Lab2

February 20, 2019

Problem

In this lab, we'll try to train a network for semantic segmentation task. We will learn to use PyTorch's Dataloader class and implement a more complex CNN using nn.Module(). For this task, we're going to use PASCAL VOC 2007 dataset. The dataset can be found in `/home/VOC2007/` directory. In a nutshell, the relevant directories are:

1. `JPEGImages/` contains all the image files. Note that not all the image have been annotated for semantic segmentation (because VOC is also annotated for detection and activity recognition tasks).

2. `SegmentationClass/` directory has all the ground truth segments of the images. Please see that these are not ordinary images. If you load them (which you will have to), you'll find that these contain values from 0 to 20 and occasionally 255. A zero pixel value corresponds to the background class, 1-20 pixel values correspond to the segmentation category, eg. 1 for airplane, 15 for human, etc. Pixels having values of 255 (white) can be ignored because they're used to draw a separation boundary around segments.

3. `ImageSets/Segmentation/` contains the three files: train.txt, trainval.txt and val.txt. These files provide the filenames in JPEGImages/ directory that have been annotated with semantic segmentation labels. You'll have to read this file and pick only the images that are listed in these files.

The implementation tasks for the assignment are divided into two parts:

1. Designing a *complex* network architecture using PyTorch's *nn* module.

2. Writing a dataloader to load the data in an orderly fashion.

3. Computing evaluation metrics like IOU to evaluate network's performance.

Below you will find the details of tasks required for this assignment.

1. **DataLoader:** PyTorch's dataloader feature makes it a lot easier to handle things like batching, parallelising, shuffling, perturbations, etc. In this part, you'll have to implement a `torch.utils.Dataset` class. We have initialised the class for you and also loaded the data from the text files in `ImageSets/Segmentation/` directory. Your task is to implement the `__getitem__()` function. It's input is a random index in the range between 0 and number of training images. Given an index, the function should output the image and ground truth segmentation mask for the image.

   A *segmentation mask* is a matrix of same size as the input image. If there are n categories to be segmented, you need to have n segmentation masks. Now, for a mask of i^{th} category, a pixel (x, y) should either be 0 if the pixel (x, y) in the image does not belong to category i, or be 1, if it does. This set of segmentation masks creates our ground truth and this is what the network should output, a $21 \times 256 \times 256$ dimensional output for a $3 \times 256 \times 256$ dimensional input.

2. **Network Architecture:** In yesterday's lab, we used `nn.Sequential` to construct our network. However, a Sequential container may not always be desirable, eg. in the case of a Residual Network (ResNet) where the layers are not always stacked serially. To have more control over the network architecture, we'll define a model class that implements PyTorch's `nn.Module` superclass. We have provided a skeleton code in *model.py* file consisting of a simple CNN. The idea is simple: you need to write the `forward()` function which takes a variable `x` as input. As shown in the skeleton code, you may use `__init__()` function to initialize your layers and simply stack them the way you want in `forward()` function. You are free to design your custom network. Some recommended architectures are a fully-convolutional ResNet and SegNet networks. For implementing these, you may want to use Transposed Convolutions for upsampling the feature maps the way SegNet does. Train the entire network with an MSE loss function.

3. **Training:** Please refer to *train.py* which implements the training loop. It's similar to the one you used yesterday, except the use of dataloader. Notice the input to the dataloader is the dataset class you implemented in the first part along with extra parameters like batch size, shuffle and parallel data loading. Since we're solving semantic segmentation today, you'll need to implement an evaluation metric which is commonly used to evaluate the quality of semantic segmentation: Intersection over Union (IoU). For this, you'll need to fill the `compute_iou(pred, gt)` function which takes as input the predicted and the ground truth segmentation masks. For every ground truth segment in the image, calculate the intersection of the ground truth segment with the predicted segment of the same category. Also compute the union of the two segments and finally find the ration of intersection and the union. Print these values at the end of every epoch.

4. **Validation:** Similar to training dataloader, call the dataloader with the validation set by changing the parameter `image_set` to 'val'. Compute the IoU and MSE loss of the network after every 5 epochs. Stop the training when you observe stabilization in network's performance.