CS 280 Computer Vision
Spring 2018 Efros, Malik, Yu Lecture 13

1 Lecture

We want to find visual similarity across instances (or different views, or whatever). We want to notice patterns, *RE*cognize – see something and remember it again. We'd also like to comprehend samples from the same semantic distribution ("that is a penguin, even though I've never before seen *that* particular penguin in *those* conditions").

Sometimes there might not be anything similar in the pixels at all. But the two images could both still be called the same thing. For example, a steam engine and a bullet train look nothing alike, but they're both called trains.

This is where ML comes in. ML is *data association*. We try to associate a heap of pixels with some label, through some sort of a black box classifier. The classifier should, through training, have "warped the space" such that penguins far apart in pixel space end up being close to each other in the learned representation.

Note: if you have infinite data, k-nearest neighbors is the optimal classifier. Recall: k-nearest neighbors means "ask around for the right answer, and then maybe take a majority vote." Very slow, though.

There are many modern classifiers (e.g. linear classifiers, which only store as many values as there are dimensions in the input). The one we're going to use for computer vision is the neural network.

Neural Networks

Neural networks are particularly suitable for computer vision because they don't need to have a preprocessing/feature extraction step. They operate directly on pixels, and create their own features while using the features at the same time. (Before, we had to go from an image to a feature vector, and then from the feature vector to a class decision.)

Also, even though it's a parametric method with a fixed number of parameters, it has a high capacity. It can remember a huge amount of data while also being able to test things very quickly (like linear classification as opposed to k-NN).

Classically, feature learning for object classification has been hierarchical. The first level typically involves things like edges and colors. Following that, mid-level representations such as segments are extracted from the first-level features. (It's helpful to know about edges and colors before doing segmentation!)

Neural networks follow the same general idea, except they just learn everything. They just figure out whatever representation is best for the task at hand. We also try to have enough layers so that every step – every layer – learns something *just a little bit harder than the previous one*. (Then each transition becomes very simple. The last layer should contain an embedding for which the network is *almost* ready to complete the task.)

It is here that neural networks shine: at learning representations for very high-dimensional, noisy inputs. That's why they're especially good for vision and audio.

Note: if we didn't have nonlinearities, the entire neural network would reduce to single linear classifier. We could collapse all of the linear layers into one; linear times linear times linear is just linear.

Convolutional Neural Networks

With a fully connected structure, huge inputs mean huge numbers of parameters. And when it comes to images, we will have huge inputs. So, as an initial redesign, we will have each neuron connect to only a few neurons in the previous layer.

However, since we want the neurons to specialize (focus on particular features) we have to make them invariant to things like translation and scaling. This means our neurons can't depend on things being at a particular place in the image – always looking at the same local region in the previous layer just won't do. The neurons need to move around somehow.

Therefore each neuron will convolve itself with the entire image in order to find whatever it's looking for. It'll find its eye, if it's looking for eyes, wherever it is in the image. And we can have multiple neurons each looking for different things. *We will let the optimizer figure out what goes in these filters.*

Note: here, we think of a neuron as a filter. The response of every filter is an image. In CS 194-129, we think of a neuron as a single application of one filter, i.e. an object that computes one *dot product.*

The optimizer will want filters to do different things, so as to get more information helpful for solving the task. It will try to keep filters apart (they should never have been the same, due to random initialization).

A **convolutional neural network** involves a sequence of convolutional layers interspersed with activation functions. Incidentally, a 1×1 convolution gets the channels to talk to each other.