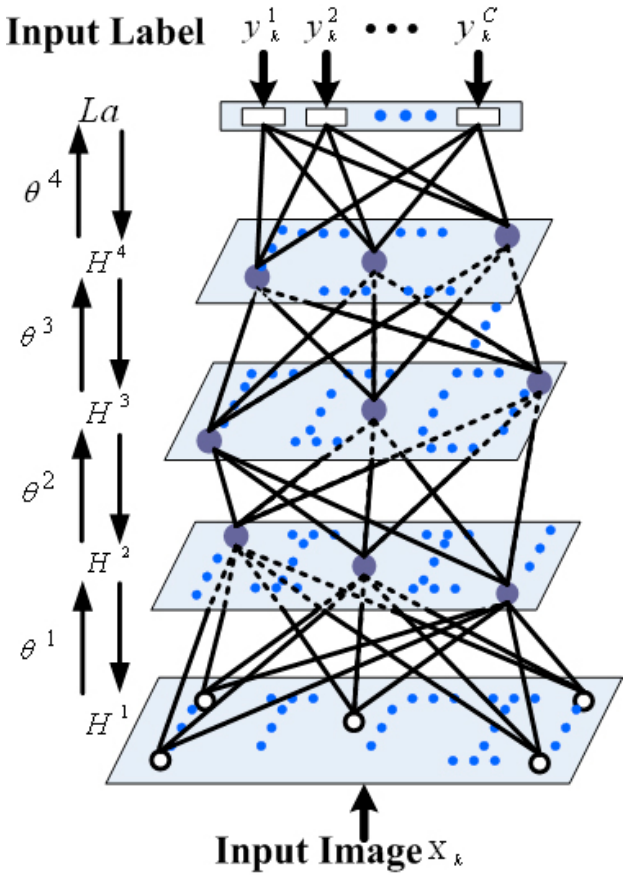
Topic 12: DEEP LEARNING

STAT 37710/CMSC 35400 Machine Learning
Risi Kondor, The University of Chicago

(some slides by Shubhendu Trivedi)

Deep neural networks



- Starting in 2006, Hinton, LeCun, Bengio and others introduced a sequence of tricks to make training deep neural nets possible.

Convolutional Neural Nets

Convolution and correlation

The **convolution** of $f : \mathbb{R} \to \mathbb{R}$ with $g : \mathbb{R} \to \mathbb{R}$ is

$$(f * g)(x) = \int f(x - y)\, g(y)\, dy.$$

The **cross-correlation** of $f : \mathbb{R} \to \mathbb{R}$ with $g : \mathbb{R} \to \mathbb{R}$ is

$$(f \circ g)(x) = \int f(x + y)\, g(y)\, dy.$$

Note that $f \circ g = f * g^-$ where $g^-(x) = g(-x)$. When neural nets people talk about convolution they usually really mean cross-correlation, but never mind.

Convolution theorem

Recall that the Fourier transform of f is

$$\widehat{f}(k) = \int f(x)\, e^{-2\pi i\, k\, x}\, dx.$$

The **convolution theorem** states that

$$\widehat{f * g}(k) = \widehat{f}(k)\, \widehat{g}(k).$$

Discrete convolution in 2D

If $f, g\colon \mathbb{Z}^2 \to \mathbb{R}$, then

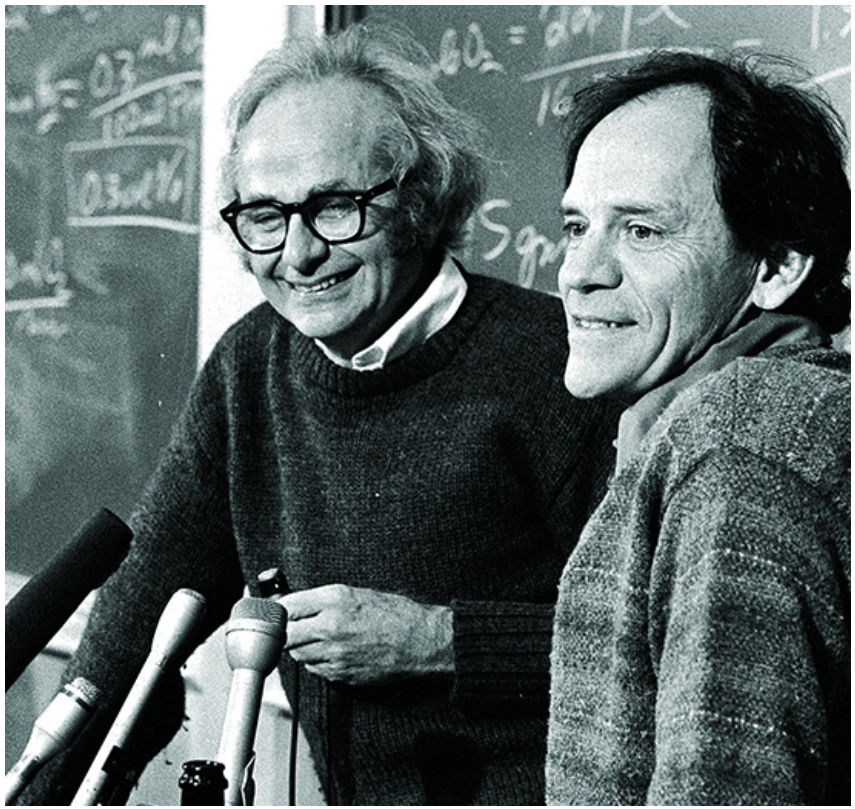
$$(f * g)(i,j) = \sum_a \sum_b f(i-a, j-b)\, g(a,b),$$

just like applying a filter at every (i,j) position on the grid. Moreover (if we turn $\mathbb{Z}^2$ into a torus $\mathbb{Z}_N \times \mathbb{Z}_N$), we have a 2D discrete Fourier transform (DFT)

$$\widehat{f}(\boldsymbol{k}) = \sum_{\mathbf{x} \in \{0,1,\ldots,N-1\}^2} f(\mathbf{x})\, e^{-2\pi i(\mathbf{k}\cdot\mathbf{x})}$$

that can be computed in $O(N^2 \log N)$ time.

Hubel-Wiesel Experiments, 1959



- Recorded activity of individual neurons in cats in response to images projected in different positions on a screen
- Neurons in the cat's early visual system responded strongly to specific patterns of light, such as oriented bars and almost not at all to other patterns.
- Neurons in the later visual system responded to more complex stimuli and responses also exhibited invariance to translations etc.

A Simplified View of Brain Function

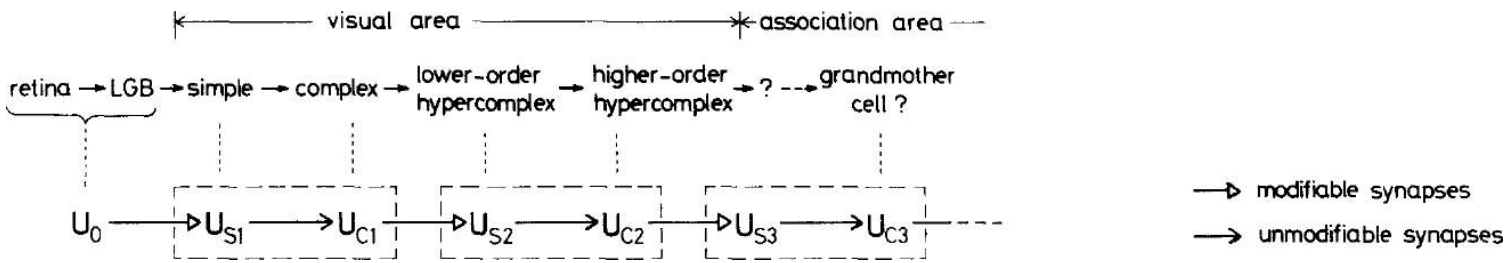


Image Source: Fukushima, 1980

- Images are projected onto the retina, neurons in retina do some simple preprocessing but do not substantially alter the representation.
- The signal channels into the LGN area through the optic nerve and then into V1.

A Simplified View of Brain Function

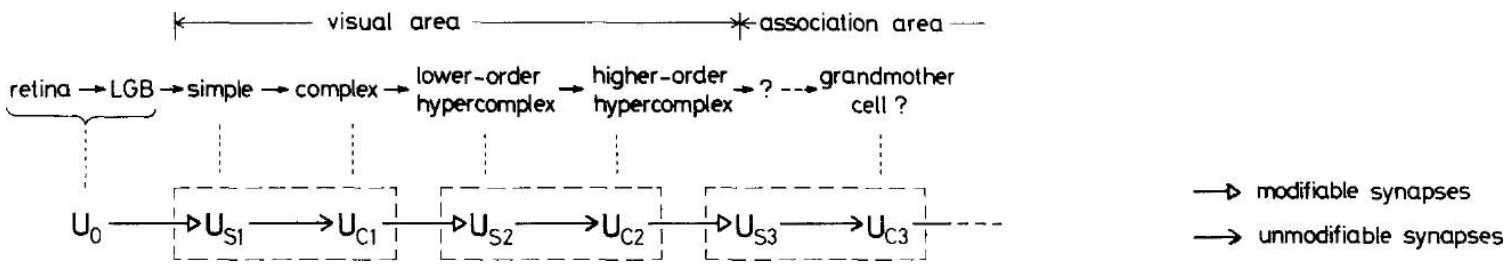


Image Source: Fukushima, 1980

- **V1 is arranged in a spatial map:**
  - 2D structure that mirrors structure of image in the retina
  - Light incident in the lower half of the retina only affects the lower half of V1

A Simplified View of Brain Function

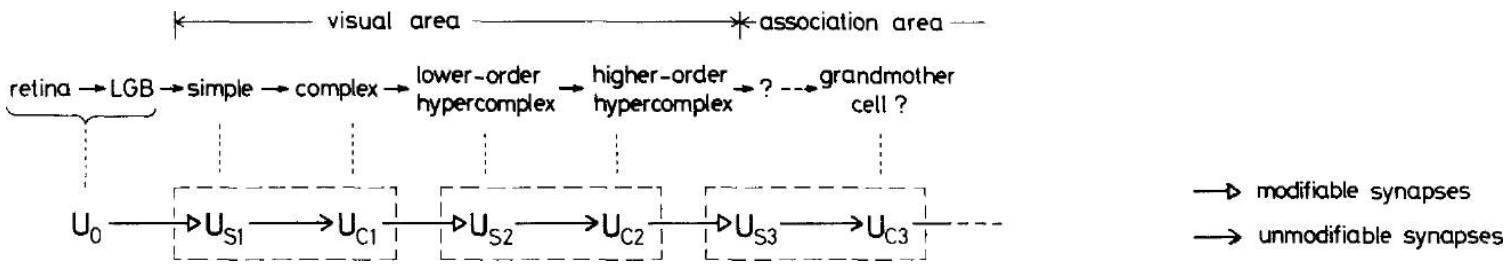


Image Source: Fukushima, 1980

- **V1 has many simple cells:** Roughly characterized by a linear function of image in small, spatially localized receptive fields (detection)
- **V1 has many complex cells:** Features detected similar to simple cells, but invariant to small shifts in position of feature (pooling)
- Also invariant to some changes in lighting

A Simplified View of Brain Function

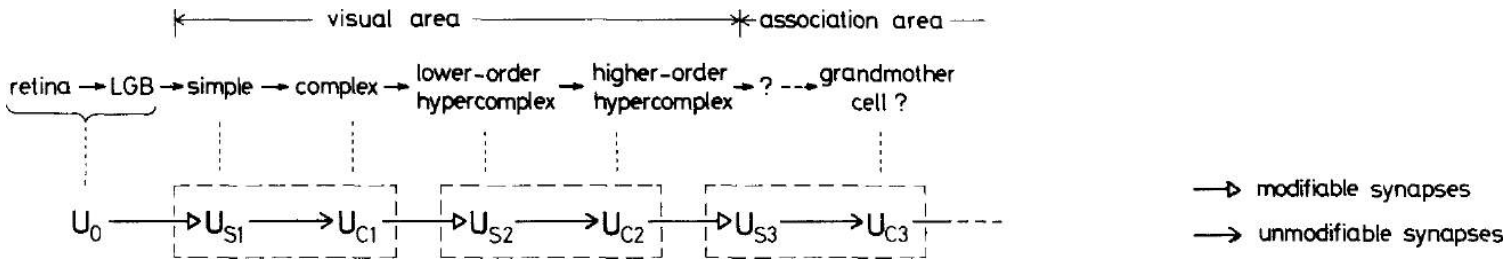


Image Source: Fukushima, 1980

- In the simplified view, this basic strategy is repeated many times
- After multiple layers, we find cells that respond to only specific concepts and are invariant to many transformations of the input (grandmother cells in the medial temporal lobe)

Simple and Complex Cells

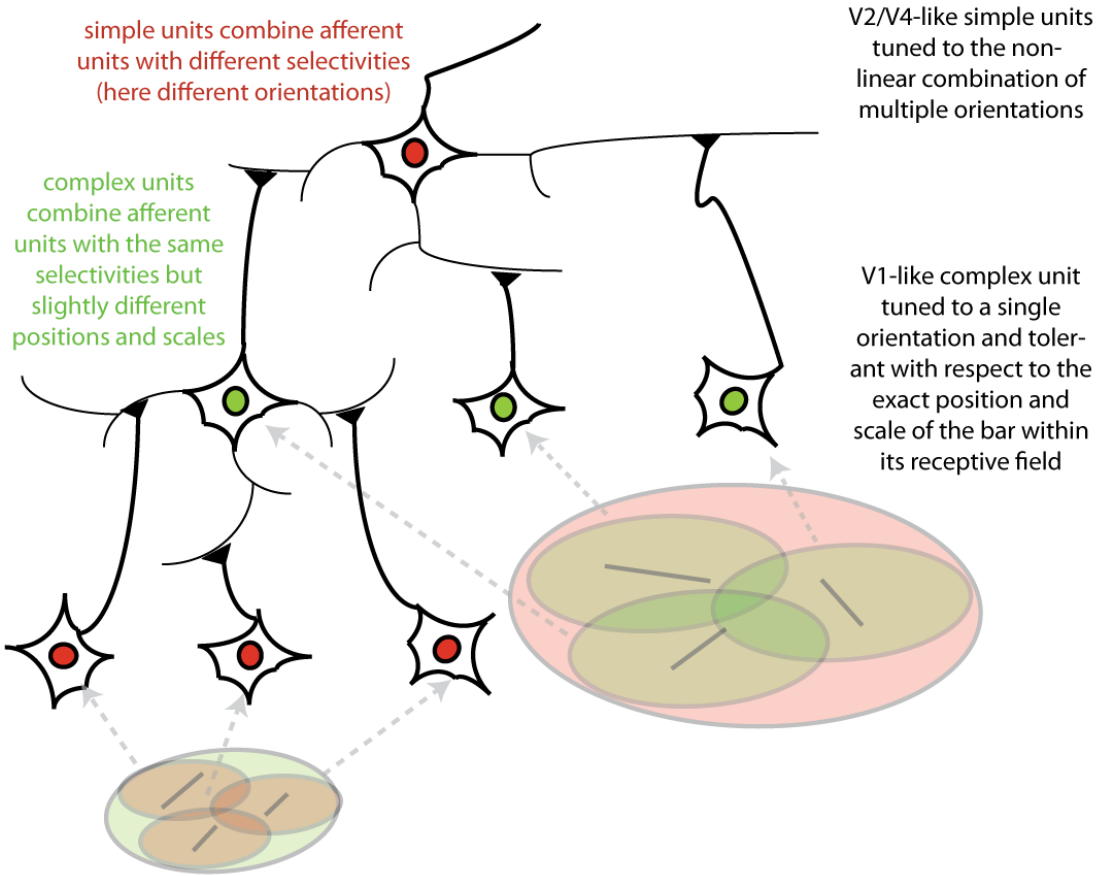
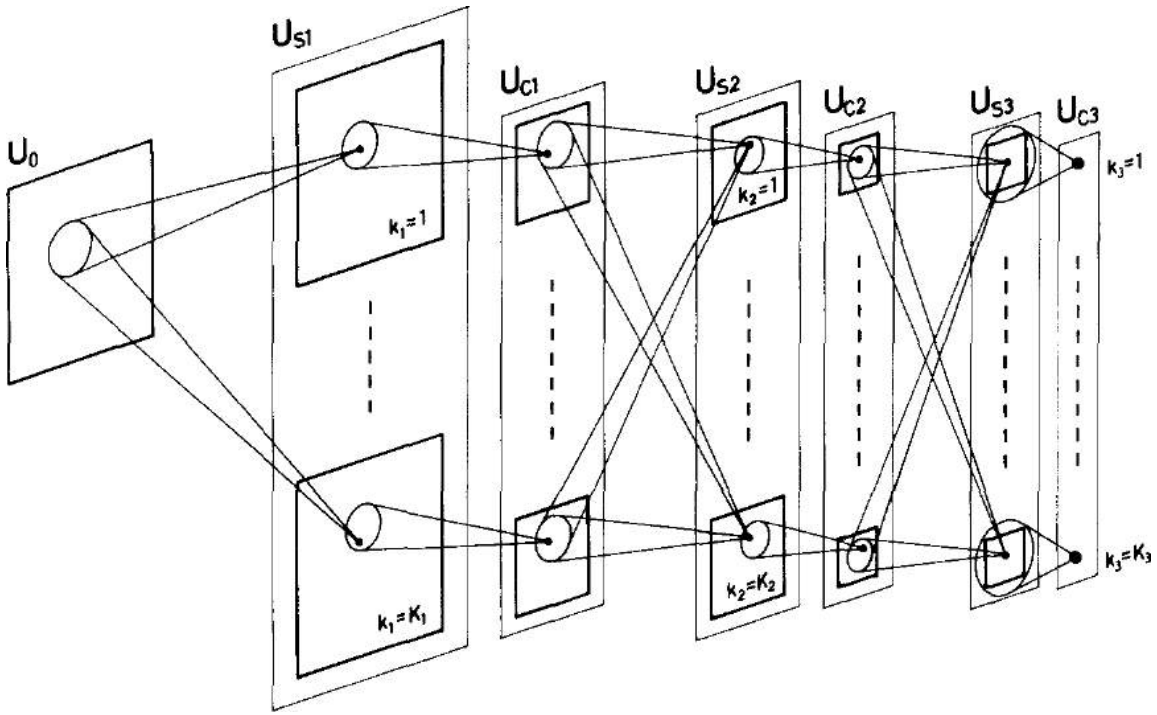


Image Source: Scholarpedia

Neocognitron (Fukushima, 1980)

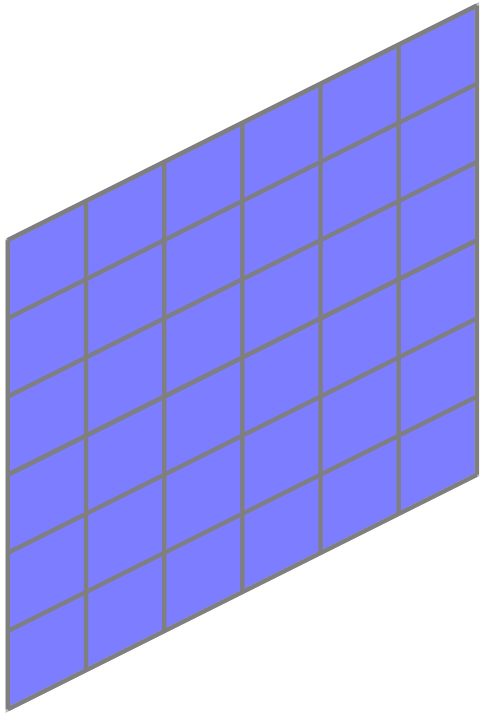


- Fukushima used this simplified view of brain function to build a neural network which had detector units and pooling units one after the other
- Was trained by an unsupervised procedure

TDNNs and CNNs

- Waibel and Hinton introduced a 1-D Convolutional Network and trained it by backpropagation
- Convolutional Networks topology was directly inspired by the Neocognitron which was directly inspired by the Hubel-Weisel model
- TDNNs inspired the use of backpropagation for training for 2D CNNs (Yann LeCun, 1989)

Convolution

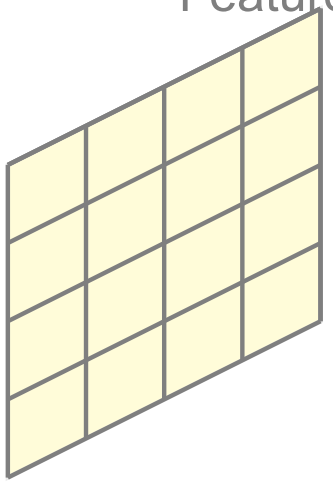


Grayscale Image

Kernel

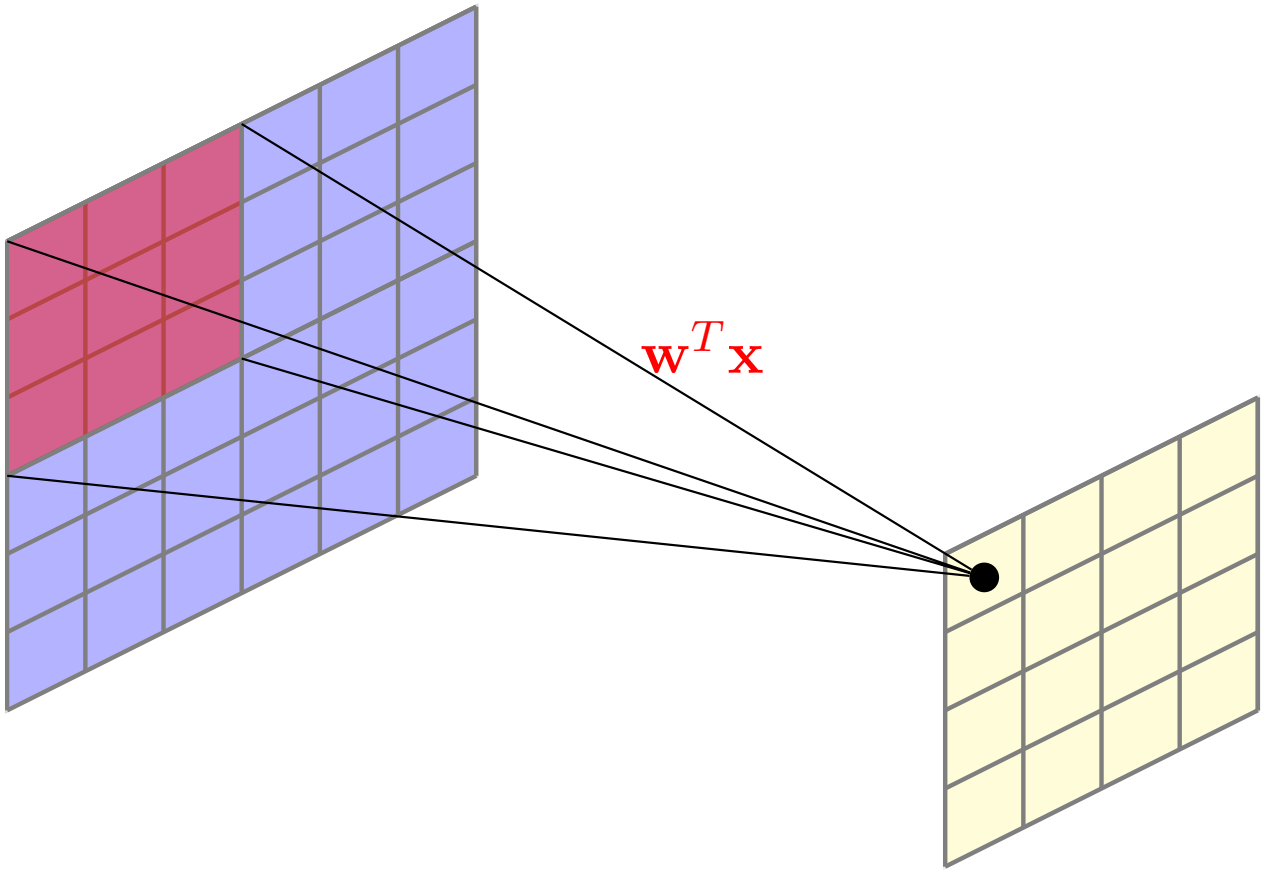
w_7	w_8	w_9
w_4	w_5	w_6
w_1	w_2	w_3

Feature Map

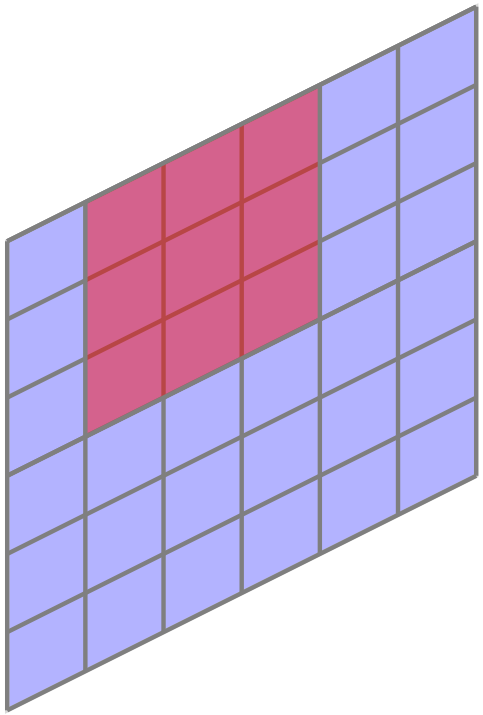


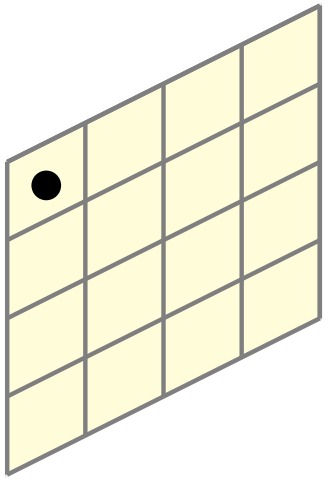
- Convolve image with kernel having weights $\mathbf{w}$ (learned by backpropagation)

Convolution

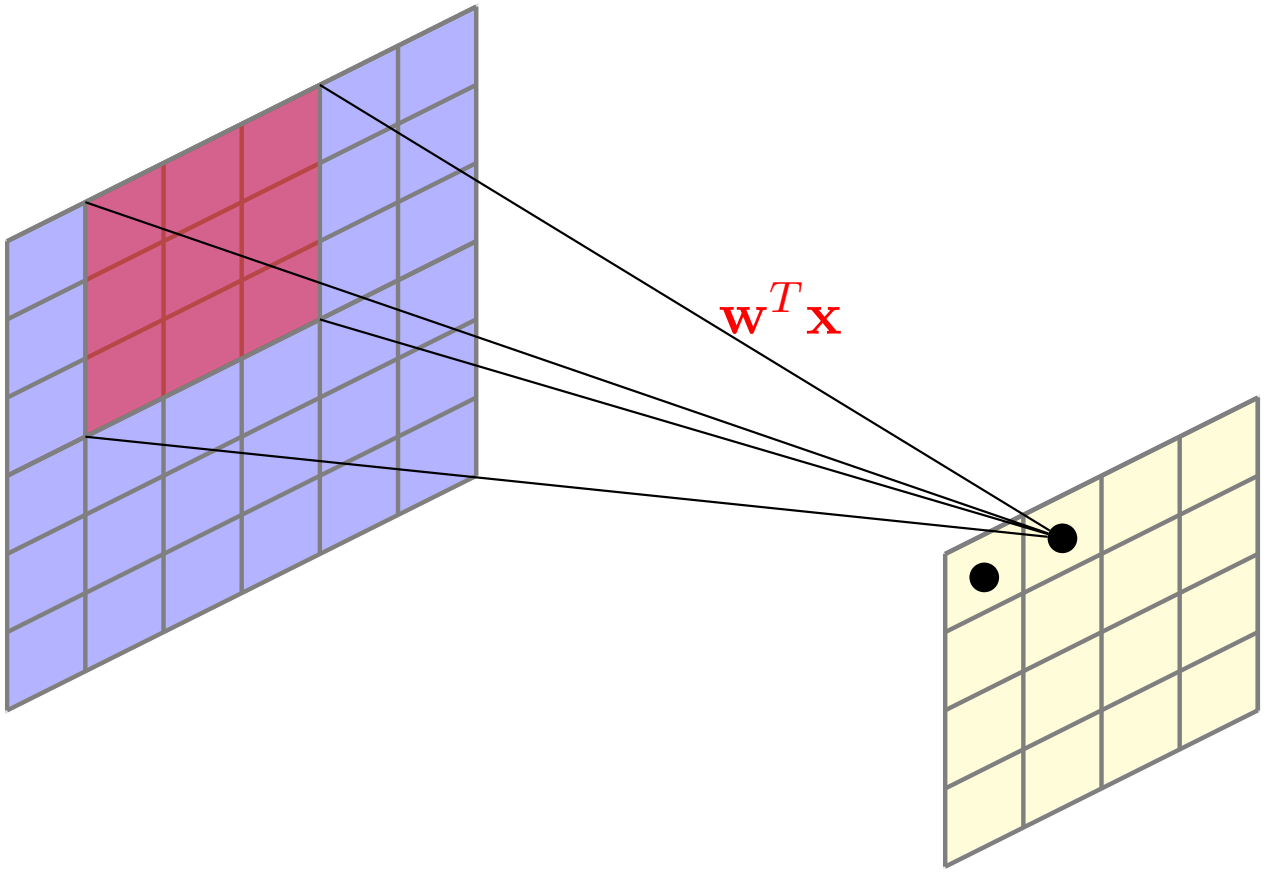


Convolution

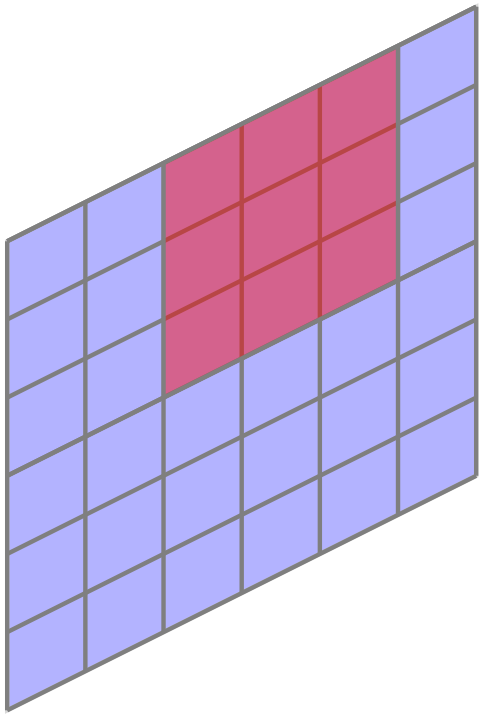


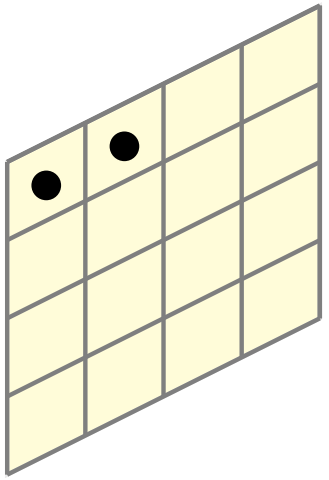


Convolution

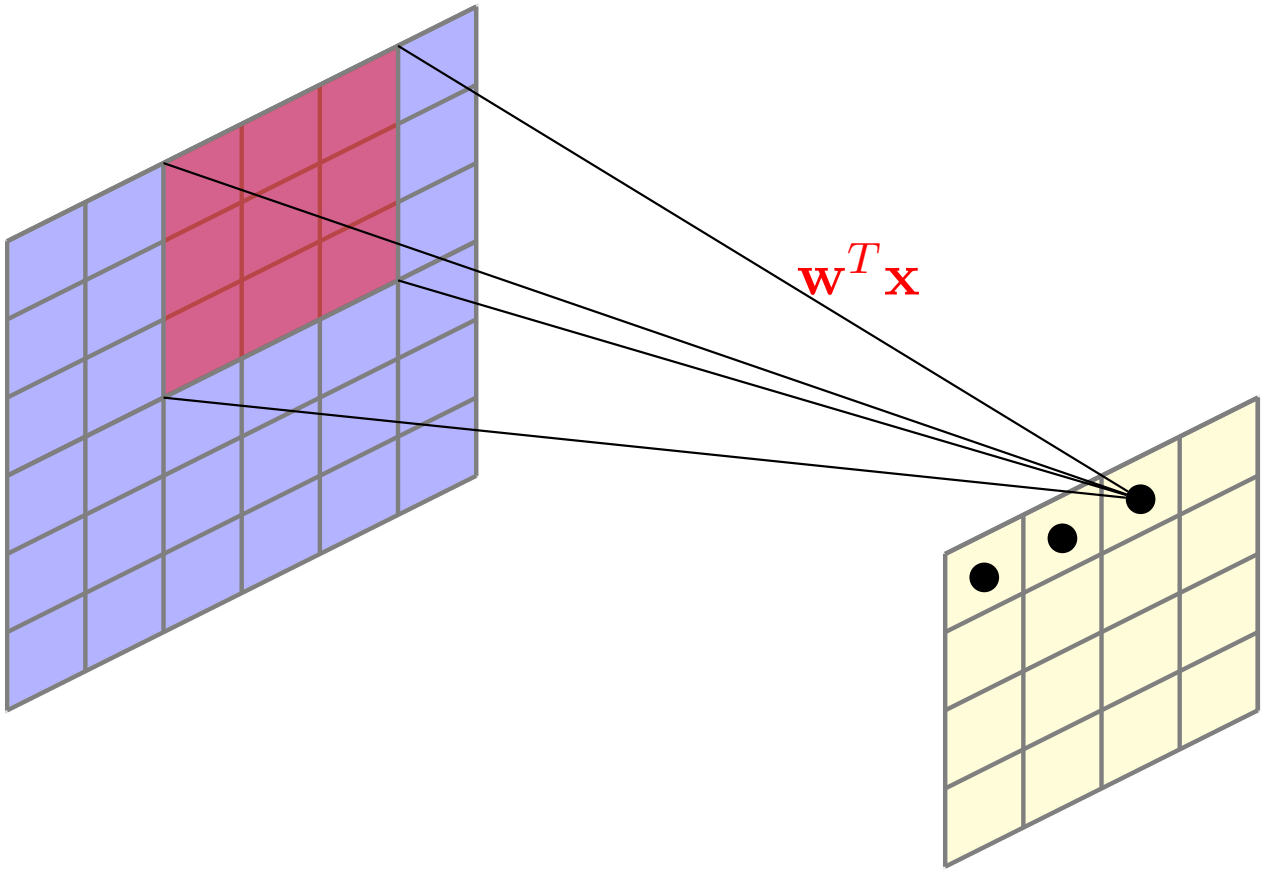


Convolution





Convolution



Output Size

- We used stride of 1, kernel with receptive field of size 3 by 3
- Output size:

$$\frac{N-K}{S}+1$$

- In previous example: $N=6, K=3, S=1$, Output size $=4$
- For $N=8, K=3, S=1$, output size is 6

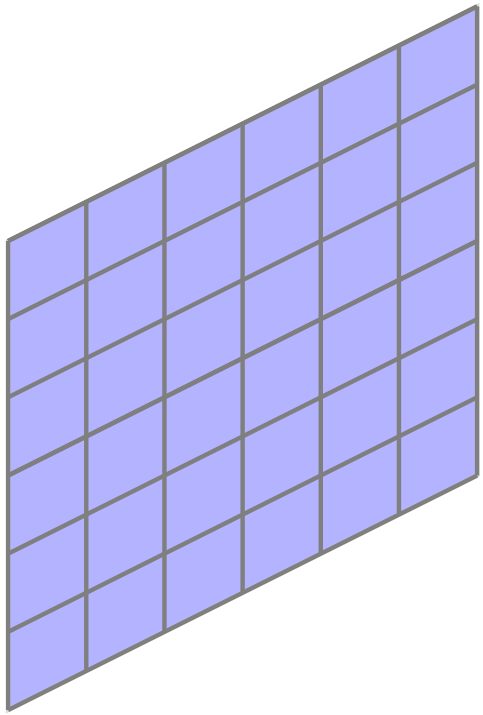
Zero Padding

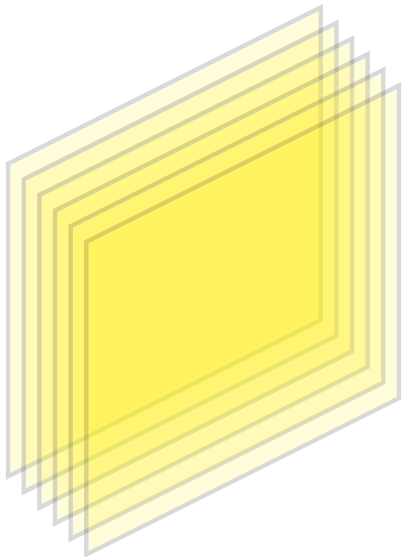
- Often, we want the output of a convolution to have the same size as the input. Solution: Zero padding.
- In our previous example:

0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

- Common to see convolution layers with stride of 1, filters of size K, and zero padding with $\frac{K-1}{2}$ to preserve size

Learn Multiple Filters





Learn Multiple Filters

- If we use 100 filters, we get 100 feature maps

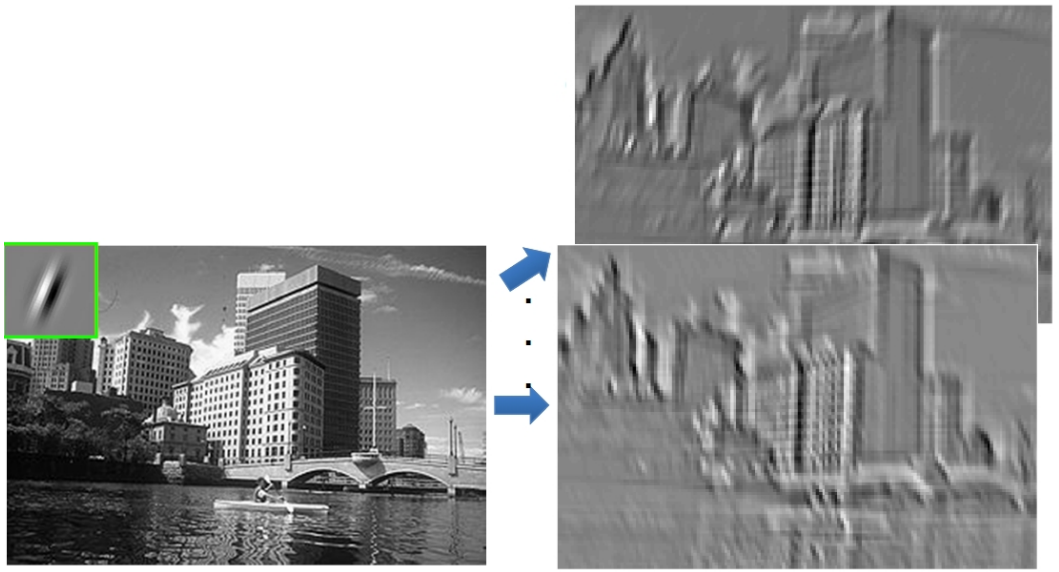
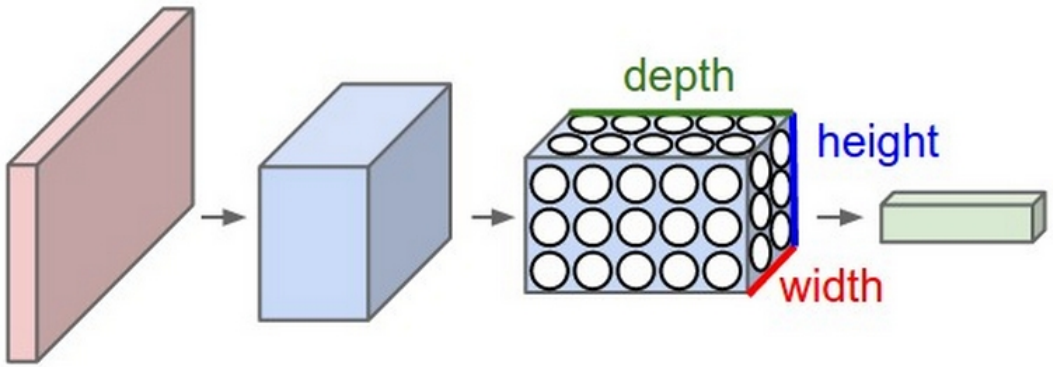


Figure: I. Kokkinos

In General

- We have only considered a 2-D image as a running example
- But we could operate on volumes (e.g. RGB Images would be depth 3 input, filter would have same depth)



In General: Output Size

- For convolutional layer:
  - Suppose input is of size $W_1 \times H_1 \times D_1$
  - Filter size is K and stride S
  - We obtain another volume of dimensions $W_2 \times H_2 \times D_2$
  - As before:

$$W_2 = \frac{W_1 - K}{S} + 1 \text{ and } H_2 = \frac{H_1 - K}{S} + 1$$

  - Depths will be equal

Convolutional Layer Parameters

Example volume: $28 \times 28 \times 3$ (RGB Image)

100 3×3 filters, stride 1

What is the zero padding needed to preserve size?

Number of parameters in this layer?

For every filter: $3 \times 3 \times 3 + 1 = 27$ parameters

Total parameters: $100 \times 27 = 270$

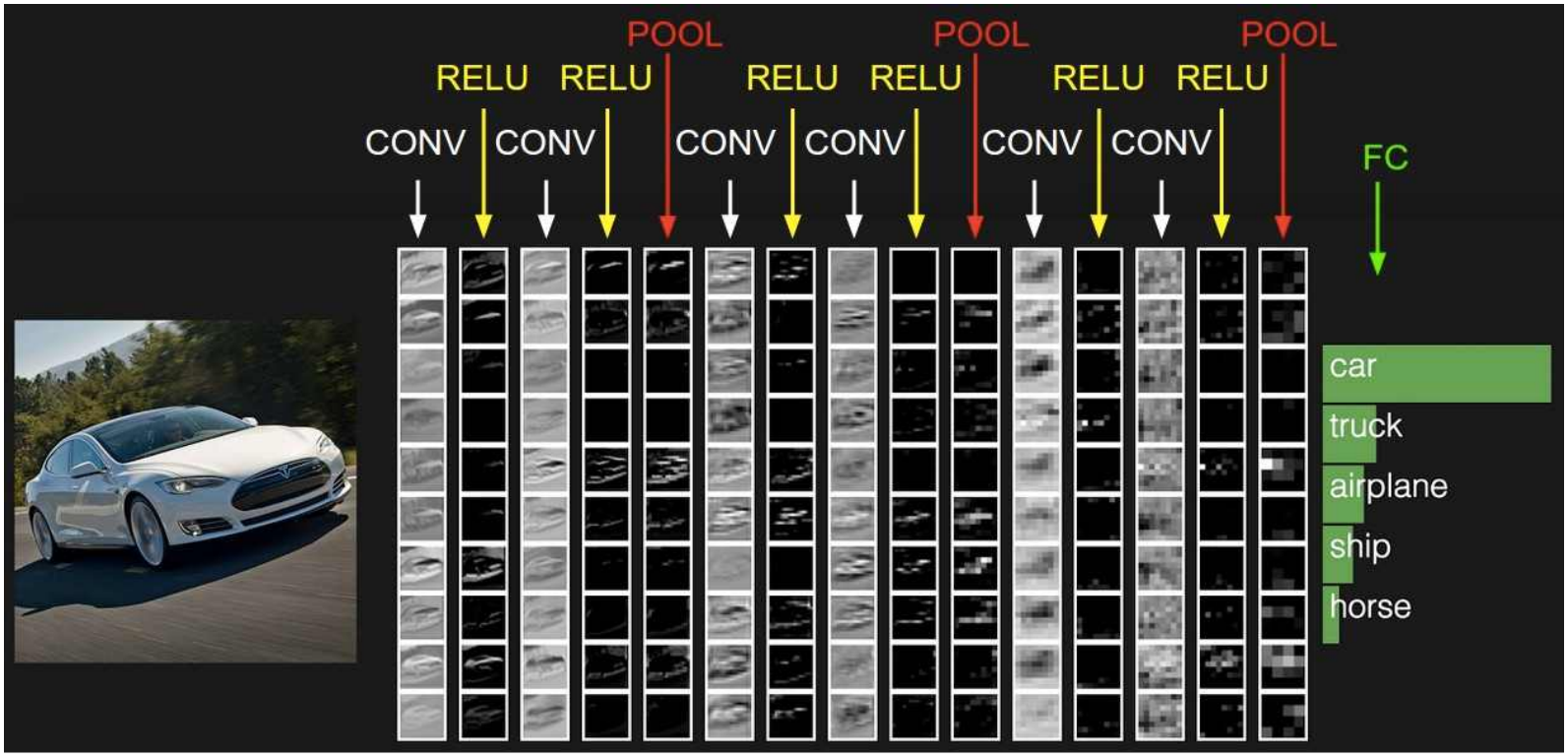
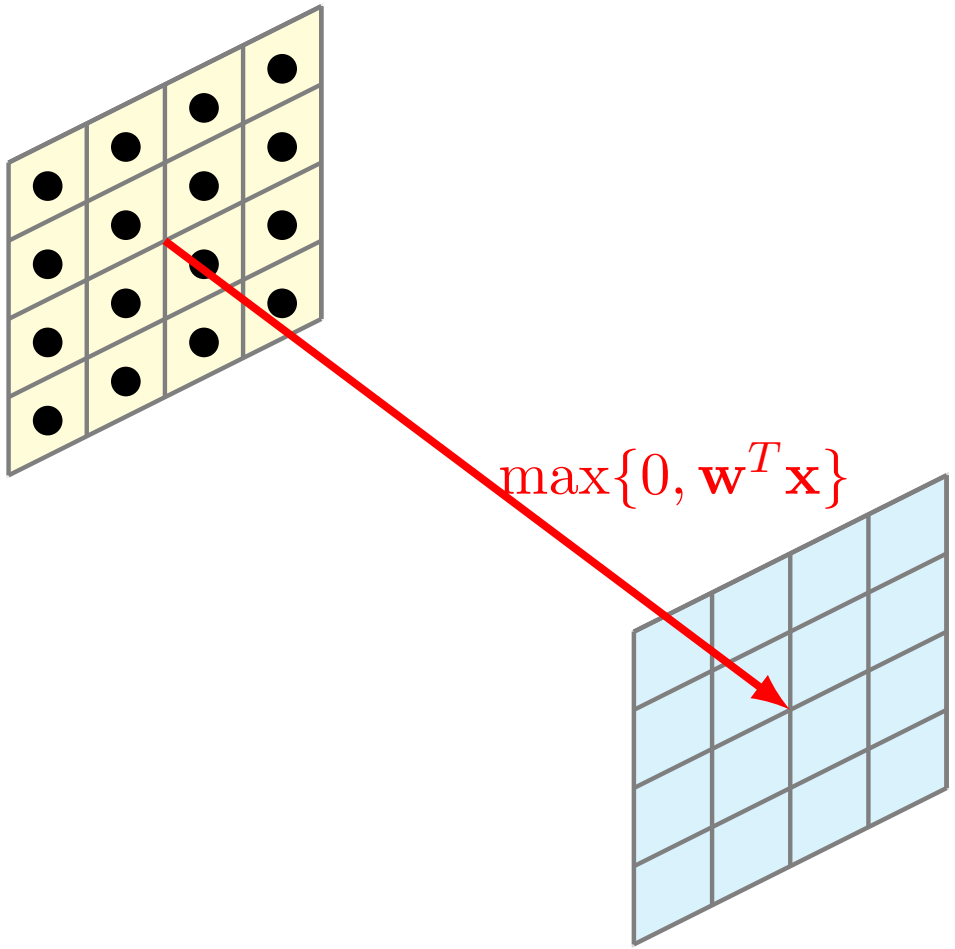


Figure: Andrej Karpathy

Non-Linearity



- After obtaining feature map, apply an elementwise non-linearity to obtain a transformed feature map (same size)

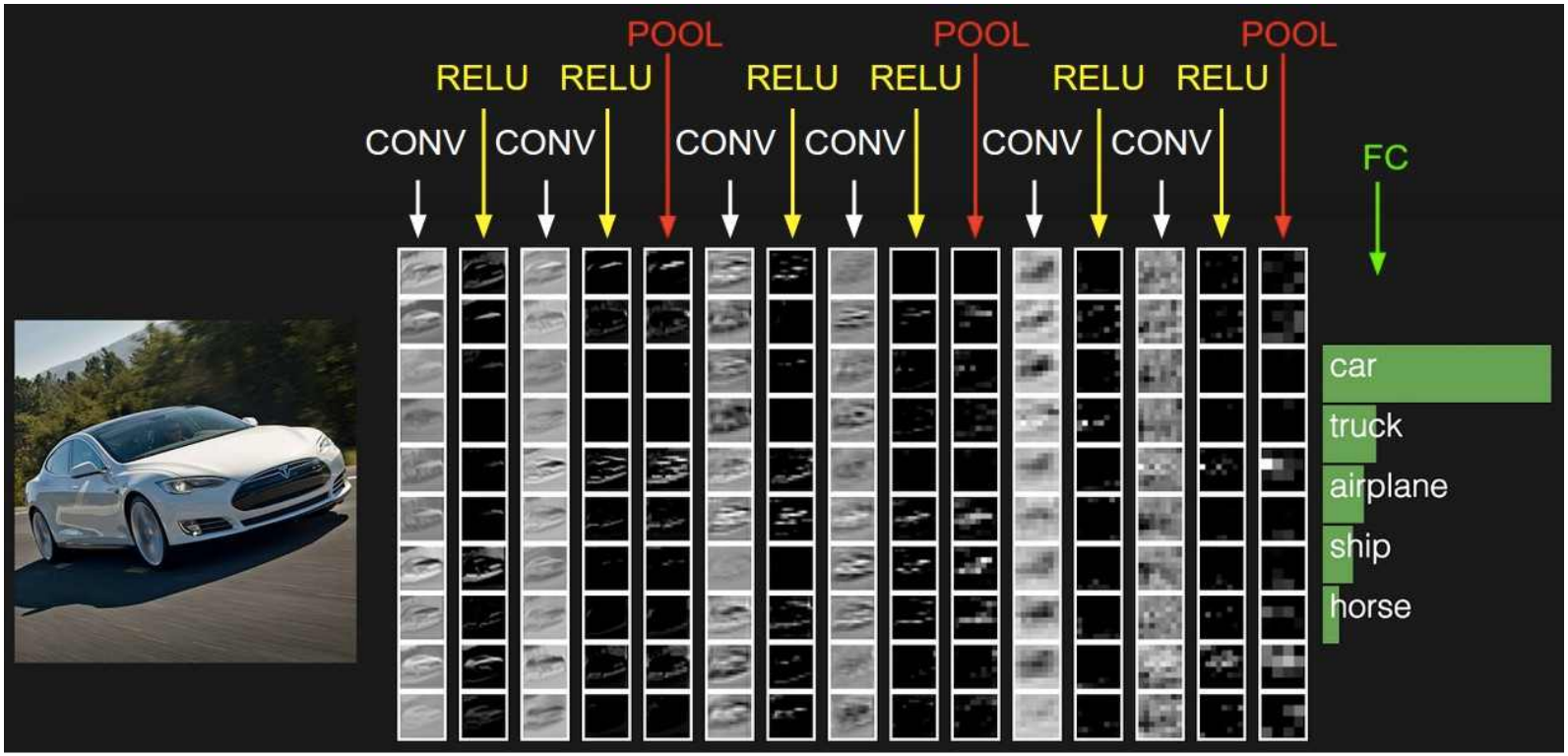
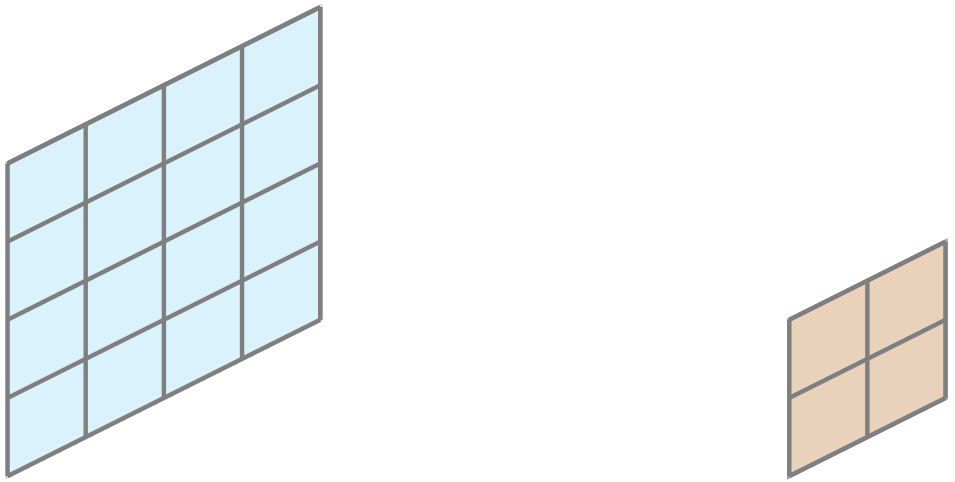
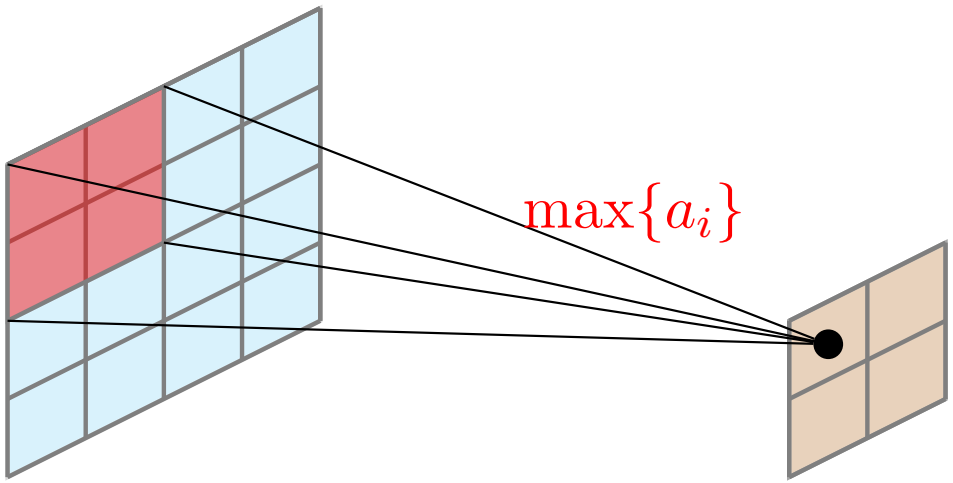


Figure: Andrej Karpathy

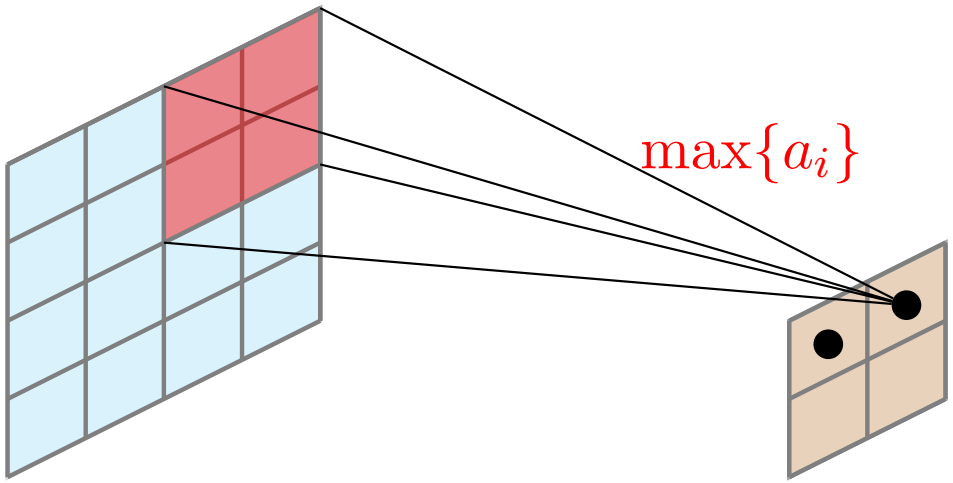
Pooling



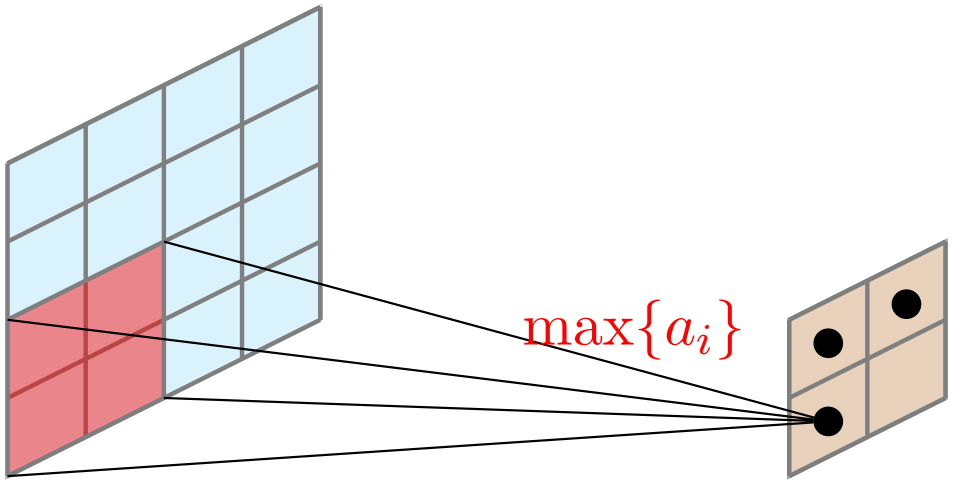
Pooling



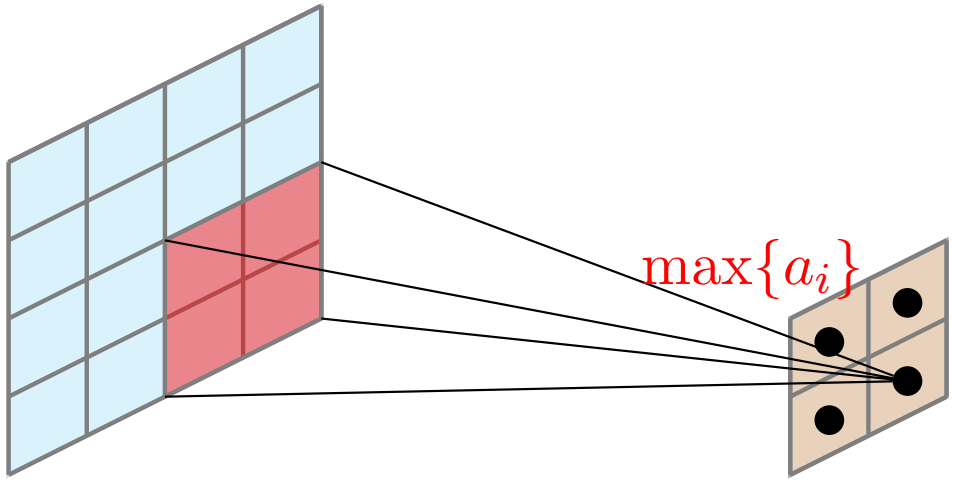
Pooling



Pooling



Pooling



- Other options: Average pooling, L2-norm pooling, random pooling

So what's left: Fully Connected Layers

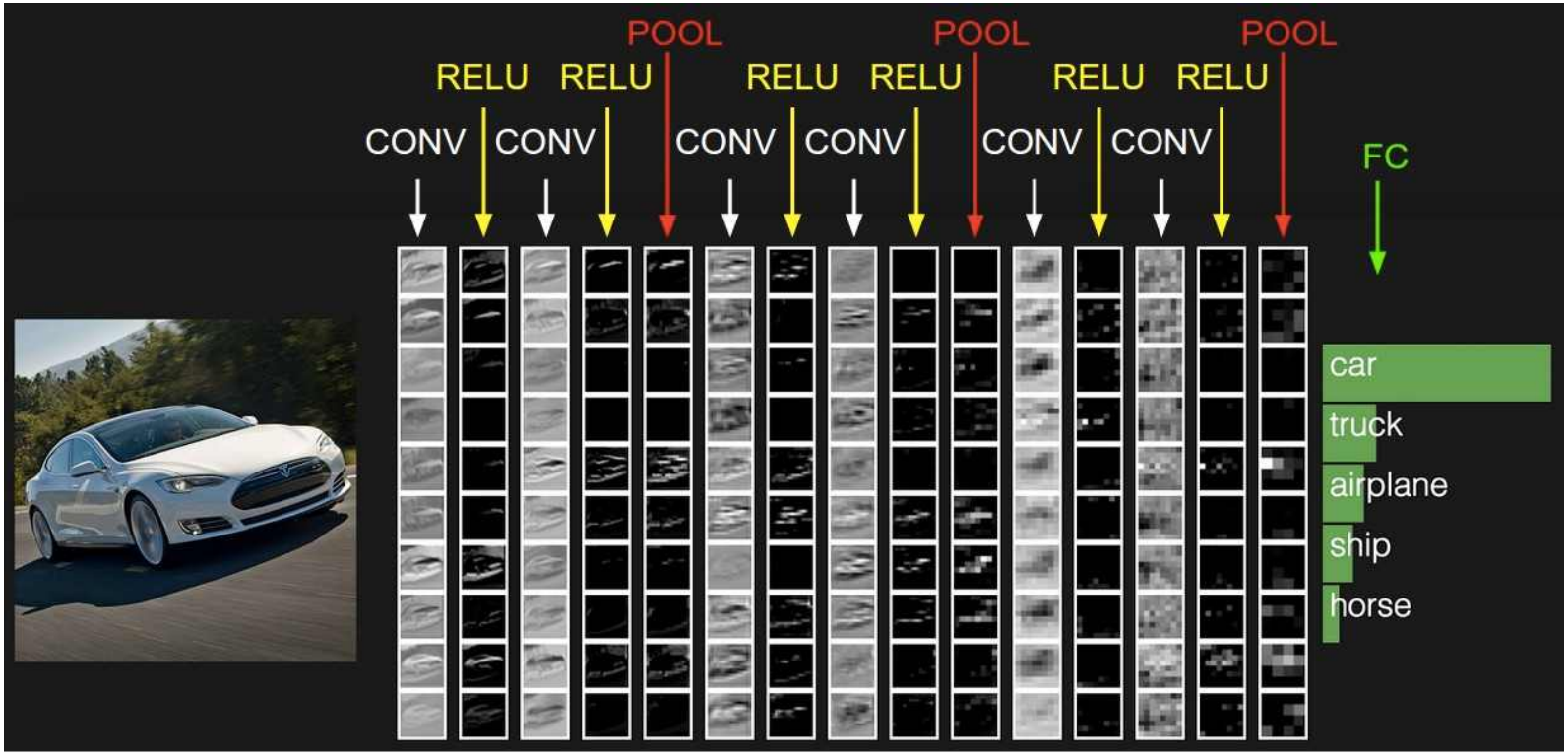
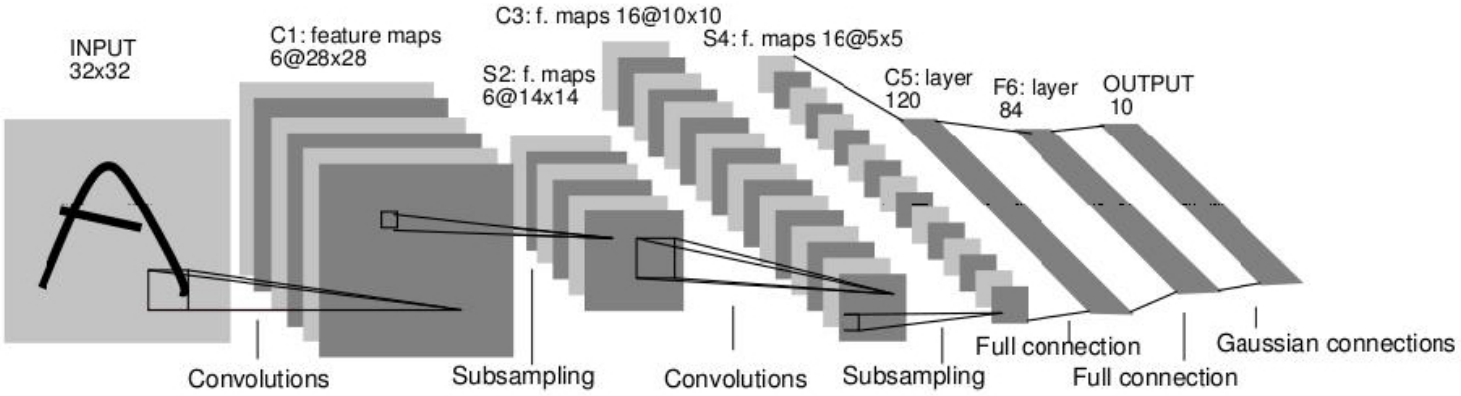


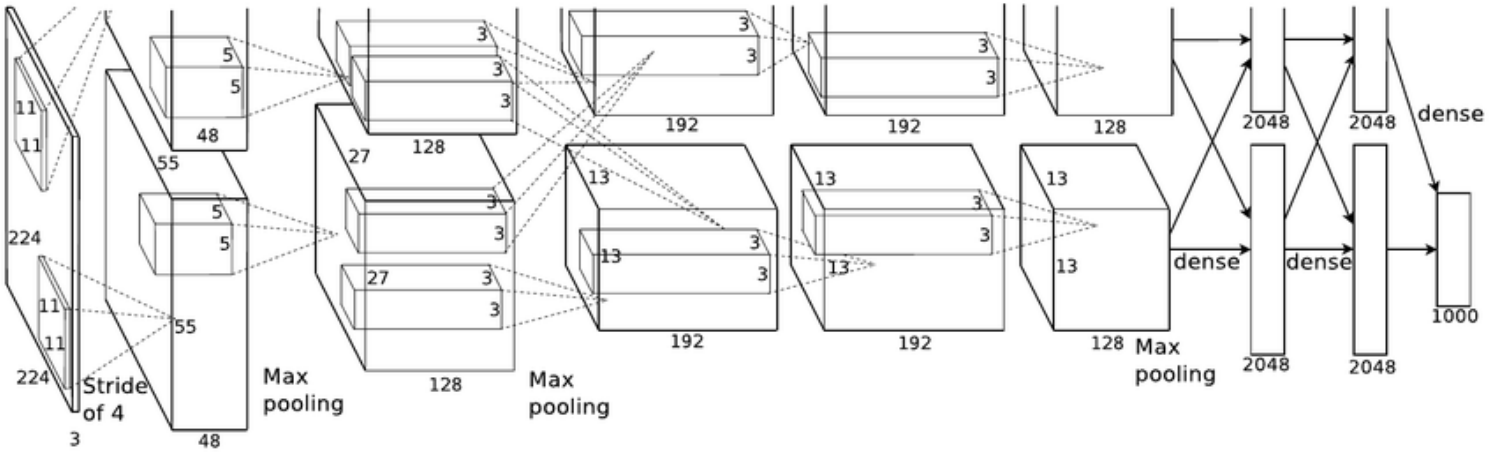
Figure: Andrej Karpathy

LeNet-5



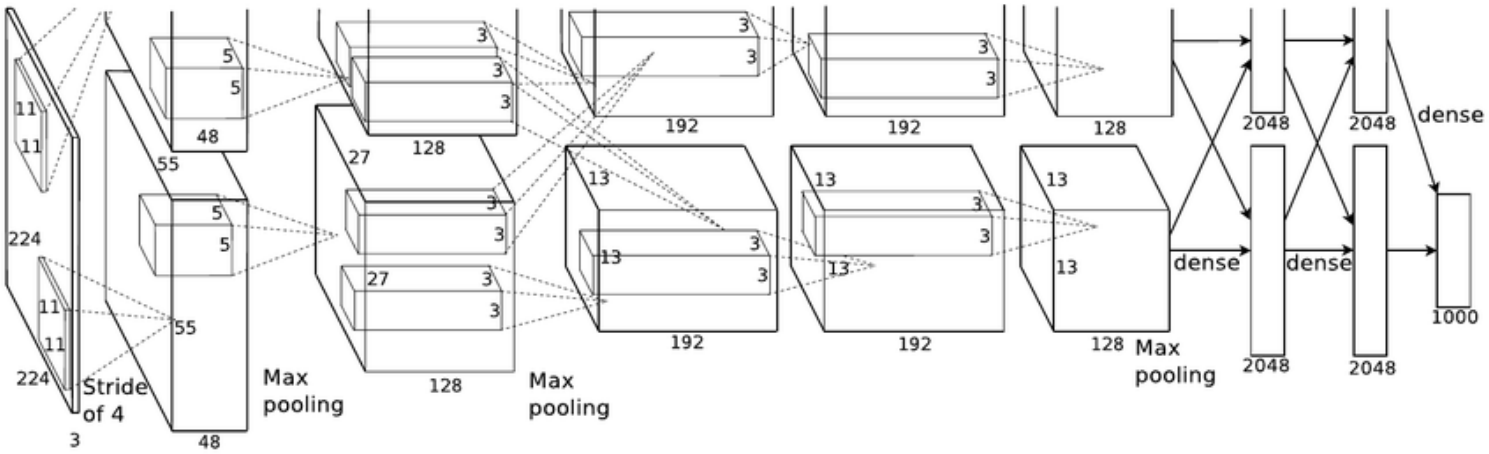
- Filters are of size 5×5, stride 1
- Pooling is 2×2, with stride 2
- How many parameters?

AlexNet



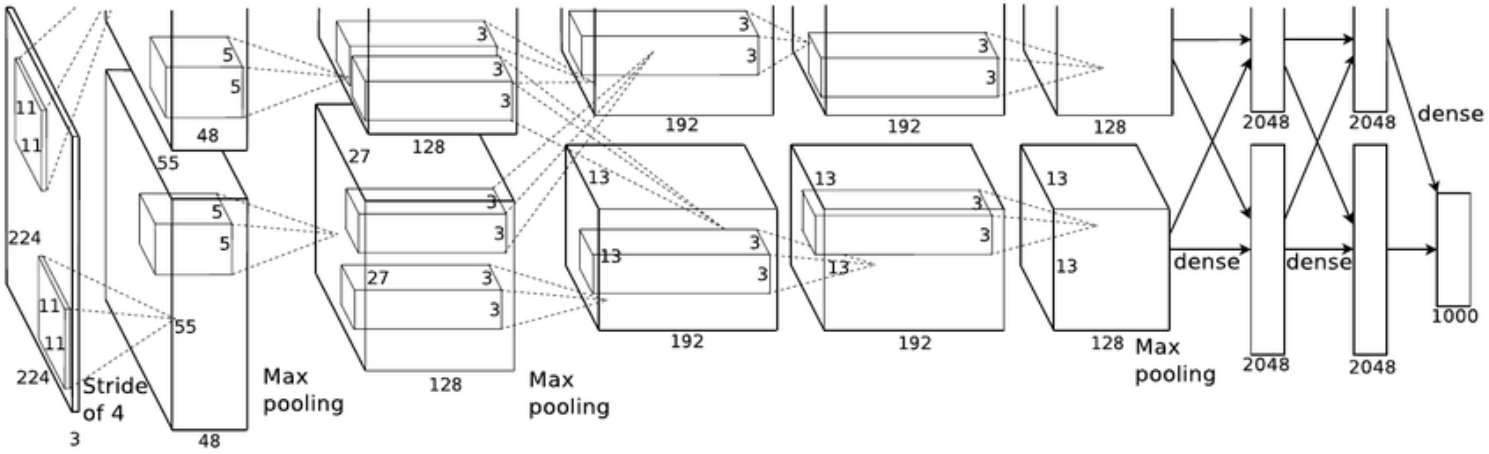
- Input image: 227 X 227 X 3
- First convolutional layer: 96 filters with K = 11 applied with stride = 4
- Width and height of output: $\frac{227-11}{4} + 1 = 55$

AlexNet



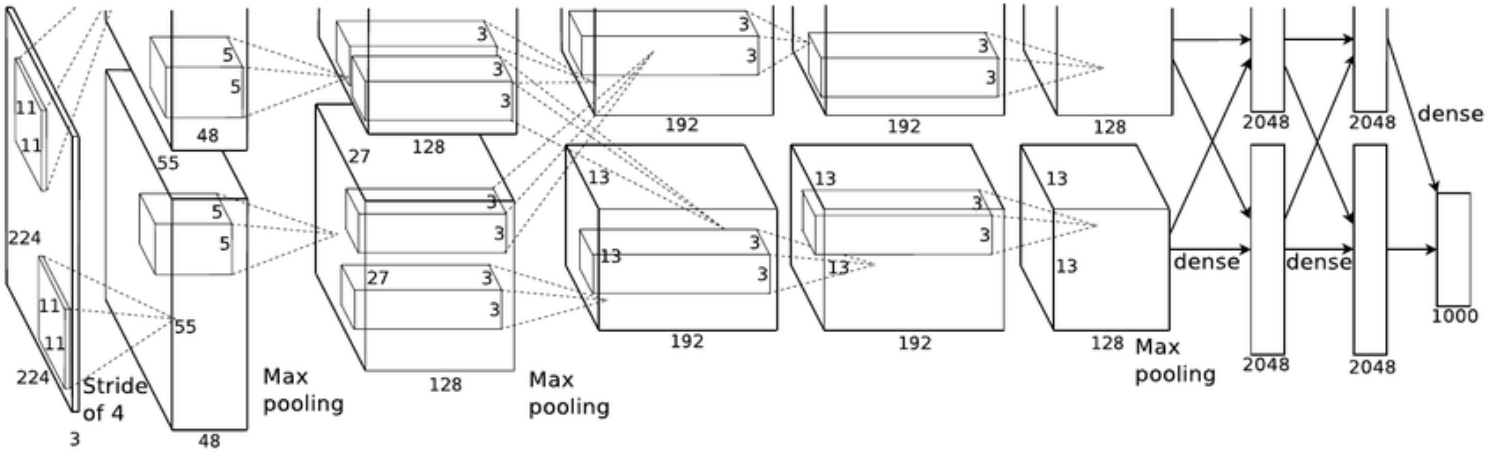
- Number of parameters in first layer?
- 11 X 11 X 3 X 96 = 34848

AlexNet



- Next layer: Pooling with 3 X 3 filters, stride of 2
- Size of output volume: 27
- Number of parameters?

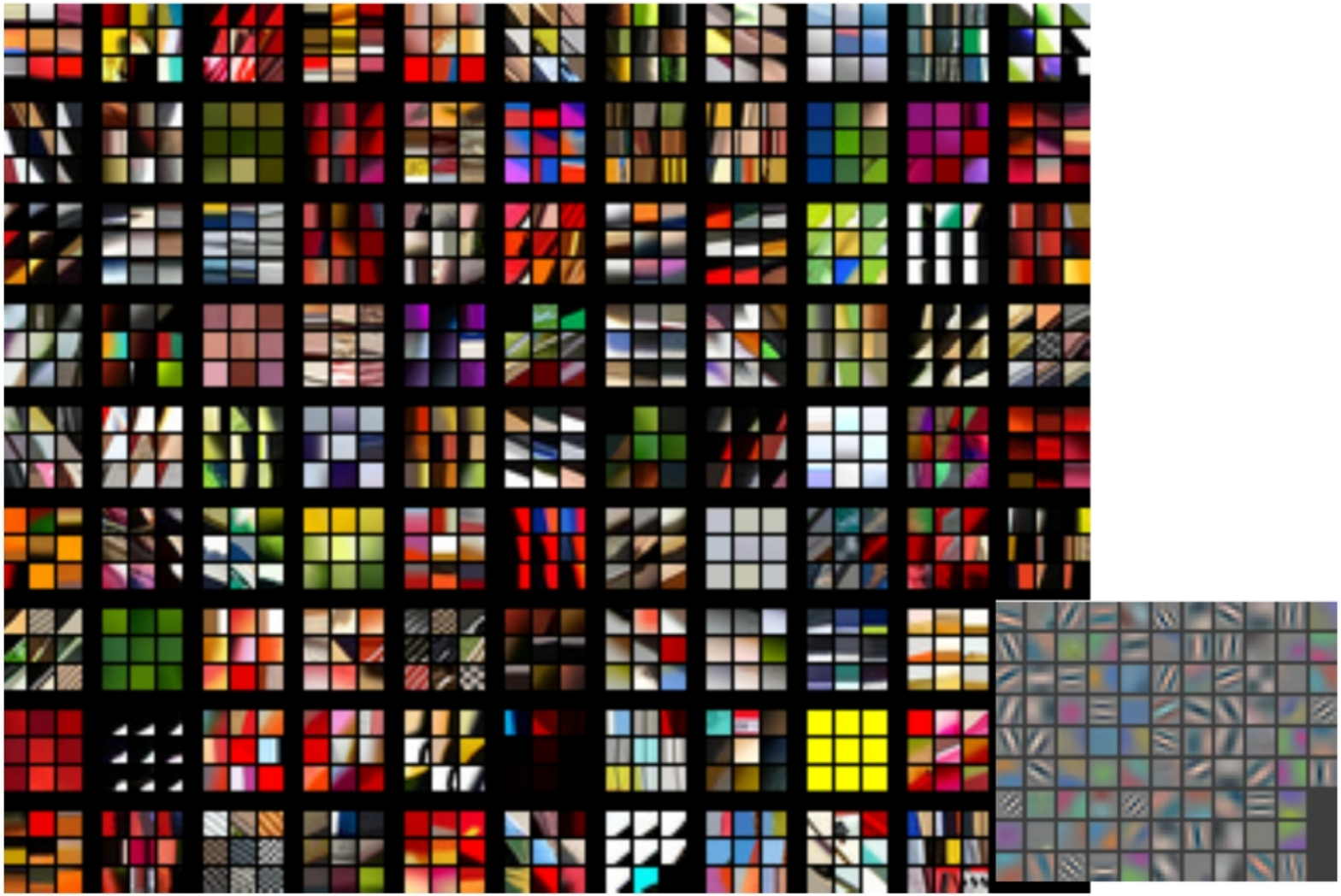
AlexNet



- Popularized the use of ReLUs
- Used heavy data augmentation (flipped images, random crops of size 227 by 227)
- Parameters: Dropout rate 0.5, Batch size = 128, Weight decay term: 0.0005 ,Momentum term α = 0.9, learning rate η = 0.01, manually reduced by factor of ten on monitoring validation loss.

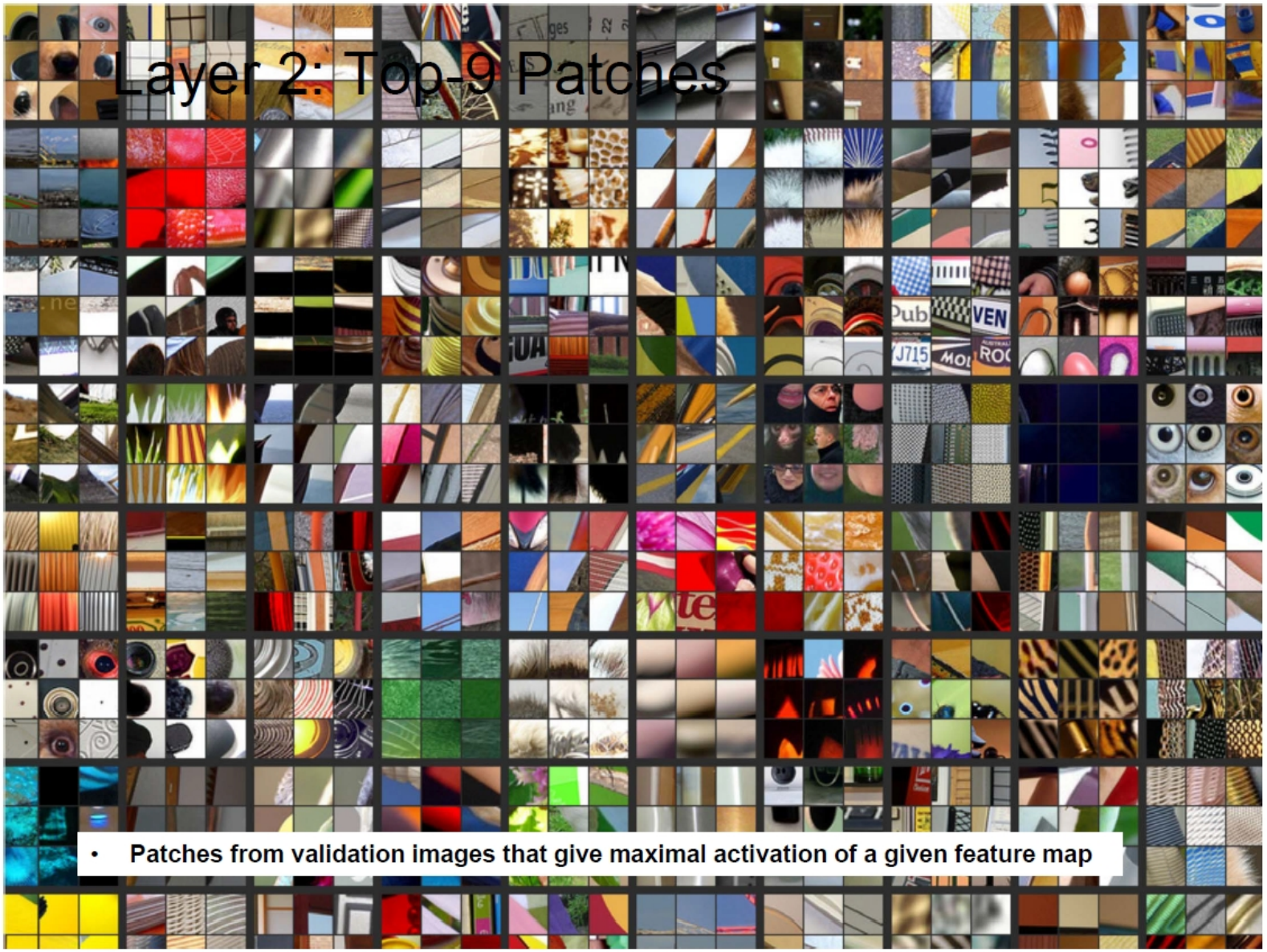
Short Digression: How do the features look like?

Layer 1 filters

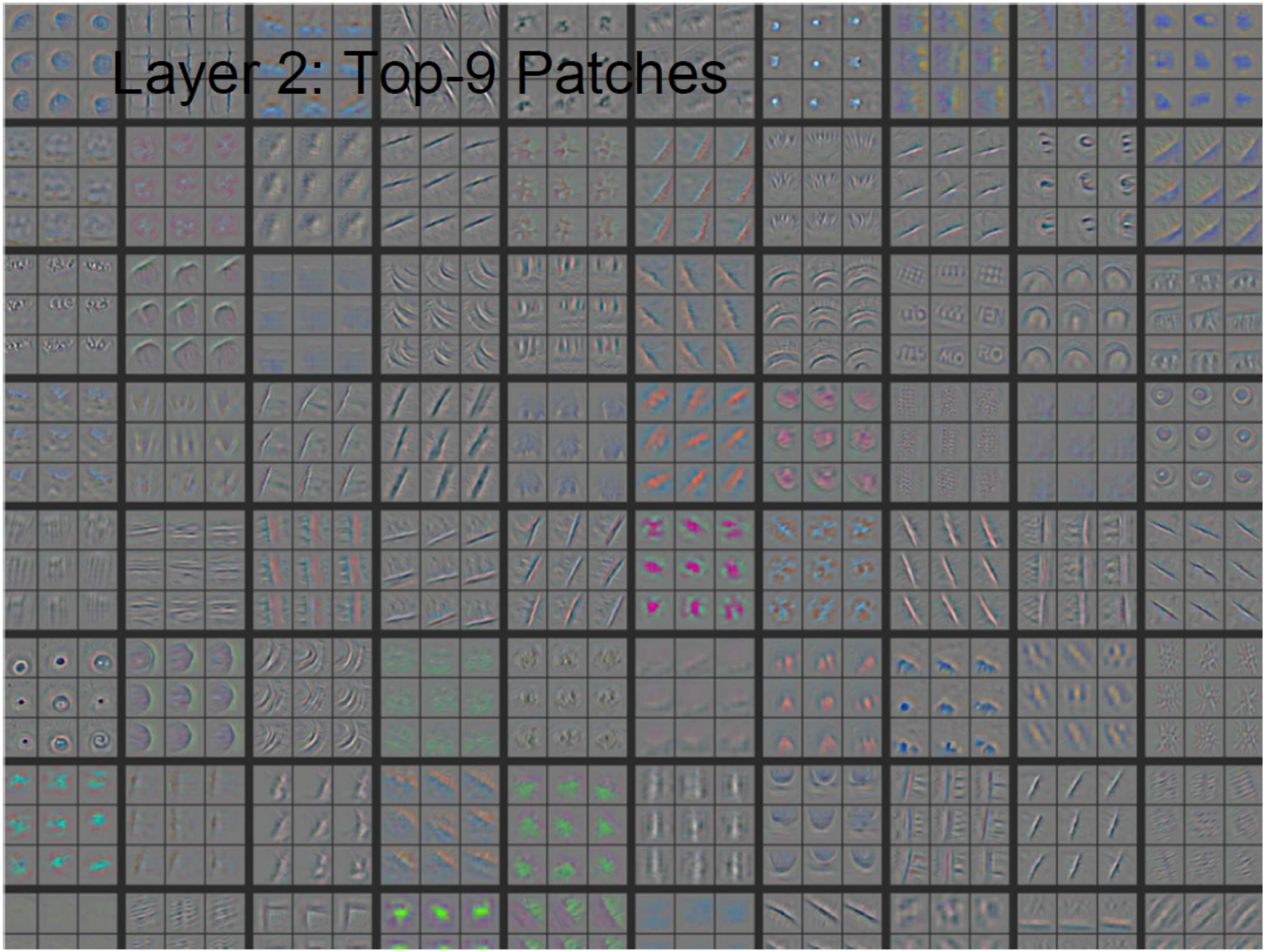


This and the next few illustrations are from Rob Fergus

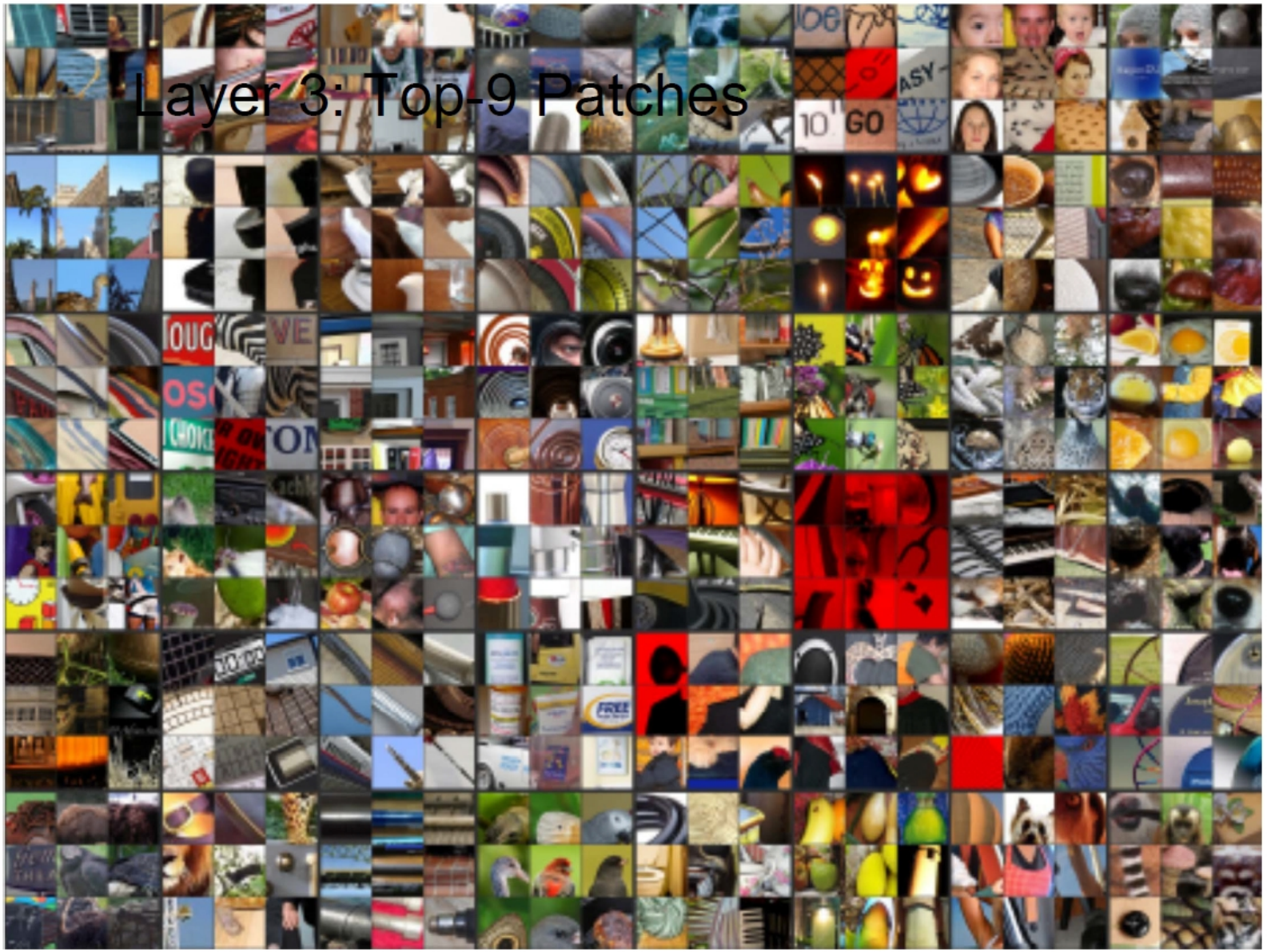
Layer 2 Patches



Layer 2 Patches

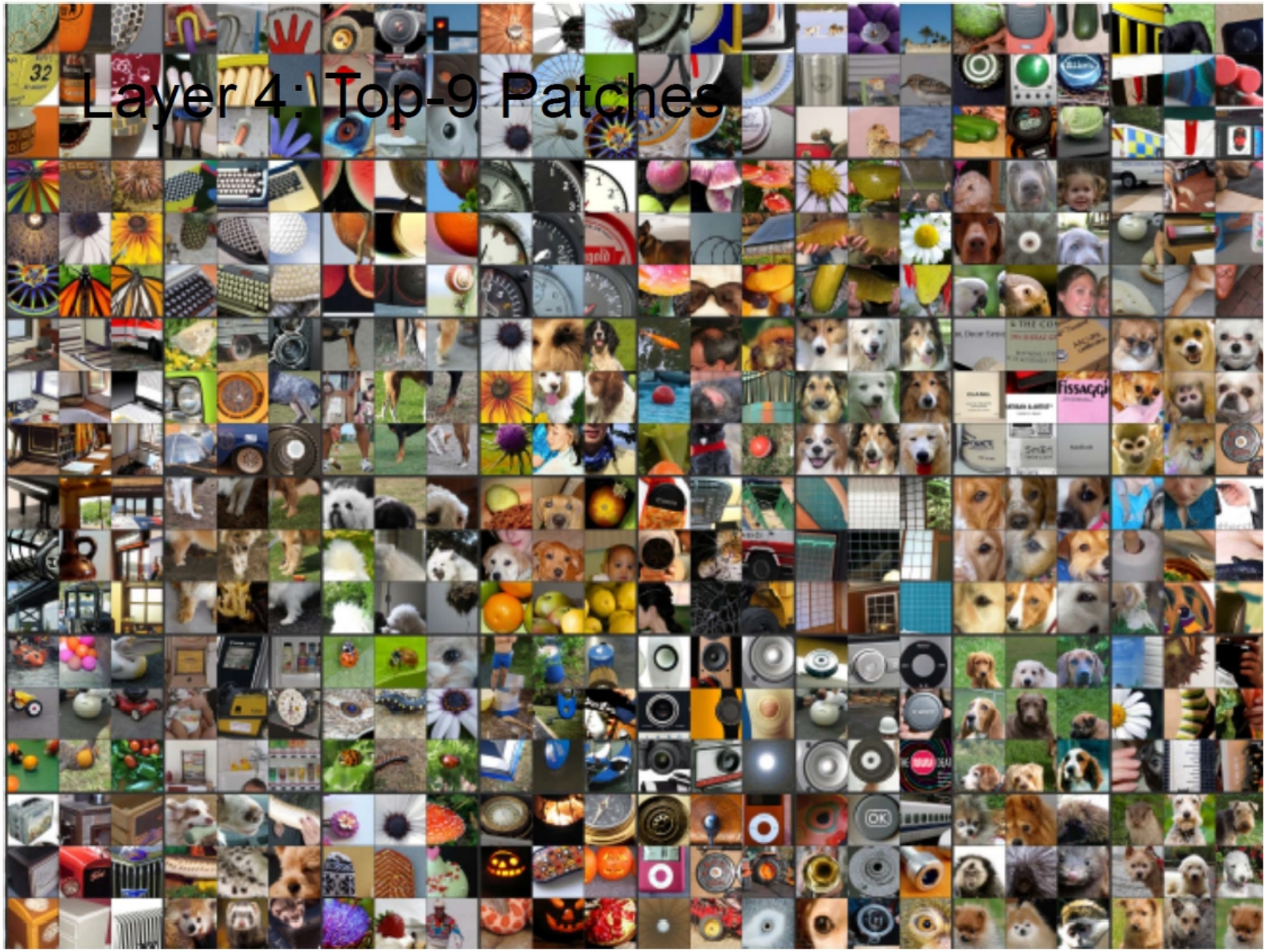


Layer 3 Patches



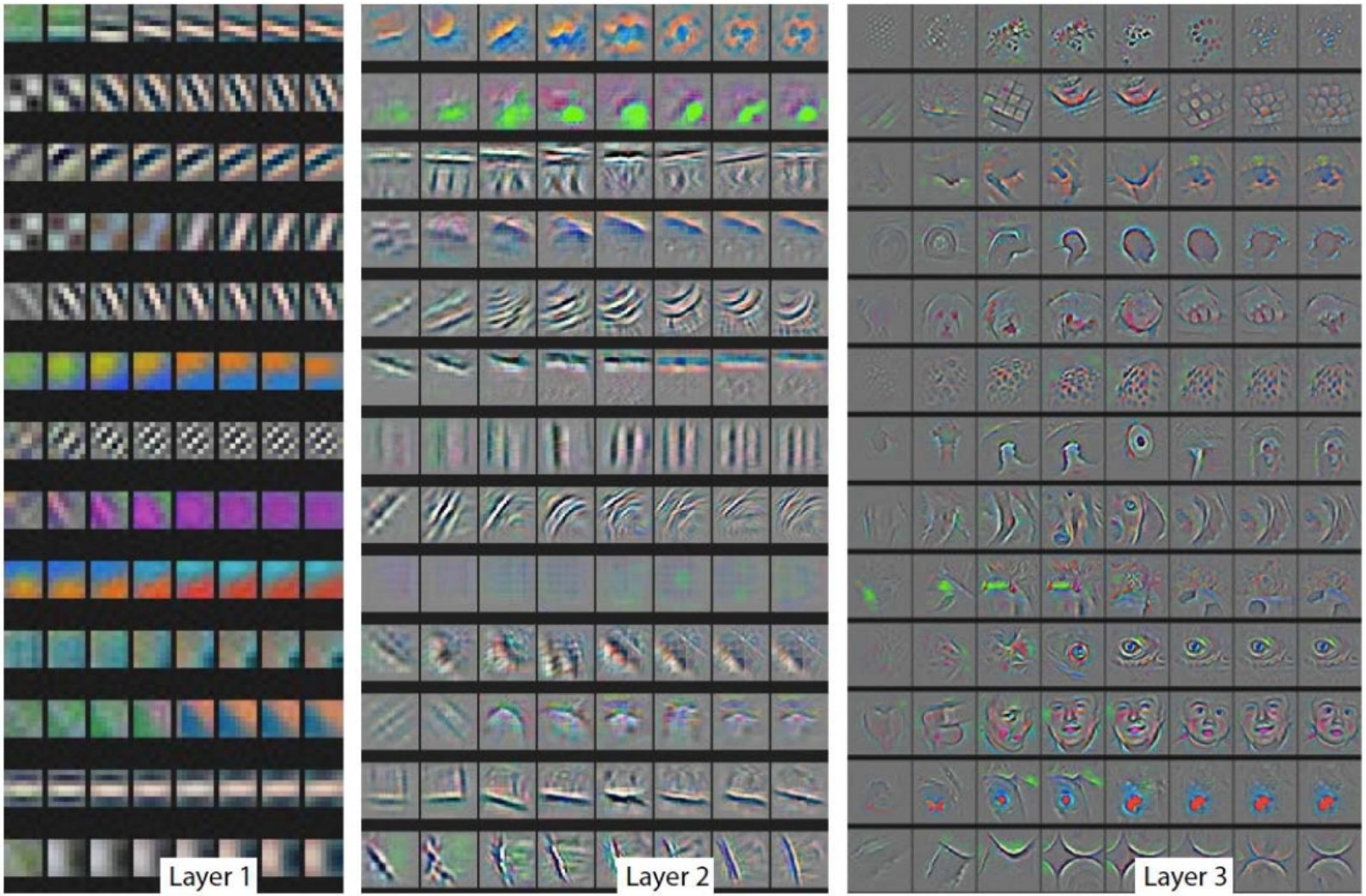
Layer 3 Patches

Layer 4 Patches

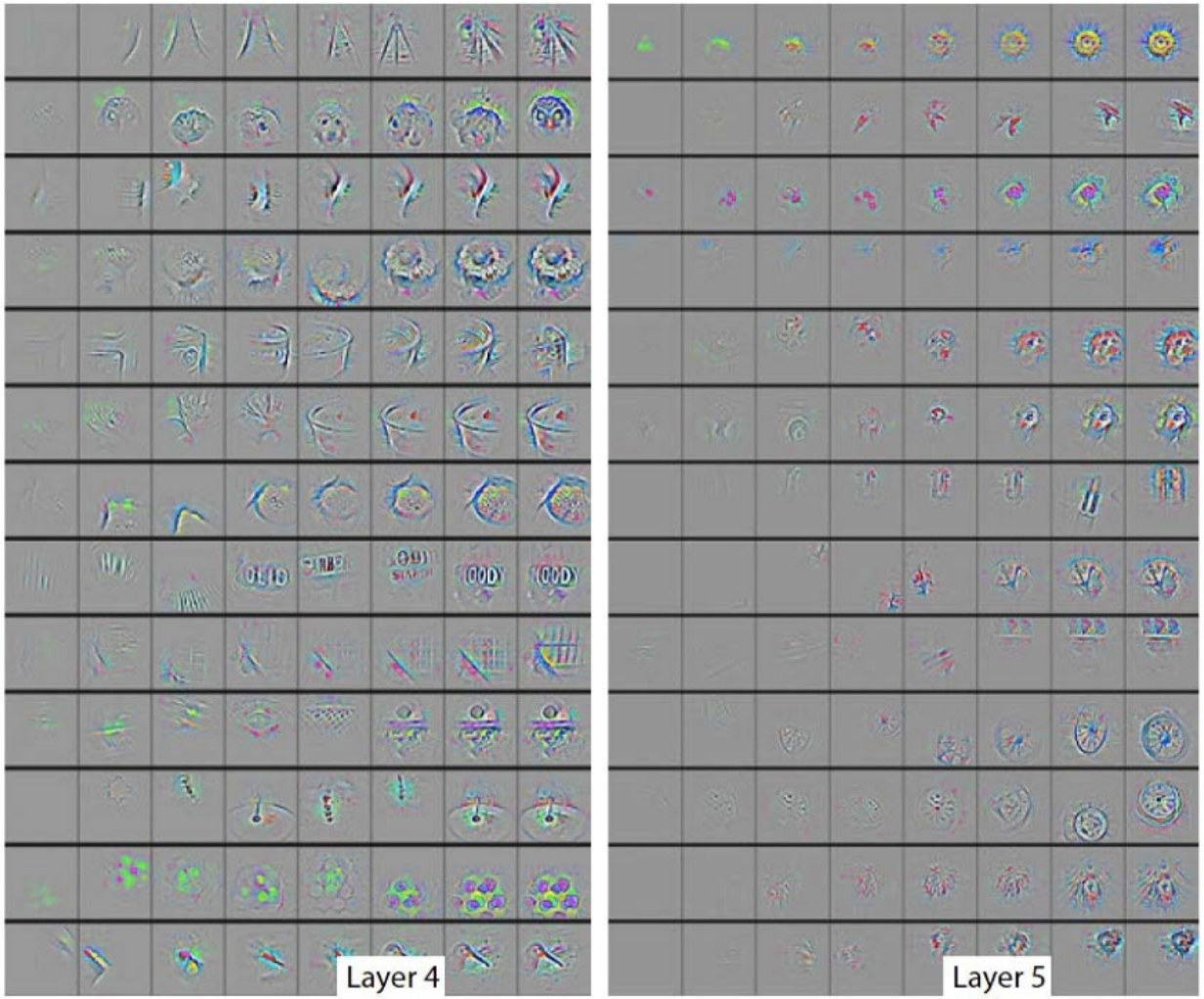


Layer 4 Patches

Evolution of Filters



Evolution of Filters



Caveat?

ImageNet 2013

- Was won by a network similar to AlexNet (Matthew Zeiler and Rob Fergus)
- Changed the first convolutional layer from 11 X 11 with stride of 4, to 7 X 7 with stride of 2
- AlexNet used 384, 384 and 256 layers in the next three convolutional layers, ZF used 512, 1024, 512
- ImageNet 2013: 14.8 % (reduced from 15.4 %) (top 5 errors)

VGGNet(Simonyan and Zisserman, 2014)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

- Best model: Column D.
- Error: 7.3 % (top five error)

VGGNet(Simonyan and Zisserman, 2014)

- Total number of parameters: 138 Million (calculate!)
- Memory (Karpathy): 24 Million X 4 bytes $\approx$ 93 MB per image
- For backward pass the memory usage is doubled per image
- Observations:
  - Early convolutional layers take most memory
  - Most parameters are in the fully connected layers

Going Deeper

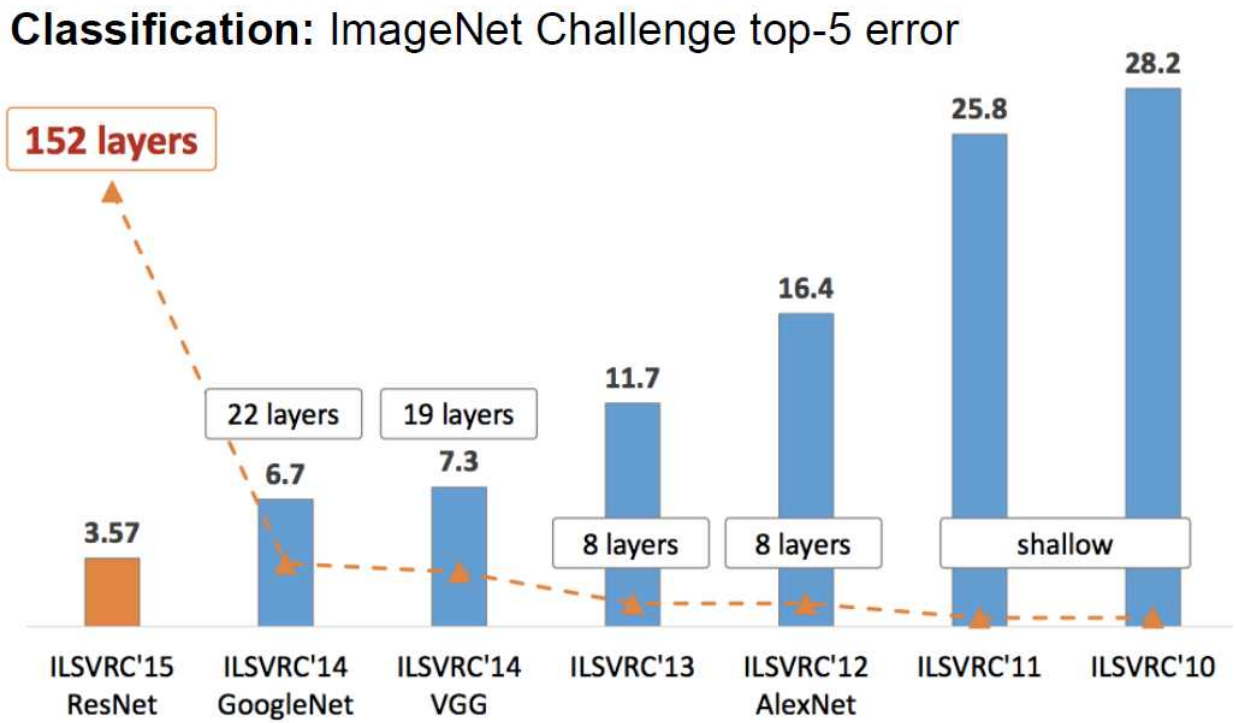
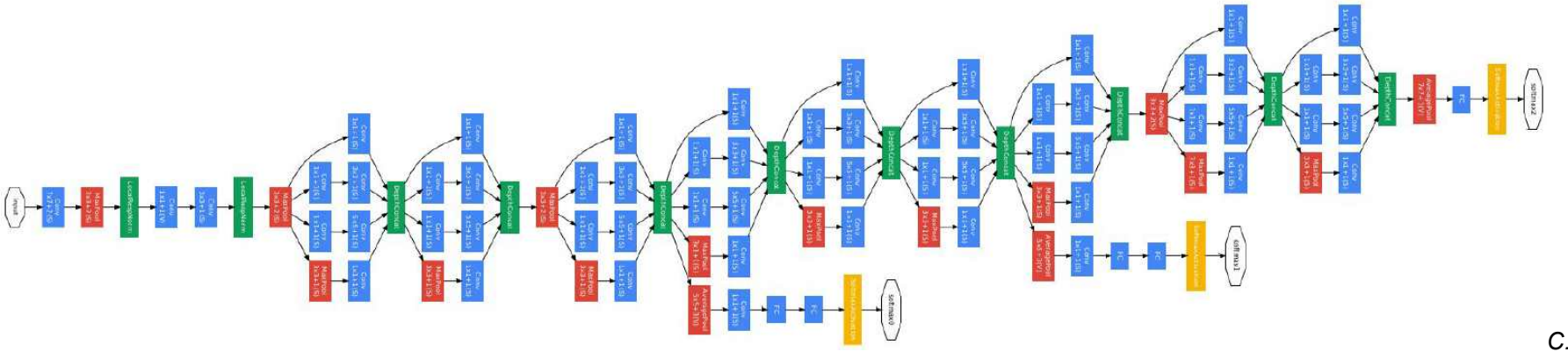


Figure: Kaiming He, MSR

Google LeNet

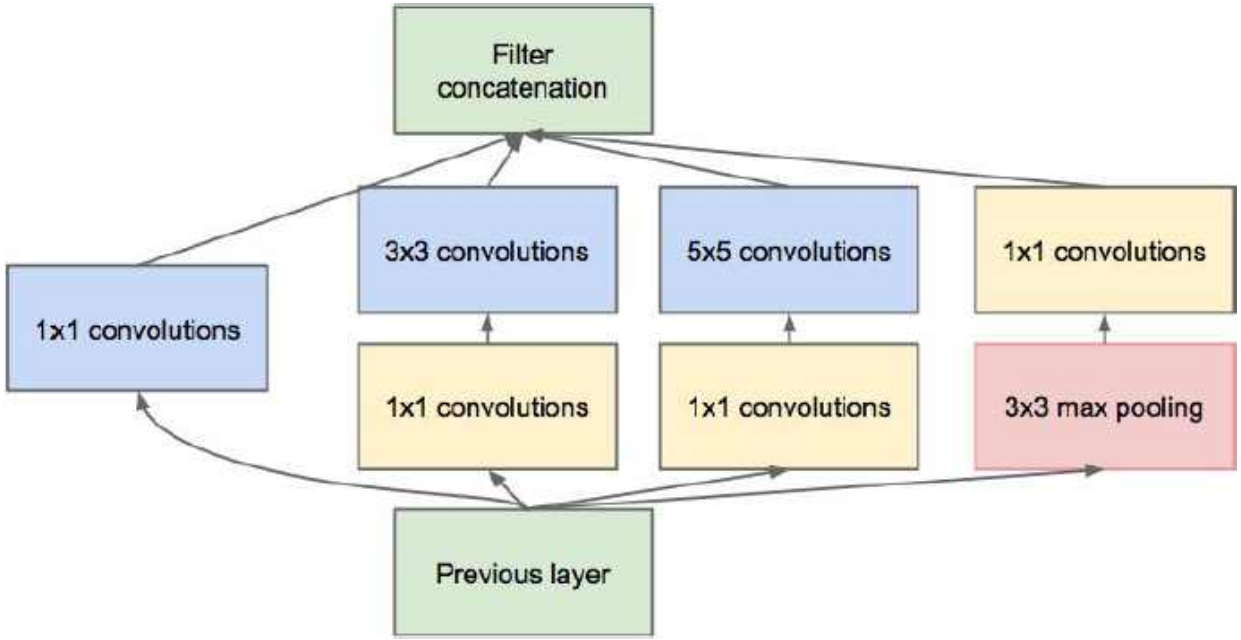


C.

Szegedy et al, Going Deeper With Convolutions, CVPR 2015

- Error: 6.7 % (top five error)

The Inception Module



- Parallel paths with different receptive field sizes - capture sparse patterns of correlation in stack of feature maps
- Also include auxiliary classifiers for ease of training
- Also note 1 by 1 convolutions

Google LeNet

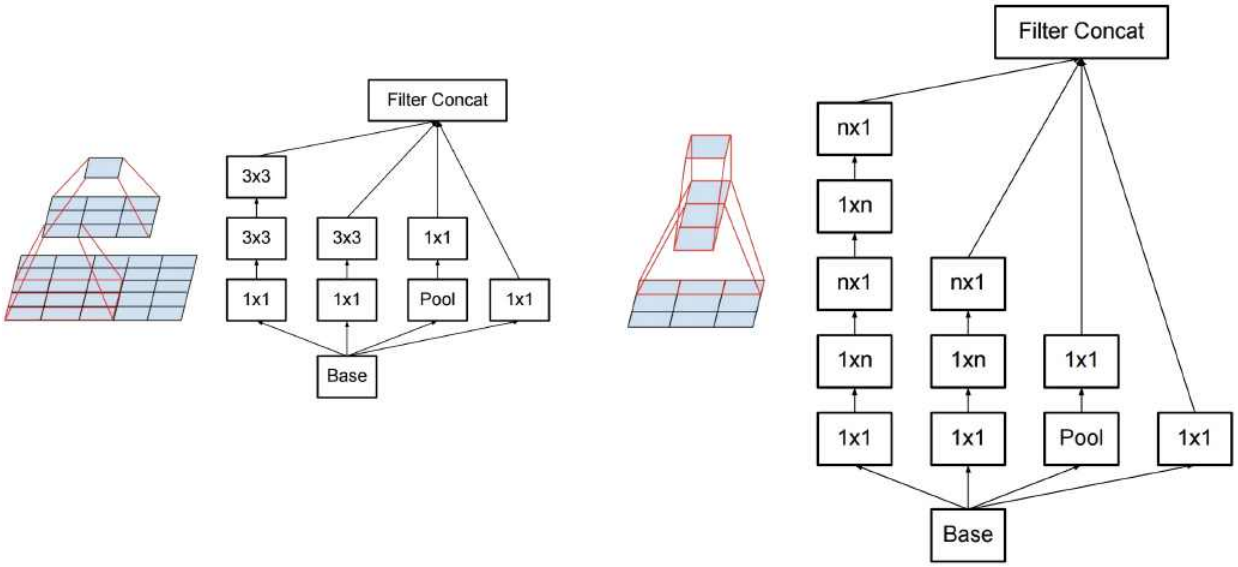
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

C. Szegedy et al, Going Deeper With Convolutions, CVPR 2015

Google LeNet

- Has 5 Million or 12X fewer parameters than AlexNet
- Gets rid of fully connected layers

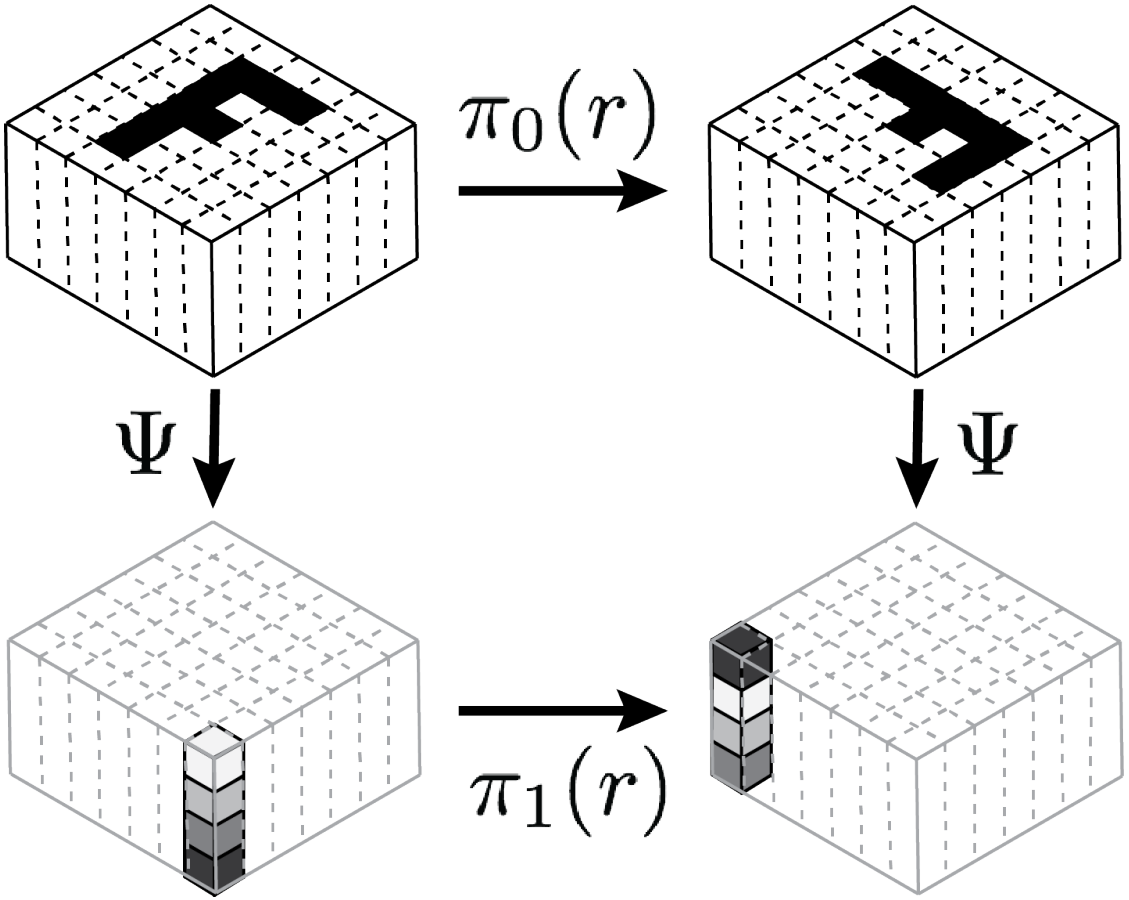
Inception v2, v3



C. Szegedy et al, Rethinking the Inception Architecture for Computer Vision, CVPR 2016

- Use Batch Normalization during training to reduce dependence on auxiliary classifiers
- More aggressive factorization of filters

Steerability



[Cohen & Welling, 2016]

Scattering networks (Mallat, 2012)