University of Michigan
EECS 442: Computer Vision
Fall 2021. Instructor: Andrew Owens.

Problem Set 7: Object Detection

Posted: Wednesday, October 27, 2021	**Due:** Wednesday, November 10, 2021

Please convert your Colab notebook to a PDF file and submit the PDF file to Gradescope. We have included the PDF conversion script at the end of the notebook. Nothing needs to be submitted to Canvas.

The starter code can be found at:
https://colab.research.google.com/drive/1M03ebaBi4wztUIFlcJPKBxEneW5FmWfk?usp=sharing

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

Problem 7.1 *Object Detection*

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

We've given you the code for the object detection system, but we've left a few key functions unimplemented. Your task is to 1) understand the model and code and 2) fill in these missing pieces. Consequently, this problem set will require you to read significantly more code than in previous problem sets. However, the amount of code you actually write will be comparable to prior problem sets.

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/validation data images with 27K labeled objects, spanning 20 classes (Figure 1).

Below, we outline the steps in the object detection pipeline and the modules that you will be implementing. The instructions here are not exhaustive, and you should refer to the comments in the provided notebook for further implementation details. Also, we encourage you to not necessarily jump directly to the part of the notebook that requires you to write code — instead, we recommend first reading the comments in the system we've provided, and to understand the utility of each function by understanding their inputs and outputs.

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

(a) We will use MobileNetv2 [2] as our backbone network for image feature extraction. This is a simple convolutional network intended for efficient computation. To speed up training, we've *pretrained* the network to solve ImageNet classification. This is already implemented for you. **(0 points)**

(b) After passing the input image through the backbone network, we have a convolutional feature map of shape $(D, 7, 7)$ which we interpret as a 7x7 grid of $D-$dimensional features. At each cell in this grid, we'll predict a set of A bounding boxes.

The format of these bounding boxes is as follows: consider a grid with center (x_c^g, y_c^g). The *prediction network*, that you will fill in later will predict offsets (t^x, t^y, t^w, t^h) with respect to this center. By applying this transformation, we get the bounding box or proposal with center, width, and height (x_c^p, y_c^p, w^p, h^p). To convert the offsets to the actual bounding box parameters, read the instructions in the notebook to implement the `GenerateProposal` function. **(2 points)**

(c) Your next task is to implement the `IoU` function, which estimates the intersection-over-union (IoU) of two bounding boxe (also called the *Jaccard similarity*). Recall that IoU for two windows W_1 and W_2 is:

$$\text{IoU}(W_1, W_2) = \frac{|W_1 \cap W_2|}{|W_1 \cup W_2|}, \tag{1}$$

where $\cap$ and $\cup$ are set intersection and union operations, respectively. This function will be used later in the object detection module to calculate the IoU between the predicted and ground-truth bounding boxes. **(3 points)**

(d) The prediction network takes the features from the backbone network as input, and outputs the classification scores and offsets for each bounding box.

For each position in the 7×7 grid of features from the fully convolutional network, the prediction network outputs C numbers to be interpreted as classification scores over the C object categories for the bounding boxes with centers in that grid cell.

In addition, for each of the A bounding boxes at each position, the prediction network outputs

offsets (4 numbers, to represent the bounding box) and a *confidence score* (where large positive values indicate high probability that the bounding box contains an object, and large negative values indicate low probability that the bounding box contains an object), as shown in Figure 2. Please refer to the notebook for further instructions, and implement the `forward` function of the `PredictionNetwork` class. **(3 points)**

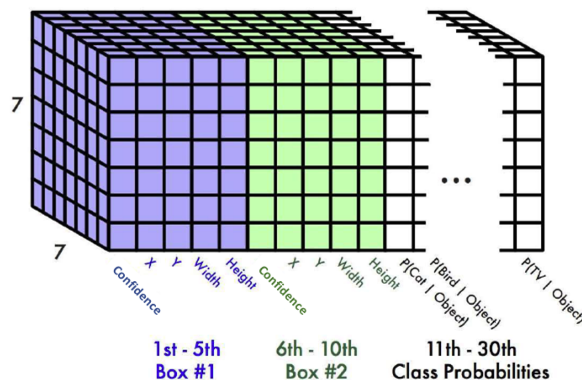

Figure 2: Output of the prediction network. Image credit: towardsdatascience

(e) 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. **(0 points)**

(f) The forward pass of the object detector inputs the images and ground-truth bounding boxes and returns the loss for the minibatch. The forward pass encapsulates all the previous modules you've written and some previously defined by us for you. We have implemented most of the forward pass of the object detector for you, but the last step has not been filled in. Please compute the losses `conf_loss`, `cls_loss`, `reg_loss`, and their sum, `total_loss`. **(1 point)**

(g) To make sure that everything is working as expected, we can try to overfit the detector to a small subset of data. This overfitting experiment should only take about a few minutes to run. After 200 epochs of training you should see a total loss of around or less than 0.13. Now that we are confident that the training code is working properly, let's train the network on more data and for longer. We will train for 50 epochs; this ideally takes about 35 minutes on the GPUs that Colab provides. You should see a total loss around or less than 0.15. **(0 points)**

Note that real object detection systems typically train for 12-24 hours, distribute training over multiple GPUs, and use much faster GPUs. As such our result will be far from the state of the art, but it should give some reasonable results!

(h) Finally, we will discard redundant bounding box predictions. Specifically, we will use *non-maximum suppresion* (NMS) to remove any bounding box prediction that significantly overlaps another bounding box prediction that has a higher confidence score. Slides 51 and 52 in the object detection lecture describe the NMS process. (**2 points**)

(i) The inference step will take validation images as input and will 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. **(0 points)**

(j) 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 12% mAP or above on the validation dataset. **(0 points)**

(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 7×7 to 13×13. For more advanced techniques to try, refer to Figure 3 taken from the YOLO v2 [3] paper. It is possible to achieve 18-20% mAP using this implementation of the YOLO detector with basic modifications. Please revert the code back for grading purposes.

	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 3: Design choices for YOLO v2 leading to performance boost

Resources

1. Lecture slides
2. YOLO v1 paper [1] and this comprehensive summary of the paper by Sik-Ho Tsang..

Acknowledgement

Some functions used in this homework and the starter code are partly adapted from the Fall 2019 version of EECS 598 Deep Learning for Computer Vision course offered by Justin Johnson. Please feel free to similarly re-use our problems while similarly crediting us.

References

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

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

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