HOMEWORK ASSIGNMENT #5
DUE: Tuesday, November 12, 2020, 2:00PM
CSCI677: Computer Vision, Prof. Nevatia
Fall Semester, 2020

This is a programming assignment to create, train, and test a CNN for the task of semantic segmentation following the FCN32s method described in https://people.eecs.berkeley.edu/~shelhamer/data/fcn.pdf(also in KRSB book, section 7.3.1).

a) **Architecture**:

Construct a Fully Convolutional Network FCN-32s. FCN-32s network is based on VGG-16 network with 3 modifications. You can obtain a pre-trained VGG16 using "torchvision.models.vgg16 (pretrained=True)"` and use "features" method to perform forward pass until before the linear layers.

(i) Replace the first FC layer by a convolutional layer (referred to as "conv6") with kernel size of 7 x 7. Number of kernels is the output dimension of the original FC layer, which is **4096**;.

(ii) Replace the second FC layers by a convolutional layer (referred to as "conv7") with a kernel size of 1 x 1. The number of kernels is **4096**.

(iii) Replace the third FC layers by a convolutional layer with a kernel size of 1 x 1. The numbers of kernels are **number of classes**.

(iv) Add a single transpose convolutional layer after the third FC layer. The configuration for the transpose convolutional layer in this assignment is stride=32, that up-sample the feature tensor back to the input image size. In PyTorch you can use `torch.nn.ConvTranspose2d` for transpose convolution.

Except for the three modifications above, other parts of FCN-32s are the same as for VGG-16. Below is the table from the VGG paper(https://arxiv.org/pdf/1409.1556.pdf), **ConvNet Configuration D** is the one we want to use.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Once we have the FCN-32s structure, we would like to construct FCN-16s as well. The main difference is that in FCN32s we directly upsample from the conv7 output, whereas in FCN-16s we additionally use the previous layer (conv6) output and combine it with the conv7 output, the latter being upsampled; 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:

(i) copy-paste the whole forward function of vgg16 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:

Default torchvision function:

```

def forward(self, x):

    x1 = self.layer1(x)

    x2 = self.layer2(x1)

    return x2

```

New function:

```

def forward(self, x):

    x1 = self.layer1(x)

    x2 = self.layer2(x1)

    return x2, x1

```

(ii) Use class `IntermediateLayerGetter` (https://github.com/pytorch/vision/blob/master/torchvision/models/_utils.py)
Here is an example provided by Pytorch, you need to replace the model and intermediate layer according to HW requirements.

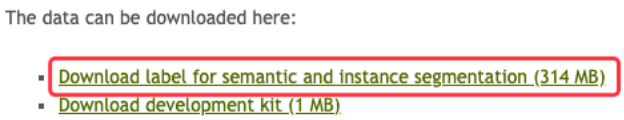
```
>>> m = torchvision.models.resnet18(pretrained=True)
>>> # extract layer1 and layer3, giving as names `feat1` and feat2`
>>> new_m = torchvision.models._utils.IntermediateLayerGetter(m,
>>>     {'layer1': 'feat1', 'layer3': 'feat2'})
>>> out = new_m(torch.rand(1, 3, 224, 224))
>>> print([(k, v.shape) for k, v in out.items()])
>>>     [('feat1', torch.Size([1, 64, 56, 56])),
>>>      ('feat2', torch.Size([1, 256, 14, 14]))]
```

b) **Dataset**

We will use the Kitti dataset for semantic segmentation.

You will need to download dataset from http://www.cvlibs.net/datasets/kitti/eval_semseg.php?benchmark=semantics2015.

- Go to website
- Download the dataset



- You will need to provide your email to get a download link
- Once downloaded, you can upload the zip file to Colab
- In the execution cell, run `"!unzip path_to_the_file" (e.g."!unzip data_semantics.zip")` to unzip the file
- For visualization, you also need to download the development kit and unzip it in the same way as above

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.

```
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.

c) **Training**:

Train the network using the images in the train set mentioned in the **Dataset** section.

We suggest using a batch size of 5, ADAM optimizer, learning rate = 0.001, momentum = 0.99. though you are free to experiment with other values.

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.

d) **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.

e) 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 single PDF 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 FCN (32s and 16s). Include the visualization of results; show some examples of successful and some failure examples.