Convolutional Neural Networks

Presented by: Ke Yu

11/13/2018

Outline

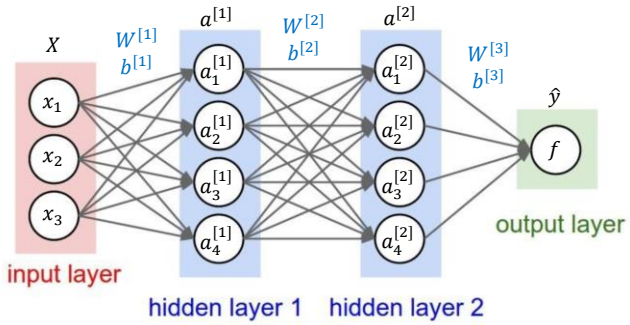
- Neural networks recap
- Building blocks of CNN
- Architectures of CNN
- Visualizing and understanding CNN
- More applications

Neural Networks Recap

Multilayer Perceptron (MLP)

Fully-connected (FC) layer

- A layer has full connections to all activations in the previous layer

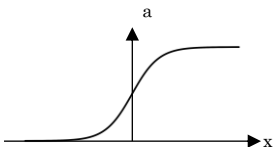


$$a^{[1]} = \sigma\left(W^{[1]}X + b^{[1]}\right)$$
$$W^{[1]}\sim(4,3), X\sim(3,m), a^{[1]}\sim(4,m)$$

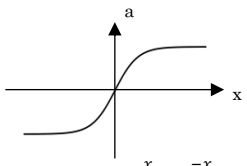
$$a^{[2]} = \sigma\left(W^{[2]}a^{[1]} + b^{[2]}\right)$$
$$W^{[2]}\sim(4,4), a^{[1]}\sim(4,m), a^{[2]}\sim(4,m)$$

$$\hat{y} = f\left(W^{[3]}a^{[2]} + b^{[3]}\right)$$
$$W^{[3]}\sim(1,4), a^{[2]}\sim(4,m), \hat{y}\sim(1,m)$$

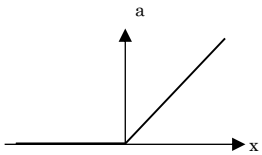
Activation Functions



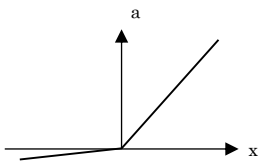
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

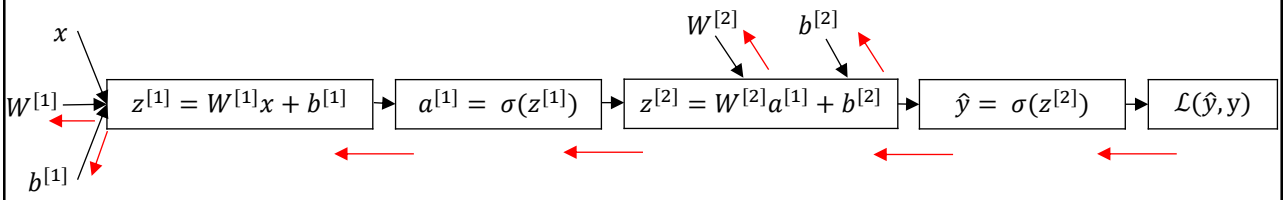


$$ReLU: \max(0, x)$$



$$Leaky\ ReLU: \max(0.1x, x)$$

Compute Gradients



$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$$

$$d[\hat{y}] = \frac{d\mathcal{L}}{d\hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$d[z^{[2]}] = \frac{d\mathcal{L}}{d\hat{y}}\frac{d\hat{y}}{dz^{[2]}} = \hat{y} - y$$

$$d[W^{[2]}] = \frac{d\mathcal{L}}{d\hat{y}}\frac{d\hat{y}}{dz^{[2]}}\frac{dz^{[2]}}{dW^{[2]}} = d[z^{[2]}]a^{[1]T}$$

$$d[b^{[2]}] = \frac{d\mathcal{L}}{d\hat{y}}\frac{d\hat{y}}{dz^{[2]}}\frac{dz^{[2]}}{db^{[2]}} = d[z^{[2]}]$$

$$d[a^{[1]}] = d[z^{[2]}]\frac{dz^{[2]}}{da^{[1]}} = W^{[2]T}d[z^{[2]}]$$

$$d[z^{[1]}] = d[a^{[1]}]\frac{da^{[1]}}{dz^{[1]}} = W^{[2]T}d[z^{[2]}] * \sigma'(z^{[1]})$$

$$dW^{[1]} = d[z^{[1]}]\frac{dz^{[1]}}{dW^{[1]}} = d[z^{[1]}]x^{T}$$

$$db^{[1]} = d[z^{[1]}]\frac{dz^{[1]}}{dW^{[1]}} = d[z^{[1]}]$$

Backpropagation Algorithm

1. The network is initialized with **randomly** chosen weights
2. Implement forward propagation to get all intermediates $z^{[l]}, a^{[l]}$
3. Compute cost function $J(W, b)$
4. Network back propagates the error and calculates the gradients
5. Adjust the weights of the network

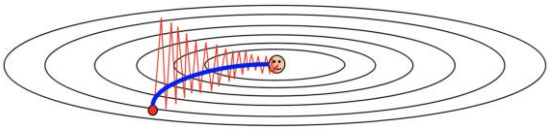
$$W^{[l]} := W^{[l]} - \alpha \cdot d[W^{[l]}]$$
$$b^{[l]} := b^{[l]} - \alpha \cdot d[b^{[l]}]$$

6. Repeat the above steps until the error is acceptable

Optimization – Learning Rate and Momentum

- Stochastic gradient descent (mini-batch gradient descent)
- SGD with momentum prevents oscillations

$$v_{dW} = \beta v_{dW} + (1-\beta)dW \qquad W = W - \alpha v_{dW},$$
$$v_{db} = \beta v_{db} + (1-\beta)db \qquad b = b - \alpha v_{db}$$



- Adaptive Learning Rate
  - RMSProp

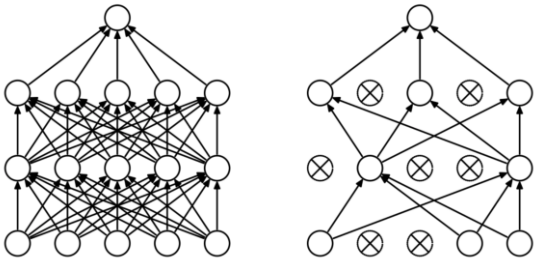
$$S_{dW} = \beta S_{dW} + (1-\beta)dW^2 \qquad W = W - \frac{\alpha}{\sqrt{S_{dW}}}dW$$

  - Adam

$$v_{dW} = \beta_1 v_{dW} + (1-\beta_1)dW \qquad S_{dW} = \beta_2 S_{dW} + (1-\beta_2)dW^2$$
$$v_{dw}^{corr} = \frac{v_{dw}}{\beta_1^t} \qquad S_{dw}^{corr} = \frac{S_{dw}}{\beta_2^t}$$
$$W = W - \frac{\alpha}{\sqrt{S_{dw}^{corr}} + \varepsilon} v_{dw}^{corr}$$

Regularization

- Parameter Regularization:
  - Adding L^1 (Lasso) , L^2 (Ridge) or sometimes combined (Elastic) to cost function
  - Other norms are computationally ineffective
- Dropout
  - Forward: multiply the output of hidden layer with mask of 0s and 1s randomly drawn from a Bernoulli distribution and remove all the links to the dropout nodes
  - Backward: do gradient descent through diminished network



Convolutional Neural Network Building Blocks

Why not just use a MLP for images?

- MLP connects each pixel in an image to each neuron and suffers from the curse of dimensionality, so it does not scale well to higher resolution images.
- For example: a small 200×200 pixel RGB image the first weight matrix of FC would have $200 \times 200 \times 3 \times \#neuron = 12{,}000 \times \#neuron$ parameters for the first layer alone

Convolution Operation

General form:

$$S(t) = \int f(a)g(t-a)da$$

Denoted by:

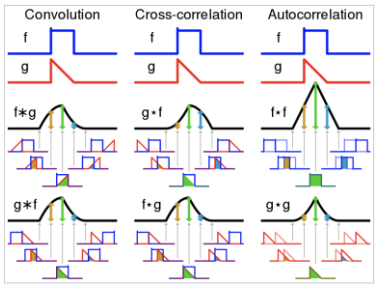
$$s(t) = (f * g)(t)$$

Network terminology:

f: input, usually a multidimensional arrays

g: kernel or filter

s: output is referred to as the feature map



Visual comparison of convolution, cross-correlation, and autocorrelation. For the operations involving function f, and assuming the height of f is 1.0, the value of the result at 5 different points is indicated by the shaded area below each point. Also, the symmetry of f is the reason $f * g$ and $f \star g$ are identical in this example.

- In practice, CNN actually uses kernels without flipping (i.e. cross-correlation)

Fast Fourier Transforms on GPUs

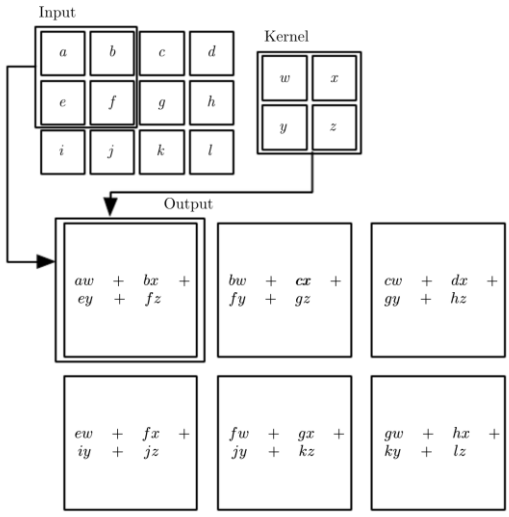
- Convolution theorem: Fourier transfer of a convolution of two signals is the pointwise product of their Fourier transforms.

$$\mathcal{F}\{x * w\} = \mathcal{F}\{x\} \cdot \mathcal{F}\{w\}$$

$$x * w = \mathcal{F}^{-1}\{\mathcal{F}\{x\} \cdot \mathcal{F}\{w\}\}$$

- Fast Fourier transfer (FFT) reduces the complexity of convolution from $O(n^2)$ to $O(n\log(n))$
- GPU-accelerated FFT implementations that perform up to 10 times faster than CPU only alternatives. (e.g. NVIDIA CUDA libraries)

2D Convolution Operation



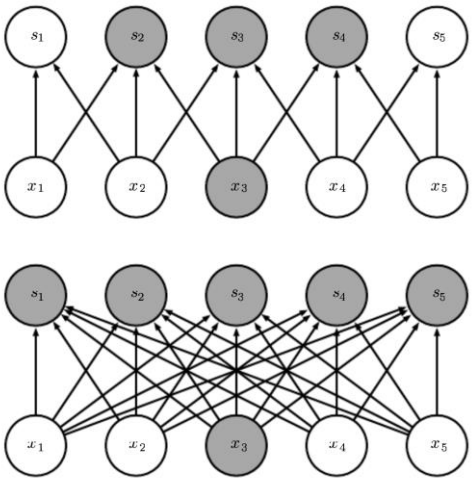
An example of 2D convolution without kernel flipping. Boxes connected by arrows indicating how the upper-left element of the output is formed by applying the kernel to the corresponding upper-left region of the input.

This process is called as template matching. The inner product between a kernel and a piece of image is maximized exactly when those two vectors match up.

Examples of kernel effects

Identity	Edge detection 1	Edge detection 2	Box blur
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
			

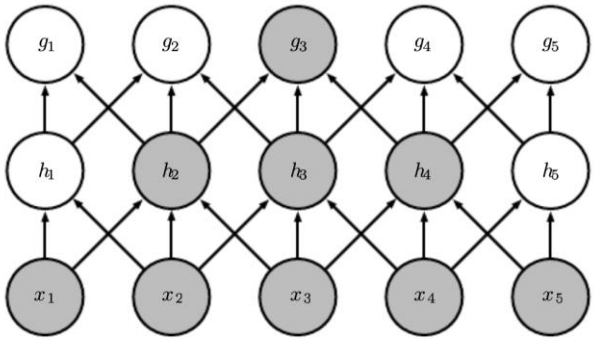
Motivation 1: Local Connectivity



- In FC layers, every output unit interacts with every input unit.
- Because kernel is usually smaller than the input, CNN typically have sparse interactions.
- Store fewer parameters which both reduces the memory requirements and improves statistical efficiency.
- Compute the output requires fewer operations.

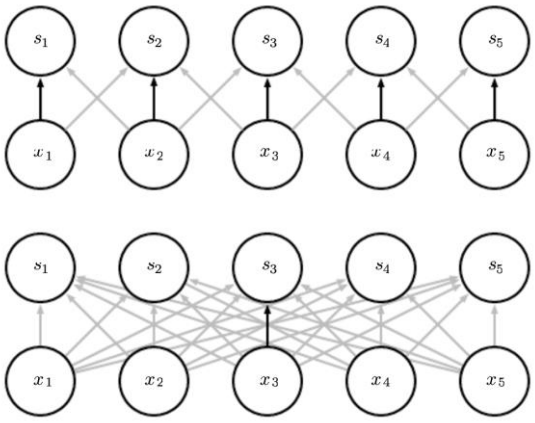
Motivation 1: Local Connectivity

Growing Receptive Fields



- In a deep convolutional network, units in the deeper layers may indirectly interact with a larger portion of the input.
- This allows the network to efficiently describe complicated interactions from constructing simple building blocks that each describe only sparse interactions.
- *For example,* h_3 is connected to 3 input variables, while g_3 is connected to all 5 input variables through indirect connections

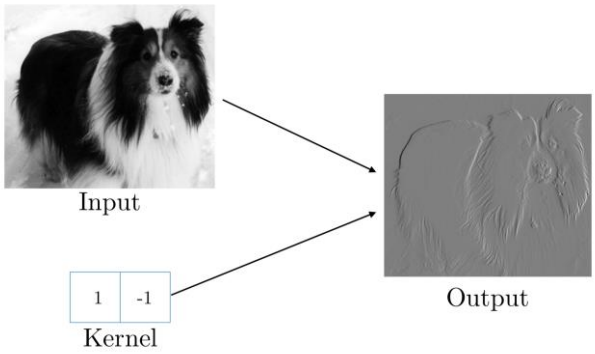
Motivation 2: Parameter Sharing



- In a traditional neural network, each element of the weight matrix is used exactly once when computing the output of a layer.
- In a convolutional neural network, each member of the kernel is used at every position of the input (except some of the boundary pixels).
- Parameter sharing means that rather than learning a separate set of parameters for every location, we learn only one set.
- It does further reduce the storage requirement of model parameters. Thus convolution is dramatically more efficient than dense matrix multiplication in terms of memory requirements and statistical efficiency

Motivation 2: Parameter Sharing

Input size: 320 by 280
Kernel size: 2 by 1
Output size: 319 by 280

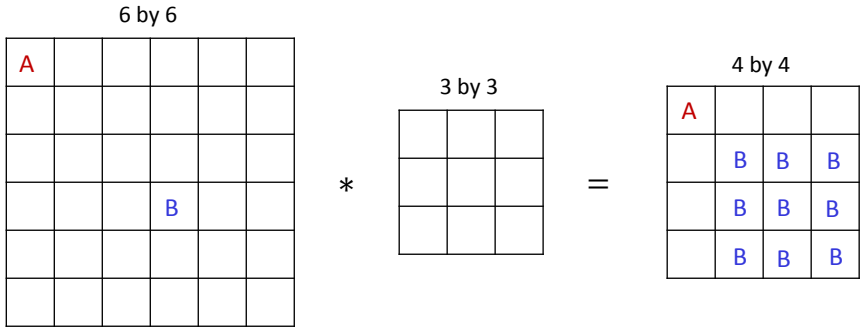


- Image on right is formed by taking each pixel and subtracting the value of its neighboring pixel. Output image shows the vertically oriented edges.
- The input image is 280 pixels tall and 320 pixels wide. The output image is 319 pixels wide.
- CNN stores 2 parameters, while to describe the same transformation with a matrix multiplication would need $320 \times 280 \times 319 \times 280 > 8e9$ weights

Motivation 3: Equivariance to Translation

- Parameter sharing causes the layer to have a property called equivariance to translation.
- With images, convolution creates a 2D feature maps. If we move the object in the input, its representation will move the same amount in the output.
- When processing images, for example, it is useful to detect edges in the first layer of a convolutional network. The same edges appear more or less everywhere in the image. Thus the same kernel can be used at different places.

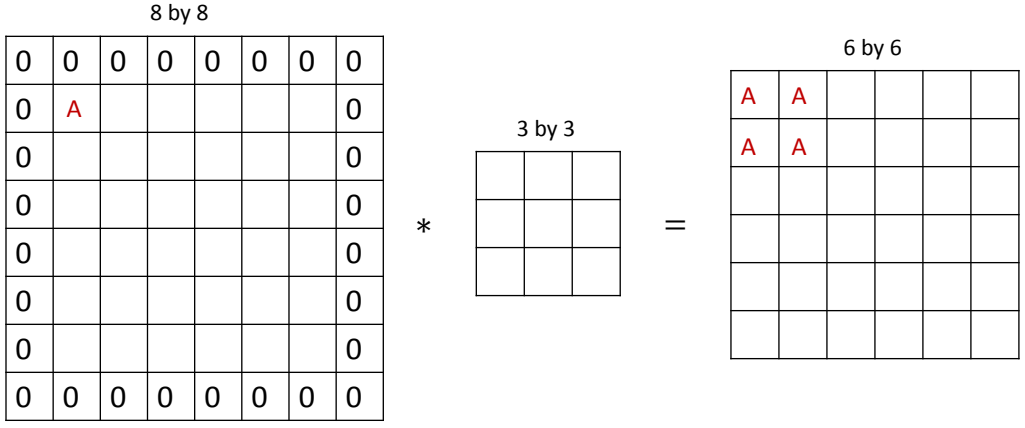
Padding



Downsides of convolution

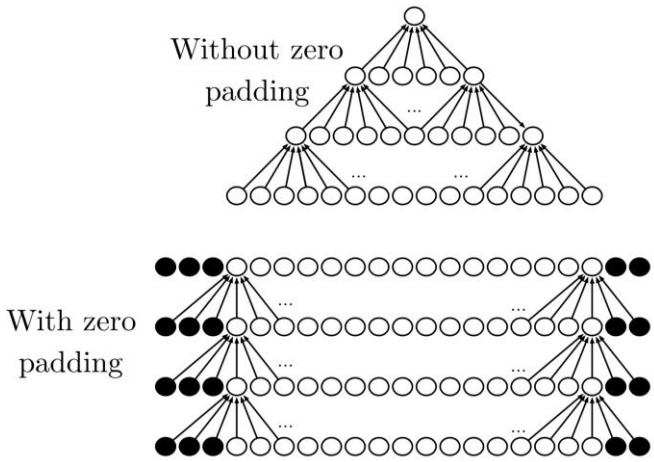
- Image shrinks after applying convolutional operation. In a very deep neural network, after many steps, we end up with a very small output.
- Pixels on the corners or edges are used much less than pixels in the middle. Lots of information from the edges of the image are throwed away.

Zero Padding



- Padding the image with additional border(s)
- Set pixel values to 0 on the border

Zero Padding Graph



- Consider a filter of width six at every layer
- Starting from an input of sixteen pixels, without zero padding, we are only able to have three convolutional layers
- Adding five zeros to each layer prevents the representation from shrinking with depth

Stride

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

$*$

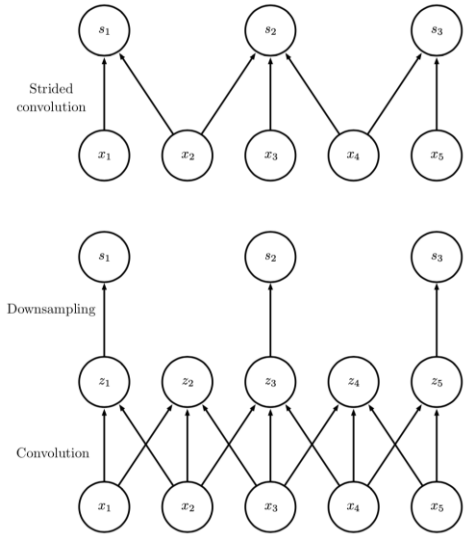
3	4	4
1	0	2
-1	0	3

$=$

91	100	83
69		

- Stride controls how far filter shifts at each step.
- Increase the stride if we want receptive fields to have less overlaps and if we want smaller output dimensions

Stride Graph

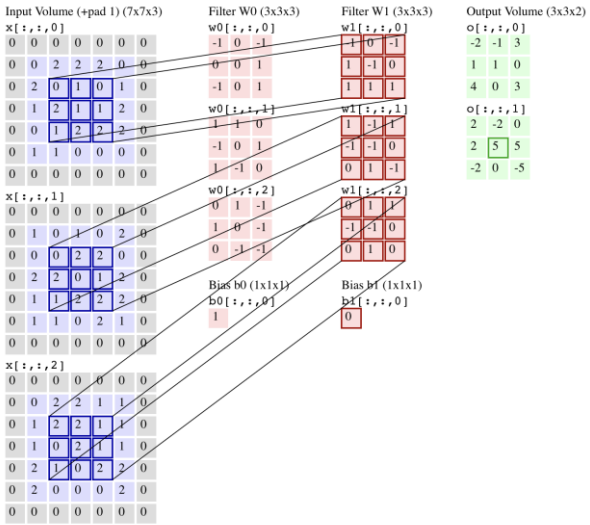


- Strided convolution is a down sampling strategy
- Having two steps approach that involves down sampling is computationally wasteful.

2D Convolution Summary

- Input size: $W_1 \times H_1$
- Hyperparameters:
  - filter size: $F \times F$
  - amount of zero padding: P
  - stride: S
- Output size:
  - $W_2 = \frac{W_1 - F + 2P}{S} + 1$
  - $H_2 = \frac{H_1 - F + 2P}{S} + 1$

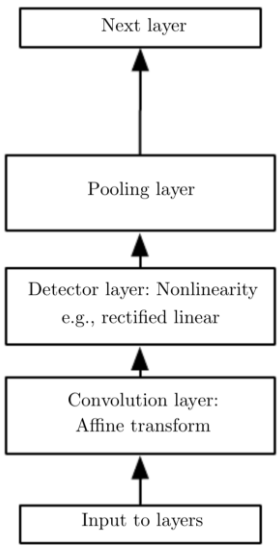
Convolutions Over Channels



http://cs231n.github.io/assets/conv-demo/index.html

- Input size: $W_1 \times H_1 \times D_1$
- Hyperparameters:
  - filter size: $F \times F \times D_1$
  - amount of zero padding: P
  - stride: S
  - number of filters: K
- Output size:
  - $W_2 = \frac{W_1 - F + 2P}{S} + 1$
  - $H_2 = \frac{H_1 - F + 2P}{S} + 1$
  - $D_2 = K$
- Number of parameters:
  - Weights: $F \times F \times D_1 \times K$
  - Bias: K

Pooling



- Pooling layer is used to reduce the spatial size of representation
- Pooling layer is usually attached after a convolutional layer
- It helps to reduce the amount of parameters and speed up the computation.
- Types:
  - Max Pooling (most popular)
  - Average Pooling
  - L2 norm of a rectangular neighborhood
- It has hyperparameters but no parameters to learn

Max Pooling

Max Pooling: $F = 2, S = 2$

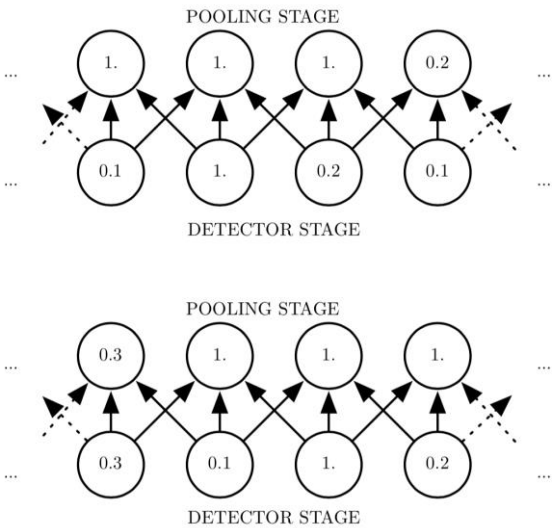
1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

9	2
6	3

Hyperparameters:
- filter size: $F \times F$
- stride: S

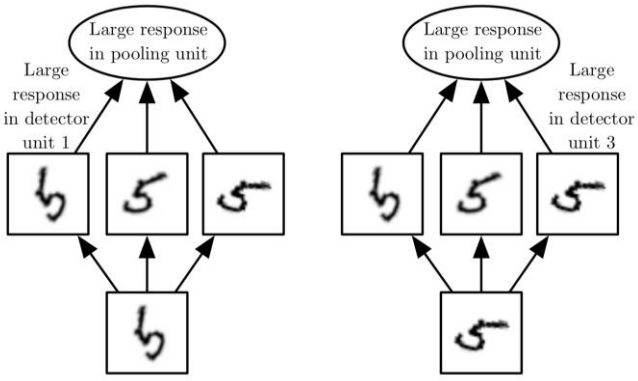
Common choices:
- $F = 2, S = 2$
- $F = 3, S = 2$

Max Pooling and Invariance to Translation



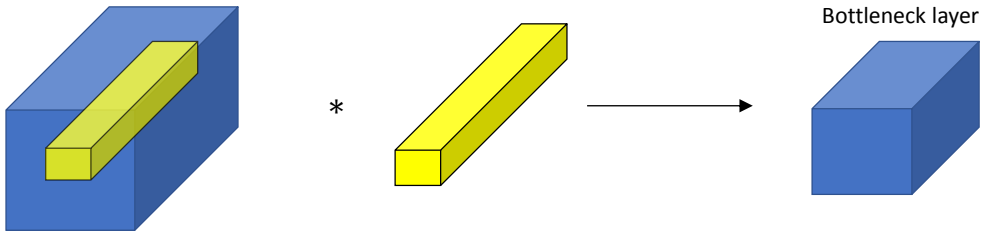
- Max pooling helps to make the representation approximately invariant to small translations of the input.
- Invariance to local translation is a useful property if we care more about whether some feature is present than exactly where it is
- *For example*, every value in the bottom row of the lower network has changed, but only half of the values in the top pooling layer have changed, because the max pooling units are sensitive only to the maximum value in the neighborhood, not its exact location

Max Pooling Cross Channels



- Pooling over multiple features (channels) can learn to be invariant to transformations of the input, such as rotation.
- *For example*, all three filters are intended to detect a hand written 5 and each filter attempts to match a slightly different orientation of the 5. The max pooling unit has a large activation regardless of which filter unit was activated.

1 by 1 Convolution



$28 \times 28 \times 192$

$1 \times 1 \times 192$

16

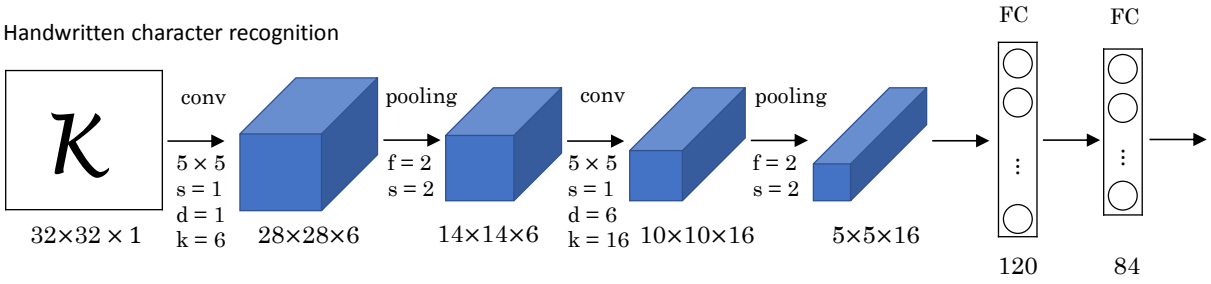
$28 \times 28 \times 16$

- 1×1 convolution shrinks the number of channels
- Creates bottleneck layer to reduce computational cost
- Used in building inception module which combines layers generated by filters with different spatial size (e.g. 1×1 , 3×3 , 5×5 , etc.)

Convolutional Neural Network Architectures

LeNet-5

Handwritten character recognition



Common pattern

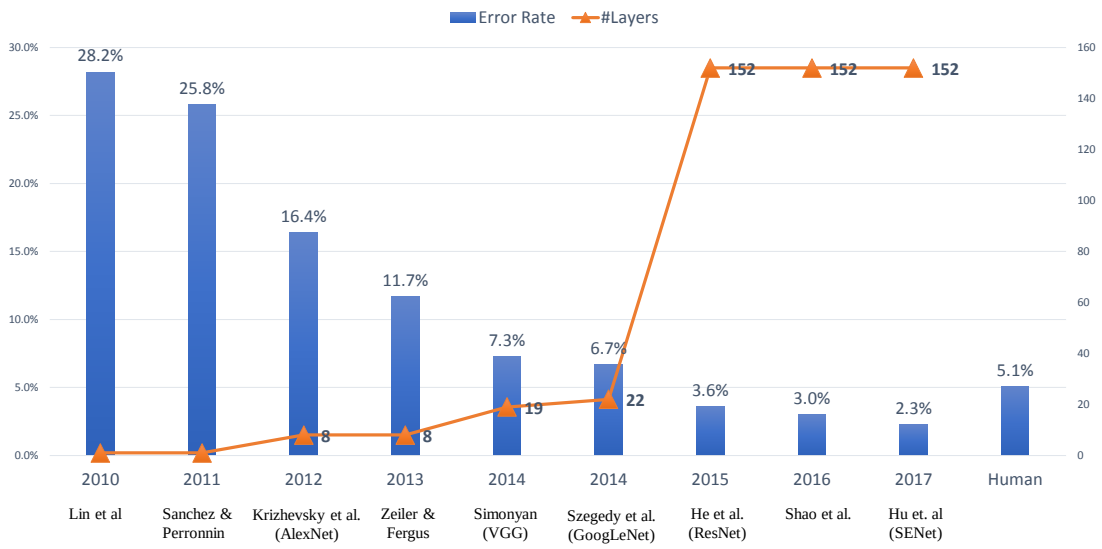
- As going deeper in the neural network, the spatial representation will usually reduce (e.g. $32 \times 32 \rightarrow 28 \times 28 \rightarrow 14 \times 14 \rightarrow 10 \times 10 \rightarrow 5 \times 5$) , while the number of channels will increase (e.g. $1 \rightarrow 6 \rightarrow 16$)
- CONV – POOL – CONV – POOL – FC – FC - SOFTMAX

LeCun, Bottou, Bengio, Haffner, "Gradient-Based Learning Applied to Document Recognition", IEEE 1998

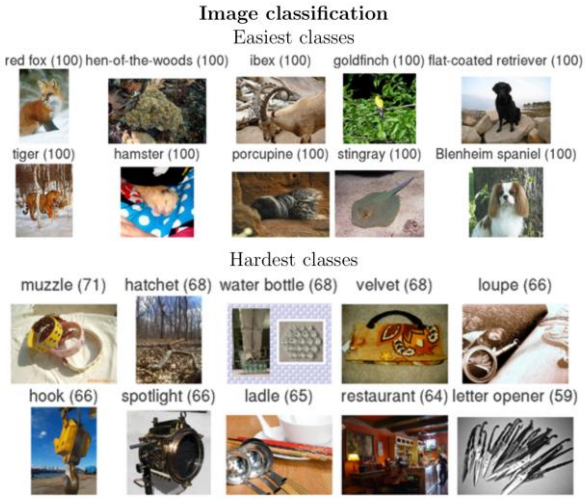
LeNet-5

	Activation shape	Activation Size	# parameters
Input:	(32,32,1)	1,024	0
CONV1 (f=5, s=1)	(28,28,6)	4,704	156
POOL1	(14,14,6)	1,176	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

ILSVRC Winners

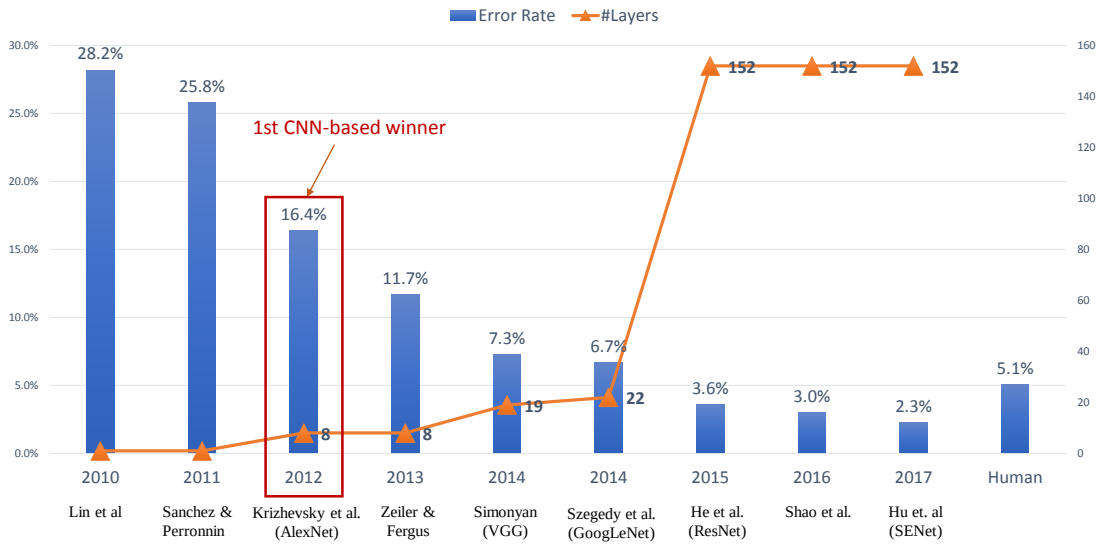


ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



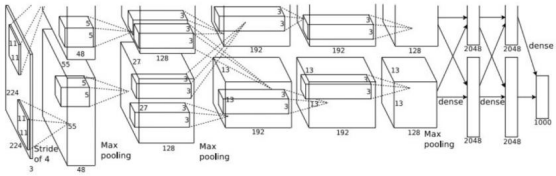
- ILSVRC is a benchmark in object category classification and detection
- The image classification: for each image, algorithm produce a list of object categories present in the image. The quality of a labeling is evaluated based on the label that best matches the ground truth label for the image.
- ~1000 images in each of 1000 categories. ~1.2 million training images, 50,000 validation images and 150,000 testing images

ILSVRC Winners - AlexNet



AlexNet

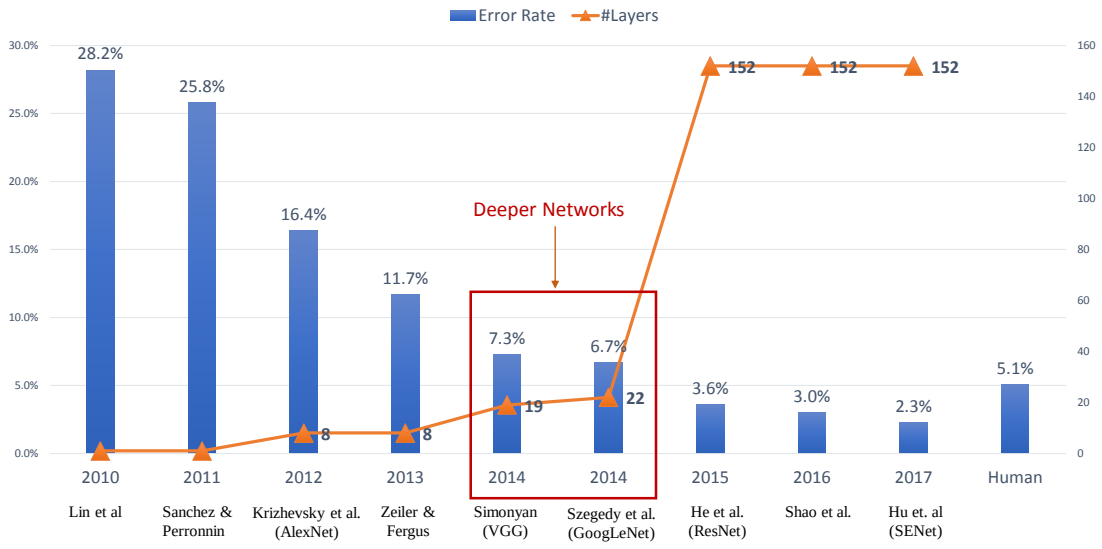
- Architecture
  - [227*227*3] Input
  - [55*55*96] CONV1: 96 11*11 filters at stride 4, pad 0
  - [27*27*96] MAX POOL1: 3*3 filters at stride 2
  - [27*27*96] NORM1: Normalization layer
  - [27*27*256] CONV2: 256 5*5 filters at stride 1, pad 2
  - [13*13*256] MAX POOL2: 3*3 filters at stride 2
  - [13*13*256] NORM2: Normalization layer
  - [13*13*384] CONV3: 384 3*3 filters at stride 1, pad 1
  - [13*13*384] CONV4: 384 3*3 filters at stride 1, pad 1
  - [13*13*256] CONV5: 256 3*3 filters at stride 1, pad 1
  - [6*6*256] MAX POOL3: 3*3 filters at stride 2
  - [4096] FC6: 4096 neurons
  - [4096] FC7: 4096 neurons
  - [1000] FC8: 1000 neurons (class scores)



- It has ~60 million parameters which is much larger than LeNet-5 (~60K parameters)
- First use of ReLU activation function
- Dropout 0.5
- SGD momentum 0.9
- Use 7 CNN ensembles to improve performance

Krizhevsky, Sutskever, Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012

ILSVRC Winners - VGG

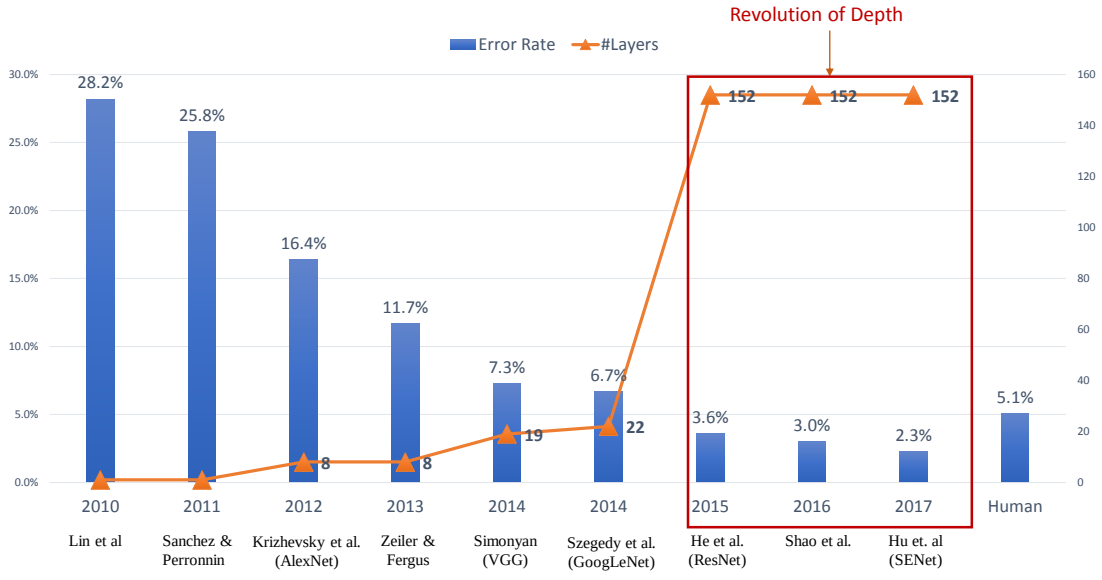


VGG

- Simplified architecture, uses uniform layer structure
  - 3*3 CONV stride 1
  - 2*2 MAX POOLING stride 2
- Stack of three 3*3 CONV stride 1 layers has the same effective receptive field as one 7*7 layer with fewer parameters 3*(3*3*K) vs. 7*7*K
- Smaller filters but deeper network and more non-linearities
- VGG16 has total 138 millions parameters
- VGG16 vs. VGG19: VGG19 is only slightly better but takes more memory
- Similar training procedure as AlexNet

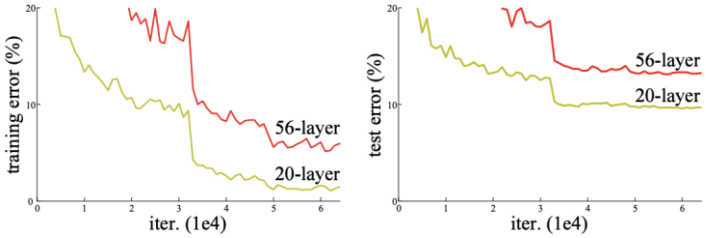
Simonyan, Zisserman, "Very Deep Convolutional Networks For Large-Scale Image Recognition", ICLR 2015

ILSVRC Winners - ResNet



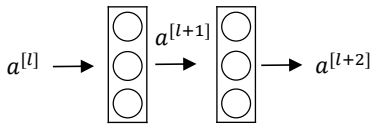
ResNet - Motivation

- What happens when we continue stacking deeper layers?
  - 56-layer model performs worse on both training and test error
  - It is not caused by overfitting
- This is a optimization problem, deeper models are harder to optimize



He, Zhang, Ren, and Sun, "Deep Residual Learning for Image Recognition", CVPR 2015

ResNets – Residual Block



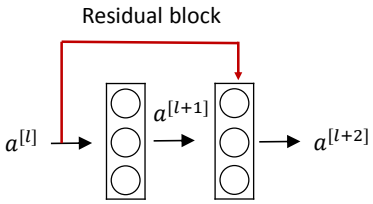
$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$

$$a^{[l+1]} = g(z^{[l+1]})$$

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+2]} = g(z^{[l+2]})$$

Residual block



$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$

$$a^{[l+1]} = g(z^{[l+1]})$$

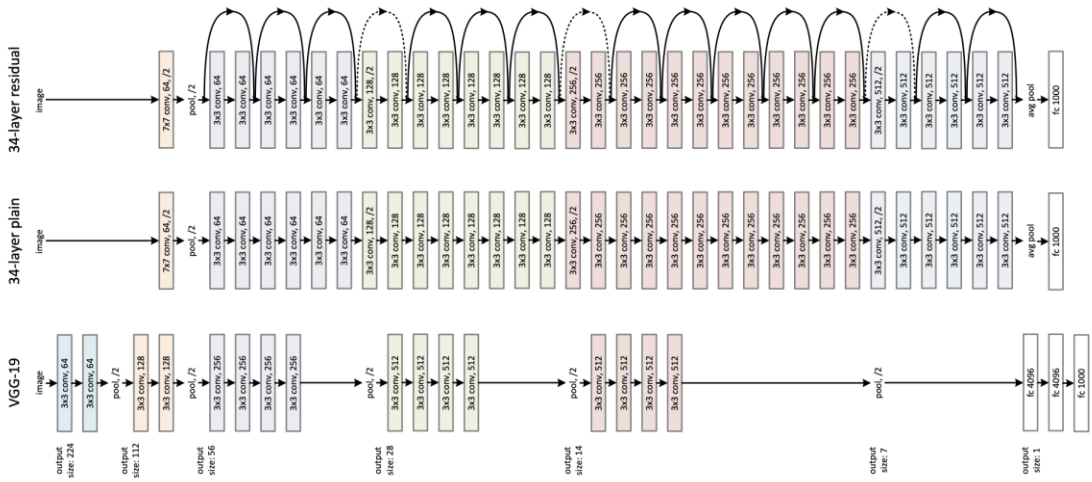
$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

$$a^{[l+2]} = g(a^{[l]}) = a^{[l]}$$

- The residual block performs as an identical function when the extra two layers doesn't learn any useful information (i.e. zero weights)
- It makes the network easy to learn with deeper layers and meanwhile guarantees extra layers don't hurt network's overall performance.

ResNet - Architecture



He, Zhang, Ren, and Sun, "Deep Residual Learning for Image Recognition", CVPR 2015

Improving ResNets

- "Good Practices for Deep Feature Fusion" [*Shao et al. 2016*]
  - Multi-scale ensembling of inception, inception-Resnet, Resnet, Wide Resnet models
  - ILSVRC'16 classification winner
- "Squeeze-and-Excitation Networks (SENet)" [*Hu et al. 2017*]
  - Add a "feature recalibration" module that learns to adaptively reweight feature maps
  - Global average pooling layer + 2 FC layers used to determine feature map weights
  - ILSVRC'17 classification winner

Transfer Learning

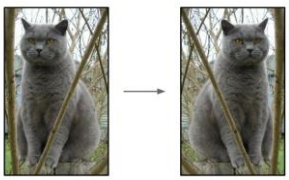
- In practice, it is rare to have a dataset of sufficient size to train an entire convolutional network from scratch.
- Pertain a CNN trained on a very large dataset (e.g. ImageNet) and use it to a related new task.
- Transfer Learning scenarios
  - When new dataset is small and similar to original dataset
    - Fixed Feature extractor: remove last classifier layer and treat the rest of the CNN as a fixed feature extractor for the new dataset.
  - When new dataset is large and similar to original dataset
    - Fine-Tuning the CNN: not only replace and retrain the last classifier, but also fine-tune all the layers, or keep some of the earlier layers fixed and only fine-tune some deeper portion of the network
  - When new dataset is large different from the original dataset
    - Weights initialization: It is very often still beneficial to initialize with weights from a pretrained model and then fine-tune through the entire network.

Data Augmentation

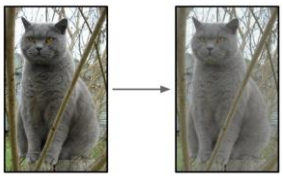
- Helps with improving model robustness and reducing overfitting
- Label-preserving transformations
- Methods:
  - Horizontal flips
  - Random crops/scales
  - Translation
  - Color jitter
  - Rotation
  - etc.

Crops/scales

Horizontal flips



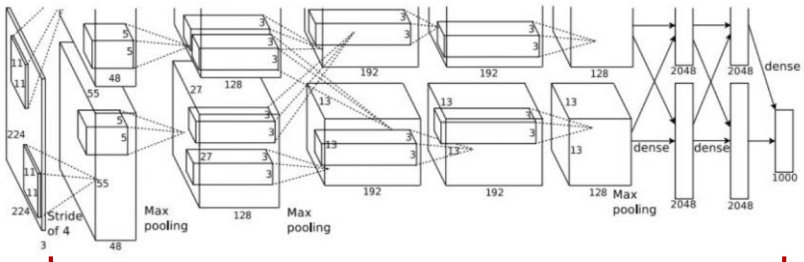
Jitter contrast



Convolutional Neural Network Visualizing and Understanding

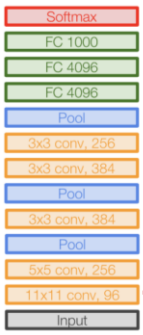
How Convolutional Networks are working?

- Can we get intuitions about what type of features in the images that CNN are looking for?
- What kind techniques we have for analyzing this internals of the network?



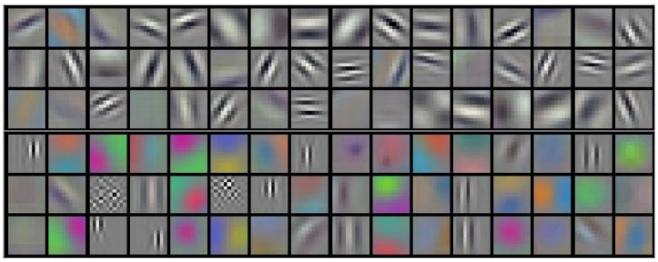
What's going on inside CNN?

First Layer



AlexNet

96 convolutional kernels of size 11 by 11 by 3 learned by the first convolutional layer on the 224 by 224 by 3 input images



Why visualize the weights of the fist layer?
Template matching - the inner product between a kernel and a piece of image is maximized exactly when those two vectors match up.

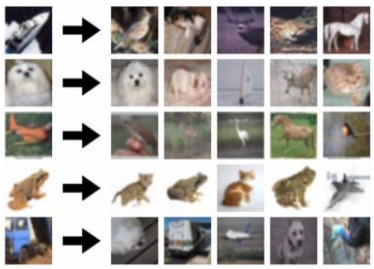
Weights in Deeper Layers

2nd convolutional filters are not very interpretable
They are connected to the nonlinear output of first layer. So this visualization shows what activation pattern after first layer would cause second convolutional layer to maximally activated.
Not very interpretable

3rd and deeper convolutional filters are getting more and more difficult to directly interpret.

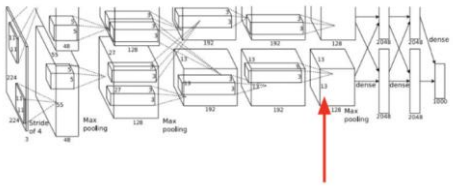
Last Layer: Nearest Neighbors

Nearest Neighbors in Pixel Space



Nearest Neighbors in Feature Space

Maximally Activating Patches



- Pick a layer and a channel, e.g. conv5, cannel 20 in AlexNet
- Run many different images through the AlexNet and record values of the chosen channel
- Visualize image patches that associated with maximal activations
- Deeper layers have larger receptive fields and look at larger objects

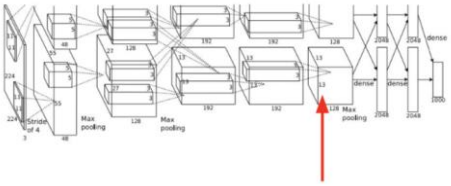
Shallower

Deeper

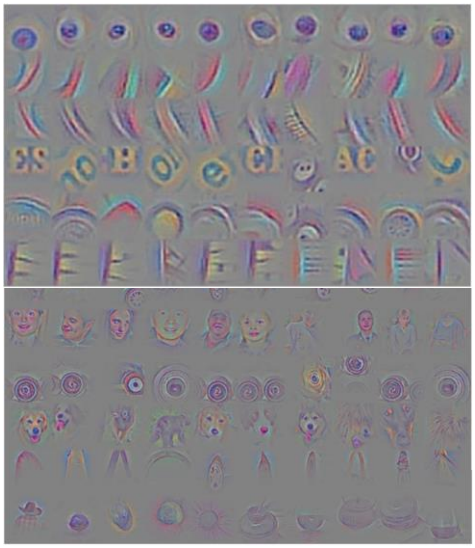
Each row is a different channel

Springenberg et al. "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

Guided Backprop



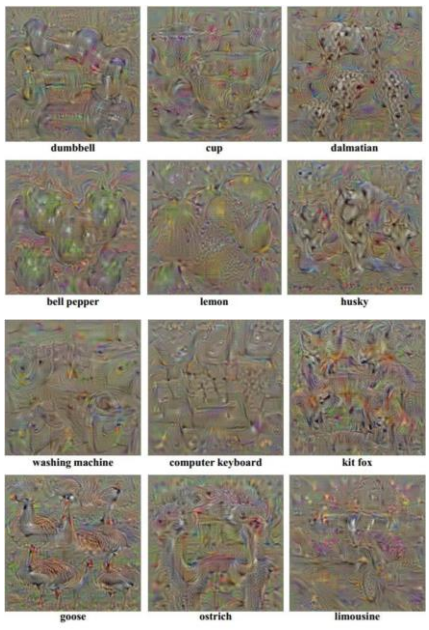
- Pick a layer and a channel, e.g. conv5, cannel 20 in AlexNet
- Compute gradient of neuron value with respect to image pixels



Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Gradient Ascent

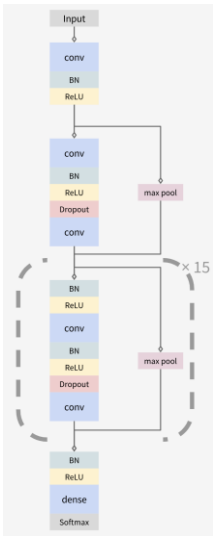
- Recall that guided backprop looks at a fixed image and tries to find which part of the image or which set of pixels influence the output of selected neuron
- Gradient Ascent tries to find what type of input in general would cause this neuron to activate
  - Fix the weights of trained network
  - Synthesize image by performing gradient ascent
  - Maximize the score of a given class or an intermediate neuron



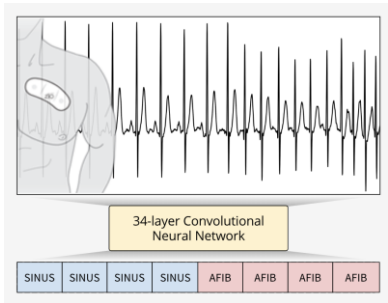
Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualizing Image Classification Models and Saliency Maps", ICLR Workshop 2014

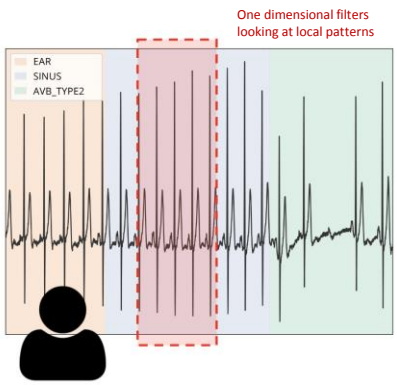
More Applications

1D Convolutional Neural Networks



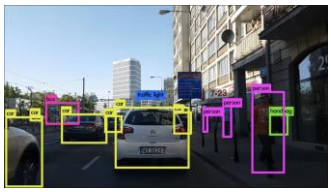
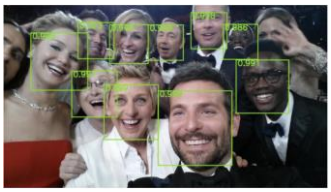
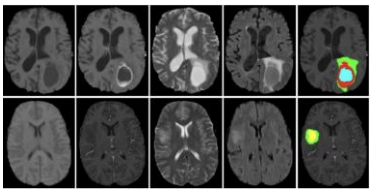
- Heart arrhythmia detection using electrocardiograms (ECG)
  - Trained a deep 34-layer CNN which maps a sequence of ECG to a sequence of rhythm classes
  - Optimization with residual blocks.
  - Achieved cardiologist-level accuracy





Other applications

- Object detection, object localization (e.g. self driving car)
- Face detection, recognition (e.g. unlock phones, prevent crime, school attendance)
- Natural language processing (e.g. key phrase recognition, question-answer matching)
- Medical diagnosis (e.g. diabetic eye disease, fMRI data tumor segmentation)
- Drug discovery (e.g. predicting interaction between molecules and biological proteins)

Packages and Frameworks

PaddlePaddle (Baidu)

Chainer

Caffe (UC Berkeley) → Caffe2 (Facebook)

MXNet (Amazon)

Developed by U Washington, CMU, MIT, Hong Kong U, etc but main framework of choice at AWS

CNTK (Microsoft)

Torch (NYU / Facebook) → PyTorch (Facebook)

Deeplearning4j

Theano (U Montreal) → TensorFlow (Google)

And others...

References

- *Gradient-Based Learning Applied to Document Recognition,* LeCun, Bottou, Bengio, Haffner, IEEE 1998
- *ImageNet Classification with Deep Convolutional Neural Networks,* Krizhevsky, Sutskever, Hinton, NIPS 2012
- *Very Deep Convolutional Networks For Large-Scale Image Recognition,* Simonyan, Zisserman, ICLR 2015
- *Deep Residual Learning for Image Recognition,* He, Zhang, Ren, and Sun, CVPR 2015
- *Good Practices for Deep Feature Fusion,* Shao et al. 2016
- *Squeeze-and-Excitation Networks (SENet),* Hu et al. CVPR 2017
- *Striving for Simplicity: The All Convolutional Net,* Springenberg et al., ICLR Workshop 2015
- *Visualizing and Understanding Convolutional Networks,* Zeiler and Fergus, ECCV 2014
- *Deep Inside Convolutional Networks: Visualizing Image Classification Models and Saliency Maps,* Simonyan, Vedaldi, and Zisserman, ICLR Workshop 2014

Tutorials

- Stanford CS231n: Convolutional Neural Networks for Visual Recognition
- Coursera Deep Learning Specialization
- CMU 11777, Lecture 3: CNN and Optimization

Thank You!
Q & A