In this article we will look at various things concerning the design and training of deep convolutional neural networks.

Convolutional neural network design

Transfer learning: start with an existing network.

This is called transfer learning. We load in a model that has already been trained on a different dataset. We then train our model starting from the parameters that were obtained by the prior training. In this way we can shorten the training time and also achieve a high accuracy.

Kernel sizes

Typically 3 or 5 kernel width works well for most datasets. A larger kernel will take up more memory and be slower to run.

Number of kernels (out-channel of each convolutional kernel)

This is typically set to 32 in the first layer and then doubled in subsequent layers. In this way we obtain more locality as we get deeper into the network

Activation functions

We typically use the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$. An alternative is to use relux $f(x) = max(0, x)$. This can sometimes lead to faster training. In terms of accuracy they are somewhat similar.

Convolutional neural network training

Learning rates

This can be a big make or break for your model. Typically start with a medium learning rate and decrease it after every few epochs. Decay typically refers to some type of exponential decrease of the learning rate over epochs.

In our lectures we have seen different learning rate adjustment methods:

1. Momentum
2. Adagrad
3. RMS prop

4. Combination of both in Adam

These can be helpful to escape saddle points which are a challenge for gradient descent. How does a simple method work?

1. Start with learning rate = .01, divide by 2 (or 5 or 10) after every 10 epochs.

How will you implement this? Sometimes a basic simple method can work very well to train deep networks.

Early stopping

This is a method used for a long time in neural network optimization. Here we stop the search before convergence, thus it is called early stopping. Note that in SGD we have no convergence criterion, we simply stop after some number of epochs.

What number of epochs should you train your model for? This can vary and depends upon the dataset and your network.

Two simple rules:

1. Stop training when training accuracy reaches 100%
2. Stop training when the drop in training accuracy exceeds a threshold δ.

Data shuffling

In SGD we read a single datapoint at a time. Does the order of data affect the optimization? Yes.

In mini batch we select a random set of points and perform our search based on them alone. Here also we can ask if the selection of points in the batch can make a difference.

Batch normalization

This makes the distribution of the data uniform across different layers and can lead to significant improvements in runtime and accuracy. It is analogous to normalizing data like we did in CS 675 to make linear classification work with the same learning rate and be faster and more accurate. We looked at four kinds of normalizations: batch normalization, instance normalization, layer normalization and group normalization.

Training accuracy

How do you know the model is trained? You can look at the training accuracy and when this reaches a 100% you know that you cannot fit the data any better. This typically means your model is trained and reached capacity.

Vanishing gradients

As we get deeper into the network the gradient terms multiply and we get a higher order polynomial. Thus small values will become even smaller resulting in zero gradients (also called vanishing gradients) after surpassing a few layers. We can alleviate this problem with something called residual connections. Relu activation can also help with this.

Optimization

We do stochastic gradient descent to avoid loading a large dataset but also to avoid overfitting.

Image resize

Smaller images require fewer parameters and may be easier to optimize than larger one. Thus resizing images to a smaller size is another useful training trick.