DATA SCIENCE

Convolutional Neural Networks

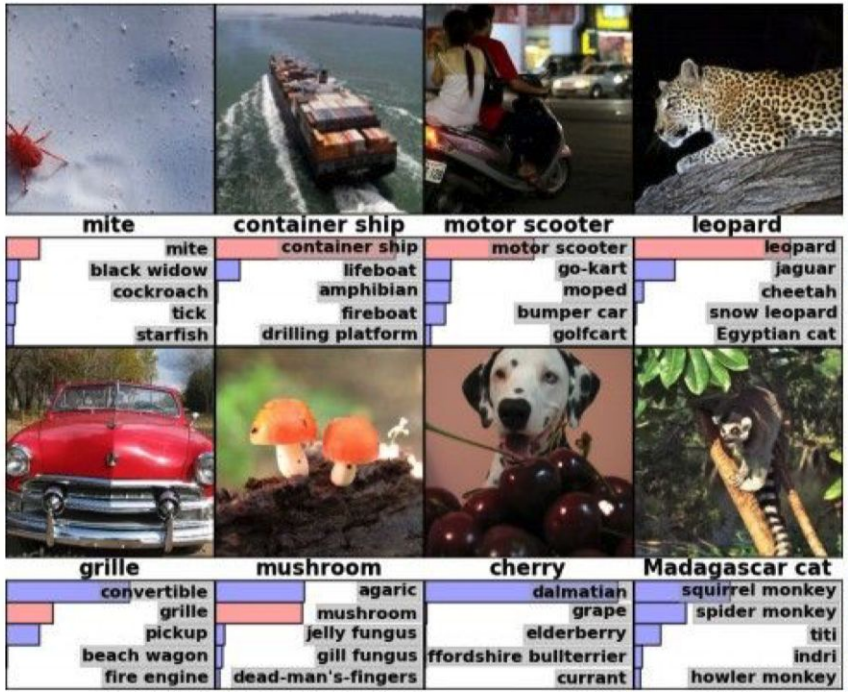
Successful CNN Architectures

N. Rich Nguyen, PhD

SYS 6016

CNNs are everywhere these days

Classification

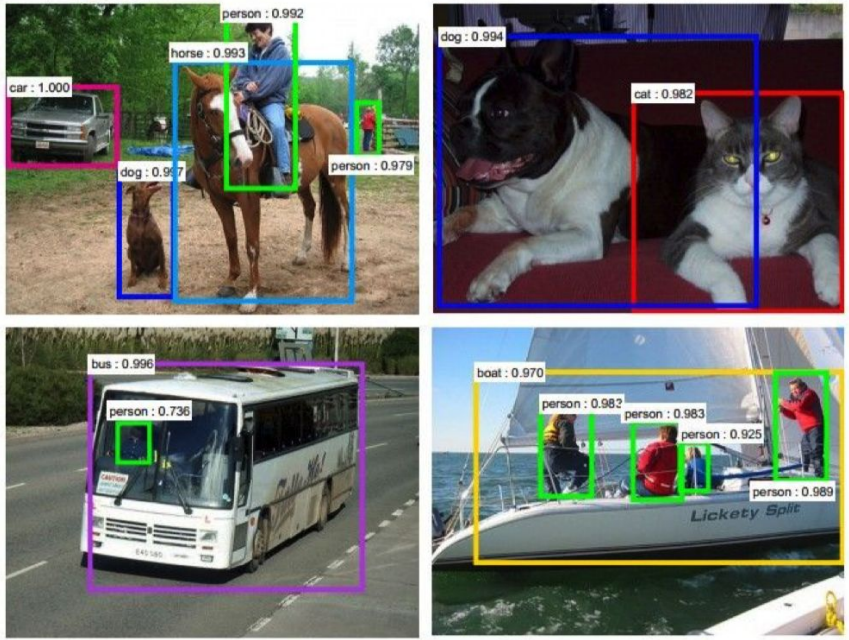


Retrieval

Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

CNNs are everywhere these days

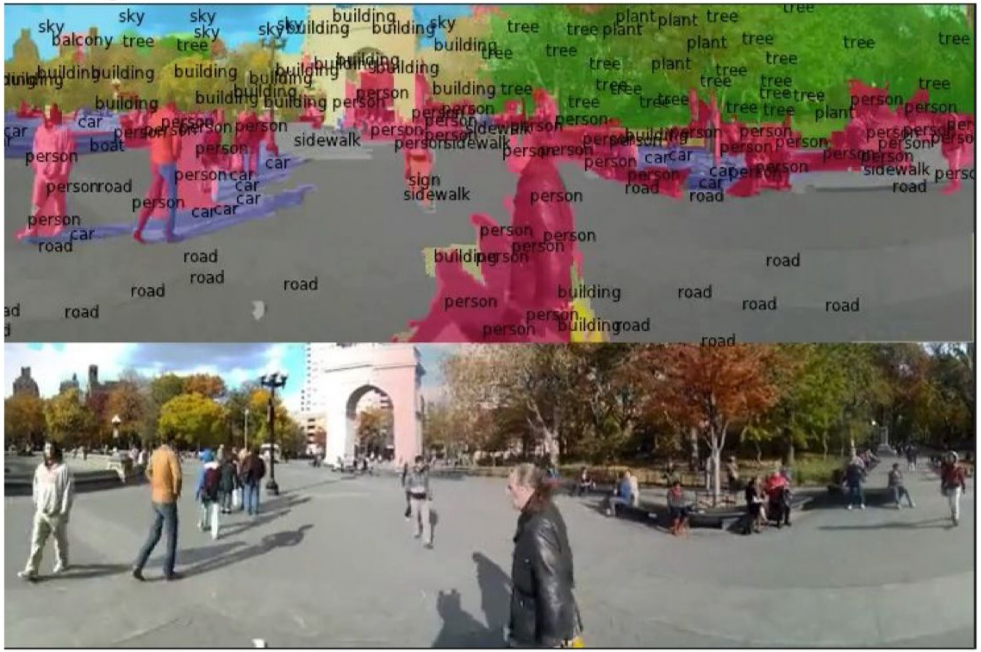
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

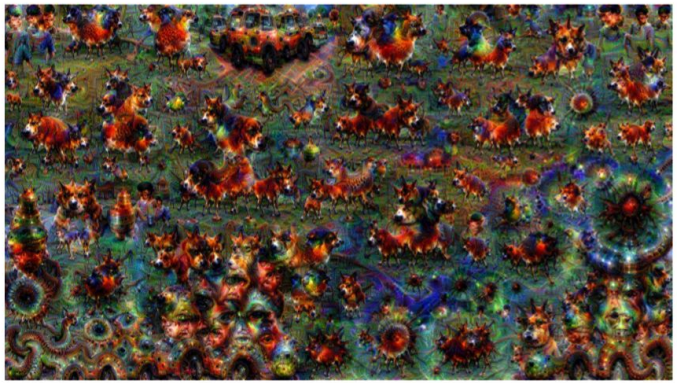
Segmentation

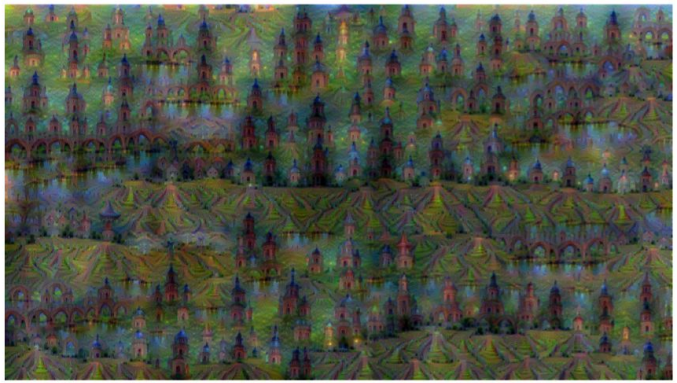


Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

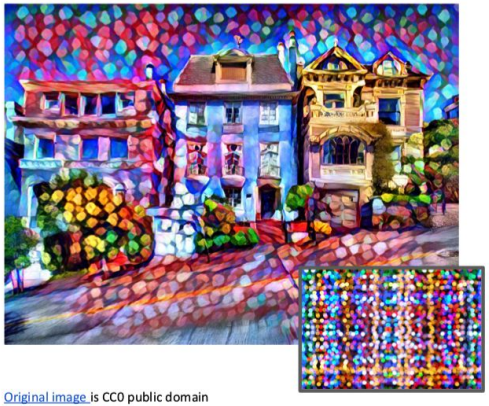
CNNs are everywhere these days

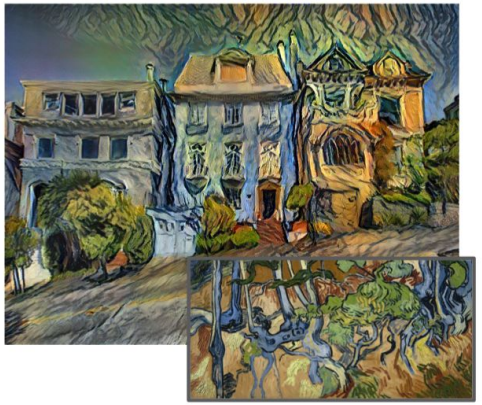




Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.

Style Transfer

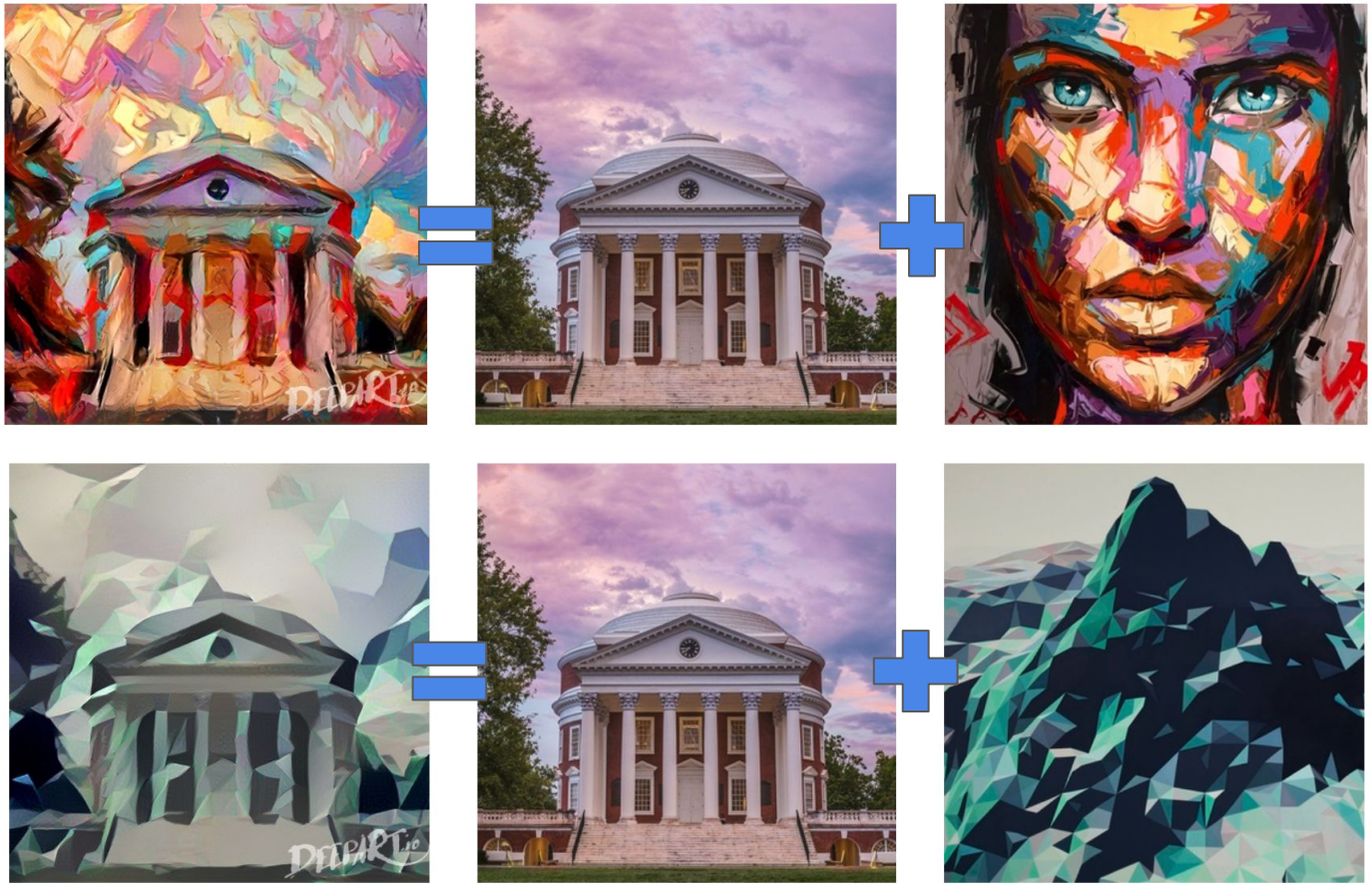




Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;

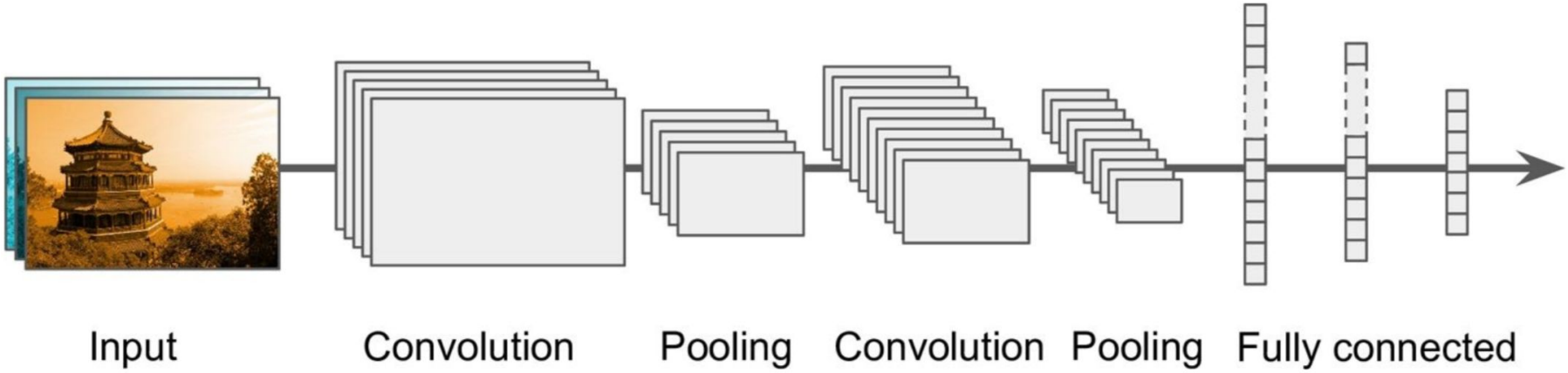
Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Style transfer is applied on UVA Rotunda



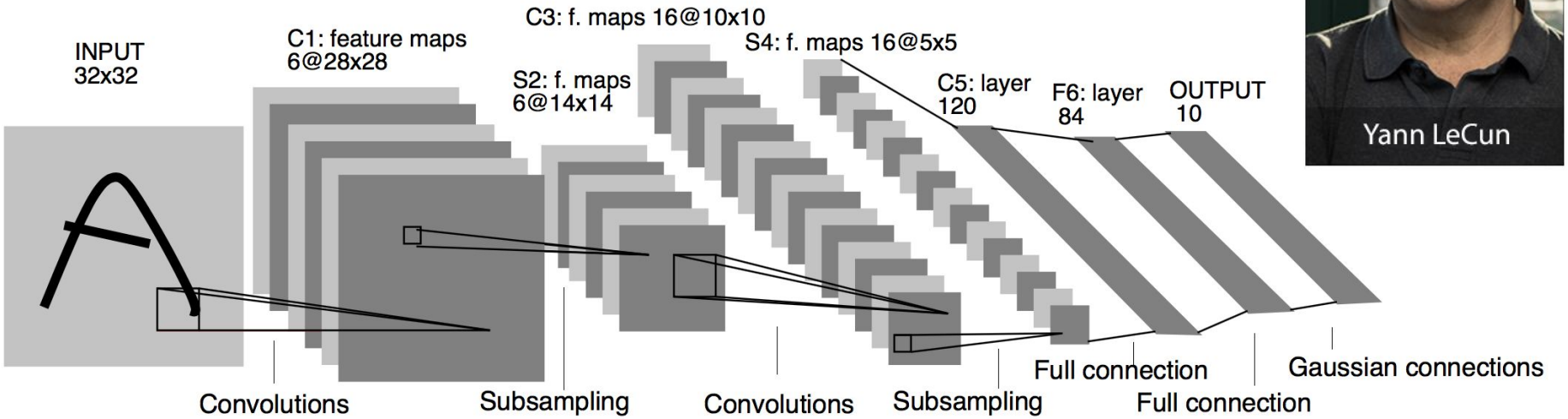
Typical CNN Architectures

Stack a few convolutional layers (each followed by a ReLU layer), then a pooling layer, then another few convolutional layers (+ReLU), then pooling layer, and so on getting deeper and smaller. At the top of the stack is the regular feedforward neural network of fully connected layers (+ReLU), and the final softmax layer.



LeNet-5

Landmark paper: LeCun et al., Gradient-based learning applied to document recognition, 1998



LeNet-5

Most widely known CNN architecture, created by Yann LeCun in 1998

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

Demo on MNIST

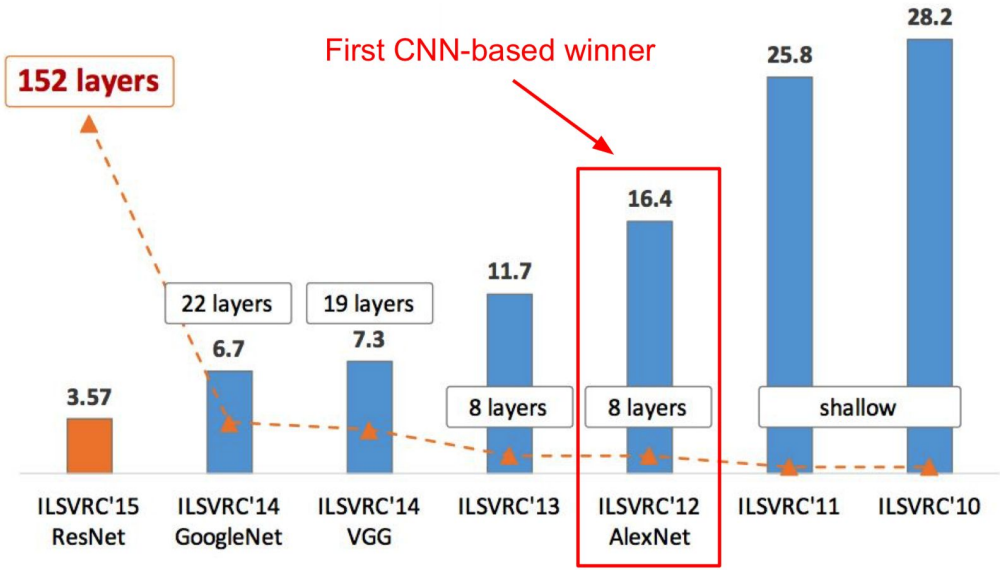
ImageNet Challenge

- Over the years, variants of the CNN architectures have been developed, leading to amazing advances in the field.
- A good measure of this progress is the **ImageNet** Large Scale Visual Recognition Challenge (ILSVRC).
- The ImageNet contain large number of images (14.2 million) of over 1000 classes, some of which are really subtle (ie. try distinguish 120 dog breeds)
- The top-5 error rate is the number of test images for which the system's top 5 predictions did not include the correct answers.

Winners of the ImageNet Challenges

Looking at the evolution of the ImageNet winning entries is a good way to understand how CNNs work.

- AlexNet (2012 winner)
- GoogLeNet (2014 winner)
- ResNet (2015 winner)



AlexNet

Developed by Alex Krizhevsky and **Geoffrey Hinton**

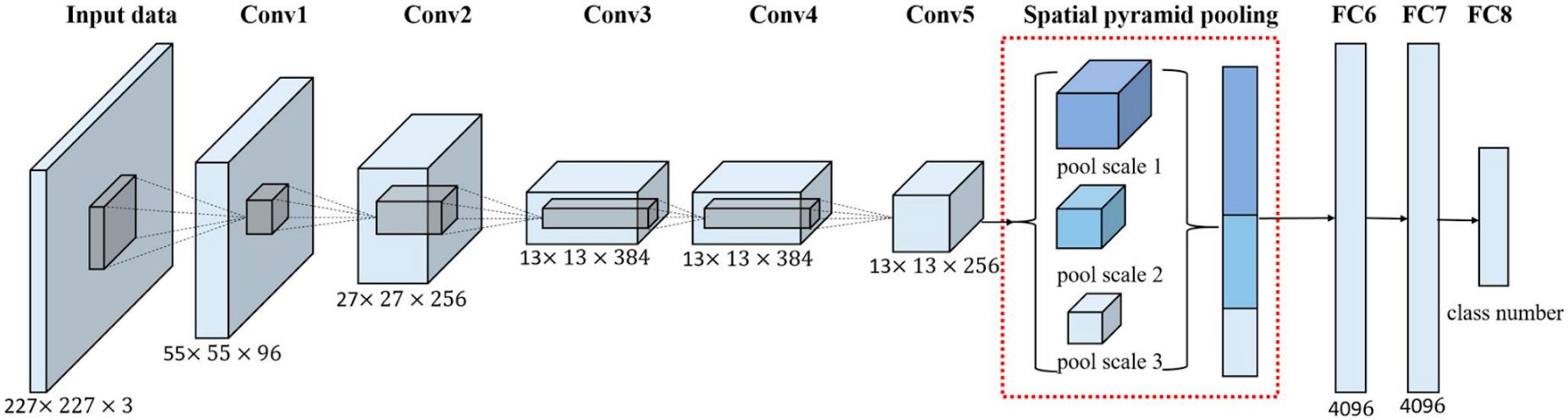
17% top-5 error rate of ImageNet Challenge in 2012

Much larger and deeper then LeNet-5

Stack convolutional layers directly on top of each other

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

AlexNet Architecture



- First use of ReLU
- Heavy data Augmentation
- Dropout at 0.5

- Batchsize 128
- SGD Momentum 0.9
- Learning rate 0.01, reduced by 10 manually when val accuracy plateaus

GoogLeNet

Developed by **Christian Szegedy** and others from Google Research

Winner of the ImageNet challenge with below 7% top-5 error rate

10 times fewer parameters then AlexNet (**6M** instead of 60M)

Much deeper than previous CNNs by introducing **inception (network within a network)** module

Leo's Inception moment at the Oscar

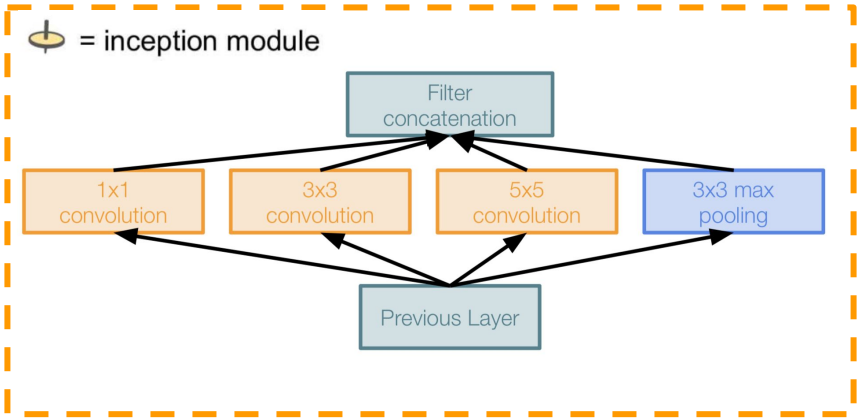
GoogLeNet Inception Module

Design a good local network topology (**network within a network**), stack them on top of each other, and concatenate all filter outputs together depth-wise

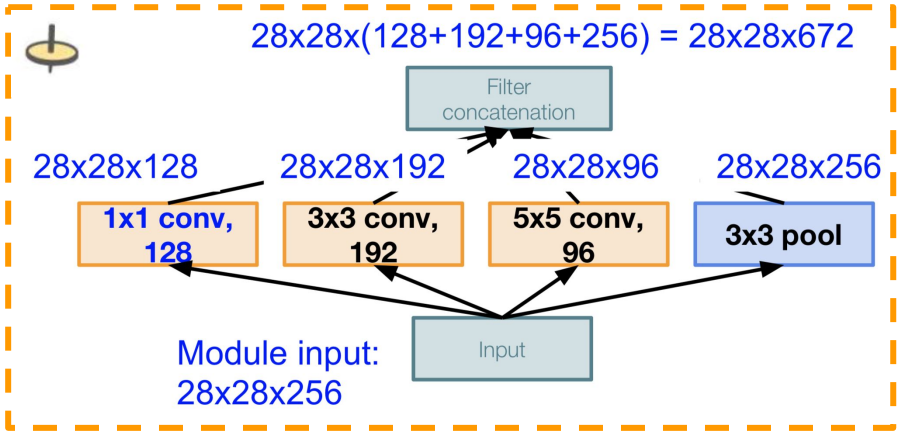
Apply **parallel** operations on the input from previous layer

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depthwise



Naive Inception Computational Complexity



Conv Ops:

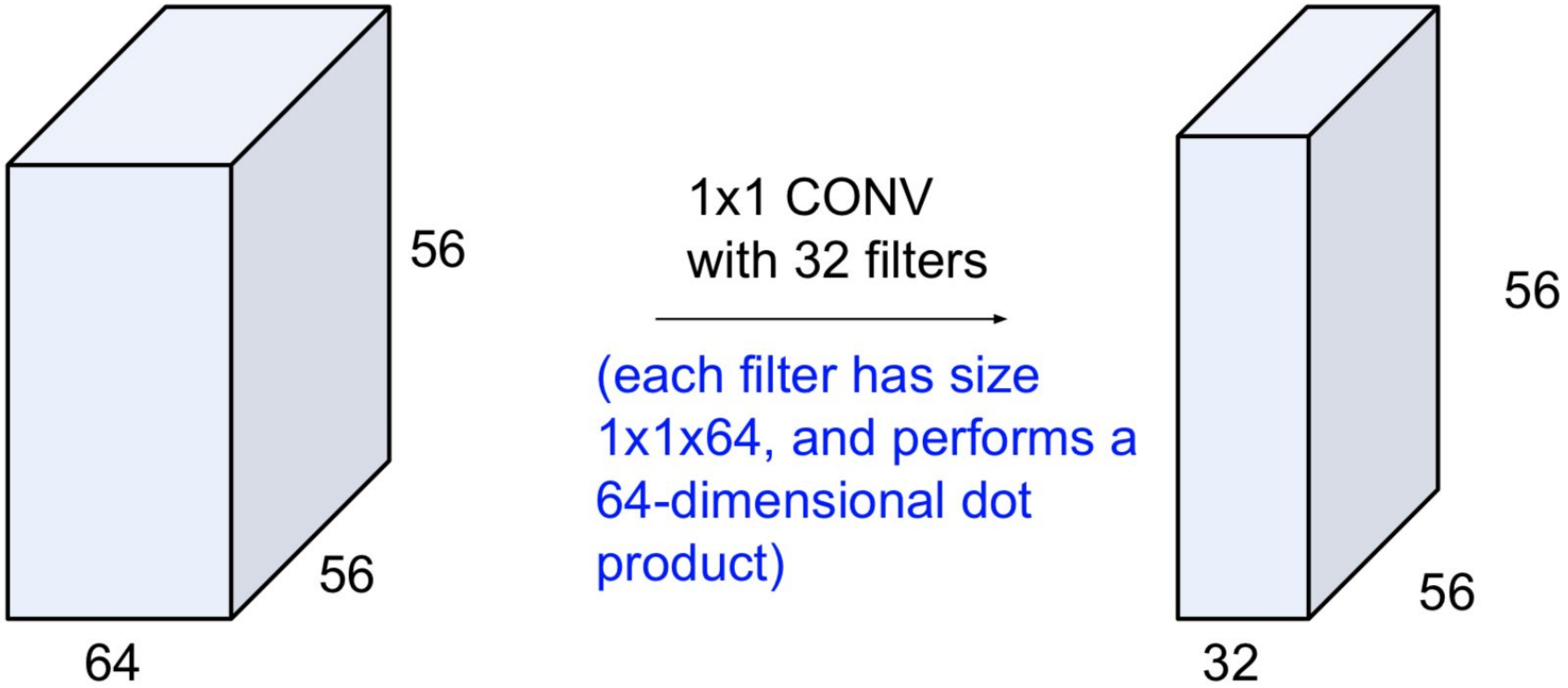
- [1x1 conv, 128] 28x28x128x1x1x256
- [3x3 conv, 192] 28x28x192x3x3x256
- [5x5 conv, 96] 28x28x96x5x5x256
- Total: **854M ops**

Very expensive to compute

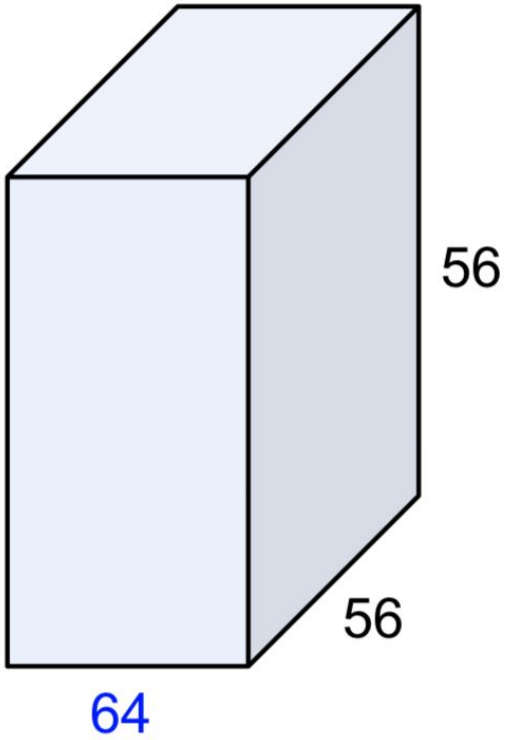
Pooling layer preserves feature depth, which means the total depth after concatenation can only **grow** at every layer!

How to reduce feature depth growing at every layer?

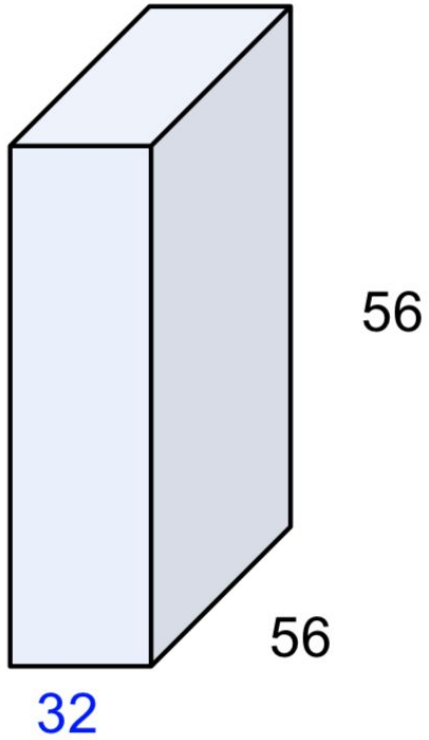
Solution: 1x1 convolution



1x1 convolution



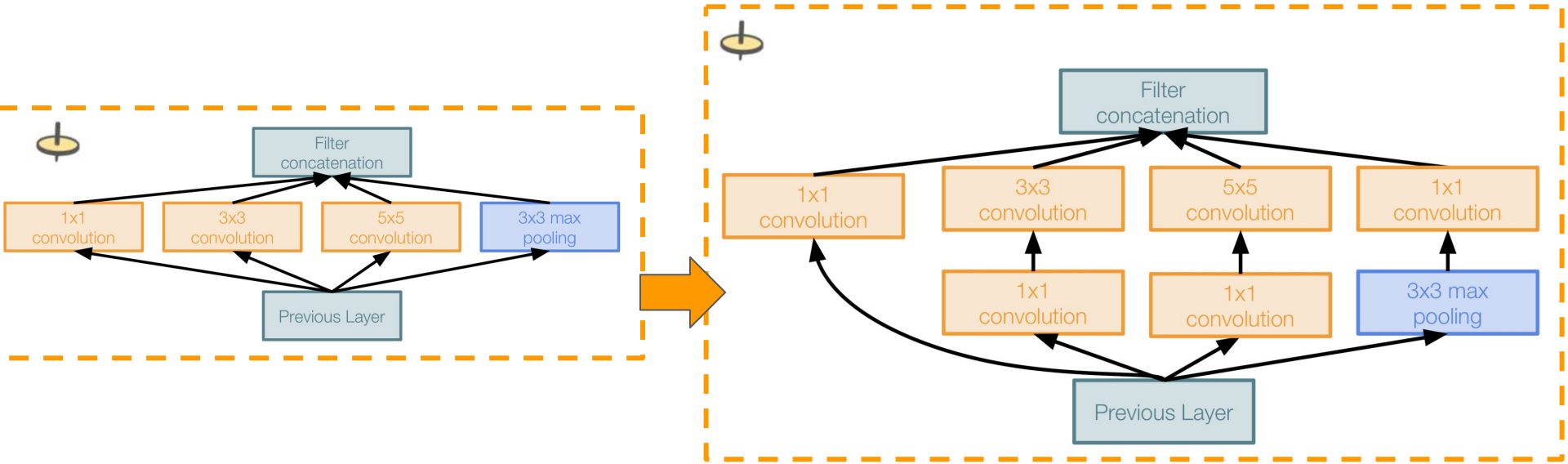
1x1 CONV
with 32 filters



preserves spatial dimensions, reduces depth!

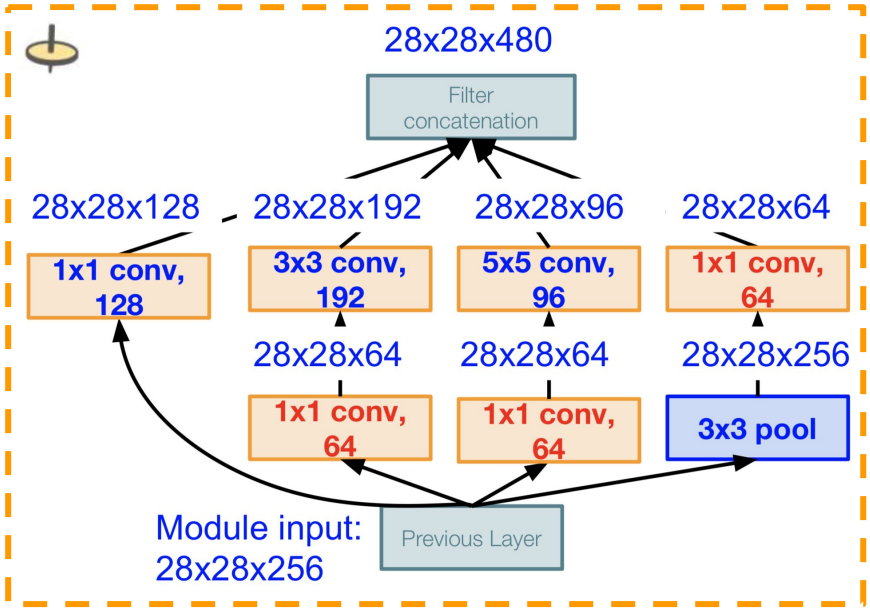
Projects depth to lower dimension (combination of feature maps)

Inception Modele with Dimensionality Reduction



Solution: Use **1x1 convolutions** to reduce feature depth

Inception Module Computational Complexity



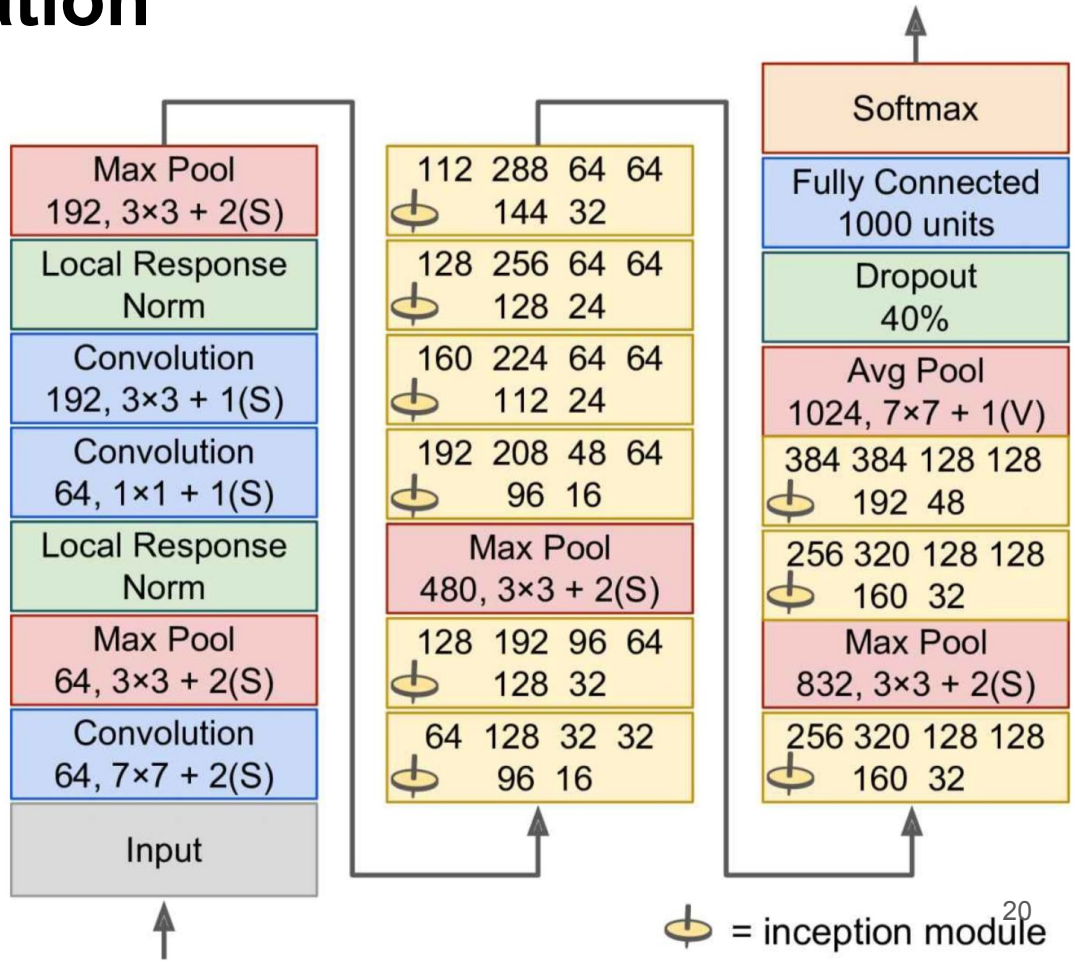
Conv Ops:

- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 128] 28x28x128x1x1x256
- [3x3 conv, 192] 28x28x192x3x3x64
- [5x5 conv, 96] 28x28x96x5x5x64
- [1x1 conv, 64] 28x28x64x1x1x256
- Total: **358M ops**

Comparing to 854M ops for naive version.

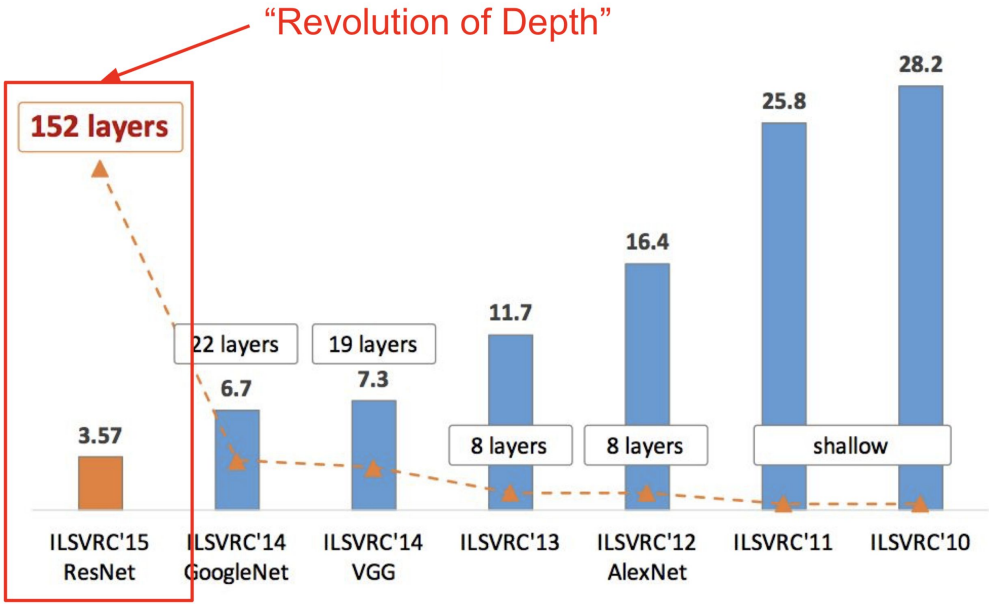
GoogLeNet Configuration

- Deeper network (22 layers)
- Stack efficient Inception modules on top of each other.
- No FC layers (except the output)
- 10x less params than AlexNet
- Some variants were proposed after including Inception-v3 and Inception-v4.
- Inspired the development of Xception (2017)

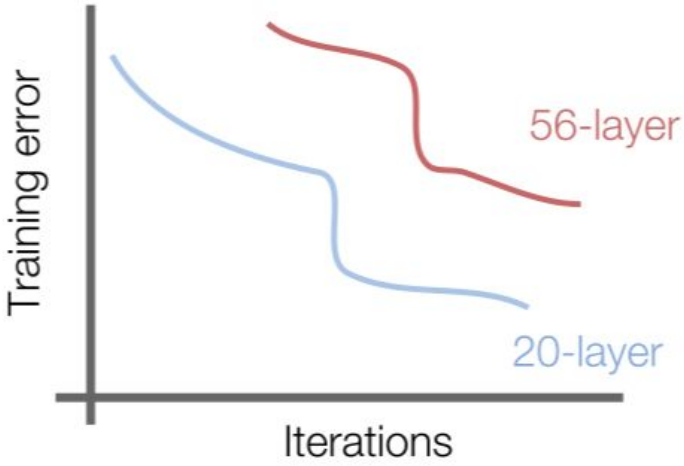


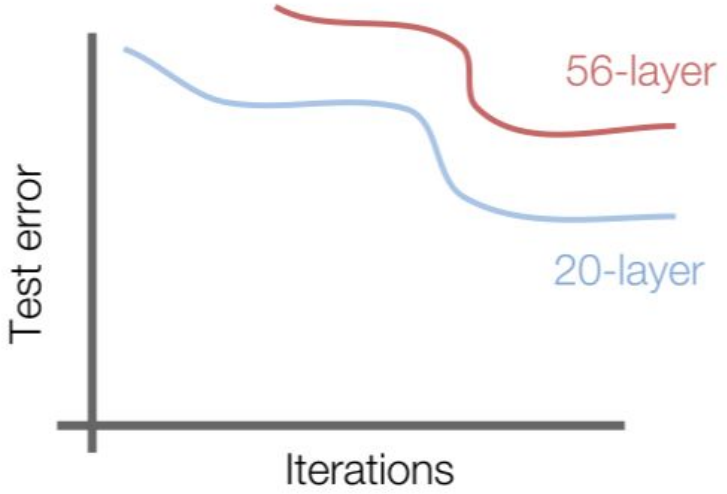
ResNet

- Developed by Kaiming He et al., extremely deep CNN with **152 layers**
- Winner of ImageNet Challenge in 2015 with **3.6%** top-5 error rate
- **2-3 weeks** of training time on a 8 GPU machine
- Does more layers means better?



More layers equals better?





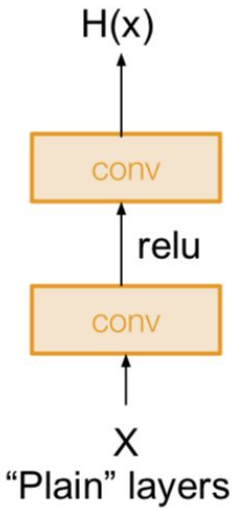
He *et al* 2015 shows that 56-layer model performs worse that 20-layer model.

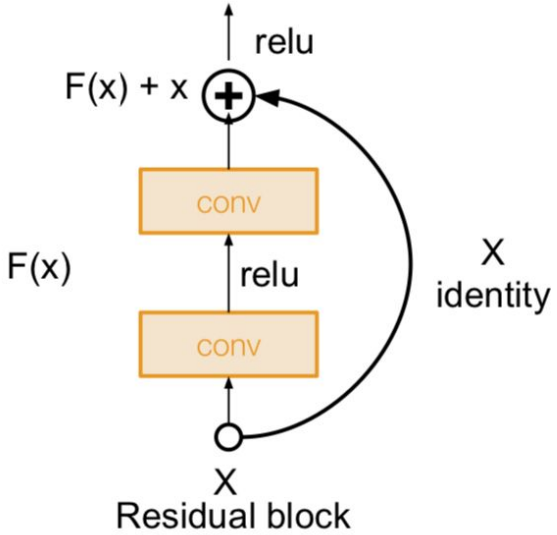
Does that cause by overfitting?

Not really, but it is the optimization -- deeper network is harder to optimize!

ResNet

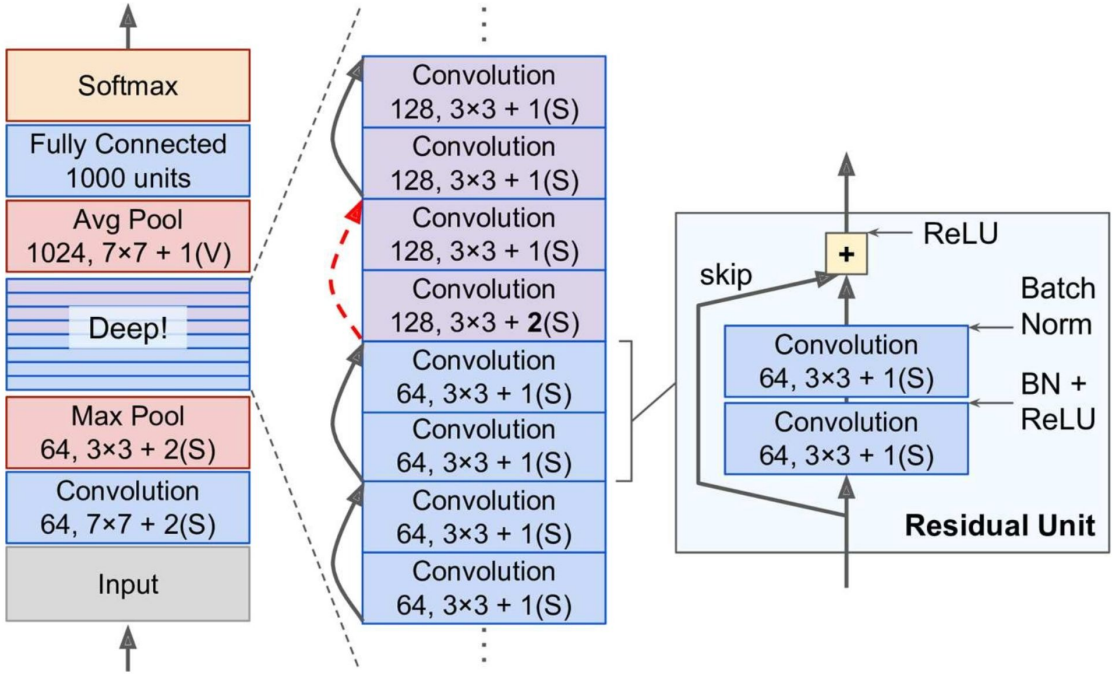
- The deeper model should be able to perform at least as well as the shallow one.
- Usage of **skip (shortcut)** connections: Copy a solution from the shallower model and setting additional layers to identity mapping.
- Network will be forced to model F(x) = H(x) -x rather than H(x) → **residual learning**



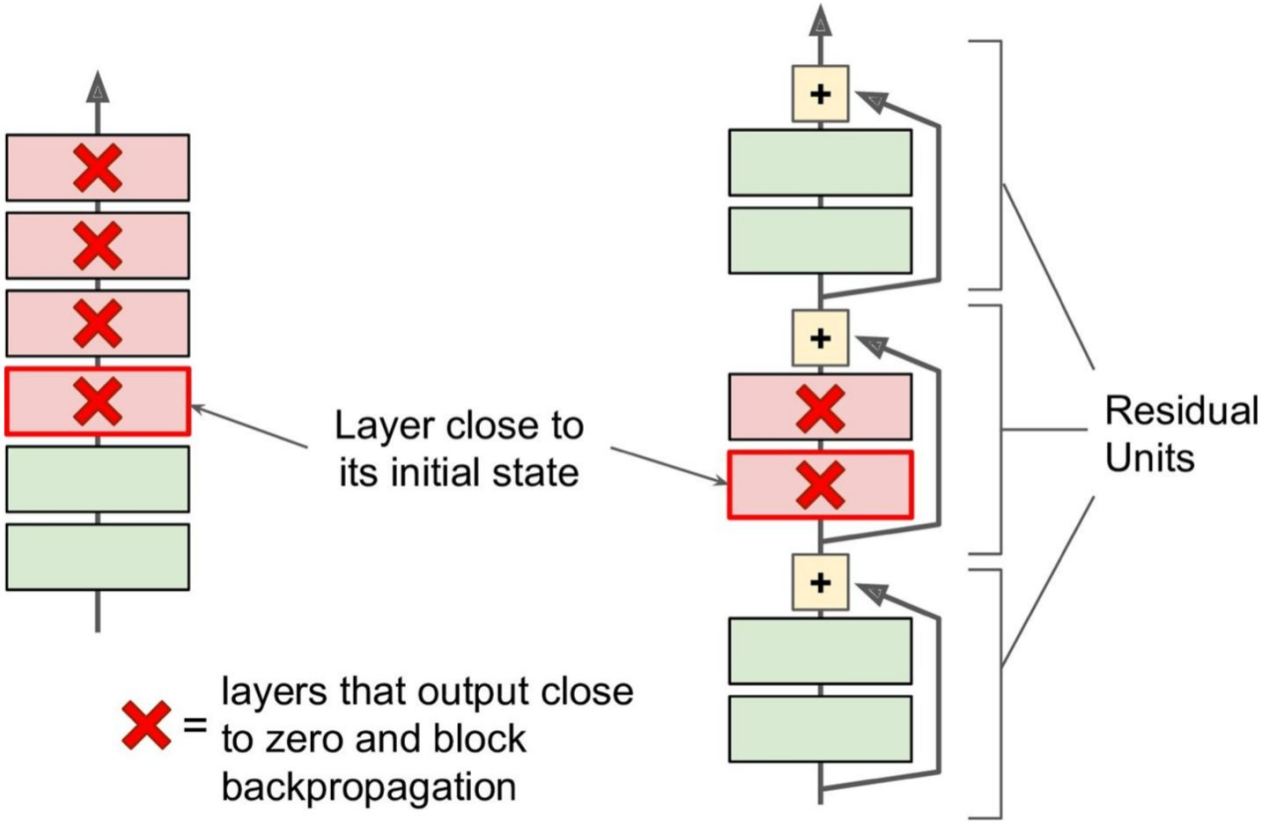


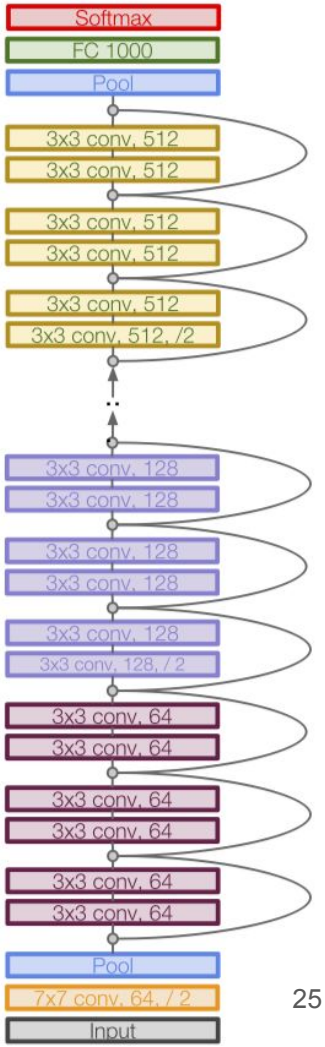
ResNet Configuration

- Batch Normalization after every CONV layer
- He Initialization
- SGD + Momentum (0.9)
- Learning rate (0.1), divided by 10 when validation error plateaus
- Mini-batch size 256
- No dropout used

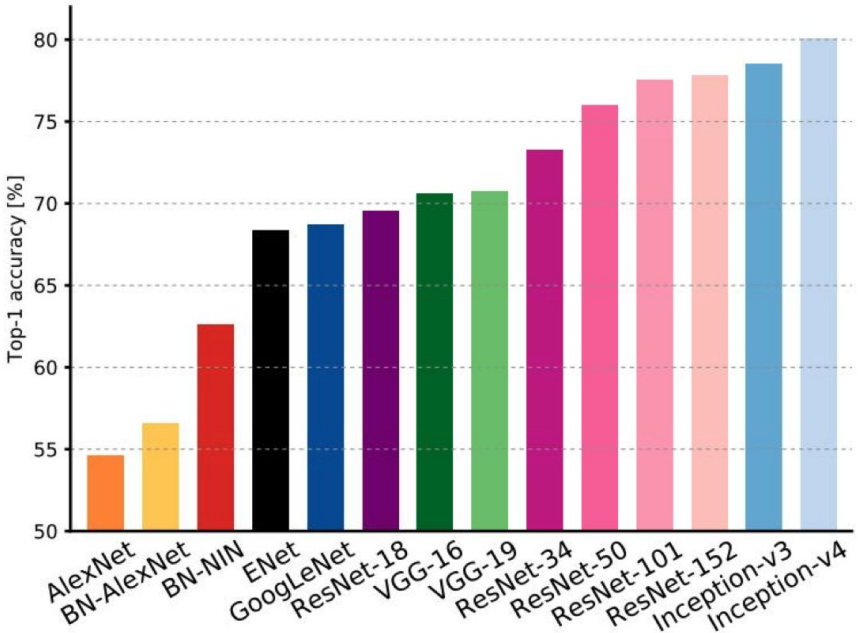


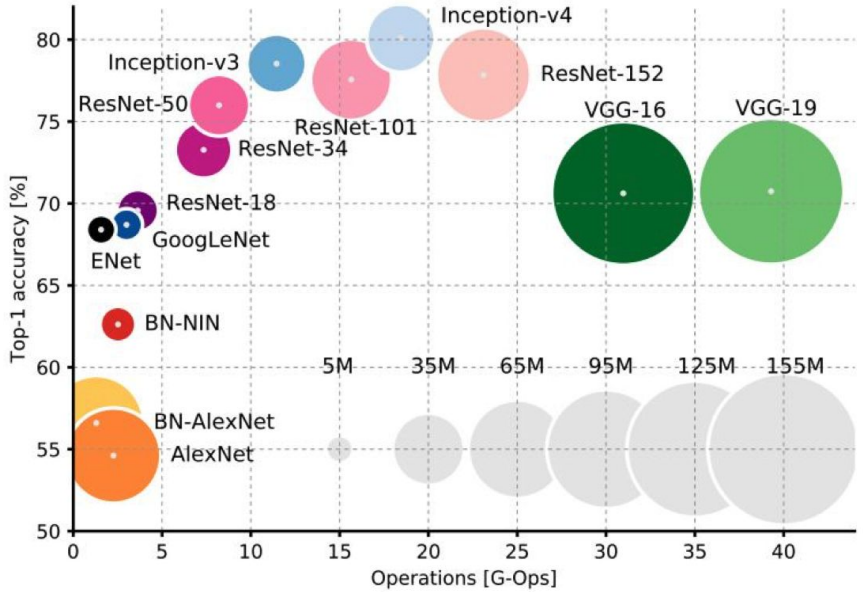
Regular DNN vs. ResNet





Complexity vs. Accuracy

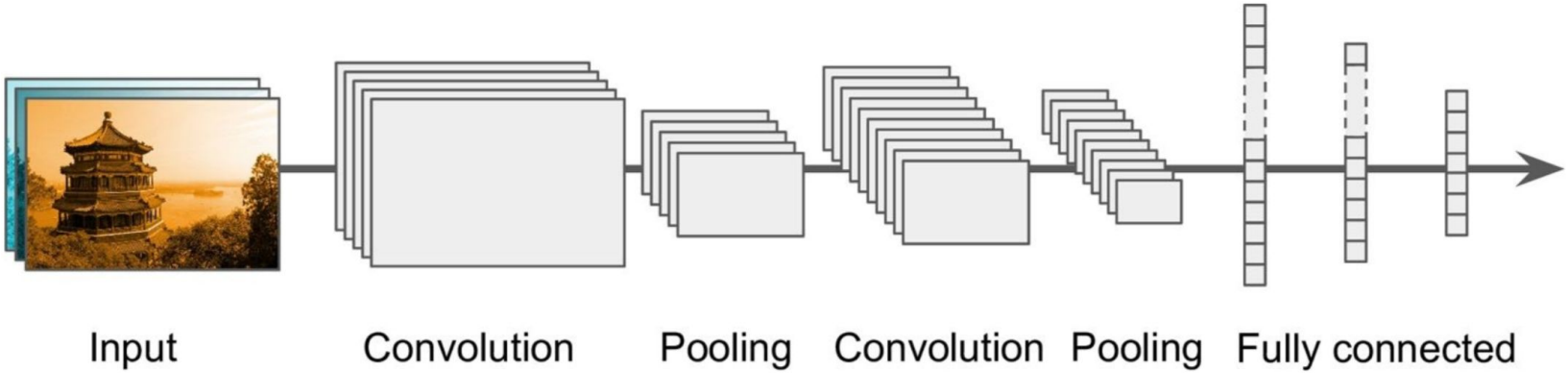




An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Summary

- The field is moving rapidly, with all sort of architectures popping out every year
- AlexNet, GoogLeNet, ResNet and their variations are all widely used
- One clear trend that CNNs keep getting deeper and lighter, requiring less parameters (ResNet is both simplest and most powerful)
- 2016 ImageNet winner, GBD-Net, uses ensemble learning to train combinations of the previous models to achieve < 3% top-5 error



Bonus Slides

CNNs are everywhere these days

Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Style transfer is applied on my photo

Xception

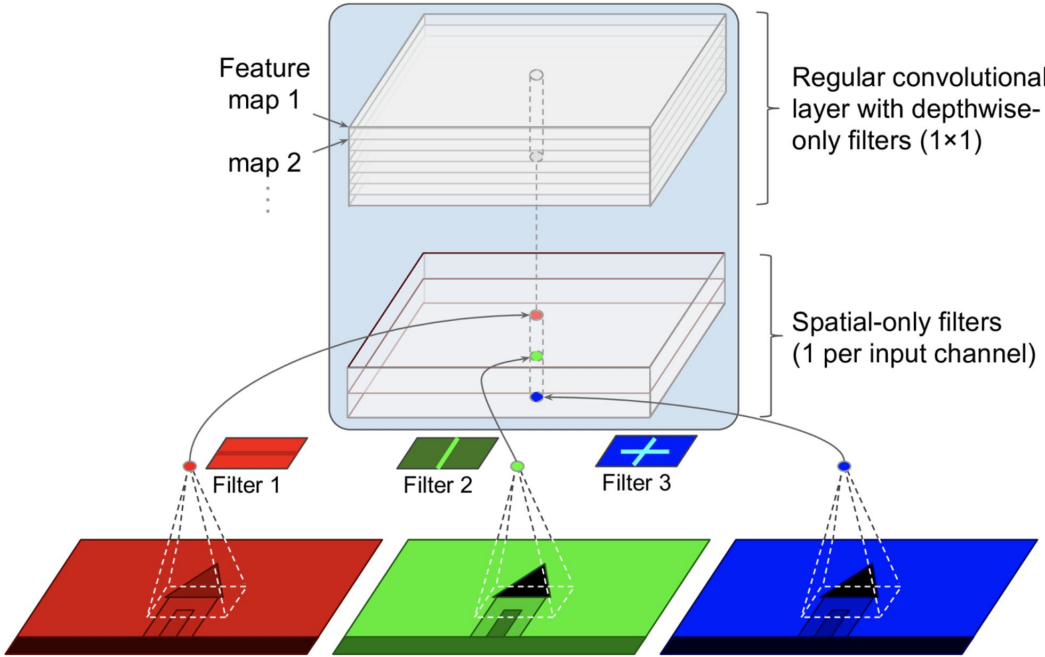
Another variant of GoogLeNet architecture is Xception (Extreme Inception)

Proposed in 2016 by Francois Chollet (the author of Keras)

Merges the ideas of **GoogLeNet** and **ResNet**, but replaces the inception modules with “depthwise separable convolution layer”

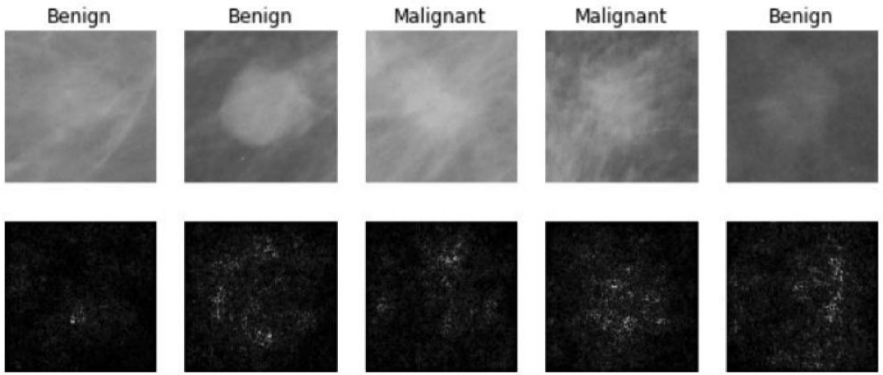
Make a strong assumption that spatial patterns and cross-channel patterns can be modeled separately.

Separate convolutional layers



It uses fewer parameters, less memory, and few computations than regular conv layer, and it performs better in general.

CNNs are everywhere these days



[Levy et al. 2016]

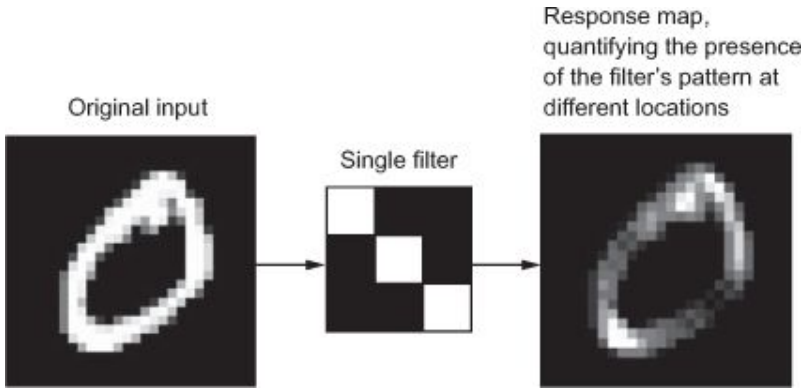
[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

[Sermanet et al. 2011]
[Ciresan et al.]

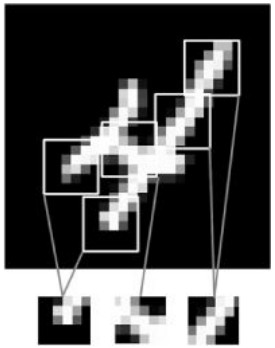
Photos by Lane McIntosh.
Copyright CS231n 2017.

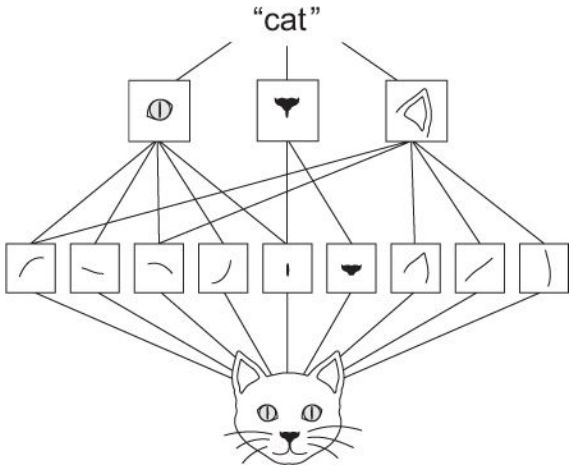
Image kernel/filter



- A neuron's weights can be represented as a small image the size of the receptive field. These weights are called **filters** (or convolutional kernels).
- A convolution works by sliding this filter (size 3x3) over the input image, stopping at every possible location, then operates a dot product between a filter weight with its correspond location on the input image.

Convolution Layers learn local patterns





This characteristic give CNNs two interesting properties:

- After learning a certain local pattern, a CNN can recognize it anywhere (**equivariance to translation**).
- They can learn **spatial hierarchies** of patterns: 1st conv layer will learn small local patterns (edges), 2nd conv layer will learn larger patterns (eyes or ears) made of the features from the 1st layer, and so on.

Hyperparameters and Memory Requirements

- Unfortunately, convolutional layers have **lots of hyperparameters**: number of kernels, their height and width, strides, and padding types → finding the right set of parameter can be very time-consuming.
- Convolutional layers also requires a **huge amount of memory** during training because the backward pass requires all intermediate values computed.
  - Consider a layer with 5x5 kernels, outputting 200 feature maps of size 150 x 100 (which is 15,000 neurons), with stride 1 and SAME padding, each neuron computes the weighted sum of its 5x5x3 = 75 inputs. That's a total of **225 million** float multiplications.
  - Moreover, feature maps will occupy 200x150x100x32=96 millions bits (11.4 MB) of memory for each training instance. So for a batch of 100 instance, that's **1 GB**!