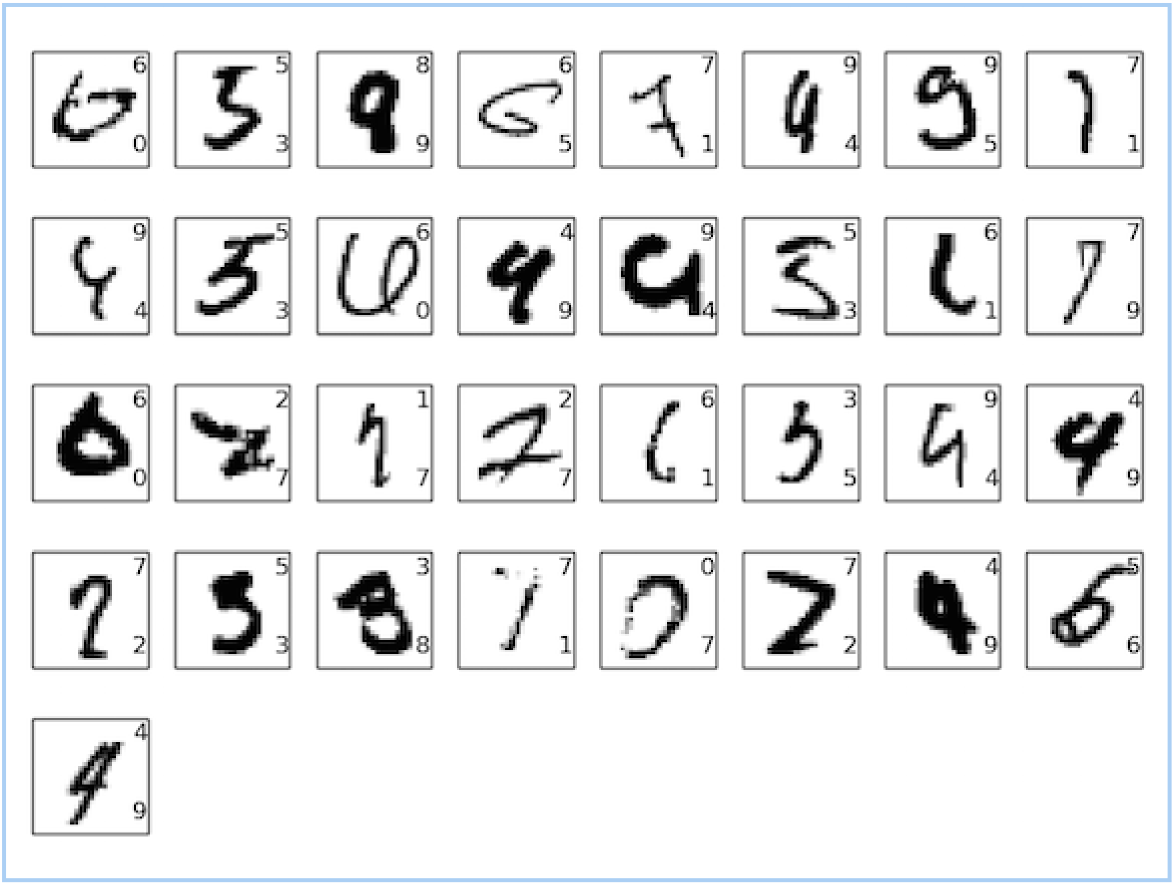
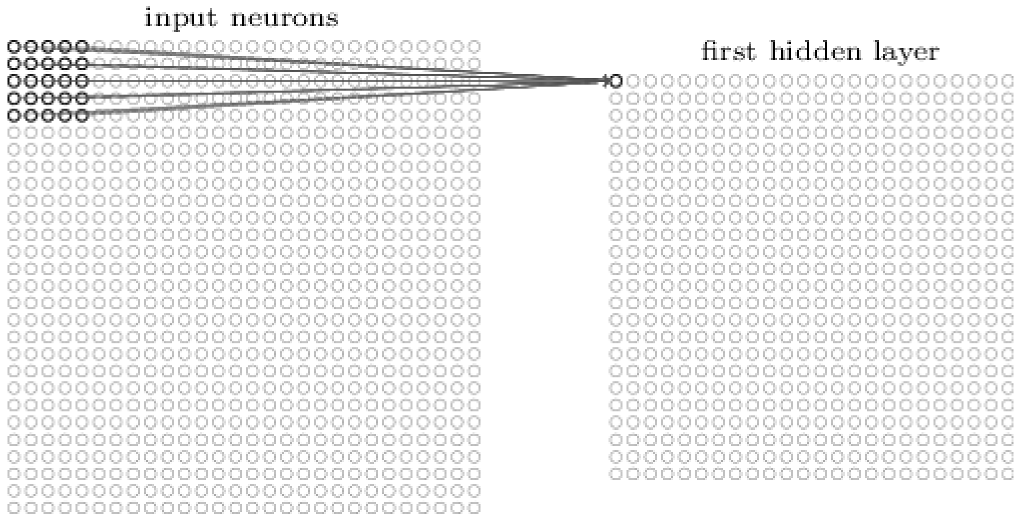
chapter 6: deep learning

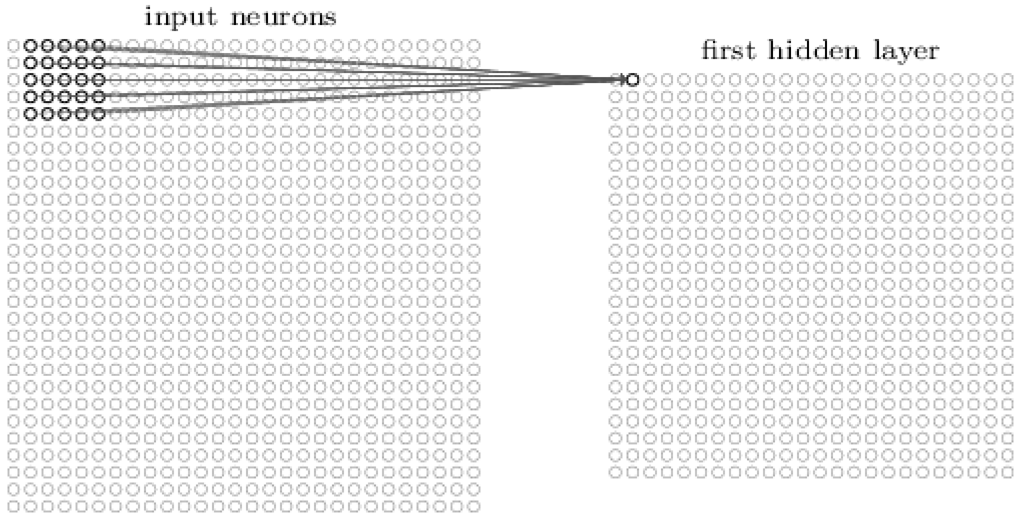
- we'll introduce ways to train deep nets, which we suspect are powerful but found hard to train
- in particular, we will look at convolutional nets, recurrent nets, LSTM units and a few more popular techniques.
- Of the 10,000 MNIST test images - images not seen during training! - the convnet system we'll build will classify 9,967 correctly.
- here are the misclassifications:



Many of these are tough even for a human to classify. Consider, for example, the third image in the top row. To me it looks more like a "9" than an "8", which is the official classification. Our network also thinks it's a "9". This kind of "error" is at the very least understandable, and perhaps even commendable.

- it's strange to use networks with fully-connected layers to classify images. The reason is that such a network architecture does not take into account the spatial structure of the images. For instance, it treats input pixels which are far apart and close together on exactly the same footing.
- what if we use an architecture which tries to take advantage of the spatial structure? In this section I describe *convolutional neural networks*. These networks use a special architecture which is particularly well-adapted to classify images. Using this architecture makes convolutional networks fast to train. This, in turn, helps us train deep, many-layer networks, which are very good at classifying images.
- Local receptive fields: convnets have local receptive fields:

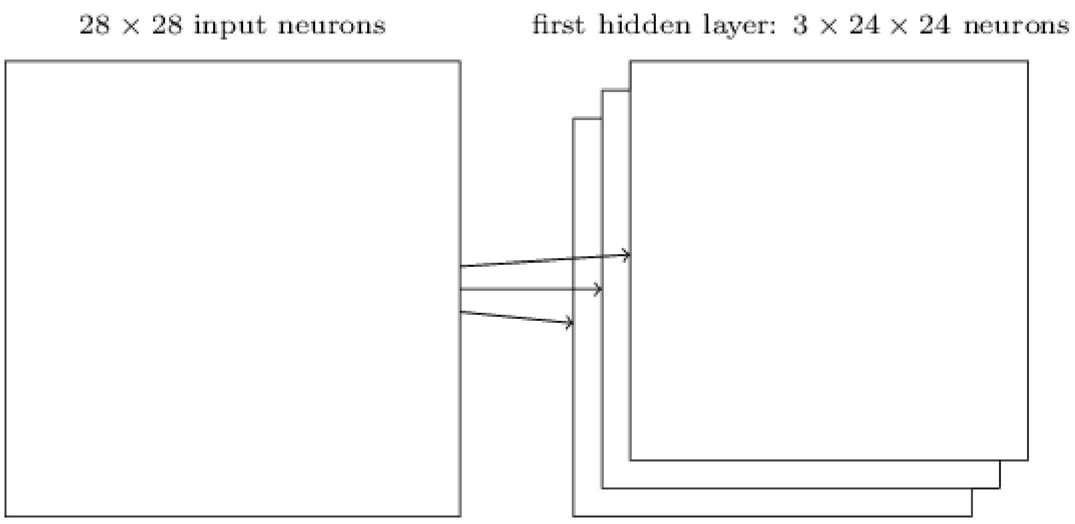




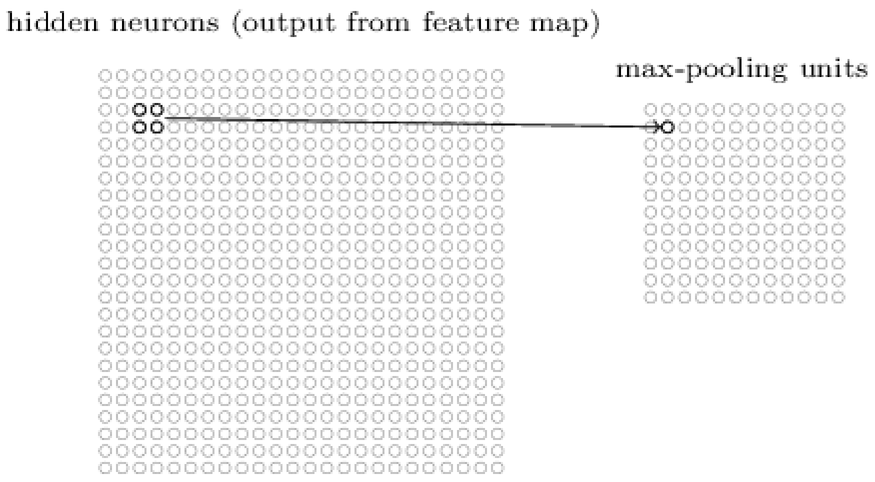
- here the stride length is 1 (the number we move over by), but this need not be 1, it can be picked using validation data just like other hyperparams.
- In general, larger local receptive fields tend to be helpful when the input images are significantly larger.

- shared weights and biases:
  - we're going to use the *same* weights and bias for each of the 24×24 hidden neurons. In other words, for the j, k^{th} hidden neuron, the output is: $\sigma(b + \sum_m \sum_l w_{l,m} a_{j+l,k+m})$ → the operation is sometimes called a convolution, hence the name.
  - “This means that all the neurons in the first hidden layer detect exactly the same feature. I haven't precisely defined the notion of a feature. Informally, think of the feature detected by a hidden neuron as the kind of input pattern that will cause the neuron to activate: it might be an edge in the image, for instance, or maybe some other type of shape., just at different locations in the input image.”
  - To put it in slightly more abstract terms, convolutional networks are well adapted to the translation invariance of images: move a picture of a cat (say) a little ways, and it's still an image of a cat.
  - For this reason, we sometimes call the map from the input layer to the hidden layer a *feature map*. We call the weights defining the feature map the *shared weights*. And we call the bias defining the feature map in this way the *shared bias*. The shared weights and bias are often said to define a *kernel* or *filter*

- To do image recognition we'll need more than one feature map. And so a complete convolutional layer consists of several different feature maps (could be a few tens of features maps)



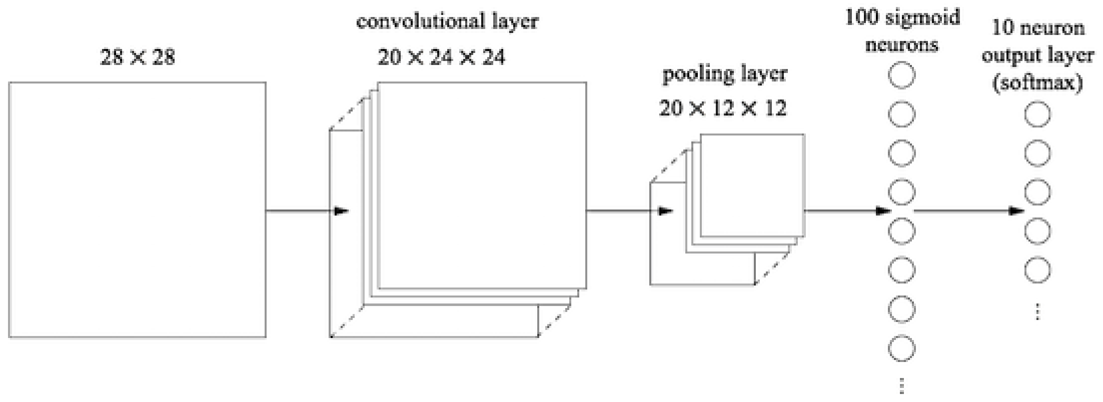
- Note that since this feature map (or maybe like 20 feature map) has fixed weights and biases, it has a lot fewer parameters than the total number of parameters in the first layer of a fully connected network. This might be the thing that makes it easier to train convnets → a lot fewer parameters.
- Pooling layers:
  - Pooling layers are usually used immediately after convolutional layers. What the pooling layers do is simplify the information in the output from the convolutional layer.
  - In detail, a pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map. For instance, each unit in the pooling layer may summarize a region of (say) 2×2 neurons in the previous layer. As a concrete example, one common procedure for pooling is known as *max-pooling*. In max-pooling, a pooling unit simply outputs the maximum activation in the 2×2 input region (like asking is the given feature found anywhere in the region of the image represented by this 2x2 neurons in the convolutional layer:



  - The intuition is that once a feature has been found, its exact location isn't as important as its rough location relative to other features.
  - As mentioned above, the convolutional layer usually involves more than a single feature map. We apply max-pooling to each feature map separately.

- *L2 pooling*. Here, instead of taking the maximum activation of a 2×2 region of neurons, we take the square root of the sum of the squares of the activations in the 2×2 region.
- If you're really trying to optimize performance, you may use validation data to compare several different approaches to pooling.

- The final layer of connections in the network is a fully-connected layer. That is, this layer connects *every* neuron from the max-pooled layer to every one of the 1010 output neurons.
- As usual, we will train our network using stochastic gradient descent and backpropagation.
- Michael trains with the above input → convolutional → max pooling → fully connected output layer network and gets 97.8% accuracy, which is just as good as without convnets that he did before. can we do better? he now adds more feature maps and adds an additional 100 neuron fully connected layer between the pooling and output layers.

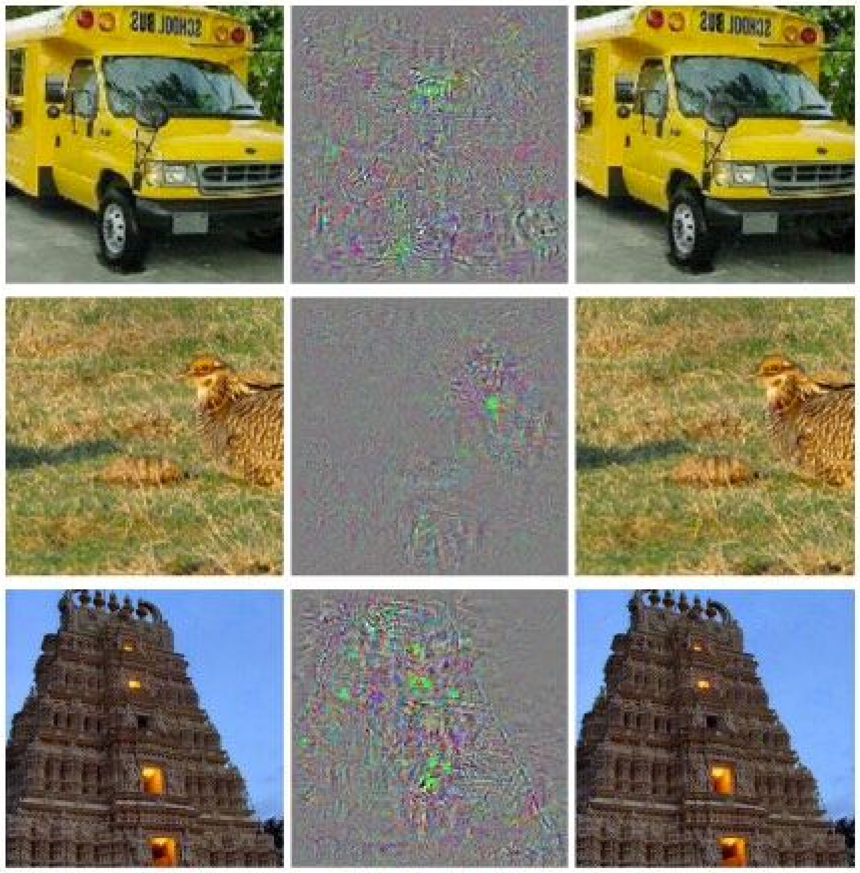


- what does it even mean to apply a second convolutional-pooling layer? In fact, you can think of the second convolutional-pooling layer as having as input "images" each of the feature maps, whose "pixels" represent the presence (or absence) of particular localized features in the original input image. So you can think of this layer as having as input a version of the original input image. That version is abstracted and condensed, but still has a lot of spatial structure, and so it makes sense to use a second convolutional-pooling layer.
- That's a satisfying point of view, but gives rise to a second question. The output from the previous layer involves separate feature maps (lets say 20 of them, each 12x12 after pooling), and so there are 20×12×12 inputs to the second convolutional-pooling layer. It's as though we've got 20 separate images input to the convolutional-pooling layer, not a single image, as was the case for the first convolutional-pooling layer. How should neurons in the second convolutional-pooling layer respond to these multiple input images? In fact, we'll allow each neuron in this layer to learn from *all* 20×5×5 input neurons in its local receptive field. More informally: the feature detectors in the second convolutional-pooling layer have access to *all* the features from the previous layer, but only within their particular local receptive field.
- Note: This issue would have arisen in the first layer if the input images were in color. In that case we'd have 3 input features for each pixel, corresponding to red, green and blue channels in the input image. So we'd allow the feature detectors to have access to all color information, but only within a given local receptive field.
- To make further improvements, “across all my experiments I found that networks based on rectified linear units consistently outperformed networks based on sigmoid activation functions.”

- What makes the rectified linear activation ($max(0, z)$) function better than the sigmoid or tanh functions? At present, we have a poor understanding of the answer to this question. Some people have said, unlike sigmoid functions they don't saturate for large z. But that's hardly an explanation.
- "And I also expect that in coming decades a powerful theory of activation functions will be developed."
- Ways to expand MNIST training data: by rotating, translating, and skewing the MNIST training images. They also developed a process of "elastic distortion", a way of emulating the random oscillations hand muscles undergo when a person is writing. By combining all these processes they substantially increased the effective size of their training data, and that's how they achieved 99.6 percent accuracy.
- Question: Are convolutional layers rotation invariant in addition to translational? or just translational?
- While using dropout to CNNs, Michael only uses dropout for the fully connected layers → not for convolutional and pooling layers. the convolutional layers have considerable inbuilt resistance to overfitting. The reason is that the shared weights mean that convolutional filters are forced to learn from across the entire image. This makes them less likely to pick up on local idiosyncracies in the training data. And so there is less need to apply other regularizers, such as dropout.
- Using an ensemble of networks:
    - An easy way to improve performance still further is to create several neural networks, and then get them to vote to determine the best classification. Suppose, for example, that we trained 5 different neural networks using the prescription above, with each achieving accuracies near to 99.6 percent. Even though the networks would all have similar accuracies, they might well make different errors, due to the different random initializations. It's plausible that taking a vote amongst our 5 networks might yield a classification better than any individual network. This sounds too good to be true, but this kind of ensembling is a common trick with both neural networks and other machine learning techniques.
- Why are we able to train? We saw in the last chapter that there are fundamental obstructions to training in deep, many-layer neural networks. But here: (1) Using convolutional layers greatly reduces the number of parameters in those layers, making the learning problem much easier; (2) Using more powerful regularization techniques (notably dropout and convolutional layers) to reduce overfitting, which is otherwise more of a problem in more complex networks; (3) Using rectified linear units instead of sigmoid neurons, to speed up training - empirically, often by a factor of 3-5; (4) Using GPUs and being willing to train for a long period of time.
- Of course, we've used other ideas, too: making use of sufficiently large data sets (to help avoid overfitting); using the right cost function (to avoid a learning slowdown); using good weight initializations (also to avoid a learning slowdown, due to neuron saturation); algorithmically expanding the training data.
- The real breakthrough in deep learning was to realize that it's practical to go beyond the shallow 1- and 2-hidden layer networks that dominated work until the mid-2000s. That really was a significant breakthrough, opening up the exploration of much more expressive models. But beyond that, the number of layers is not of primary fundamental interest. Rather, the use of deeper networks is a tool to use to help achieve other goals - like better classification accuracies.
- As a rule of thumb, the world of research is mainly focused on currently unsolvable problems that researchers think they can be on the cusp of a solution if they spend some time working on. MNIST used to get a lot of interest. Now its been solved basically. Its good for teaching, but researchers have moved on to further problems.
- "I've tried to avoid results which are fashionable as I write, but whose long-term value is unknown. In science, such results are more often than not ephemera which fade and have little lasting impact....Such a skeptic is right that some of the finer details of recent papers will gradually diminish in perceived

importance. With that said, the past few years have seen extraordinary improvements using deep nets to attack extremely difficult image recognition tasks. Imagine a historian of science writing about computer vision in the year 2100. They will identify the years 2011 to 2015 (and probably a few years beyond) as a time of huge breakthroughs, driven by deep convolutional nets. That doesn't mean deep convolutional nets will still be used in 2100, much less detailed ideas such as dropout, rectified linear units, and so on. But it does mean that an important transition is taking place, right now, in the history of ideas. "

- ImageNet is a image-label dataset with millions of data points (~15 million today). with thousands of categories → some of these images are very challenging even for humans.
- History:
  - ~2012, Google LRMD paper: Quoc Le, Marc'Aurelio Ronzato, Rajat Monga: achieved 15.8%, surpassing previous 9.3% on the ImageNet dataset.
  - ~2012 ImageNet challenge (a subset of the ImageNet dataset with 1.2 mil training images and few tens of thousands test and validation from 1000 categories) entry: Krizhevsky, Sutsekever, Hinton (KSH): "One difficulty in running the ILSVRC competition is that many ImageNet images contain multiple objects. an algorithm was considered correct if the actual ImageNet classification was among the 5 classifications the algorithm considered most likely." By this top-5 criterion, KSH's deep convolutional network achieved an accuracy of 84.7 percent, vastly better than the next-best contest entry, which achieved an accuracy of 73.8 percent. This was a deep conv net (AlexNet), trained on 2 GPUs → one GPU's memory couldn't hold the whole network
  - The KSH network has 77 layers of hidden neurons. The first 55 hidden layers are convolutional layers (some with max-pooling), while the next 22 layers are fully-connected layers. The output layer is a 1,0001,000-unit softmax layer, corresponding to the 1,0001,000 image classes.
  - The KSH network takes advantage of many techniques. Instead of using the sigmoid or tanh activation functions, KSH use rectified linear units, which sped up training significantly. KSH's network had roughly 60 million learned parameters, and was thus, even with the large training set, susceptible to overfitting. To overcome this, they expanded the training set using the random cropping strategy we discussed above. They also further addressed overfitting by using a variant of l2 regularization, and dropout. The network itself was trained using momentum-based mini-batch stochastic gradient descent.
  - In 2014, The winning team, based primarily at Google used a deep convolutional network with 22 layers of neurons. They called their network GoogLeNet, as a homage to LeNet-5. GoogLeNet achieved a top-5 accuracy of 93.33 percent, a giant improvement over the 2013 winner (Clarifai, with 88.3 percent), and the 2012 winner (KSH, with 84.7 percent).
  - While the results are genuinely exciting, there are many caveats that make it misleading to think of the systems as having better-than-human vision. The ILSVRC challenge is in many ways a rather limited problem - a crawl of the open web is not necessarily representative of images found in applications! And, of course, the top-55 criterion is quite artificial. We are still a long way from solving the problem of image recognition or, more broadly, computer vision.
  - For instance, a 2013 paper showed that deep networks may suffer from what are effectively blind spots. Consider the lines of images below. On the left is an ImageNet image classified correctly by their network. On the right is a slightly perturbed image (the perturbation is in the middle) which is classified *incorrectly* by the network. The authors found that there are such "adversarial" images for every sample image, not just a few special ones.



- This is a disturbing result. The paper used a network based on the same code as KSH's network - that is, just the type of network that is being increasingly widely used. While such neural networks compute functions which are, in principle, continuous, results like this suggest that in practice they're likely to compute functions which are very nearly discontinuous. Worse, they'll be discontinuous in ways that violate our intuition about what is reasonable behavior. That's concerning. Furthermore, it's not yet well understood what's causing the discontinuity: is it something about the loss function? The activation functions used? The architecture of the network? Something else? We don't yet know.
- For example, one recent paper (Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images) shows that given a trained network it's possible to generate images which look to a human like white noise, but which the network classifies as being in a known category with a very high degree of confidence.
- There are fundamental phenomena which we still understand poorly, such as the existence of adversarial images. When such fundamental problems are still being discovered (never mind solved), it is premature to say that we're near solving the problem of image recognition.

* Neural networks with this kind of time-varying behaviour are known as *recurrent neural networks* or *RNNs*. For instance, the behaviour of hidden neurons might not just be determined by the activations in previous hidden layers, but also by the activations at earlier times. Indeed, a neuron's activation might be determined in part by its own activation at an earlier time.
* the broad idea is that RNNs are neural networks in which there is some notion of dynamic change over time. And, not surprisingly, they're particularly useful in analyzing data or processes that change over time. Such data and processes arise naturally in problems such as speech or natural language, for example.

- RNNs have also been used in recent years to attack many other problems. They've been particularly useful in speech recognition. Approaches based on RNNs have, for example, set records for the accuracy of phoneme recognition. They've also been used to develop improved models of the language people use while speaking. Better language models help disambiguate utterances that otherwise sound alike. A good language model will, for example, tell us that "to infinity and beyond" is much more likely than "two infinity and beyond", despite the fact that the phrases sound identical. RNNs have been used to set new records for certain language benchmarks.
- The unstable gradient problem actually gets worse in RNNs, since gradients aren't just propagated backward through layers, they're propagated backward through time. If the network runs for a long time that can make the gradient extremely unstable and hard to learn from. Fortunately, it's possible to incorporate an idea known as long short-term memory units (LSTMs) into RNNs. The units were introduced by Hochreiter and Schmidhuber in 1997 with the explicit purpose of helping address the unstable gradient problem.
- In a feedforward network, we specify the input activations, and they determine the activations of the feature neurons later in the network. A generative model like a DBN (deep belief nets) can be used in a similar way, but it's also possible to specify the values of some of the feature neurons and then "run the network backward", generating values for the input activations. More concretely, a DBN trained on images of handwritten digits can (potentially, and with some care) also be used to generate images that look like handwritten digits. In Geoffrey Hinton's memorable phrase, to recognize shapes, first learn to generate images.
- A second reason DBNs are interesting is that they can do unsupervised and semi-supervised learning. For instance, when trained with image data, DBNs can learn useful features for understanding other images, even if the training images are unlabelled. And the ability to do unsupervised learning is extremely interesting both for fundamental scientific reasons, and - if it can be made to work well enough - for practical applications.
- Why have DBNs gotten so much less interest and CNNs and RNNs so much attention? “The marketplace of ideas often functions in a winner-take-all fashion, with nearly all attention going to the current fashion-of-the-moment in any given area. It can become extremely difficult for people to work on momentarily unfashionable ideas, even when those ideas are obviously of real long-term interest. My personal opinion is that DBNs and other generative models likely deserve more attention than they are currently receiving.”
- “Let me finish this section by mentioning a particularly fun paper. It combines deep convolutional networks with a technique known as reinforcement learning in order to learn to play video games well (see also this followup). The idea is to use the convolutional network to simplify the pixel data from the game screen, turning it into a simpler set of features, which can be used to decide which action to take: "go left", "go down", "fire", and so on. What is particularly interesting is that a single network learned to play seven different classic video games pretty well, outperforming human experts on three of the games. Now, this all sounds like a stunt, and there's no doubt the paper was well marketed, with the title "Playing Atari with reinforcement learning". But looking past the surface gloss, consider that this system is taking raw pixel data - it doesn't even know the game rules! - and from that data learning to do high-quality decision-making in several very different and very adversarial environments, each with its own complex set of rules. That's pretty neat.”
- we understand neural networks so poorly. Why is it that neural networks can generalize so well? How is it that they avoid overfitting as well as they do, given the very large number of parameters they learn? Why is it that stochastic gradient descent works as well as it does? How well will neural networks perform as data sets are scaled? For instance, if ImageNet was expanded by a factor of 10, would neural

networks' performance improve more or less than other machine learning techniques? These are all simple, fundamental questions. And, at present, we understand the answers to these questions very poorly.

- Conway's law:
    - "Any organization that designs a system... will inevitably produce a design whose structure is a copy of the organization's communication structure."
- Is there a version of Conway's law that applies to problems which are more science than engineering?
    - To gain insight into this question, consider the history of medicine. In the early days, medicine was the domain of practitioners like Galen and Hippocrates, who studied the entire body. But as our knowledge grew, people were forced to specialize. We discovered many deep new ideas: think of things like the germ theory of disease, for instance, or the understanding of how antibodies work, or the understanding that the heart, lungs, veins and arteries form a complete cardiovascular system. Such deep insights formed the basis for subfields such as epidemiology, immunology, and the cluster of inter-linked fields around the cardiovascular system. And so the structure of our knowledge has shaped the social structure of medicine. This is particularly striking in the case of immunology: realizing the immune system exists and is a system worthy of study is an extremely non-trivial insight. So we have an entire field of medicine - with specialists, conferences, even prizes, and so on - organized around something which is not just invisible, it's arguably not a distinct thing at all.
- *And so the structure of our knowledge shapes the social organization of science. But that social shape in turn constrains and helps determine what we can discover.* This is the scientific analogue of Conway's law.
-