Questions

“Training”

1. Describe the idea of a “distributed” encoding.
2. What is data augmentation?
3. How can we make a CNN architecture fully-convolutional?
4. Describe the following strategies for preventing overfitting: data augmentation, data normalization, model averaging, dropout, weight decay, and early stopping.
5. In PyTorch, why is it important to mark whether a model is currently used for *training* or for *testing*?
6. Describe, intuitively, why large weights are indicative of overfitting.

“Generalization”

1. What are the advantages of fully-convolutional architecture?
2. Consider augmenting an image dataset by shifting the training images by a random (small) number of pixels. For which type of neural network would this type of data augmentation have a bigger impact: a convolutional network or a fully-connected network?
3. Suppose you have two networks: network A has 10 million hidden units (artificial neurons) and 1 million weights, network B has 10 million weights and 1 million hidden units. Which of the two networks has higher capacity? Which of the two networks is more likely to overfit?
4. In PyTorch, the implementation of weight decay in an optimizer (e.g. `optim.SGD(model.parameters(), weight_decay=0.0001)`) performs weight decay for **all** parameters, including biases. Why might we want to allow large biases, and only decay neural network *weights*?
5. We discussed data augmentation for images. Suppose that you are instead working with voice recordings. What are some ways to augment a dataset of voice recordings?