Introduction to Deep Learning for Computer Vision
Assignment 6: Convolutional Neural Networks I

Due: March 8th, 2019

Abstract

In this assignment we will implement a configurable Convolutional Neural Network (CNN) script, with both PyTorch's Conv2d module and our custom module.

Please read the instructions carefully.

1 Instructions

1.1 Submission package

Within the homework assignment package, there should be a `submission-package.zip`, which contains the directory structure and empty files for you to get started. Please edit the contents within the file for code, and then create a `zip` archive with the file name `submission-package.zip`, and submit it. **Do not use other archive formats such as `rar` or `tar.gz`.** Also, submit the report in `pdf`, **as a separate file alongside** the archive with the codes.

All assignments should be submitted electronically. Hand written reports are **not** accepted. You can, however, include scanned pages in your report. For example, if you are not comfortable with writing equations, you can include a scanned copy.

1.2 Assignment Report

For this assignment, all execution results should be summarised in an assignment report. Reports should be in `pdf` format. Though not mandatory, students are encouraged to submit their reports written in LaTeX. In the assignment package, you should have been given an empty skeleton file for you to get started.

However, it is **not required** for you to explain your code in the reports. Reports are for discussing results. You **should** however, provide comments in your code, well enough to be understood by directly reading the code.

1.3 Code

All assignments should be in `Python` 3. Codes that fail to run on `Python` 3 will receive 20% deduction on the final score. In other words, do **not** use `Python` 2.7.

For this assignment, you should **not** need to create additional files. Fill in the skeleton files in the submission package. Do **not** change the name of these scripts.

It is **strongly encouraged** to follow `PEP8`. It makes your code much more readable, and less room for mistakes. There are many open source tools available to automatically do this for you.

1.4 Delayed submission

In case you think you will not meet the deadline due to network speed or any other reasons, you can send an email with the `SHA-256` hash of your `.zip` archive first, and then submit your assignment through email later on. This will **not** be considered as a delay.

Delayed submissions are subject to 20% degradation per day. For example, an assignment submitted 1 minute after the deadline will receive 80% of the entire mark, even if it was perfect.

Submission will close 24 hours after the original deadline, as the solution for the assignment will be released to be used for the next assignment.

1.5 Use of open source code

Any library under any type of open source license is allowed for use, given full attribution. This attribution should include the name of the original author, the source from which the code was obtained, and indicate terms of the license. Note that using copyrighted material without an appropriate license is not permitted. Short snippets of code on public websites such as StackOverflow may be used without an explicit license, but proper attribution should be given even in such case. This means that if you embed a snippet into your own code, you should properly cite it through the comments, and also embed the full citation in a LICENSES file. However, if you include a full, unmodified source, which already contains the license within the source file, this is unnecessary. Please note that without proper attribution, *it will be considered plagiarism.*

In addition, as the assignments are intended for you to learn, (1) if the external code implements the core objective of the task, no points will be given; (2) code from other CSC486B/CSC586B students will count as plagiarism. To be more clear on (1), with the assignment, we will release a `requirements.txt` file, that you can use with `pip` to setup your environment. On top, you are also allowed to use `OpenCV3.X`, which we will not include in `requirements.txt` as `OpenCV` installation depends on your own framework.

2 Implementing `model.py/MyNetwork`

2.1 `__init__` (25 points)

Implement the creation of convolution layers according to the comments in the code. The default configuration should result in a network architecture as the following (the module names can be different):

```
MyNetwork(
  (conv_conv2d_0_0): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1))
  (conv_relu_0_0): ReLU()
  (conv_pool_0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv_conv2d_1_0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv_relu_1_0): ReLU()
  (conv_pool_1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv_conv2d_2_0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv_relu_2_0): ReLU()
  (conv_pool_2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc_linear_0): Linear(in_features=128, out_features=128, bias=True)
  (fc_relu_0): ReLU()
  (fc_linear_1): Linear(in_features=128, out_features=128, bias=True)
  (fc_relu_1): ReLU()
  (fc_linear_2): Linear(in_features=128, out_features=128, bias=True)
  (fc_relu_2): ReLU()
  (output): Linear(in_features=128, out_features=10, bias=True)
)
```

2.2 `forward` (5 points)

Use the layers created in `__init__` for the forward pass.

3 Implementing `model.py/MyConv2d`

3.1 `forward` (50 points)

Implement the custom convolution layer. Your implementation should give you identical results compared to the `PyTorch`'s `Conv2d` module, at least for the default configurations.

We will have the following grading guideline for this part of the code:

- 50 points: Implementation gives identical test performance and the speed is over 85% of the original PyTorch's implementation on a CPU machine.

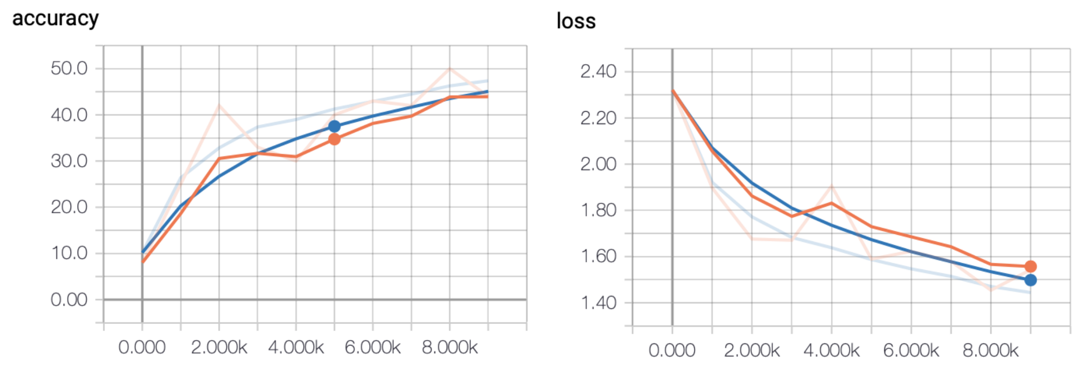

Figure 1: Example result with default parameters.

- 40 points: Implementation gives identical test performance and the speed is over 50% of the original PyTorch's implementation on a CPU machine.
- 30 points: Implementation gives identical test performance.
- 15 points: Implementation does not give identical test performance due to implementation bugs, but the concept of convolution is there, and the output shape is correct.
- 5 points: Output shape is correct.

4 Reporting

4.1 Reproducing default architecture results – Testing (5 points)

With the provided models, you should obtain the following test performance. Your results also for `--conv2d=‘‘custom’’` and `--conv2d=‘‘torch’’` should be identical. Note that the provided models are trained for much more number of epochs instead of the default configuration of 25 epochs.

Table 1: Expected test accuracy.

Configuration	Test Accuracy (%)
Default	60.33

4.2 Reproducing default architecture results – Training (5 points)

With the default parameters, you are supposed to obtain a result similar to Fig. 1. Reproduce these results. In my laptop, this takes around 8 minutes to run the 10,000 iterations.

4.3 CNN architectures (10 points)

Try minimum of three different architectures using the command-line arguments. Report your results on how it changes.