ECE590-10/11: Lab #1
Deep Neural Network Visualization

Hai Li and Yiran Chen

ECE Department, Duke University — September 3, 2019

Objectives

Lab # 1 is designed to help you develop a preliminary understanding on how deep neural networks (DNNs) work. The following instructions will lead you through the process of visualizing the information captured by DNNs, e.g., Convolutional Neural Networks(CNNs). You will be asked to modify part of the code, run the program and explain your observation. The objectives of Lab #1 is to grasp the basic ideas on

- How CNNs see;
- How different layers work; and
- The impact of quantization-based model compression techniques.

You are encouraged to use your own computer to run the code for this lab. Please refer to the in-class NumPy/PyTorch tutorial for environment setup on your computer. For all of the 3 assignments in Lab #1, please use the files in `src/`. If you intend to use the OIT server, please refer to the appendix for environment setup.

Warning: You are asked to complete the assignment independently.

This lab has a total of **30** points. You must submit your report through **Sakai** before **11:59pm, Wednesday, September 11**. Please clearly demonstrate all your results and observations in the report. The final report must be in PDF format. Only the report is needed for this lab. Code submission is not required.

1 How CNNs see

The objective of this task is to visualize *which pixels of the input images affect the neurons most*. Here we use a CNN as an example. CNN is a class of deep neural networks, most commonly applied to analyzing visual images. More details will come in Lecture 4. At the moment, you are not required to fully understand the entire process of a DNN model, but to develop the basic concept.

First download the lab1 package from Sakai and start with `src/Lab1.pynb` All the three assignments in Lab #1 require to modify this file only.

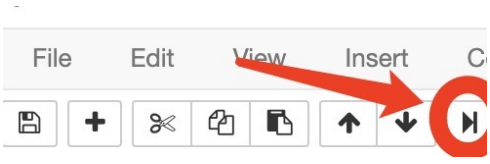

Figure 1: Jupyter Notebook running demo.

Assignment 1 (10 pts)

Start with `src/Lab1.pynb` and run Assignment 1 directly by clicking the forward button (see Figure 1) four (4) times. The code is used to calculate how *sensitive* the CNN is for each pixel of an input image. We prepared three different images, corresponding to `target_example`=0, 1, and 2, respectively. The true class labels for the three examples are king snake, mastiff, and spider, respectively.

(a) (3pt) Change `target_example` each time, run the code, and record your result. Note that the gray color in the resulting image indicates the pixel is meaningless for the classification results. In your report, describe and analyze what you see. Why input pixels show different responses at the output? What is the indication to the classification algorithm design?

(b) (3pt) Compare the results of the three example input images. Describe your observation and understanding. What types of features are more important for classification? What types of images are relatively easier or harder for the classification task by the CNN?

(c) (4pt) When `target_example`=1, the CNN pays different attentions on the dog and the cat in the image. To your best knowledge, what could be the reason that results in the difference?

2 How different layers work

A DNN model is usually considered as a *black-box* because we cannot explicitly explain what "knowledge" is learned when passing an image through layers. A CNN is made up of many layers, and each of which consists of many filters. As the layer is getting deeper, the features captured by the layer are more and more abstract. However, it is difficult to directly visualize or transfer a network model into something that can be visualized and understood by a human.

Instead, we play with input images to see what kind of input patterns induce stronger responses on different layers. To avoid extra training, we have prepared a pretrained (the training is already done and ready for use) VGG-16 model which is commonly used for image classification. The architecture of VGG-16 network is visualized as below:

```
0.  Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
1.  ReLU()
2.  Conv2d(64, 64, kernel_size=3, stride=1, padding=1)
3.  ReLU()
4.  MaxPool2d(kernel_size=2, stride=2, padding=0)
5.  Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
6.  ReLU()
7.  Conv2d(128, 128, kernel_size=3, stride=1, padding=1)
8.  ReLU()
9.  MaxPool2d(kernel_size=2, stride=2, padding=0)
10.  Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
11.  ReLU()
12.  Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
13.  ReLU()
14.  Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
15.  ReLU()
16.  MaxPool2d(kernel_size=2, stride=2, padding=0)
17.  Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
18.  ReLU()
19.  Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
20.  ReLU()
21.  Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
22.  ReLU()
23.  MaxPool2d(kernel_size=2, stride=2, padding=0)
24.  Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
```

```
25.  ReLU()
26.  Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
27.  ReLU()
28.  Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
29.  ReLU()
30.  MaxPool2d(kernel_size=2, stride=2, padding=0)
```

By using the absolute value of one filter's output response as the feedback signal, and applying gradient descent searching on the input image, we can generate an image with particular patterns to approximate the response of the corresponding filter in VGG-16 model.

Assignment 2 (10 points)

You may start with **Assignment 2** in the jupyter notebook file `src/Lab1.pynb`. In this notebook, variable `cnn_layer` denotes the layer index for the VGG-16 network. As the layer index starts from 0, we use `cnn_layer=0` to denote the first layer of VGG-16 architecture, which is a 3×3 convolution with 64 output filters. Variable `filter_pos` represents the filter index to visualize. Suppose you want to visualize the filter in the first layer, the filter index should range from 0 to 63 as there are 64 filters in total in the first layer. By default, we set `cnn_layer` to 2 and `filter_pos` to 0 to help you get started on visualizing the responses of DNNs.

(a) (4pt) Use a fixed variable `filter_pos=0` and change the value of `cnn_layer` to see the visualization of responses in different DNN layers. Plot the visualization results of `all convolution layers (but only convolution layers)` on your lab report. How do responses change in different convolution layers? Can you intuitively explain your understanding?

(b) (2pt) Use a fixed `cnn_layer=10`, change the filter index `filter_pos` to $1, 3, 15, 63, 255$ each time and observe their responses. What happens to the response of these filters? Plot the results and intuitively explain your findings.

(c) (4pt) In VGG-16 model, each `Conv2d` layer is followed by a `ReLU` layer. `ReLU` is one of the most commonly used nonlinear units in CNN models. However, `ReLU` layer is likely to drop incoming information as negative activation values are all clipped to 0. Check the first `Conv-ReLU` pair and the last `Conv-ReLU` pair in the given CNN architecture and visualize the response of filters before and after the `ReLU` non-linearity. What effect does `ReLU` non-linearity have on the filter responses? Does `ReLU` non-linearity have different behaviors on the first convolutional layer and the last convolutional layer? Intuitively explain your findings.

3 What is the impact of quantization

Model compression is used to create light-weight DNN architectures that can be applied to mobile and edge devices. We will further discuss model compression techniques later this semester and practice these techniques in Lab #3. This lab will be an initial test to demonstrate how the quantization technique can affect the model accuracy. *Weight quantization*, as indicated by its name, lowers the data precision of weight parameters to reduce the memory footprint of the DNN model. Moreover, models with lower precision requires less computation resources in execution, which makes it more feasible to deploy on mobile devices.

By visualizing the filter responses under both floating-point models and quantized models, we can get some basic understanding about how quantization affects the quality of output feature maps produced by each DNN layer.

Assignment 3 (10 points)

This assignment will start with **Assignment 3** in the jupyter notebook file `src/Lab1.pynb`. Again, the same pretrained VGG-16 in Assignment 2 will be used. The code responsible for applying quantization on the VGG-16 model has been completely written for you. The default setting of the code block in **Assignment 3** will generate the response of the 4^{th} filter in the 24^{th} layer after applying 16-bit quantization. By using different bitwidth to quantize the weight parameters in the VGG-16, the quality of the filter response, compared to what are obtained in Assignment 2, could degrade dramatically.

(a) (5pt) Configure the quantization precision of weight parameters of VGG-16 by setting different values on `bits` argument in function `fake_quantize`. For example, if you want all of the neural network weights to have only 2 bits, you should set `bits=2` to get a 2-bit quantized VGG-16 model. Configure the quantization precision bits by setting `bits=8,4,3,2` respectively, run the same visualization function in **Assignment 2** again to see the effects of quantization on output filter responses. How does the response of your chosen filter change after enabling the quantization mechanism? How does the response change when we gradually reduce weight precision from 8 bits to 2 bits? Plot the results and intuitively explain your findings.

(b) (5pt) In general, DNN models are very sensitive to the precision of the weight parameters. Once the precision level is lower than a certain level, the model accuracy quickly drops. In practice, `bits=8` is commonly used for compression without too much accuracy drop. The optimal configuration, however, also depends on the model and dataset in use. Based on the results in this assignments, what are the Pros and Cons of conducting quantization? The **LASER** metrics learned in Lecture 1 can be used as the reference of analysis here.

Appendix: Running Lab 1 on the OIT Server

Please visit https://vm-manage.oit.duke.edu/containers and log into your Jupyter Notebook Environment. The files for this lab needs to be downloaded from Sakai as a zip file, then you can upload the whole zip file to the server by clicking the button shown in Figure 2:

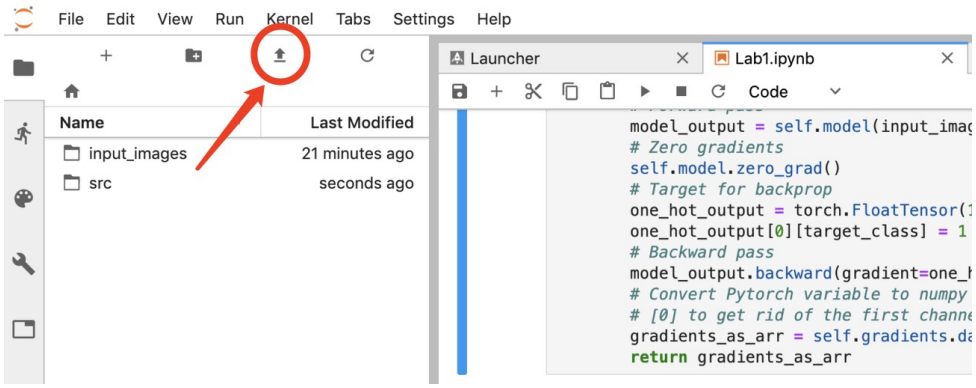

Figure 2: Uploading files Instruction

- Upload the zip file
- Press the '+' button and click on "terminal" in the right-hand side "Launcher" column.
- In the terminal, type $unzip\ Lab\_1.zip$

After that, please use `src/Lab1.ipynb` to start your first lab. Have fun!

Notice: After finishing the lab, please make sure you kill your current process by right-clicking on the `Lab1.ipynb` file and select "Shutdown Kernel", as shown in Figure 3:

Please note that there is a 30-minute idle timeout for GPU access set on the OIT server. If you find that you can no longer access the GPU due to the timeout, simply save your progress, log out, restart your browser and log back in, then you can keep working again.

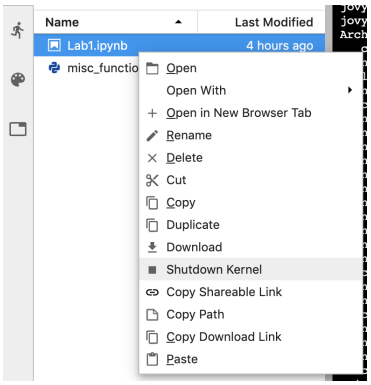

Figure 3: Shutdown kernel before exiting