CSE 251B: Programming Assignment 3

Winter 2021

Instructions

Due Monday, February 15^{th}:

1. Please submit your assignment on Gradescope. There are two components to this assignment: written homework (Problems I.1-6), and a programming part. As in previous assignments, your report for the programming part will be written using LaTeXor Word in **NeurIPS format** (NeurIPS is the top machine learning conference, and it is now dominated by deep nets - it will be good practice for you to write in that format!). The templates, both in **Word** and LaTeX are available from the 2015 NeurIPS format site.

2. For the programming part, please work in teams of 4 or 5 individuals. Again, **don't forget to include a paragraph for each team member in your report that describes what each team member contributed to the project.**

3. You need to submit all of the source codes files and a *readme.txt* file that includes detailed instructions on how to run your code.

   You should write clean code with consistent format, as well as explanatory comments. Do not submit any of your output plot files or .pyc files, just the .py files and a readme.txt that explains how to run your code.

4. Using any off-the-shelf code is strictly prohibited.

5. Any form of copying, plagiarizing, grabbing code from the web, having someone else write your code for you, etc., is cheating. We expect you all to do your own work, and when you are on a team, to pull your weight. Team members who do not contribute will not receive the same scores as those who do. Discussions of course materials and homework solutions are encouraged, but you should write the final solutions to the written part alone. Books, notes, and Internet resources can be consulted, but not copied from. Working together on homework must follow the spirit of the **Gilligan's Island Rule** (Dymond, 1986): No notes can be made (or recording of any kind) during a discussion, and you must watch one hour of Gilligan's Island or something equally insipid before writing anything down. Suspected cheating has been and will be reported to the UCSD Academic Integrity office.

6. **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.

7. You will be using PyTorch (v 1.4), a Deep Learning library, for the tasks in this assignment. Also you are allowed to only use Numpy, Matplotlib and PIL Imaging library apart from the libraries that come along with the Python installation. Steps to access the UCSD GPU cluster are explained in the following sections.

Learning Objectives

1. Understand the basics of convolutional neural networks, including convolutional and de-convolutional layer mechanics, max-pooling, 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. Learn best practices of transfer learning model by fine tuning a model for Semantic Segmentation.

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. We expect each team member to complete this task individually

For questions 1-4, consider the $(5 \times 5 \times 2)$ input with values in range $[-1, 1]$ 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 $(3 \times 3 \times 1)$.) You can assume a bias of 0.

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

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

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

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

Input Feature maps

Filter

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. **[Note** - the cells on the left are for each channel (this step is **not** in the Stanford example!)] If there are any unused cells after completing the process in the provided tables, place an × in them. The output you get on the right is the output after **combining** the results of each channel. (3 pts)

Filtered Inputs

Output Feature Map

2. What kind of features do the kernels in Part 1 detect? (just describe them in English) (1 pt)

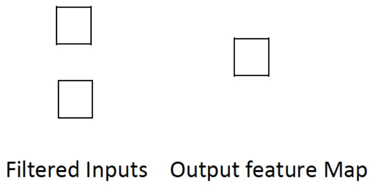
3. Think about what ideal 3×3 patch from each of the input channels would maximally activate the corresponding 3 × 3 filter. Fill in these maximally activating patches in the area below. If there are any unused cells (i.e., cells that would not make any difference) after completing the process in the provided tables, place an × in them. (2pts)

Maximally
Activating Patch

4. **Spatial pooling:** Using your **output feature map** in question 1, apply *max-pooling* using a [2 × 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. (2 pts)

Max Pooled
Output

5. **Dilated Convolutions:** Maxpooling helps the network learn the global context of input feature maps but loses spatial resolution. Dilated convolutions allow for larger filters without adding additional parameters. You can read more on Dilated convolutions here. Consider the same input feature map used for question 1. Also consider the same filters provided for question 1. Fill in the values below after dilated convolution (with dilation factor 2) between the input feature map and the filters. (2 pts)



6. **Number of learnable parameters and feature map dimensions**
   Suppose we had the following architecture:

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

   All convolutions have stride 1 and no zero padding unless mentioned. conv1 has a 5x5 kernel with 5 output channels. conv2 has a 7x7 kernel with 10 output channels. maxpool has 3x3 kernel with stride 2. conv3 has single output channel with a 13x13 kernel and stride=10. fc2 has 512 input units. You are supposed to predict 10 class outputs.
   If ReLU is the activation function between each layer and the inputs are [256 × 256] RGB images, what are:

   (a) **conv1**, **conv2**, **conv3** filter shape
   (b) **conv1** output feature map shape
   (c) **conv2** output feature map shape
   (d) **maxpool** output feature map shape
   (e) **conv3** output feature map shape
   (f) Number of input and output units of **fc1** and **fc2** layer

   Show your work with calculations (6 pts)

Part II

Full Convolutional Network 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 *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.

We will be using **India Driving Dataset** for the task of semantic segmentation. This dataset has pixelwise annotation for 27 object categories. The statistics of the dataset can be found here. We will be using a subset of this dataset containing total ~7k images with a $4:2:1$ split in train, val, test data. 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.

Implementation Instructions

Please familiarize yourself with Pytorch before implementing. One of the good places to start with Pytorch is this tutorial.

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

The following are the implementation steps to follow.

1. **Evaluation metrics**

   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:

   (a) 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 can't be ignored. Therefore, there are alternative metrics that are better at dealing with this issue.

   (b) 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 validation 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 **dataloader.py**.

   For each experiment, Please report the following:

   (a) Average Pixel Accuracy; Do not include *Unlabeled* class(26)

   (b) Average IoU; Do not include *Unlabeled* class(26)

   (c) IoUs for the classes with id - road(0), sidewalk(2), car(9), billboard(17), sky(25).

2. **Dataset loading**

   (a) The dataset has already been uploaded to DSMLP servers. The files in starter code **train.csv**, **val.csv**, **test.csv** contains lines of tuples (*path_ image,path_ label*) for these images and their labels.

   (b) **dataloader.py** contains custom PyTorch Dataset class already completed for you. However, you need to write code for data pre-processing here such as normalization, cropping, resizing and any other transformations you may want to use. Have a look at PyTorch abstract transformations class here. (**Note:** The datasets contain RGB images.)

   (c) Dataloader in **starter.py** uses the dataset object and iterates over your dataset while training.

3. **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 semantic 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 **starter.py**, 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. Use early stopping to train the baseline model. A naive implementation of this model should reach average pixel accuracy of 0.7 and average IoU of 0.55 on validation dataset. **NOTE:** You *might* need to resize/crop your input images such that its dimensions are a power of 2, otherwise the code will throw an error during forward pass. If it does, it would be good practice to think why this is happening.

   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.

4. **Improving baseline model:** The baseline model we trained above is a basic implementation which can be improved.

   (a) 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!

   (b) 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. You must implement your own weighted/balanced loss criterion.

5. **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 semantic segmentation problem using a deep CNN. You are welcome to make a copy of the **basic_fcn.py** file and **starter.py** for this purpose. Use all techniques to improve the model performance and handle the problem of class imbalance.

   (a) Try and come up with an architecture of your own - 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! You can take inspiration from already exisiting models and try to combine their strengths.

   (b) 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?

   (c) 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*:

1. **Abstract:** (5 pt)
   The Abstract should serve as ∼ 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. Please mention any key findings, interesting insights, and final results (i.e., percent correct, not loss numbers)

2. **Introduction:** (5 pt)
   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.

3. **Related Work:** (5 pt)
   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.

4. **Methods:** (20 pt)
   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.

   (a) **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.

   (b) **Experimentation:** Describe your two experimental CNN architectures(parts 5.a and 5.b) 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).

5. **Results** (50 pt)
   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 Evaluation metrics (part 1) for **each implementation on your validation set results**. Please organize these results into a series of concise tables. The formatting is your choice, so long as it is easily interpretable.

   For each architecture include:

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

   (b) Validation 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. Use the color coding mapping in the **dataloader.py** for this.

6. **Discussion** (40 pt)
   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. Please discuss the following important points: How did the performance of your implementations differ? Discuss the performance differences with respect to the baseline (part 3), improved baseline (part 4) and compare the other implementations (part 5a, 5b, 5c). Provide detailed analysis for the same. Draw insights from the plotted curves, tables and the visualizations.

7. **Authors' Contributions and References** (1 pt)
   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.

UCSD GPU cluster

UCSD provides access to its GPU cluster through UCSD Datahub. You are free to use any other platforms but we won't be responsible for any glitches, bugs or delays in them.
Steps to access UCSD datahub are as follows.

1. **Method 1**
   (a) Log into UCSD Datahub using Single Sign On method.
   (b) There will be two notebooks visible for the class. Choose the one that allocates you GPU.
   (c) Once the notebook is spawned, A JupyterHub Dashboard will appear.
2. **Method 2**
   (a) ssh into dsmlp-login.ucsd.edu using *ssh username@dsmlp-login.ucsd.edu.* Before that, you will need to connect through VPN if you are not connected to on campus network.
   (b) Use *launch-scipy-ml-gpu.sh* to launch a Jupyter Notebook with GPU access.

More details on using Datahub is provided on this link

Start early on the PA. Datahub GPUs might get over utilized towards the end of PA and you may not be able to get access to GPU when you need it. Happy Coding !!