

## Exercises

1. If you get a `CUDA_ERROR_OUT_OF_MEMORY` when starting your TensorFlow program, what is probably going on? What can you do about it?
2. What is the difference between pinning an operation on a device and placing an operation on a device?
3. If you are running on a GPU-enabled TensorFlow installation, and you just use the default placement, will all operations be placed on the first GPU?
4. If you pin a variable to `"/gpu:0"`, can it be used by operations placed on `/gpu:1`? Or by operations placed on `"/cpu:0"`? Or by operations pinned to devices located on other servers?
5. Can two operations placed on the same device run in parallel?
6. What is a control dependency and when would you want to use one?
7. Suppose you train a DNN for days on a TensorFlow cluster, and immediately after your training program ends you realize that you forgot to save the model using a `Saver`. Is your trained model lost?
8. Train several DNNs in parallel on a TensorFlow cluster, using different hyperparameter values. This could be DNNs for MNIST classification or any other task you are interested in. The simplest option is to write a single client program that trains only one DNN, then run this program in multiple processes in parallel, with different hyperparameter values for each client. The program should have command-line options to control what server and device the DNN should be placed on, and what resource container and hyperparameter values to use (make sure to use a different resource container for each DNN). Use a validation set or cross-validation to select the top three models.
9. Create an ensemble using the top three models from the previous exercise. Define it in a single graph, ensuring that each DNN runs on a different device. Evaluate it on the validation set: does the ensemble perform better than the individual DNNs?
10. Train a DNN using between-graph replication and data parallelism with asynchronous updates, timing how long it takes to reach a satisfying performance. Next, try again using synchronous updates. Do synchronous updates produce a better model? Is training faster? Split the DNN vertically and place each vertical slice on a different device, and train the model again. Is training any faster? Is the performance any different?

Solutions to these exercises are available in [Appendix A](#).

---

# Convolutional Neural Networks

Although IBM's Deep Blue supercomputer beat the chess world champion Garry Kasparov back in 1996, until quite recently computers were unable to reliably perform seemingly trivial tasks such as detecting a puppy in a picture or recognizing spoken words. Why are these tasks so effortless to us humans? The answer lies in the fact that perception largely takes place outside the realm of our consciousness, within specialized visual, auditory, and other sensory modules in our brains. By the time sensory information reaches our consciousness, it is already adorned with high-level features; for example, when you look at a picture of a cute puppy, you cannot choose *not* to see the puppy, or *not* to notice its cuteness. Nor can you explain *how* you recognize a cute puppy; it's just obvious to you. Thus, we cannot trust our subjective experience: perception is not trivial at all, and to understand it we must look at how the sensory modules work.

Convolutional neural networks (CNNs) emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s. In the last few years, thanks to the increase in computational power, the amount of available training data, and the tricks presented in [Chapter 11](#) for training deep nets, CNNs have managed to achieve superhuman performance on some complex visual tasks. They power image search services, self-driving cars, automatic video classification systems, and more. Moreover, CNNs are not restricted to visual perception: they are also successful at other tasks, such as *voice recognition* or *natural language processing* (NLP); however, we will focus on visual applications for now.

In this chapter we will present where CNNs came from, what their building blocks look like, and how to implement them using TensorFlow. Then we will present some of the best CNN architectures.