STOR566: Introduction to Deep Learning

Lecture 8: Convolutional Neural Networks

Yao Li
UNC Chapel Hill

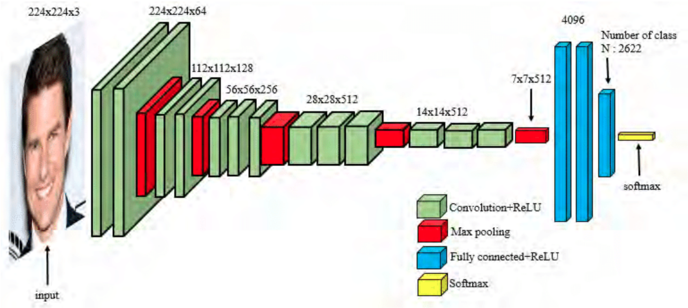
Sep 13, 2022

Materials are from *Learning from data (Caltech)* and *Deep Learning (UCLA)*

Convolutional Neural Network

The structure of CNN

- Structure of VGG



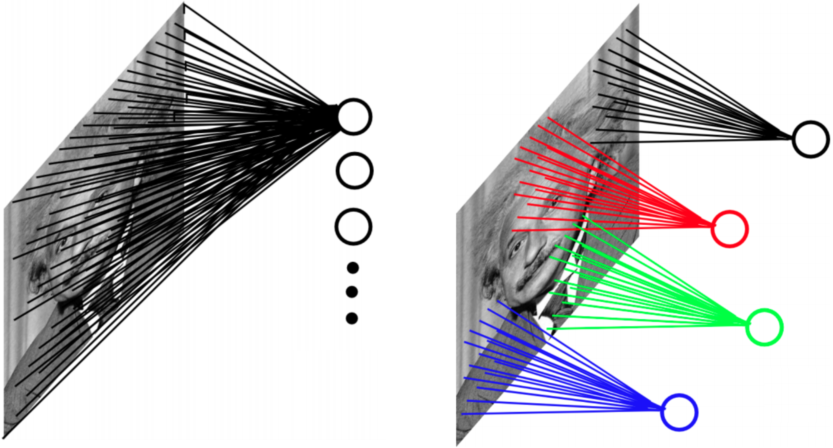
- Two important layers:
  - Convolution
  - Pooling

Convolution Layer

- Fully connected layers have too many parameters

  $\Rightarrow$ poor performance
- Example: VGG first layer
  - Input: $224 \times 224 \times 3$
  - Output: $224 \times 224 \times 64$
  - Number of parameters if we use fully connected net: $(224 \times 224 \times 3) \times (224 \times 224 \times 64) =$ 483 billion
- Convolution layer leads to:
  - Local connectivity
  - Parameter sharing

Local connectivity



(Figure from Salakhutdinov 2017)

Parameter Sharing

- Making a reasonable assumption:

If one feature is useful to compute at some spatial position (x, y), then it should also be useful to compute at a different position (x_2, y_2)

- Using the convolution operator

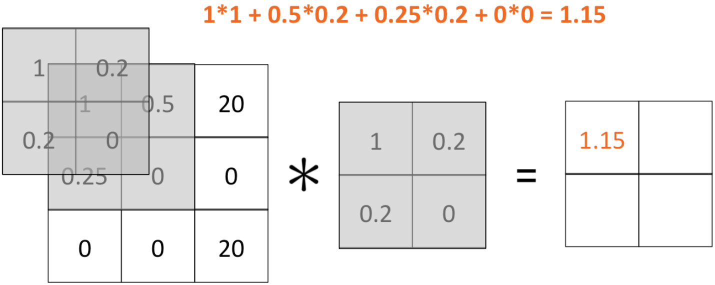
Convolution

- The convolution of an image x with a kernel k is computed as

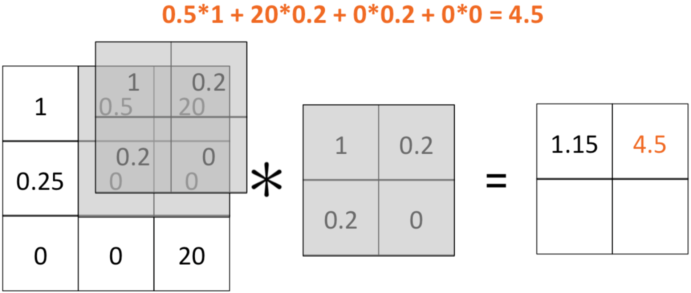
$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{p,q}$$

$$\begin{array}{|c|c|c|}\hline 1 & 0.5 & 20 \\ \hline 0.25 & 0 & 0 \\ \hline 0 & 0 & 20 \\ \hline \end{array} \; * \; \begin{array}{|c|c|}\hline 1 & 0.5 \\ \hline 0.25 & 0 \\ \hline \end{array} \; = \; \begin{array}{|c|c|}\hline & \\ \hline & \\ \hline \end{array}$$

Convolution

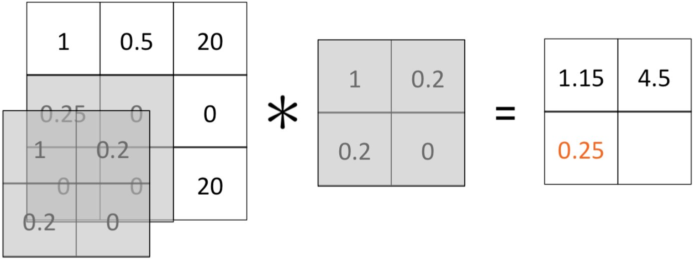


Convolution



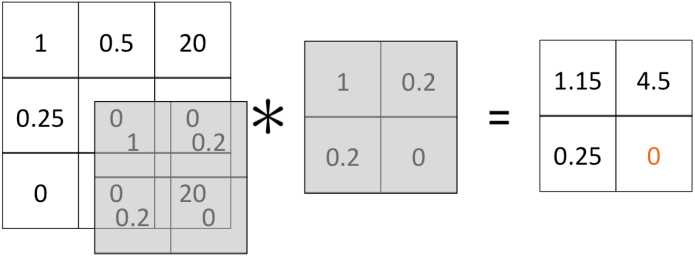
Convolution

0.25*1 + 0*0.2 + 0*0.2 + 0*0 = 0.25



Convolution

0*1 + 0*0.2 + 0*0.2 + 20*0 = 0



Multiple Channels

- Multiple input channels:

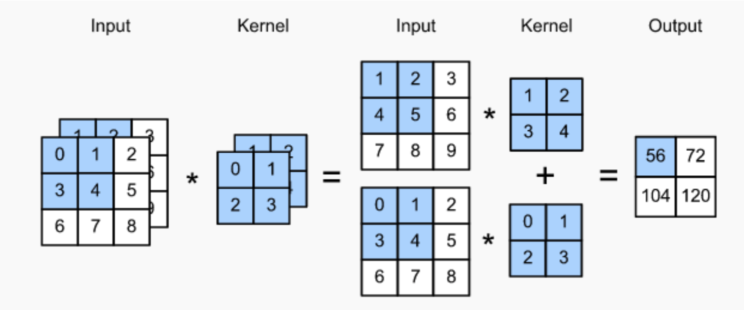
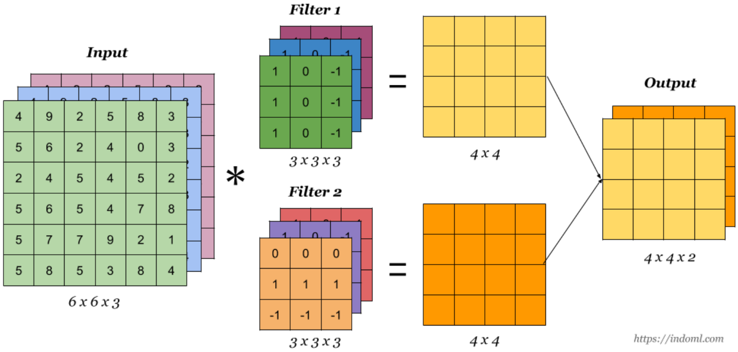


Image from Dive into Deep Learning

- $(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56$

Multiple Channels

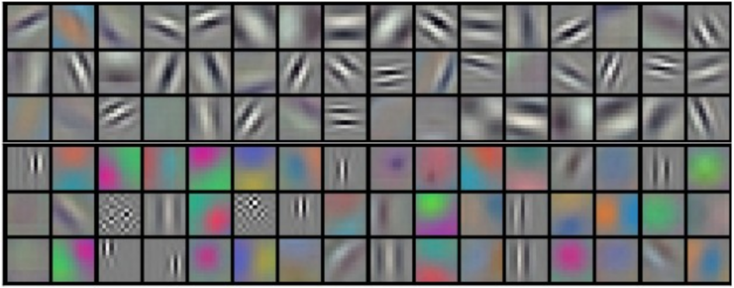
- Multiple input channels and output channels:



- Number of parameters: $k_1 \times k_2 \times d_{in} \times d_{out} + d_{out}$

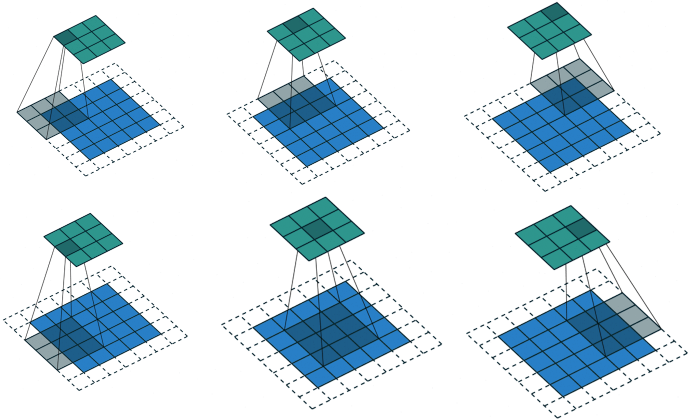
Learned Kernels

- Example kernels learned by AlexNet



Strides

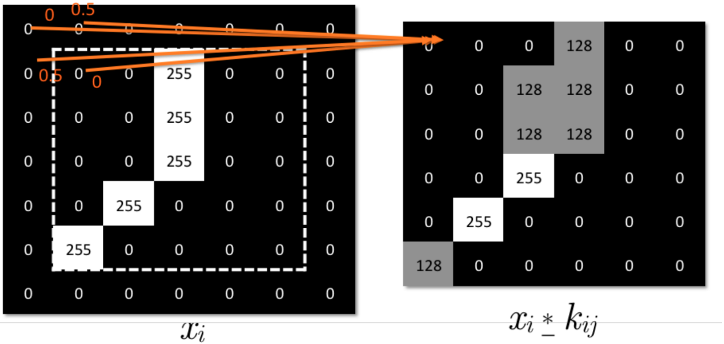
- Stride: The amount of movement between applications of the kernel to the input image
- Strude $= (1, 1)$: no stride



stride = (2,2)

Padding

- Use zero padding to allow going over the boundary
  - Easier to control the size of output layer



Output Feature Map Shape

- The shape of a output feature map depends on: shape of the input feature map, kernel size, stride, padding, etc.

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

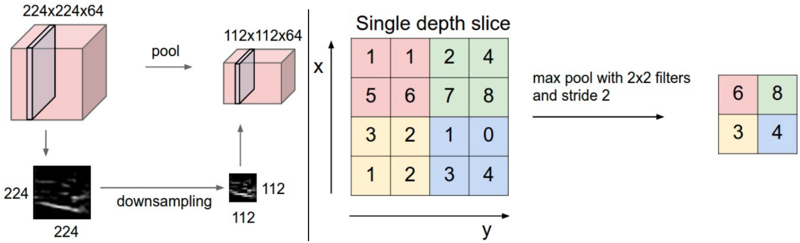
$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

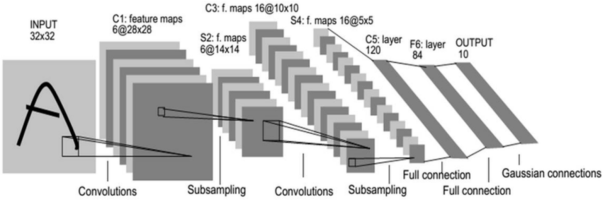
(The formula is taken from pytorch conv2d document)

Pooling

- It's common to insert a pooling layer in-between successive convolutional layers
- Reduce the size of representation, down-sampling
- Example: Max Pooling



Example: LeNet5

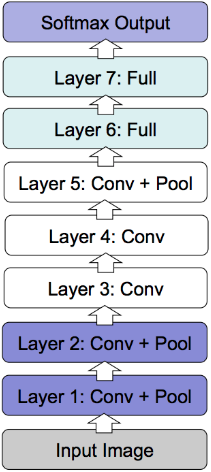


- Input: 32×32 images (MNIST)
- Convolution 1: 6 5×5 filters, stride 1
  - Output: 6 28×28 maps
- Pooling 1: 2×2 max pooling, stride 2
  - Output: 6 14×14 maps
- Convolution 2: 16 5×5 filters, stride 1
  - Output: 16 10×10 maps
- Pooling 2: 2×2 max pooling with stride 2
  - Output: 16 5×5 maps (total 400 values)
- 3 fully connected layers: $120 \Rightarrow 84 \Rightarrow 10$ neurons

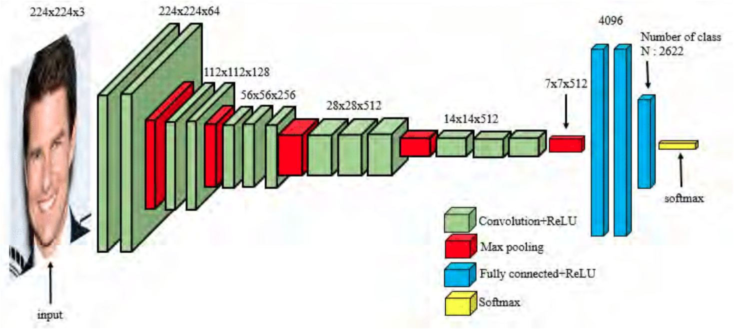
AlexNet

- 8 layers in total, about 60 million parameters and 650,000 neurons.
- Trained on ImageNet dataset

  "ImageNet Classification with Deep Convolutional Neural Networks", by Krizhevsky, Sustskever and Hinton, NIPS 2012.

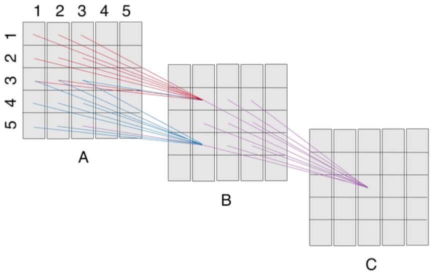


Example: VGG Network



What do the kernels learn?

- The receptive field of a neuron is the input region that can affect the neuron's output
- The receptive field for a first layer neuron is its neighbors (depending on kernel size) ⇒ capturing very local patterns
- For higher layer neurons, the receptive field can be much larger ⇒ capturing global patterns



Data Augmentation

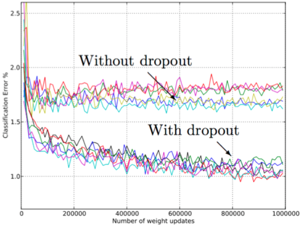
- Increase the size of data by
  - Rotation: random angle between $-\pi$ and π
  - Shift: 4 directions
  - Rescaling: random scaling up/down
  - Flipping
  - Gaussian noise
  - Many others
- Can be combined perfectly with SGD (augmentation when forming each batch)

Dropout: Regularization for neural network training

- One of the most effective regularization for deep neural networks!

Method	CIFAR-10	CIFAR-100
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	**37.20**
Conv Net + maxout (Goodfellow et al., 2013)	**11.68**	38.57

Table 4: Error rates on CIFAR-10 and CIFAR-100.

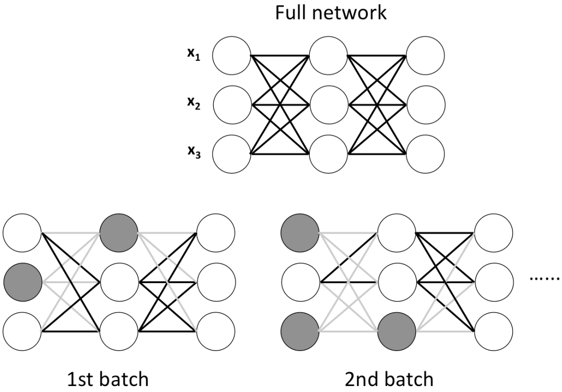


Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", 2014.

Dropout (training)

Dropout in the **training** phase:

- For each batch, turn off each neuron (including inputs) with a probability $1 - \alpha$
- Zero out the removed nodes/edges and do backpropagation.



Dropout (test time)

- Training: Each neuron computes

$$x_i^{(l)} = B\sigma(\sum_j W_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)})$$

  where B is a Bernoulli variable that takes 1 with probability α

- The expected output of the neuron:

$$E[x_i^{(l)}] = \alpha\sigma(\sum_j W_{ij}^{l} x_j^{l-1} + b_i^{l})$$

- Use the expected output at test time

  $\Rightarrow$ multiply all the weights by α

Explanations of dropout

- For a network with n neurons, there are 2^n possible sub-networks
- Dropout: randomly sample over all 2^n possibilities
- Can be viewed as a way to learn Ensemble of 2^n models

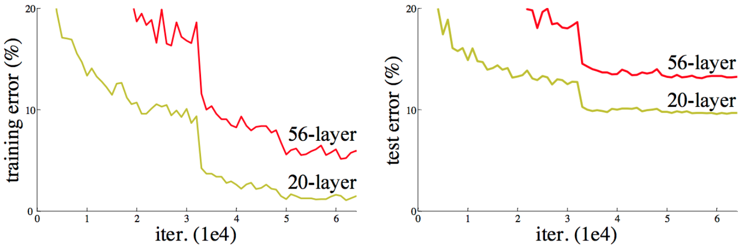
Revisit Alexnet

- Dropout: 0.5 (in FC layers)
- A lot of data augmentation
- Momentum SGD with batch size 128, momentum factor 0.9
- L2 weight decay (L2 regularization)
- Learning rate: 0.01, decreased by 10 every time when reaching a stable validation accuracy

Residual Networks

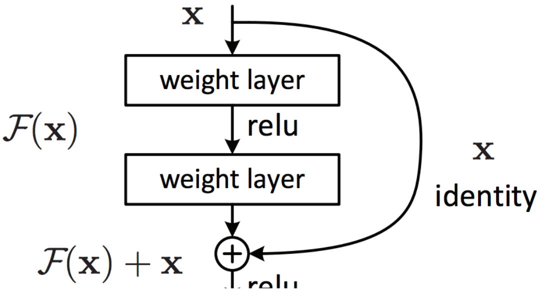
- Very deep convnets do not train well

  vanishing gradient problem



Residual Networks

- Key idea: introduce "pass through" into each layer

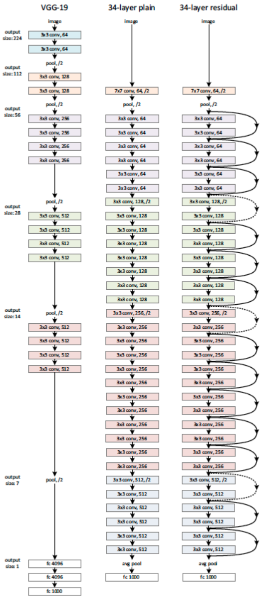


- Thus, only residual needs to be learned

Residual Networks

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43[†]
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except [†] reported on the test set).



Conclusions

- Convolution
- Pooling
- AlexNet

Questions?