CS 280 Computer Vision
Spring 2018 Efros, Malik, Yu Lecture 12

1 Lecture

Last time we talked about various ways of capturing some of the information that we know the visual cortex is processing (e.g. filters that compute oriented derivatives). Our ideas were to either compute one number per filter (like the mean) or, alternatively, to compute some higher-order statistics: mean plus variance, kurtosis, etc.... or just a histogram of values, one per oriented filter.

Then we could accumulate histograms for all the filter response images and use this information to synthesize a novel texture with the same distribution. This shows that such histograms capture enough information to create novel imagery, and are therefore applicable encodings of texture – they capture the "essence."

Finally, we can try to see how all the filters together jointly represent visual information. For this, we can take histograms of "the computational version of textons," i.e. histograms of "visual words." (Recall that textons are vaguely defined textural elements.)

The "visual words" moniker arises though analogy to the bag-of-words representation in text. In the bag-of-words representation, a text is represented as a histogram of words. Since all of the histograms are the same length, we can easily compute statistics over them. Furthermore, these histograms aptly characterize the essence of the documents. Imagine seeing the words "sensory," "brain," and "cerebral cortex" in one text, and "China," "trade," and "exports" in another.

In the case of "histograms of visual words," the idea is that the descriptors, these visual words, are going to be more alike for textures that are similar. So we will use this scheme to represent textures. We will define certain pieces in the visual world to be "words," and then histogram these things in our images.

But how do we get those pieces out of the visual world? Unlike text words, pixel patches almost never repeat. And we want to have bins that grow. So how do we go from all these unique patches to something like a dictionary of visual words?

We use k-means! Recall that k-means is a clustering method for collapsing a huge number of instances into a dictionary of elements, i.e. quantization into clusters. These clusters can then become "visual words."

More specifically, we first apply our filters (we might have 50) to the area around each pixel, and thereby form feature vectors containing those responses. Then we cluster those representations. We will call the clustered pieces of our image the textons (aka visual words).

Note: there are various things we can cluster. In the case of textons we cluster filter bank responses, but we can also cluster raw image patches or SIFT descriptors. Also, our ultimate dictionary of codewords can be either image-specific or *universal* (over a bunch of images).

Classification

So far, we've talked about learning image representations. How do we use those representations to do classification on a new image? We can simply do k-nearest neighbors on the features of the new image!

Empirically, simple texton matching can classify images about as well as humans *if* humans only have about 37 ms to look at the image. In other words, the model captures more or less the same information as humans do with very early/fast pre-attentive vision!

This information is simple texture (e.g. forests might have a lot of vertical stuff, and the model can recognize that). We're treating the structure in our image as texture, and just computing local statistics.

Note: if we scramble an image around while preserving the statistics, it will often still be computed as the same "texture." And the algorithm will be fooled. But this is also the case for many algorithms today.

One weakness: the texton matching method cannot distinguish between instances.

Histograms

There are two kinds of histograms: joint and marginal. If we have a set of one-dimensional things (e.g. pixel intensities, 0 to 255), there's only one histogram to compute. If we have two or more dimensions (e.g. images with three color channels), how do we histogram those?

- We can do **marginal histograms**, i.e. take every feature and do a linear histogram of each. For example, we could have a red histogram, a green histogram, and a blue histogram for each image. This misses out on correlations between the histograms, however.
- We can therefore also do **joint histograms**, where each histogram is no longer 1D (a vector of counts), but nD if we have n dimensions. For each feature, we'll have another dimension. In the case of our RGB images, we could have a three-dimensional tensor.

Joint histograms are better at representing the information, but require a lot more data. So if we don't have enough data, marginal histograms are better (we don't have enough data to capture correlations; we have barely enough to get *something* out).

SIFT

Another way of representing patches is via SIFT descriptors. A SIFT descriptor is just another way to histogram our filter responses. We take an image patch, divide it into a 4×4 grid, and basically histogram the set of oriented filters that we compute within each cell of the grid.

So we'll have a little histogram at every grid cell, and we'll also histogram the x, y locations in a joint fashion. If we have 8 oriented filters, we'll end up with $4 \times 4 \times 8$ dimensions in our descriptors. Again we are taking V1-like information and turning it into a representation!

SIFT descriptors include some invariance (e.g. shift, scale, translational).

The HoG descriptor does the same kind of histograms of different orientations, but on the scale of the full image. The statistics are also slightly different.

Note: both of these descriptors were painstakingly handcrafted to take information from a V1-like representation and make it into a feature.

There are many, many representation processes, each of which cluster and histogram some underlying signal. Other feature descriptors include *shape context* (make features more foveated, i.e. denser at the center, leading to polar coordinate histograms), *geometric blur* (just blur signal to get some invariance), *blur plus half-wave rectification* (compute flow fields in x and y and blur them), and *spatial pyramid matching*.

All of these representations have the same conceptual steps. We first compute some low-level features (e.g. oriented gradients if taking after V1). Then we aggregate them somehow (e.g. via k-means and histograms, maybe involving pooling or averaging or blurring). Basically we compress the space of *filter vectors that will never repeat* into *features that are more repeatable*. And finally we use these features in our visual representation.

But how do we find the best features?

But How Do We Find the Best Features?

Let's try writing an algorithm that finds the best features.

Olshausen and Field said "instead of blindly copying what V1 seems to be doing, can we develop an algorithm

that automatically figures out the best representation for our signal?" The goal: to come up with the optimal set of filters in a data-driven way. They said "if our ultimate goal is to represent natural visual imagery, well, a very small subset of possible 10×10 image patches are ever going to exist in the real world. Can we find the optimal representation for the type of data that we actually see in our world?"

In other words: if the natural world gives us natural images that have *this* kind of structure, what is the best representation for that particular space of images?

They also provided a quantifiable error metric. The idea was that they would try to represent each patch as a linear combination of a set of filters. Then quality could be measured by the distance between an original image patch and the best reconstruction of that image patch using a weighted sum of the k basis vectors. Essentially they wanted the best representation for preserving information about the image.

So far, this is just like PCA, though. And PCA is too generic; it tries to model the average representation for everything. So Olshausen and Field added another constraint, saying that it should be rare to have all of the basis filters participating in a particular patch's representation. Based on studies in neuroscience, usually only a few active cells would ever be firing at once. Hence they added a *sparsity* constraint, saying that a few weights should be close to 1 while the rest should be 0.

And so the representation became "PCA plus sparseness."

The results? Something very much like the oriented filters that the Nobel Prize guys found all those years ago! Yet there was no such information involved in the optimization, nothing directing representations toward gradients or simple/complex edges. Olshausen et al. simply started with random patches and optimized for things that would satisfy their equation, *on natural images*.

So this (oriented filters, like V1, like what *humans* have) seems to be the right representation for the kind of visual data we are seeing. Evolution...?

The Story So Far

- Neurons get more and more specific to a particular visual pattern as we progress down the visual pathway. We start with neurons that are photoreceptors, then go to neurons that are spots, then go to neurons that are oriented simple cells, then go to neurons that are oriented complex cells (i.e. with shift invariance)... and so on.
- As we go to V1, that specificity is in the orientation space, i.e. V1 simple and complex cells are tuned for orientation.
- Convolution (with a linear kernel, and followed by simple nonlinearities) is a pretty good model for this story, i.e. computation in the retina, LGN, and V1.
- Models of visual systems do well when they're hierarchical and mostly feedforward.

Neural networks are built on top of these results, this tendency toward hierarchical layers in our visual models. Perhaps the key paper for this was the one concerning Fukushima's neocognitron work in the 1980s. All of the modern, newfangled stuff was in this paper: the ReLUs, the hierarchical structure, etc.

This is probably the first and most important paper with regard to the conception of neural networks. The only thing it didn't have is backpropagation. It did a kind of unsupervised training instead (a hierarchical version of k-means). So there was no task, no loss: it was just a mechanism for going from pixels to some higher-order features.

Yann LeCun wanted to do something practical. So he went beyond just representation, and applied for the first time a CNN model to digit recognition. He then introduced backpropagation in order to get this supervised pipeline to work.

The magical thing: the low-level, first layer features (i.e. filters) in these hierarchical networks *again* looked like our V1 filters.

In summary: Hubel and Wiesel figured out what was in V1 and told it to the world. Julesz started modeling the idea of textons and how we could represent the visual world with a dictionary of visual elements. Olshausen and Field developed an algorithm that, given natural images, automatically invented a dictionary representation which (amazingly!) simulated the visual cortex. Fukushima modeled the whole thing – not just V1. Finally LeCun applied the neural network model to tasks. And at the end of the day, we arrived at a representation that paralleled Hubel and Wiesel's early observations about the things that happen in the visual cortex. It all came full circle.