CSci 4270 and 6270
Computational Vision,
Spring Semester 2021
Homework 6
Due: Thursday, April 22, at 11:59:59 pm EDT

Overview

This homework assignment is worth **75 points** toward your homework grade. You may use up to three late days on it. Please be aware that HW 7 will be due on Wednesday May 5 at 11:59:49 pm EDT and no late days will be allowed on it because it is due at the end of the semester.

The core of this assignment is to develop, train, test and evaluate a system that takes as input an image and its region proposals and produces as output a set of detected objects including their class labels and bounding boxes. At the heart of this is a neural network that takes a region proposal as input and predicts both a classification decision about the region and the bounding box for the region. The region proposals are generated for you. Once the predictions are made by the network for all proposals, the results must be post-processed to produce the actual detections. The classification decisions include the *nothing* class, which means there is no object in the region proposal. In this case, no bounding box will be extracted from your network. The possible classes are provided with the initial Python code we are giving you. Your goal is to maximize the mean Average Precision of the detections for each image.

Importantly, you will not need to implement the network from scratch like you did for HW 5. Instead you can use a pretrained ("backbone") network and add the classification and bounding box regression layers on top of it.

Starter python code for reading the training data, for designing your network, and for the post-processing and evaluation steps are provided for you in the zip file distributed with the assignment.

Training Input

The overall input will be a dictionary of training images from the Pascal VOC 2012 training data set. For each training image, there will be a list of candidate regions. Each candidate region will be a dictionary containing three items.

- The image coordinates of the bounding rectangle for the region.
- The ground truth correct class for the region.
- The image coordinates of the correct bounding rectangle.

The class will be an integer, and we've provided a dictionary mapping the integer to its class name. Each bounding rectangle will be a 4-tuple of pixel coordinates, x_0, y_0, x_1, y_1 giving the upper left and lower right corners of the rectangle. You should crop each region, and then resize to be a square M pixels wide. This resizing will change the aspect ratio of the region, distorting its appearance. The value of M is for you to select. (The backbone network you select may determine the minimum size region you can handle.) The regions must be gathered across all the training images and formed into a tensor of cropped and resized image regions. (If there are n_i training rectangles in the i-th image then there will be $\sum_i n_i$ training rectangles.) This tensor is the input data for training your network.

For the purposes of training, the image coordinates for the ground truth correct bounding rectangle should be scaled relative to those of the rectangle used for cropping. A simple example is that

if they are equal, the scaled coordinates are $(0, 0, 1, 1)$. A more complicated example is if the bounding rectangle for the region is $(50, 60, 90, 120)$ and the ground truth rectangle is $(45, 65, 80, 118)$ then the scaled coordinates of the ground truth rectangle should be $(-5/40, 5/60, 30/40, 58/60)$ which is $(-0.125, 0.083, 0.75, 0.967)$. The y vector for each training region should therefore consist of the index of the correct class together with these four values.

We have provided you with two different subsets of the data sets. The first, involving only four classes (plus the *nothing* class) is here

`https://drive.google.com/file/d/1khaEx8fNSnAWwI-zR1XZ4lGFkyQXOJaZ/view?usp=sharing`

and the second involving all twenty classes is here:

`https://drive.google.com/file/d/13rpCYWeAIT8u3eEFDLgJ8LrumN2uP0QU/view?usp=sharing`

You can use the smaller dataset to get your model working, and then use the larger dataset to get the detailed output required for training and testing.

Your Network

Your network should consist of the following:

1. A pre-trained backbone network from which one or more of the last few layers have been removed. For ResNet the removed layers would include the final output layer, and the global average pooling layer. See the options at

   `https://pytorch.org/vision/stable/models.html`

2. A classification layer that is fully-connected to the last remaining layer of the backbone network. If there are C classes, this classification layer should have $C + 1$ neurons, where class index 0 corresponds to the label *nothing*, meaning that there is no detection for this region.

3. A regression layer that predicts $4C$ outputs, four for each non-zero class index. This regression layer should be fully connected to the last remaining layer from the backbone network.

You will use a combined loss function for the classification and regression layers when training, as was discussed during Lecture 20 and is in the notes. You might consider adding a hidden layer to the classification network or to the regression network, but don't do so right away.

As discussed above, the input to the network is a 4-dimensional tensor of candidate regions extracted from the images. The outputs from the network are the associated activation values for each input candidate region plus the $4C$ values estimated for the upper left and lower right corners of the bounding rectangles for each candidate region for each class. When you calculate your bounding box regression loss, use the ground truth labels to determine which one of the C bounding boxes will contribute to the loss.

At this point the four values are still in the normalized coordinates because the network does not know anything about the initial image region. These values will have to be mapped back to the image coordinates in order to draw the bounding boxes during testing.

Input of Testing Data

Input of the test images and regions will be somewhat different from input of the training images. Each test image will have candidate region proposals, but the regions will only be represented by their initial bounding rectangles. There will be no ground truth label and no ground truth rectangle

associated with the candidate regions. However, you will still form the normalized candidate regions into a 4-d tensor which will be input to your network for the forward calculation.

A second testing input will give the ground truth regions for each image. Each region will be represented by a class label and by a bounding rectangle in image coordinates. These will be read from a JSON file and provided to you as a dictionary. See provided code for details.

Note: The difference between training input and testing input is because of an important difference between training and test: there may be multiple training candidate regions that significantly overlap for a correct object — in fact this is expected and is an important part of the training. If the network performs well, many of these regions will predict the same class label and final bounding rectangles that are very close to each other. The evaluation step must filter these to choose a significantly reduced set of predicted rectangles and hopefully each of these will closely match a ground truth rectangle.

Post-Processing — Only At Test Time

The post-processing proceeds one image at a time. As just discussed above, this is not done during training and validation because that step stops with the evaluation of each prediction independently. At test time, however, and for what would be the actual use of the network, the region predictions must be narrowed down to form the actual "detections" made by the system. As an initial step for doing this, you must divide up the regions so that you consider just the regions for each image as a group (list). The coordinates — at least for the detections that aren't "nothing" — must also be converted back to image coordinates.

To process the list of prediction regions for an image, the first step is to eliminate regions whose class is 0 ("nothing"); these are "non-detections". Then, order the remaining regions by decreasing activation. Finally, apply the non-maximum suppression algorithm described in Lectures 19 and 20, using an IOU of 0.5 as the threshold to decide if two regions overlap. Importantly, two regions should only compared for non-maximum suppression if they have the same class label.

Evaluation

For each test image, after this post-processing step the generated "detections" must be compared to the ground truth detections to determine which are correct and to calculate mAP. For each detection d_i in decreasing order of activation, compare d_i to the ground truth regions and find all that have the same label as d_i. Among the ground truth regions having the same label as d_i, find the region having the highest IOU with d_i. Call this region g. If no ground truth regions have the same label as d_i or if $\text{IOU}(d_i, g) < 0.5$ then d_i is an incorrect detection, and a 0 should be appended to the binary decision vector $\mathbf{b}$ (see formulation of mean Average Precision from class.) Otherwise, d_i is a correct detection, and a 1 should be appended to $\mathbf{b}$. Finally, if d_i is correct then g must be removed from the list of ground truth regions so that it does not get matched more than once.

Repeat the foregoing process for each of your detected regions for a given image, entering a 0 or 1 into $\mathbf{b}$ for each r_i. Consider up to at most 10 total regions. From this, compute the average precision (AP) score at 10 detections for this image. If your network had only $j < 10$ regions r_j, then you will make decisions about detection r_0 through r_{j-1} and enter them into $\mathbf{b}$, and fill the remaining entries of $\mathbf{b}$ with 0's. Once you have $\mathbf{b}$ for a given image, you can calculate the average precision (AP) score for this image and its detections, just as we did in the Lecture 19 exercise. Ideally, at the end of the computation, each ground truth region will be matched to a detection, but those that aren't are considered "misses" or "false negatives".

Finally, averaging the AP score over all input images gives your final mean average precision (mAP).

What You Need to Do

There are three coding parts to this assignment:

1. Data preprocessing. This is where you load the training, validation and test data, map it into the input tensor for the network, and store additional information for evaluation. We have done much of this for you, providing you with a pytorch DataSet object for loading training and validation data, and another for loading test data.

2. Forming, training and validating the actual network.

3. Writing the code for the test evaluation.

Starting with the last part, we have provided you with a template code called `evaluation.py` that outlines and tests the post-processing and evaluation code you need to write. Your completed version of this file will be uploaded separately to Submitty and automatically tested. We strongly suggest that you do this first. You will then use this code to help evaluate your test results.

Output from the system should be different for training / validation and for testing. For training and validation the output should show:

- A plot of the training and validation losses as a function of the epoch.

- A confusion matrix on the classification decisions for training and validation at the end of training.

- The average IOU between the predicted and correct bounding rectangles, only for the regions where the classifier is correct.

During testing, detailed output from the network should be given for the first 10 test images.

- The original image with the bounding boxes of the final regions drawn on top. Each bounding box should have text giving the class label. Detections that are correct (IOU with a ground truth bounding box having the same class label is at least 0.5) should be shown in green. Detections that are incorrect should be shown in red. Ground truth detections that are missed should be shown in yellow. All of the necessary information is produced by the `predictions_to_detections` and `evaluate` functions that you need to write in `evaluate.py`.

- The average precision for the image.

The final output should show

- The overall mean average precision across all test images.

- For each class, the percentage of detections of that class that are true positives and the percentage of ground truth detections that are found. Again, this information can be gathered using the results of the `evaluate` function that you need to write in `evaluate.py`.

Submission

In addition to the Submitty upload of `evaluate.py` as described above, please submit a zip file containing all your code and a pdf gathering all your results. The pdf file should include

- An explanation of your neural network model and any variations you tested.
- The training output described above on the full dataset.
- The testing output described above on the full dataset.
- A summary written assessment of your results.

Finally, to reiterate, only show results for the small starter data set if you can't get the network working on the full dataset.