1 Intro Lesson 3

So far, we've talked about new networks in general. But if you know something about your data, for example, if it's an image or a sequence of things you can do a lot better.

2 Color Quiz

The idea is very simple. If your data has some structure and your network doesn't have to learn that structure from scratch, it's going to perform better. Imagine for example, that you're trying to classify those letters, and you know that color is really not a factor in what makes an A an A. What do you think would be easier for your classifier to learn? A model that uses the color image or a model that only looks at the grayscale.

3 Color Solution

Intuitively, if a new letter comes up in a color that you have never seen before it's going to be lot easier for your model that ignores the color to begin with to classify that letter.

4 Statistical Invariance

Here's another example. You have an image and you want your network to say it's an image with a cat in it. it doesn't really matter where the cat is, it's still an image with a cat. If your network has to learn about kittens in the left corner and about kittens in the right corner, independently, that's a lot of work that it has to do. How about you telling it instead, explicitly, that objects and images are largely the same whether they're on the left or on the right of the picture. That's what's called translation invariance,. Different positions, same kitten. Yet another example. Imagine you have a long text that talks about kittens. Does the meaning of kitten change depending on whether it's in the first sentence or in the second one? Mostly not. So if you're trying to network on text, maybe you want the part of the network that learns what a kitten is to be reused every time you see the word kitten, and not have to relearn it every time. The way you achieve this in your own networks is using what is called weight sharing. When you know that two inputs can contain the same kind of information, then you share the weights and train the weights jointly for those inputs. it is a very important idea. Statistical invariants, things that don't change on average across time or space, are everywhere. For images, the idea of weight sharing will get us to study convolutional networks. For text and sequences in general, it will lead us to embeddings and recurrent neural networks.

5 Feature Map Sizes Quiz

Imagine that you have a 28 by 28 image. You run a 3 by 3 convolution on it with an input depth of 3 and an output depth of 8. What are the dimensions of your output feature maps? When you're using same padding with a stride of 1, when you're using valid padding with a stride of 1, and when you're using valid padding with a stride of 2.

6 Feature Map Sizes Solution

If you're using the so-called same padding and a stride of 1, the output width and height are the same as the input. We just add zeroes to the input image to make the sizes match. If you use the so-called valid padding and a stride of 1, then there is no padding at all. So if you want to fit your little filter on the input image without doing any padding, you're going to have to remove one row and one column of the image on each side. So you're left with 26 features in each of the maps at the output. If in addition you use a stride of

2, then you only get half as many outputs. So 13 in width and 13 in height. In all cases, the output depth isn't changed

7 Convolutions continued

That's it. You can build a simple covenant with just this. Stack up you convulsions, which thankfully you don't have to implement yourselves, then use trieds to to reduce the dimensional ID and increase the depth of your network, layer after layer. And once you have a deep and narrow representation, connect the whole thing to a few regular fully connected layers and you're ready to train your classifier. You might wonder what happens to training, into the chain rule in particular, when you use shared weights like this. Nothing really happens. The math just works. You just add up the derivates from all of the possible locations on the image.

8 Explore The Design Space

Now that you've seen what a simple convnet looks like, there are many things that we can do to improve it. We're going to talk about three of them, pooling, one by one convolutions and something a bit more advanced called the inception architecture. The first improvement is a better way to reduce the spatial extent of your feature maps in the convolutional pyramid. Until now, we've used striding to shift the filters by a few pixel each time and reduce the future map size. This is a very aggressive way to downsample an image. It removes a lot of information. What if instead of skipping one in every two convolutions, we still ran with a very small stride, say for example one. But then took all the convolutions in a neighborhood and combined them somehow. That operation is called pooling, and there are a few ways to go about it. The most common is max pooling. At every point in the future map, look at a small neighborhood around that point and compute the maximum of all the responses around it. There are some advantages to using max pooling. First, it doesn't add to your number of parameters. So you don't risk an increasing over fitting. Second, it simply often yields more accurate models. However, since the convolutions that run below run at a lower stride, the model then becomes a lot more expensive to compute. And now you have even more hyper parameters to worry about. The pooling region size, and the pooling stride, and no, they don't have to be the same. A very typical architecture for a covenant is a few layers alternating convolutions and max pooling, followed by a few fully connected layers at the top. The first famous model to use this architecture was LENET-5 designed by Yann Lecun to the character recognition back in 1998. Modern convolutional networks such as ALEXNET, which famously won the competitive ImageNet object recognition challenge in 2012, used a very similar architecture with a few wrinkles. Another notable form of pooling is average pooling. Instead of taking the max, just take an average over the window of pixels around a specific location. It's a little bit like providing a blurred low resolution view of the feature map below. We're going to take advantage of that shortly.

9 Convolutions

But first, I want to introduce you to another idea, it's the idea of one by one convolutions. You might wonder, why might one ever want to use one by one convolutions? They're not really looking at a patch of the image, just that one pixel. Look at the classic convolution setting. It's basically a small classifier for a patch of the image, but it's only a linear classifier. But if you add a one by one convolution in the middle, suddenly you have a mini neural network running over the patch instead of a linear classifier. Interspersing your convolutions with one by one convolutions is a very inexpensive way to make your models deeper and have more parameters without completely changing their structure. They're also very cheap, because if you go through the math, they're not really convolutions at all. They're really just matrix multiplies, and they have relatively few parameters. I mention all of this, average pooling and one by one convolutions because I want to talk about a general strategy that has been very successful at creating covnets that are both smaller and better than covnets that simply use a pyramid of convolutions.

10 Inception Module

It's called an inception module. It's going to look a little more complicated. And you can skip this section if you'd like. It's not essential to the rest of the lecture. The idea is that at each layer of your cognate you can make a choice. Have a pooling operation, have a convolution. Then you need to decide is it a 1 by 1 convolution, or a 3 by 3, or a 5 by 5? All of these are actually beneficial to the modeling power of your network. So why choose? Let's use them all. Here's what an inception module looks like. Instead of having a single convolution, you have a composition of average pooling followed by a 1 by 1, then a 1 by 1 convolution, then a 1 by 1 followed by a 3 by 3. Then a 1 by 1 followed by a 5 by 5. And at the top, you simply concatenate the output of each of them. It looks complicated, but what's interesting is that you can choose these parameters in such a way that the total number of parameters in your model is very small. Yet the model performs better than if you had a simple convolution.

11 Conclusion

This is one of the fun things about neural networks because you have this general framework and lots of small building blocks that you can assemble in that framework. You can explore lots of ideas quickly and come up with interesting model architectures for your problem. Next, we're going to look at how to handle text as an input to a deep model.

12 Next Assignment ConvNets

The next assignment takes the deep networks that you've trained so far and turns them into convolutional nets. There are lots of architectures that you could try on this problem, including inception. Note though that this is where GPUs begin to shine. Covenants take a lot of compute. Well, the code in the assignment should run final CPU, and that's all you really need to complete the assignment. Doing a lot of exploration might not be practical, unless you have a big GPU, and modify the code to use it.