CS 194-129 Deep Neural Networks
Spring 2018 Efros Lecture 11

1 Lecture

The Case Against Direct Supervision

In deep learning, the half-life of knowledge ("in X amount of time, half of what we think we know will turn out to be wrong") isn't so long. There's often much more to the story than whatever we think at first.

As it happens, networks don't need to do a lot of things that we think they need to do (find bounding boxes, etc.) in order to classify an image. This is evidenced by the fact that some networks produce the same output even after the image's pixels are scrambled.

Basically, we don't know exactly how our networks produce the output they do! And maybe we're only *fooling* ourselves that they've learned what we meant them to learn.

As another example, networks are happy to recognize an action just by looking for something in a single frame (instead of accumulating observations from the whole *sequence* of frames). In this case, they're not actually doing what we think they're doing; they're doing something much simpler and much sillier. But they still seem to give the right results. Hmm.

So what's happening here? Dataset bias is a problem, and so is our level of complacency. However, dataset bias will not go away – and meanwhile the amount of ways the network can cheat is essentially unbounded. We cannot foresee all of the potential pitfalls and have enough training data to plug those holes; the number of ways the network can cheat is infinite, but the amount of training data we have is finite!

Add to that the fact that ML people don't really care about this problem. In classical ML, if we do well on the test set, we're done. We're good. This is not the case in computer vision, where the real-world distribution is typically much more demanding than the distribution of the training data. We *can't* get a distribution of training data that's going to match our real-world data. There are simply too many scenarios to consider.

The takeaway is that if dataset bias won't go away (even with more data), we need to make better use of our data. Note that *direct supervision is basically rote memorization.* This is good enough for test sets, but not good enough for the real world. We need to do much more than just memorize input/output pairs. We need to make the computer *study harder.*

In other words, we want our models to use the same data in smarter ways, e.g. we want to prevent them from cheating by memorizing. If our computers study harder, they'll learn the material better and then generalize better than those which utilize superficial memorization or pattern matching.

Making Computers Study Harder

Three directions that Prof. Efros's lab is trying to pursue are

- **Self-supervision:** use data as its own supervision.
- **Meta-supervision:** supervise with constraints on the data (rather than with actual data).
- **Curiosity / intrinsic motivation:** incentivize general learning. Don't supervise at all.

Self-Supervision

Context as Supervision

Word2vec: convert from a word to a feature representing that word, such that similar words are close to each other in feature space. **Context prediction:** given a word, predict what other words are going to be in its context (which words are likely to co-occur nearby).

Let's extend context prediction to the case of images. We'll give the computer K patches, each corresponding to a different part of the image. Where should patch A go in the image? Where should patch B go in the image? Notice that our model will have to know about how the world is structured in order to solve this task! It will have to know something about buses in order to tell where parts of a bus should go.

Then, by taking an embedding from one of the later parts of the context prediction network, we use context as a way to create correspondences between representations of images with similar content. It is essentially Image2vec, i.e. a form of representation learning.

In representation learning, compression (autoencoding) tends to not do well on a test set. Therefore, we will try to formulate a prediction task. We will take one portion of the data and try to predict the other portion of the data. For example: in colorization, we can split an image into a grayscale image and a color image, then try to predict one from the other. We can put this into the middle of a split-brain autoencoder: we will both predict the color from the grayscale, and the grayscale from the color.

The context problem allows networks to learn a semantically-minded representation!

How do we define a loss function for our colorization problem? L2 loss doesn't work because there are multiple modes; birds can be blue or red while having the same shape. And L2 will optimize to the means; it will try to please two gods at the same time and end up pleasing neither. We'll end up with a bunch of sepia-colored birds instead of blue or red birds. But we really just want the net to pick one or the other.

So instead we can predict a distribution over color classes. This is multimodal because it can pick either the red or the blue class. In this context, our loss will then be cross-entropy over the color distribution. Note: this might give us *too much* color.

With our handcrafted loss functions, we're like King Midas. We got more than we asked for. But we'd still like a universal loss! For this we turn to GANs.

Conditional GANs

Humans could tell us whether our images are bad. But that's too expensive. So let's use a classifier... or a GAN. In our conditional GAN setup (vanilla GANs don't work well enough), we'll have a generator G which takes x to $G(x)$. Then we'll pass the result of this through D, the discriminator, which will tell us whether the image is real or fake.

G tries to synthesize fake images that fool (the best) D, while D is trying to identify the fakes. We have two opposing algorithms dueling with each other! And we the consumers will benefit.

At least in the context of conditional GANs, from G's perspective D is a loss function. Rather than being hand-designed, it is learned. It's a loss function that's tailored to the problem.

$$\arg\min_G \max_D \mathbb{E}_{x,y}[\log D(x, G(x)) + \log(1 - D(x, y))]$$

The discriminator has to say whether the pairs of (image, generated image) are real or fake.

Meta-Supervision

Here, we might not provide the exact correct answer, but instead just say "be part of this set." (To reiterate, we won't say which *instance* of the set.) One type of meta-supervision is *cycle-consistency.* In this, we have a pair of images and want to find the correspondences between them.

We can find for every pixel in one image a flow vector that goes to the correct pixel in the other image. However, we don't have an official ground truth. So cycle-consistency says that if we apply the inverse, we should end up at the same place. (This is 2-cycle consistency: $F_{i,j} \circ F_{j,i} = 0$. The composition of the two functions should do nothing.)

We can also do 3-cycle consistency: if we go to the correspondences across three images, we should end up at the same point. (We really only need to worry about 2- and 3-cycle consistency.)

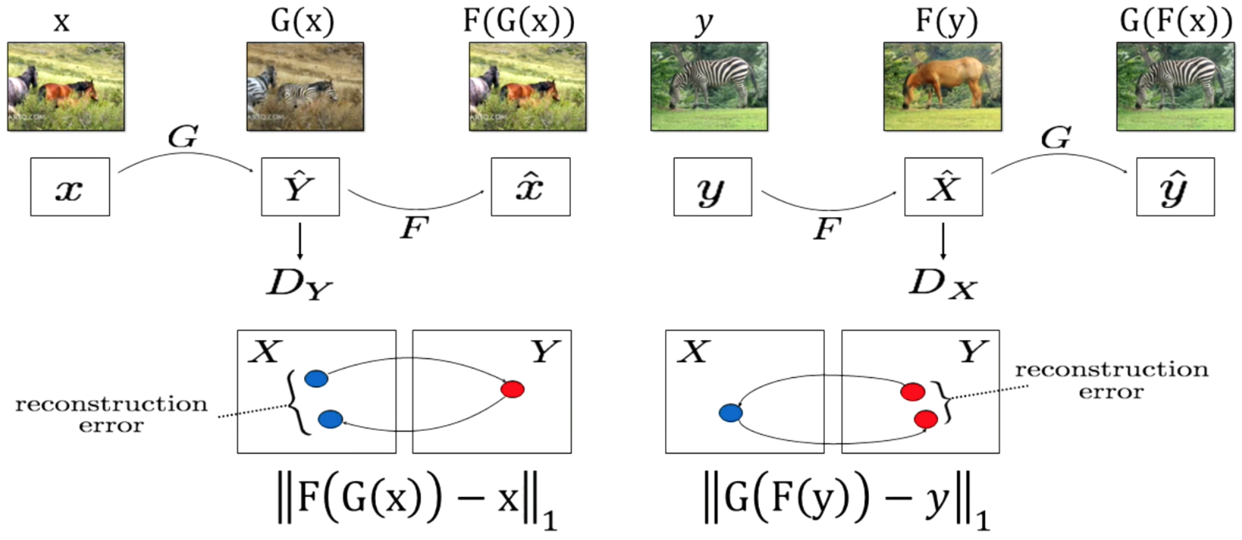
Now, we're trying to use cycle-consistency as supervision. How do we know what went wrong? We can look at the amount of inconsistency: how far we end up from the starting point. The only problem is that we can be both consistent and wrong. So we might need to include some anchors, e.g. synthetic data where we know the actual correspondences. Then we can do, say, 4-cycle consistency where one of the correspondences is the ground truth.

This supervision is theoretically pretty weak, but notice that we no longer need any actual annotated labels. The supervision is now a property of the data. And regardless of any perceived weakness, in practice this method works pretty well!

In the real world, we often don't have access to ground-truth correspondences. Say there are two domains. We can use meta-supervision i.e. cycle-consistency, and train a model G with two discriminative losses:

$$\begin{tikzcd} \boxed{X} \arrow[r, bend left, "G"] \arrow[d] & \boxed{Y} \arrow[l, bend left, "F"] \arrow[d] \\ D_X & D_Y \end{tikzcd}$$

Figure 1: CycleGAN architecture. G takes us from image X to image Y, while F takes us from Y to X. There is a class (or style) for both the X images and the Y images.



source: slides from Alexei Efros's CS 194-129 lecture

We use D to make sure that an image is indistinguishable from a class of images. On the left: we apply G to x and end up with some example $\hat{Y}$ in the class of zebras. Then we back-translate and apply F to it, and see that reconstruction error is low: when we go back, we should be very close to what we started with.

This is basically an autoencoder with deep supervision – an adversarial loss in the bottleneck. The CycleGAN in the example encodes a horse in the domain of the zebra.