University of Michigan
EECS 504: Foundations of Computer Vision
Winter 2020. Instructor: Andrew Owens.

Problem Set 6: Object Detection

Posted: Tuesday, February 18, 2020 **Due:** Tuesday, February 25, 2020

Please submit your solution to Canvas as a notebook file (`.ipynb`), containing the visualizations that we requested. Your `.ipynb` notebook should be named as `<unique_name>_<umid>.ipynb`. Example: `adam_01100001.ipynb`. Also, please remember to put your name and unique name in the first text block of the notebook.

Credit: This problem was originally developed by Justin Johnson and the course staff for EECS 498/598.

The starter code can be found at:
https://drive.google.com/open?id=1BSv7dIoDd_ZKtmPJzmQjn-3Zp9rh9P5M

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 6.1 *Object Detection*

In this problem set, we will implement a **single-stage** object detector, based on YOLO v1 [1] and v2 [2]. Unlike the (better-performing) R-CNN models, single-stage stage detectors predict bounding boxes and classes without explicitly cropping region proposals out of the image or feature map. This makes them significantly faster to run.

We'll train and evaluate our detector on the PASCAL VOC dataset, a standard dataset for object detection tasks. The full dataset contains a total of 11K train/val data images with 27K labeled objects, spanning 20 classes (Figure 1). To make training faster, though, we will only use a subset that contains 2.5K images.

Below, we outline the steps in the object detection pipeline and the modules that you will be implementing. You should refer to the comments in the provided notebook for further implementation details.

1. We will use MobileNetv2 [3] as our backbone network for image feature extraction. This is a simple convolutional network intended for efficient execution. We have removed the final fully connected layers so that the network will output feature maps instead of class predictions. To speed up training, we've *pretrained* the network to solve ImageNet classification. This is already implemented for you.

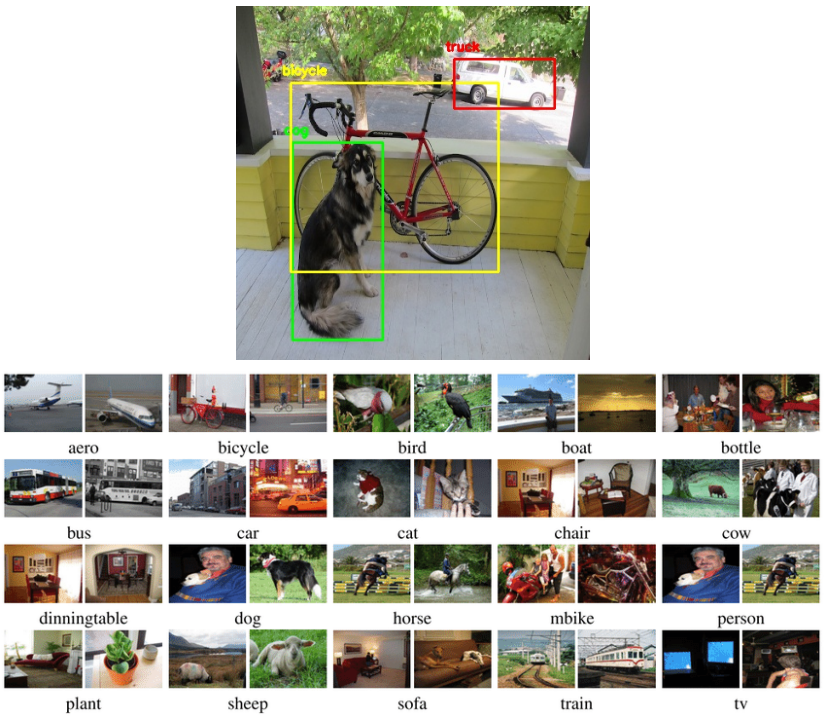

Figure 1: Example images and detections from the PASCAL VOC dataset

2. As in YOLOv2, we'll start by generating a fixed set of *anchor boxes*, each of which we will use as a possible location for an object. At each point in the 7×7 feature map from the backbone network, we will consider a set of anchor boxes of different sizes and shapes (Figure 2). For each anchor box, the prediction network will classify the object contained in the region. You are responsible for implementing the **anchor box and proposal generation** code. (3 points)

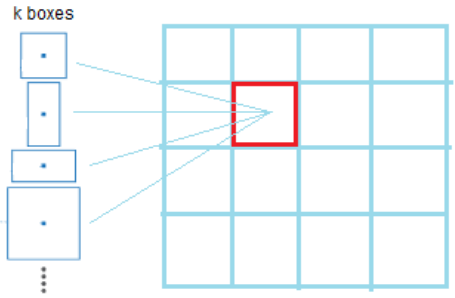

Figure 2: Anchor boxes of varying sizes and shapes

3. You will implement the **prediction network** which inputs features from the backbone network and outputs a set of scores and transformations for each anchor box [See Figure 3] (3 points):

   (a) transformation to convert from anchor box to region proposal (4 numbers)

   (b) confidence score representing the likelihood that an anchor contains an object

   (c) probability score for each of the C classes

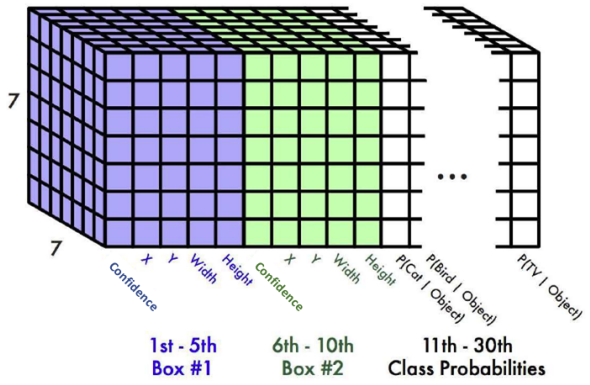

Figure 3: Output of the prediction network. Image credit: towardsdatascience

4. Define the loss function. We'll regress the three quantities: confidence score, bounding box offset, and class score. We provided the code for computing each of these losses individually, but you will need to compute a weighted sum for the total loss.

5. Implement the **forward pass** of the object detector. The forward pass inputs the images and ground-truth bounding boxes and returns the loss for the minibatch. The forward pass should encapsulate all the previous modules you've written (feature extraction, anchor and proposal generation, etc). (3 points)

6. **Non-maximum suppression** (NMS) is a technique for removing overlapping/redundant bounding box predictions. In non-max suppression we consider the bounding boxes with the highest confidence values first and iteratively eliminate boxes with IOU greater than some threshold value. We do not require you to implement this; however please read the documentation to understand how the algorithm works.

7. The inference step will input validation images and predict bounding boxes, confidence scores, and class labels for each object in the image. Low confidence and repeated bounding boxes will be filtered by thresholding and NMS. This has already been implemented for you.

8. To evaluate the performance of your detector, we will use the **mean Average Precision (mAP)** metric. mAP is computed by taking the average of precision over all the classes in the dataset. For your convenience, we've also provided code to compute mAP, so you are only required to run the inference code and confirm that your detector achieves around 11% mAP on the validation dataset. (1 point for meeting mAP threshold)

Optional: As a challenge, try to improve the performance of your object detector. You can try increasing the number of samples in the training set or train the model for more epochs. Additionally, you can try to increase the resolution of the input by increasing the feature map dimension from 7 x 7 to 13 x 13. For more advanced techniques to try, refer to Figure 4 taken from the YOLO v2 [2] paper. It is possible to achieve 18-20% mAP using this implementation of the YOLO detector with basic modifications.

Since this homework contains many modules, we've provided sanity checks for each part so that you can verify the implementation. It is important that you pass all the sanity checks to ensure correctness and for the network to achieve the desired performance. Finally, it should

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Figure 4: Design choices for YOLO v2 leading to performance boost

take you around 30 minutes to train the final network so feel free to take a break while it is running and check on it half an hour later.

Some tips and resources:

1. Lecture slides!
2. YOLOv2 paper [2] and tutorial by Joseph Redmon.
3. Lilian Weng's object detection tutorial

References

[1] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 2016.

[2] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CVPR*, 2017.

[3] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.