19 Convolutional Neural Networks

CONVOLUTIONAL NEURAL NETWORKS (ConvNets; CNNs)

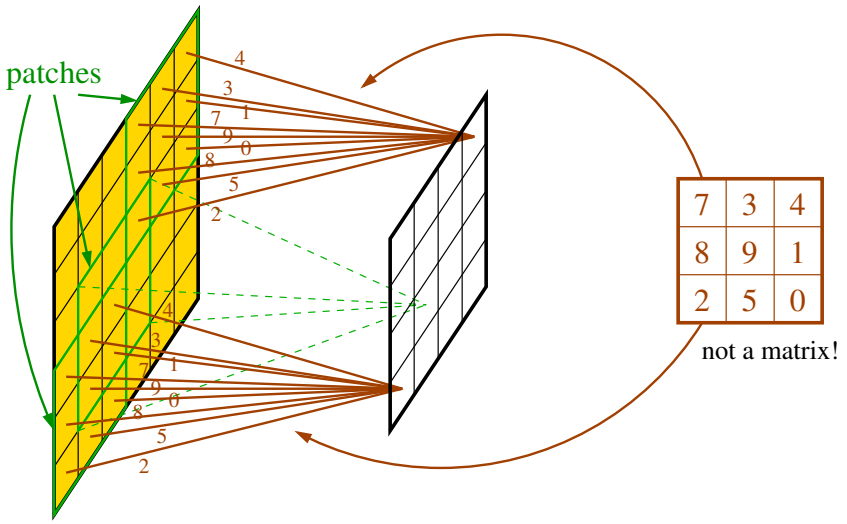
[Convolutional neural nets drove a big resurgence of interest in neural nets starting in 2012. Often you'll hear the buzzword deep learning, which refers to neural nets with many layers. Most image recognition networks are deep and convolutional. In 2018, the Association for Computing Machinery gave the Alan M. Turing Award to Geoff Hinton, Yann LeCun, and Yoshua Bengio for their work on deep neural networks.]

Vision: inputs are images. 400×400 image = 160,000 pixels.
If we connect them all to 160,000 hidden units $\rightarrow$ 25.6 billion connections.
[With so many weights, the network is very slow to train or even to use once trained.]

[Remember that early in the semester, I told you that you can get better performance on the handwriting recognition task by using edge detectors. Edge detectors have two interesting properties. First, each edge detector looks at just one small part of the image. Second, the edge detection computation is the same no matter which part of the image you apply it to. Let's apply these two properties to neural net design, plus one new idea: we'll learn the edge detectors instead of hard-coding them.]

ConvNet ideas:

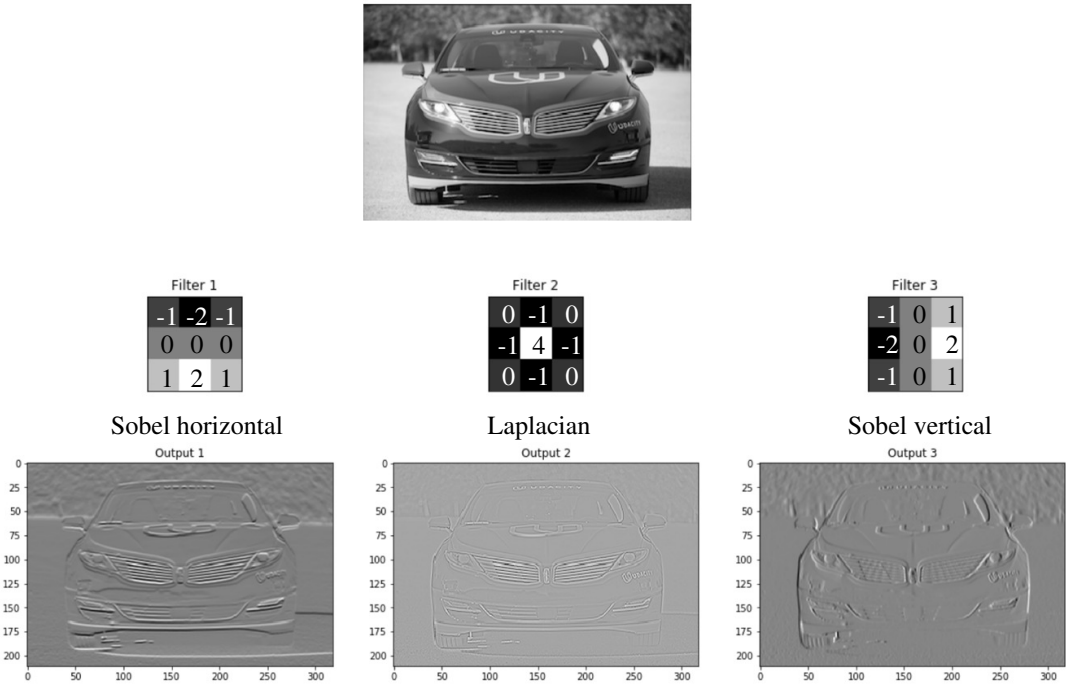
- Local connectivity: A hidden unit connects only to a small patch of units in previous layer.
  [A unit in the first hidden layer doesn't look at the whole image. It looks only at a small number of pixels—typically 9, 25, or 49 pixels. This speeds up both training and classification considerably.]
- Shared weights: Groups of hidden units share same set of weights, called a filter aka mask aka kernel.
  Each filter operates on every patch of image.



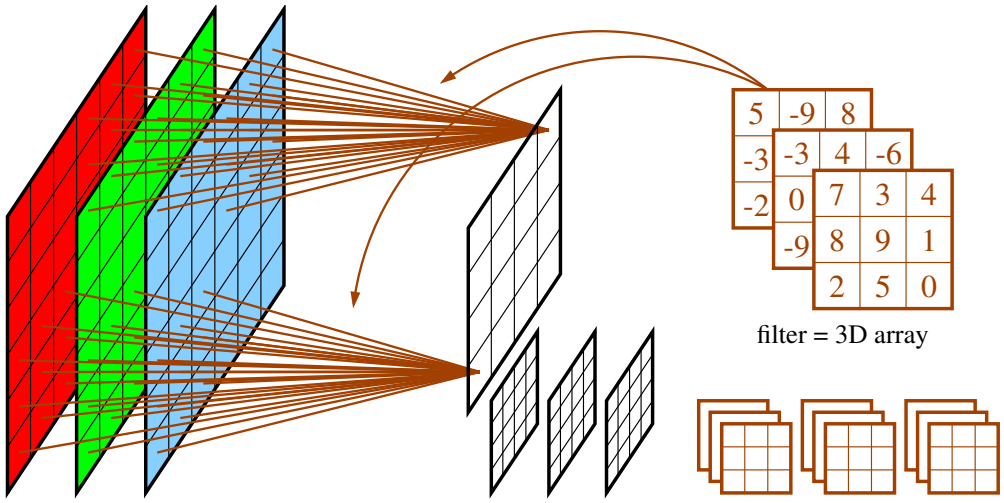
convlayer.pdf [Applying a filter to an image. Every hidden unit uses the same nine shared weights. In this example, a 6×6 image is covered by 16 overlapping 3×3 patches, yielding a 4×4 activation map of hidden units. We learn the filter by backpropagation.]

If image size is $J \times K$ and filter size is $M \times M$, the activation map is $(J - M + 1) \times (K - M + 1)$ hidden units—one for each patch.

A convolutional layer learns multiple filters. There is one activation map per filter. A channel is an activation map, OR another dimension such as the red/green/blue channels of an input image.



caredges.pdf (Cezanne Camacho) [Two Sobel filters (one horizontal, one vertical) and a Laplacian filter applied to an image, yielding three activation maps (three channels). Note that these filters were not learned by a CNN (but they could be).]



convchannels.pdf [A convolutional layer (of edges). If a layer's input has more than one channel, then each filter is represented by a three-dimensional matrix. The set of all filters is represented by a four-dimensional matrix.]

[The output of a convolutional layer has multiple channels, and usually so does the input. The layer's output has one channel per filter, and these channels becomes inputs to downstream convolutional layers. The input to the neural network often has multiple channels too; most commonly, the color channels of a color image.]

If edge layer l has $C^{(l-1)}$ channels in and $C^{(l)}$ channels out ($C^{(l)}$ filters),
- # of weights/filter = $C^{(l-1)} \times M \times M$;
- # of weights in layer = $C^{(l-1)} \times C^{(l)} \times M \times M$;
- # of units out = $C^{(l)}\times$ # of patches = $C^{(l)} \times (J - M + 1) \times (K - M + 1)$.

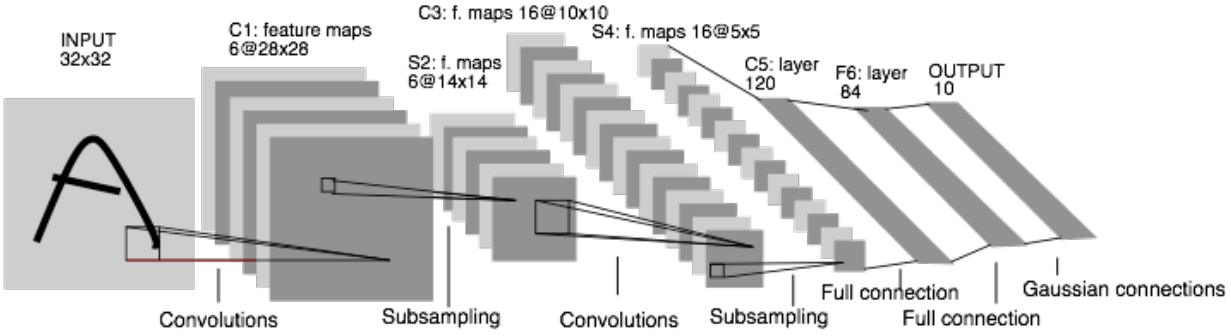
Typically, each convolutional hidden unit ends with a ReLU activation.
[But there are exceptions in many modern CNNs.]

Options for bias terms:
- Untied bias: $C^{(l)} \times (J - M + 1) \times (K - M + 1)$ bias terms—one per unit out.
- Tied bias: $C^{(l)}$ bias terms—one per filter/channel out.
- No bias terms. [This option is usually *not* immediately followed by a ReLU.]

[Untied bias terms are a lot of extra weights, sometimes more weights than the filters! Sometimes they give better test accuracy, but not always, so try validating both ways.]

Benefits of shared weights:
- Much less memory needed. [Better cache behavior too.]
- Regularization. [It's unlikely that a weight will become spuriously large if it's used in many places.]
- If one filter learns to detect edges, *every* patch has an edge detector.
  [Because the filter that detects edges is applied to every patch.]
  ConvNets exploit repeated structure in images, audio.
- A filter destined to become an edge detector learns on edges in *every* part of every image.
  [So it can learn the idea faster.]

[In a neural net, you can think of hidden units as added features that we learn, as opposed to added features that you code up yourself. Convolutional neural nets take them to the next level by learning features from multiple patches simultaneously and then applying those features everywhere, not just in the patches where they were originally learned.]



LeNet5.png Architecture of LeNet5.

[ConvNets were first popularized by the success of Yann LeCun's "LeNet 5" handwritten digit recognition software. LeNet 5 has six hidden layers! Hidden layers 1 and 3 are *convolutional layers* with shared weights. Layers 2 and 4 are *pooling layers* that make the image smaller, with no weights at all. Layers 5 and 6 are fully-connected layers of hidden units with no shared weights. A great deal of experimentation went into

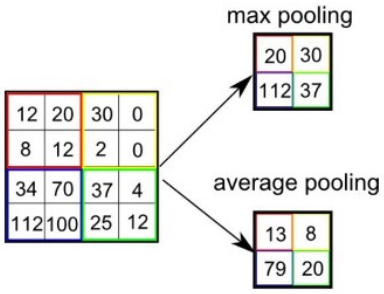
figuring out the number of layers and their sizes. At its peak, LeNet 5 was responsible for reading the zip codes on 10% of US Mail. Another Yann LeCun system was deployed in ATMs and check reading machines and was reading 10 to 20% of all the checks in the US by the late 90's. LeCun is one of the Turing Award winners I told you about earlier.]

[Show Yann LeCun's video LeNet5.mov, illustrating LeNet 5.]

Downsampling

[At the output of LeNet 5, we have to compress the information down to a single 10-unit output. Experience shows that this is best done by slowly compressing the information in the image through a sequence of layers, rather than connecting a very large layer of hidden units directly to the output. This observation echoes classic image processing techniques that were developed before neural networks. The two popular methods of downsampling are called *pooling* and *strided convolution*.]

Max pooling: Reduce a $J \times K$ image to $\lceil J/2 \rceil \times \lceil K/2 \rceil$ or $\lceil J/3 \rceil \times \lceil K/3 \rceil$ [as illustrated].

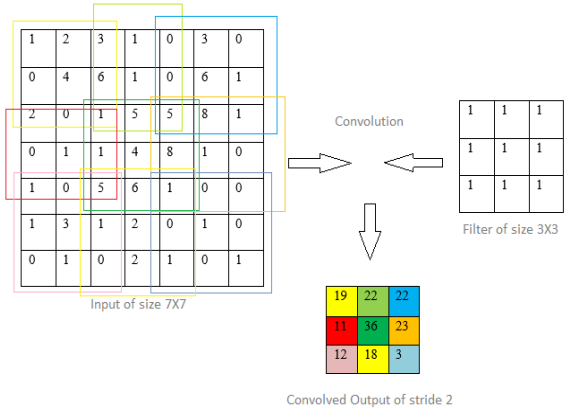


maxavgpool.pdf (Ekpenyong et al.) [Max pooling and average pooling.]

Average pooling: likewise, but each unit out is the average of four units in.
[Average pooling was used in LeNet 5, but max pooling seems more popular now.]

Pooling layers have *no weights*. Nothing to train!
[But you still have to think carefully about how to do backpropagation through them.]

Strided convolution: Patches overlap less (or not at all).



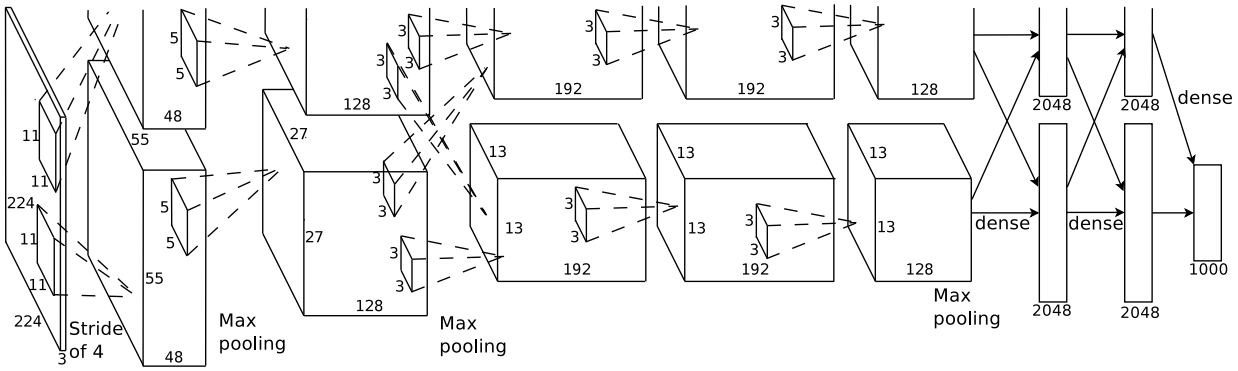
stride.pdf (Vadlamani & Patel) [Strided convolution.]

AlexNet

[I told you three lectures ago that neural net research was popular in the 60's, but the 1969 book *Perceptrons* killed interest in them throughout the 70's. They came back in the 80's, but interest was partly killed off a second time in the 00's by . . . guess what? By support vector machines. SVMs work well for a lot of tasks, they're much faster to train, and they more or less have only one hyperparameter, whereas neural nets take a lot of work to tune.]

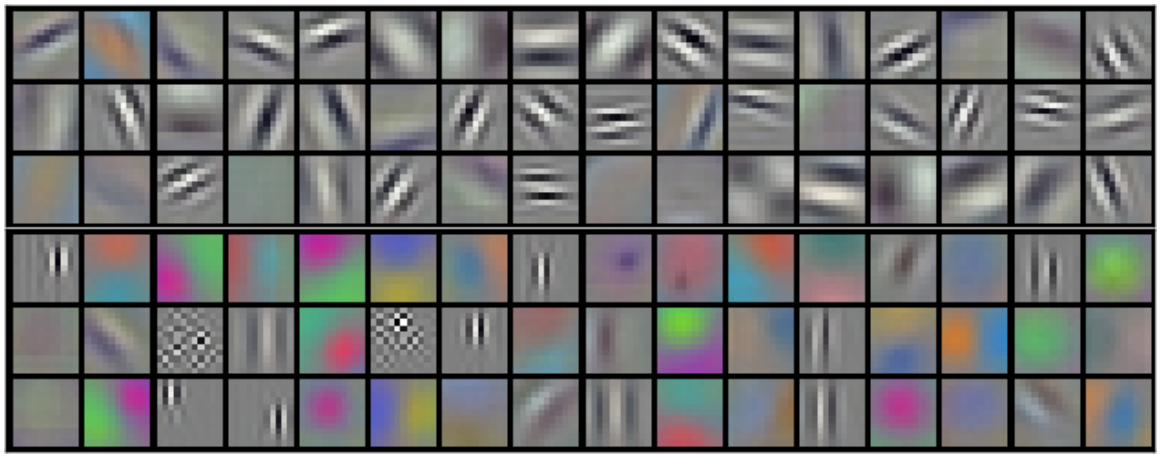
[Neural nets are now in their third wave of popularity. The single biggest factor in bringing them back is probably big data. Thanks to the internet, we now have absolutely huge collections of images to train neural nets with, and researchers have discovered that neural nets often give better performance than competing algorithms when you have huge amounts of data to train them with. In particular, convolutional neural nets are now learning better features than hand-tuned features. That's a recent change.]

[The event that brought attention back to neural nets was the ImageNet Image Classification Challenge in 2012. The winner of that competition was a neural net, and it won by a huge margin, about 10%. It's called AlexNet, and it's surprisingly similarly to LeNet 5, in terms of how its layers are structured. However, there are some recent innovations that led to their prize-winning performance, in addition to the fact that the training set had 1.4 million images: they used ReLUs, dropout, data augmentation, and GPUs for training.]



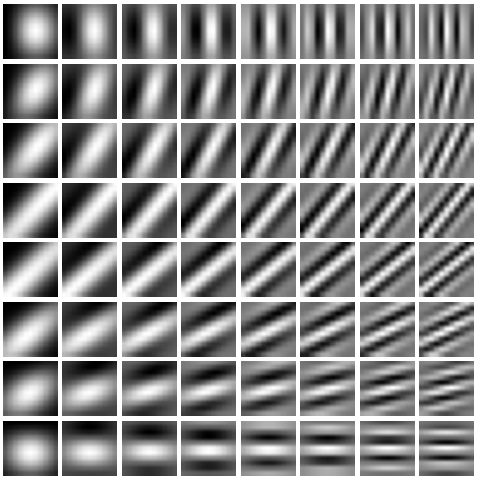
alexnet.pdf (Krizhevsky et al., 2012, 2017) [Architecture of AlexNet.]

[When ConvNets were first applied to image analysis, researchers found that some of the learned filters are edge detectors! Here are the first layers of filters learned by AlexNet.]



filtersalex.png (Krizhevsky et al., 2012, 2017) [Filters learned by the first layer of AlexNet.]

[Not all of the features are edge detectors; many of them are more concerned with color. But more than half of them resemble mathematical functions called Gabor filters, which detect edges and also textures.]

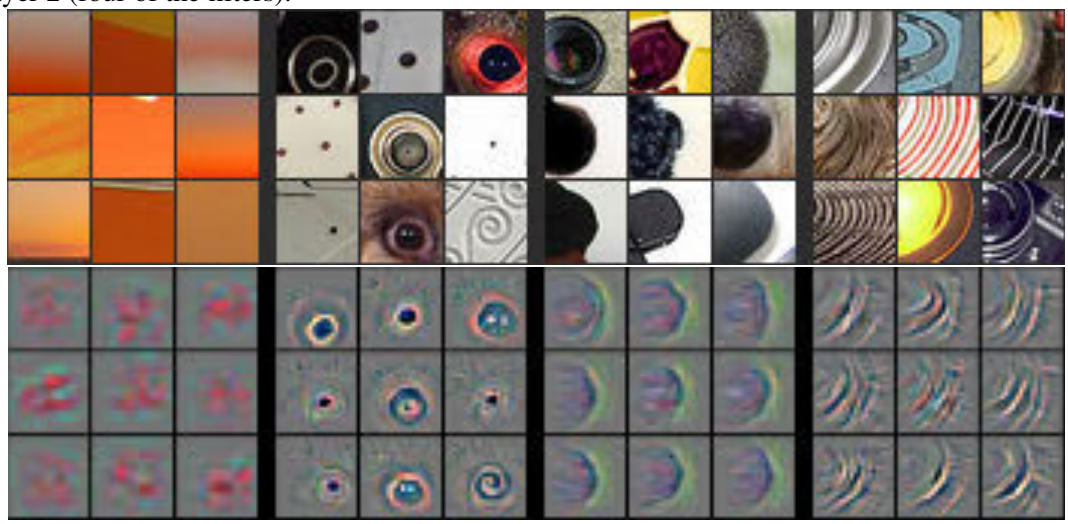


gabor.pdf (Bishop, Figure 10.11) [Gabor filters. *Not* learned; these are math functions.]

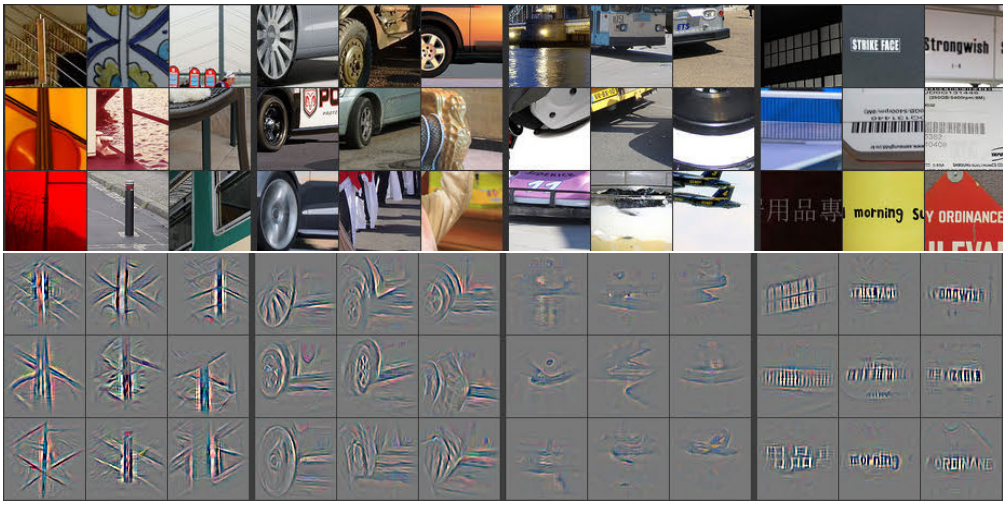
[AlexNet learned some simple color-specific edge detectors, but I find it noteworthy that the higher-frequency texture detectors are not sensistive to color at all. Apparently, the CNN decided it can separate fine texture from color.]

[Unfortunately, we can't just draw the filters learned by subsequent convolutional layers, because they're 3D arrays that don't carry much visual information. Instead, Zeiler and Ferguson (2013) have a technique where they determine which patches from the training set most trigger a particular filter, and they draw nine of those. They also have a technique for determining which pixels of those patches are most relevant to triggering the filter, and they plot the relevance of each patch pixel. This reveals that in the third example for convolutional layer 4, the filter is primarily responding to the grass in these images.]

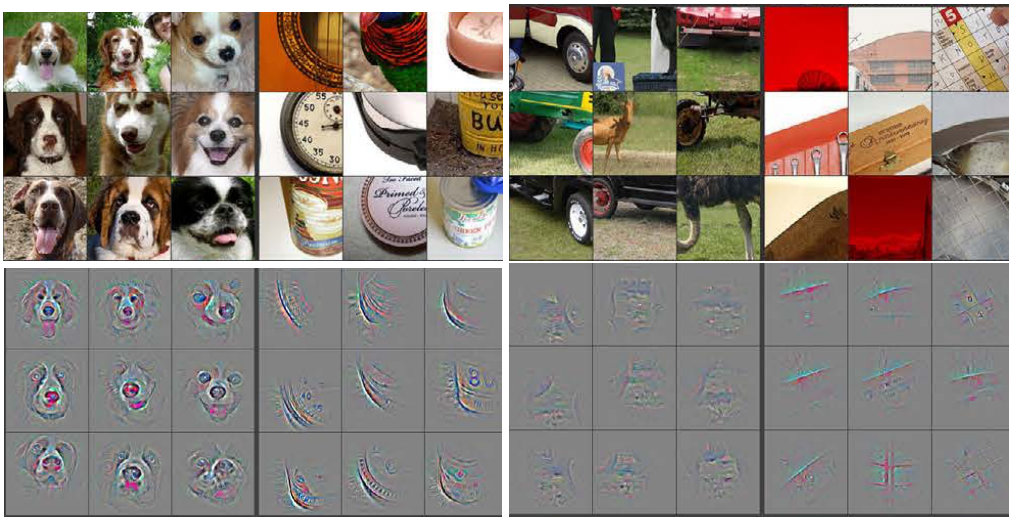
Conv layer 2 (four of the filters):



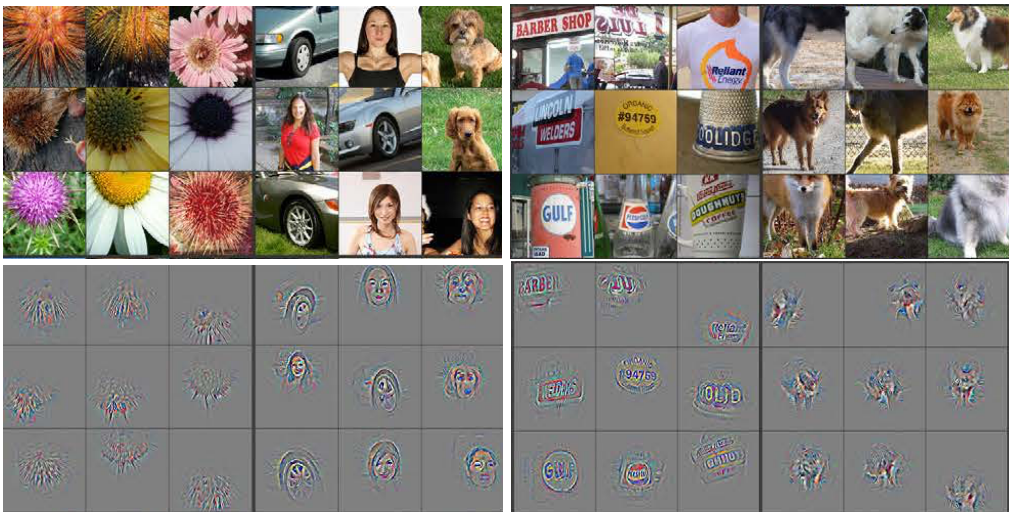
Conv layer 3:



Conv layer 4:



Conv layer 5:



The V1 Visual Cortex

[The idea to exploit local connectivity in CNNs was inspired by the human visual system, as well as by techniques used in image processing.]

[Show slides on the visual cortex, available from the CS 189 web page. Sorry, readers, there are too many images to include here. The narration is below.]

[Neurologists can stick needles into individual neurons in animal brains. After a few hours the neuron dies, but until then they can record its action potentials. In this way, biologists quickly learned how some of the neurons in the retina, called retinal ganglion cells, respond to light. They have interesting receptive fields, illustrated in the slides, which show that each ganglion cell receives excitatory stimulation from receptors in a small patch of the retina but inhibitory stimulation from other receptors around it.]

[The signals from these cells propagate to the V1 visual cortex in the occipital lobe at the back of your skull. The V1 cells proved harder to understand. David Hubel and Torsten Wiesel of the Johns Hopkins University put probes into the V1 visual cortex of cats, but they had a very hard time getting any neurons to fire there. However, a lucky accident unlocked the secret and ultimately won them the 1981 Nobel Prize in Physiology.]

[Show video HubelWiesel.mp4, taken from YouTube: https://www.youtube.com/watch?v=IOHayh06LJ4]

[The glass slide happened to be at the particular orientation the neuron was sensitive to. The neuron doesn't respond to other orientations; just that one. So they were pretty lucky to catch that.]

[The simple cells act as line detectors and/or edge detectors by taking a linear combination of inputs from retinal ganglion cells. It's fascinating, and surely not a coincidence, that humans and CNNs for vision both have edge detectors in their early layers.]

[The complex cells act as location-independent line detectors by taking inputs from many simple cells, which are location dependent. It's reminiscent of max pooling.]

[Later researchers showed that local connectivity runs through the V1 cortex by projecting certain images onto the retina and using radioactive tracers in the cortex to mark which neurons had been firing. Those images show that the neural mapping from the retina to V1 is retinatopic, i.e., locality preserving. This is a big part of the inspiration for convolutional neural networks!]

[Unfortunately, as we go deeper into the visual system, layers V2 and V3 and so on, we know less and less about what processing the visual cortex does.]