Machine Learning Engineer Nanodegree

Object Detection on Command

Seyeon Lee
September 14th, 2019

I. Definition

Project Overview

Object detection is a technology that detects objects, classify, and locate them in an image. Many ideas have been proposed for this task. Region proposals is a method that allows Convolutional Neural Network, also known as CNN, to detect objects among many cropped regions with help of selective search (Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013)[1]. CNN is a state-of-the-art algorithm for computer vision that works with image pixels and the data they inherit. By detecting edges, depth, colors, and many other features of images, CNN can identify shapes and motions. Models like RCNN (Region-based Convolutional Neural Networks)[2] and YOLO (You Only Look Once)[3] make use of these algorithms and accomplish high accuracy on object detection tasks.

This project will explore different CNN models using transfer learning, a method that uses a pre-trained neural network and adapt the model to perform different tasks with different dataset, and design a model that can find different objects and locate them with precision. PASCAL VOC (Visual Object Classes) 2012 dataset[4] will be in use of training models, and the model will be tested on the validation set provided by PASCAL VOC Challenge.

Problem Statement

The goal is to build a program that intakes video data and do the following tasks:

1. Identify objects with labels from PASCAL VOC (Visual Object Classes) dataset.
2. Localize object by drawing bounding boxes around the object.

[1] J. R. Uijlings , K. E. Sande , T. Gevers , A. W. Smeulders, Selective Search for Object Recognition, International Journal of Computer Vision, v.104 n.2, p.154-171, September 2013 [doi>10.1007/s11263-013-0620-5]

[2] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*. doi:10.1109/cvpr.2014.81

[3] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:10.1109/cvpr.2016.91

[4] Everingham, M. et al, "The {PASCAL} {V}isual {O}bject {C}lasses {C}hallenge 2012 {(VOC2012)} {R}esults", http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html

This is an object detection task. It will classify 20 objects from PASCAL VOC datset: person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, and tv/monitor. Then, the bounding boxes of each object in a test image will be predicted and output. I will use transfer learning with CNN architecture, ResNet50, with additional and trainable layers of the combination of Conv2D, BatchNormalization, and Activation layers.

Metrics

PASCAL VOC documentation suggests to use mAP (mean Average Precision) method for object detection models.

> The AP summarizes the shape of the precision/recall curve, and is defined as the mean precision at a set of eleven equally spaced recall levels [0,0.1,...,1]

In order to calculate Average Precision for each class, being an unbalanced supervised learning dataset, this object detection model can be measured in terms of precision and recall. Parameters are True Positives, False Positives, True Negatives, and False Negatives.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

II. Analysis

Data Exploration

This dataset contains large number of images for classification and detection task with 20 different classes: bird, cat, cow, dog, horse, sheep, airplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, tv/monitor, and person.

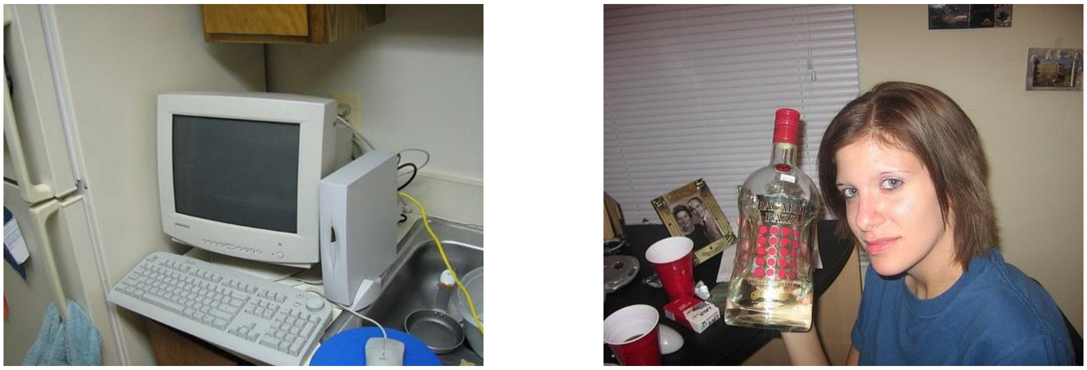
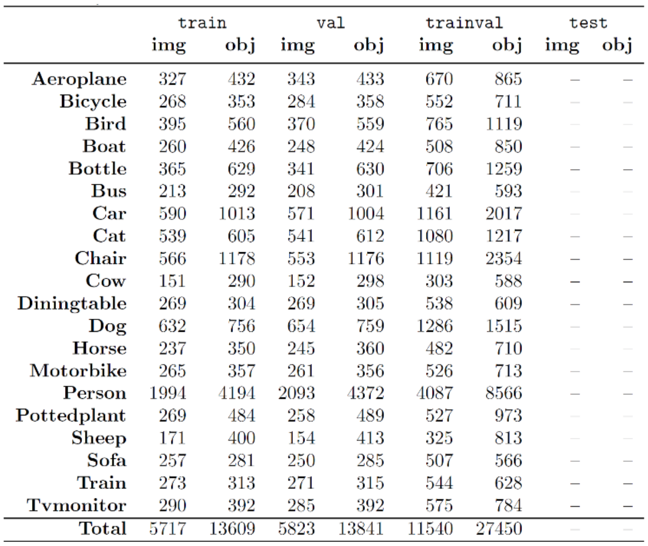


Figure 1 Images from the PASCAL VOC 2012 dataset

Fundamentally a supervised-learning, the training set of labelled images is provided. Below is the table that includes statistics of the image sets for each 20 classes shown on the official PASCAL VOC 2012 documentation. You can see it is an imbalanced dataset.

Table 1 Statistics of the main image sets. Object statistics list only the 'non-difficult' objects used in the evaluation.[5]



	train		val		trainval		test	
	img	obj	img	obj	img	obj	img	obj
Aeroplane	327	432	343	433	670	865	–	–
Bicycle	268	353	284	358	552	711	–	–
Bird	395	560	370	559	765	1119	–	–
Boat	260	426	248	424	508	850	–	–
Bottle	365	629	341	630	706	1259	–	–
Bus	213	292	208	301	421	593	–	–
Car	590	1013	571	1004	1161	2017	–	–
Cat	539	605	541	612	1080	1217	–	–
Chair	566	1178	553	1176	1119	2354	–	–
Cow	151	290	152	298	303	588	–	–
Diningtable	269	304	269	305	538	609	–	–
Dog	632	756	654	759	1286	1515	–	–
Horse	237	350	245	360	482	710	–	–
Motorbike	265	357	261	356	526	713	–	–
Person	1994	4194	2093	4372	4087	8566	–	–
Pottedplant	269	484	258	489	527	973	–	–
Sheep	171	400	154	413	325	813	–	–
Sofa	257	281	250	285	507	566	–	–
Train	273	313	271	315	544	628	–	–
Tvmonitor	290	392	285	392	575	784	–	–
Total	5717	13609	5823	13841	11540	27450	–	–

The annotations for each image are consisted of the following fields:

- width: width of the frame
- height: height of the frame
- Nobj : The number of objects in the frame
- fileID: The png file name
- For each frame, there are at most 56 objects in one frame. The infomation of the ith object is:
    - bbx_i_nm: The type of object inside of the bounding box i e.g.,
    - bbx_i_xmin: The x coordinate of the minimum corner in bounding box i
    - bbx_i_ymin: The y coordinate of the minimum corner in bounding box i
    - bbx_i_xmax: The x coordinate of the maximum corner in bounding box i
    - bbx_i_ymax: The y coordinate of the maximum corner in bounding box i

Exploratory Visualization

Since PASCAL VOC is already a well-known dataset amongst data scientists and programmers, a lot of analyzations have been done. I chose two plots from this blog (https://fairyonice.github.io/Part_1_Object_Detection_with_Yolo_for_VOC_2014_data_anchor_box_clustering.html) by Yumi. One is a histogram of the number of objects per image and the other is the number of each object of the twenty classes in the dataset. These are helpful figuring out how well and balanced each class will be trained and predicting the rank of AP scores for each class.

[5] Everingham, M. et al, 2012

Each image in the training/validation set contains one or more objects. The histogram below shows the number of objects per image.

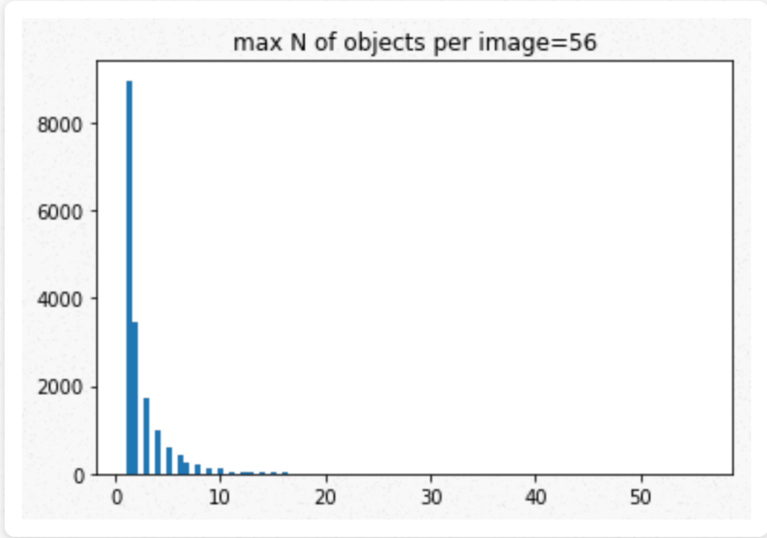


Figure 2 You can see the plot is heavily right-tailed. Maximum number of objects per image is 56, but the number of objects per image ranges between 0 and 20. Most of the images contain only one object per image.

While the plot above shows the number of objects per image, the one below will deal with the number of objects per class.

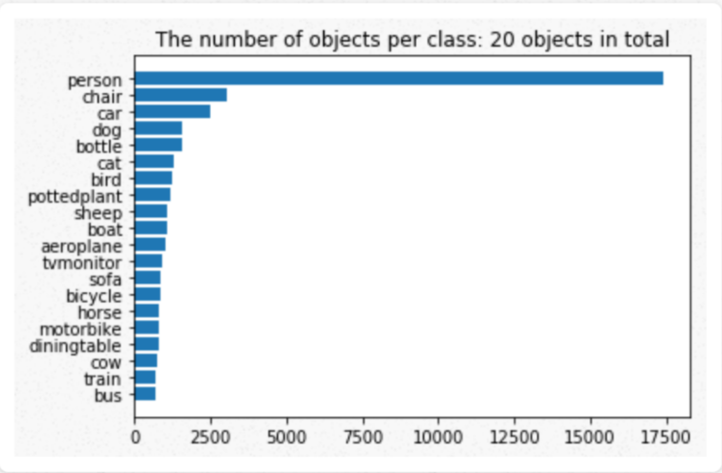


Figure 3 The number of objects per class. There are 20 objects in total, and "person" class is the largest with 17401 objects in this dataset. Rest of the classes have 500 – 3100 objects. This means the model will perform better at detecting persons than any other objects.

Algorithms and Techniques

Object Detection task being a famous application of machine learning, several mehtods have been suggested, such as RCNN[6], Fast – RCNN[7], YOLO[8], and so on. The very feature that encompasses all these methods is called Convolutinal Neural Network, a neural network that is specifiacally designed for computer vision tasks. Convolutional Neural Network, also known as CNN, is a state-of-the-art algorithm for computer vision that was inspired by animals' visual cortex.[9] CNN works with image pixels and the data they inherit. By detecting edges, depth, colors, and many other features of images, CNN can identify shapes and motions. CNN usually deals with three information of an image: x and y coordinates of a pixel and the color (0 -255). By a small filter with certain kernel size strides through an original image, CNN detects any difference in pixels and pixels around.

One of Keras Convolutional layers that was used the most in this project, Conv2D, entails following parameters (decription referred from TensorFlow documentation[10]):

- **filters**: the number of output filters
- **kernel_size**: the height and width of the 2D convolution window
- **strides**: strides of the convolution along the height and width
- **padding**: one of "valid" or "same". Decide whether to exclude or include the very boundary pixels.
- **data_format**: A string, one of channels_last (default) or channels_first
- **dilation_rate**: the dilation rate to use for dilated convolution
- **activation**: Activation function to use.
- **use_bias**: whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the kernel weights matrix.
- **bias_initializer**: Initializer for the bias vector.
- **kernel_regularizer**: Regularizer function applied to the kernel weights matrix.
- **bias_regularizer**: Regularizer function applied to the bias vector.
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation")..
- **kernel_constraint**: Constraint function applied to the kernel matrix.
- **bias_constraint**: Constraint function applied to the bias vector.

Since this project is focused on object detection on a real-time video, the processing time of an algorithm is a key element. When they were trained on PASCAL VOC 2007 dataset, the number of frames on which could conduct object detection per second was 0.5 for R-CNN, 5 seconds for Fast R-

[6] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2016). Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence,38*(1), 142-158. doi:10.1109/tpami.2015.2437384

[7] Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence,39*(6), 1137-1149. doi:10.1109/tpami.2016.2577031

[8] Redmon, J., et al (2016). You Only Look Once: Unified, Real-Time Object Detection.

[9] Fukushima, Kunihiko (1980). "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position" (PDF). Biological Cybernetics. 36 (4): 193–202. doi:10.1007/BF00344251. PMID 7370364.

[10] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

CNN Resnet, and 67 for YOLO v2 416*416.[11] Normal video data has a frame rate of 24 frames per second (FPS) to 30 FPS. This is why YOLO was chosen as an algorithm to achieve object detection on a video. Samely, we will divide images into grids, run regression, and will also utilize the conpet of "Anchor boxes." This conept will be dealt in detail in "Implementation."

In order to use YOLO, following parameters should be considered.

- Batch Size (the number of images to look during a single training step)
- Image Height (pixels of an image height as an output)
- Image Width (pixels of an image width as an output)
- Grid Height (the number of the grids in height)
- Grid Width (the number of the grids in width)
- Anchor Box (explained in Implementation chapter)
- Neural Networks
    - Number of Layers
    - Filters
    - Kernel Size
    - Layer Types

Benchmark

For the benchmark model, there is an example detector that is provided by the offical PASCAL VOC 2012 website, which is trained on PASCAL VOC 2012 training set. It is then applied to validation set, and the result is also provided in the development kit of the dataset.

> *"The file example detector.m contains a complete implementation of the detection task. For each VOC object class a simple (and not very successful!) detector is trained on the train set; the detector is then applied to the val set and the output saved to a results file in the format required by the challenge; a precision/recall curve is plotted and the 'average precision' (AP) measure displayed."*

Table 2 Result for the benchmark model -- AP for each class (mean Average Precision (mAP) = 0.00105)

Class	Aeroplane	Bicycle	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow
AP	0.006	0.000	0.000	0.000	0.000	0.001	0.000	0.005	0.000	0.000
Class	Diningtable	Dog	Horse	Motorbike	Person	Pottedplant	Sheep	Sofa	Train	Tvmonitor
AP	0.001	0.003	0.001	0.001	0.002	0.000	0.000	0.001	0.001	0.000

[11] Hui, Jonathan (2019, March 26). Object detection: Speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and... Retrieved from https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

III. Methodology

(approx. 3-5 pages)

Data Preprocessing

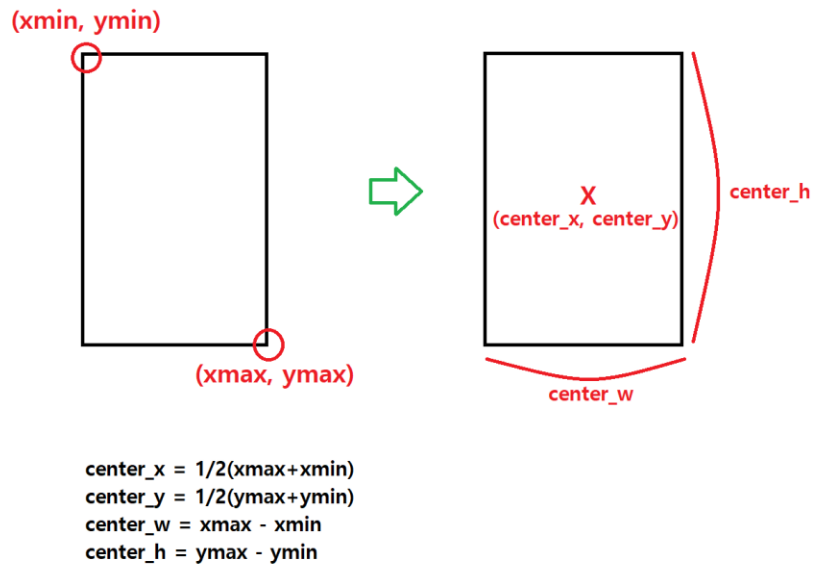
Data Preprocessing for the Object Detection task in this project follows the steps below.

1. Randomize the images.
2. Parse annotation.
3. Input/output Encoding

Parse Annotation: The program reads in xml files for each image data that includes the parameters mentioned above (Data Exploration). This step sorts out the information for each parameter and put them in arrays called “train_image” and “seen_labels”. This step makes the specific information for each image easily accessible for the program.

Input Encoding: YOLO reads in pre-specified shape of images. For each image input, no matter what size they portrait, they will be encoded into (208, 208, 3). Meaning, they will be input as a form of 208 * 208 pixels with color. Parameters xmin, xmax, ymin, and ymax will be adjusted accordingly.

Output Encoding deals with mathematical ways of expressing bounding boxes. PASCAL VOC dataset annotation method is mentioned above. However, in order to make it easier to find an object in gird cells, we need to give a little modification on the original annotation. This undergoes the process that is illustrated below.



center_x = 1/2(xmax+xmin)
center_y = 1/2(ymax+ymin)
center_w = xmax - xmin
center_h = ymax - ymin

Figure 4 Bounding Box Encoding.

This step leads to the next step where it rescales the parameters with respect to 7*7 grid cells. All the parameters are rescaled regarding each grid cell a unit cell with the side of 1. Rather than depicted in pixels, the size of bounding boxes are rescaled relatively to the unit cell. For example, if there is an object with the center width of 200 pixels, the new value for its width will be:

$$200 \Big/ \frac{208\ (pixels\ of\ one\ side\ of\ an\ original\ input\ image)}{7\ (grid\ height\ or\ width)} = \frac{200 * 7}{208}$$

Implementation

YOLO algorithm has its inherit CNN model. This project, however, attempts to utilize another pretrained model. In this case, we use Resnet50 model trained with Imagenet training data. The implementation process includes several sub-steps.

a. Parsing annotations.
b. Input/Output Encoding
c. Generate Batches
d. Model Architecture
e. Loading pre-trained Resnet50 model
f. Define the loss function
g. Train the network, calculate mAP

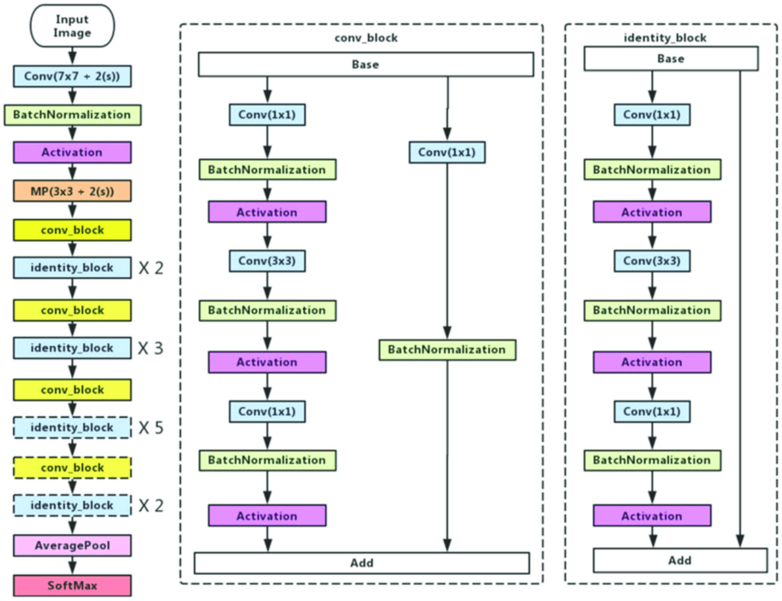


Figure 5 Architecture of Resnet 50. It is mainly composed of Conv2D, Batch Normalization, Activation, and Pooling layers. Illustration by Ji, Qingge [12]

[12] Ji, Qingge, et al. "Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images." Algorithms, vol. 12, no. 3, 2019, p. 51., doi:10.3390/a12030051.

While YOLO algorithm has its own architecture, I decided to use transfer learning with the base model of Resnet50. There were many other choices like VGG16 or Xception, but because Resnet50 performed better in first few epochs, it served as a base model with some modifications. This Resnet50 was pre-trained with Imagenet dataset. Therefore, all they layers in Resnet50 were set untrainable.

Four layers of Conv2D and three Batch Normalization and Activation layers each are added to the final identity block of the Resnet50 model, excluding the AveragePool layer. The final layer localizes and classifies objects in a 7*7 grid cell. Our encoded input images are 208 * 208 size with color, so the input layer intakes the tensor shape of (208, 208, 3), and the output layer produces (7, 7, 25) tensors as an output. After the training, this model will undergo evaluation stage to produce Average Precision score for each class, hence the mAP score in the end.

Refinement

The initial result of this model was uncalculatable. Loss diverged and AP scores fluctuated over the classes inconsistently. For an attempt to fix this, following adjustments were made.

1. Batch size was increased from 32 to 64 prevent the loss from diverging.
2. Learning rate is lowered since it seemed to contribute to the divergence of the loss function.
3. Trainable layers were simplified to one single layer of Conv2D, which eventually fixed loss nan problem.
4. Anchor Boxes are added.

The concept I have come across with during the research for project was called “anchor box.” In this github forum (https://github.com/pjreddie/darknet/issues/568), username “vkmenon” defines anchor boxes as follows: “…most bounding boxes have certain height-width ratios. Instead of directly predicting a bounding box, YOLOv2 (and v3) predict off-sets from a predetermined set of boxes with particular height-width ratios - those predetermined set of boxes are the anchor boxes.” By doing this, it expected to achieve the following benefits:

1. Reduce training / processing time since the program eliminates the need to look for every pixel for a potential object.
2. Making real-time object detection feasible.
3. Restricted selection of candidate image patches containing an object reduces great amount of calculation for GPU.

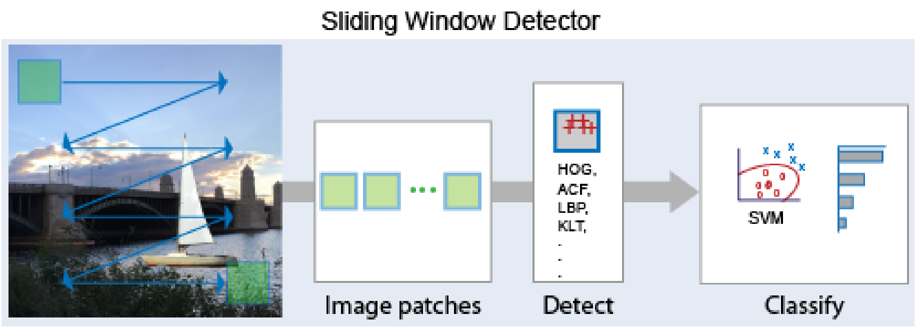


Figure 6 How Anchor Boxes work. *https://www.mathworks.com/help/vision/ug/anchor-box-basics.html*

After these refinements, the model started to train with consistent decrease of the loss value. After 37 epochs, being early stopped by three consecutive non-improvement of the loss, the model achieved the loss of 9.46433, all the way from 47569982.09983. Though it was clearly a great improvement for this model, when evaluated, the mAP score remained under 0.03 for some reason.

IV. Results

Model Evaluation and Validation

A validation set was used as a test set since no data from the validation set was used during the training.

As it was mentioned in Refinement chapter, my original model was having a serious problem with loss nan error. It led to numerous modifications on the model, which includes learning rate, batch size, optimizer, model architecture, and even the introduction of “anchor boxes.”

Learning rate was seriously decreased from 0.01 to 0.5e-4. Batch size was increased from 32 to 64. After trying out all Adam, SGD, and RMSprop, Adam seems to produce the most reasonable and consistent learning. Hence, Adam was chosen as an optimizer. However, this might have differed due to the parameters they entailed. 4 layers of combination of Conv2D, BatchNormalization, and Activation layers were reduced to one layer of Conv2D. This modification eventually solved the loss nan error.

Shape of input data was (416, 416, 3). However, in order to reduce time for training and operation, I have decreased the dimension to (208, 208, 3), and it worked well both times. This induced some deterioration in performance indeed.

Robustness of the model is not guaranteed since I have not tried image augmentation nor training on other datasets. Though it cannot be said with certainty, I am positive it would sustain its performance level due to the use of transfer learning. Resnet50 model I used is a model provided by Keras that is pre-trained on a massive Imagenet dataset. It will be able to maintain its ability to discern patterns in various circumstances.

The credibility of the model is supposed to be very high for the algorithm with anchor boxes is a well-known, state-of-the-art object detection mechanism called YOLO13. This project differs from the actual research with its model architecture and the way data are pre-processed, but the very backbone of this project, YOLO, is already a proven method in Object Detection tasks. However, the low mAP score of my model cannot promise trustworthy result.

Justification

Compared to the benchmark model, my model did not achieve significant improvement in mAP score. However, this may have been a problem in my evaluation metric system, loss function, or somewhere other. As you can see from Table 3, it failed to perform better than the benchmark model. Regarding the

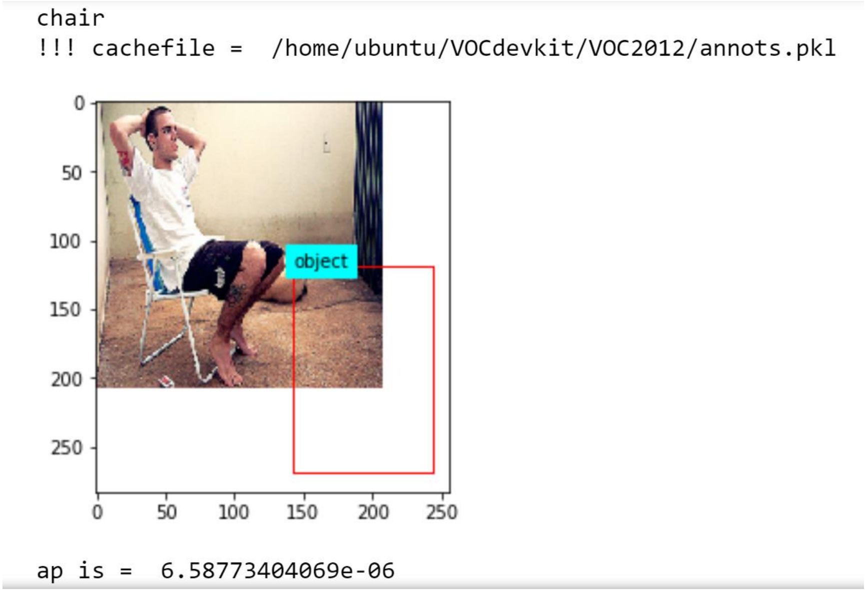
[13] Redmon, J., et al (2016).

training caused the loss value to decrease every epoch and batch, it seems like there was an error either in transforming raw data of prediction into actual coordinates or in evaluation stage.

Table 3 Result for the final model -- AP for each class (mean Average Precision (mAP) = 3.64e-06)

Class	Aeroplane	Bicycle	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow
AP	7.04e-07	0.00	1.91e-07	8.98e-06	0.00	2.14e-06	3.59e-06	0,00	6.59e-06	1.95e-05
Class	Diningtable	Dog	Horse	Motorbike	Person	Pottedplant	Sheep	Sofa	Train	Tvmonitor
AP	0.00	1.54e-05	0.00	0.00	5.49e-06	5.58e-08	5.34e-06	1.20e-07	1.10e-06	0.00

To discern this issue, I visualized my prediction. The particular example above (Figure 7 below) shows that the model being successful in classification but failed in localization. Another issue you can observe is that the bounding box is located outside of the 208 * 208 frame. This makes it highly suspicous that the prediction is not correctly interpreted into actual pixel coordinates.



```
chair
!!! cachefile =  /home/ubuntu/VOCdevkit/VOC2012/annots.pkl
```

```
ap is =  6.58773404069e-06
```

Figure 7 Model predicting object location and classification

V. Conclusion

Free-Form Visualization

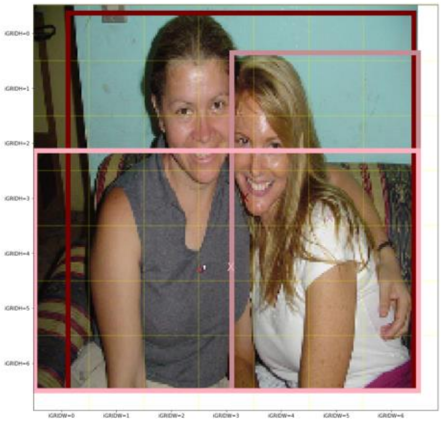


*Figure 8 Visualization of a batch. In this particular image, bounding boxes are drawn around two people and a sofa. As you can see this image is divided into 7*7 grid cells.*

After parsing annotations, a batch generator create batches from the training dataset. You can choose how many grid cells we want and the shape of the output data. In this case, I divided each image into 7 * 7 grid cells as you can see from above. The input size is 208 * 208 as I rescaled during the data pre-processing stage.

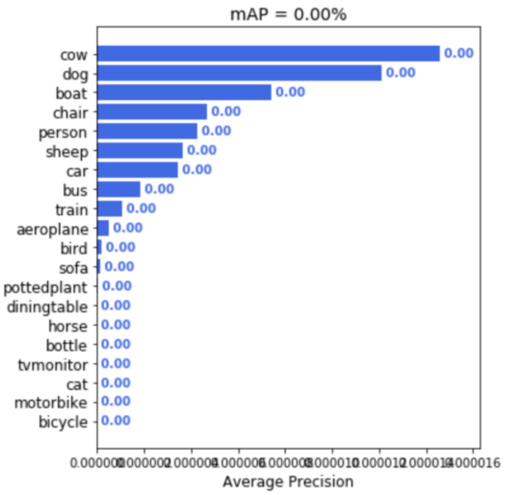


Figure 9 Visualization of AP scores of each class.

This visualization of the data implemented the code written by João Cartucho on his github (https://github.com/Cartucho/mAP). After calculating precision and recall based on if the iou score exceeds 0.5 or not, the area under the precision-recall graph is calculated to get AP score. After summing all the AP scores from each class, it was divided into the number of classes of PASCAL VOC 2012 and acquired the mAP score. Due to some unknown problems, AP scores turned out very small as you can see from the graph above. However, it clearly shows that each class had different AP scores with heavily tailed distribution.

Reflection

The steps that were taken in order to accomplish the object detection task is can be divided as follows:

- Step 0: Understanding PASCAL VOC 2012 dataset
- Step 1: Data Pre-processing
- Step 2: Batch Generator
- Step 3: Model Architecture
- Step 4: Custom Loss Function
- Step 5: Training
- Step 6: Evaluation

In Step 0, I have parsed annotations from the dataset given and divided training and validation set. Here, I also analyzed the size of each image and counted the number of images. In Step 1, I encoded the input and output. Various sizes of images was rescaled down to 208 * 208 consistently (original plan was to have them 416 * 416, but in order to shorten the training time, dimension was halved.), and the pixel coordinate of the bounding boxes were adjusted according to the new size of images. The way to express bounding boxes, which used to be (xmin, xmax, ymin, ymax), were changed to the form of (central x, central y, width, height) of the bounding boxes. In Step 2, a batch generator was designed. It intakes 208*208 images and outputs (7, 7, 25) tensors which are accounted for 7 * 7 grids with the information about the bounding boxes and an object classification. In Step 3, I utilized the pre-trained Resnet 50 model whose weights were pre-calculated through Imagenet dataset. One more Conv2D layer was added on this base model. The shape of the layer was (batch_size, 7, 7, 4, 25). I found Step 4 and 5 most challenging since the model kept failing to throw the loss that is not nan. The extremely bulky training dataset, hence the slow speed of training have cost me so much time for trials and errors. Parameters for Adam optimizer were adjusted constantly to make the loss not diverge. In step 6, I collected mAP score to evaluate the model.

The interesting aspect I found was that though it achieved great improvement in getting low loss value, the result may be very different. It would be deceitful to state that it was not disappointing at all, but watching the model gets better every epoch gave me a heart-warming experience, especially after struggling so hard to get it run.

Improvement

Improvement I would like to make in this project will be real-time object detection. Not only processing still and expected data of images, real-time video calculation will require a lot faster operation speed. Running this program on a mobile phone and make it accessible in daily use will emancipate users from limited practicality. Not only finbd objects from videos from a steady web-cam, the program will extend its use through various applications of mobile phones that are capable of dynamic physical movements.

Another improvement can be made in the model architecture. I simplified my output model into one layer due to a loss nan error. However, a model architecture that is well compatible with YOLO custom loss function will perform better in this task and, maybe, in processing time, also.