CSCI 677: Advanced Computer Vision - Fall 2021

Instructor: Prof. Nevatia

Assignment 5

Due on November 16, 2021 before 17:00 PDT

Instructions

This is a programming assignment to create, train, and test a CNN for semantic segmentation following the version of FCN32s and FCN16s described in https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf.

Architecture

Construct a modified version of Fully Convolutional Network FCN-32s. The paper describes a version using VGG-16 (though some figures are based on AlexNet); in this assignment, you are asked to replace the VGG-16 backbone model with the ResNet-18 model which is specified in Figure 1 (from the ResNet paper).

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 1: Architectures of ResNet models in the original ResNet paper. Building blocks are shown in brackets, with the numbers of blocks stacked. Downsampling is performed by *conv*3_1, *conv*4_1, and *conv*5_1 with a stride of 2.

You need to do the following modifications to convert the network to fully convolutional form:

- Replace the average pool layer by an average pooling layer (referred to as "avgpool") with kernel size of 7 x 7 with stride as 1 and no padding.
- Add a new convolutional layer with a kernel size of 1 x 1. The number of kernels is (number of classes + 1), including a background class; this layer will compute the probabilities of each class over its spatial extent.
- Add a transpose convolution layer with stride=32 and kernel_size as 64, that up-samples the classifier tensor back to the input image size. In PyTorch you can use torch.nn.ConvTranspose2d.

As discussed in class, the average pooling layer further reduces resolution (beyond 32x reduction). We can get around this difficulty by zero-padding the input image by 100 on each side. For the upsampled images, you may need to crop the center to make it the same size as the input image. You can follow the example for VGG given in FCN implementation. A neater approach that avoids 100 pixel paddings was also discussed in class; you may follow that approach instead if you prefer.

You can obtain a pre-trained ResNet-18 using torchvision.models.resnet18(pretrained = True). To start from the pretrained ResNet-18 model, once you have modified the FCN-32s structure, we would like to construct a modified version of FCN-16s also. In FCN-16s, we additionally use the previous layer (output of final layer of conv4_x) output and combine it with the upsampled feature from avgpool; the procedure is defined in the FCN paper.
Note that for FCN-16s you would need output from a previous layer and the default module obtained from torchvision doesn't contain a function to obtain output from a different layer. There are two ways to go about this:

1. Copy-paste the whole forward function of ResNet-18 and create a new function which does forward pass only till the required layer, or change the outputs of the forward function to return all the layers. For example, for default torchvision function:

```
    def forward(self, x):
        x1 = self.layer1(x)
        x2 = self.layer2(x1)
        return x2
```

   After modification, it should be

```
    def forward(self, x):
        x1 = self.layer1(x)
        x2 = self.layer2(x1)
        return x2, x1
```

2. Use class IntermediateLayerGetter. Please refer to this discussion.

Dataset

In this assignment, you'll use the Kitti dataset for sementic segmentation. You can download the data from http://www.cvlibs.net/datasets/kitti/eval_semseg.php?benchmark=semantics2015. You'll need

1. Go to website
2. Download the dataset listed in Figure 2.
3. You will need to provide your email to get a download link.
4. Once downloaded, you can upload the zip file to Colab.
5. In the execution cell, run !unzip path_to_the_file (e.g. !unzip data_semantics.zip) to unzip the file
6. For visualization, you also need to download the development kit and unzip it in the same way as above

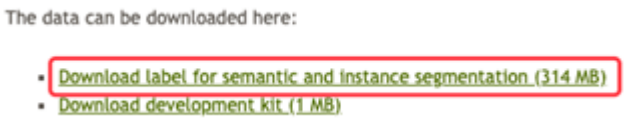

Figure 2: The dataset enclosed in the red box is what we need for this experiment.

You should now have

1. **data_semantics** which contains images and ground truth, and
2. **devkit_semantics** which contains metadata of Kitti.

data_semantics has two folders: training and testing, we will not use testing folder. In training folder, you can find four folders: image_2, instance, semantic, semantic_rgb. We will not use the instance folder. For model training, use image_2 and semantic folder.

image_2 contains 200 images and semantic contains corresponding ground truth. You can use 'opencv' or 'skimage' to read image and ground truth. Ground truth gives the category of each pixel. There are 35 categories (see figure below) defined in Kitti (not all of them exist in a single image but we need not care about the missing classes), you can find the corresponding label in the file devkit_semantics/devkit/helpers/labels.py. In this label file (devkit_semantics/devkit/helpers/labels.py), we only care about 3 columns: "name", "id" and "color"; "name" is the human readable label for each class, "id" is label for each class in the ground truth, "color" gives the mapping form class to RGB color. For visualization, you will need to map your prediction to color according to ("id", "color") correspondence. In our experiment, we will use the first 34 categories, and it is possible for some specific categories, the number of instances is 0. Please mark these numbers as N/A in your report and do not take it into account for average IoU.

```
labels = [
    #       name                     id    trainId   category            catId     hasInstances   ignoreInEval   color
    Label(  'unlabeled'            ,  0 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'ego vehicle'          ,  1 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'rectification border' ,  2 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'out of roi'           ,  3 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'static'               ,  4 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'dynamic'              ,  5 ,      255 , 'void'            , 0       , False        , True         , (111, 74,  0) ),
    Label(  'ground'               ,  6 ,      255 , 'void'            , 0       , False        , True         , ( 81,  0, 81) ),
    Label(  'road'                 ,  7 ,        0 , 'flat'            , 1       , False        , False        , (128, 64,128) ),
    Label(  'sidewalk'             ,  8 ,        1 , 'flat'            , 1       , False        , False        , (244, 35,232) ),
    Label(  'parking'              ,  9 ,      255 , 'flat'            , 1       , False        , True         , (250,170,160) ),
    Label(  'rail track'           , 10 ,      255 , 'flat'            , 1       , False        , True         , (230,150,140) ),
    Label(  'building'             , 11 ,        2 , 'construction'    , 2       , False        , False        , ( 70, 70, 70) ),
    Label(  'wall'                 , 12 ,        3 , 'construction'    , 2       , False        , False        , (102,102,156) ),
    Label(  'fence'                , 13 ,        4 , 'construction'    , 2       , False        , False        , (190,153,153) ),
    Label(  'guard rail'           , 14 ,      255 , 'construction'    , 2       , False        , True         , (180,165,180) ),
    Label(  'bridge'               , 15 ,      255 , 'construction'    , 2       , False        , True         , (150,100,100) ),
    Label(  'tunnel'               , 16 ,      255 , 'construction'    , 2       , False        , True         , (150,120, 90) ),
    Label(  'pole'                 , 17 ,        5 , 'object'          , 3       , False        , False        , (153,153,153) ),
    Label(  'polegroup'            , 18 ,      255 , 'object'          , 3       , False        , True         , (153,153,153) ),
    Label(  'traffic light'        , 19 ,        6 , 'object'          , 3       , False        , False        , (250,170, 30) ),
    Label(  'traffic sign'         , 20 ,        7 , 'object'          , 3       , False        , False        , (220,220,  0) ),
    Label(  'vegetation'           , 21 ,        8 , 'nature'          , 4       , False        , False        , (107,142, 35) ),
    Label(  'terrain'              , 22 ,        9 , 'nature'          , 4       , False        , False        , (152,251,152) ),
    Label(  'sky'                  , 23 ,       10 , 'sky'             , 5       , False        , False        , ( 70,130,180) ),
    Label(  'person'               , 24 ,       11 , 'human'           , 6       , True         , False        , (220, 20, 60) ),
    Label(  'rider'                , 25 ,       12 , 'human'           , 6       , True         , False        , (255,  0,  0) ),
    Label(  'car'                  , 26 ,       13 , 'vehicle'         , 7       , True         , False        , (  0,  0,142) ),
    Label(  'truck'                , 27 ,       14 , 'vehicle'         , 7       , True         , False        , (  0,  0, 70) ),
    Label(  'bus'                  , 28 ,       15 , 'vehicle'         , 7       , True         , False        , (  0, 60,100) ),
    Label(  'caravan'              , 29 ,      255 , 'vehicle'         , 7       , True         , True         , (  0,  0, 90) ),
    Label(  'trailer'              , 30 ,      255 , 'vehicle'         , 7       , True         , True         , (  0,  0,110) ),
    Label(  'train'                , 31 ,       16 , 'vehicle'         , 7       , True         , False        , (  0, 80,100) ),
    Label(  'motorcycle'           , 32 ,       17 , 'vehicle'         , 7       , True         , False        , (  0,  0,230) ),
    Label(  'bicycle'              , 33 ,       18 , 'vehicle'         , 7       , True         , False        , (119, 11, 32) ),
    Label(  'license plate'        , -1 ,       -1 , 'vehicle'         , 7       , False        , True         , (  0,  0,142) ),
]
```

Kitti does not offer ground truth for testing set so for this assignment, you should split the original training set into training/validation/testing sets. Sort images alphabetically and split by ratio 70%/15%/15% for training/validation/testing. (**Hint:** os.listdir() to read files may give file order shuffled, use sorted() function after getting files) Folder semantic_rgb contains the visual ground truth of the segmentations. You will use it in visual comparison with your prediction.

PyTorch does not offer a predefined dataset class for Kitti. You will need to implement a KittiDataset class by yourself.

Training

Train the network using the images in the train set mentioned in the Dataset section with your own hyperparameters. Please note that the number of epoch should be no more than 50, otherwise you may need to adjust your learning rate for your setting.

Use cross entropy loss over the individual pixels as the loss function. You could use 'torch.nn.CrossEntropyLoss'.

It should not be necessary to do data augmentation as there are many samples in each image. However, if you choose to experiment with augmentations, please note that you would need to perform the same augmentation on the ground-truth segmentation mask as well. As an example, in classification setting, a flipped dog is still a dog, however, in the image segmentation setting, a flipped dog will require a flipped mask. Record the IoU and loss after each iteration so that you can monitor it and plot it to show results.

Evaluation metric

Use the following two evaluation metrics

1. Pixel-level intersection-over-union (IoU): Pixel-level IoU = TP/(TP+FP+FN), where TP, FP, and FN are the numbers of true positive, false positive, and false negative pixels, respectively. Pixel-level IoU should be computed on each class separately (treat other classes as negative when computing for on class). NOTE: You should compute IoU over all testing images not on a single image.
2. Mean Intersection-over-Union (mIoU): Simple average of per-class pixel-level IoUs, it reflects the model's generality on all classes. Please write your own code to evaluate network performance.

Results

Map your output labels to color images which should look like the images in the `semantic_rgb` folder. Color mapping is provided in the file `devkit_semantics/devkit/helpers/labels.py`.

SUBMISSION

You should turn in a PDF report along with your code in a compressed file with the following components:

1. A brief description of the programs you write, including the source listing.
2. Evolution of loss function with multiple steps.
3. A summary and discussion of the results, including the effects of parameter choices. Compare the 2 versions of modified FCN (32s and 16s). Include the visualization of results; show some examples of successful and some failure examples.