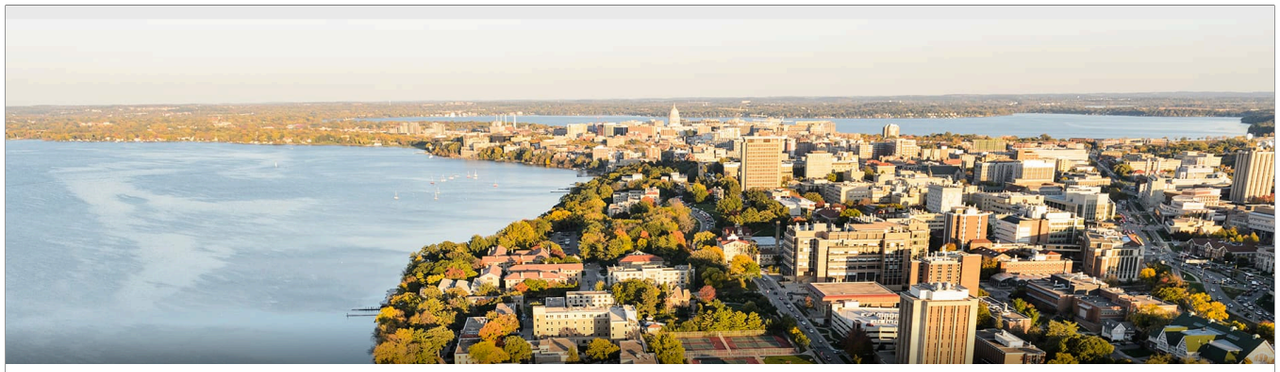


CS540 Introduction to Artificial Intelligence

Deep Learning II: Convolutional Neural Networks

Yingyu Liang
University of Wisconsin-Madison

Nov 4, 2021

Slides created by Sharon Li [modified by Yingyu Liang]

Outline

- Brief review of convolutional computations
- Convolutional Neural Networks
  - LeNet (first conv nets)
  - AlexNet
  - ResNet

Review: 2-D Convolution

Input

0	1	2
3	4	5
6	7	8

*

Kernel

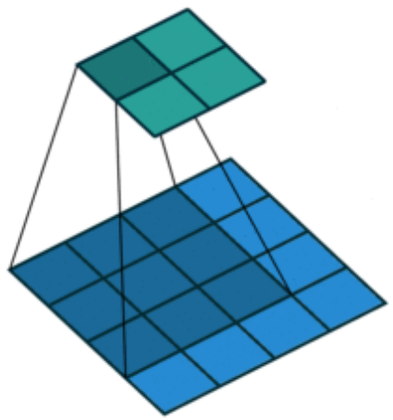
0	1
2	3

=

Output

19	25
37	43

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

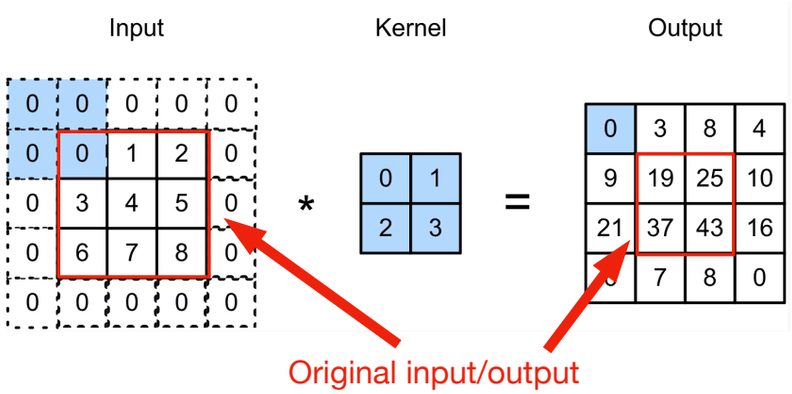


(vdumoulin@ Github)

Recall convolution: put a sliding window of the same size as the kernel, multiply the corresponding entries in the window and the kernel, sum up to get the output entry; move the sliding window to different locations on the input, and compute the corresponding output entry.

Padding

Padding adds rows/columns around input



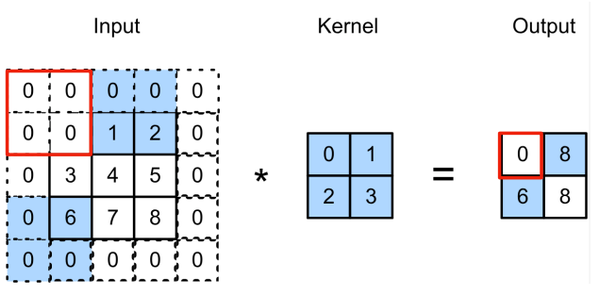
$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

We can pad rows/columns of 0’s around the original input, and then do convolution on the padded input. This is to increase the size of the output.

Stride

- Stride is the #rows/#columns per slide

Strides of 3 and 2 for height and width



$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$

We can move >1 rows/columns each time we move the sliding window.

Output shape

Kernel/filter size

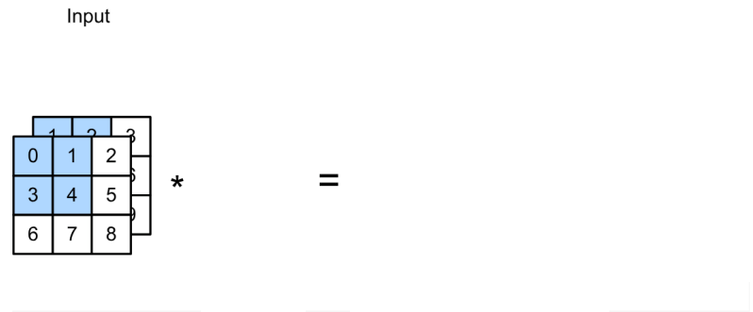
$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

Input size Pad Stride

The output shape has a closed form solution. The derivation can be found in the last lecture.

Review: Multiple Input Channels

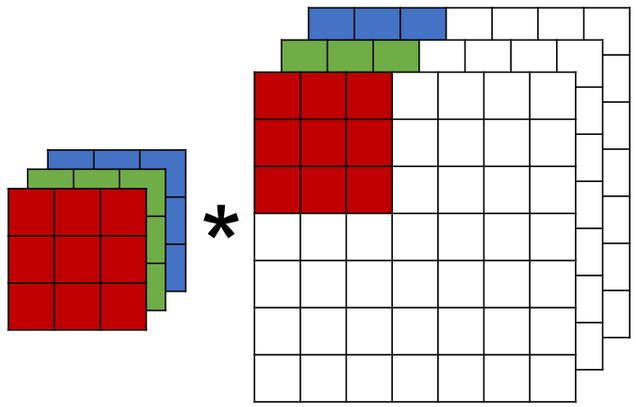
- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels



When the input has multiple channels, the kernel should also have the same number of channels (can also view each channel as one kernel matrix). Do convolution on each channel and then sum up the results.

Review: Multiple Input Channels

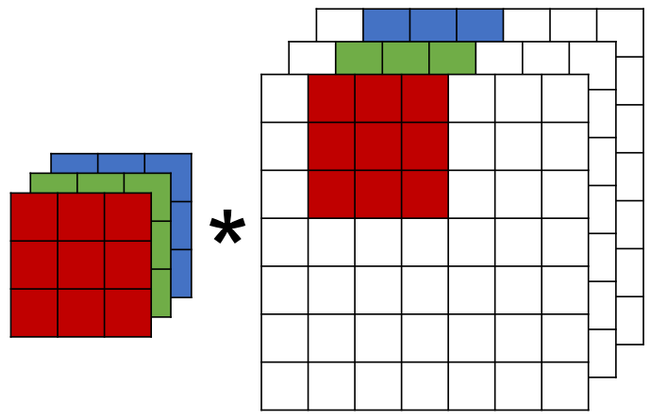
- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels



We should move the sliding window simultaneously across all the input channels as illustrated in the figures.

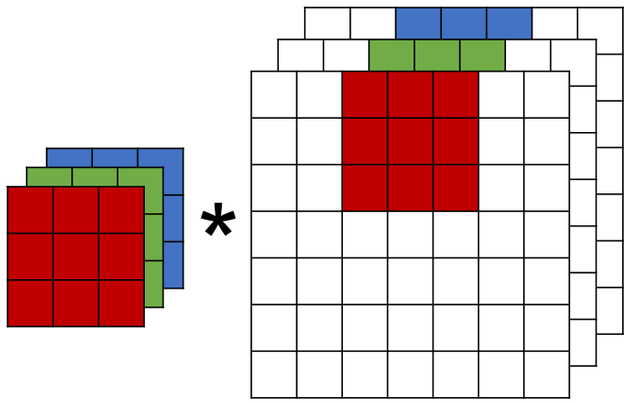
Review: Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels



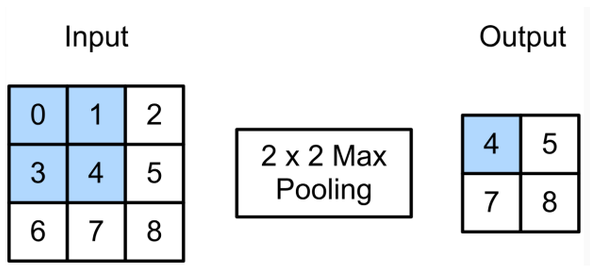
Review: Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels

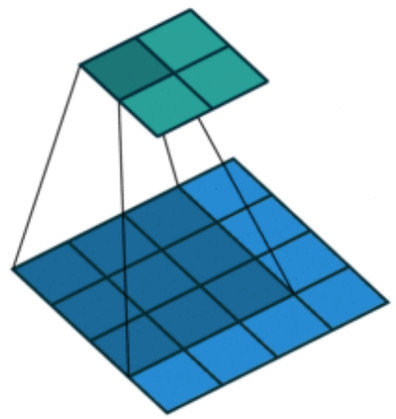


Review: 2-D Max Pooling

- Returns the maximal value in the sliding window



$$\max(0,1,3,4) = 4$$



Pooling: also have a sliding window, output the max number in the window; move the sliding window around and compute the corresponding output entries.

Also can have padding/stride/multiple channels.

Two key differences from convolution:
1. No parameters
2. Apply pooling on each input channel to get an output channel. No sum over the input channels.

Q1: Consider a convolution layer with 16 filters. Each filter has a size of 11x11x3, a stride of 2x2. Given an input image of size 22x22x3, if we don’t allow a filter to fall outside of the input (no padding), what is the output size?

A. 11x11x16

B. 6x6x16

C. 7x7x16

D. 5x5x16

Q1: Consider a convolution layer with 16 filters. Each filter has a size of 11x11x3, a stride of 2x2. Given an input image of size 22x22x3, if we don't allow a filter to fall outside of the input (no padding), what is the output size?

A. 11x11x16

B. 6x6x16

C. 7x7x16

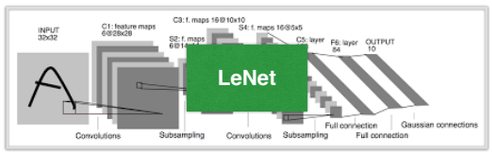
D. 5x5x16

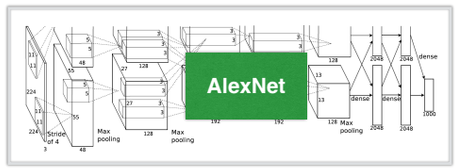
$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

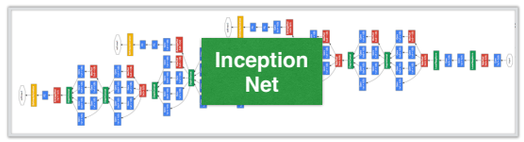
Since we have 16 filters, the output has 16 channels. Each channel's shape can be computed by the given expression.

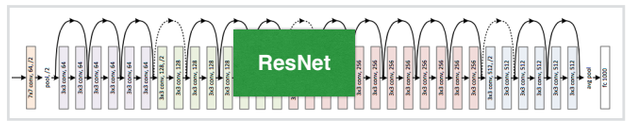
Convolutional Neural Networks

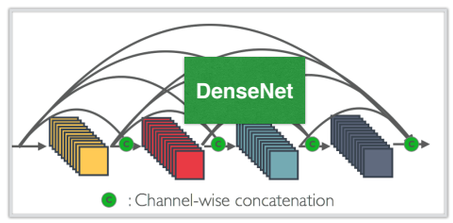
Evolution of neural net architectures





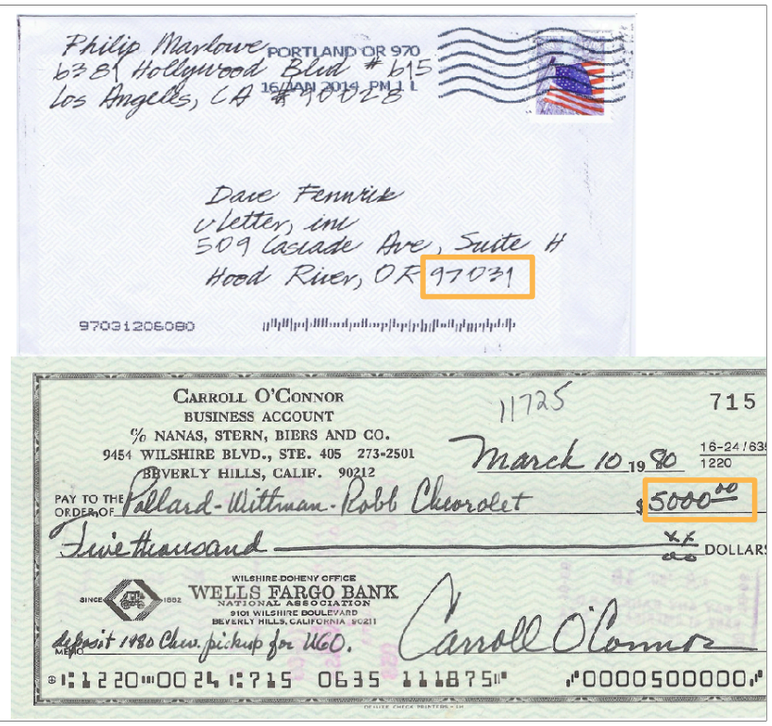






A brief history of convolutional neural networks (CNNs):
LeNet: the first convolutional network
AlexNet: convince the community that deep networks can achieve superior performance; lead to paradigm shift in computer vision, machine learning and AI
Later models: Inception Net, ResNet, DenseNet, and many others. ResNet proposes the idea of residual connections, which is a simple yet powerful idea that has been a “backbone” idea for many modern network design.

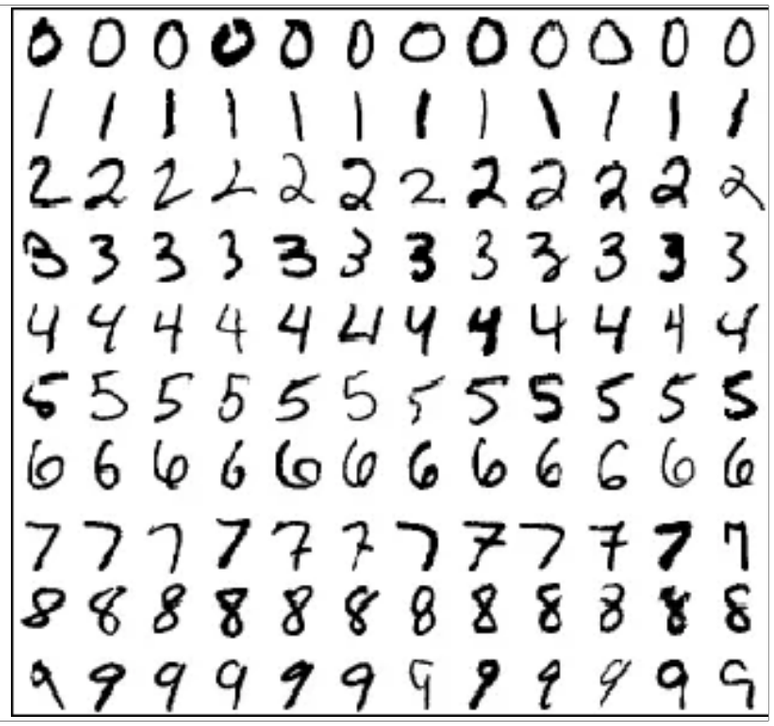
Handwritten Digit Recognition



LeNet was proposed for the task of Handwritten digit recognition task. Eg., we hope the machine can automatically recognize the zip code on the envelop.

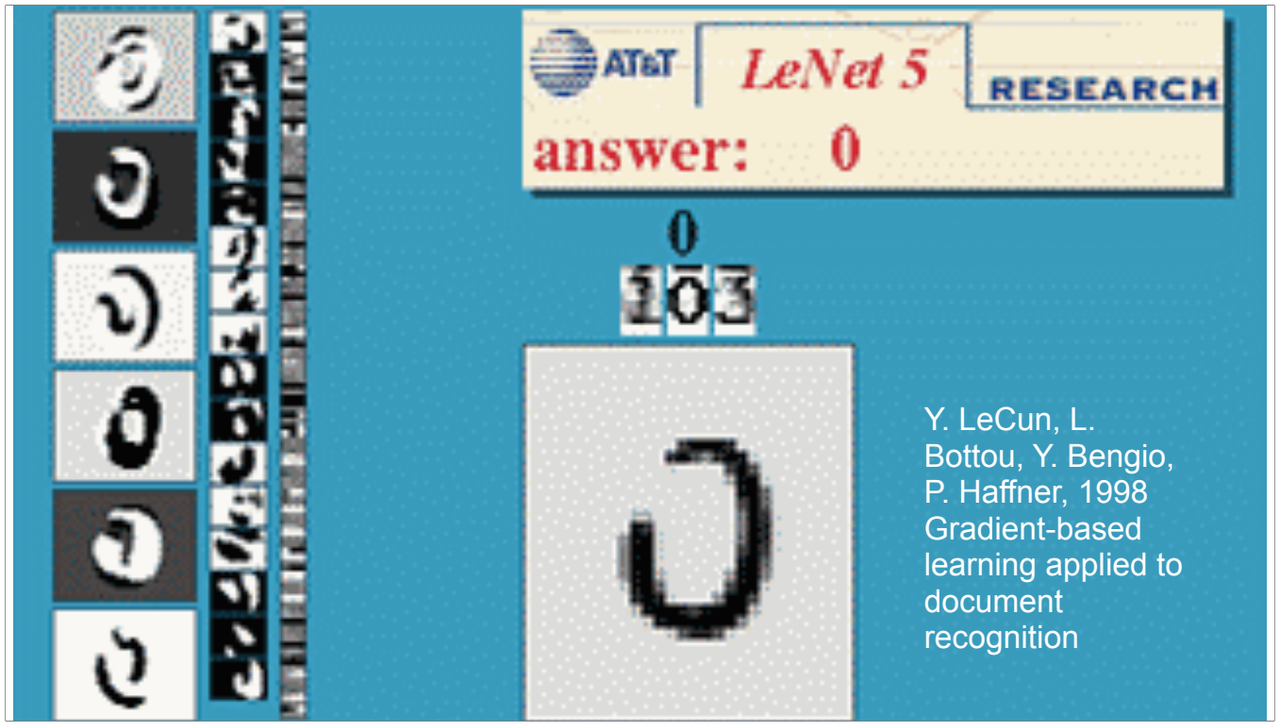
MNIST

- Centered and scaled
- 50,000 training data
- 10,000 test data
- 28 x 28 images
- 10 classes



For the task, a dataset called MNIST was collected. It contains centered and scaled images of handwritten digits (0 - 9). The images are of size 28x28, single channel (black-and-white image).

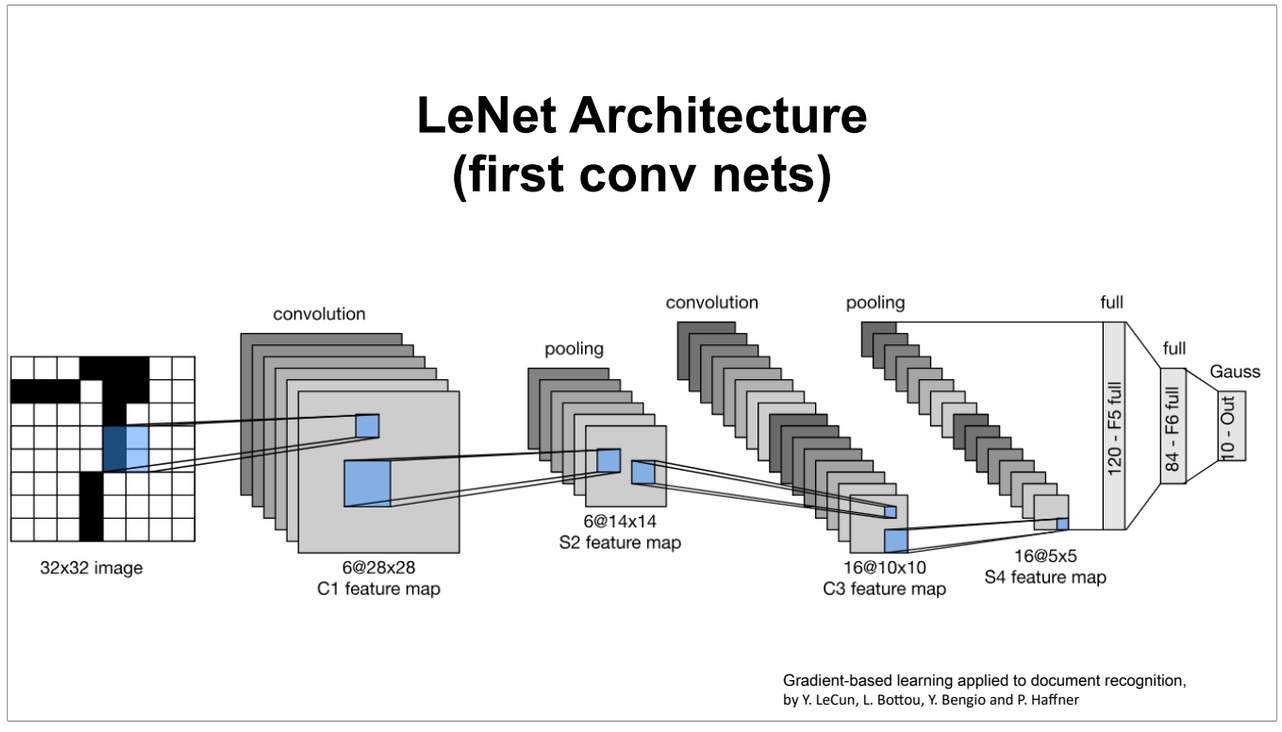
MNIST is a frequently used benchmark dataset for machine learning/AI. It is simple and small (from the modern perspective) and thus can be used as the first dataset to sanity check an algorithm. For machine learning/AI research, it’s like the fruit-fly for biology.



Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998 Gradient-based learning applied to document recognition

LeNet was published in the paper “Gradient-based learning applied to document recognition” in 1998. They also built an interface as shown in the slide.

Two of the authors, Yann LeCun and Yoshua Bengio, are among the three people who received the Turing award in 2018 (the third person is Geoffrey Hinton), for their contributions to deep learning. The three are sometimes referred to the "Godfathers of Deep Learning”.

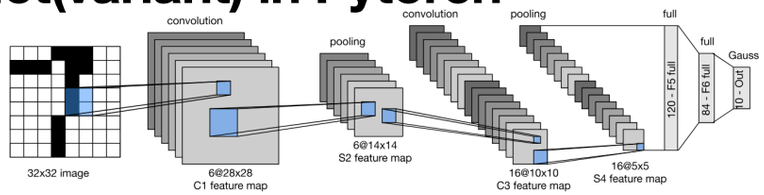


LeNet: conv1 + pooling1 + conv1 + pooling1 + three fully connected layers. (Ignore the activation function for now.)

The input is supposed to be 32x32. Since MNIST images are of 28x28, this means 4 rows/columns are padded.

Conv1: 6 kernels, kernel size 5x5, stride 1. This leads to 6 output channels (called feature maps), each of size 28x28.
Pooling1: 2x2 pooling, stride 2. This reduces the feature map size by half. Recall that pooling is applied on each input channels individually, so this gives 6 feature maps of size 14x14.
Conv2: 16 kernel, each kernel is 3D and of size 6x5x5 (6 channels to be compatible with the input). Stride 1. This leads to 16 feature maps, each of size 10x10.
Pooling2: 2x2 pooling, stride 2. This reduces the feature map size by half, giving 16 feature maps of size 5x5. The output is then flattened to a 1dim vector of size 16x5x5=400.
Fully connected layers: weight matrices of size 400x120, 120x84, 84x10. Final output of size 10: can pass to softmax to get the probabilities over the 10 classes.

LeNet(variant) in Pytorch




```
def __init__(self):
    super(LeNet5, self).__init__()
    # Convolution (In LeNet-5, 32x32 images are given as input. Hence padding of 2 is done below)
    self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1, padding=2, bias=True)
    # Max-pooling
    self.max_pool_1 = torch.nn.MaxPool2d(kernel_size=2)
    # Convolution
    self.conv2 = torch.nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1, padding=0, bias=True)
    # Max-pooling
    self.max_pool_2 = torch.nn.MaxPool2d(kernel_size=2)
    # Fully connected layer
    self.fc1 = torch.nn.Linear(16*5*5, 120)   # convert matrix with 16*5*5 (= 400) features to a matrix of 120 features (column
    self.fc2 = torch.nn.Linear(120, 84)       # convert matrix with 120 features to a matrix of 84 features (columns)
    self.fc3 = torch.nn.Linear(84, 10)        # convert matrix with 84 features to a matrix of 10 features (columns)
```

https://github.com/bollakarthikeya/LeNet-5-PyTorch/blob/master/lenet5_gpu.py

Implementation of (a variant of) LeNet using modern library PyTorch: only need to initialize the layers, and then define the network

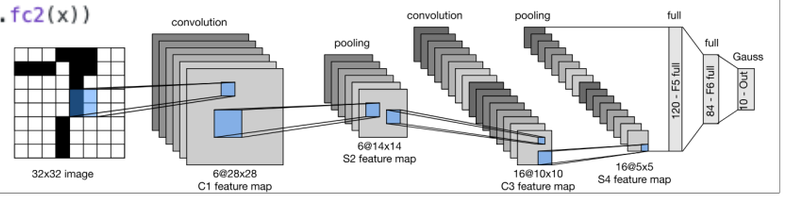
Note:

1. In PyTorch, conv1 has parameter padding=2. This actually means padding 4 rows/columns to the input. Note this difference from our notation p_h or p_w.
2. Here we use max pooling. The original LeNet used average pooling.

```python
def forward(self, x):
    # convolve, then perform ReLU non-linearity
    x = torch.nn.functional.relu(self.conv1(x))
    # max-pooling with 2x2 grid
    x = self.max_pool_1(x)
    # convolve, then perform ReLU non-linearity
    x = torch.nn.functional.relu(self.conv2(x))
    # max-pooling with 2x2 grid
    x = self.max_pool_2(x)
    # first flatten 'max_pool_2_out' to contain 16*5*5 columns
    # read through https://stackoverflow.com/a/42482819/7551231
    x = x.view(-1, 16*5*5)
    # FC-1, then perform ReLU non-linearity
    x = torch.nn.functional.relu(self.fc1(x))
    # FC-2, then perform ReLU non-linearity
    x = torch.nn.functional.relu(self.fc2(x))
    # FC-3
    x = self.fc3(x)

    return x
```

LeNet(variant) in Pytorch



The sentence x = x.view(-1, 16*5*5) means flattening the 16 feature maps output by the second pooling to a 1dim vector, so that it can be used as input to the fully connected layers.

Here we use ReLU activation. The original LeNet used sigmoid.

Deng et al. 2009

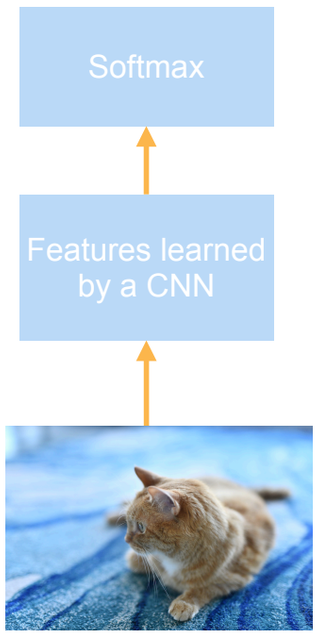
AlexNet was made possible, thanks to ImageNet which came out around 2009. This is by far one of the largest computer vision benchmark datasets: millions of images from ~1000 classes.

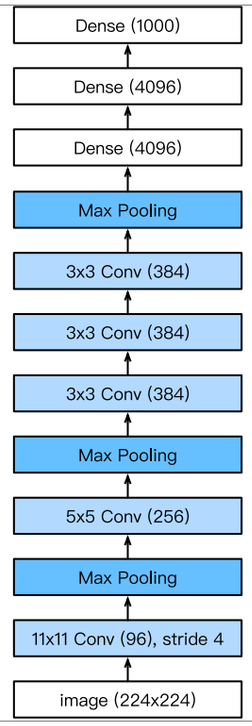
A group of researchers (in particular, Fei-Fei Li) decided to collect large scale image data to allow the training of large machine learning models. They also hosted the ImageNet competition, asking researchers to submit their solutions to the image classification tasks on ImageNet. The collection of such a large labeled dataset is possible due to: 1) there are many unlabeled images on the Internet; 2) they used mechanical turk from Amazon to label the collected unlabeled images.

ImageNet in turn enables the development of deep networks, which leads to paradigm shift in computer vision/machine learning/AI.

AlexNet

- AlexNet won ImageNet competition in 2012
- Deeper and bigger LeNet
- Paradigm shift for computer vision

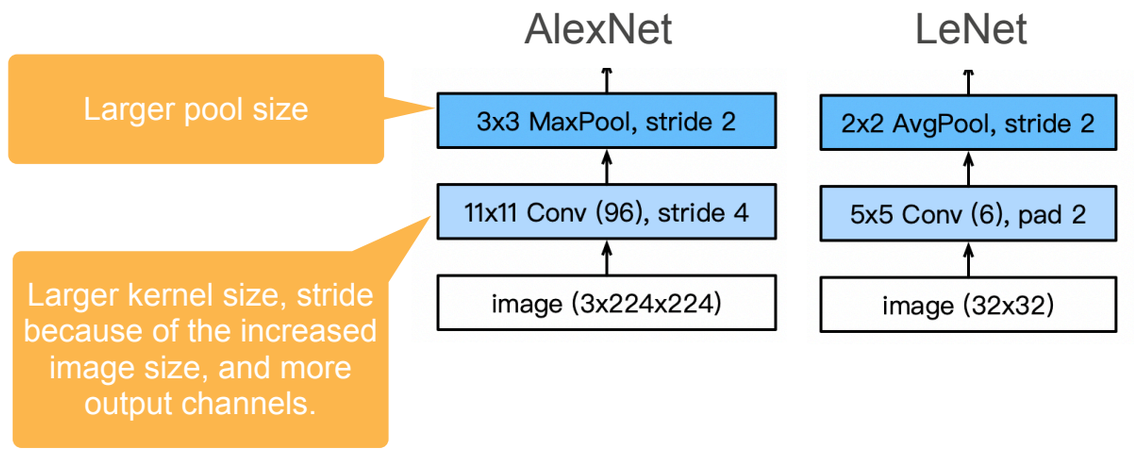




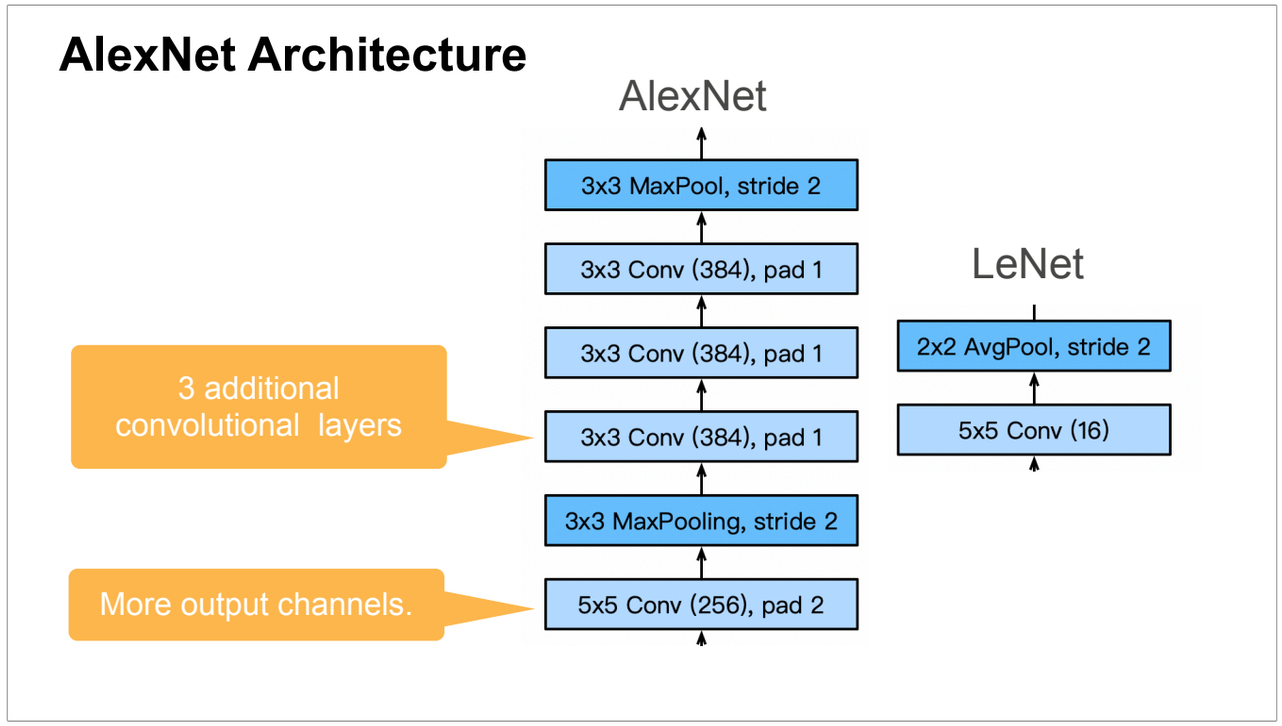
AlexNet won the ImageNet competition in 2012, with significant advantage over the other methods. After that, the community turned to deep learning.

Most models for image classification (including AlexNet) can be viewed as first extracting features from the images, and then pass the extracted features to a classifier with softmax leading to probabilities over the classes. Before AlexNet, the feature extraction step was usually done by handcrafted methods. AlexNet proposed to use CNN to extract the features, similar to what LeNet has done.

AlexNet Architecture

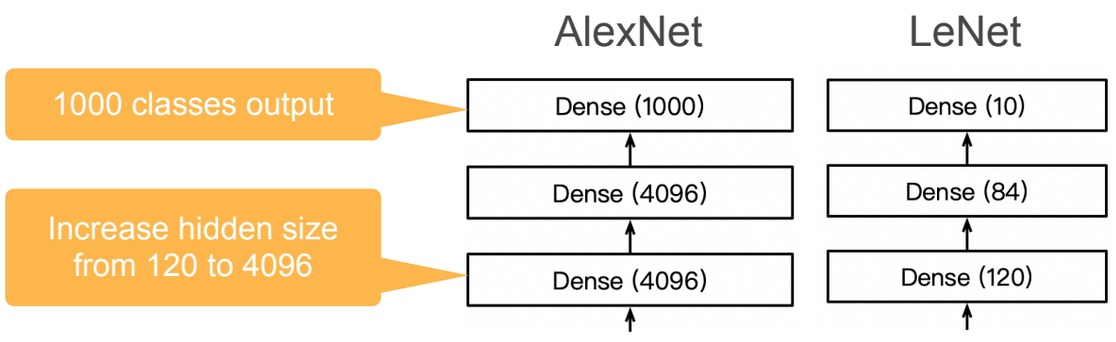


In the lower level, AlexNet also used conv+pooling. But the kernel size is 11x11 larger than that in LeNet. It used 96 kernel (each of size 3x11x11), so more feature maps (output channels). Also use larger pool size.



In the middle levels, AlexNet used more conv layers. Also keep more output channels.

AlexNet Architecture



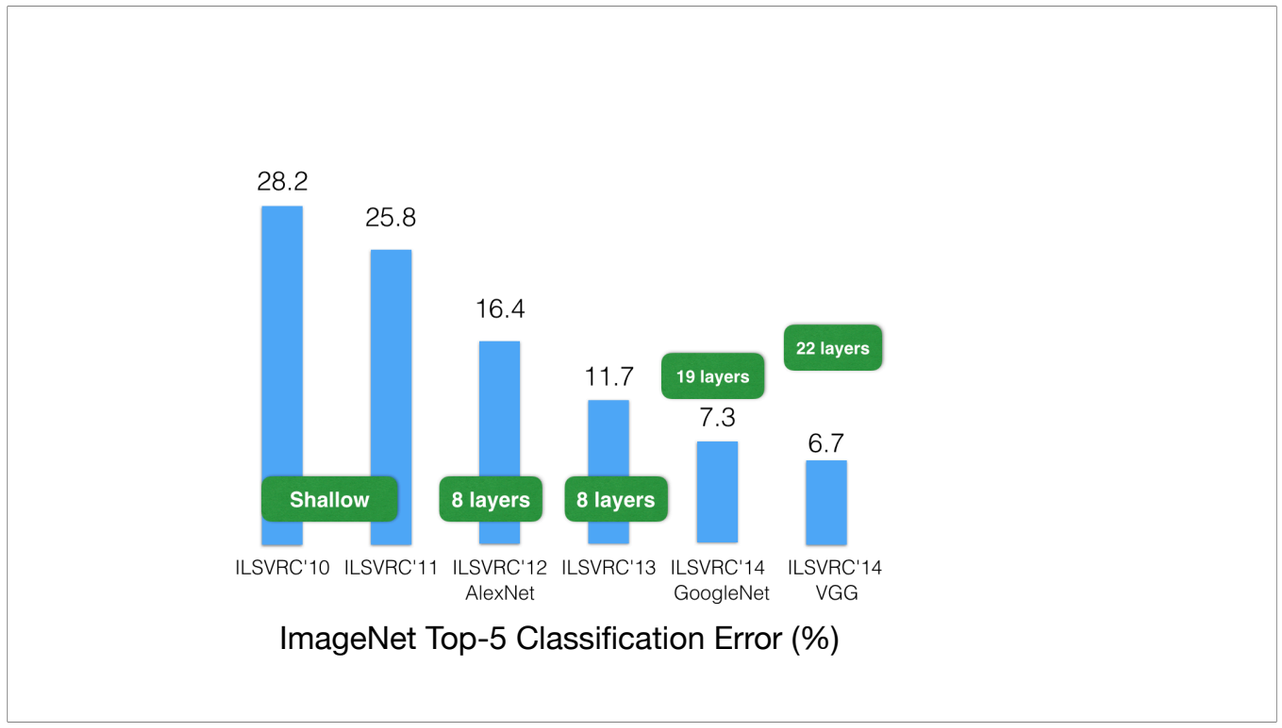
The fully connected layers are also larger. In particular, ImageNet has 1000 classes, so the last layer needs to output a vector of size 1000. The layers before that also need to be large. The number of parameters is thus >1000 times larger than that of LeNet.

More Differences…

- Change activation function from sigmoid to ReLu (no more vanishing gradient)
- Data augmentation

Some other technical differences:

1. Sigmoid —> ReLU
2. Data augmentation: for an training image, do transformations (color change, brightness, cropping etc) to get various variants of the image with the same label; add the augmented images to the training dataset, which increases the performance.



AlexNet reduced the error by a large amount. After AlexNet, the winners of the competition are all deep networks, and the number of layers keeps growing.

Q2: Which of the following are true about AlexNet? Select all that apply.

A. Let's view convolution+pooling as a composition layer. Then AlexNet contains 8 layers. The first five are convolutional layers.

B. The last three layers are fully connected layers.

C. Some of the convolutional layers are followed by max-pooling (layers).

D. AlexNet achieved excellent performance in the 2012 ImageNet challenge.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* (pp. 1097–1105).

Historically, convolution+pooling is viewed as a compositional convolutional layer. Convolution without pooling following is viewed as the standard convolutional layer.

Q2: Which of the following are true about AlexNet? Select all that apply.

A. Let's view convolution+pooling as a composition layer. Then AlexNet contains 8 layers. The first five are convolutional layers.

B. The last three layers are fully connected layers.

C. Some of the convolutional layers are followed by max-pooling (layers).

D. AlexNet achieved excellent performance in the 2012 ImageNet challenge.

All options are true!

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* (pp. 1097–1105).

AlexNet: 2 compositional convolutional layers (each with convolution+pooling) + 2 standard convolutional layers + 1 compositional convolutional layer + 3 fully connected layers.

Q3: Which of the following is true about the success of deep learning models?

A. Better design of the neural networks

B. Large scale training dataset

C. Available computing power

D. All of the above

Q3: Which of the following is true about the success of deep learning models?

A. Better design of the neural networks

B. Large scale training dataset

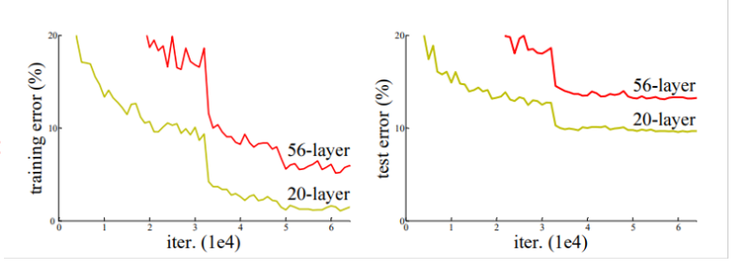
C. Available computing power

D. All of the above

Simple Idea: Add More Layers

- VGG: 19 layers. ResNet: 152 layers. Add more layers sufficient?
- No! Some problems:
  - Vanishing gradients: more layers more likely
  - Instability: can't guarantee we learn identity maps

Reflected in training error:



He et al: "Deep Residual Learning for Image Recognition"

It looks like that we can simply add more layers to get more powerful networks and get better performance.

This is not the case: simply adding more layers may not work. The authors of the ResNet paper "Deep Residual Learning for Image Recognition" reported 56-layer networks got worse performance (training/test errors) than 20-layer networks.

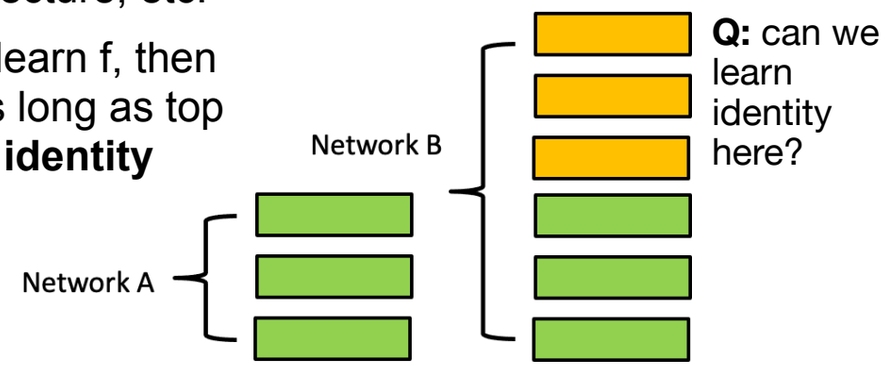
The training error can be large:

1. Vanishing gradients: the gradients w.r.t. the lower level parameters are a chain of multiplications. More layers means a longer chain of multiplication which more likely leads to vanishing gradients.
2. The instability issue: the layers may not learn the identity function easily. The identity function is simply f(x)=x. This issue leads to worse results of deep networks than shallower networks, as detailed in the next slide.

Another issue (though much less significant for this particular case and not so related to our discussion here) is that more layers means larger models, which may more likely overfit.

Depth Issues & Learning Identity

- Why would more layers result in worse performance
  - Same architecture, etc.
  - If the A can learn f, then so can B, as long as top layers learn **identity**



Idea: if layers can learn identity, **can't get** worse.

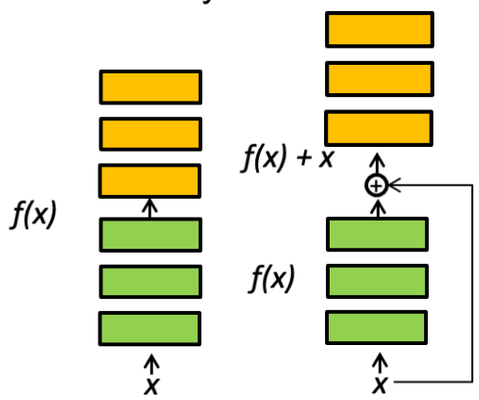
The thought experiment by the ResNet paper argues that: if there is no instability issue (i.e., the network can learn identity map easily), then deeper networks can get no worse results than shallower networks. The empirical results show that deeper networks get worse results, which means there is this instability issue and the issue leads to the worse results.

Thought experiment: Suppose we have 3-layer network A and 6 layer network B, and the layers of the networks are the same.
Suppose A can learn a function f. If the top 3 layers of B can learn the identity function, and the bottom 3 layers can learn f (as A does), then B can also learn f. Then network B can achieve as good performance as A.

However, the reality is that the top layers cannot learn the identity map easily.

Residual Connections

- **Idea:** identity might be hard to learn, but zero is easy!
  - Make all the weights tiny, produces zero for output
  - Can easily transform learning identity to learning zero:



Left: Conventional layer blocks

Right: Residual layer blocks

To learn identity f(x) = x, layers now need to learn f(x) = 0 ➔ easier

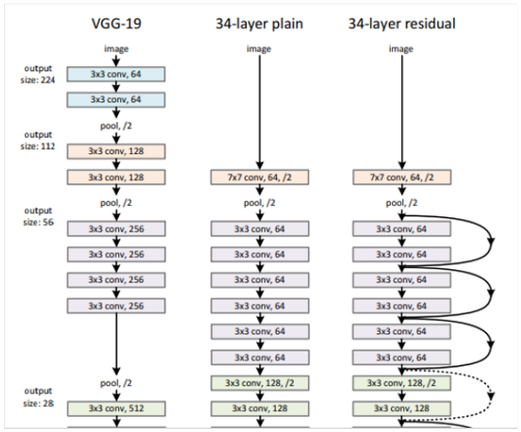
Key idea of ResNet: residual connections.

This is motivated by turning the hard task of learning identity function to the easy task of learning zero function. Learning zero function is easy: very small parameter values can guarantee close-to-0 outputs.

How to turn learning identity function to learning zero function:
Suppose we have 6-layer network, and we would like the bottom 3 layers to learn the identity function. This is not easy.
Now add a skip connection from input x to the output of the bottom 3 layers. The output of the whole block (bottom 3 layers + the skip connection) is then f(x)+x. If we want the whole block to learn the identity, we only need the bottom 3 layers to learn f(x)=0 which is easy. The block is called residual layer blocks.

ResNet Architecture

- **Idea:** Residual (skip) connections help make learning easier
  - Example architecture:
  - Note: residual connections
    - Every two layers for ResNet34
  - **Vastly better** performance
    - No additional parameters!
    - Records on many benchmarks



He et al: “Deep Residual Learning for Image Recognition”

ResNet: add residual connections every few layers, e.g., every 2 layers for 34-layer version of ResNet. There are some other versions of ResNet with different number of layers but using the same idea of residual connections.

This elegant design, though simple, is very powerful.

1. It adds no new parameters.
2. It’s easy to implement
3. It leads to significantly better performance. State-of-the-art at that time on many important benchmarks (including image classification on ImageNet)

From then on, residual connection has become a widely used idea in deep network design.

Q4: Which of the following is **NOT** true?

- A. Adding more layers can improve the performance of a neural network.
- B. Residual connections help deal with vanishing gradients.
- C. CNN architectures use no more than ~20 layers to avoid problems such as vanishing gradients.
- D. It is usually easier to learn a zero mapping than the identity mapping.

Q4: Which of the following is **NOT** true?

•A. Adding more layers can improve the performance of a neural network. (Yes, as long as we're careful, e.g., ResNets.)

•B. Residual connections help deal with vanishing gradients. (Yes, this is an explicit consideration for residual connections.)

•C. CNN architectures use no more than ~20 layers to avoid problems such as vanishing gradients. (No, much deeper networks.)

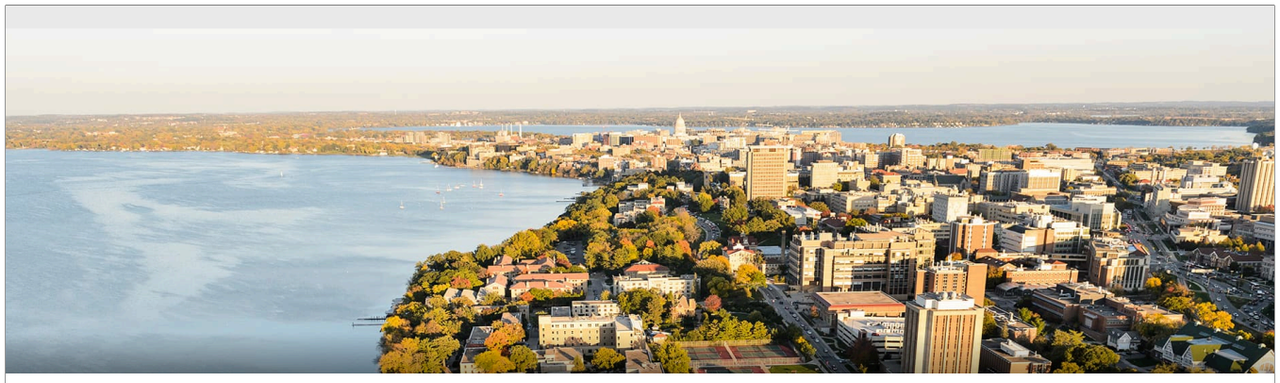
•D. It is usually easier to learn a zero mapping than the identity mapping. (Yes: simple way to learn zero is to make weights zero)

For B: consider a residual block taking as input z and outputs f(z)+z. For parameters w in the layers lower than z, their gradients contains the derivative of f(z)+z over w. This is equal to (derivative of f(z) over w) + (derivative of z over w).

(derivative of z over w) involves a much shorter chain of multiplication than (derivative of f(z) over w). This thus reduces the vanishing gradient issue.

What we've learned today

- Brief review of convolutional computations
- Convolutional Neural Networks
  - LeNet (first conv nets)
  - AlexNet
  - ResNet



Acknowledgement:

Some of the slides in these lectures have been adapted/borrowed from materials developed by Yin Li (https://happyharrycn.github.io/CS540-Fall20/schedule/), Alex Smola and Mu Li:
https://courses.d2l.ai/berkeley-stat-157/index.html