CNN

ENEE 4583/5597

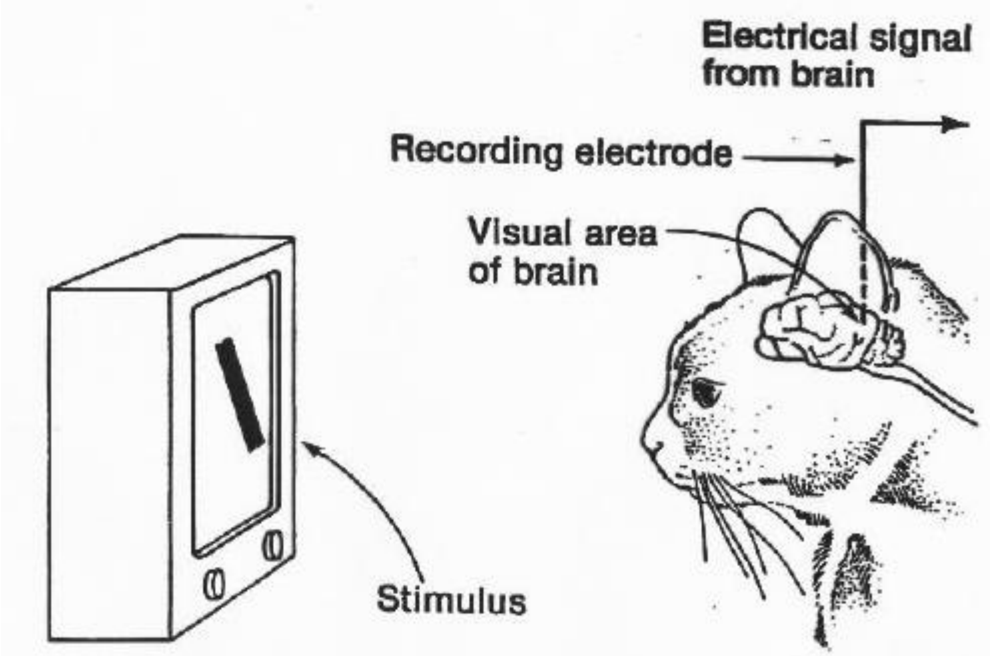
Deep Learning

Dr. Alsamman

Slide Credits: M Neilson, R. Bhiksha,

Hubel-Wiesel (1959)

- Nobel Prize
- Experiments on cats (1959)
  - Later on monkeys
  - light of different patterns
  - measured neural responses in striate cortex

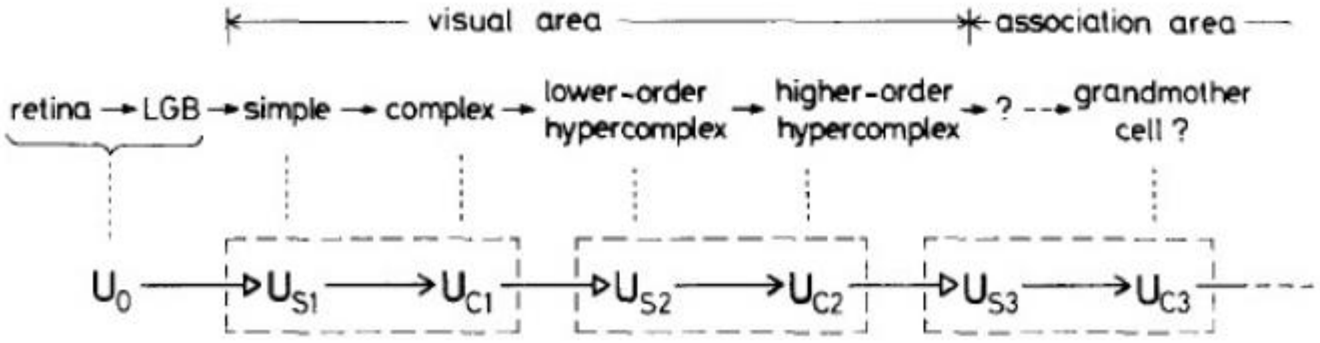


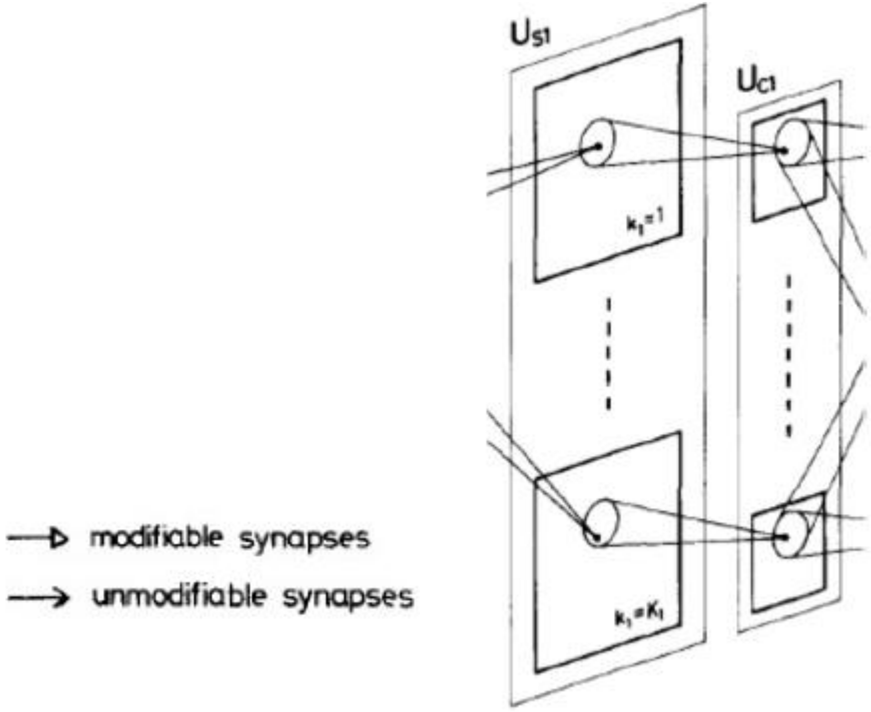
Model

- Receptive fields: Specific retinal areas caused firing of single cortical units
- Fields subdivided into excitatory and inhibitory regions
- Light must fall on excitatory regions and NOT fall on inhibitory regions
- Fields could be oriented in a vertical, horizontal or oblique manner
- Complex buildup: complex cells filtered noisy patterns
- C-cells relied on a bank of simple (s) cells
- Deeper more complex cells relying on a bank of c-cells
- Model can’t accommodate for color, position invariance, size invariance

Kunihiko Fukushima (1980)

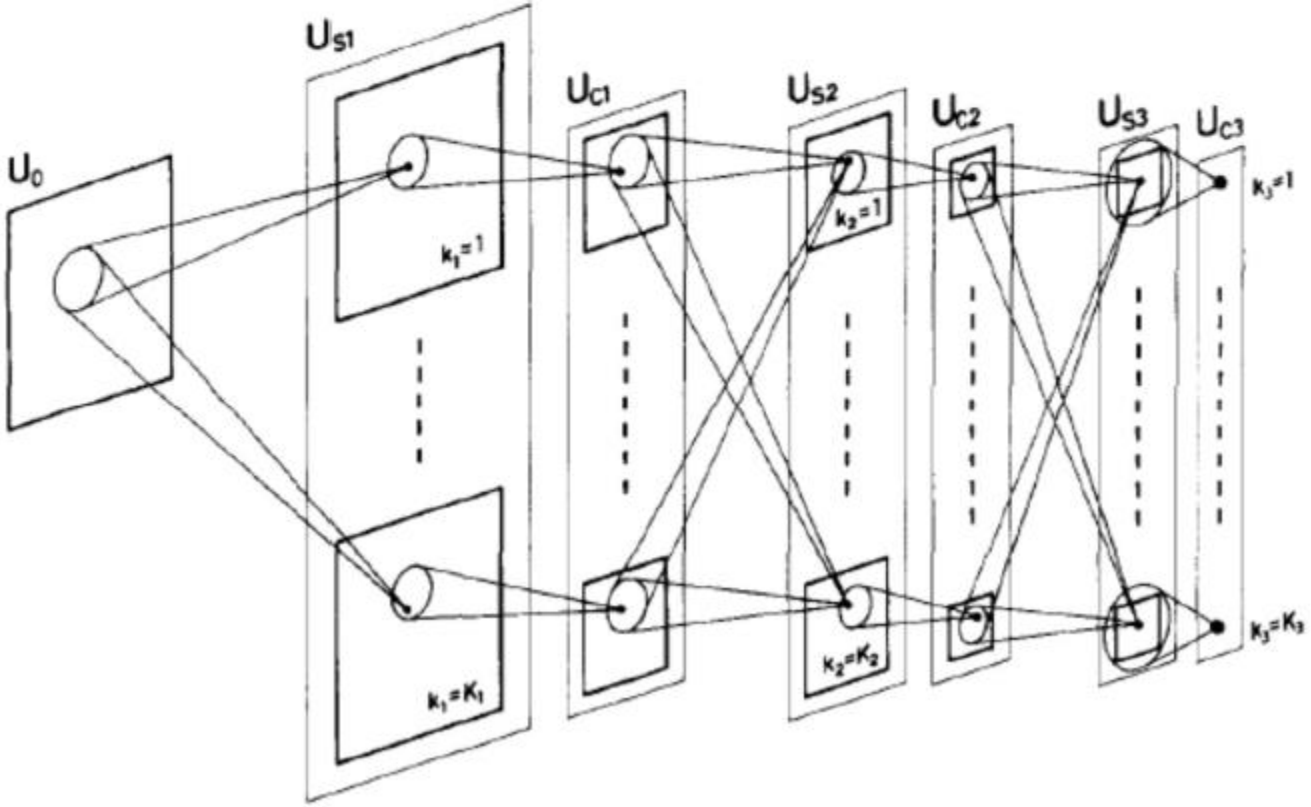
- Modified Hubel-Wiesel model for position invariance
- S-cells: learnable
  - Learns to respond to input
- C-cells: fixed synapses
  - Learns to confirm
- Hierarchical Model: train of C-S





Model

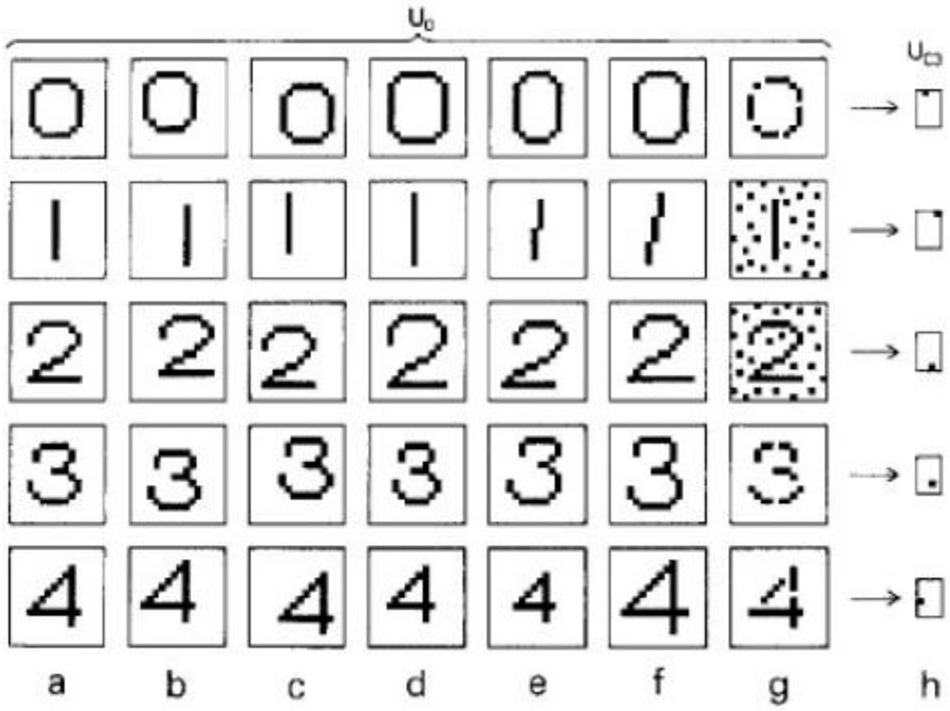
- S-cells: ReLU like activation
- C-cells: ReLU like with inhibitory bias
    - Only a strong combination of s-cells causes it to fire
    - More like a max operation
- Deeper layers have larger receptive fields
    - Downsampling
- Cell planes get smaller with each layer
    - Building complex features
- Number of planes increase with each layer
    - Number of features increases with each layer



NeoCognitron

- Unsupervised learning
- Randomly initialize S cells
- Hebbian learning updates: $\Delta w_{ij} = x_i y_i$
- Updates are distributed across all cells within the plane
  - Shared weights
- Within any layer, only the maximum S from all the layers is selected for update
  - Position invariance
- Largest max is selected from a single plane
  - Invariance to noise/fuzziness

NeoCognitron



Modifications

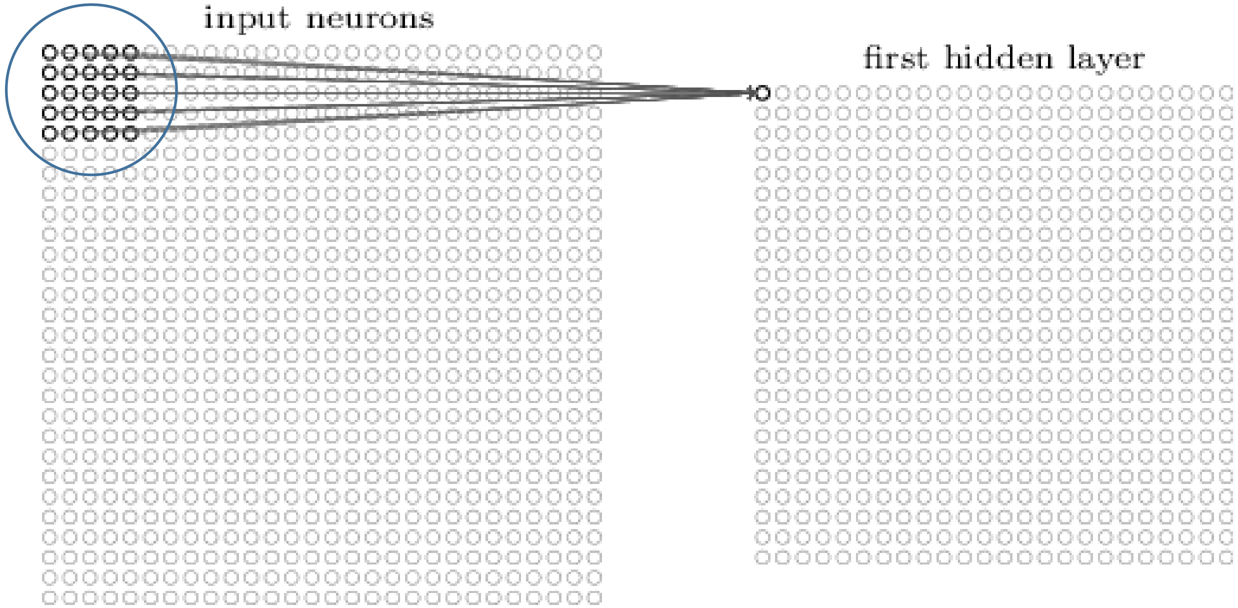
- Temporal correlation: Homma, Atlas, Marks (1988)
- Time-delayed NN: Weibel, Hanzawa, Hinton, Shikano (1989)
- Convolutional NN: Lecun (proposed 1989)

Convolutional Neural Networks

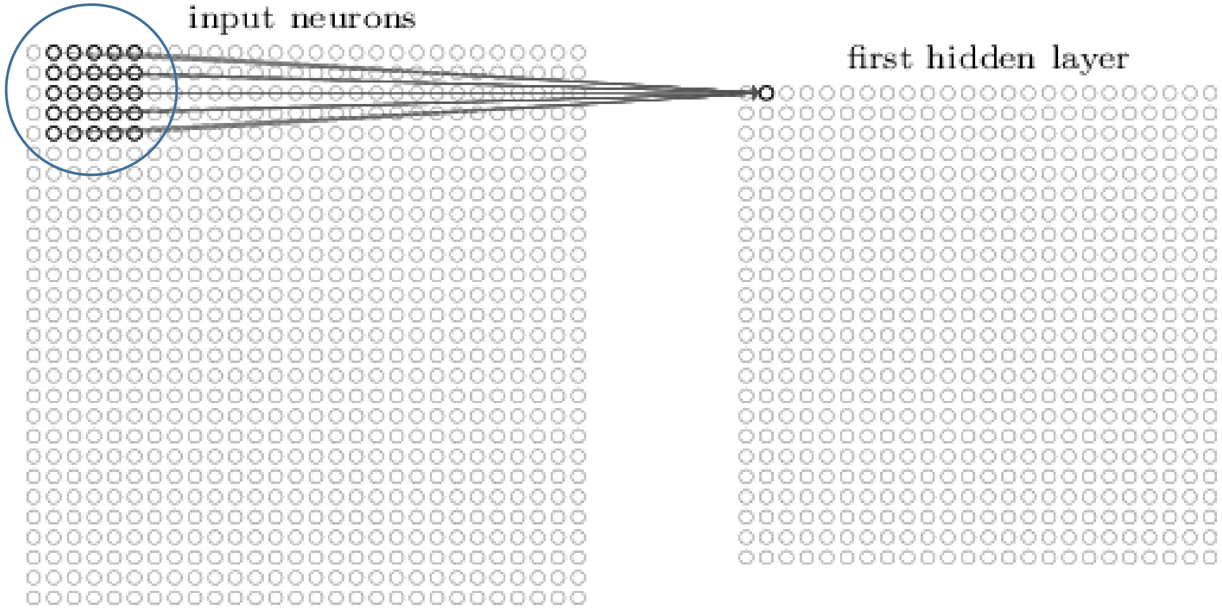
- Aka Covnet
- Biologically inspired: primary visual cortex
  - Simple neurons in early layers respond to specific patterns of light
  - Complex neurons are invariant to shift
  - “Grandmother” cells in deep layers
- Data in grid like topology
  - Well suited for image data
- Convolution is a linear operation
- Use 3 ideas:
  - Local receptive fields & Feature maps
  - Shared Weights
  - Pooling

Local Receptive Field

- Idea: exploit data in neighboring pixels
  - Image is a matrix
- Local receptive field: Sub regions of the input plane
  - Neurons connected spatially
  - Odd square sizes
- Sliding window
- *Stride length* measures distance between window centers
  - Represent a linear “down-sampling”



Local Receptive Field



Convolution

- 1D:

$$x(t) \star y(t) = \int x(a)y(t-a)da$$

$$x[i] \star y[i] = \frac{1}{n}\sum_{a=-\infty}^{\infty} x[a]y[i-a] = \frac{1}{n}\sum_{a=-\infty}^{\infty} x[i-a]y[i]$$

- 2D:

$$x[i,j] \star y[i,j] = \frac{1}{mn}\sum_{a}\sum_{b} x[a,b]y[i-a,j-b] = \frac{1}{mn}\sum_{a}\sum_{b} x[i-a,j-b]y[a,b]$$

- Correlation

$$x[i,j] \star y[i,j] = \frac{1}{mn}\sum_{a}\sum_{b} x[a,b]y[i+a,j+b]$$

- Covnets implement correlation

Padded Convolution, Stride = 1

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

https://towardsdatascience.com/semantic-image-segmentation-using-fully-convolutional-networks-bf0189fa3eb8

Padding, Stride

- Padding with zeros is used to make sure the output has the same dimension as the input image.
  - Add rows above the image = rows of the kernel/2 (integer division)
    - Same rows added below the image
  - Add columns left of the image = columns of the kernel/2 (integer division)
    - Same columns added right of the image
- Stride = 1 => preserves image size (default)
  - Stride > 1 => reduce the image size (linearly)

Unpadded, Stride = 1

60	113	56	139	85
73	121	54	84	128
131	99	70	129	127
80	57	115	69	134
104	126	123	95	130

Kernel

0	-1	0
-1	5	-1
0	-1	0

266		

Unpadded, Stride = 2

60	113	56	139	85
73	121	54	84	128
131	99	70	129	127
80	57	115	69	134
104	126	123	95	130

Kernel

0	-1	0
-1	5	-1
0	-1	0

266	

Convolution Algorithm

```
Given image, I, with size MxN
Given kernel, k, size is mxn
Initialize the output: Conv = zeros(M-m/2*2,N-n/2*2)
for r = m/2 to M-m/2                                  //non-padded. Integer division.
        for c = n/2 to N-n/2
                for i = r-m/2 to r+m/2                //compute weighted sum
                        for j = c-n/2 to r+c/2
                                conv(r-m/2,c-n/2) = conv(r-r-m/2,c-n/2) + k(i,j)*I(i,j)
conv = conv/(m*n)
```

Convolution Algorithm Similar to Padding

```
Given image, I, with size MxN
Given kernel, k, size is mxn
Initialize the output: Conv = zeros(M,N)
for r = 0 to M-1
        for c = 0 to N-1
                for i = -m/2 to m/2-1                    //compute weighted sum
                        for j = -n/2 to n/2-1
                                if ((r+i>= 0) AND (r+i<M)) AND (((c+j>= 0) AND (c+j<N)))
                                        conv(r,c) = conv(r,c) + k(r+i,c+j)*I(2+i,2+j)
conv(r,c) = conv(r,c)/(m*n)
```

Feature Map

- The output of the convolution
- Convolution is applied over all channels
- Given:

  $M \times N \times C$ input: rows x columns x channels

  $h \times w \times C$ filter: rows x columns x number of filters

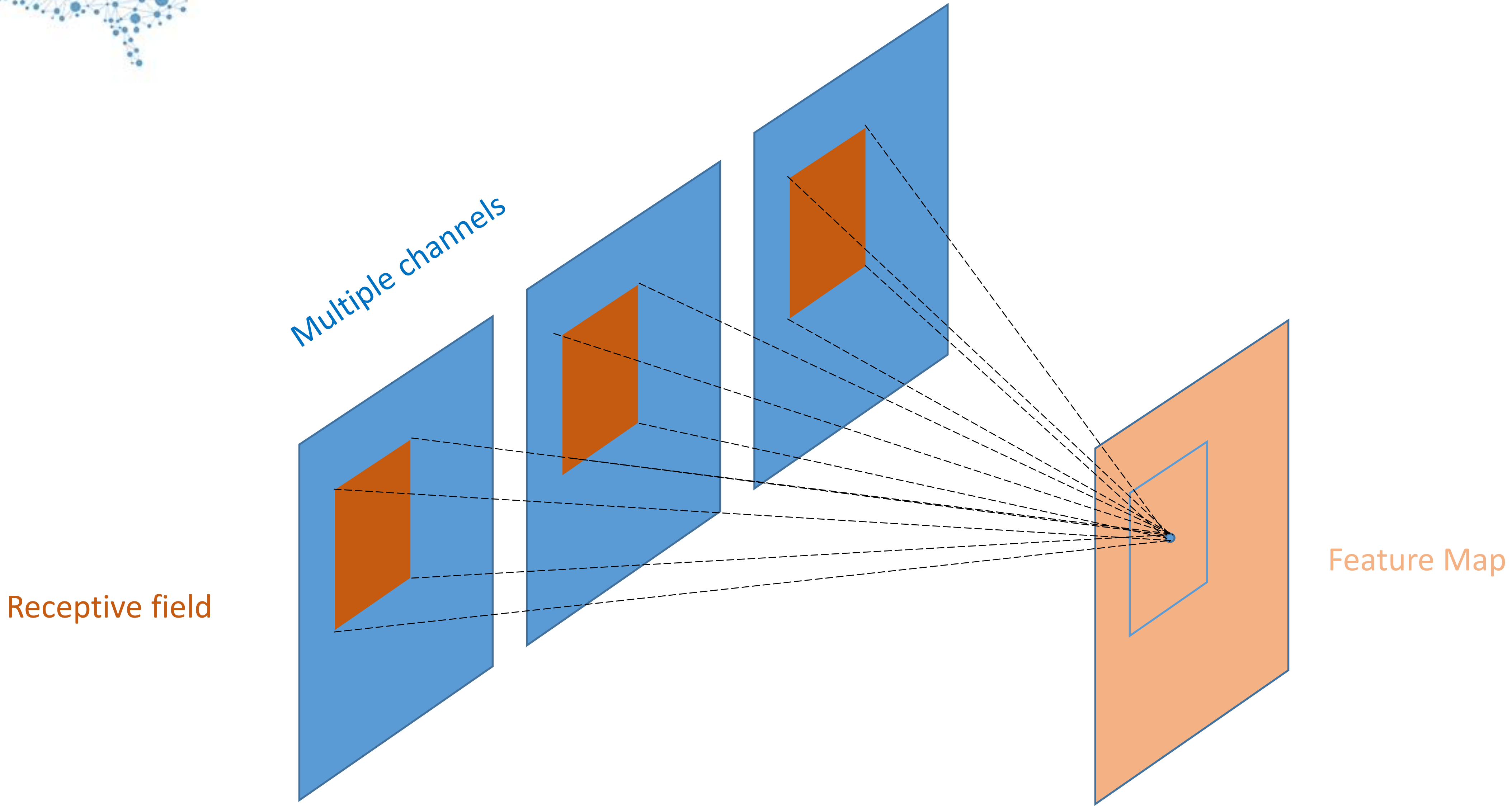
  n filters

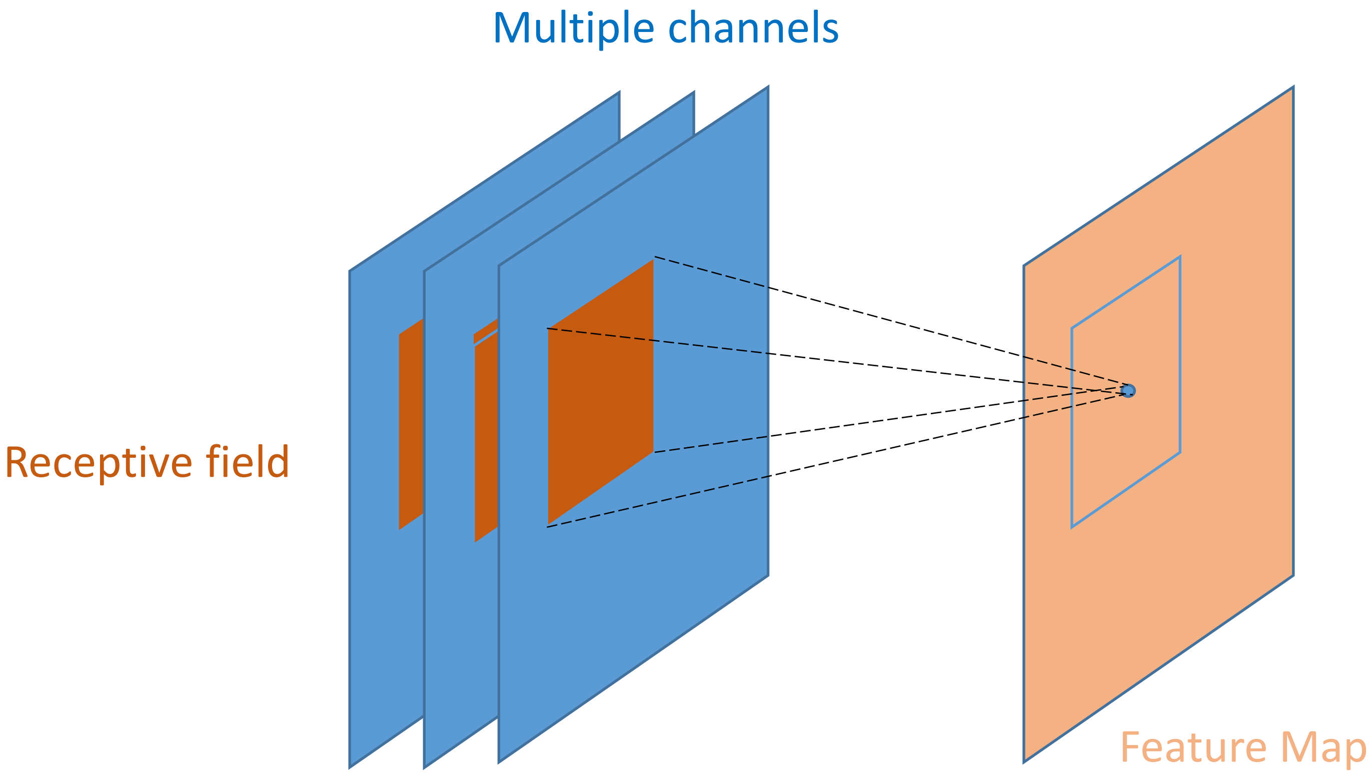
  s stride

  - Unpadded Feature Map size: $\left(\frac{M-h}{s}\right) \times \left(\frac{N-w}{s}\right) \times n$

  - Padded Feature Map size: $\left(\frac{M}{s}\right) \times \left(\frac{N}{s}\right) \times n$
  - Channel can be R,G,B or filtered output from previous layer

Convolution over Channels



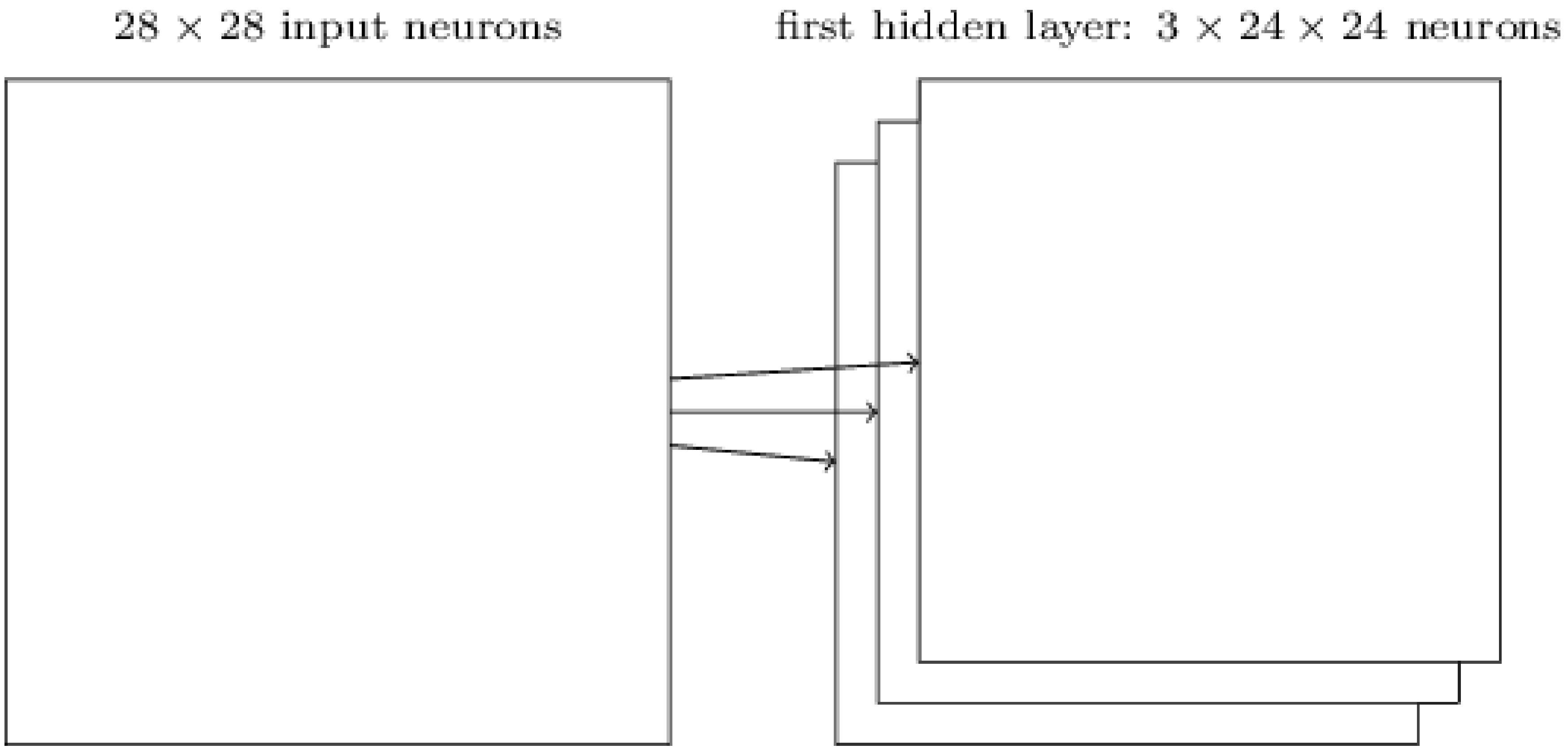

Multiple channels
Receptive field
Feature Map

Why Convolution ?

- Convolution can be used to filter the image
  - Pixel enhancements, e.g. detect edges, denoise, etc.
  - Detect gradients, e.g. 1st order gradients, 2nd order gradients (aka Laplacians)
  - Detect objects: object you want to detect becomes your kernel

Shared Weights

- Idea: in each layer “look” for the same feature
  - E.g. vertical lines
- Hidden neuron connects to 1 receptive field
  - Weight for each input, one bias
- Each hidden neuron in a layer has the same weights and bias
- Shared weights and biases: kernel or filter
  - Convolution
- Hidden layer: feature map
  - Convolution layer
- To process multiple feature maps/kernels: generate parallel hidden layers
- Shared parameters => conservation of parameters => faster learning
- For $j \times k$ receptive field and l hidden layers: number of parameters = $(jk + 1)l$
  - E.g. 5x5 field and 3 hidden layers => 78 parameters

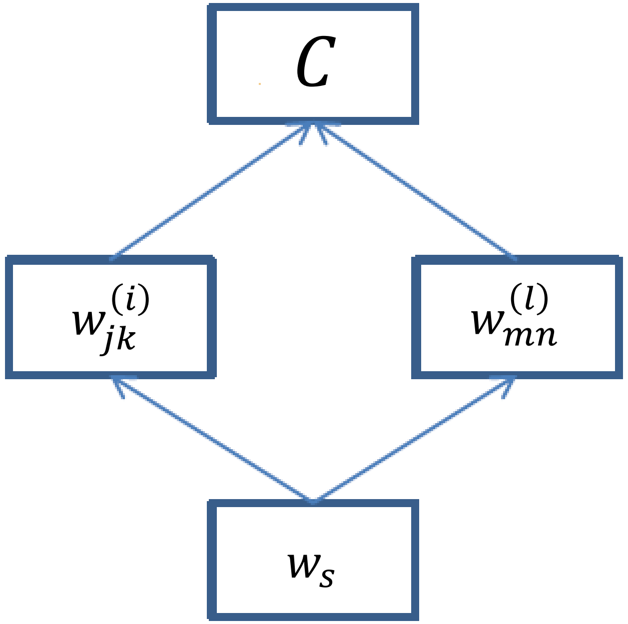

28×28 input neurons
first hidden layer: $3 \times 24 \times 24$ neurons

Backrpop for Shared Weights

$$w_{jk}^{(i)} = w_{mn}^{(l)} = w_s$$

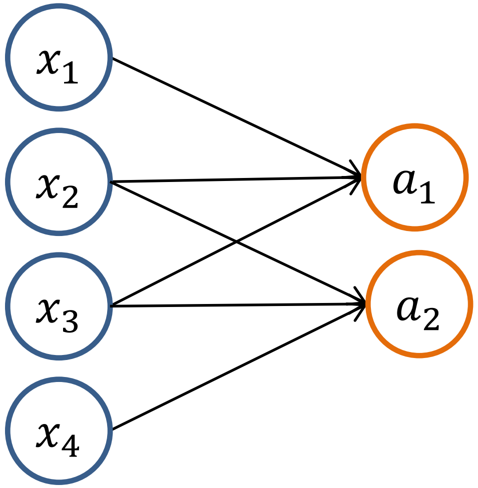
$$\frac{\partial C}{\partial w_s} = \frac{\partial C}{\partial w_{jk}^{(i)}} \frac{\partial w_{jk}^{(i)}}{\partial w_s} + \frac{\partial C}{\partial w_{mn}^{(l)}} \frac{\partial w_{mn}^{(l)}}{\partial w_s}$$

$$= \frac{\partial C}{\partial w_{jk}^{(i)}} + \frac{\partial C}{\partial w_{mn}^{(l)}}$$



- For shared weight, $w_{jk}^{(i)}$:

$$\frac{\partial C}{\partial w_{jk}^{(i)}} = \sum_{paths} \frac{\partial C}{\partial w_{jk}^{(i)}} = \frac{\partial C}{\partial a_{path}} \sum_{paths} \frac{\partial a_{path}}{\partial w_{jk}^{(i)}} = \frac{\partial C}{\partial a_{path}} \sum_{paths} inputs^{(i-1)}$$



$$a_1 = g(Z_1) = g(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3)$$

$$a_2 = g(Z_2) = g(w_0 + w_1 x_2 + w_2 x_3 + w_3 x_4)$$

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial a_1}\frac{\partial a_1}{\partial w_i} + \frac{\partial C}{\partial a_2}\frac{\partial a_2}{\partial w_i}$$

$$\frac{\partial C}{\partial w_1} = \delta\, a_1' x_1 + a_2' x_2$$

Activation

- ReLU activation typically applied to feature maps
- Blocks negative values, keeps positive values.

Pooling

- Idea: Compress the feature map
- Look for measure in a $u \times v$ region of feature map
  - Typically 2x2: keeps image size even
  - Stride = 2: reduce feature map size by half
  - Even image dimensions required, power of 2 preferred.
  - Stop pooling when you hit odd size
- Max-pooling: find the max value in a $u \times v$
  - Keep interest points not their position
  - More invariant to spatial shifts
- L2 pooling: find square root of the sum of the squares in a $u \times v$
  - Average out feature response
- For a $m \times n$ feature map and $v \times w$ pooling, pooling layer is $\frac{m}{u} \times \frac{n}{v}$
  - E.g.: 24x24 feature map and 2x2 pooling, layer is 12x12

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

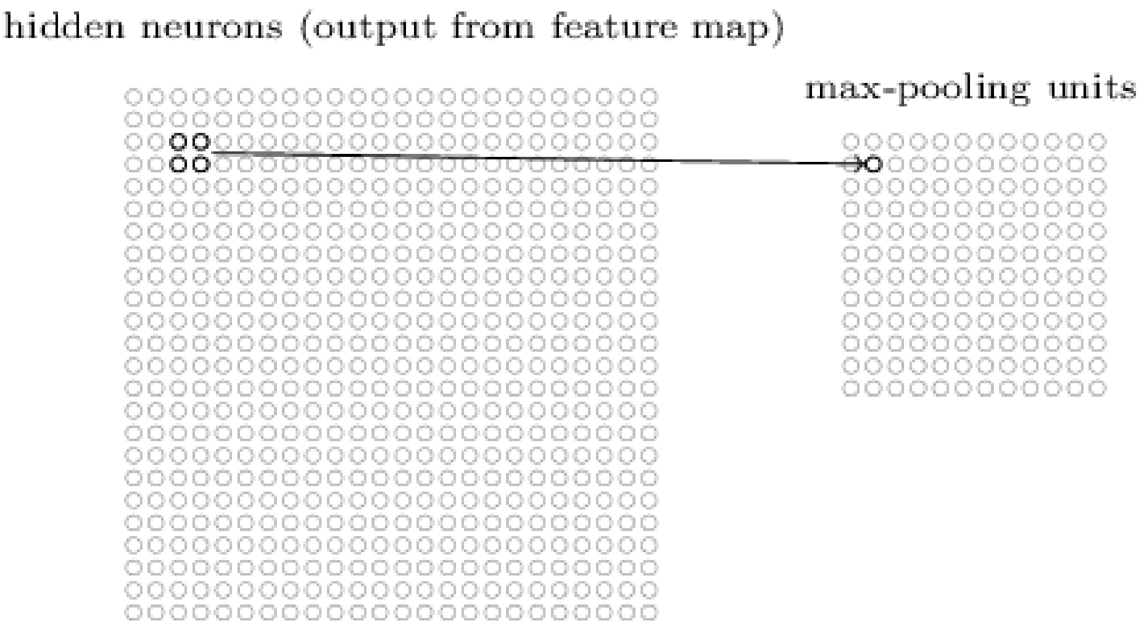
max pool with 2x2 filters and stride 2 $\longrightarrow$

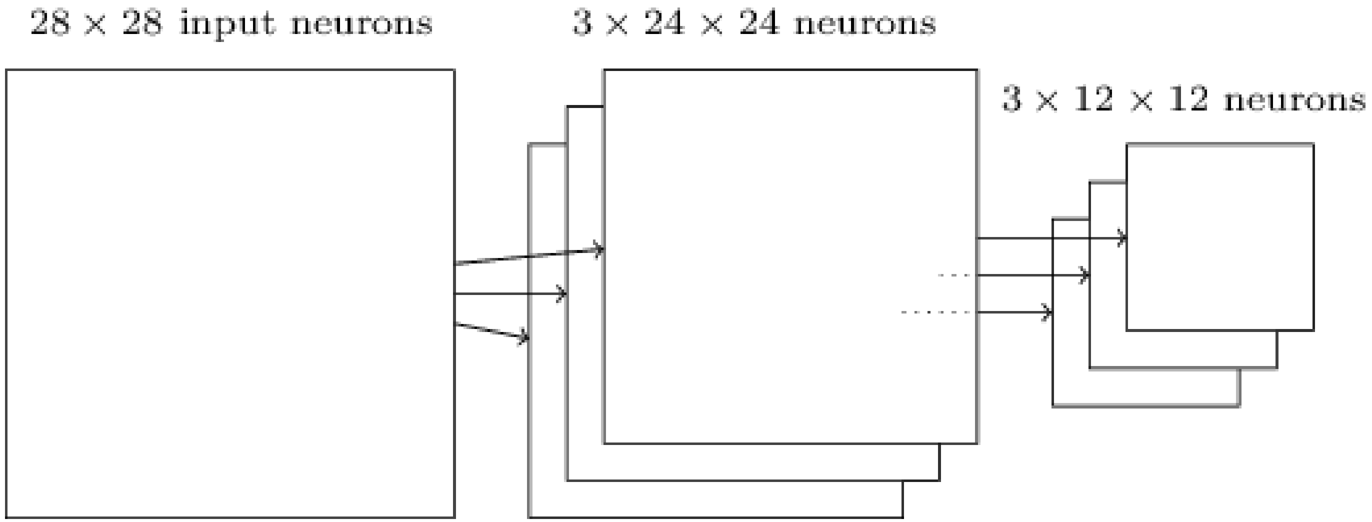
6	8
3	4

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Mean pool with 2x2 filters and stride 2 →

3.25	5.25
2	2


hidden neurons (output from feature map)
max-pooling units


28 × 28 input neurons
3 × 24 × 24 neurons
3 × 12 × 12 neurons

Max Pooling Algorithm

```
Given image, I, with size MxN
Given kernel, k, size is mxn (typically 2x2)
Given a stride, s (typically s=2)
Initialize the output: pooling = zeros(M/s,N/s)
for r = 0 to (M-1)-m, step=s
        for c = 0 to (N-1)-n, step=s
                max = I(r,c)                                        //initialize max
                for i = 1 to m-1
                        for j = 1 to n-1
                                if max < (I(r+i, c+j))
                                        max = I(r+i,c+j))
                pooling(r/s,c/s) = max
```

Pooling Alternatives

- Pooling is a down-sampling process
  - Reduces area
  - Size/location invariance
- Pooling: learned filter
  - Shared parameter network
- Pooling: convolution layer
  - Stride > 1 for downsamping
  - Averaging is a flat filter convolution

Backprop Pooling

- Max pooling:

$$a = g(Z) = \max(\forall_i\ x_i) = \begin{cases} x_n & \text{if } x_n \text{ is max} \\ 0 & \text{others} \end{cases}$$

$$\frac{\partial a}{\partial x_i} = \begin{cases} 1 & \text{for } x_n \text{ is max} \\ 0 & \text{others} \end{cases}$$

- L2 pooling:

$$a = g(Z) = \|X\|_2^2 = \left(\sqrt{\sum_i x_i^2}\right)^2 = \sum_i |x_i|^2$$

$$\frac{\partial a}{\partial x_i} = 2x_i$$

Additional Layer

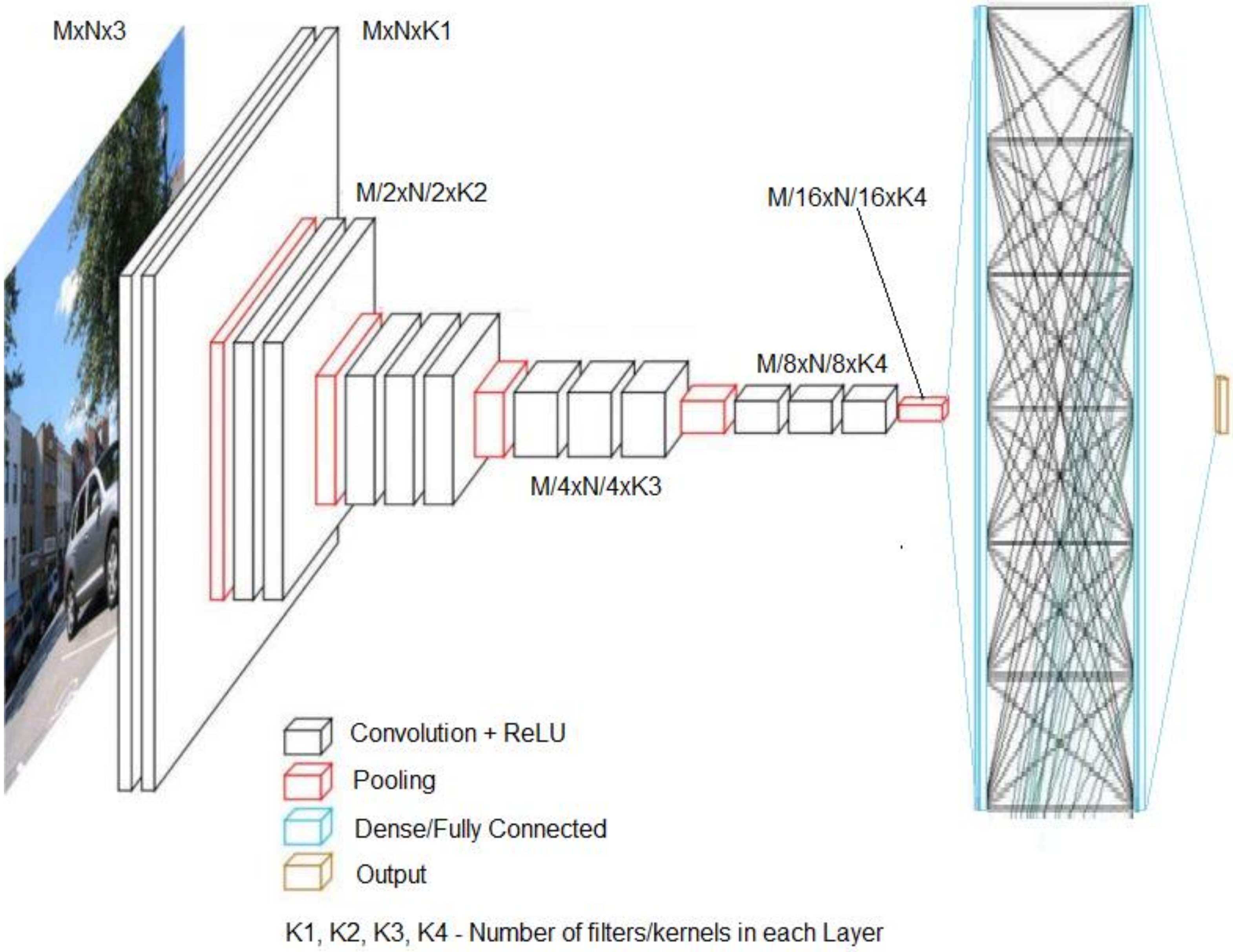
- Output layer is needed
- Can add a other hidden layer neurons
- Fully connected layer
- For classification, use softmax

CNN Architecture

- X: input, Y: output
- Hidden Layers consist of:
  - Convolutional layer
  - ReLU
  - Pooling
- Each hidden layer will reduce size (typically by half)
- After several hidden layers use a dense fully connected network:
  - Flatten all the pixels into a column vector
  - Use a fully connected network
- Output layer:
  - Linear/sigmoidal/softmax

Typical CNN Architecture

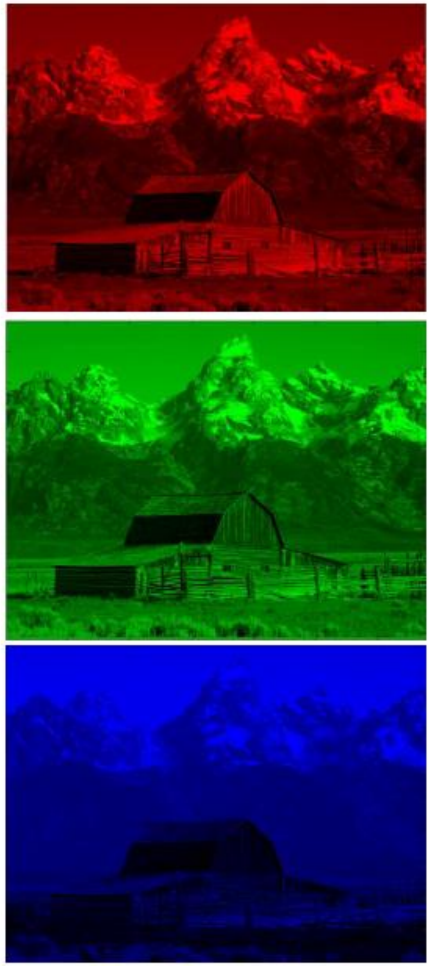
- Convolution-ReLU for each block
  - Generates feature maps
- Pooling applied to down-sample size
- Width of block = number of filters used
  - Width = feature maps generated.
- Each block = size of feature map x filters used
- Flattened input is a column row of last block
- Dense network: fully connect each input toa hidden layer
  - Can have multiple hidden layers
  - Sigmoid/tanh/ReLu typical activations
- Goal of training: optimize kernels that produce the best output



Modified from
https://neurohive.io/wp-content/uploads/2018/11/vgg16-neural-network.jpg

Practical Considerations: Color

- Images are colored
  - 3D matrix: RGB matrices
- Grayscale conversion
- Color is a channel on input
- Convert into other color models



Practical Considerations: Size

- Even size images must be used
  - Crop/resize
  - Pad when needed
  - Must preserve even size as we pool/down-sample (except for the last down-sample)
- Larger images don’t increase the number of parameters
  - Number of parameters = number of kernels * size of each kernel
  - They will require more time to filter

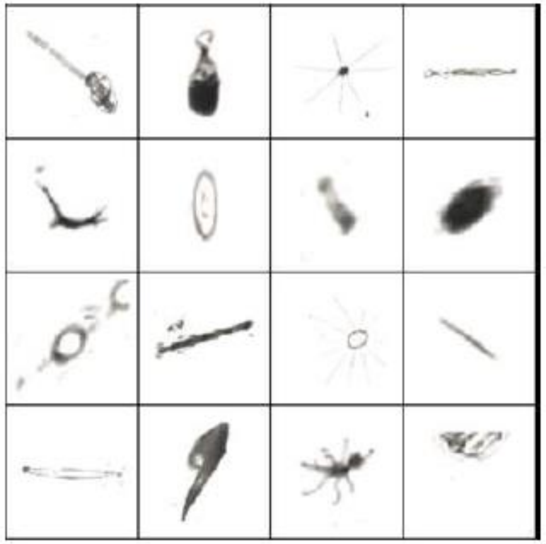
Practical Considerations: Kernels

- Number of filters/kernels
  - Number of feature Maps produced by each convolutional layer
  - More kernels = more weights = more parameters = more robust system
  - Large number of kernels at the early layers = more processing time
  - Undesirable: Early layers have large image size
  - More desirable: add kernels to the down-sampled layers
- Kernel size
  - should be odd size (centered)
  - Recent research use 3x3 (popularized by VGG net)

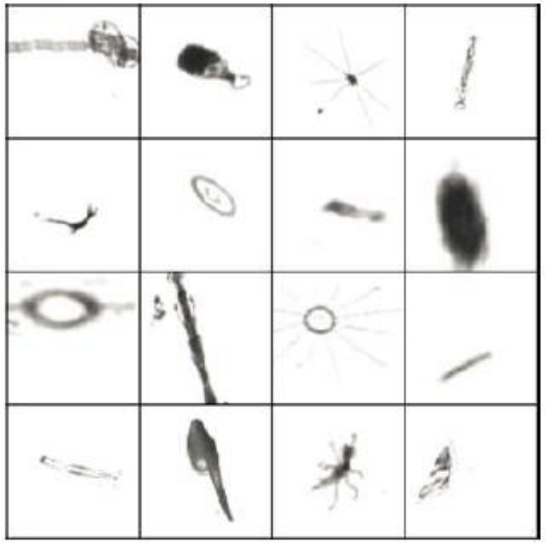
Practical Considerations: Training Data

- Design purpose: invariance to scale, position, rotation
- Limitation in data => limitation in invariance
- Synthetically augment data by random modifications:
  - Rotation: 0 to 360 (uniformly)
  - Translation: -10 to 10 pixels
  - Scaling: 1/1.6 to 1.6 (log-uniform)
  - Shearing: -20 to 20
  - Stretching: 1/1.3 to 1.3 (log uniform)

Original data



Augmented data



Augmenting: Image Modifications

- Modification applied as a linear operation to image coordinates
- Given input coordinates (v, w) and output coordinates (x, y):

$$(x, y) = T(v, w)$$
$$x = t_{11}v + t_{12}w + t_{13}$$
$$y = t_{21}v + t_{22}w + t_{23}$$

- Matrix form:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{12} & t_{22} & t_{23} \\ t_{12} & t_{32} & t_{33} \end{bmatrix} \begin{bmatrix} v \\ w \\ 1 \end{bmatrix}$$

- Scale: $c_x, c_y > 1 \Rightarrow$ expansion

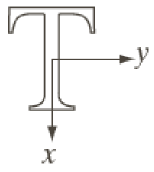
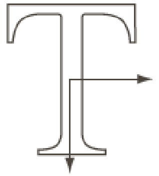
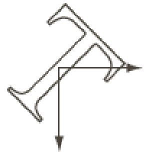
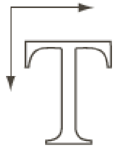
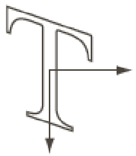
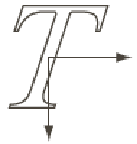
  $0 < c_x, c_y < 1 \Rightarrow$ contraction

- Rotation angle is CCW

- Translation (movement) by pixels

- Shear by pixels

TABLE 2.2
Affine transformations based on Eq. (2.6.–23).

Transformation Name	Affine Matrix, T	Coordinate Equations	Example
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = w$	
Scaling	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = c_x v$ $y = c_y w$	
Rotation	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v\cos\theta - w\sin\theta$ $y = v\cos\theta + w\sin\theta$	
Translation	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$	$x = v + t_x$ $y = w + t_y$	
Shear (vertical)	$\begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v + s_v w$ $y = w$	
Shear (horizontal)	$\begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = s_h v + w$	

Modified rom DIP Gonzalez & Woods

Augmentation Algorithm: Forward mapping

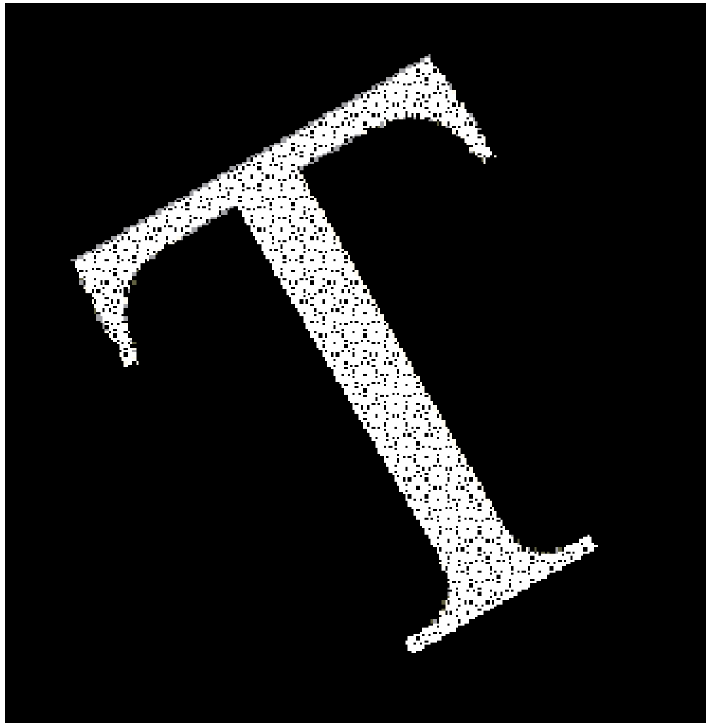
- Forward mapping: $(x, y) = T(v, w)$
- Given a translation matrix
  - Multiply each input coordinate pair (v,w) in the input with translation matrix to calculate (x,y)
  - Assign output(x,y) = input(v,w)

Forward Mapping Algorithm

```
Given image, I, with size MxN
Initialize out = zeros(M,N)
Given a transformation parameters (cx,cy,tx,ty,θ,sv,sh)
Create a transformation matrix T
for v = 0 to (M-1)
        for w = 0 to (N-1)
                x,y,1 = round(T*[v;w;1])                //dot product
                if (x>=0 AND x<=M) AND (y>0 AND y<M)
                        out(x,y) = I(v,w)
```

Augmentation Algorithm: Backward mapping

- Forward mapping problems:
  - Create gaps in output due to rounding
- Backward mapping: $(v, w) = T^{-1}(x, y)$



Backward Mapping Algorithm

```
Given image, I, with size MxN
Initialize out = zeros(M,N)
Given a transformation parameters (cx,cy,tx,ty,θ,sv,sh)
Create a transformation matrix inverse of T
for x = 0 to (M-1)
        for y = 0 to (N-1)
                v,w,1 = round(Tinv*[x;y;1])            //dot product
                if (v>=0 AND v<M) AND (w>=0 AND w<M)
                        out(x,y) = I(v,w)
```

Practical Considerations: Regularization and Depth

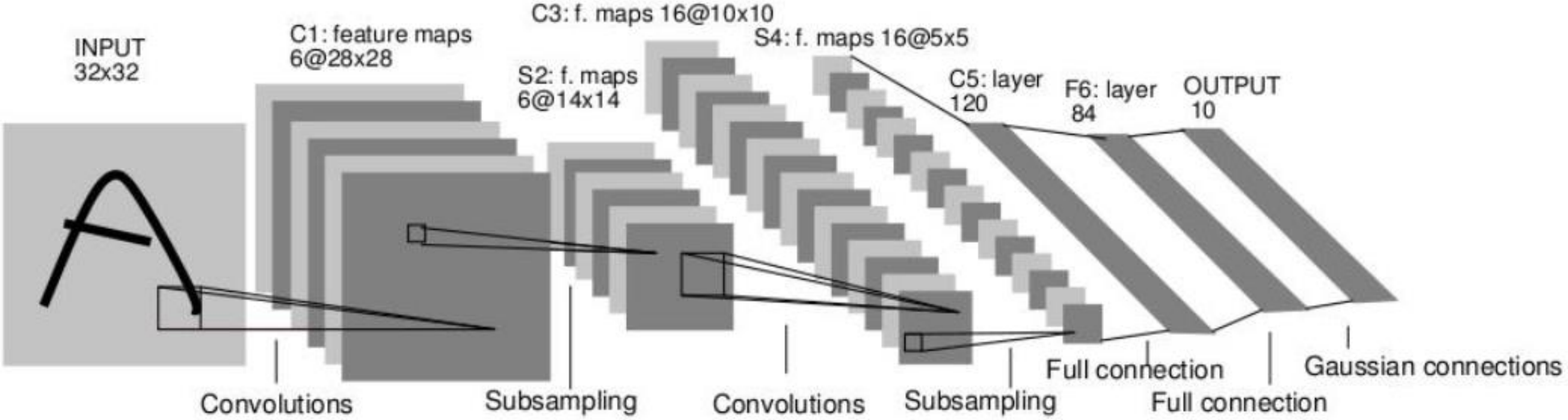
- Covnets reduce the number of parameters
  - Hidden neurons not fully connected
    - convolution/pooling layers
  - Shared weights
  - Reduction in the number of layers of covnet
- Dropout only applied to fully connected layers
  - Not convolution/pooling layers
- Very deep networks
  - 100+ layers
- Batch Normalization
  - Aka Layer normalization: applied to before every convolution
  - Normalization is a whitening operation: $\mu = 0,\ \sigma = 1$
    - Subtract the images in the batch by batch mean
    - Divide the image in the batch by batch standard deviation

Tensors

- Vector of varying-dimensional vectors
  - Similar to a class in math
- Problem of data dimensions:
  - Images are $row \times column \times color$
  - Conv layers needed for different feature scale
  - Conv layers needed for different stride (down-sample)
  - Data is divided into batches

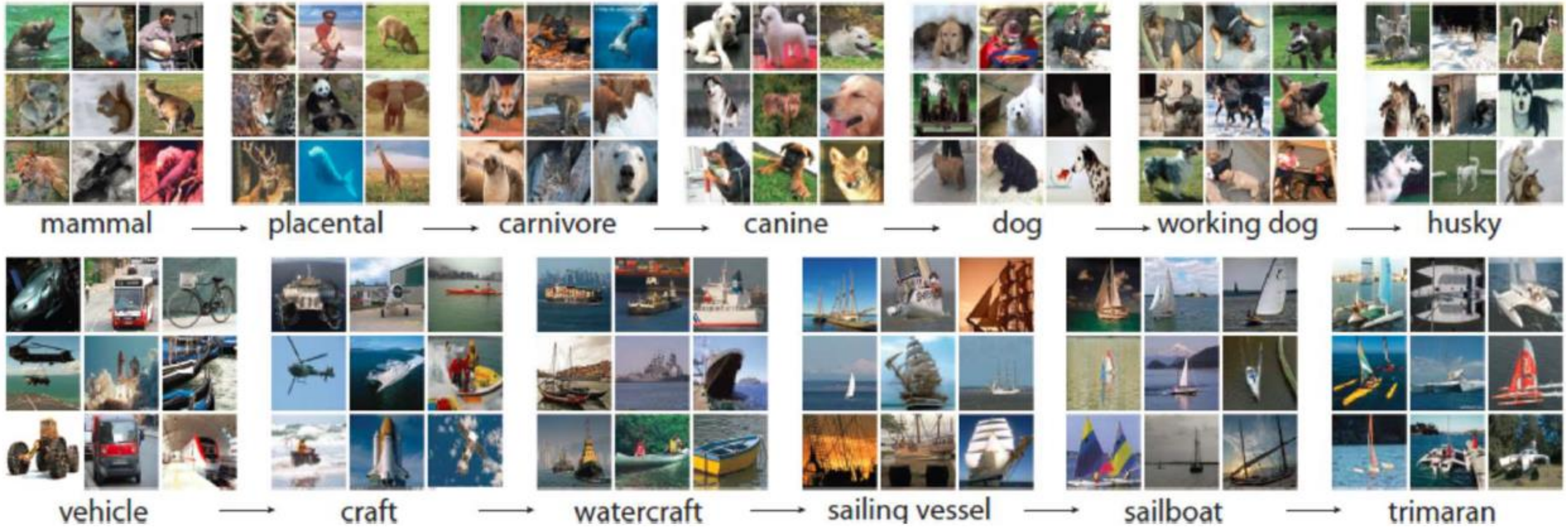
Architectures: Le-net5 (1998)

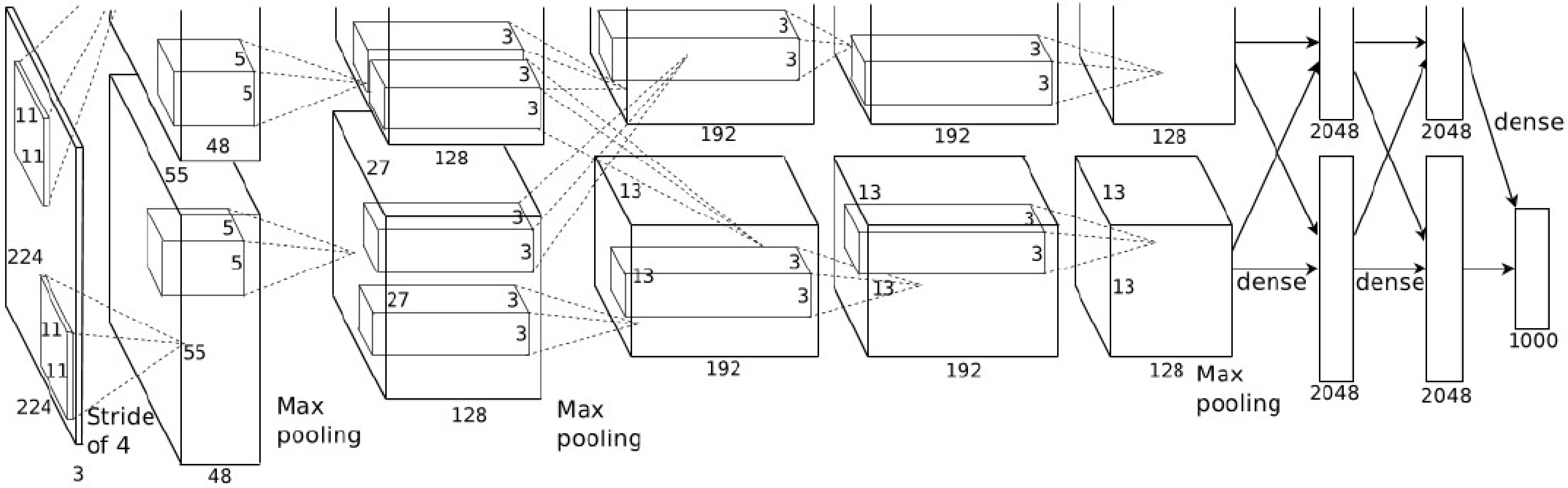
- http://yann.lecun.com/exdb/lenet/
- Digit recognition on MNIST (32x32 images)
- Conv1: 6 5x5 filters in first conv layer (no zero pad), stride 1
  - 6x 28x28 maps
- Pool1: 2x2 max pooling, stride 2
  - 6@14x14 maps
- Conv2: 16 5x5 filters in second conv layer, stride 1, no zero pad
  - 16@10x10 maps
- Pool2: 2x2 max pooling with stride 2 for second conv layer
  - 16@5x5 maps (400 values in all)
- MLP: 3 layers
  - 120 neurons, 84 neurons, and finally 10 output neurons


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

ALEXNET (2010)

- A. Krizhevsky, I. Sutskever, and G. Hinton
- Won Imagenet Large Scale Visual Recognition Challenge (ILSVRC)
  - http://www.image-net.org/challenges/LSVRC/
- Actual dataset: Many million images, thousands of categories




224
224
3
11
11
11
11
Stride of 4
55
55
48
48
5
5
5
5
Max pooling
27
27
128
128
3
3
Max pooling
13
13
192
192
128
Max pooling
dense
dense
dense
2048
2048
2048
2048
1000

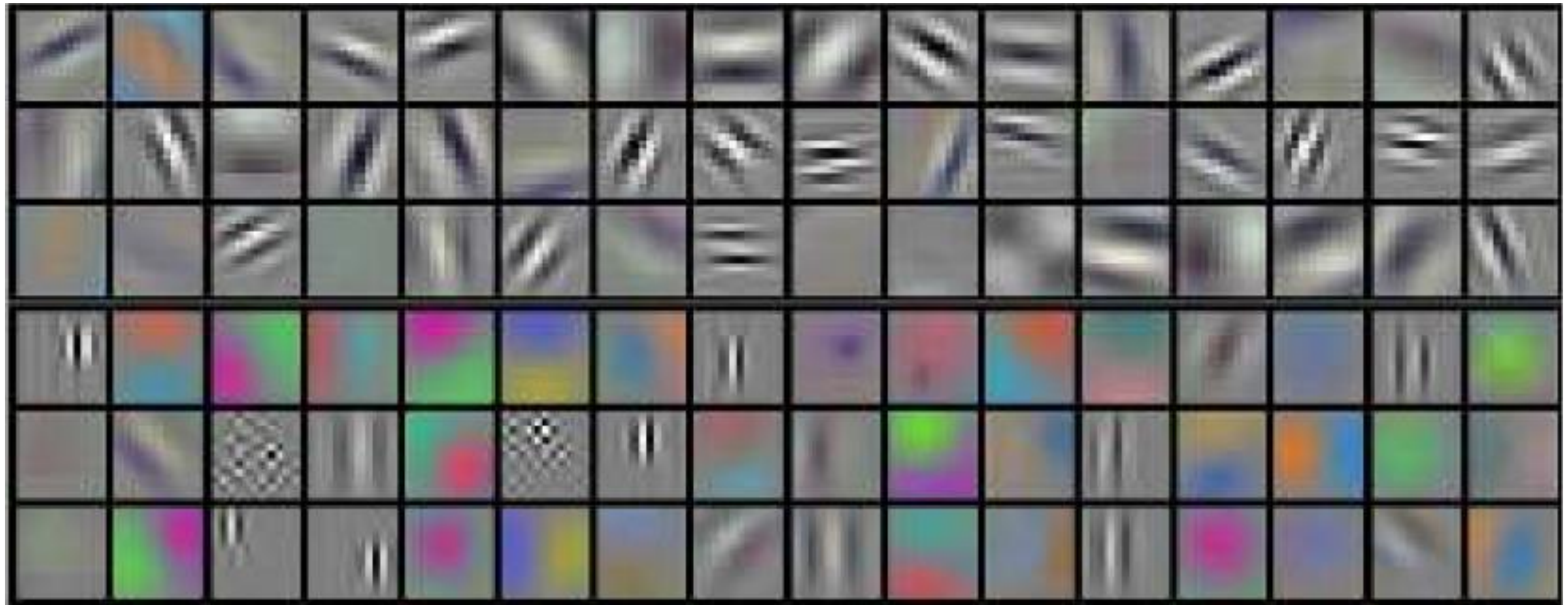
Alexnet Architecture

- Input: 227x227x3 images
- Conv1: 144@11x11 filters, stride 4, no zeropad
- Pool1: 3x3 filters, stride 2
- Normalization layer [Unnecessary]
- Conv2: 384@5x5 filters, stride 2, zero pad
- Pool2: 3x3, stride 2
- Normalization layer [Unnecessary]
- Conv3: 576@ 3x3, stride 1, zeropad
- Conv4: 576@ 3x3, stride 1, zeropad
- Conv5: 384@ 3x3, stride 1, zeropad
- Pool3: 3x3, stride 2
- MLP: 3 layers,
  - 6144 neurons, 6144 neurons, 1000 output neurons

Alexnet Settings

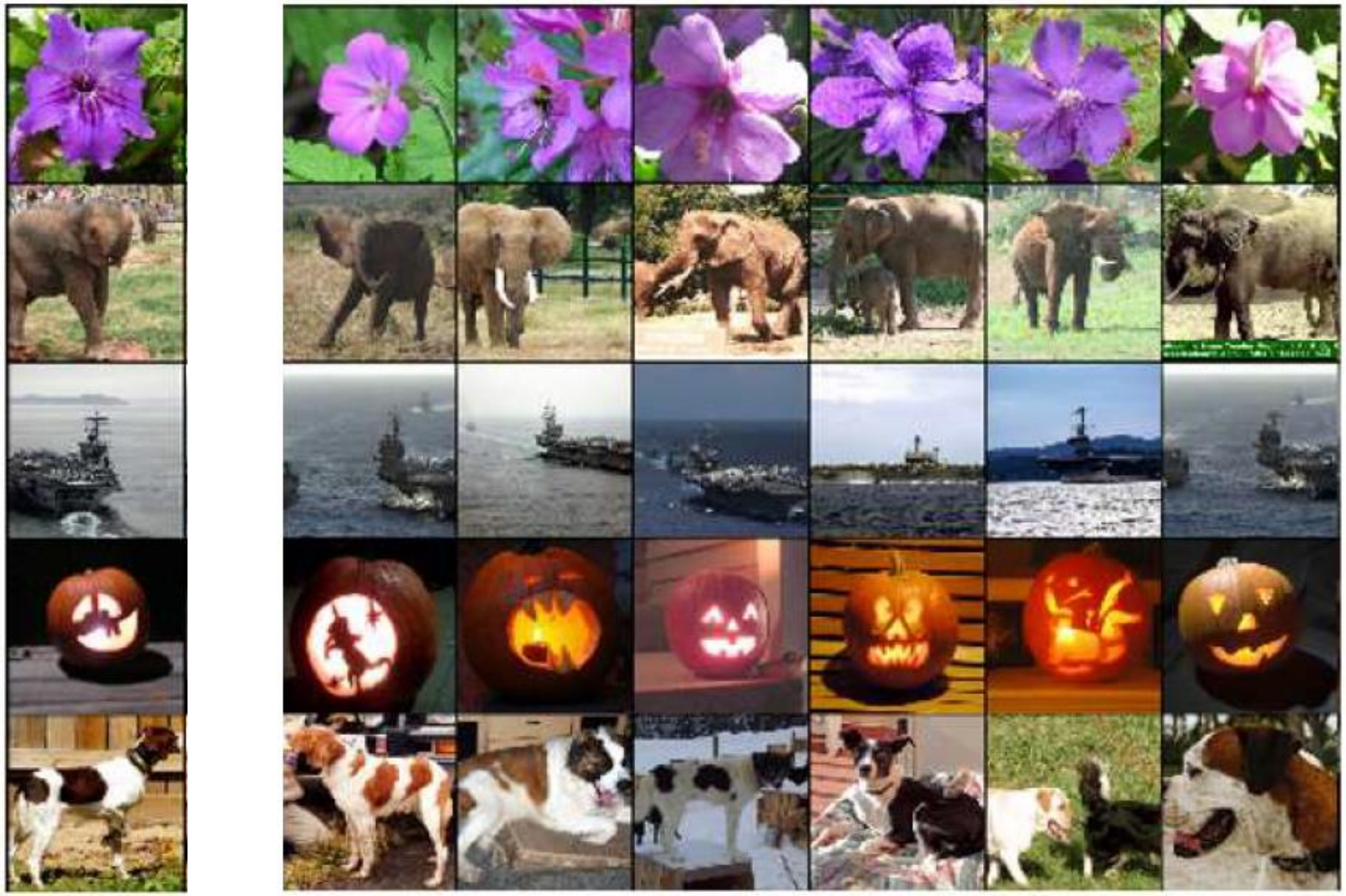
- “Dropout” – 0.5 (in FC layers only)
- Large amount of data augmentation
- SGD with mini batch size 128
- Momentum, with momentum factor 0.9
- L2 weight decay 5e-4
- Learning rate: 0.01, decreased by 10 every time validation accuracy plateaus
- Evaluated using: Validation accuracy
- Final top-5 error:
  - 18.2% with a single net,
  - 15.4% using an ensemble of 7 networks
  - Lowest prior error using conventional classifiers: > 25%

Alexnet Learned Filters



Automatic Labeling

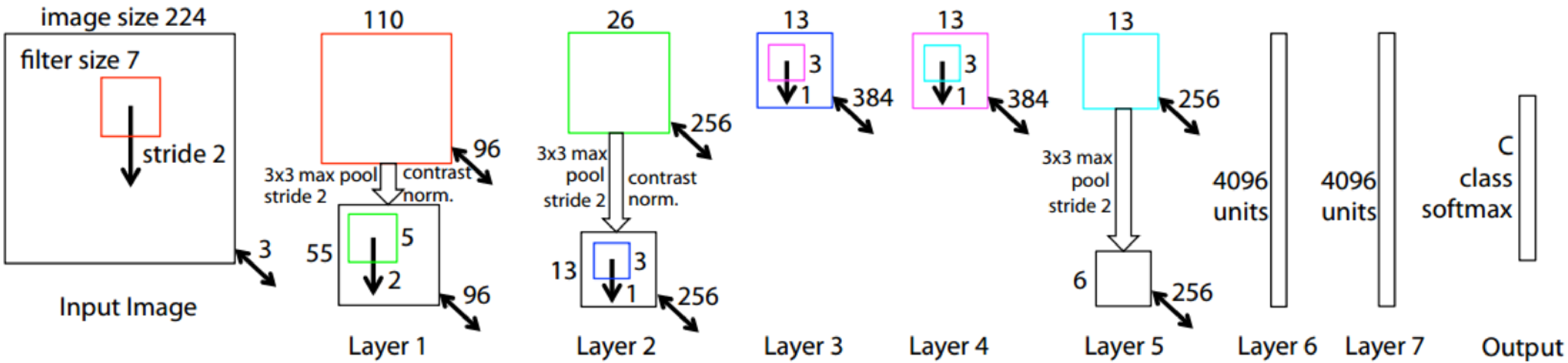
Matching



ZFNet (2013)

- Inspired by Alexnet: improved error from 15.4 to 14.8%
- Input: 224x224x3
- Conv1: 96@7x7, stride 2
    - Result: 96@110x110
- Pool1: 96@3x3, stride 2
    - Result: 96@55x55
- Conv2: 256@5x5, stride 2
    - Result: 256@26x26
- Pool2: 256@3x3, stride 2
    - Result: 256@13x13
- Conv3: 512@3x3, stride 1
    - Result: 512@13x13
- Conv4: 1024@3x3, stride 1
- Conv5: 512@3x3, stride 1
- MLP
    - 4096, 4096, 1000 softmax.

ZFnet Architecture



VGGNet (2014)

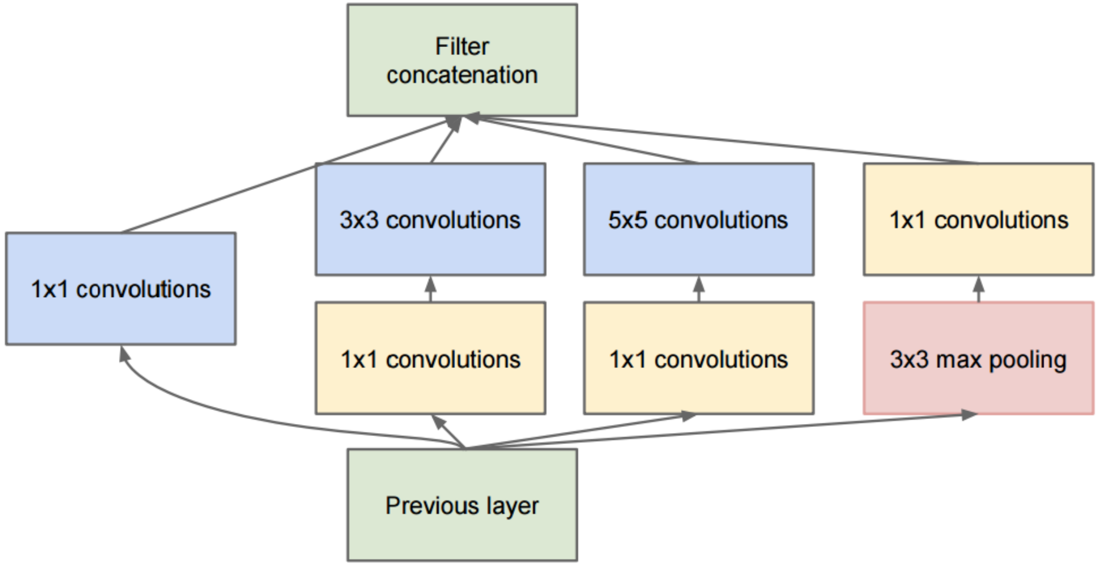
- Simonyan and Zisserman
- Used 3x3 filters, stride 1, pad 1
  - $n \times n \Rightarrow n^2 + 1$ parameters (weights + 1 bias)
- Only used 2x2 pooling filters, stride 2
- ReLU layer between successive cov layers
- Finally obtained 7.3% top-5 error using 13 conv layers and 3 FC layers
  - Combining 7 classifiers
  - Subsequent to paper, reduced error to 6.8% using only two classifiers
- Final arch: 682M parameters!

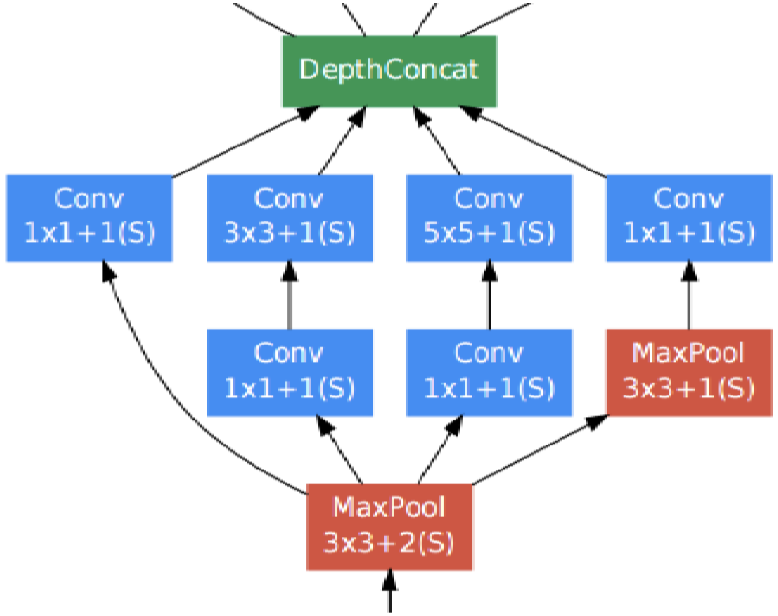
VGGnet Architecture

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					

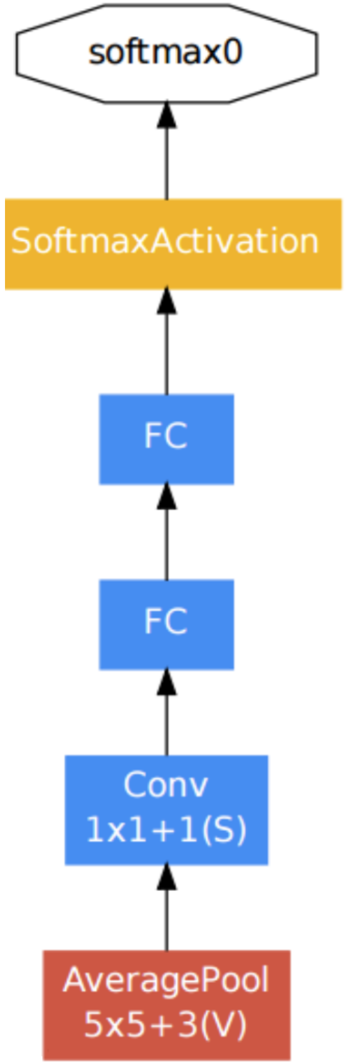
Googlenet: Inception

- ILSVRC 2014 winner
- Inception modules:
  - Parallel paths, different receptive field sizes
    - capture sparse patterns of correlations in the stack of feature maps
  - Use 1x1 convolutions for dimensionality reduction
    - Reduce parameters

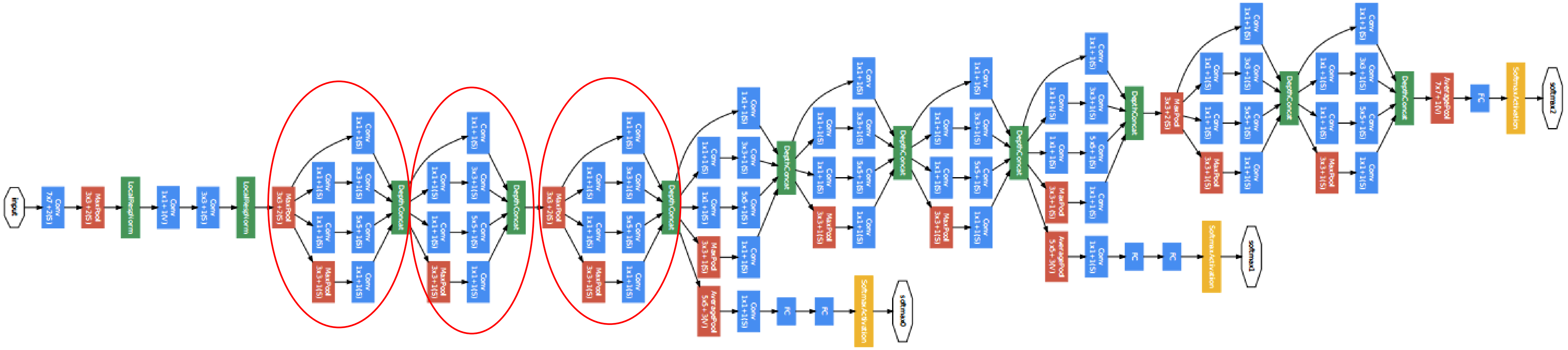




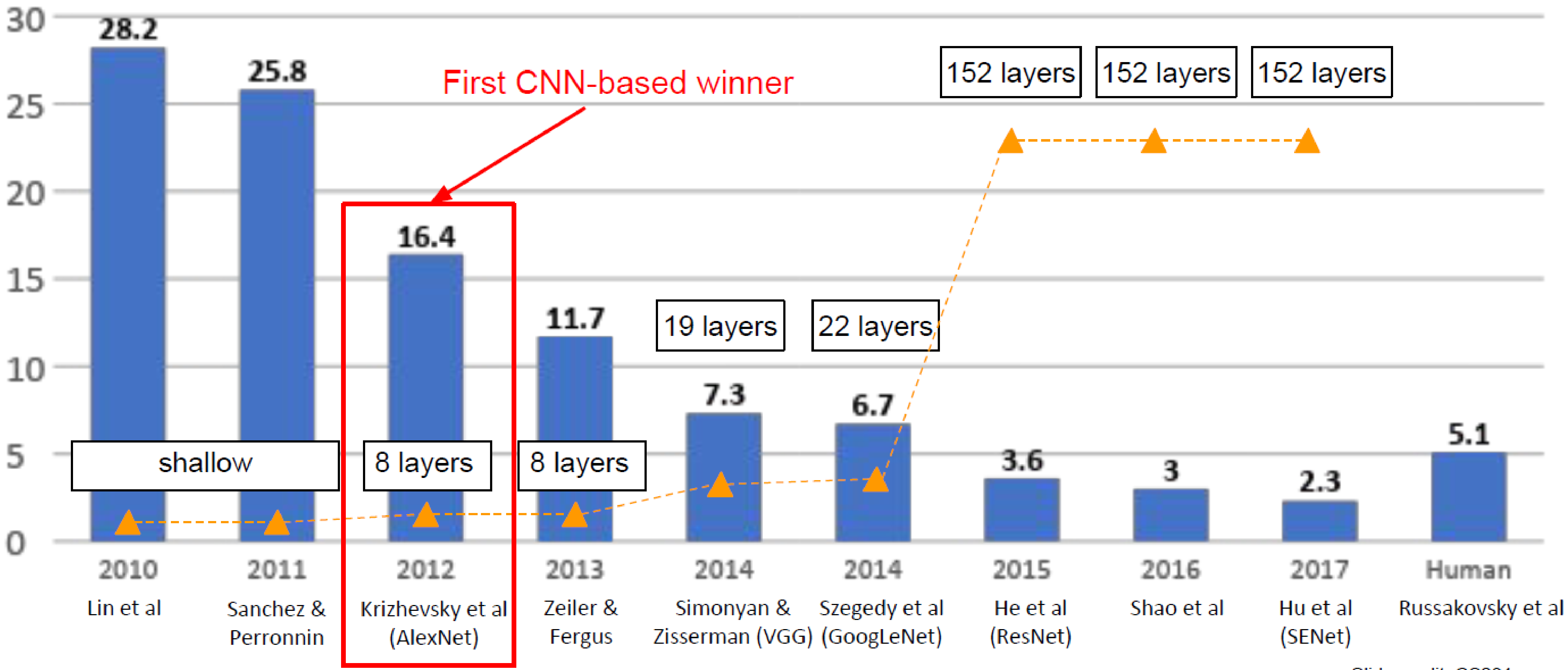
Googlenet: Auxliary Classifier



Googlenet: Architecture



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide credit: CS231n

Architectures: Test Your Own

- https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html