Lecture 10: Image Classification

Roger Grosse

1 Introduction

Vision feels so easy, since we do it all day long without thinking about it. But think about just how hard the problem is, and how amazing it is that we can see. A grayscale image is just a two dimensional array of intensity values, and somehow we can recover from that a three-dimensional understanding of a scene, including the types of objects and their locations, which particular people are present, what materials things are made of, and so on. In order to see, we have to deal with all sorts of "nuisance" factors, such as change in pose or lighting. It's amazing that the human visual system does this all so seamlessly that we don't even have to think about it.

Even talking about "images" masks a lot of complexity; the human retina has to deal with 11 orders of magnitude in intensity variation and uses fancy optics that let us recover detailed information in the fovea of our visual field, for a variety of wavelengths of light.

There is a large and active field of research called computer vision which tries to get machines to see. The field has made rapid progress in the past decade, largely because of increasing sophistication of machine learning techniques and the availability of large image collections. They've formulated hundreds of interesting visual "tasks" which encapsulate some of the hidden complexity we deal with on a daily basis, such as estimating the calorie content of a plate of food or predicting whether a structure is likely to fall down. But there's one task which has received an especially large amount of attention for the past 30 years and which has driven a lot of the progress in the field: **object recognition**, the task of classifying an image into a set of object categories.

Object recognition is also a useful example for looking at how conv nets have changed over the years, since they were a state-of-the-art tool in the early days, and in the last five years, they have re-emerged as the state-of-the-art tool for object recognition as well as dozens of other vision tasks. When conv nets took over the field of computer vision, object recognition was the first domino to fall. Computers have gotten dramatically faster during this time, and the networks have gotten correspondingly bigger and more powerful, but they're still based on more or less the same design principles. This lecture will talk about some of those design principles.

2 Object recognition datasets

Recall that object recognition is a kind of supervised learning problem, which means there's a particular behavior we would like our system to achieve (labeling an image with the correct category), and we need to provide the system with labeled examples of the correct behavior. This means we need to come up with a **dataset**, a set of images with their corresponding labels. This raises questions such as: how do we choose the set of categories? What sorts of images do we allow, how many do we need, and where do we get them? Do we **preprocess** them in some way to make life easier for the algorithm? We'll look at just a few examples of particularly influential datasets, but we'll ignore dozens more, which each have their virtues and drawbacks.

2.1 USPS and MNIST

Before machine learning algorithms were good enough to recognize objects in images, researchers' attention focused on a simpler image classification problem: **handwritten digit recognition**. In the 1980s, the US Postal Service was interested in automatically reading zip codes on envelopes. This task is a bit harder than handwritten digit recognition, since one also has to identify the locations and orientations of the individual digits, but clearly digit recognition would be a useful step towards solving the problem. They collected a dataset of images of handwritten digits (now called the **USPS Dataset**) by hand-segmenting individual digits from handwritten zip codes. To make things easier for the algorithm, the digits were **normalized** to be a consistent size and orientation. Despite this normalization, the dataset still included a lot of sources of variability: digits were written in a variety of writing styles and using different kinds of writing instruments. Many of the digits are ambiguous, even to humans.

Classifying USPS digits became the first practical use of conv nets: in 1989, a group of researchers at Bell Labs introduced a conv net architecture, which involved several convolution and subsampling layers, followed by a fully connected layer. This network was able to classify the digits with 91.9% accuracy.

Almost a decade later, researchers created a slightly larger handwritten digit dataset. They made some modifications to a dataset produced by the National Institute of Standards and Technology, so the dataset was called Modified NIST, or **MNIST**. Similar to the USPS Dataset, MNIST images were normalized by centering the digits within the image and normalizing them to a standard size. The main difference is that the dataset is larger: there were 70,000 examples, of which 60,000 are used for training and 10,000 are used for testing. Yann LeCun and colleagues introduced a larger conv net architecture called **LeNet** which was able to classify images with 98.9% accuracy, and used this network in the context of a larger system for automatically reading the numbers on checks. (Because LeNet was trained on segmented and normalized digit images, this system had to solve the problems of automatic segmentation and normalization, among other things.) This was the first automatic check reading system that was accurate enough to be practically useful. This was one of the big success stories of AI in the 1990s — and interestingly, it happened during the "neural net winter", showing that good ideas can still work even when they fall out of fashion.[1]

LeCun is now one of the leading researchers in the field, and directs Facebook AI Research.

Apart from its initial practical uses, MNIST has served as one of the most widely used machine learning benchmarks for two decades. Even though the test errors have long been low enough to be practically meaningless, MNIST has driven a lot of progress in neural net research. As recently

[1] LeCun et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

as 2012, Geoff Hinton and collaborators introduced dropout (a regularization method discussed in Lecture 9) on MNIST; this turned out to work well on a lot of other problems, and has become one of the standard tools in the neural net toolbox.

2.2 Caltech101 and the perils of dataset design

In 2003, researchers at Caltech released the first major object recognition dataset which was used to train and benchmark object recognition algorithms. Since their dataset included 101 object categories, they called it **Caltech101**.[2] Here's how they approached some of the key questions of dataset design.

- *Which object categories to consider?* They chose a set of 101 object categories by opening a dictionary to random pages and choosing from the nouns which were associated with images.
- *Where do the images come from?* They used Google Image Search to find candidate images, and then filtered by hand which images actually represented the object category.
- *How many images?* They didn't target a particular number of objects per category, but just collected as many as possible. The numbers of objects per category were very unbalanced as a result, but in practice, when the dataset is used for benchmarking, most systems are trained with a fixed number of images per category (which is usually between 1 and 20).
- *How to normalize the images?* They normalized the images in a variety of ways to make things simpler for the learning algorithms. Images were scaled to be about 300 pixels wide. In order to reduce variability in pose, they flipped some of the images so that a given object was always facing the same direction. More controversially, images of certain object categories were rotated because the authors' proposed method had trouble dealing with vertically oriented objects.

For about 5 years, Caltech101 was widely used as a benchmark dataset for object recognition, and academic papers showed rapid improvements in classification accuracy. Unfortunately, the dataset had a number of idiosyncrasies, known as **dataset biases**. E.g., for some reason, objects always appeared at a consistent location within an image, with the result that if one averages the raw pixel values, the average image still resembles the object category.[3] Also, as mentioned above, images of certain categories were rotated, leading to distinctive rotation artifacts.

Dataset bias results in a kind of overfitting which is different from what we've talked about so far. In our lecture on generalization, we observed that a training set might happen to have certain accidental regularities which don't occur in the test set; algorithms can overfit if they exploit these regularities. If the training and test images are drawn from the same

[2] `https://www.vision.caltech.edu/Image_Datasets/Caltech101/`

[3] `https://www.vision.caltech.edu/Image_Datasets/Caltech101/averages100objects.jpg`

distribution, this kind of overfitting can be eliminated if one builds a large enough training set. Dataset bias is different — it consists of systematic biases in a dataset resulting from the way in which the data was collected. These regularities occur in both the training and the test sets, so algorithms which exploit them appear to generalize well on the test set. However, if those regularities aren't present in the situation where one actually wants to use the classifier (e.g. a robot trying to identify objects), the system will perform very poorly in practice. (If an image classifier only recognizes minarets by exploiting rotation artifacts, it's unlikely to perform very well in the real world.)

If dataset bias is strong enough, it encourages the troubling practice of **dataset hacking**, whereby researchers engineer their learning algorithms to be able to exploit the dataset biases in order to make their results seem more impressive. In the case of Caltech101, the dataset biases were strong enough that dataset hacking became essentially the only way to compete. After about 5 years, Caltech101 basically stopped being used for computer vision research. Dozens of other object recognition datasets were created, all using different methodology intended to attenuate dataset bias; see this paper[4] for an interesting discussion. Despite a lot of clever attempts, creating a fully realistic dataset is an elusive goal, and dataset bias will probably always exist to some degree.

An interesting tidbit: both human researchers and learning algorithms are able to determine with surprisingly high accuracy which object recognition dataset a given image was drawn from.

2.3 ImageNet

In 2009, taking into account lessons learned from Caltech101 and other computer vision datasets, researchers built ImageNet, a massive object recognition database consisting of millions of full-resolution images and thousands of object categories. Based on this dataset, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) became one of the most important computer vision benchmarks. Here's how they approached the same questions:

- *Which object categories to consider?* ImageNet was meant to be very comprehensive. The categories were taken from WordNet, a lexical database for English constructed by cognitive scientists at Princeton. WordNet consists of a hierarchy of "synsets", or sets of synonyms which all denote the same concept. ImageNet was intended to include as many synsets as possible; as of 2010, it included almost 22,000 synsets, out of 80,000 noun synsets in WordNet. The categories are very specific, including hundreds of different types of dogs. Out of these categories, 1000 were chosen for the ILSVRC.

- *How many images?* The aim was to come up with hundreds of labeled images for each synset. The ILSVRC categories all have hundreds of associated training examples, for a total of 1.2 million images.

- *Where do the images come from?* Similarly to Caltech101, candidate images were taken from the results of various image search engines,

[4]A. Torralba and A. Efros. An unbiased look at dataset bias. *Computer Vision and Pattern Recognition (CVPR)*, 2011.

and then humans manually labeled them. Labeling millions of images is obviously challenging, so they paid Amazon Mechanical Turk workers to annotate images. Since some of the categories were highly specific or unusual, they had to provide the annotators with additional information (e.g. Wikipedia articles) to help them, and carefully validated the process by measuring inter-annotator agreement.

- *How are the images normalized?* In contrast to Caltech101, the images in the dataset itself are not normalized. (However, the object recognition systems themselves might perform some sort of preprocessing.)

Because the object categories are so diverse and fine-grained, and images can contain multiple objects, there might not be a unique right answer for every image. Therefore, one normally reports **top-5 accuracy**, whereby the algorithm is allowed to make 5 different predictions for each image, and it gets it right if any of the 5 predictions are the correct category.

ImageNet is an extremely challenging dataset to work with because of its scale and the diversity of object categories. The first algorithms to be applied were not neural nets, but in 2012, researchers in Toronto entered this competition using a neural net called **AlexNet** (in honor of its lead creator, Alex Krizhevsky). It achieved top-5 error of 28.5%, which was substantially better than the competitors. This result created a big splash, leading computer vision researchers to switch to using neural nets and prompting some of the world's largest software companies to start up research labs focused on deep learning. Since AlexNet, error rates on ImageNet have fallen dramatically, hitting 4.5% error in 2015 (the last year the competition was run), and all of the leading approaches have been based on conv nets. This even beat human performance, which was measured at 5.1% error (although this can vary significantly depending how one measures).

3 LeNet

Let's look at a particular conv net architecture: LeNet, which was used to classify MNIST digits in 1998. The inputs are grayscale images of size 32×32. One detail I've skipped over so far is the sizes of the outputs of convolution layers. LeNet uses **valid convolutions**, where the values are computed for only those locations whose filters lie entirely within the input. Therefore, if the input is 32×32 and the filters are 5×5, the outputs will be 28×28. (The main alternative is **same convolution**, where the output is the same size as the input, and the input image is padded with zeros in all directions.) The LeNet architecture is shown in Figure 1 and summarized in Table 1.

- *Convolution layer C1.* This layer has 6 feature maps and filters of size 5×5. It has $28 \times 28 \times 6 = 4704$ units, $28 \times 28 \times 5 \times 5 \times 6 = 117,600$ connections, and $5 \times 5 \times 6 = 150$ weights and 6 biases, for a total of 156 trainable parameters.

- *Subsampling layer S2.* In LeNet, the "subsampling layers" are essentially pooling layers, where the pooling function is the mean (rather

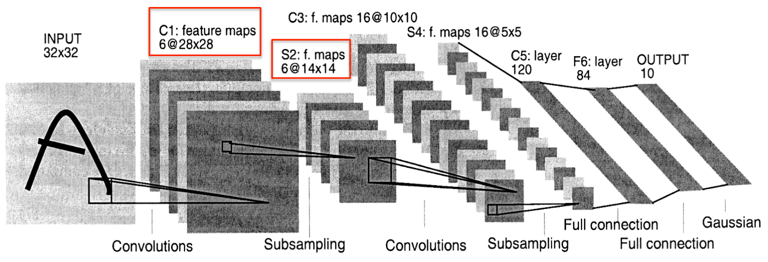

Figure 1: The LeNet architecture from 1998.

than max). They use a stride of 2, so the image size is shrunk by a factor of 2 along each dimension.

- *Convolution layer C3.* This layer has 16 feature maps of size 10×10 and filters of size 5×5. Therefore, it has $10 \times 10 \times 16 = 1600$ units. If all the feature maps were connected to all the feature maps, this layer would have $10 \times 10 \times 5 \times 5 \times 6 \times 16 = 240,000$ connections and $5 \times 5 \times 6 \times 16 = 2400$ weights.[5]

- *Subsampling layer S4.* This is another pooling layer with a stride of 2, so it reduces each dimension by another factor of 2, to 5×5.

- *Fully connected layer F5.* This layer has 120 units with a full set of connections to layer S4. Since S4 has $5 \times 5 \times 16 = 400$ units, this layer has $400 \times 120 = 48,000$ connections, and hence the same number of weights.

- *Fully connected layer F6.* This layer has 84 units, fully connected to F5. Therefore, it has $84 \times 120 = 10,080$ connections and the same number of weights.

- *Output layer.* The original network used something called radial basis functions, but for simplicity we'll pretend it's just a linear function, followed by a softmax over 10 categories. It has $84 \times 10 = 840$ connections and weights.

These calculations are all summarized in Table 1. After sitting through all this tedium, we can draw a number of useful conclusions:

- Most of the units are in the first convolution layer.
- Most of the connections are in the second convolution layer.
- Most of the weights are in the fully connected layers.

These observations correspond to various resource limitations when designing a network architecture. In particular, if we want to make the network as big as possible, here are some of the limitations we run into:

[5]Since this would have been a lot of connections by the standards of 1998, they skimped on connections by connecting only a subset of the feature maps. This brought the number of connections down to 156,000.

Layer	Type	# units	# connections	# weights
C1	convolution	4704	117,600	150
S2	subsampling	1176	4704	0
C3	convolution	1600	240,000	2400
S4	subsampling	400	1600	0
F5	fully connected	120	48,000	48,000
F6	fully connected	84	10,080	10,080
output	fully connected	10	840	840

Table 1: LeNet architecture, with the sizes of layers.

- Running the network to compute predictions (equivalently, the forward pass of backprop) requires approximately one add-multiply operation per connection in the network. As observed in a previous lecture, the backwards pass is about as expensive as two forward passes, so the total computational cost of backprop is proportional to the number of connections. This means the convolution layers are generally the most expensive part of the network in terms of running time.
- Memory is another scarce resource. It's worth distinguishing two situations: **training time**, where we train the network using backprop, and **test time**, the somewhat misleading name for the setting where we use an already-trained network.
  - Backprop requires storing all of the activations in memory.[6] Since the number of activations is the number of units times the mini-batch size, the number of units determines the memory footprint of the activations at training time. The activations don't need to be stored at test time.
  - The weights also need to be stored in memory, both at training time and test time.
- The weights constitute the vast majority of trainable parameters of the model (the number of biases generally being far smaller), so if you're worried about overfitting, you could consider cutting down the number of weights.

LeNet was carefully designed to push the limits of all of these resource constraints using the computing power of 1998. As we'll see, conv nets have grown substantially larger in order to exploit modern computing resources.

Try increasing the sizes of various layers and checking that you're substantially increasing the usage of one or more of these resources.

4 Modern conv nets

As mentioned above, AlexNet was the conv net architecture which started a revolution in computer vision by smashing the ILSVRC benchmark. This

[6]This isn't quite true, actually. There are tricks for storing activations for only a subset of the layers, and recomputing the rest of the activations as needed. Indeed, frameworks like TensorFlow implement this behind the scenes. However, a larger of units generally implies a higher memory footprint.

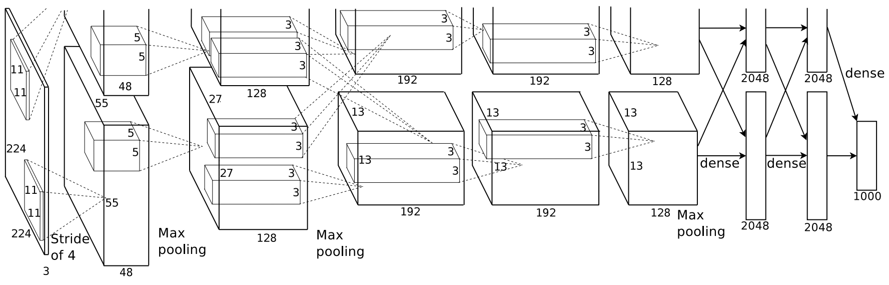

Figure 2: The AlexNet architecture from 2012.

	LeNet (1989)	**LeNet (1998)**	**AlexNet (2012)**
classification task	digits	digits	objects
dataset	USPS	MNIST	ImageNet
# categories	10	10	1,000
image size	16×16	28×28	$256 \times 256 \times 3$
training examples	7,291	60,000	1.2 million
units	1,256	8,084	658,000
parameters	9,760	60,000	60 million
connections	65,000	344,000	652 million
total operations	11 billion	412 billion	200 quadrillion (est.)

Table 2: Comparison of conv net classification architectures.

architecture is shown in Figure 2. Like LeNet, it consists mostly of convolution, pooling, and fully connected layers. It additionally has some "response normalization" layers, which I won't talk about because they're not believed to make a big difference, and have mostly stopped being used.

By most measures, AlexNet is 100 to 1000 times bigger than LeNet, as shown in Table 2. But qualitatively, the structure is very similar to LeNet: it consists of alternating convolution and pooling layers, followed by fully connected layers. Furthermore, like LeNet, most of the units and connections are in the convolution layers, and most of the weights are in the fully connected layers.

Computers have improved a lot since LeNet, but the hardware advance that suddenly made it practical to train large neural nets was **graphics processing units (GPUs)**. GPUs are a kind of processor geared towards highly parallel processing involving relatively simple operations. One of the things they especially excel at is matrix multiplication. Since most of the running time for a neural net consists of matrix multiplication (even convolutions are implemented as matrix products beneath the hood), GPUs gave roughly a 30-fold speedup in practice for training neural nets.

AlexNet set the agenda for object recognition research ever since. In 2013, the ILSVRC winner was based on tweaks to AlexNet. In 2014, the second place entry was **VGGNet**, another conv net based on more or less similar principles.

The winning entry for 2014, **GoogLeNet**, or **Inception**, deserves mention. As the name suggests, it was designed by researchers at Google. The architecture is shown in Figure 3. Clearly things have gotten more complicated since the days of LeNet. But the main point of interest is that they went out of their way to reduce the number of trainable parameters (weights) from AlexNet's 60 million, to about 2 million. Why? Partly it was to reduce overfitting — amazingly, it's possible to overfit a million images if you have a big enough network like AlexNet.

The other reason has to do with saving memory at "test time", i.e. when the network is being used. Traditionally, networks would be both trained and run on a single PC, so there wasn't much reason to draw a distinction between training and test time. But at Google, the training could be distributed over lots of machines in a datacenter. (The activations and parameters could even be divided up between multiple machines, increasing the amount of available memory at training time.) But the network was also supposed to be runnable on an Android cell phone, so that images wouldn't have to be sent to Google's servers for classification. On a cell phone, it would have been extravagant to spend 240MB to store AlexNet's 60 million parameters, so it was really important to cut down on parameters to make it fit in memory.

They achieved this in two ways. First, they eliminated the fully connected layers, which we already saw contain most of the parameters in LeNet and AlexNet. GoogLeNet is convolutions all the way. It also avoids having large convolutions by breaking them down into a sequence of convolutions involving smaller filters. (Two 3×3 filters have fewer parameters than a 5×5 filter, even though they cover a similar radius of the image.) They call this layer-within-a-layer architecture "Inception", after the movie about dreams-within-dreams.

This is analogous to how linear bottleneck layers can reduce the number of parameters.

Performance on ImageNet improved asonishingly fast during the years the competition was run. Here are the figures:

We'll put off the last item, deep residual nets (ResNets), until a later lecture since they depend on some ideas that we won't cover until we talk about RNNs.

Year	**Model**	**Top-5 error**
2010	Hand-designed descriptors + SVM	28.2%
2011	Compressed Fisher Vectors + SVM	25.8%
2012	AlexNet	16.4%
2013	a variant of AlexNet	11.7%
2014	GoogLeNet	6.6%
2015	deep residual nets	4.5%

It's really unusual for error rates to drop by a factor of 6 over a period of 5 years, especially on a task like object recognition that hundreds of researchers had already worked hard on and where performance had seemed to plateau.

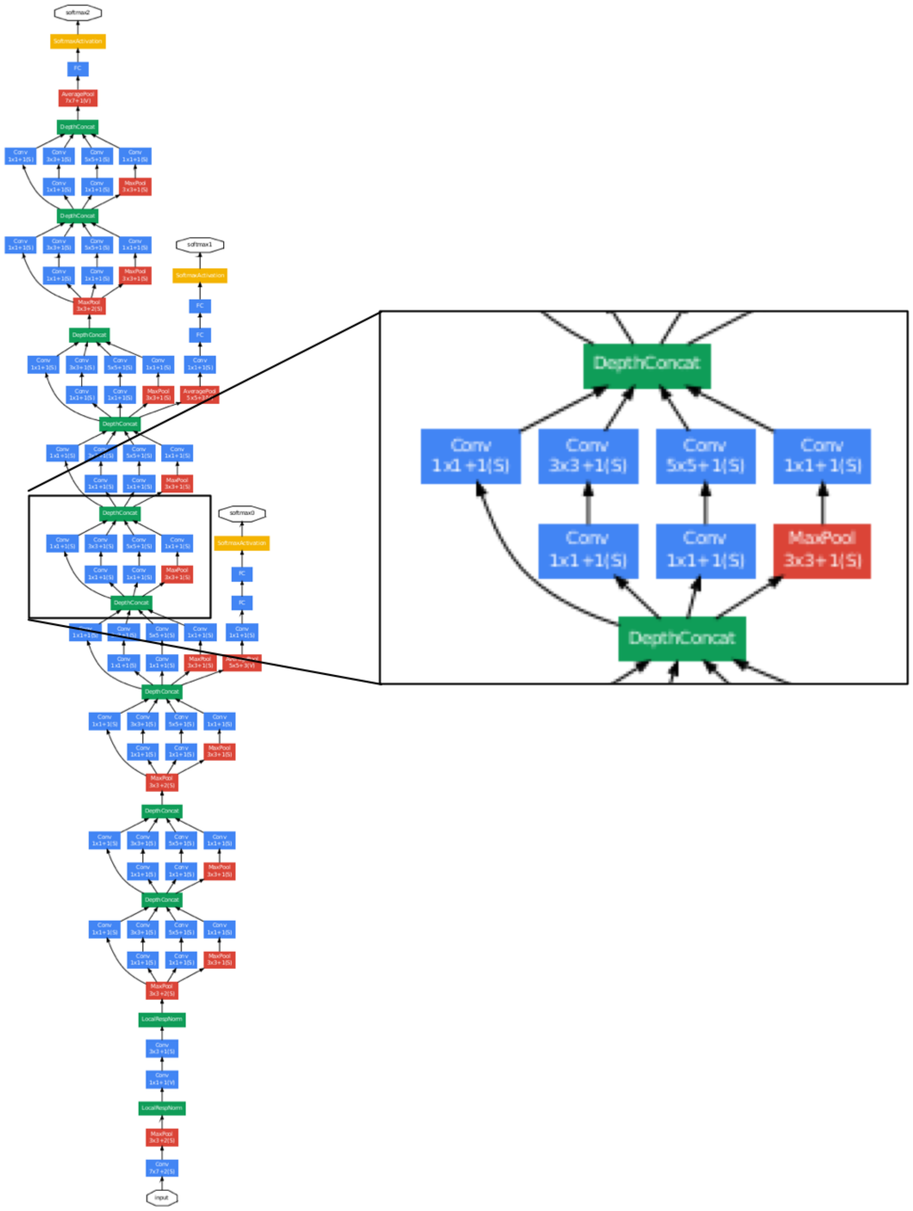

Figure 3: The Inception architecture from 2014.