CS 280 Computer Vision
Spring 2018 Efros, Malik, Yu Lecture 14

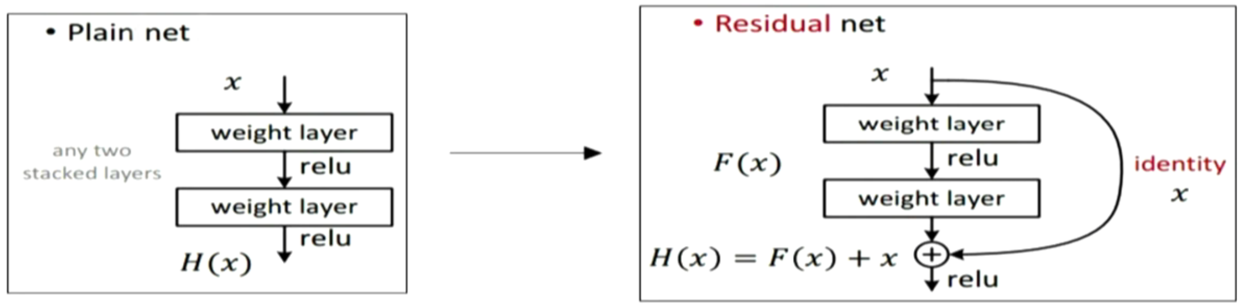
1 Lecture

Pooling layers add robustness to translation, and are inspired by complex cells. A complex cell also takes the loudest cell within a local region, and doesn't care about exactly which position it came from.

Empirically, it seems to be easier to optimize networks that are deeper and narrower, even though the same networks can theoretically also be represented with extremely wide two-layer networks.

ResNet

ResNet is 152 layers. And aside from its depth it'd be a pretty standard CNN if it weren't for one thing: *skip layers*, i.e. a way to navigate the network beyond just "feedforward." These allow for the skipping of any particular layer, or even two or three (and so on). "You can either go through the weights, or you can go directly." If you go directly, you do nothing, meaning an identity.



source: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

This forces the skipped layers to be some residual: "how can we change things from what we had before; what's the delta that we're adding?" The backup plan is always "do nothing," and alternatively we can add something to what we already have. In a standard network, each layer creates something new from what it got, while in a ResNet, we start with an initial solution and at every layer add or subtract something extra to refine it.

A ResNet works very well as a multi-scale type of thing. We start with a coarse answer and progressively refine it. It also allows us to make the network very deep without worrying about a lack of data. The ResNet does not *have* to use all the layers; it can adjust how many it actually uses.

The only problem: very long training times.

Transfer Learning

Neural networks have a huge volume of parameters, and are generally thought to require a huge volume of data to train. However, this is not always true. We do need a lot of training data of some sort, but it doesn't need to be training data for the particular problem at hand. *Transfer learning* will take a network that does something else and tune it to the currently relevant problem.

A CNN will embed an image in a vector – potentially many vectors – as part of the forward process. According to a study done on ImageNet: deep features, i.e. higher-level features from later in the network, actually end up self-organizing themselves according to supercategory. Features for each supercategory will all be close to each other, and apart from those of other supercategories. Note: this is only the case for embeddings in the deepest layers.

This is amazing because we're talking about *supercategories*: things like "invertebrate," "vehicle," and "bird." The network was never told that a train, a plane, and a bus are somehow related, but it ended up representing them similarly anyway. Likewise, it was never told that a raven, a rock dove, and a black-capped chickadee are of the same supercategory... but it could figure this kind of thing out on its own.

The bottom line: ImageNet was given only a thousand flat categories with no relationships between them, and yet it discovered these supercategory relationships by itself. It realized that a bus and a train are closer to each other than a bus and a St. Bernard. It found commonalities and used them to do this grouping.

Also, similar features just correspond to similar images – and not only "another dog," but "another dog with a similar pose!" Feature-wise, networks are learning something more than what they were trained on. While learning to recognize dogs, the network was, additionally, inadvertently learning intermediate concepts like parts and pose and geometry... the kinds of features that might be applied to many task domains.

So at the end, the exciting thing: we can use these features as generic representations. We can get rid of the last layer and use the deep "Image2vec" embeddings for other things. Now we might train on ImageNet, because ImageNet is large enough, and then repurpose the embeddings for other tasks just by swapping out the last few layers and updating only those. (The early portion of the network should already be producing useful features.)

Suddenly, we only need a little bit of data to update the few parameters in the final layers. Or, if we have enough data, we can update the rest of the network too and just "fine-tune" it.

Fancier Networks

Siamese Networks

Siamese networks are for comparing two things. We might want to have a binary judgment: are these two things the same or different? (Maybe we don't have enough data to train a full detector.) All we want to know is whether two things are the same or not.

We will again go from images to vector representations. We'll have a CNN for one image and a CNN for the other image, where weights are shared between the two. They'll spit out representations and we'll compare them: things that are the same should be close in the representation space, and things that are different should be farther away.

For training, we'll pass in a pair of images of the same thing and minimize the distance between their embeddings. We'll also pass in pairs of images of different things and make sure they're far apart from each other. The loss for that is

$$L_n(x_q, x_n) = \max(0, m^2 - \|x_q - x_n\|_2^2)$$

where x_q is one image's embedding and x_n is the other image's embedding. m is the desired margin. Note: if we didn't pass in images of different examples, the network would just push everything together in the joint embedding space.

Multi-Modal

Instead of having two CNNs in our Siamese network, we can have one CNN and then an RNN for text embeddings. In other words, we can compare images and text – it doesn't have to be images and images. And of course our additions aren't limited to text. We could compare images and audio too.

Multi-Task

We can also have multiple final layers (in 194-speak, multiple "heads" coming out of the network). Each of these outputs will be used for a different task. If we do many things at once, the hope is that each one will get better (because representation will be shared and accumulated).

Generic DAG

The graph can be set up in an arbitrary fashion, with arbitrary pieces, as long as it is a DAG. Then it will be differentiable via backpropagation. For instance, an RNN must be unrolled in order to be usable.

Fully Convolutional

Maybe we want to operate on variable-sized images with same-sized outputs. In short, we want to compute outputs on a per-pixel basis. Naively, we could duplicate a CNN for every pixel. However, this is bad and weird. So the fully convolutional scheme simply gets rid of the last fully connected layers and replaces them with CONV layers. Now the input can be any size, since everything is now represented in terms of it.

Note: in order to prevent a loss of resolution, we typically add skip layers that allow the end of the network to access the earlier layers in all of their high-resolution glory. The low resolutions (i.e. later layers) give us the big semantic understanding, so here we're just combining them with their high resolution brethren.