CS260: Machine Learning Algorithms

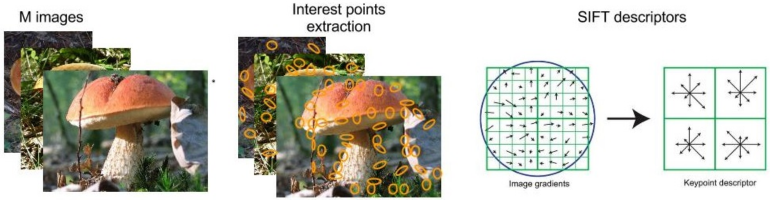
Lecture 11: Convolutional Neural Networks

Cho-Jui Hsieh
UCLA

Feb 25, 2019

Image classification without CNN

- Input an image
- Extract “interesting points” (e.g., corner detector)
- For each interesting points, extract 128-dimensional SIFT descriptor
- Clustering of SIFT descriptor to get “visual vocabulary”
- Then transform image to a feature vector (bag of visual words)
- Run classification (SVM)

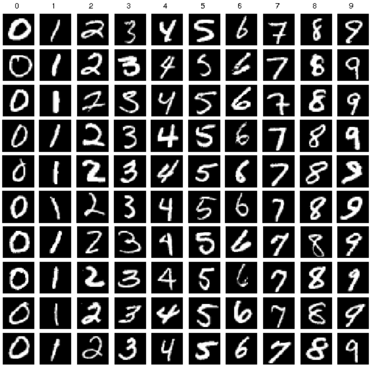


* Mushroom image by Tifred25 (http://commons.wikimedia.org/wiki/File:Bolet_Orange_01.jpg)

(picture from `http://bitsearch.blogspot.com/2013/08/image-recognition-system-classify-mushrooms.html`)

MNIST

- Hand-written digits (0 to 9)
- Total $60,000$ samples, 10-class classification.



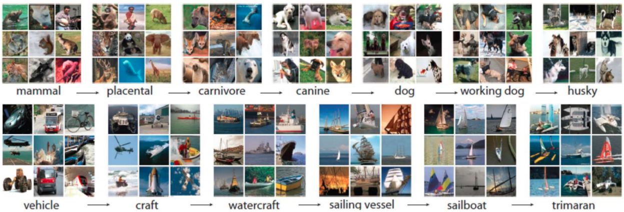
MNIST Classification Accuracy

- See the nice website by Yann LeCun:

  `http://yann.lecun.com/exdb/mnist/`

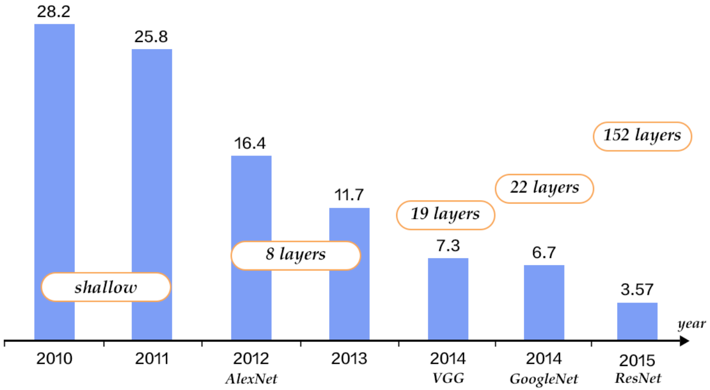
Classifier	Test Error
Linear classifier	12.0 %
SVM, Gaussian kernel	1.4%
SVM, degree 4 polynomial	1.1%
Best SVM result	0.56%
2-layer NN	$\sim$ 3.0%
3-layer NN	$\sim$ 2.5%
CNN, LeNet-5 (1998)	0.85%
Larger CNN (2011, 2012)	$\sim$ 0.3%

ImageNet Data



- ILSVRC competition: 1000 classes and about 1.2 million images
- Full imagenet: $> 20,000$ categories, each with about a thousand images.

ImageNet Results

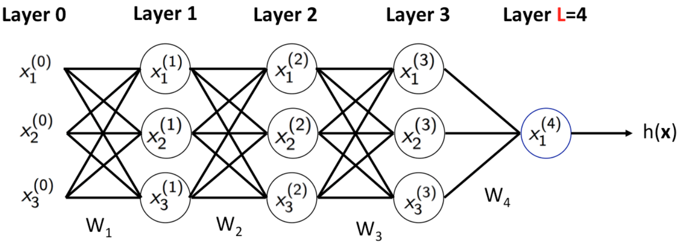


Top-5 error rates on ILSVRC image classification

picture from `http://www.paddlepaddle.org/documentation/book/en/0.14.0/03.image_classification/index.html`

Convolutional Neural Network

Neural Networks

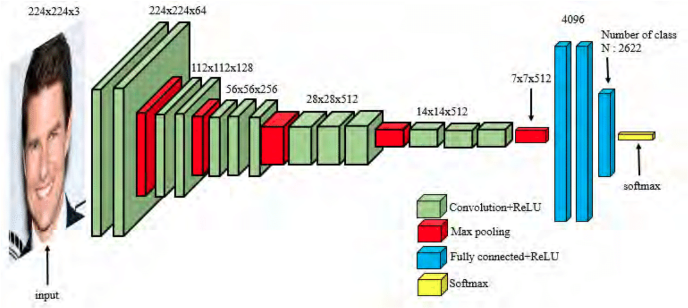


$$h(\mathbf{x}) = x_1^{(4)} = \theta(W_4\mathbf{x}^{(3)}) = \theta(W_4\theta(W_3\mathbf{x}^{(2)}))$$
$$= \cdots = \theta(W_4\theta(W_3\theta(W_2\theta(W_1\mathbf{x}))))$$

Fully connected networks $\Rightarrow$ doesn't work well for computer vision applications

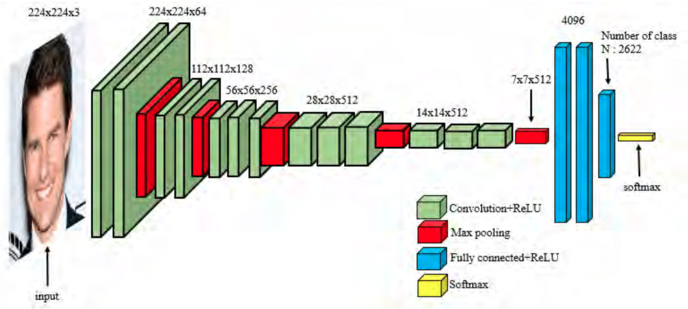
The structure of CNN

- Structure of VGG



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: $(224 \times 224 \times 3) \times (224 \times 224 \times 64) =$ 483 billion

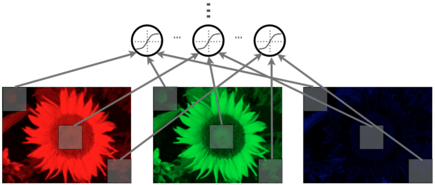
Convolution Layer

- 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: $(224 \times 224 \times 3) \times (224 \times 224 \times 64) =$ 483 billion
- Convolution layer:
  - Local connectivity
  - Parameter sharing

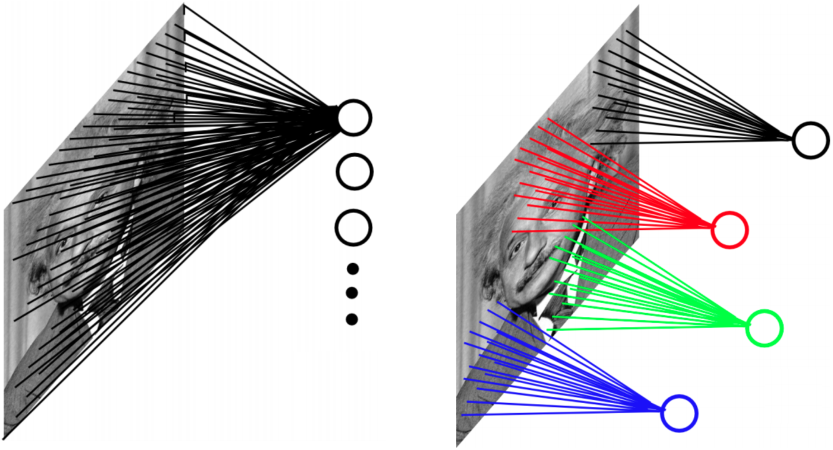
Local connectivity

- Each hidden unit is connected only to a sub-region of input
- It is connected to all channels (R, G, B)



(Figure from Salakhutdinov 2017)

Local connectivity



(Figure from Salakhutdinov 2017)

Parameter Sharing

- Making one 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

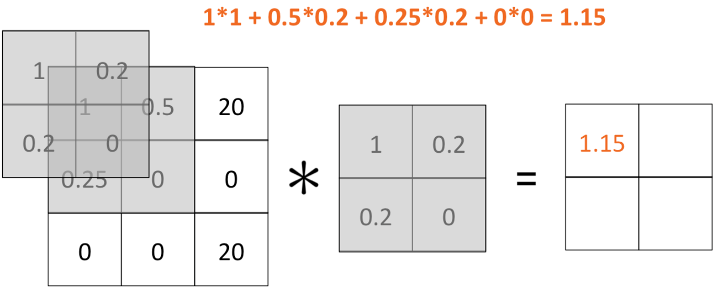
Convolution

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

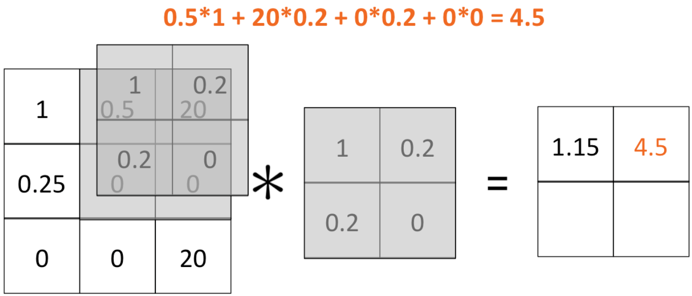
$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{p,q}$$

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

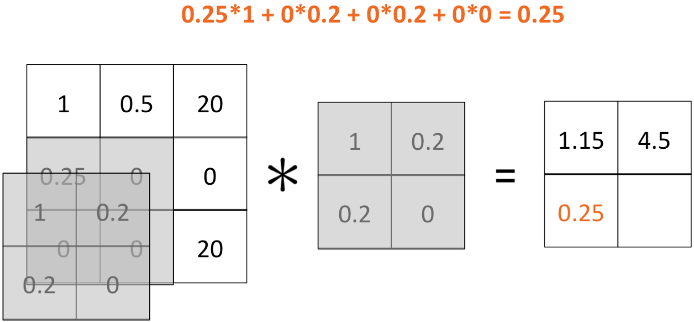
Convolution



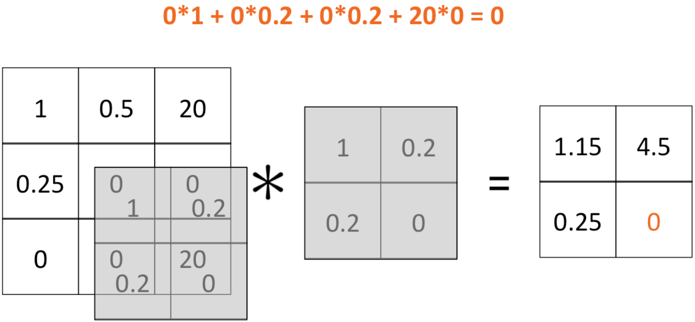
Convolution



Convolution



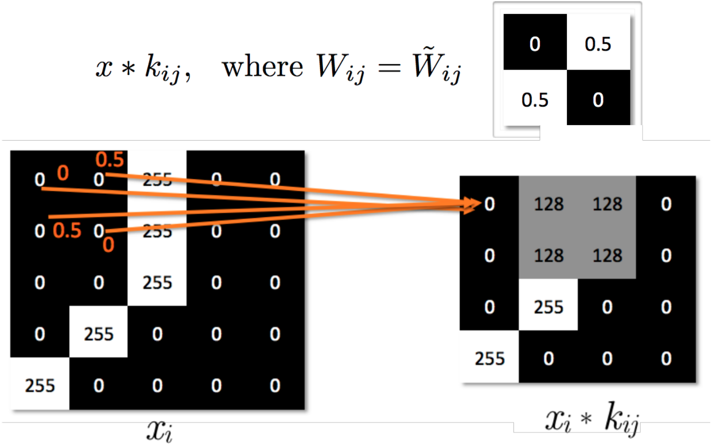
Convolution



Convolution

Illustration

$$x * k_{ij}, \quad \text{where } W_{ij} = \tilde{W}_{ij}$$



Convolution

- Element-wise activation function after convolution

  $\Rightarrow$ detector of a feature at any position in the image

$$x * k_{ij}, \quad \text{where } W_{ij} = \tilde{W}_{ij}$$

0	0.5
0.5	0

0	0	255	0	0
0	0	255	0	0
0	0	255	0	0
0	255	0	0	0
255	0	0	0	0

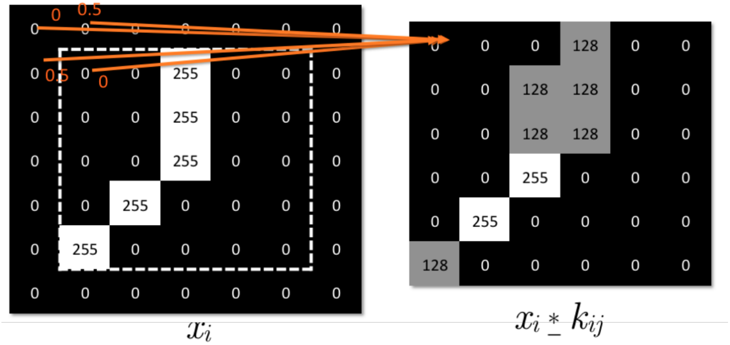
x_i

0.02	0.19	0.19	0.02
0.02	0.19	0.19	0.02
0.02	0.75	0.02	0.02
0.75	0.02	0.02	0.02

$\mathrm{sigm}(0.02\ x_i * k_{ij}\ \text{-}4)$

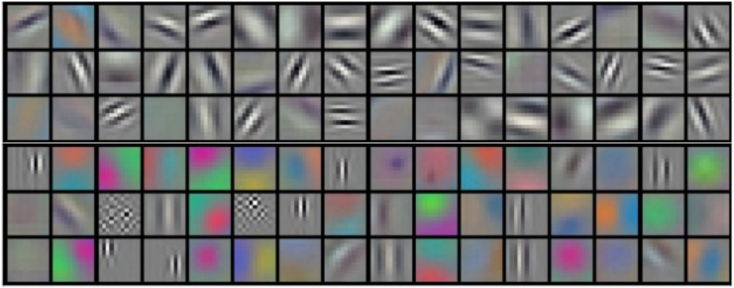
Padding

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



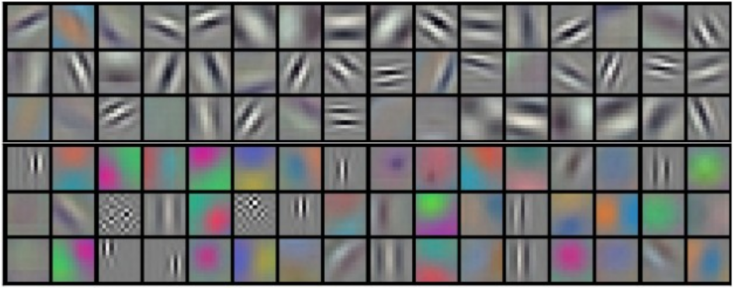
Learned Kernels

- Example kernels learned by AlexNet



Learned Kernels

- Example kernels learned by AlexNet



Number of parameters:

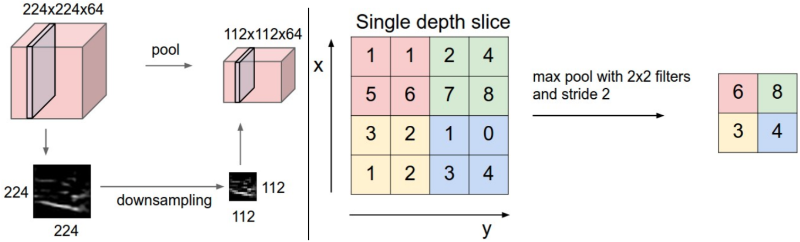
- Example: 200 $\times$ 200 image, 100 kernels, kernel size 10 $\times$ 10
- $\Rightarrow 10 \times 10 \times 100 = 10\text{K}$ parameters

Pooling

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

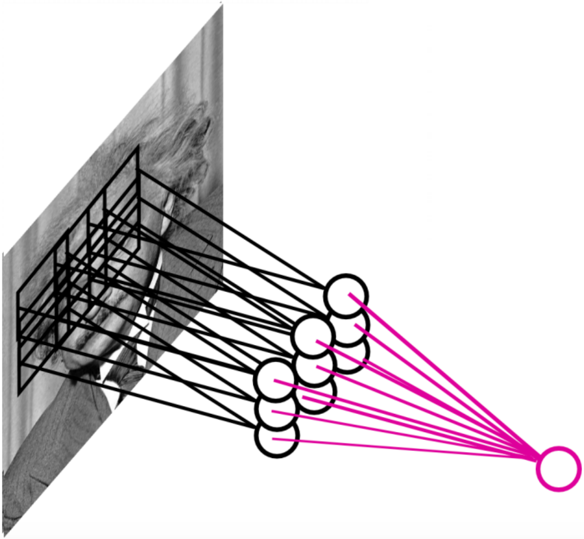
Pooling

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

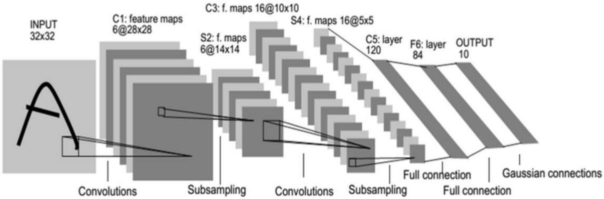


Pooling

- By pooling, we gain robustness to the exact spatial location of features



Example: LeNet5

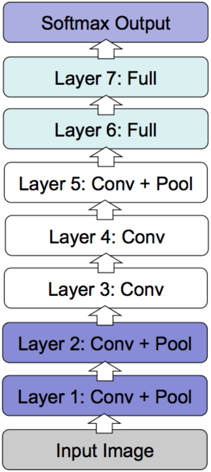


- Input: 32×32 images (MNIST)
- Convolution 1: 6 5×5 filters, stride 1
  - Output: 6 28×28 maps
- Pooling 1: 2×2 max pooling, stride 2
  - Output: 6 14×14 maps
- Convolution 2: 16 5×5 filters, stride 1
  - Output: 16 10×10 maps
- Pooling 2: 2×2 max pooling with stride 2
  - Output: 16 5×5 maps (total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

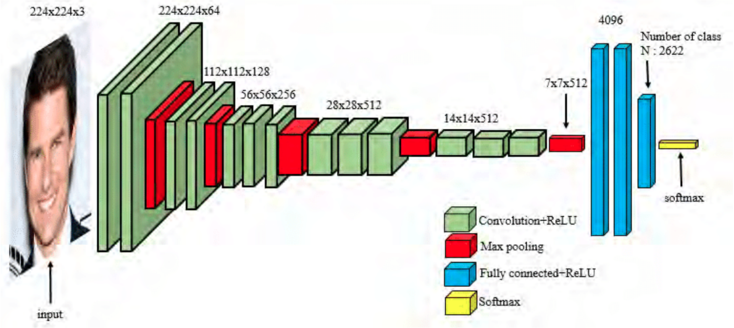
AlexNet

- 8 layers in total, about 60 million parameters and 650,000 neurons.
- Trained on ImageNet dataset
- 18.2% top-5 error

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



Example: VGG Network



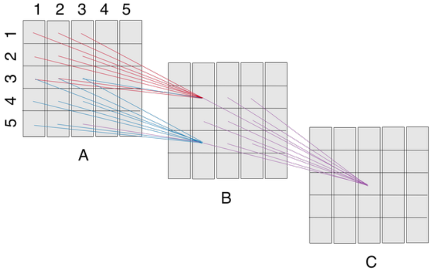
Example: VGG Network

```
INPUT: [224x224x3]        memory:  224*224*3=150K   weights: 0
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory:  112*112*64=800K   weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory:  7*7*512=25K  weights: 0
FC: [1x1x4096]  memory:  4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory:  4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory:  1000 weights: 4096*1000 = 4,096,000
```

Output provides an estimate of the conditional probability of each class

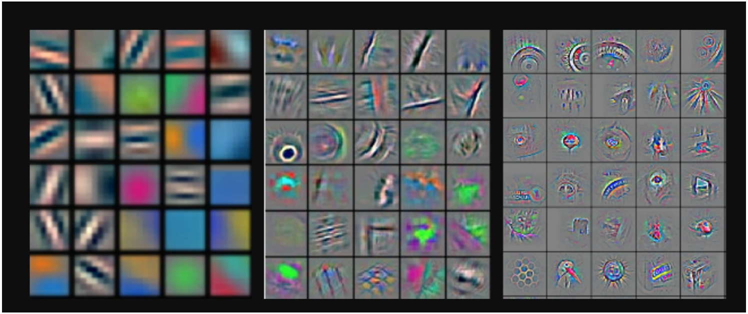
What do the filters learn?

- 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) $\Rightarrow$ capturing very local patterns
- For higher layer neurons, the receptive field can be much larger $\Rightarrow$ capturing global patterns



What do the filters learn?

- For higher layer neurons, the receptive field can be much larger $\Rightarrow$ capturing global patterns



Training

- Training:
  - Apply SGD to minimize in-sample training error
  - Backpropagation can be extended to convolutional layer and pooling layer to compute gradient!
- Millions of parameters $\Rightarrow$ easy to overfit

Data Augmentation

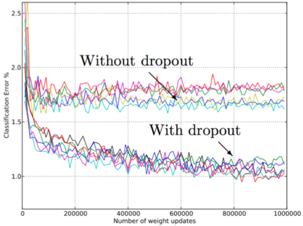
- Increase the size of data by
  - Rotation: random angle between $-\pi$ and π
  - Shift: 4 directions
  - Rescaling: random scaling up/down
  - Flipping
  - Many others
- Can be combined perfectly with SGD (augmentation when forming each batch)

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.

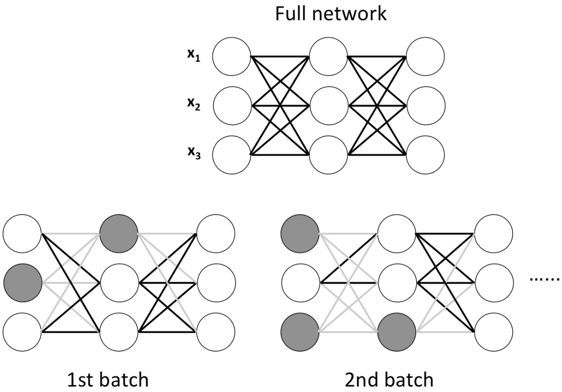


Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", 2014.

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)

- The model is different from the full model:
- 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

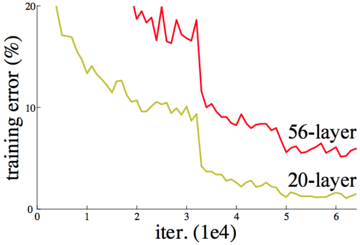
Revisit Alexnet

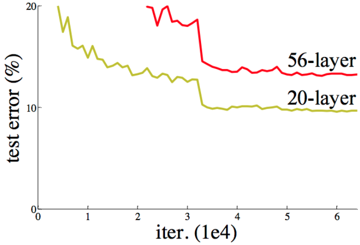
- Dropout: 0.5 (in FC layers)
- A lot of data augmentation
- Momentum SGD with batch size 128, momentum factor 0.9
- L2 weight decay (L2 regularization)
- Learning rate: 0.01, decreased by 10 every time when reaching a stable validation accuracy

Residual Networks

- Very deep convnets do not train well

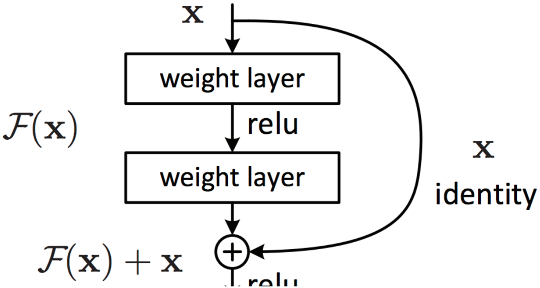
  vanishing gradient problem





Residual Networks

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

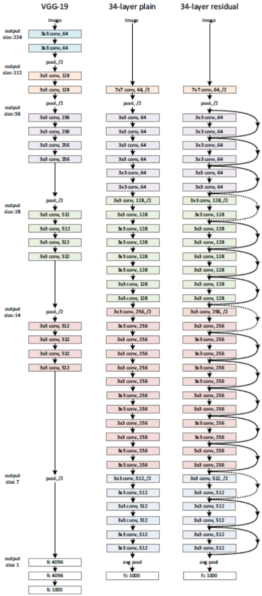


- Thus, only residual needs to be learned

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

- CNN and how to train a good image classifier.

Questions?