CSE 253: Programming Assignment 3
Semantic Segmentation

Winter 2020

Instructions

Due Sunday, February 16^{th}:

1. Start early! If you have any questions or uncertainties about the assignment instructions in Part 1 or Part 2, please ask about them as soon as possible (preferably on Piazza, so everybody can benefit). We want to minimize any possible confusion about this assignment and have tried very hard to make it understandable and easy for you to follow.
2. You will be using PyTorch (v 1.4), a Deep Learning library, for the tasks in this assignment. See the material on Piazza as to how to access the UCSD GPU server, the data, and PyTorch. Additionally, any updates or modifications to the assignment will be announced on Piazza.
3. Please work in teams of 4 or 5 individuals (no more than 5).
4. Please submit in your assignment via Gradescope. The report should be in NeurIPS format or another "top" conference format (e.g., IEEE CVPR, ICML, ICLR) - we expect you to write at a high academic-level (to the best of your ability). Make sure your code is readable and well-documented - you may want to reuse it in the future. Your report should be written as if you are writing a real conference paper.

Learning Objectives

1. Understand the basics of convolutional neural networks, including convolutional layer and de-convolutional layer mechanics, Relu, and dimensions of layers.
2. Learn how to implement a CNN architecture in PyTorch for Semantic Segmentation using best practices.
3. Build intuition on the effects of modulating the design of a CNN by experimenting with your own (or "classic") architectures.
4. Visualize and interpret learned feature maps of a CNN.

Part I

Understanding Convolutional Network Basics

This portion of the assignment is to build your intuition and understanding the basics of convolutional networks - namely how convolutional layers learn feature maps and how max pooling works - by manually simulating these. This part of the assignment has to be submitted independently.

For questions 1-3, consider the $(5 \times 5 \times 2)$ input with values in $\{-1, 0, 1\}$ and the corresponding $(3 \times 3 \times 1)$ filter shown below.
(As in PyTorch, we refer to this *single* filter as being $(3 \times 3 \times 1)$ despite having two sub-filters, because it would produce an output with 1 channel. In a case where the input had 4 channels, the filter would have 4 sub-filters but still be $(3 \times 3 \times 1)$.)

Input Features

-1	0	0	-1	1
0	1	1	-1	0
1	1	1	1	1
-1	0	-1	1	0
0	0	1	1	1

-1	-1	-1	0	-1
-1	0	-1	-1	0
0	-1	0	0	1
0	1	1	1	1
1	0	1	1	1

Filter 0

-1	-1	-1
0	0	0
1	1	1

1	0	-1
1	0	0
1	1	1

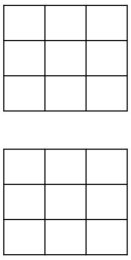
1. In the provided area below, fill in the values resulting from applying a convolutional layer to the input with no zero-padding and a stride of 1. Calculate these numbers before any activation function is applied. If there are any unused cells after completing the process in the provided tables, place an $\times$ in them. You can assume a bias of 0. The Output Feature Map will be a combination of the Filtered inputs, as in the Stanford example.

Filtered Inputs

Output Feature Map

2. Think about what ideal 3×3 patch (values in range [-1, 1]) from each of the input channels would maximally activate the corresponding 3×3 filter. Fill in these maximally activating patches in the area below. If there are any cells for which the input values don't matter, place an $\times$ in them.

Maximally Activating Patch



3. **Spatial pooling:** Using your **output feature map** in question 1, apply *max-pooling* using a $[2 \times 2]$ kernel with a stride of 1. Recall from lecture that spatial pooling gives a CNN invariance to small transformations in the input, and max-pooling is more widely used over sum or average pooling due to empirically better performance.

Max Pooled Output

4. **Number of learnable parameters**
Suppose we had the following architecture:

```
inputs -> conv1 -> conv2 -> conv3 -> maxpool -> fc1 -> fc2 (outputs)
```

where all convs have a stride of 1 and no zero-padding. Conv1 has a kernel size of $[8 \times 8]$ with 12 output channels, conv2 has a $[8 \times 8]$ kernel with 10 output channels, conv3 has a $[6 \times 6]$ kernel with 8 output channels, and maxpool has a $[3 \times 3]$ kernel.
If ReLU is the activation function between each layer and the inputs are $[512 \times 512]$ greyscale images, what are:

(i) The number of input channels to **conv1**: _____

(ii) The number of input channels to **conv2**: _____

(iii) The number of input channels to **conv3**: _____

(iv) In order to add a fully-connected layer following the maxpool layer, we need to reshape the convolutional outputs to feed them into this hidden layer. Based on this architecture, what will the incoming dimensions to **fc1** be? Show your work.

Part II

Fully Convolutional Networks for Semantic Segmentation

Problem statement: Deep convolutional neural networks have been applied to a broad range of problems and tasks within Computer Vision. In this part of the assignment, we will explore a *semantic segmentation* task. Semantic segmentation means that every pixel in an image is classified as belonging to some category. This means that at the output level, there is a softmax over all of the categories *for each pixel in the image*. This is increasingly relevant and important in the industry. We will build deep CNN architectures and train them from scratch to predict the segmentation masks of the images.

The **Cityscapes** dataset contains 34 classes in total and the statistics of the dataset can be found here. An overview of the classes and labelling policy can be found here. The main goal of this challenge is to recognize objects from a number of visual object classes in realistic scenes (i.e., not pre-segmented objects). It is fundamentally a supervised learning learning problem in that a training set of labelled images is provided. More information about this dataset can be found here.

Implementation Instructions

We have provided you with a data loader; a custom PyTorch Dataset class, specifically designed for this dataset, and have separated the training and validation datasets. Please familiarize yourself with the code and read the comments.

1. **Evaluating your model:**
   Before we discuss the neural network details, we must clarify how you will accurately and transparently evaluate your model's performance. Here are some of the essential metrics for this:

   (i) Pixel Accuracy: percent correct predictions $= \frac{\text{total correct predictions}}{\text{total number of samples}}$
   The issue with this measure is called *class imbalance*. When our classes are extremely imbalanced, it means that a class or some classes dominate the image, while some other classes make up only a small portion of the image. This means the network can reduce the error considerably just by labeling everything with the majority class. Unfortunately, class imbalance is prevalent in many real world datasets, so it canât be ignored. Therefore, there are alternative metrics that are better at dealing with this issue.

   (ii) Intersection-Over-Union (IoU, Jaccard Index): Simply put, the IoU is the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth. It is computed on a per-class basis, and then these are averaged over the classes. It is measured as, IoU $= \frac{TP}{TP+FP+FN}$, where TP, FP, and FN are the numbers of true positive, false positive, and false negative pixels, respectively, determined over the whole test set. See this blog for an example of the difference between pixel accuracy and IoU. For a particular class, let's say **car**, TP is the number of pixels for which the ground truth is **car** and the network labels them as **car**. FP is the count of pixels that the network labeled as **car**, but weren't cars. FN is the number of pixels that the network should have labeled **car**, but didn't.

   Complete the implementation of these functions in the utils.py file provided. A detailed explanation of the class label ids and their mapping is provided in the file: dataloader.py

   For each experiment, Please report the following:
   Average Pixel Accuracy and average IoU (include only classes with trainId $\neq 255$) along with the IoU for the classes - building(11), traffic sign(20), person(24), car(26), bicycle(33).

2. **Create a baseline model:**
   An outline for creating the model is provided in basic_fcn.py for the purposes of (i) getting acquainted with PyTorch, (ii) getting initial results to compare with more complex architectures and approaches to solving this

instance segmentation problem. The baseline architecture contains an encoder and a decoder structure. Using the starter code provided, complete the implementation for the architecture as described in the comments, replacing the instances of __ with its corresponding missing value. This includes finishing the **__init__()** and **forward()** functions.

To run the **basic_fcn.py** model, we have additionally provided a basic Jupyter notebook, **starter.ipynb**, containing nearly all the code needed to train the model. Based on your reasoning for determining the activation function for the output layer of the network, determine which loss criterion you should be optimizing - this may be something you are already familiar with (note that PyTorch loss criteria can be found in the **torch.nn** package). Additionally, use the Adam gradient descent optimizer, which can be found in the **torch.optim** package. Train the baseline model until the loss stops decreasing on the validation set.

Please note that this is a very bare-bones implementation which might not perform very well. As such, this architecture may have "good" overall classification accuracy, but fail to address the class-imbalance problem.

In addition, make note that the data loader can apply any specified color-scale conversion and transformations to your images *on-line*. In the next section, you are welcome to optimize this as you see fit (including preprocessing the complete dataset and saving a copy in your **home directory in the Kubernetes environment**).

3. **Experimentation and your solution:**
To get a better sense of how your design choices affect model performance, we encourage you to experiment with various approaches to solving this instance segmentation problem using a deep CNN. You are welcome to make a copy of the **basic_fcn.py** file and **starter** notebook for this purpose. At a minimum, you should do the following:

   (i) Augment your dataset by applying transformations to the input images: this can include using mirror flips, rotating the images slightly, or different crops. Make sure to apply the same transformations to the corresponding labels!

   (ii) Try one other additional architecture - this includes making significant changes in the number of layers, activation functions, dimensions of the convolutional filters, etc. Simply adding an additional layer is not sufficient!

   (iii) Address the rare class or imbalanced class problem. This is often done by **pressuring** the network to categorize the infrequently seen classes. You can use: (1) weighted loss, which weights infrequent classes more, or (2) dice coefficient loss. At a minimum, you must implement your own weighted/balanced loss criterion.

   (iv) Try Transfer Learning with any of the architectures present in the Pytorch Library to replace the encoder part of the given FCN architecture. Does this improve the performance? Why do you think you observe the changes that you do?

   (v) Refer to the U-Net architecture in this paper and implement the same for the given dataset. Note that this architecture is very similar to the FCN architecture provided with some important changes.

What to include in your report

In addition to learning very useful and applicable skills in deep learning and computer vision, this portion of the assignment will help you learn to write a scientific report. Please include an abstract and the following *7 sections*:

0. **Abstract (5 pts)**
The **Abstract** should serve as a ≈ one paragraph synopsis of the work in your report, including the task (e.g. multinomial regression on dataset X), how you approached it, and a quick overview of your results.

1. **Introduction (5 pts)**
The **Introduction** should describe the problem statement or task, why it's important, and any necessary background your audience may need to know. Keep in mind that since we're using Xavier weight initialization and batch normalization, you should understand the basic mathematics behind these concepts at a minimum, how they affect your network parameters, and why they're useful. As such, these should be discussed in either the **Introduction** or **Methods** sections.

2. **Related Work (5 pts)**
   The **Related Work** section should review any work you used to inspire your approach - this includes previous research in the specific problem you address, as well as any core ideas you build upon. You should include citations in standard reference format with this itemized in a **References** section at the end of the report. This is where using LaTeX and BibTex can come in very handy.

3. **Methods (20 pts)**
   In the **Methods** section, you should describe the implementation and architectural details of your system - in particular, this addresses how you approached the problem and the design of your solution. For those who believe in reproducible science, this should be fine-grained enough such that somebody could implement your model/algorithm and reproduce the results you claim to achieve.

   (i) **Baseline:** You should describe the baseline architecture, stating the appropriate activation function on the last layer of the network, and the loss criterion you optimized.

   (ii) **Experimentation:** Describe your two experimental CNN architectures(parts II.3.ii and II.3.iv) and the U-Net, each in a 2-column table, which the first column indicate the particular layer of your network, the second column state the layer's dimensions (e.g. in-channels, out-channels, kernel-size, padding/stride) and activation function/non-linearity.
   Describe any regularization techniques you used, parameter initialization methods, gradient descent optimization, and how you addressed the class-imbalance problem (the latter in detail).

4. **Results (50 pts)**
   In the **Results** section, you should demonstrate your models' performance compared with the baseline implementation. You should include both of the performance metrics described in Implementation Instructions (II.1) for **each implementation on your test set results**. Please organize these results into a series of concise tables. The formatting is your choice, so long as it is easily interpretable.

   Here are the evaluations, you should include for each architecture:

   (a) A single plot showing both training and validation loss curves;

   (b) Test set pixel accuracy, average IoU and IoU of the classes mentioned

   (c) Visualizations of the segmented output for the first image in the provided test.csv overlaid on the image.

5. **Discussion (40 pts)**
   The **Discussion** section should discuss the benefits (and drawbacks) of the approaches you used and some discussion of why you think you got the results you got, as well as the approaches you followed in this work. At a minimum, please discuss the following important points, but feel free to go beyond this: How did the performance of your implementations differ? Discuss the performance differences with respect to the baseline and compare the other four approaches - Section II.3, parts ii, iii, iv, and v. Provide detailed analysis for the same. Draw insights from the plotted curves, tables and the visualizations.

6. **Authors' Contributions and References (1pt)**
   Each group member must write a small section describing their contributions to the project. Do not forget to cite your references! You should not include any references that aren't cited somewhere in the paper.