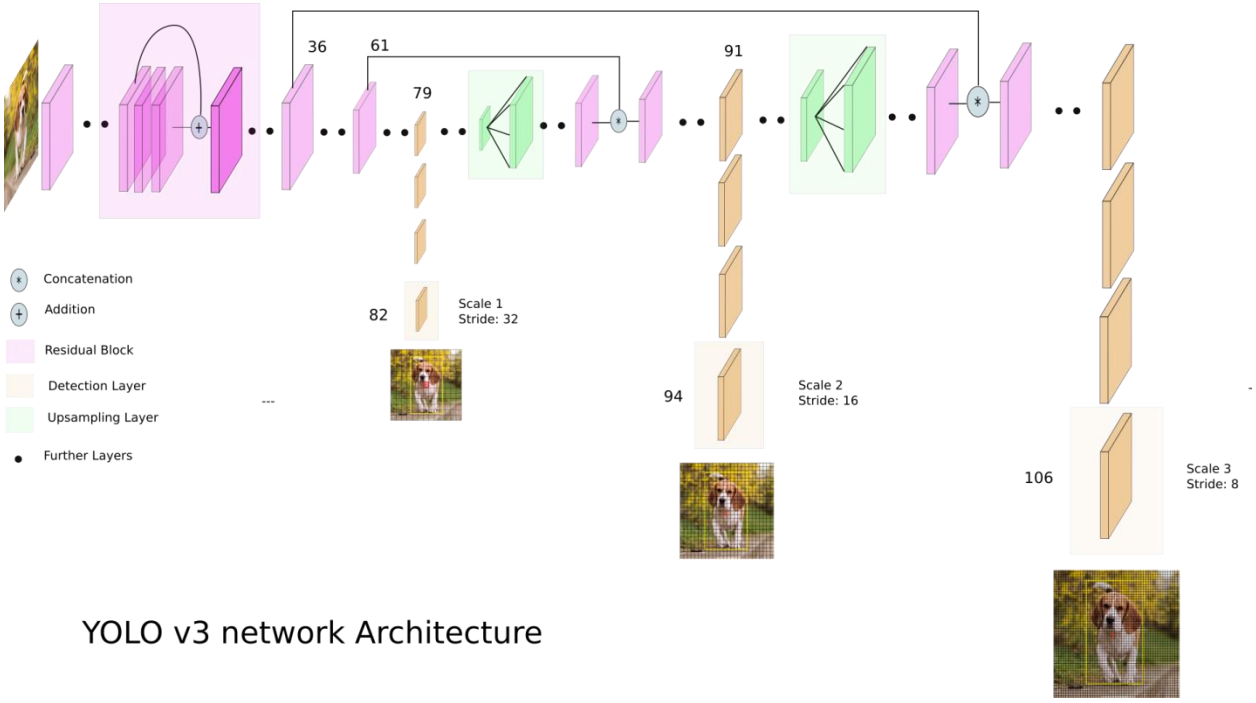

Image 2. YOLOv3 Network Architecture (Source)

There is another reason, besides the fact that YOLOv3 is a CNN, is a good fit for this project. Many CNN architectures utilize softmax for final class prediction. However, YOLOv3 utilizes logistic classifiers, which uses binary cross-entropy loss for the class predictions. Paper publication on YOLOv3 claims that use of logistic classifiers instead of softmax for class prediction is beneficial for complex dataset such as Open Images Dataset. In Open Images Dataset, there are many overlapping labels (i.e. woman and person). Softmax classification assumes that each detection box contains only one class, which is not the case in Open Images Dataset. Therefore, use of logistic classifiers in YOLOv3 fits well for this project.

Also, ensemble of different models will be used for the final prediction file. Firstly, I will use weights pre-trained on COCO dataset to predict 80 of 500 classes. This effectively reduces amount of train images from 1.7 million to 1.5 million. 1.5 million train images will be further reduced to 240k images after data imbalance has been addressed. It was a hard choice to reduce images for training since golden rule in deep learning is to have as much data as possible. However, I learned after a week of training that I did not have capable GPU and enough time to effectively train a model with 1.7 million images over 500 images and get a reasonable result.

Benchmark

I used mAP score obtained from YOLOv3 prediction using weight pre-trained on COCO dataset as a benchmark model. COCO dataset only had 80 classes versus 500 classes of Open Image Dataset. Therefore, it may not be an apple to apple comparison. However, considering difficulty of training a network over 1.7 million training images and 500 classes, I determined using pre-trained weights on COCO dataset was acceptable. I was able to achieve mAP of 6.79% on 99,999 testing images provided by Google AI Open Images – Object Detection Track with the pre-trained weight.

III. Methodology

Data Preprocessing

As mentioned in **Data Exploration** section, there is a serious imbalance in number of train objects in train images. For example, "Man" class has 1,418,594 occurrences over in testing images while there are only 14 occurrences of "Pressure Cooker" in the same testing images. Unfortunately, YOLOv3 using Darknet, which is written in C, did not have data sampling features such as ADASYN and SMOTE. In order to balance train dataset, I manually tweaked testing environment so that model uses more images containing minority classes and uses less images containing majority classes. Also, as mentioned in **Algorithms and Techniques** section, images containing 80 classes that will be covered by weight pre-trained on COCO dataset were removed from training pool. Therefore, after removing data with classes covered by COCO weight and data balance, 1.7 million train images reduced to 243,768 images, which now only contains objects of 420 classes. Below **Diagram 4** shows a pretty good balance of object count of each class compared to imbalance shown in **Diagram 2**, for which log scale had to be implemented. Y-axis shows object count of each class and x-axis represents 420 classes.

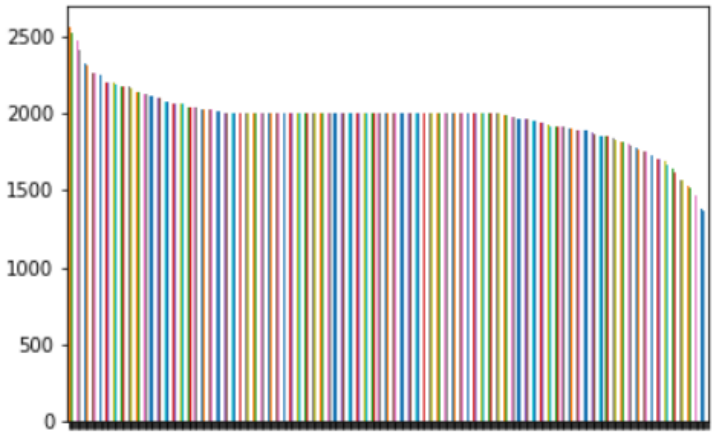

Diagram 2. Train Bounding Box Count After Balancing

After the data processing, max label occurrence of a class is 2,574 while minimum label occurrence of a class is 1340. This ratio between max and min count of occurrences is a huge improvement over 1,418,594 occurrences of “Man” class and 14 occurrences of “Pressure Cooker” as seen before data pre-processing.

Elaborating a little bit more on above data pre-processing, images containing objects with majority classes were simply removed training pool. For images with minority classes, we simply made the model to pick image containing minority classes more often. This is not the best practice since the model can easily overfit to minority classes. However, I’m pretty convinced that this will still be better than having huge data imbalance. On top of that, we have quite handful of data augmentation implemented in YOLOv3 training to fight overfitting.

Implementation

It took more than two weeks before I was able to even start training. The biggest challenge was in setting up libraries and installations to run Darknet and preparing necessary files to start training.

Darknet architecture was developed by Joseph Redmon in C language for Linux platform. I was unsuccessful in running Darknet on my Windows 10 using Ubuntu app, developed by Microsoft to allow one to run Linux commands on Windows. However, I was able to find a GitHub repository that allowed installation of Darknet on Windows using Microsoft Visual Studio.

Next biggest challenge was simply handling massive amount of data and files. For training in Darknet, each image required a text file with class and bounding box information as shown below. Each text file needs to supply class of the object in an image along with its x_center, y_center, width, and height of the object in respect to the image’s actual dimensions.

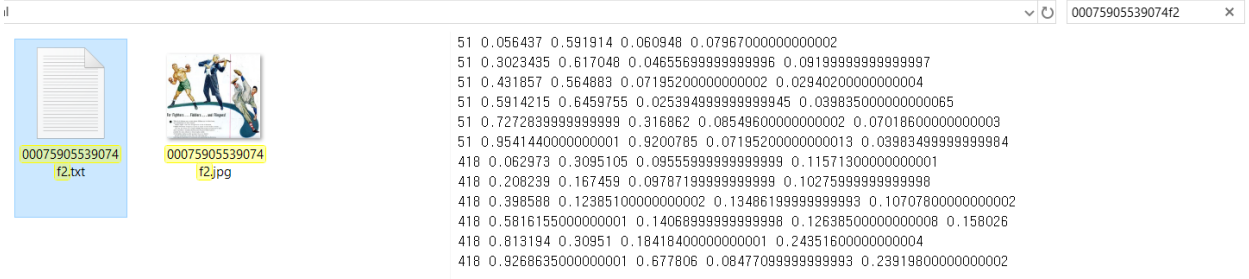

Image 3. Darknet Training Setup

Coding to generate individual text files for each image with multiple likes of class and bounding box information wasn’t too complicated. However, it took 8+ hours to generate these text files using Python code on my hardware using SSD. Initially, I made a mistake of not verifying code’s output in a small scale before pulling the trigger. I only noticed that there was a mistake in

generating these text files when training loss wasn't converging, and paid a hefty toll to fix the text files again. I now cemented a habit in me to always check every code's output in a small scale before implementing in a bigger scale.

Refinement

My initial solution was set up to train on entire imbalanced train dataset over 500 classes. I used weight pre-trained on ImageNet data and implemented fine-tuning method as my initial solution. Fine-tuning is done by freezing earlier layers of weight pre-trained ImageNet. In terms of final results, fine tuning will decrease detection accuracy compared to transfer learning or learning from scratch, but it will speedup training by re-using Convolutional layers trained on ImageNet dataset. I initially wished to pursue this direction because I believed I did not have computing power and resources to obtain meaningful results to complete the project. I trained over 1.6 million train images for 500 classes with fine-tuning method for about 8 days, completing 67800 iterations. Using this weight, I was only able to obtain 1.78% mAP, which was measured using Tensorflow Object Detection API implemented as a part of evaluation in the Kaggle competition.

Intermediate solution was simply to combine predictions from initial solution with the predictions gained from weight pre-trained on COCO dataset over 80 classes. COCO weight was able to result in 6.79% mAP over 500 classes. This was incredible since COCO only predicted 80 classes. Both predictions were merged and resulted in mAP of 8.04%. Result wasn't straight up addition of 1.78% and 6.79% since 1.78% prediction included 80 classes predicted by COCO weight as well.

Final solution was obtained by training over images containing only 420 classes, after 80 classes that will be predicted by weight pre-trained on COCO dataset were removed. Also, class balancing was performed as covered in **Data Preprocessing** section. Just after 24000 iterations of training over 240k train images, I was able to obtain 1.79% mAP over 420 classes. Merging two predictions resulted in mAP of 8.58%, 1.79% + 6.79%, over 100k test images including all 500 classes. Merging predictions, or ensemble of model, was very effective at saving training for a beginner deep learner.

IV. Results

Model Evaluation and Validation

Parameters for Darknet training were mostly kept the same as provided from repo on most part. This was because I was already changing a lot of testing variables such as training image numbers, number of trained classes, existence of frozen layers, and etc. I wanted a solid testing

base before getting deep into fine tuning. One parameter I tweaked was the learning rate. Initial learning rate considered "good" on most cases was 0.001. When I noticed training loss wasn't converging closer to 0, I decreased loss factor by multiplying 0.1 to the learning rate. For sake of discussing some of important parameters used for training, refer to below bullet points:

- Weight Decay = 0.0005
    - Regularization parameter on the network weights to reduces overfitting
- Learning Rate = 0.001
    - Determines by what factor the gradients are updated and affects training speed
- Random = 1
    - Enables training image in different resolutions
- Momentum = 0.9
    - Helps "smooth" the gradient updates that uses a leaky integrator (source)

Finally, robustness of the model was tested by obtaining mAP metric over 500 classes using testing dataset, which are not used in training or validation of models.

Justification

Final model was definitely a better solution than the initial model even though final mAP of my current final model wasn't impressive at all. Initial model only achieved mAP of 1.78% on 100k test dataset even with 8 days of constant training. However, the final ensemble model was able to achieve 8.58% in just 3 days with fairly easy data pre-processing.

At present time, I want to say deep learning in general is an art with a lot of trial and error, not to mention constant experimentation to find the right approach for a certain problem. I don't find 8.58% mAP satisfactory at all compared to solutions developed for different dataset such as COCO and ImageNet, which scored above 50% mAP. However, it is a better solution than developing an algorithm that will detect and always predict one class which will only give 0.2% mAP.

V. Conclusion

Free-Form Visualization

Below **Image 4** and **Image 5** shows predictions made on same picture, but with different models. **Image 4** shows an image with object detection and prediction made using weight pre-trained on COCO dataset, which detected classes "car" and "person".

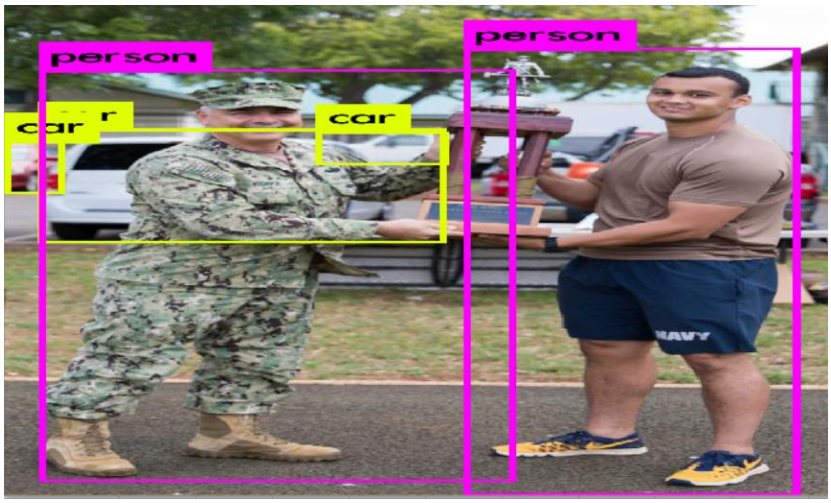

Image 4. Prediction with COCO Pre-trained Weight over 80 Classes

Image 5 shows predictions made on the same picture, but with weight trained over 420 classes not included in COCO dataset.

Image 5. Predictions with Self-Trained Weight on 420 Classes

As you can see, the two models are predicting over different set of classes. Outputs for the same image for the two models were merged and submitted for calculating final mAP.

Reflection

This project was very tough to tackle initially since I did not have much experience with object detection, which required different implementation than classifying an image as a whole. Final solution suggested in this project deviated from my initial plan since I ended up using ensemble of the two models. I did not mention about real time object detection in previous discussions, but YOLOv3 excels at real time object detection. Using Pascal Titan X as a GPU, it is able to process images at 30 FPS and have a mAP of 57.9% on COCO dataset per developer of YOLOv3

(YOLO). It would have been a great bonus to the final model if I was able to achieve a good mAP using just one model.

Improvement

One improvement, which builds up on idea of using ensemble of models, is to break up classes into different clusters depending on how many images are available for each class. Earlier, we noticed that huge imbalances existed between 500 classes, ranging from just 14 occurrences of "Pressure Cooker" class versus whopping 1,418,594 occurrences of "Man" class. By grouping different classes with different object occurrences, we minimize our dependence data pre-preprocessing techniques such as under-sampling or over-sampling which have their pros and cons.

My proposal of breaking classes into different clusters and training separate models is well backed by **Chart 1** below which shows mAP vs iteration of the two models.

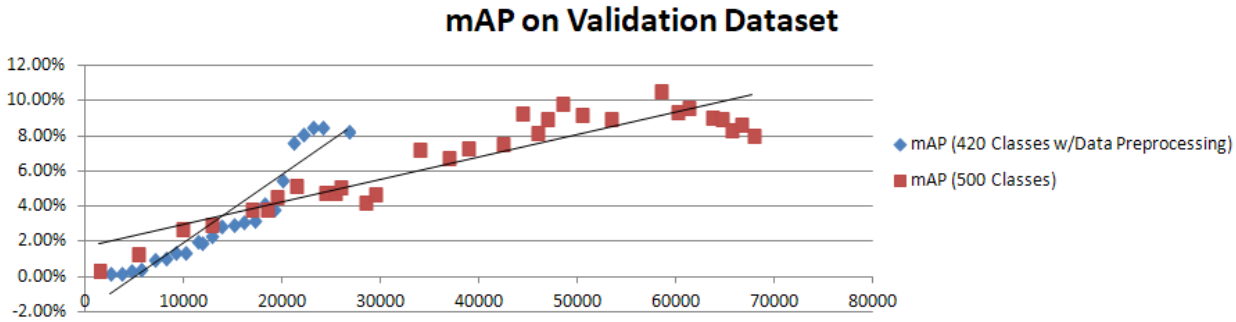

Chart 1. mAP on Validation Dataset vs Iteration

Red squares show mAP obtained on validation dataset using a model that was trained on imbalanced train dataset over 500 classes, which took 60000 iterations or about 9 days to train before it started to overfit to the train data. On contrary, model trained over balanced train dataset over 420 classes only took about 3 days to train before it overfit to the train dataset. mAP on same validation dataset is very comparable between the two models. It will take a little bit more coding and data preprocessing, but I strongly believe that training separate models with classes clustered by their object count will yield a very successful model.

Reference

Hypercitations are used in text