CS230 Hands-on session 6: “TensorFlow Blitz with PyTorch Bits”

Kian Katanforoosh, Andrew Ng

The goal of this section is to show how to set up a end-to-end pipeline for training a model in Tensorflow.

Specifically, we will cover the following topics:

- Implementing a basic pipeline for classification on the MNIST dataset
- Modifying the pipeline to take in a custom dataset (SIGNS dataset)
- Saving/loading a model
- (Time-permitting) Using a pre-trained CNN

In addition, we’ll briefly go over the implementation of a basic pipeline in Pytorch, another popular DL framework.

Part I - Basic Pipeline

In this part, we’ll walk through an end-to-end pipeline for classification on the MNIST dataset. The MNIST dataset contains images of handwritten digits that are 28x28 with labels 0-9 (examples shown below) and can be loaded directly from Tensorflow.

[https://commons.wikimedia.org/wiki/File:MnistExamples.png]

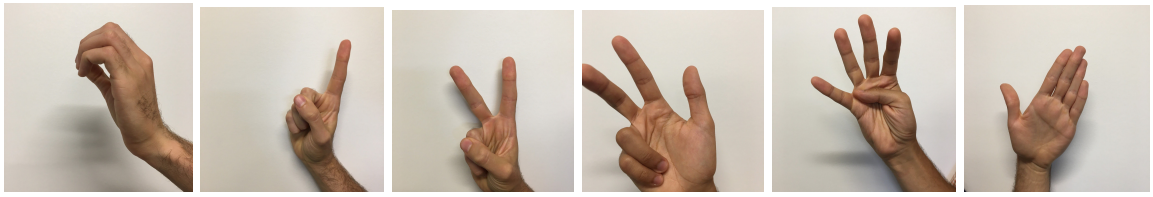
Question: What are the steps for implementing an end to end pipeline?

The code for this part can be found in “tensorflow.ipynb”.
To see a corresponding implementation in Pytorch, check out “pytorch.ipynb”.

Part II - Custom Dataloader

Most class projects use a dataset that is not available in Tensorflow/Pytorch. Let’s see how we can modify our pipeline to read in a custom dataset. We will do this by building a data loader object that will return a batch of (image, label) pairs on each call.

For this part, we will work with a sample of the SIGNS dataset. The dataset contains images of hand signs depicting numbers 0,1,2,3,4 and 5 (examples shown below).



Question: Why do we need a data loader object? Can't we just load the entire dataset in memory to use for training and testing?

Question: Each image has a corresponding label (the number being depicted). What are some ways to store this mapping between images and labels?

The code for this part will be implemented in 'Part II' of "tensorflow.ipynb".

Part III - Saving/Loading a model

Lastly, we'll go over how to save and load a model in Tensorflow. You'll need to save models from every experiment you run, so that you can then load the best model to evaluate on the test set.

The code for this part will be implemented in 'Part III' of "tensorflow.ipynb".

Part IV - Using a pre-trained CNN

In Part 1, we used a simple 2 layer neural network for the classification task. In this section, we'll see how we can use a pre-trained CNN (pre-trained on ImageNet) for our task.

The code for this part has been implemented in 'Part IV' of "tensorflow.ipynb".