Introduction to Computer Vision: Assignment 5

Instructions

- This homework is due by **PM 3:00 on Wednesday, May 28th, 2025**.
- The submission includes two parts:
  1. Submit an `ipynb` file with all executed output.
     We have indicated questions where you have to do something in code in red.
     Ensure that every cell's output (including results, plots, etc.) is visible, and save the file. This code is optimized to run `Colab` environment, and simple modifications such as changing the directory path are allowed.
  2. Submit a `pdf` file as your write-up, including your answers to all the questions and key choices you made.
     We have indicated questions where you have to do something in the report in blue.
     You might like to combine several files to make a submission. Here is an example online link for combining multiple PDF files: `https://combinepdf.com/`.
     The write-up must be an electronic version. **No handwriting, including plotting questions**.

Google Colab Environment

- For this assignment, upload your `ipynb` file and assignment images to your personal Google Drive.
- Mount your Drive in Colab, and complete your code using Python. You can access Google Colab at: `https://colab.research.google.com/`. We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Object Detection

In this problem set, we will implement a **single-stage** object detector, based on `YOLO v1`. 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.

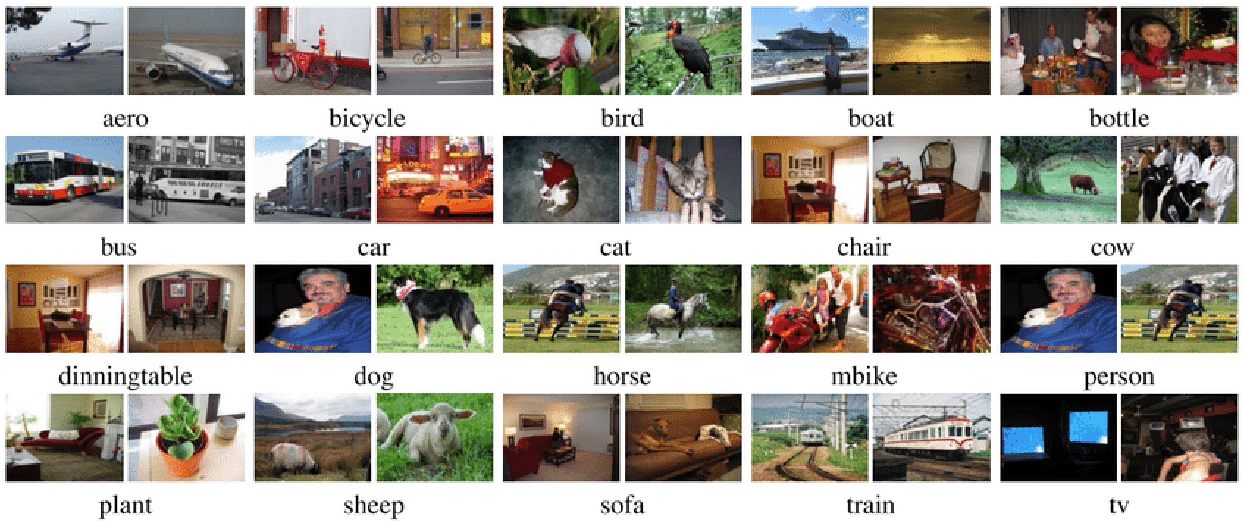

Figure 1: Example images and detections from the PASCAL VOC dataset.

(a) Detector Backbone Network [1 pts]

We will use **MobileNetv2** 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.

In your report, describe the key characteristics and advantages of `MobileNetV2`, and explain the main benefits of initializing the backbone with pretrained weights.

(b) Activation and Proposal [3 pts]

(2 pts) After passing the input image through the back 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.

In your report, insert the images showing the 7×7 activation grid overlaid on sample inputs. Then briefly describe how the code expands that grid to match the offset tensors' dimensions, extracts the predicted offsets, converts them into bounding-box parameters, and assembles those values into the final proposals tensor.

(c) Prediction Networks [4 pts]

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

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

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.

In your report, explain in plain terms what Intersection-over-Union measures—namely, how it quantifies the overlap between a predicted box and the ground-truth box—and why this metric is critical for judging localization accuracy in object detection.

(d) Bounding-Box & Class Prediction Head [4 pts]

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

For each position in the 7 X 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.

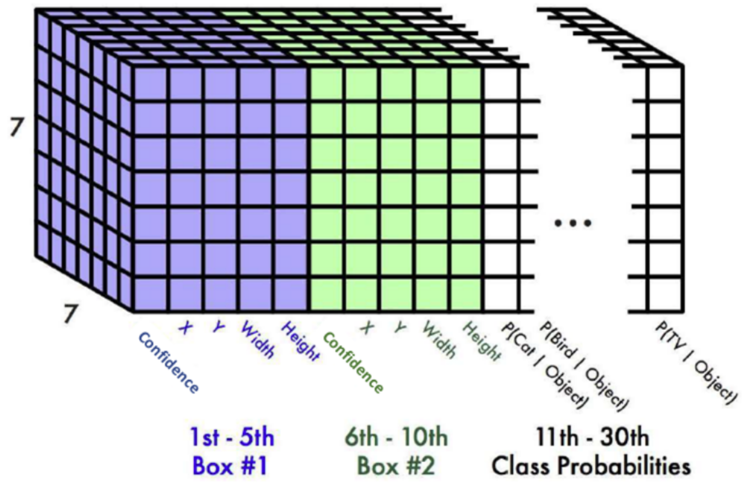

Figure 2: Output of the prediction network.

In your report, explain how you implemented the forward pass of the `PredictionNetwork` describe how you fed the backbone feature map through your custom layers to produce a unified output tensor, and then how you split that tensor into the three outputs (bbox offsets, confidence scores, class scores).

(e) Loss Function [1 pts]

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.

In your report, briefly explain the three loss components—confidence score regression, bounding-box regression, and object classification.

(f) Object Detector Code [2 pts]

(1 pts) 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, their sum, `total_loss`.

In your report, explain how you weight and sum those three loss components to produce the final total detection loss.

(g) Train the Object Detector [1 pts]

To make sure that everything is working as expected, we can try to overfit the detector a small subset of data. This overfitting experiment should only take about a few minutes to run.

In your report, briefly state your experimental setup (batch size, learning rate, optimizer, number of epochs, subset size, hardware) and insert the plot generated by your experiment code and briefly describe it.

(h) Non-Maximum Suppression(NMS) [3 pts]

(2 pts) Finally, we will discard redundant bounding box predictions. Specifically, we will use *non-maximum suppression* (NMS) to remove any bounding box prediction that significantly overlaps another bounding box prediction that has a higher confidence score.

In your report, explain that you apply non-maximum suppression to remove redundant overlapping detections by repeatedly selecting the highest-scoring box, discarding any remaining boxes whose IoU with that box exceeds a chosen threshold, and continuing until no boxes remain.

(i) Inference [1 pts]

The inference step will take validation images as input and will predict bounding boxes, confidence scores, and class labels for each object in the images. Low confidence and repeated bounding boxes will be filtered by thresholding and NMS.

In your report, insert the inference prediction result images and provide a brief overall summary of performance, common errors, and trends.

(j) Evaluation [1 pts]

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.

In your report explain how Average Precision (AP) is calculated, insert the class-wise AP bar chart, describe what that chart conveys, and then discuss the meaning of the overall mAP value across all classes.