CSci 4270 and 6270
Computational Vision,
Spring Semester 2025
Homework 5
Due: Thursday, April 3 11:59:59 pm EDT

Overview

This homework assignment is worth **100 points** toward your homework grade. You may use up to three late days on it.

The goal 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 — by a simulated region proposal network.) 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 of interest in the region proposal. In this case, no bounding box will be extracted from your network. The possible classes are provided with the initial Jupyter notebook we are giving you. Your goal is to maximize the mean Average Precision of the detections across the images.

Importantly, you will not need to implement the network from scratch. Instead you can use a pretrained ("backbone") network — in particular ResNet-18 — and add the classification and bounding box regression layers on top of it.

Starter python code with the `Dataset` subclass, 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 and Validation Input

The input for training and for validation will be managed by the `HW5Dataset` class, a subclass of PyTorch's `Dataset`. As with HW 4 Part 2, `HW5Dataset` implements the `__init__`, `__len__` and `__getitem__` methods. Each call to the latter returns a cropped and resized candidate image region, the ground truth bounding box, and the ground truth label. Here are some details:

- All candidate regions will be cropped from the image and resized to 224x224. This is done before they are given to you. Multiple regions will be pulled from each image. Forcing them to be a square will distort the region, but this is expected. Each region will be a 3x224x224 tensor.

- The ground truth bounding box will be a 1x4 tensor of floats giving the upper left and lower right corners of the bounding box in the coordinate system of the rectangle. Importantly, these will be specified as fractions of the region width and height. For example,the tensor $(0.1, 0.25, 0.8, 0.95)$ is the bounding box $(22.4, 56., 179.2, 212.8)$.

- The ground truth label is an integer, with the value 0 indicating that there is no correct label for this region. This is the "nothing" class. Note that three will be far more "nothing" results than real objects.

As with HW 4 Part 2, your data set object will be passed to a Dataloader which will return minibatches of 4-dimensional tensors of size Bx3x224x224, where B is the size of the minibatch.

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

`https://drive.google.com/drive/folders/1uRz6BFNPGBqzfyxJGwbBE9eiwq-bMzTK?usp=drive_link`

This is the dataset where you should focus your effort. The second, larger data set involves all twenty classes here:

`https://drive.google.com/drive/folders/1Xgkf3QBdJAEz2vkBl4uy029--OPLoNqm?usp=sharing`

You can experiment with the larger dataset for a small amount of extra credit once you have good results on the smaller one.

Your Network

Your network should consist of the following:

1. A pre-trained backbone network from which one or more of the last layers have been removed. We provide you a starter `HW5Model` class which already has this pre-trained backbone using `torchvision.models.resnet18`. We have removed the last fully-connected layer, meaning the last layer of the backbone is a global average pooling layer. See the final comments section at the end of these instructions for more information on this. This backbone will be frozen, meaning you will not be training its weights. You can choose to unfreeze the backbone (fully or partially) and potentially achieve better results, but your model will take longer to train.

2. Add 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*.

3. A regression layer that predicts $4C$ regression bounding box parameters, 4 for each of C classes. This regression layer should be fully connected to the last remaining layer from the backbone network.

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 (and therefore the back propagation error), use the ground truth label to determine which one of the C bounding boxes will contribute to the loss.

You will use a combined loss function for the classification (softmax and binary cross entropy) and regression layers (sum of square errors) when training, as was discussed during Lecture 18. You might consider adding a hidden layer to the classification network or to the regression network, but don't do so right away. You are welcome to use any optimization method you wish, or you can stick with stochastic gradient descent.

Finally, note that even though the B candidate regions in a minibatch are passed through the network together, they will each be evaluated independently during training. This is very different from what will happen with the test input after the network has been trained — to be discussed below.

The Validation Step

Validation input will be in exactly the same form as the training input. Computing the validation loss is straight forward. You can also compute the accuracy. This will require that you count the number of correct decisions the network makes on the validation candidate regions. A decision is correct if the neuron with the highest activation corresponds to the ground truth label **and**, when the correct label is not 0, the IOU between the ground truth bounding box and the predicted bounding box is greater than the IOU threshold which is usually 0.5, but not always.

You should evaluate your model on the validation data after every epoch. This will give you an idea of how well your model works on data outside your training set, and you should choose the model parameters which give you the best validation performance. One of these parameters is the number of epochs you train for, and if you find that training for a certain number of epochs results in better performance you should train for that long for your final model.

After you train the best model you can based on the validation data, then proceed to the final testing steps.

Input of Testing Data

Input of the test data is managed by `HW5DatasetTest`, and is organized by images. The call to the `__getitem__` method is more complicated than for `HW5Dataset` used for training and validation: it will return an image (a numpy array) together with all of its candidate image regions (each normalized to a 3x224x224 tensor), candidate bounding boxes, ground truth bounding boxes, and ground truth labels. Unlike during training, all bounding boxes are provided in the **coordinate system of the original image**.

At test time, the candidate image regions will be fed into the network as a four-dimensional tensor in just the same way as a minibatch is during training. The post-processing to evaluate the results is very different however, as we will now see.

Post-Processing — Only At Test Time

The post-processing proceeds one image at a time. This is not done during the training and validation steps because these stop with the evaluation of each candidate region and its prediction independently. At test time, however, and for what would be the actual use of the network, the predictions for each candidate region in a single image must be combined and filtered down to form the actual "detections" made by the system. As an initial step for doing this, the predicted coordinates — for the detections that aren't "nothing" — must 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 non-maximum suppression using an IOU of 0.5 as the threshold to decide if two regions overlap (in image coordinates). Importantly, as discussed in class, two regions should only be compared for non-maximum suppression if they have the same class label. The result of the non-maximum suppression is the set of actual ("predicted") detections.

Evaluation

For each test image, after this post-processing step the predicted detections must be compared to the ground truth detections to determine which are correct and to calculate the AP for each image and the mAP over all images.

An intermediate goal here is to form the binary vector **b** for the computation of the AP for each image. For each predicted detection d_i in decreasing order of activation, compare d_i to the ground truth regions for the image 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 **b**. Otherwise, d_i is a correct detection, and a 1 should be appended to **b**. In this case, 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 **b** for each d_i.

Next, compute the precision and recall for each entry of **b**. Precision is computed as

$$\mathbf{p} = \text{cumsum}(\mathbf{b}) / \text{arange}(1, n+1)$$

where n is the length of **b**. Recall is

$$\mathbf{r} = \text{cumsum}(\mathbf{b}) / m$$

where m is the number of ground-truth detections for the image.

The second to last step is to compute precision-recall measurement pairs:

$$(p_i, r_i) \text{ for } i \in 0, 1, \ldots n-1$$

The final step for computing the average precision for an image is to compute the area under the precision-recall curve that is implicitly constructed from the (p_i, r_i) pairs. This is done by evaluating the curve at 11 uniformly-spaced steps in recall:

$$\hat{r}_j = j/10 \text{ for } j \in 0, \ldots, 10$$

At each recall $\hat{r}_j$ — addressing the non-monotonicity issue discussed in class — the precision is the maximum precision among all recall values greater than $\hat{r}_j$:

$$\hat{p}_j = \max\{p_i \text{ such that } r_i \geq \hat{r}_j.\}$$

The area under the P-R curve is now simply

$$\frac{1}{11} \sum_{j=0}^{j=10} \hat{p}_j.$$

This is the AP value for the detections for a single image. We average this across all test images to obtain MAP. **Important note:** These last two steps are slightly different and easier to implement than what I presented in class.

What You Need to Do

There are several coding parts to this assignment:

1. Forming your network based on ResNet18, including its loss functions.
2. Writing your training and testing functions.
3. Writing the code for the test evaluation.

Starting with the last part, we have provided you with a template code in a start ipynb file that outlines and tests the post-processing and evaluation code you need to write.

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 test images 1, 16, 28, 30, 39, 75, 78, 88, 93, and 146:

- 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

Please submit a single Jupyter notebok with your code. This notebook should start with a markdown cell explaining your network model, any variations you tested, and anything else you'd like us to know. It should then include the evaluation functions and the test cells for the evaluation functions, all before starting the actual neural network code. The notebook should include the training output described above, the testing output described above, and a summary written assessment of your results.

Final Comments

- If you are using Google Colab, it might be worth the money to upgrade your subscription from the free version.
- Each training epoch on the A100's in the Pro version should take a few minutes. If your code is much slower than this then there may be significant inefficiencies. For instance during training you should only be using for loops over the epochs and dataloaders. Also, we suggest that you only compute your validation loss and accuracy at the end of each epoch.

- We urge you to first test your ability to simply predict the correct class for each candidate region, without regard for the bounding boxes. If this is as far as you get you will still get at least 75% credit on the network part of this assignment.
- On the other hand, if you get good results on the full data set you can earn up to 10 points extra credit for the assignment.
- We found similar bounding box regression results between using the 512 dimension output of the global average pooling layer, which includes no explicit spatial information, and the 7x7x512 output of the layer just ahead of this which has 49 times more parameters but has some spatial information. We therefore suggest that you use the output of your global average pooling layer as your input to the fully-connected layer.
- One inefficiency in the dataset is that each image is loaded and then cropped to the candidate region in the `__getitem__` function. A better idea might be to cache each cropped image in a file that is read. You are welcome to make this change.
- You are also welcome to add augmentation to the Dataset.