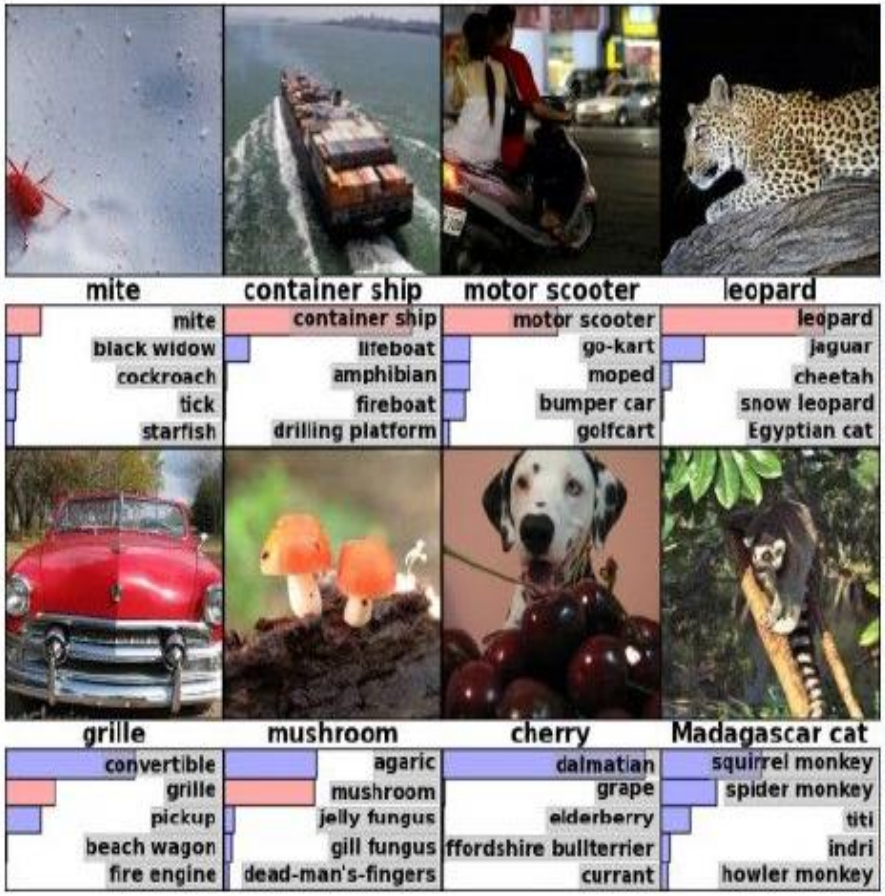
Convolutional Neural Networks

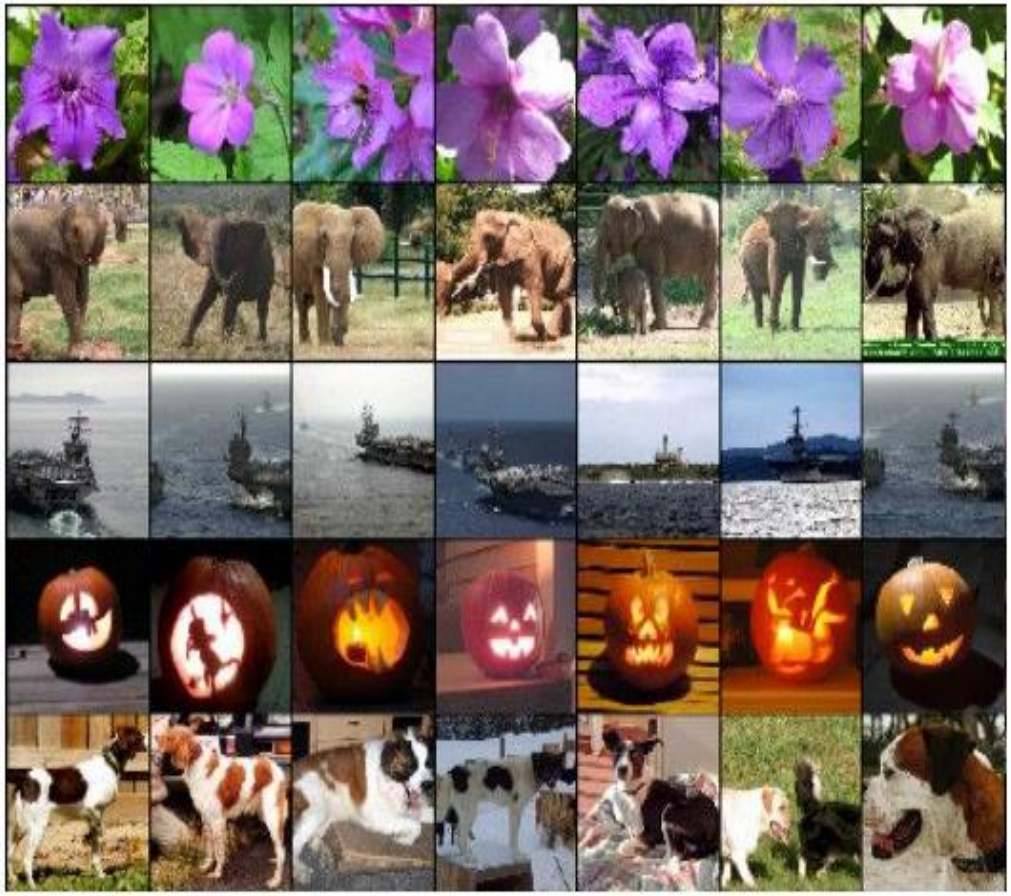
Saeed Sharifian

ConvNets

Classification

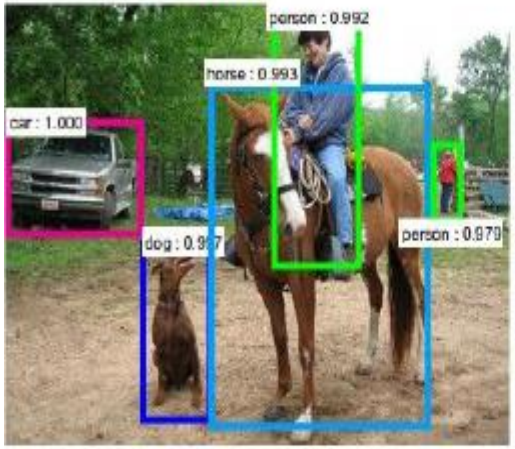


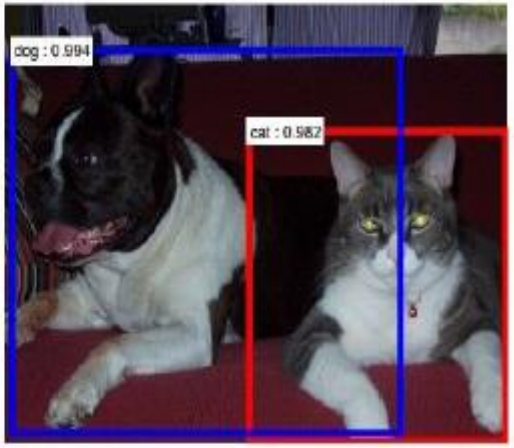
Retrieval

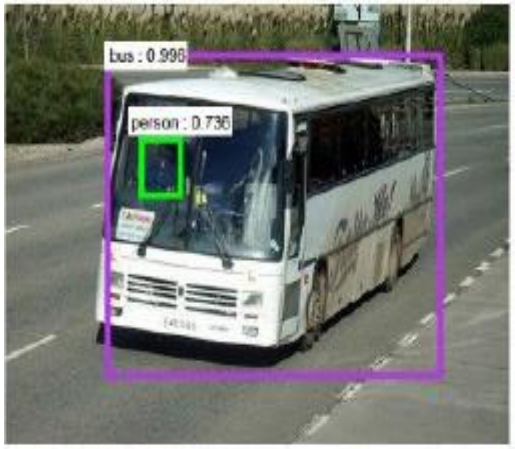


ConvNets

Detection



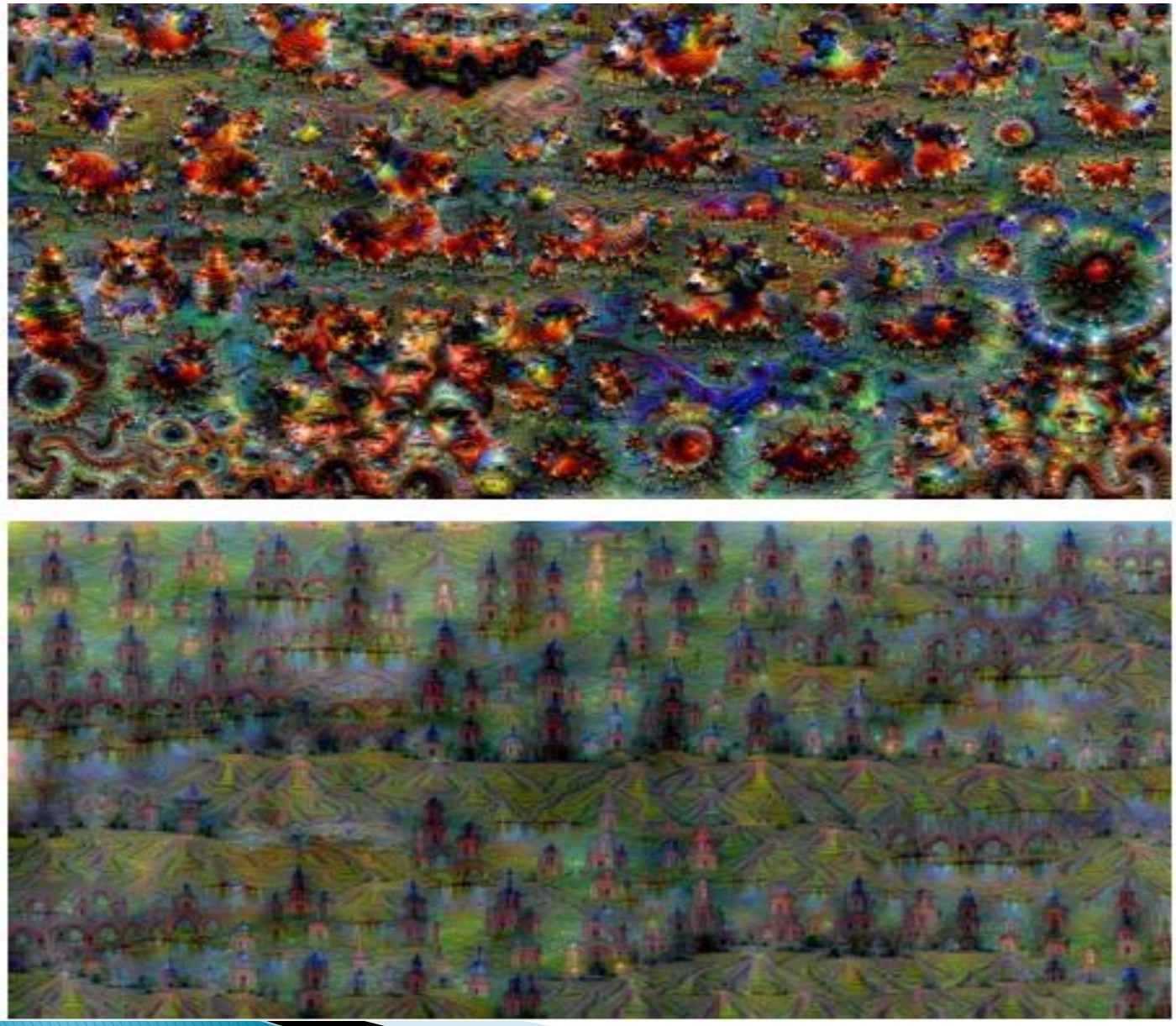




Segmentation

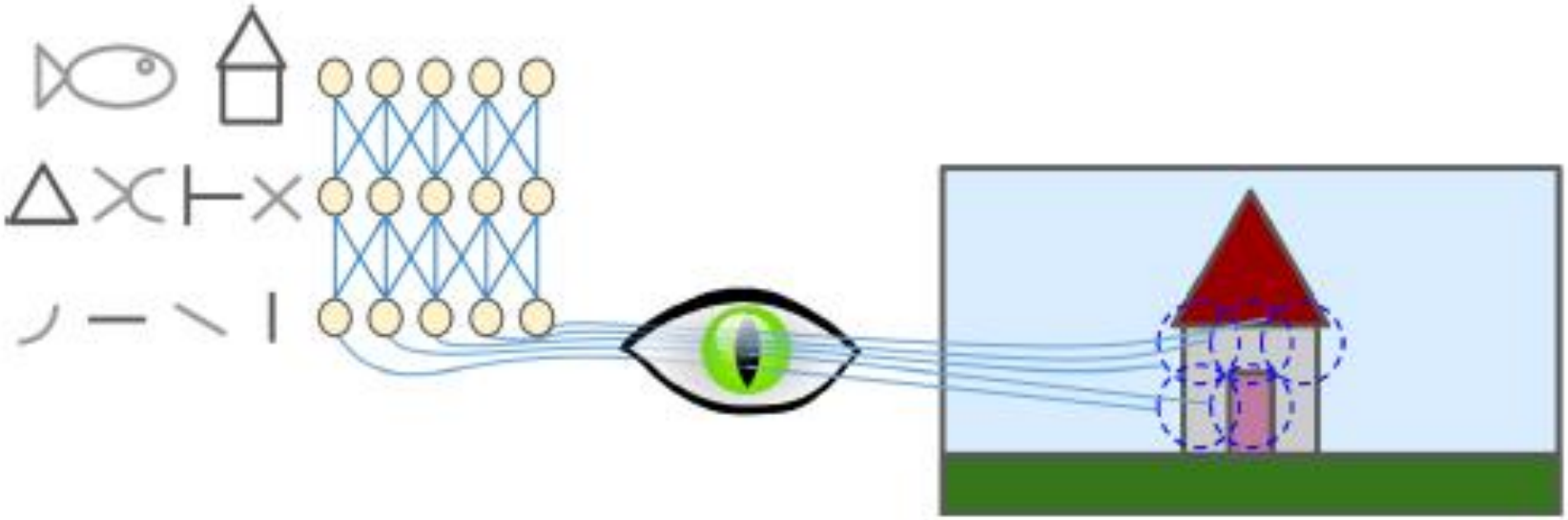
ConvNets

ConvNets



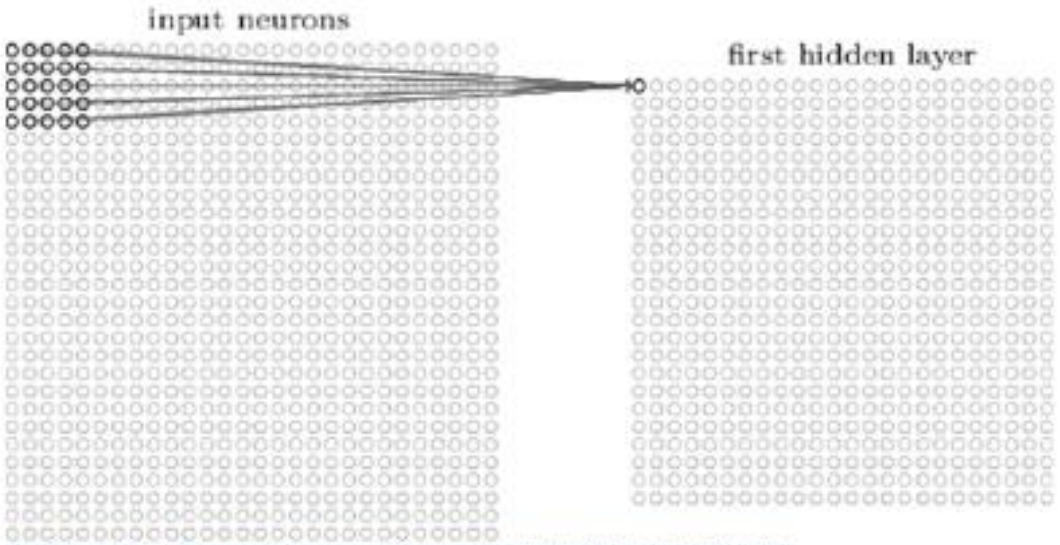
Brain Visual Cortex Inspired CNNs

- 1959, David H. Hubel and Torsten Wiesel.
- Many neurons in the visual cortex have a small local receptive field.
- They react only to visual stimuli located in a limited region of the visual field.



Receptive Fields and Filters

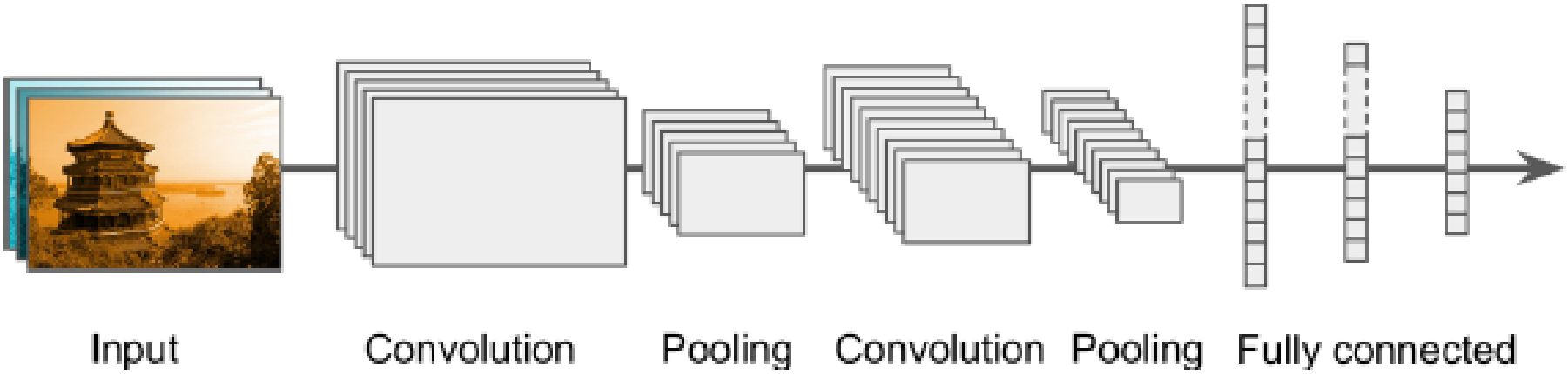
- Imagine a flashlight that is shining over the top left of the image.
- The region that it is shining over is called the receptive field.
- This flashlight is called a filter.
- A filter is a set of weights.
- A filter is a feature detector, e.g., straight edges, simple colors, and curves.



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

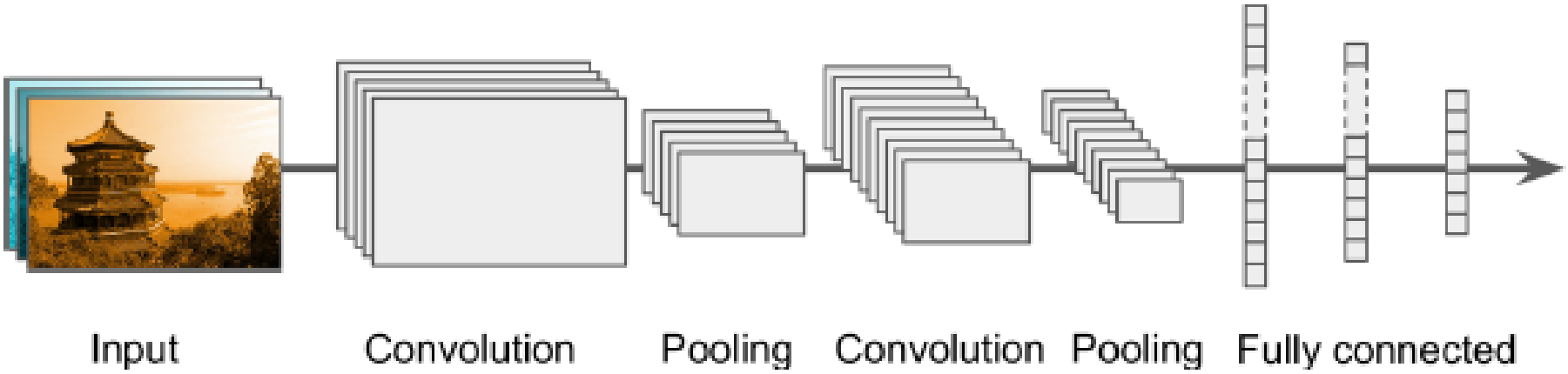
CNN Components

- Convolutional layers: apply a specified number of convolution filters to the image.
- Pooling layers: downsample the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time.
- Dense layers: a fully connected layer that performs classification on the features extracted by the convolutional layers and downsampled by the pooling layers.



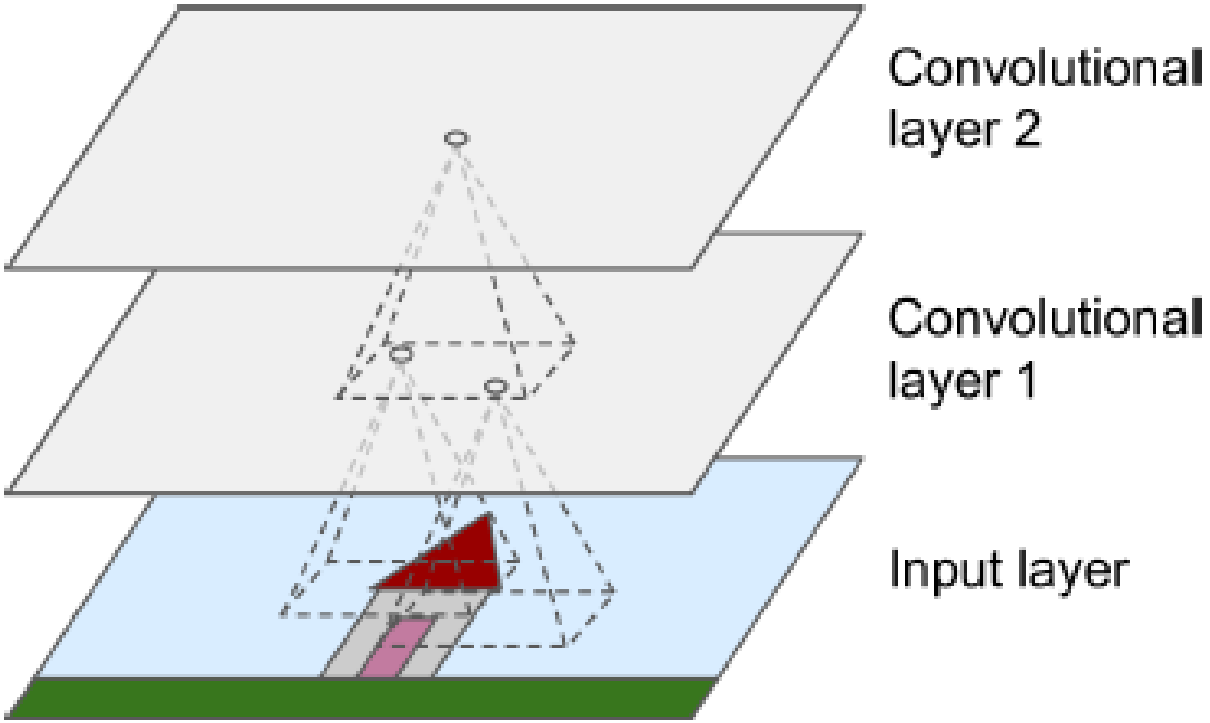
CNN Components

- A CNN is composed of a stack of convolutional modules.
- Each module consists of a convolutional layer followed by a pooling layer.
- The last module is followed by one or more dense layers that perform classification.
- The final dense layer contains a single node for each target class in the model, with a softmax activation function.



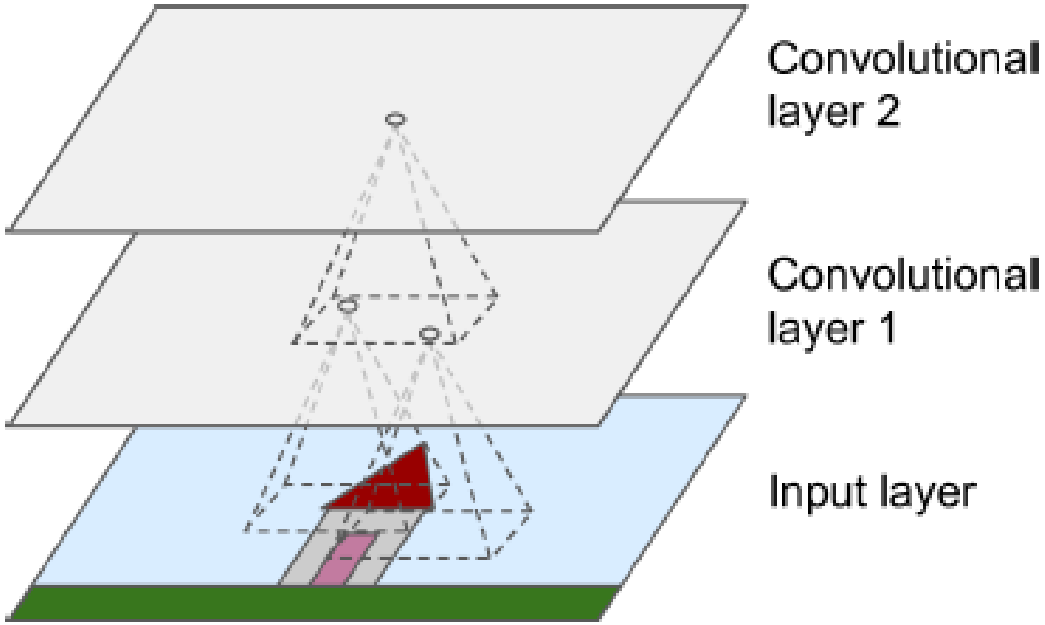
Convolutional Layer

- Sparse interactions
- Each neuron in the convolutional layers is only connected to pixels in its receptive field (not every single pixel).



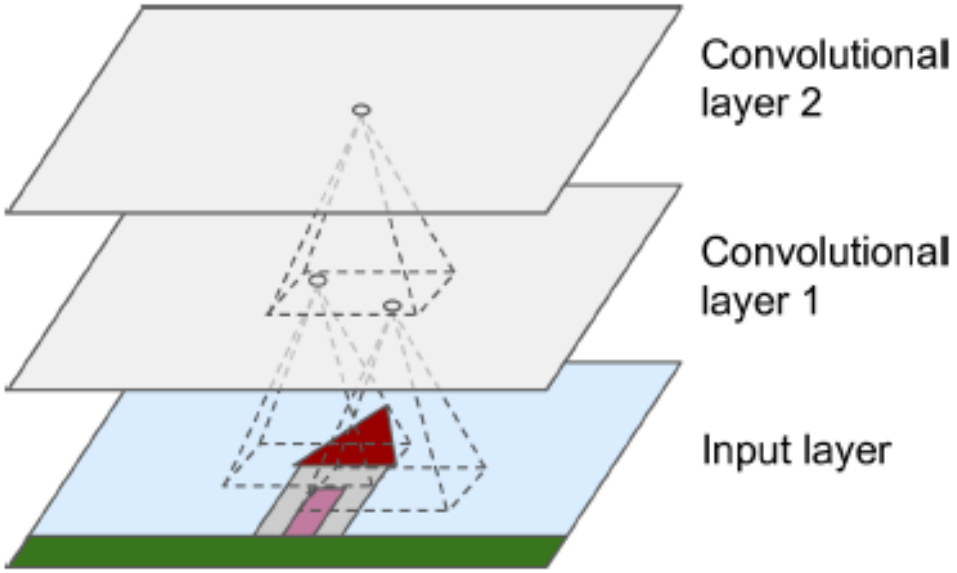
Convolutional Layer

- Each neuron applies filters on its receptive field.
  - Calculates a weighted sum of the input pixels in the receptive field.
- Adds a bias, and feeds the result through its activation function to the next layer.
- The output of this layer is a feature map (activation map)



Convolutional Layer

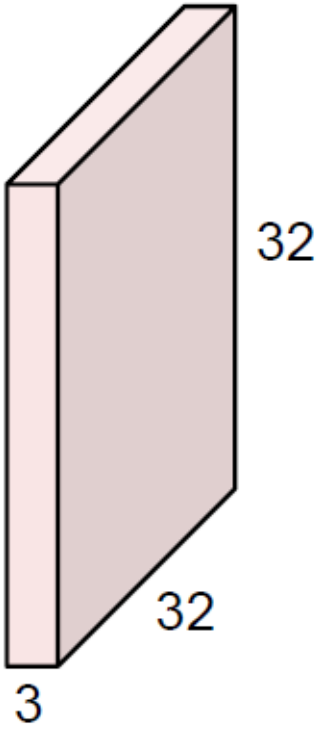
- Parameter sharing
- All neurons of a convolutional layer reuse the same weights.
- They apply the same filter in different positions.
- Whereas in a fully-connected network, each neuron had its own set of weights.



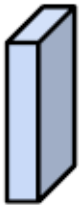
CNN Components

Filters always extend the full depth of the input volume

32x32x3 image

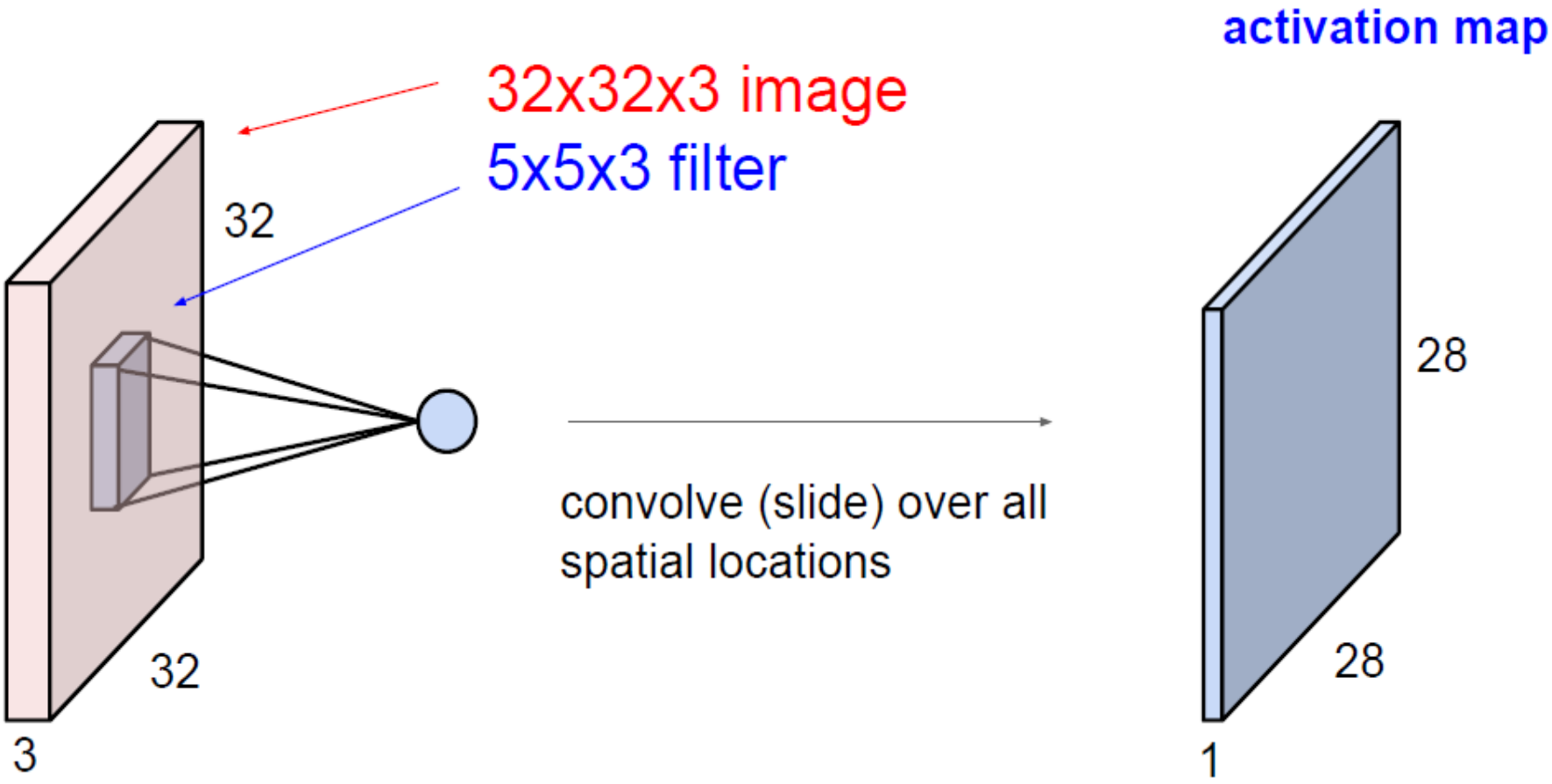


5x5x3 filter

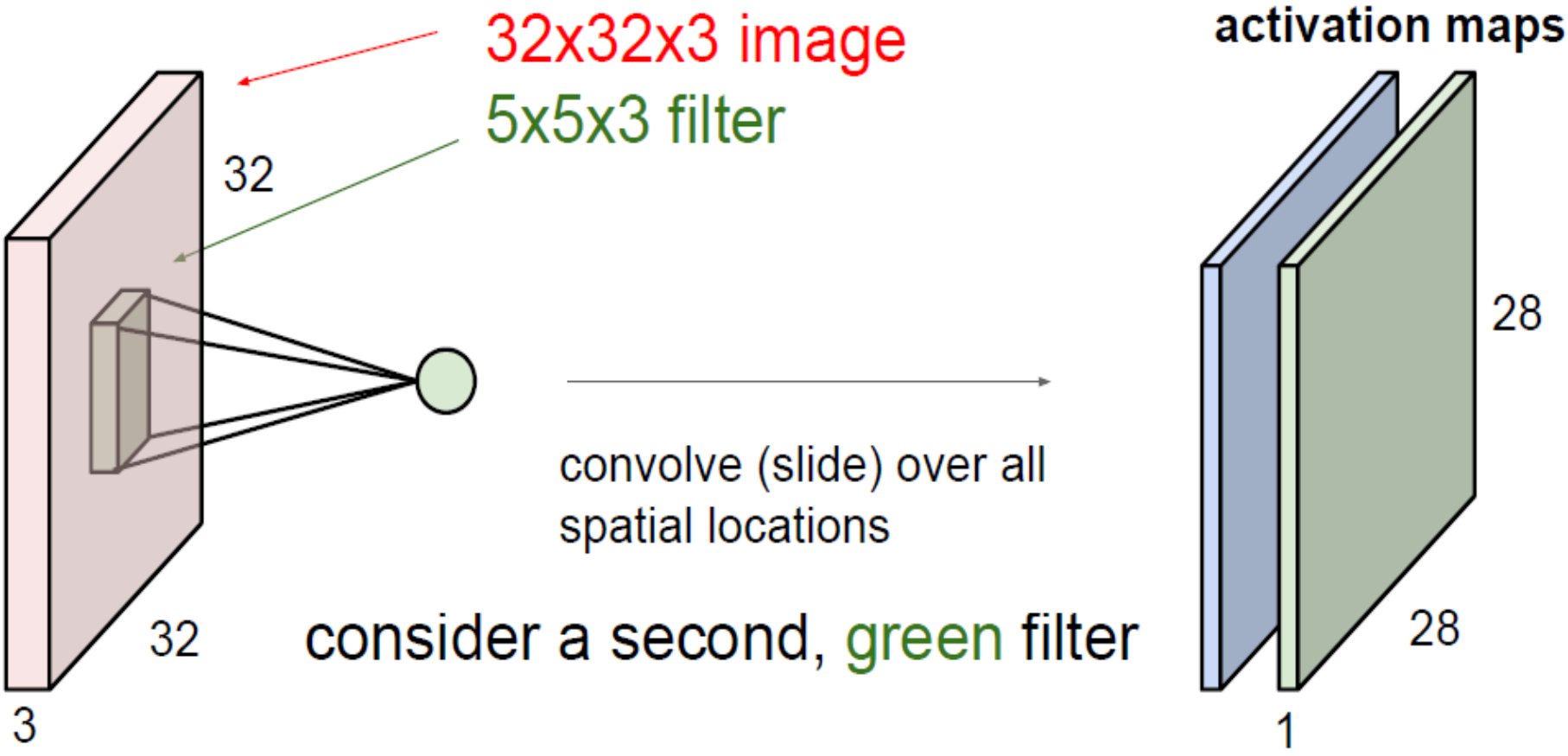


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

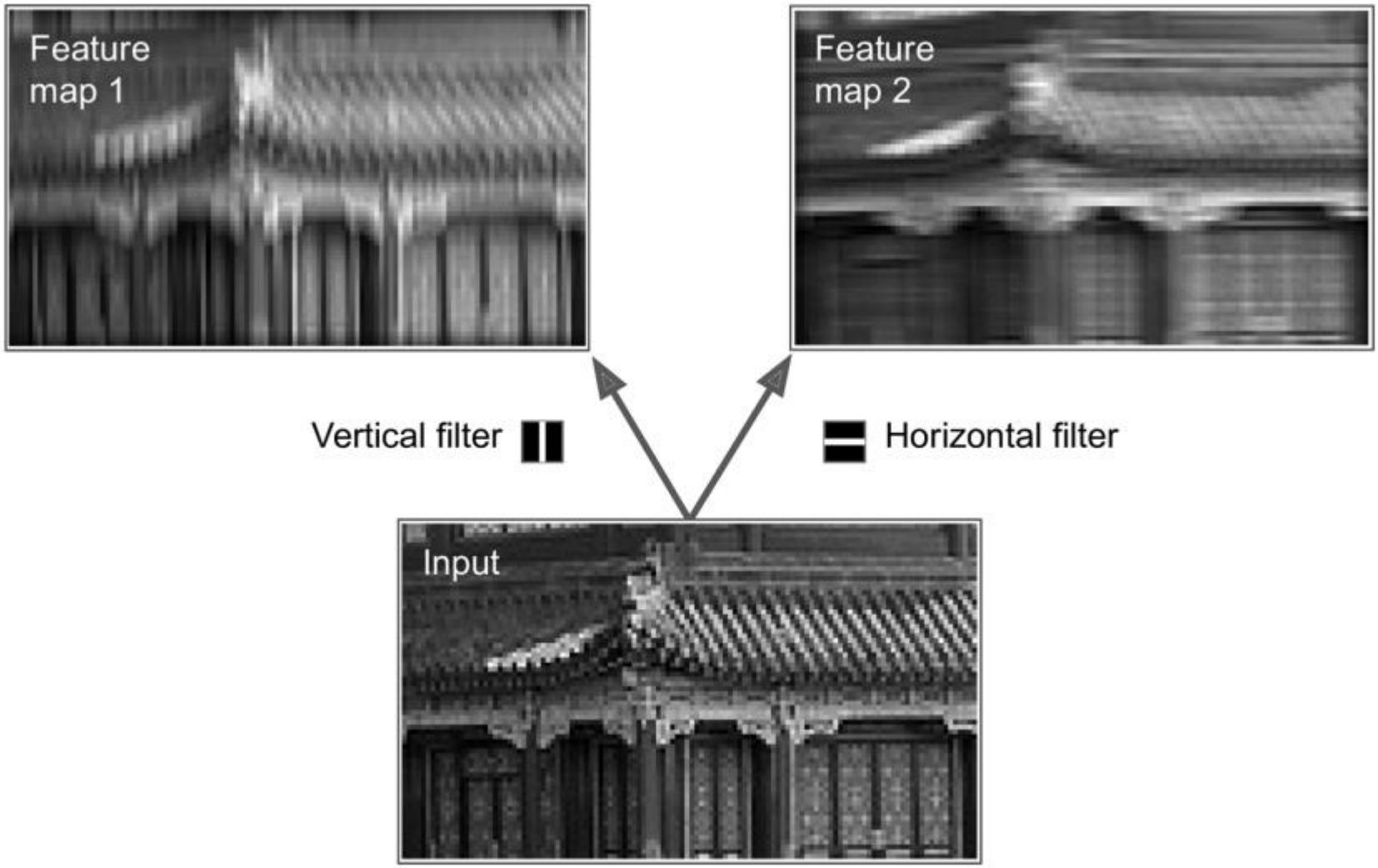
CNN Components



CNN Components

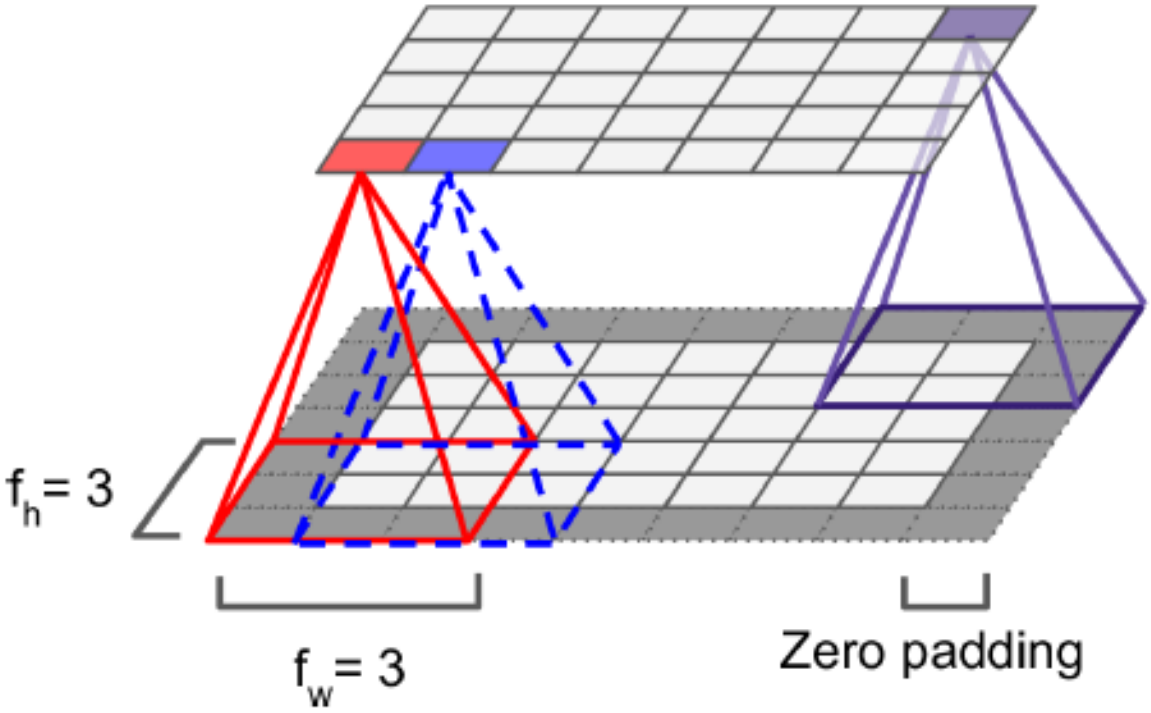


CNN Components



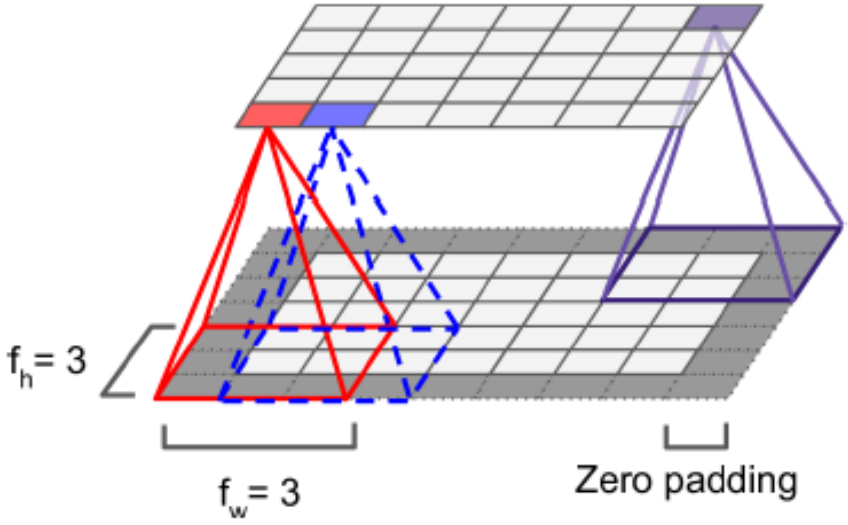
Convolutional Layer

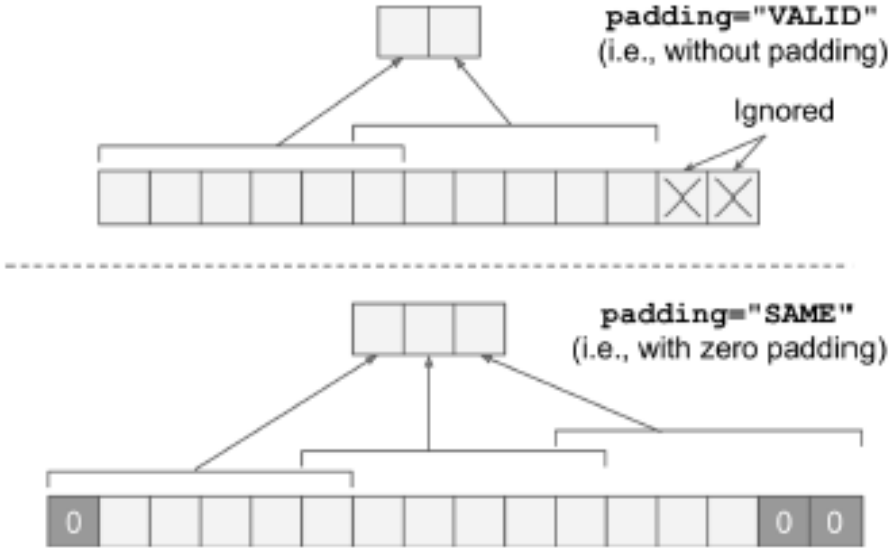
- Assume the filter size (kernel size) is $f_w \times f_h$.
  - f_h and f_w are the height and width of the receptive field, respectively.
- A neuron in row i and column j of a given layer is connected to the outputs of the neurons in the previous layer in rows i to $i + f_h - 1$, and columns j to $j + f_w - 1$.



Padding

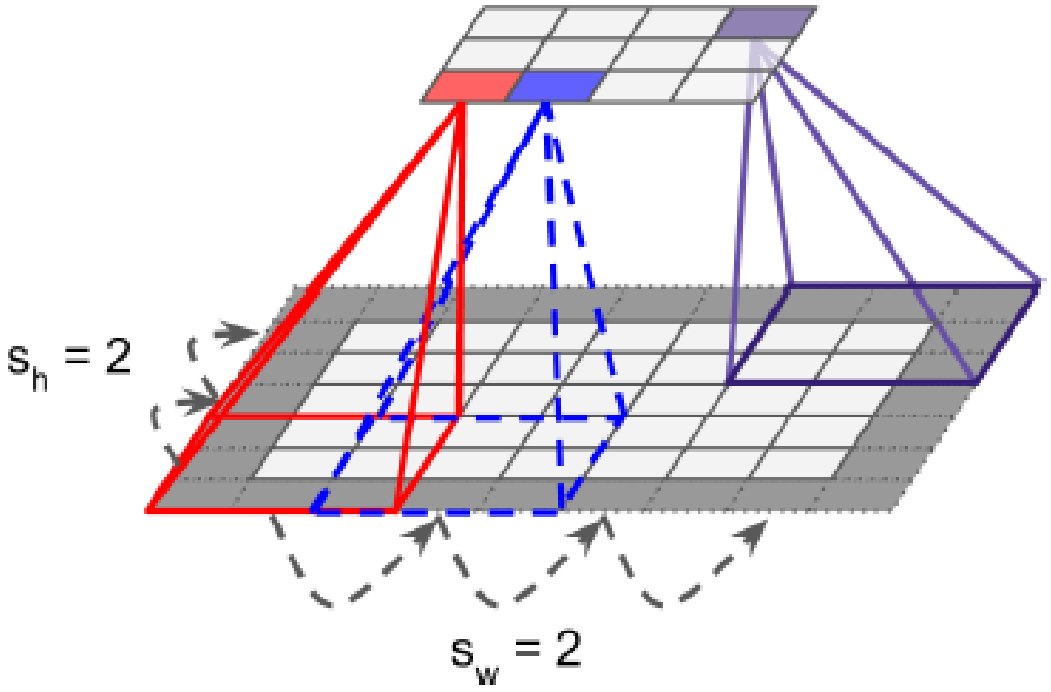
- What will happen if you apply a 5x5 filter to a 32x32 input volume?
  - The output volume would be 28x28.
  - The spatial dimensions decrease.
- Zero padding: in order for a layer to have the same height and width as the previous layer, it is common to add zeros around the inputs.
- In TensorFlow, padding can be either `SAME` or `VALID` to have zero padding or not.



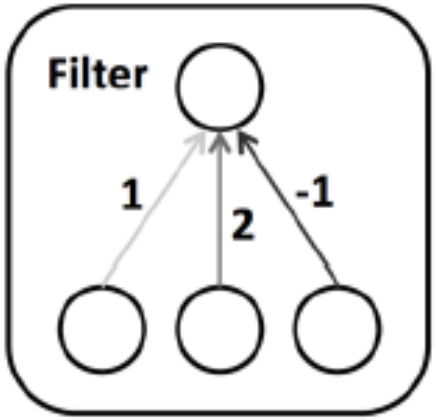


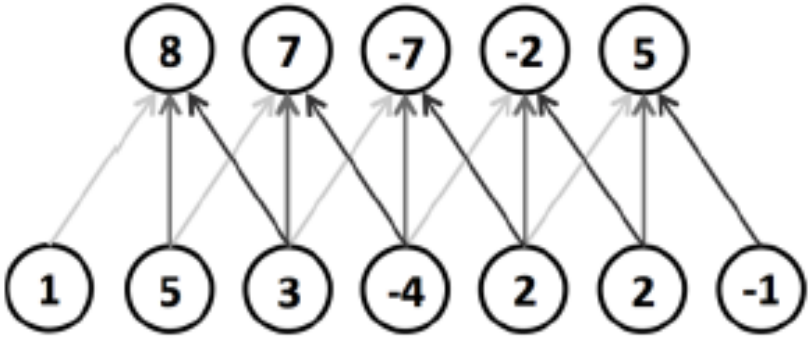
Stride

- The distance between two consecutive receptive fields is called the stride.
- The stride controls how the filter convolves around the input volume.
- Assume s_h and s_w are the vertical and horizontal strides, then, a neuron located in row i and column j in a layer is connected to the outputs of the neurons in the previous layer located in rows $i \times s_h$ to $i \times s_h + f_h - 1$, and columns $j \times s_w$ to $j \times s_w + f_w - 1$.

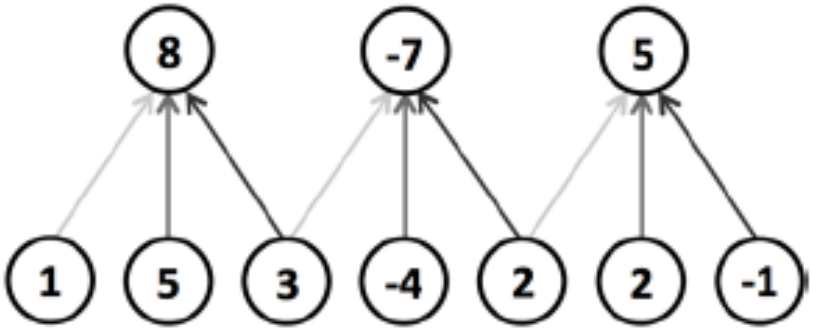


Stride





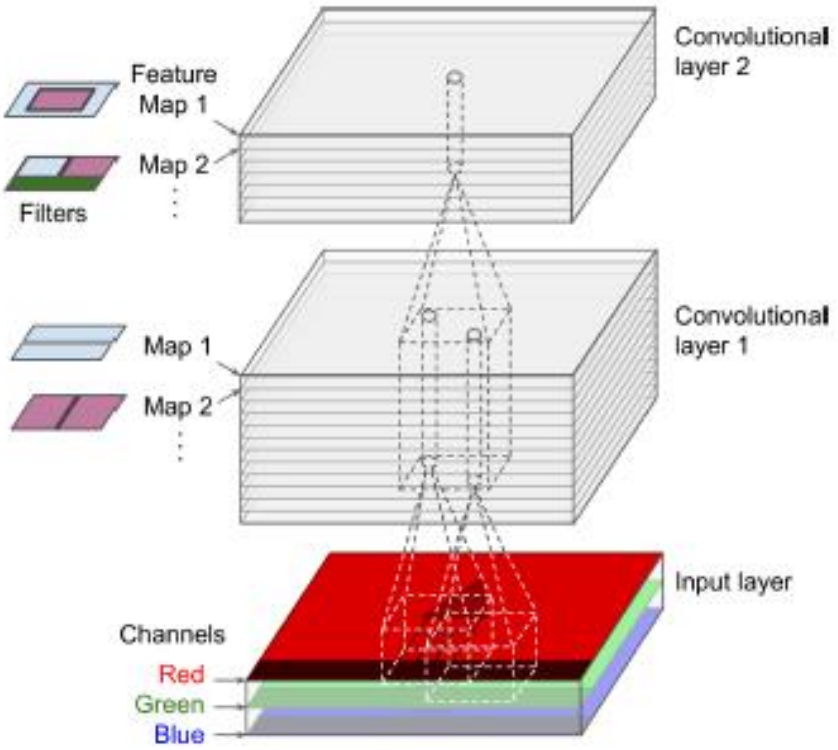
S = 1



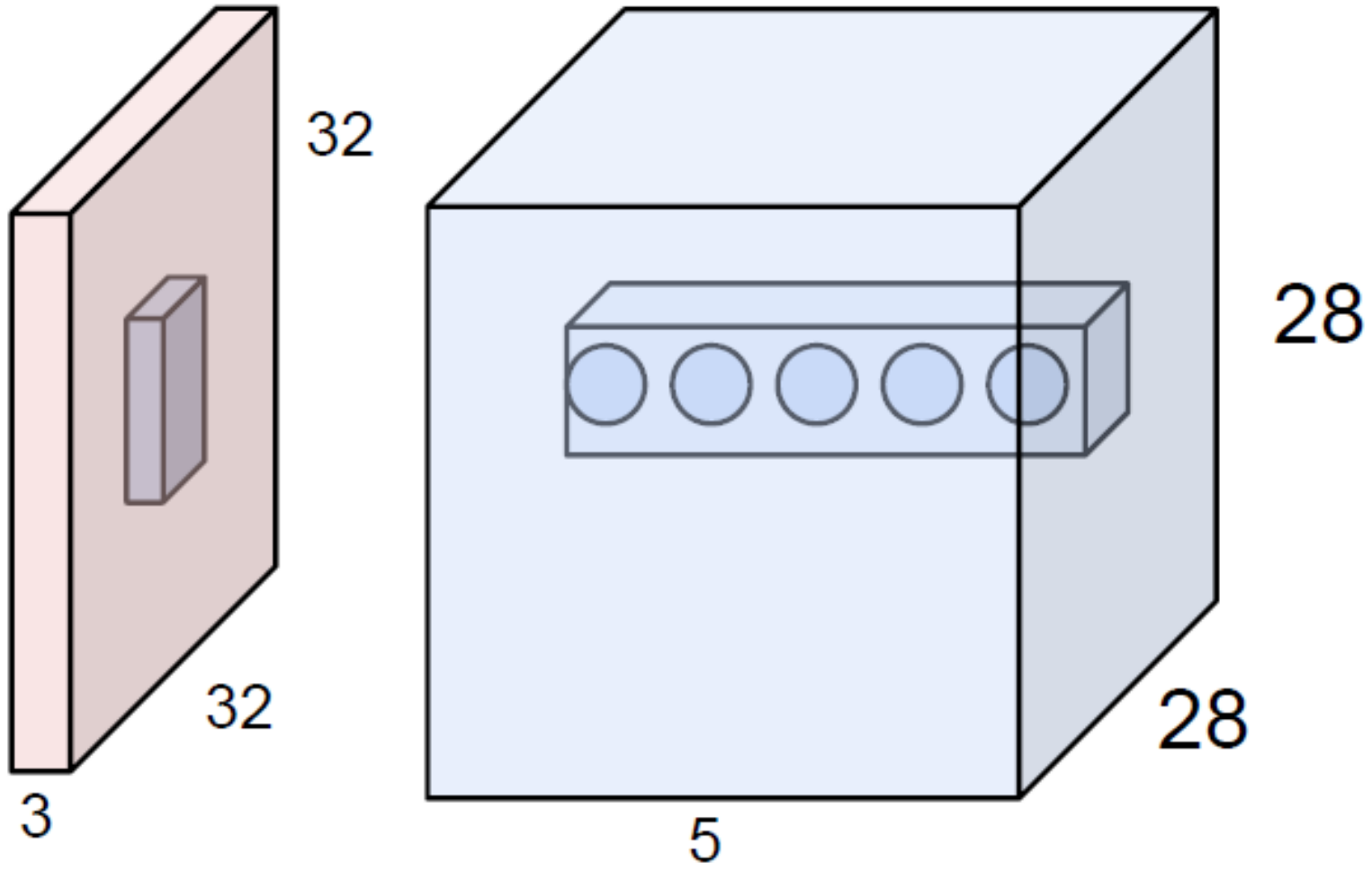
S = 2

Stacking Multiple Feature Maps

- Up to now, we represented each convolutional layer with a single feature map.
- Each convolutional layer can be composed of several feature maps of equal sizes.
- Input images are also composed of multiple sublayers: one per color channel.
- A convolutional layer simultaneously applies multiple filters to its inputs.

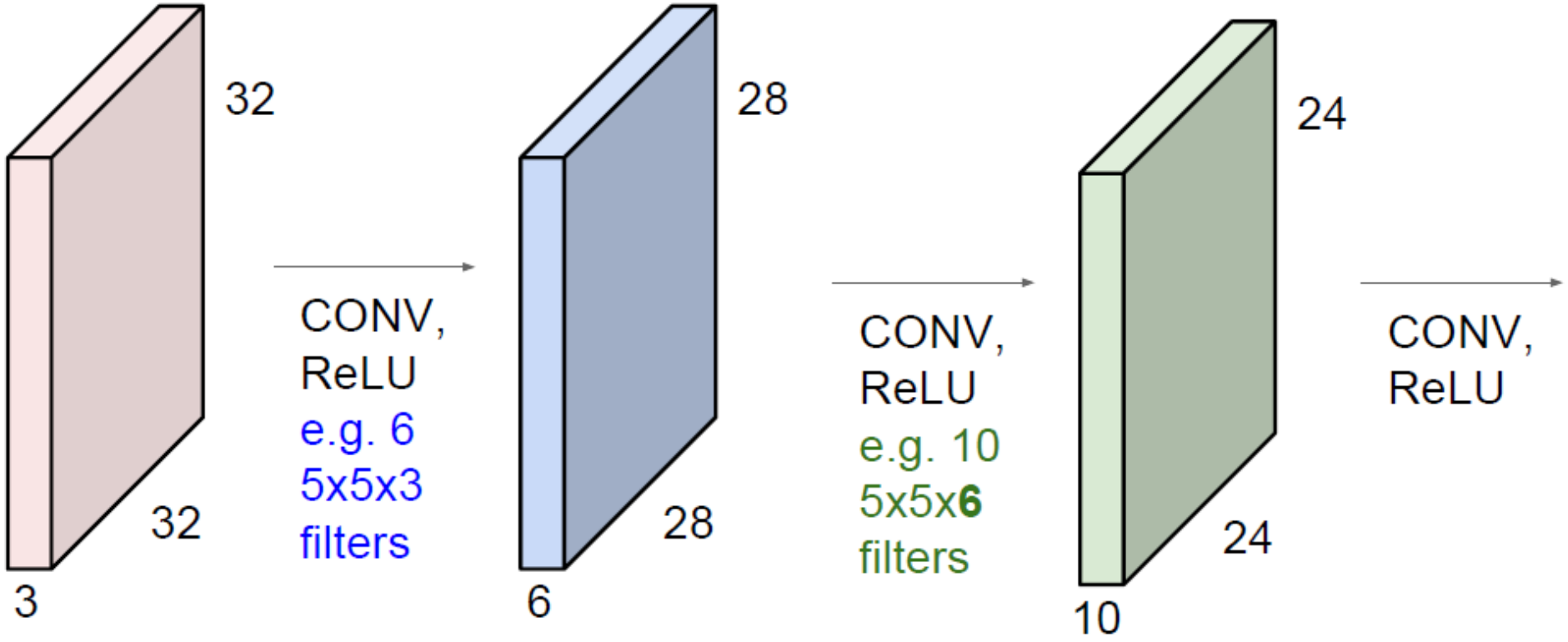


CNN Components



Stacking Multiple Feature Maps

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



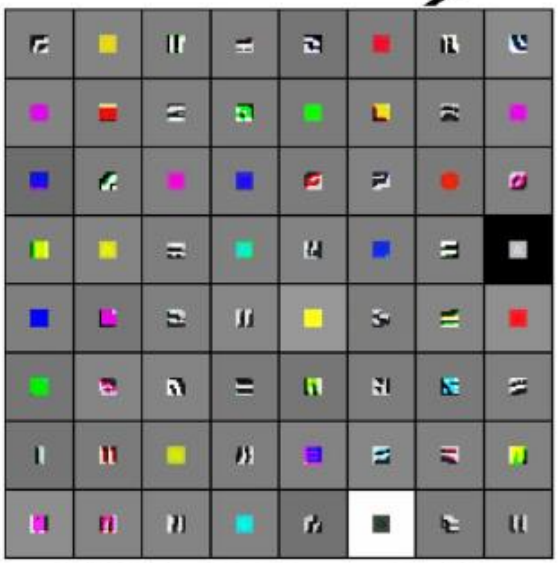
Stacking Multiple Feature Maps

Low-level features

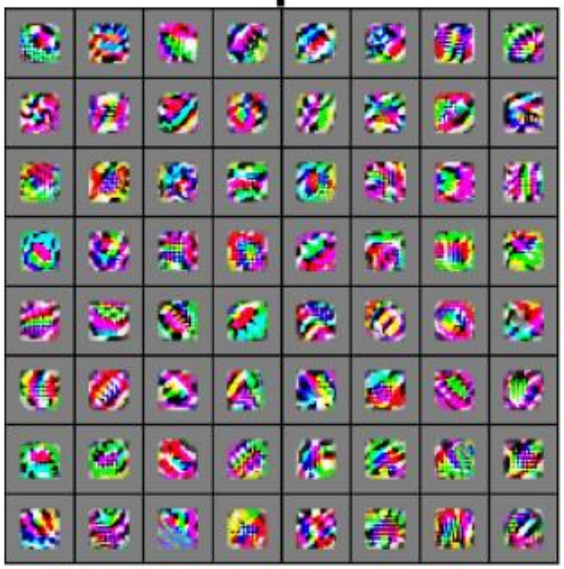
Mid-level features

High-level features

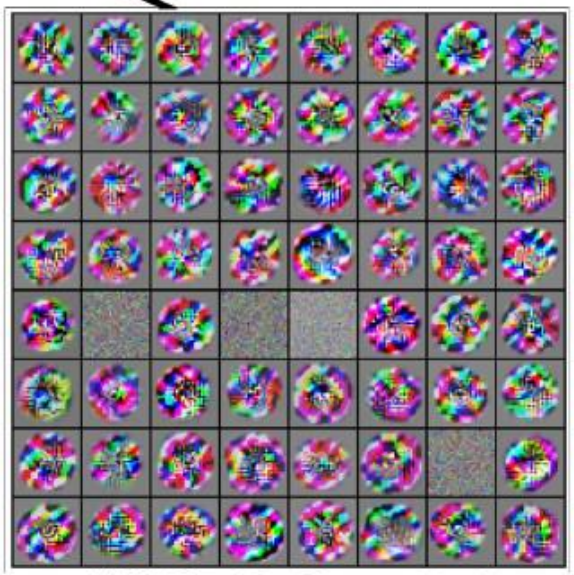
Linearly separable classifier



VGG-16 Conv1_1



VGG-16 Conv3_2



VGG-16 Conv5_3

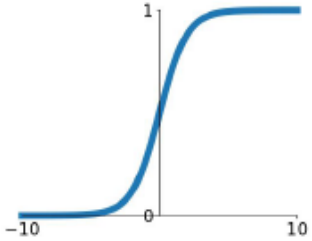
Activation Function

- After calculating a weighted sum of the input pixels in the receptive fields, and adding biases, each neuron feeds the result through its ReLU activation function to the next layer.

- The purpose of this activation function is to add non linearity to the system.

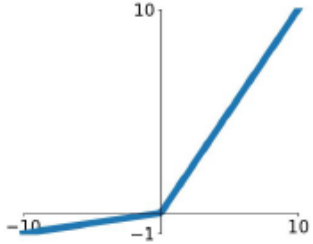
Sigmoid

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



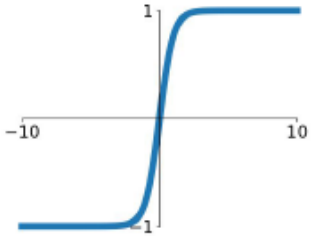
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

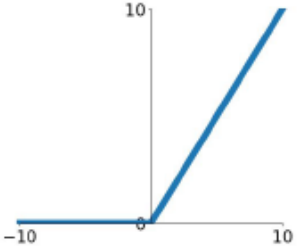


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

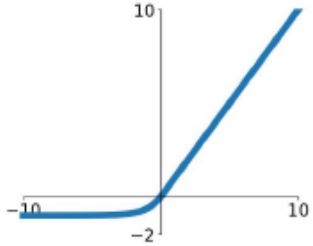
ReLU

$$\max(0, x)$$



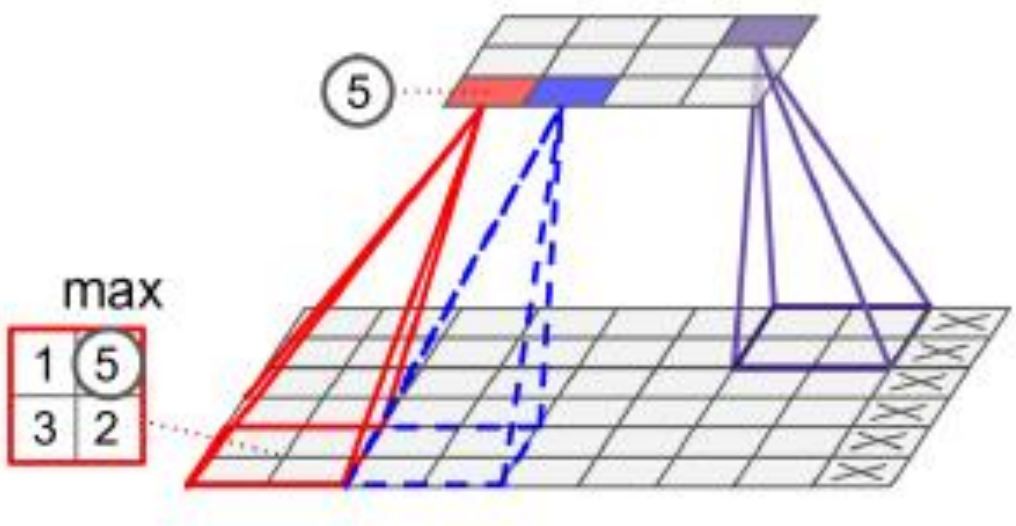
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



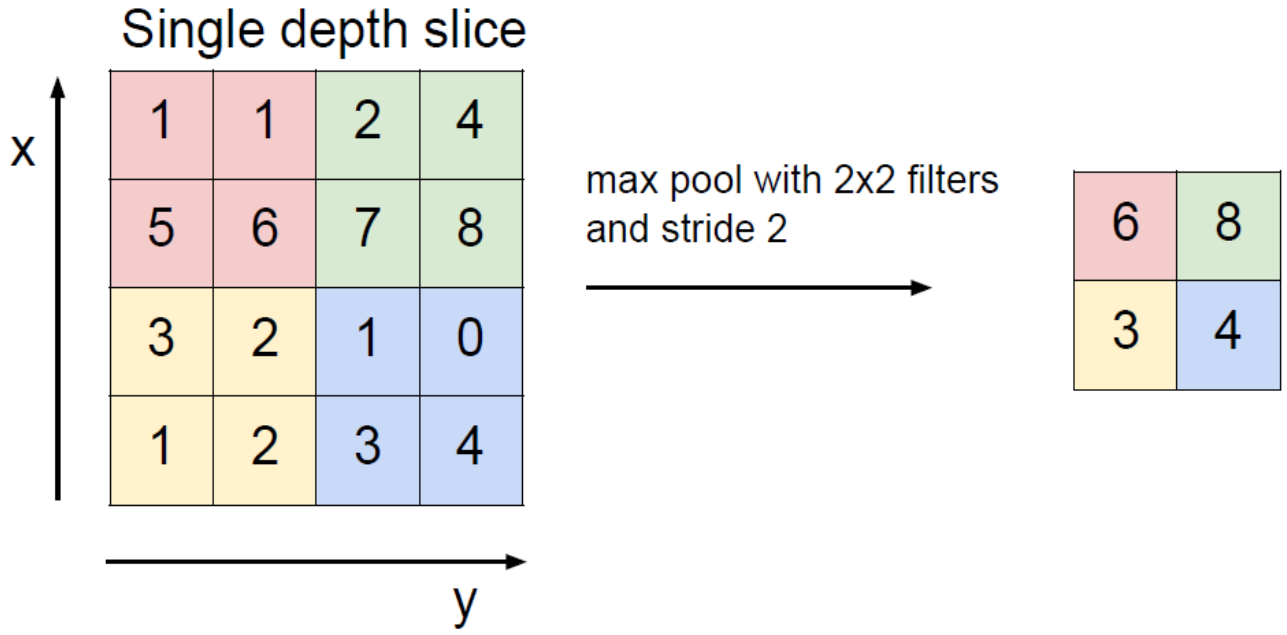
Pooling Layer

- After the activation functions, we can apply a pooling layer.
- Its goal is to subsample (shrink) the input image.
  - To reduce the computational load, the memory usage, and the number of parameters.



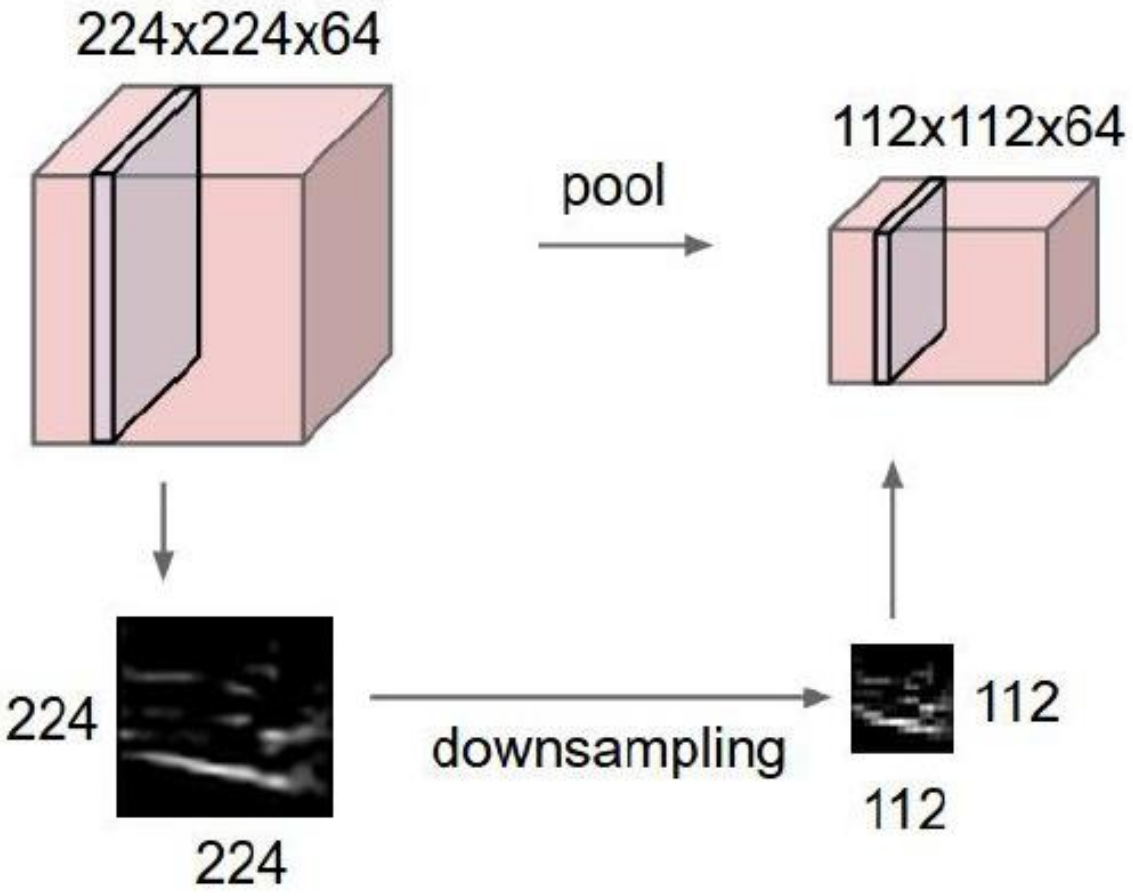
Pooling Layer

- Each neuron in a pooling layer is connected to the outputs of a receptive field in the previous layer.
- A pooling neuron has no weights.
- It aggregates the inputs using an aggregation function such as the max or mean.



Pooling Layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

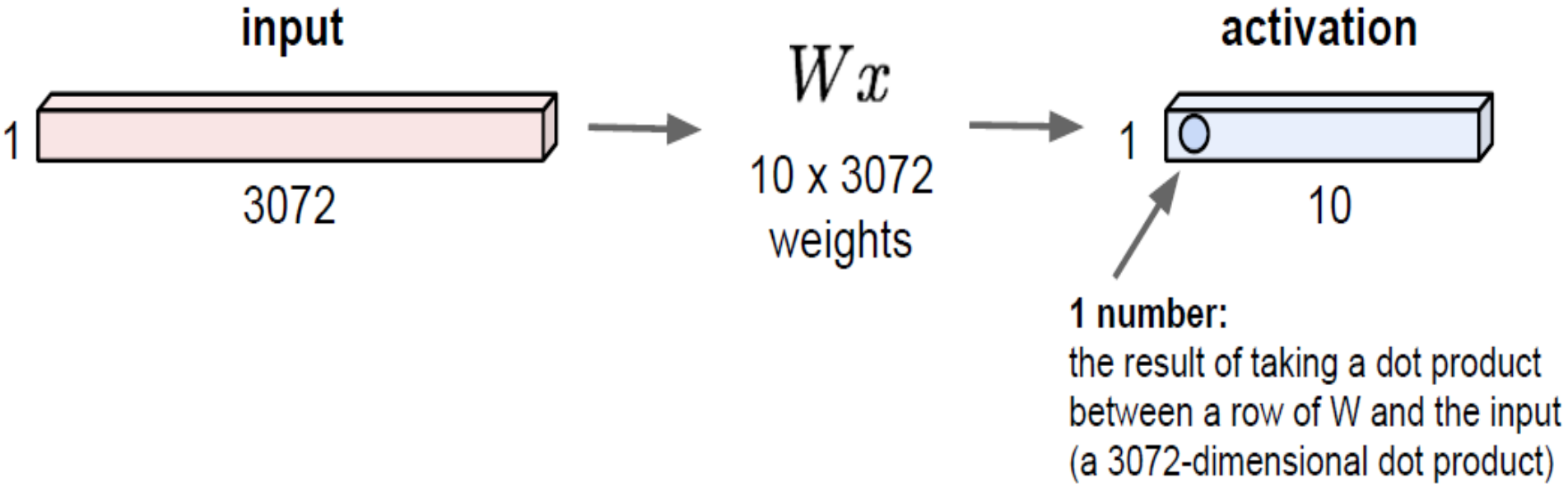


Fully Connected Layer & Flattening

- This layer takes an input from the last convolution module, and outputs an N dimensional vector.
  - N is the number of classes that the model has to choose from.
- For example, if you wanted a digit classification model, N would be 10.
- Each number in this N dimensional vector represents the probability of a certain class.
- We need to convert the output of the convolutional part of the CNN into a 1D feature vector.
- This operation is called flattening.
- It gets the output of the convolutional layers, flattens all its structure to create a single long feature vector to be used by the dense layer for the final classification.

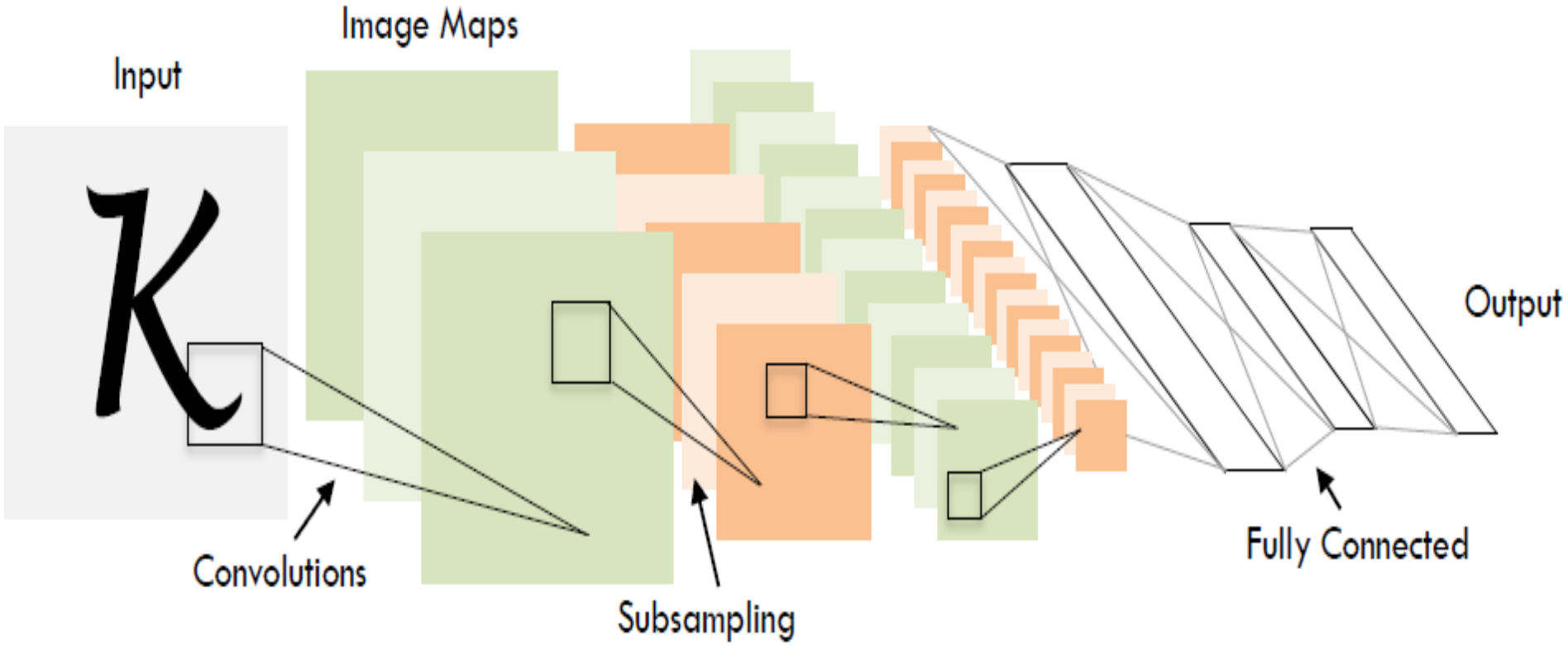
Fully Connected Layer & Flattening

32x32x3 image -> stretch to 3072 x 1



CNN Architectures

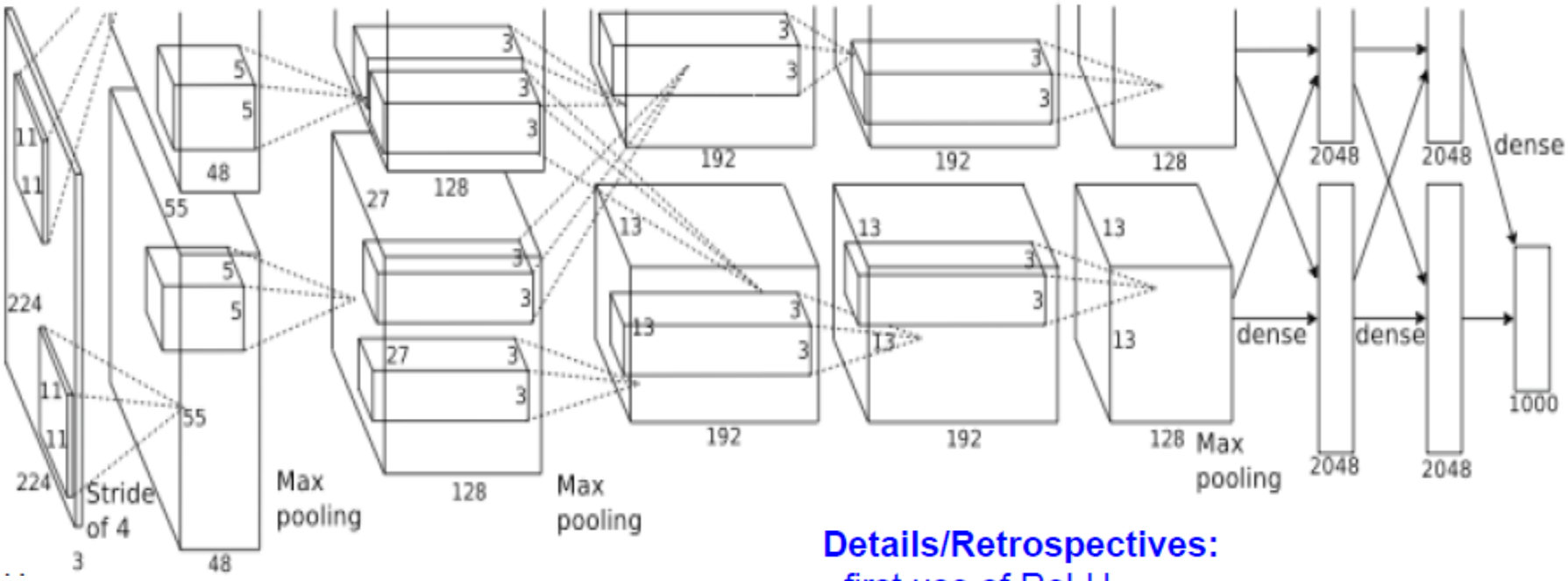
LeNet-5 (1998)


Input
Image Maps
Output
Convolutions
Subsampling
Fully Connected

LeNet-5 (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	–	–	–

AlexNet (2012)



Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

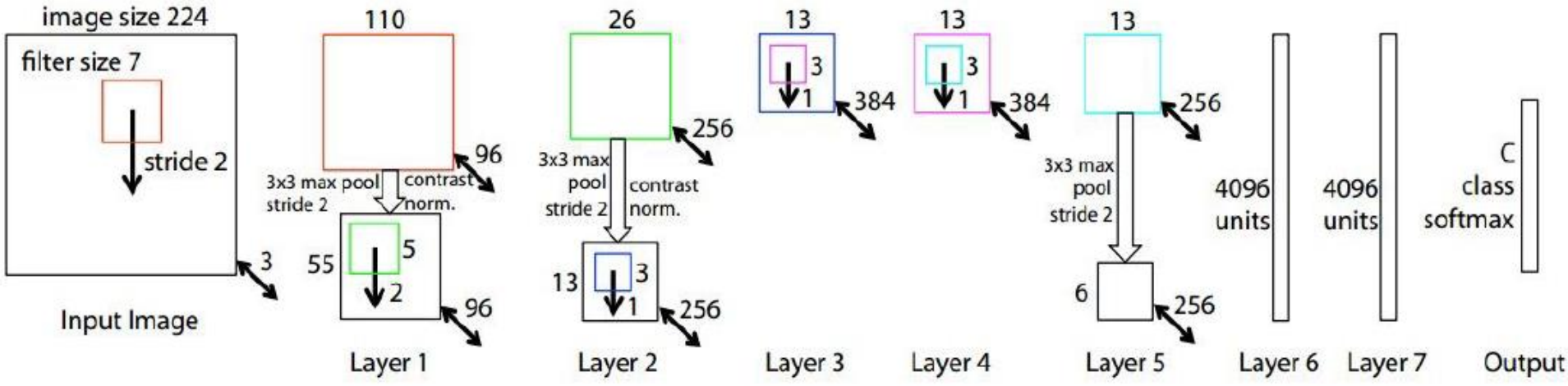
AlexNet (2012)

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

AlexNet (2012)

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	`valid`	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

ZFNet (2013)



AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

VGG Net (2014)

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

VGG Net16 (2014)

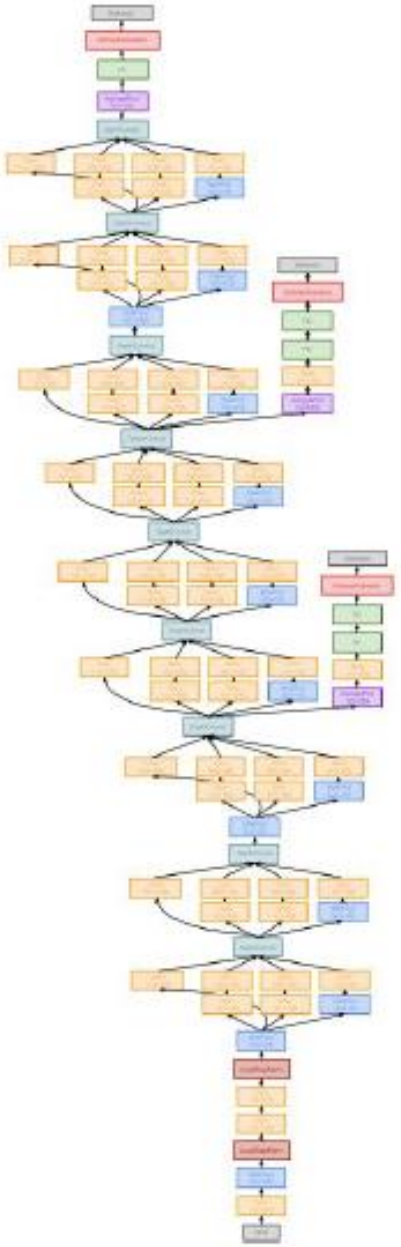
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M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

TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)
TOTAL params: 138M parameters

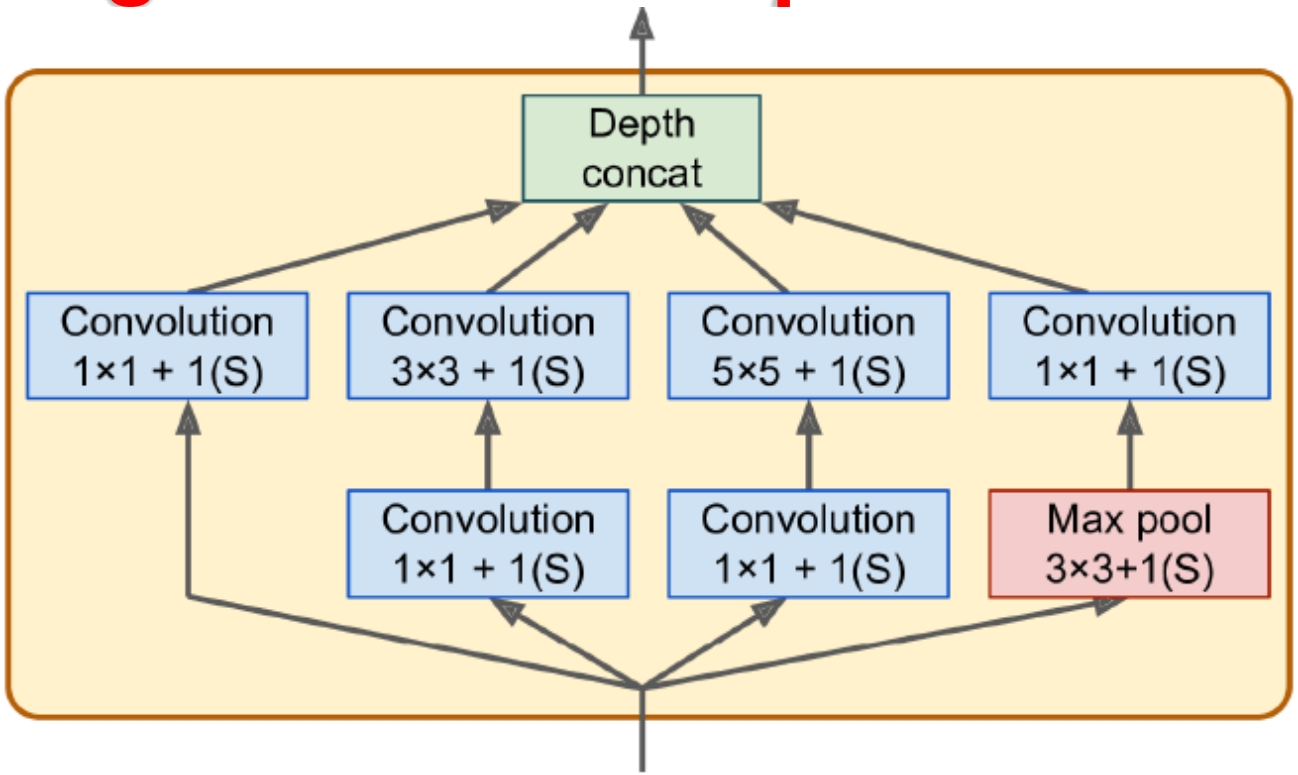
GoogLeNet - Inception module (2014)

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC’14 classification winner
  (6.7% top 5 error)



GoogLeNet - Inception module

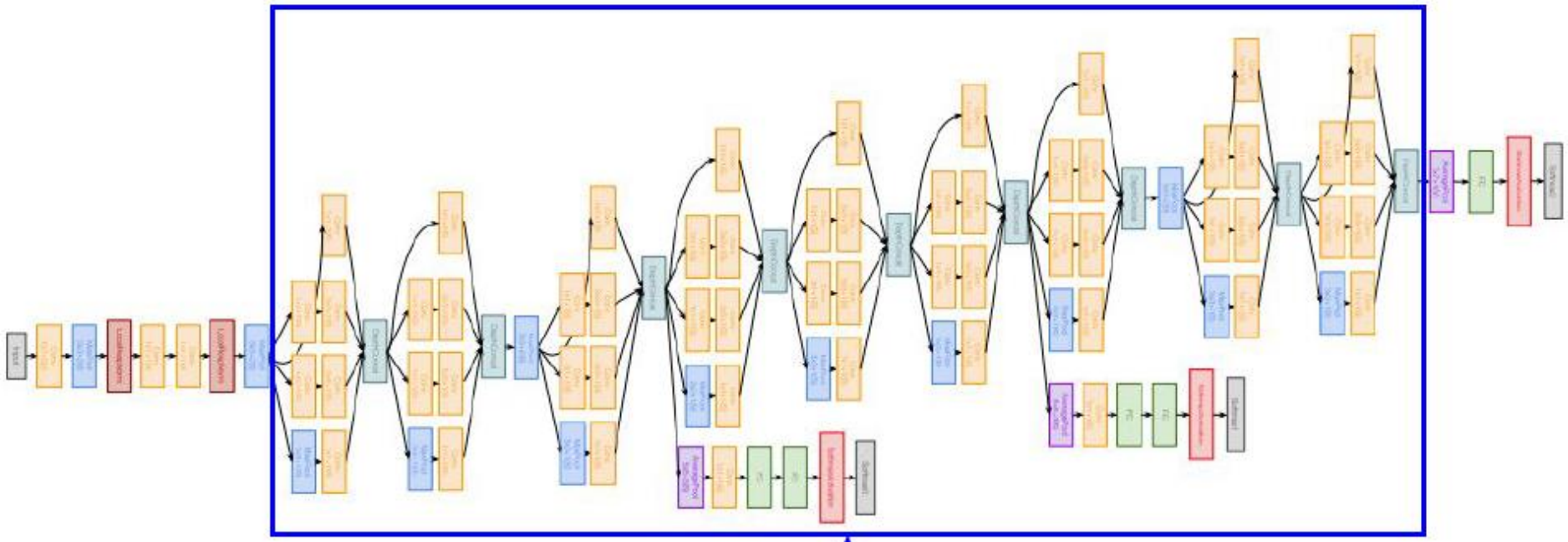


Concatenate all filter outputs together depth-wise

Apply parallel filter operations on the input from previous layer:

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

GoogLeNet (2014)



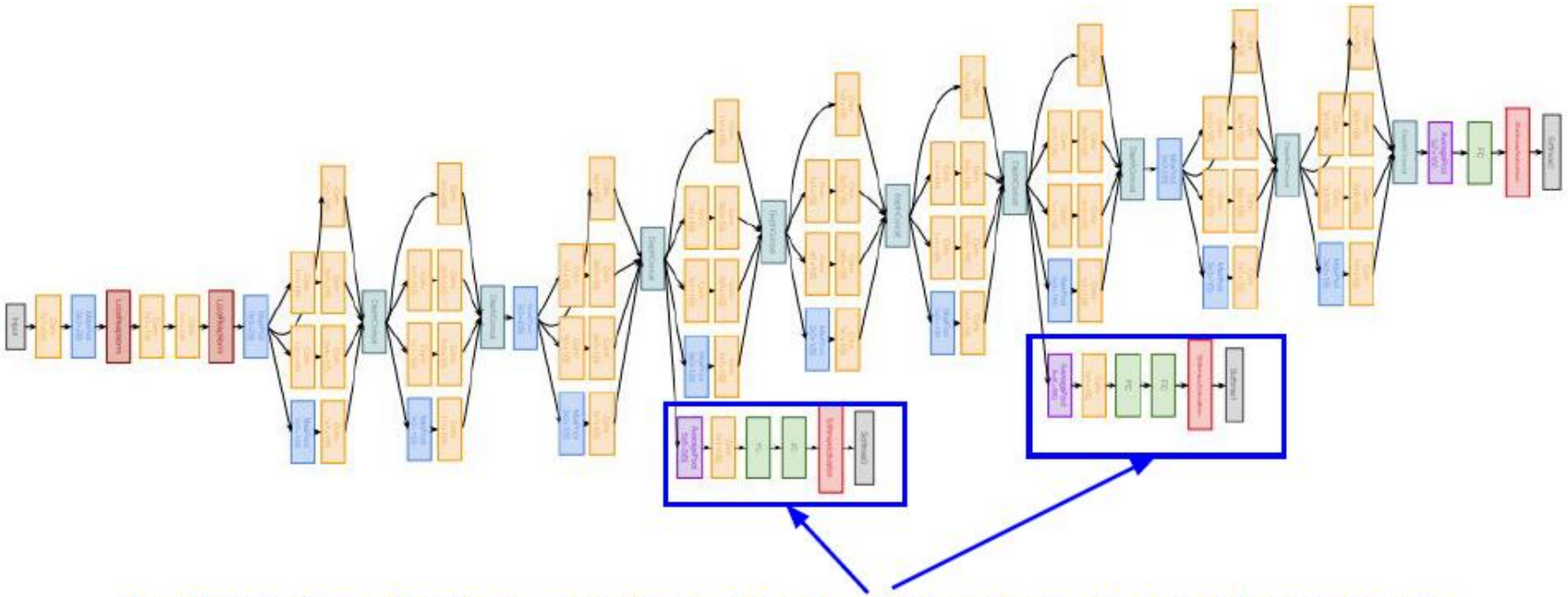
Stem Network:
Conv-Pool-
2x Conv-Pool

Stacked Inception Modules

Classifier output

after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

GoogLeNet (2014)



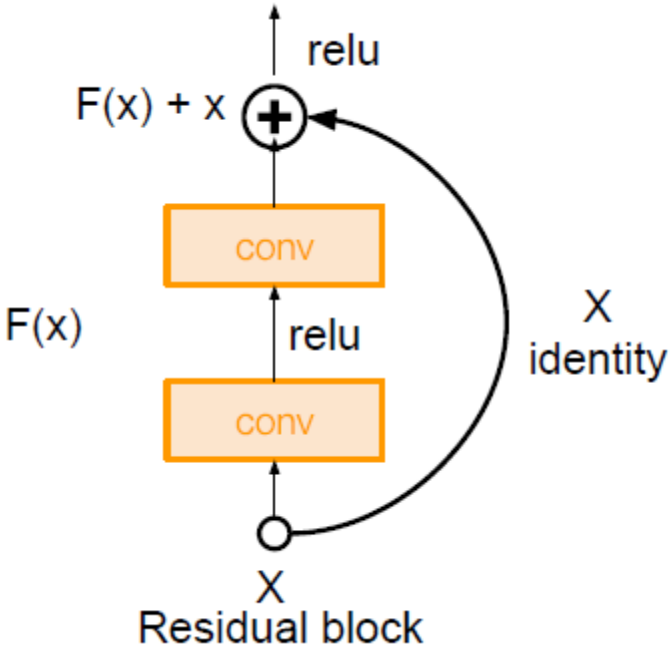
Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

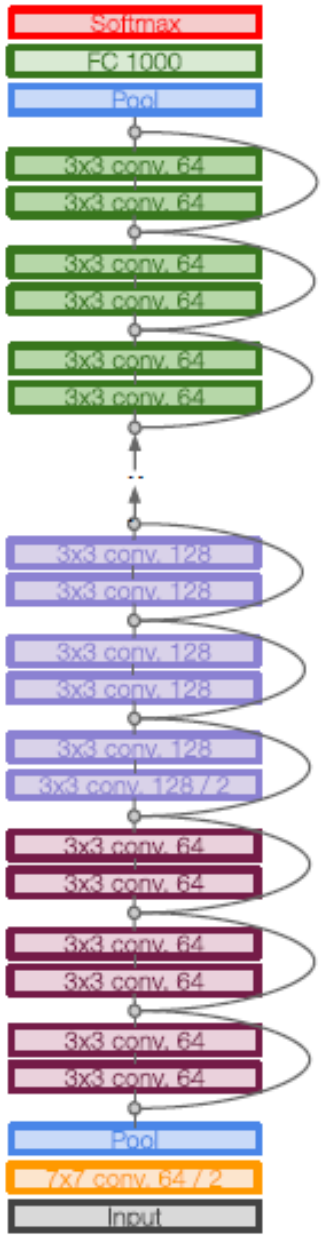
22 total layers with weights
(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

Residual Network (or ResNet) 2015

Very deep networks using residual connections

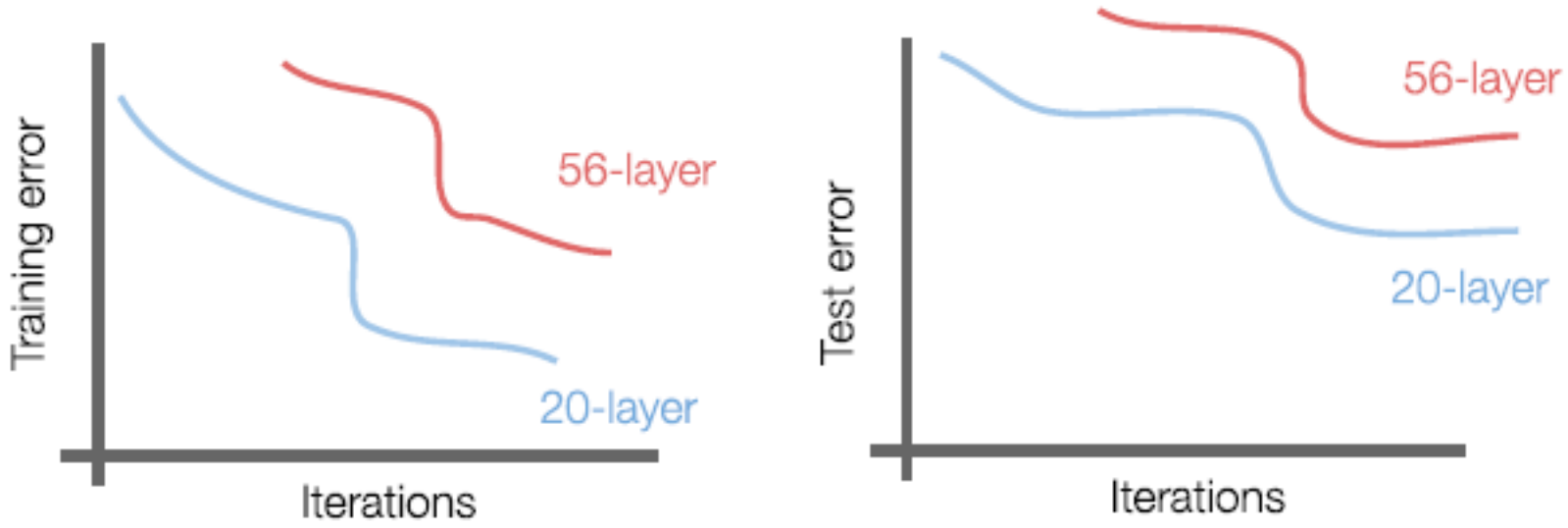
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!





Regular and deep residual network

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

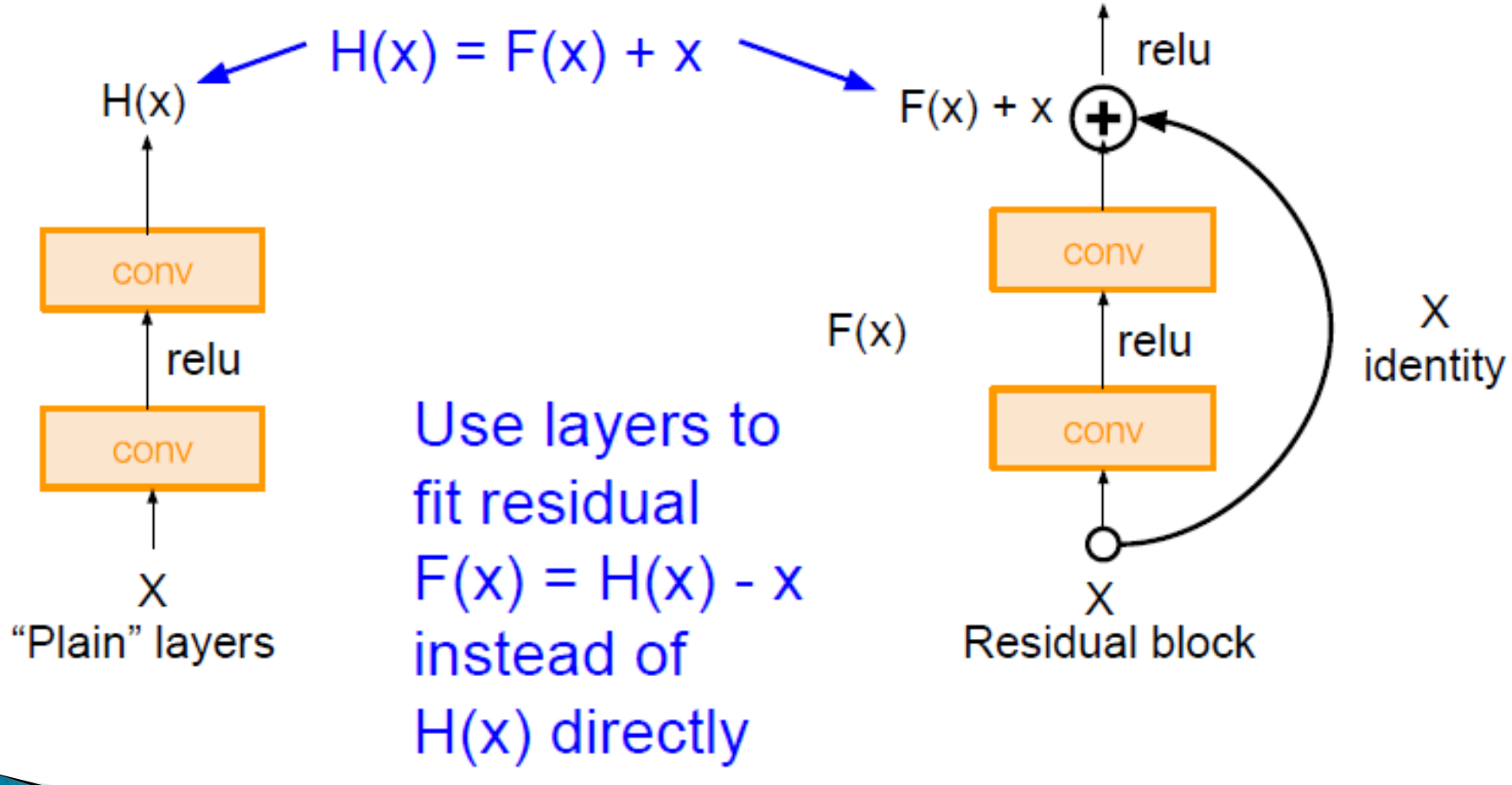


56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

Residual Network (or ResNet) 2015

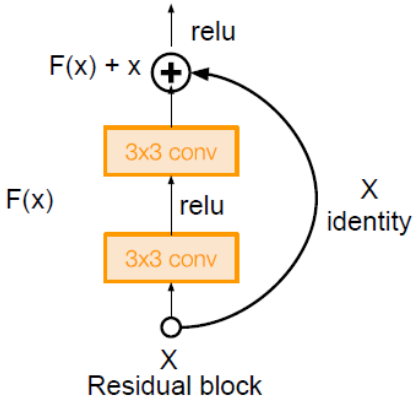
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

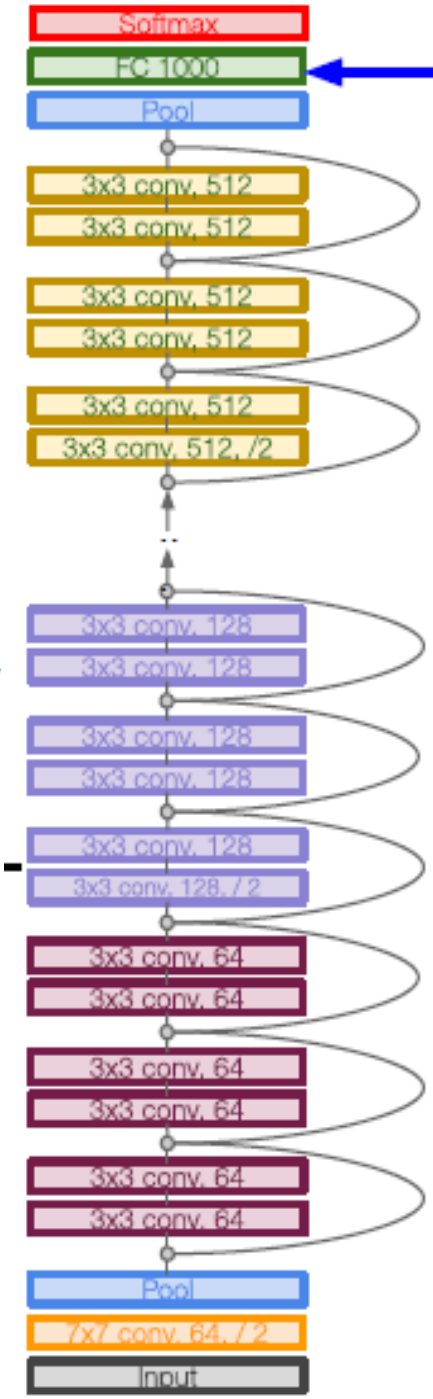


ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)





No FC layers besides FC 1000 to output classes

Global average pooling layer after last conv layer

3x3 conv, 128 filters, /2 spatially with stride 2

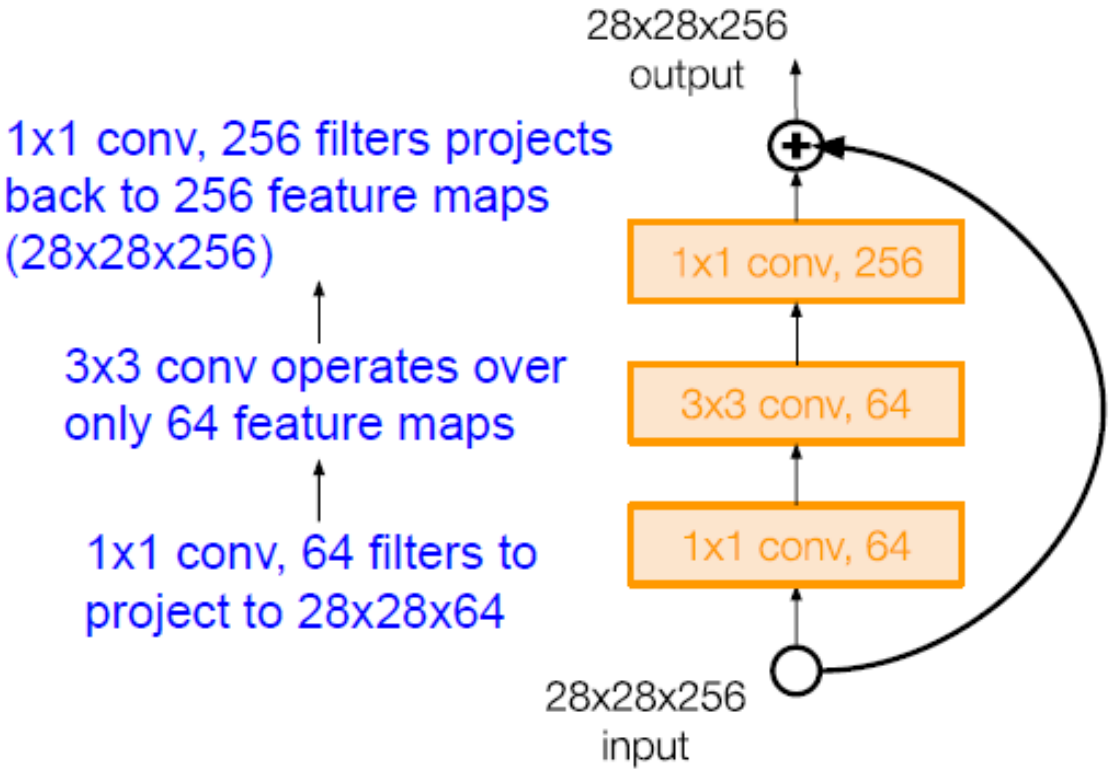
3x3 conv, 64 filters

Beginning conv layer

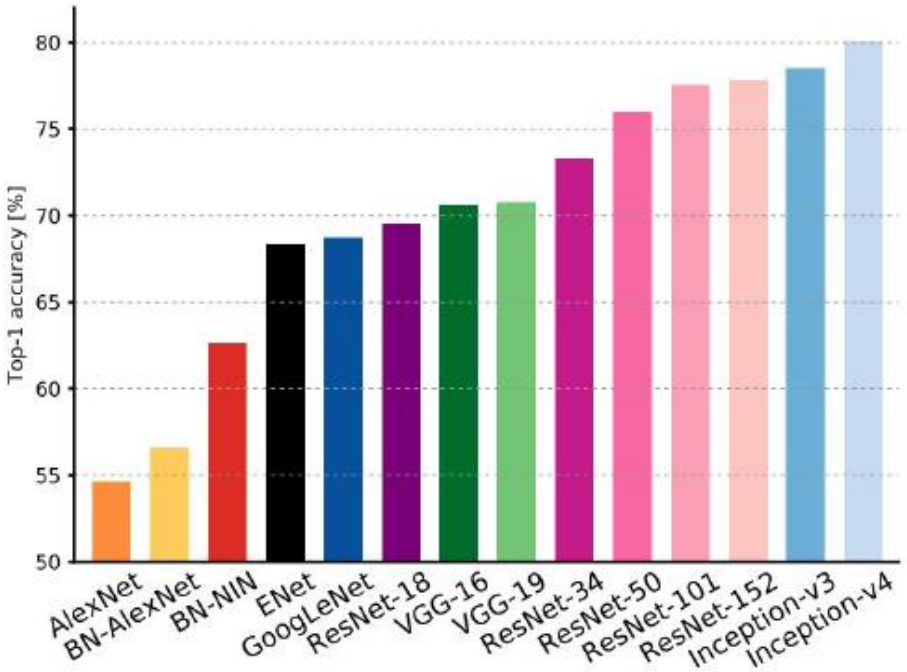
ResNet

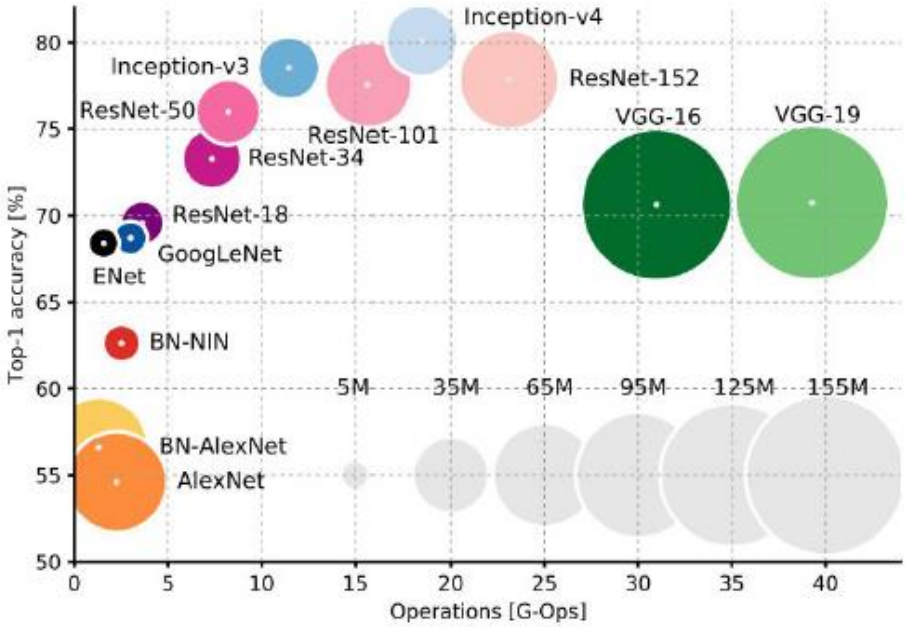
Total depths of 34, 50, 101, or 152 layers for ImageNet

For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)



Comparing complexity





Inception-v4: Resnet + Inception!

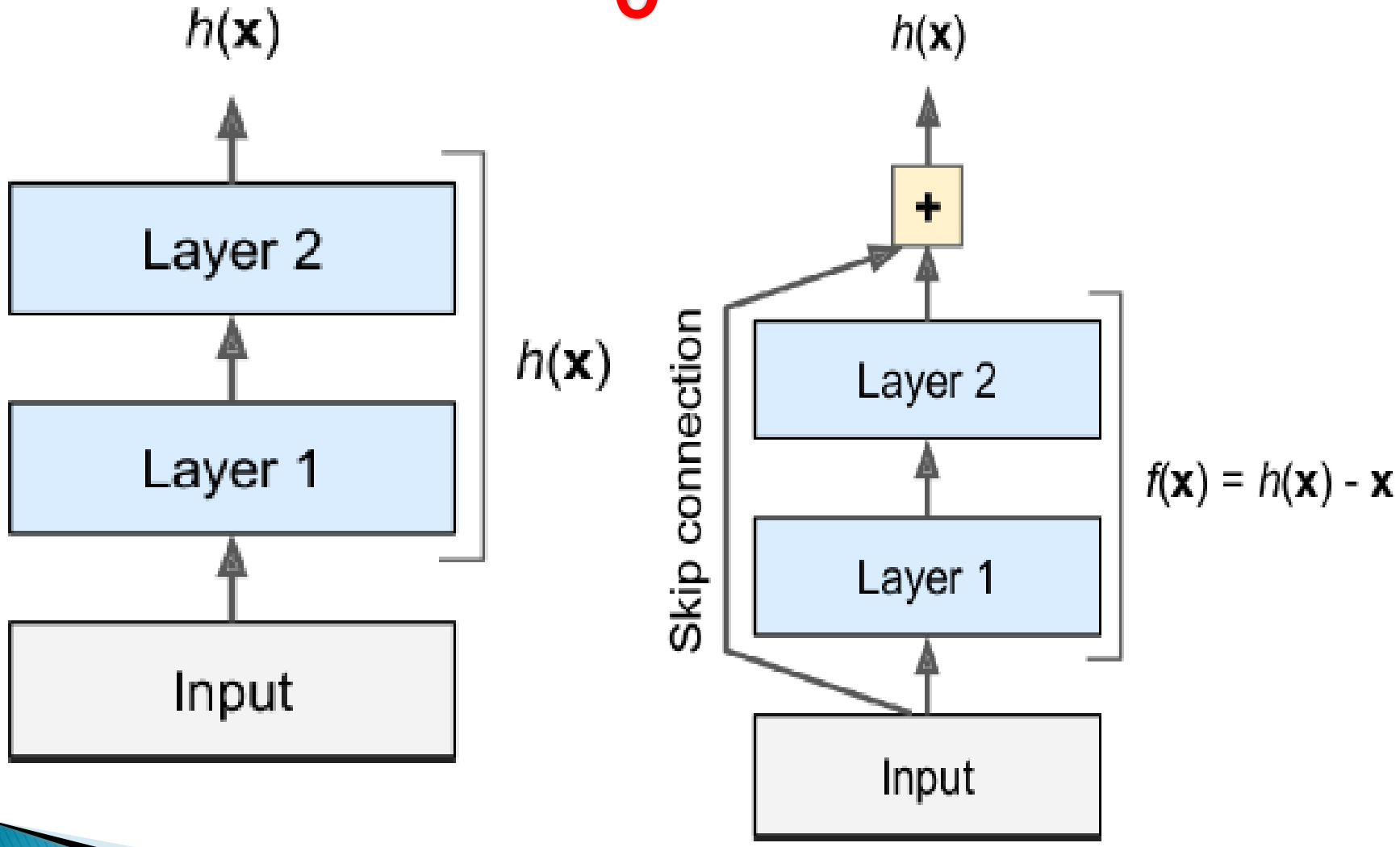
VGG: Highest memory, most operations

GoogLeNet: most efficient

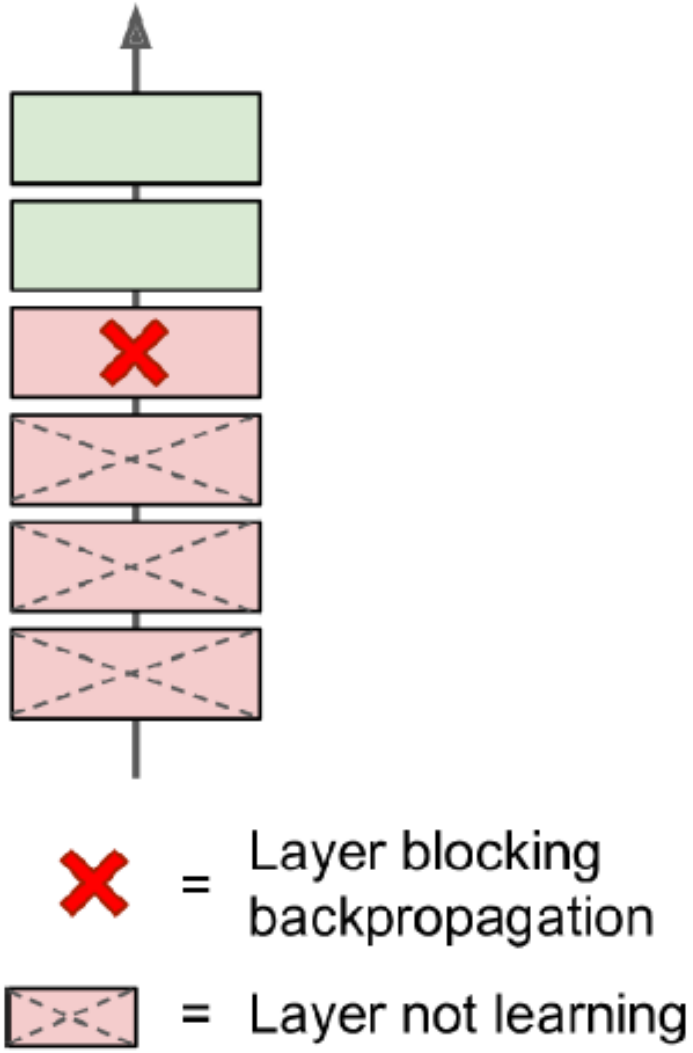
AlexNet:
Smaller compute, still memory heavy, lower accuracy

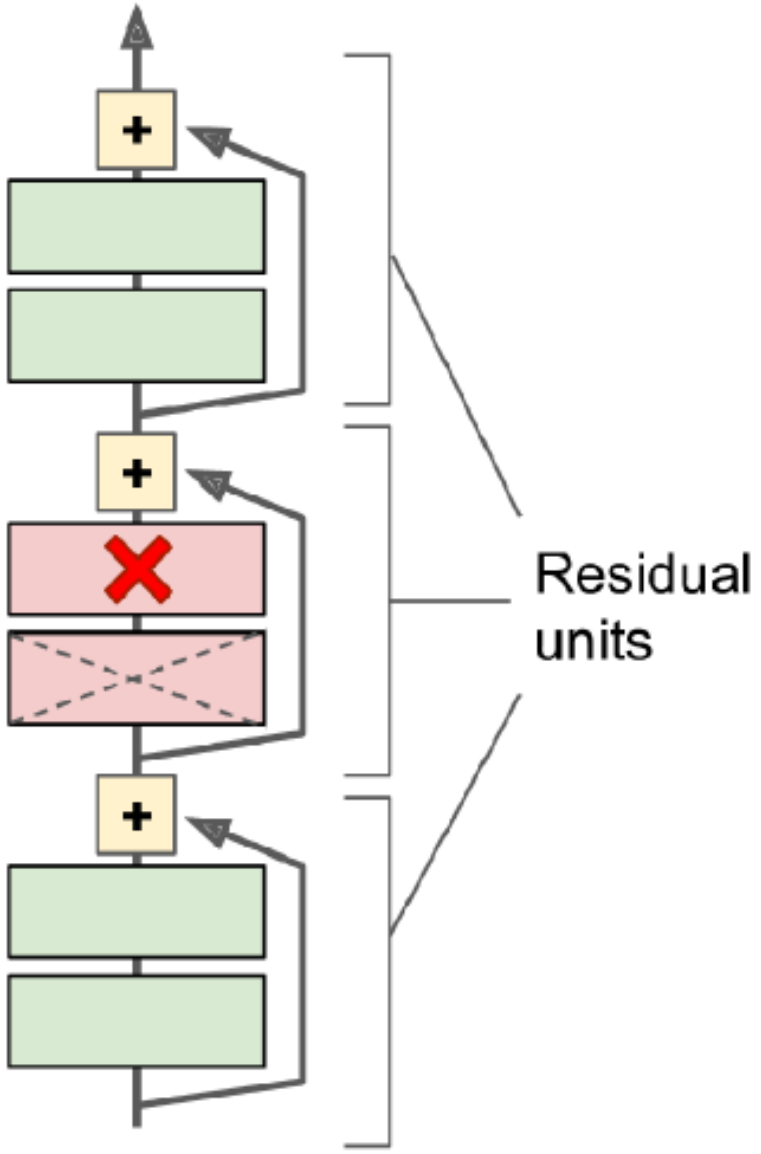
ResNet:
Moderate efficiency depending on model, highest accuracy

Residual Network (or ResNet)
2015

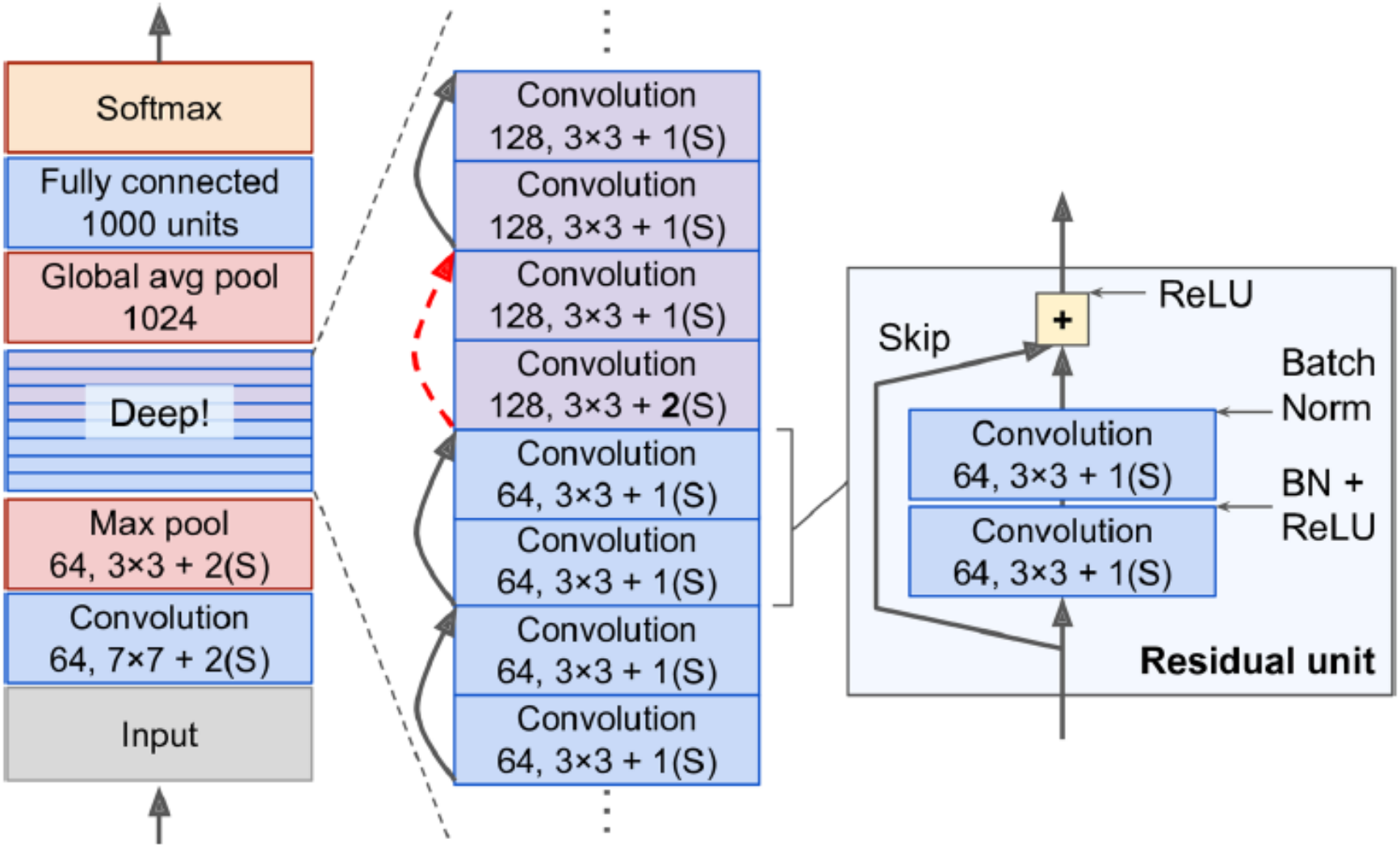


Regular and deep residual network

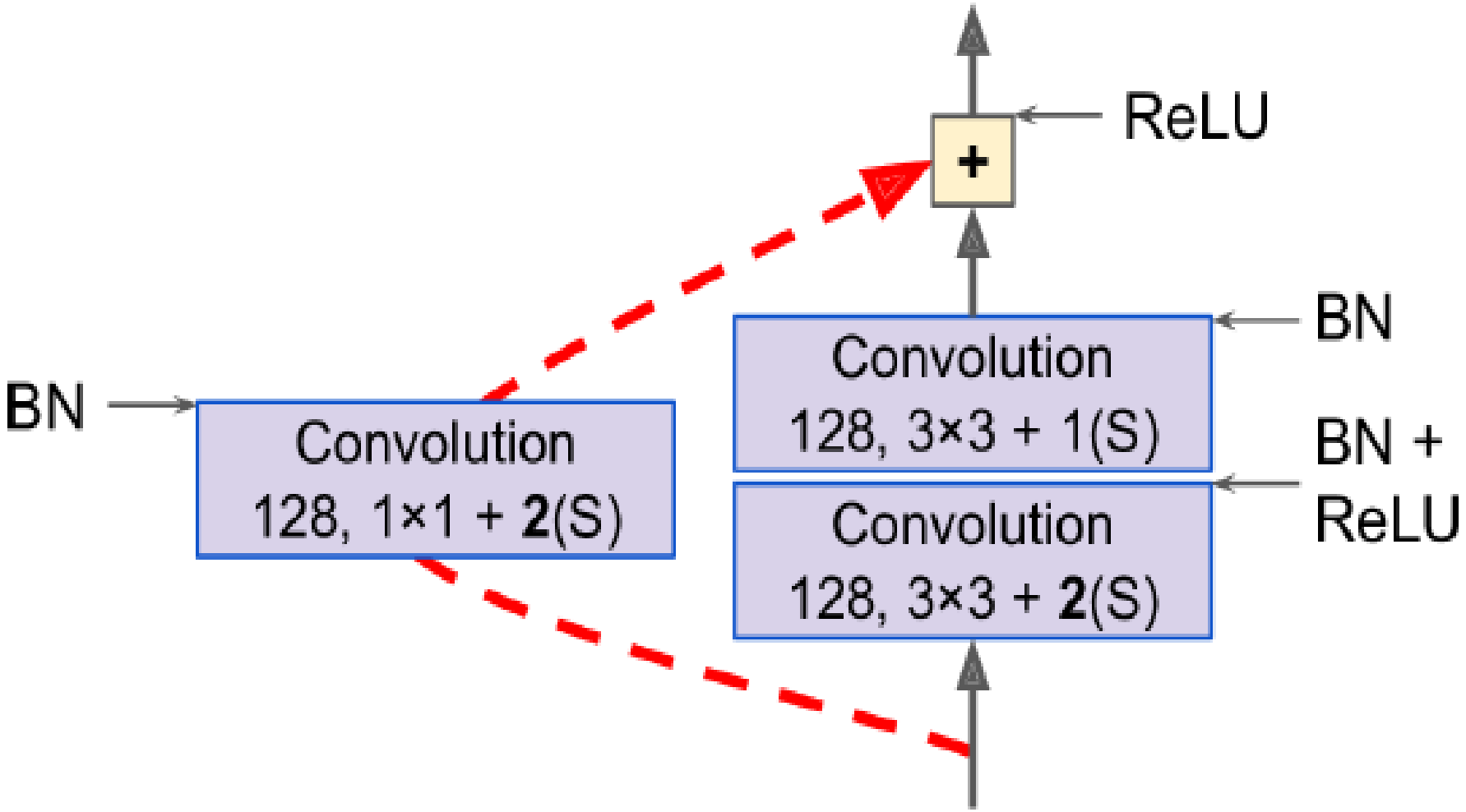




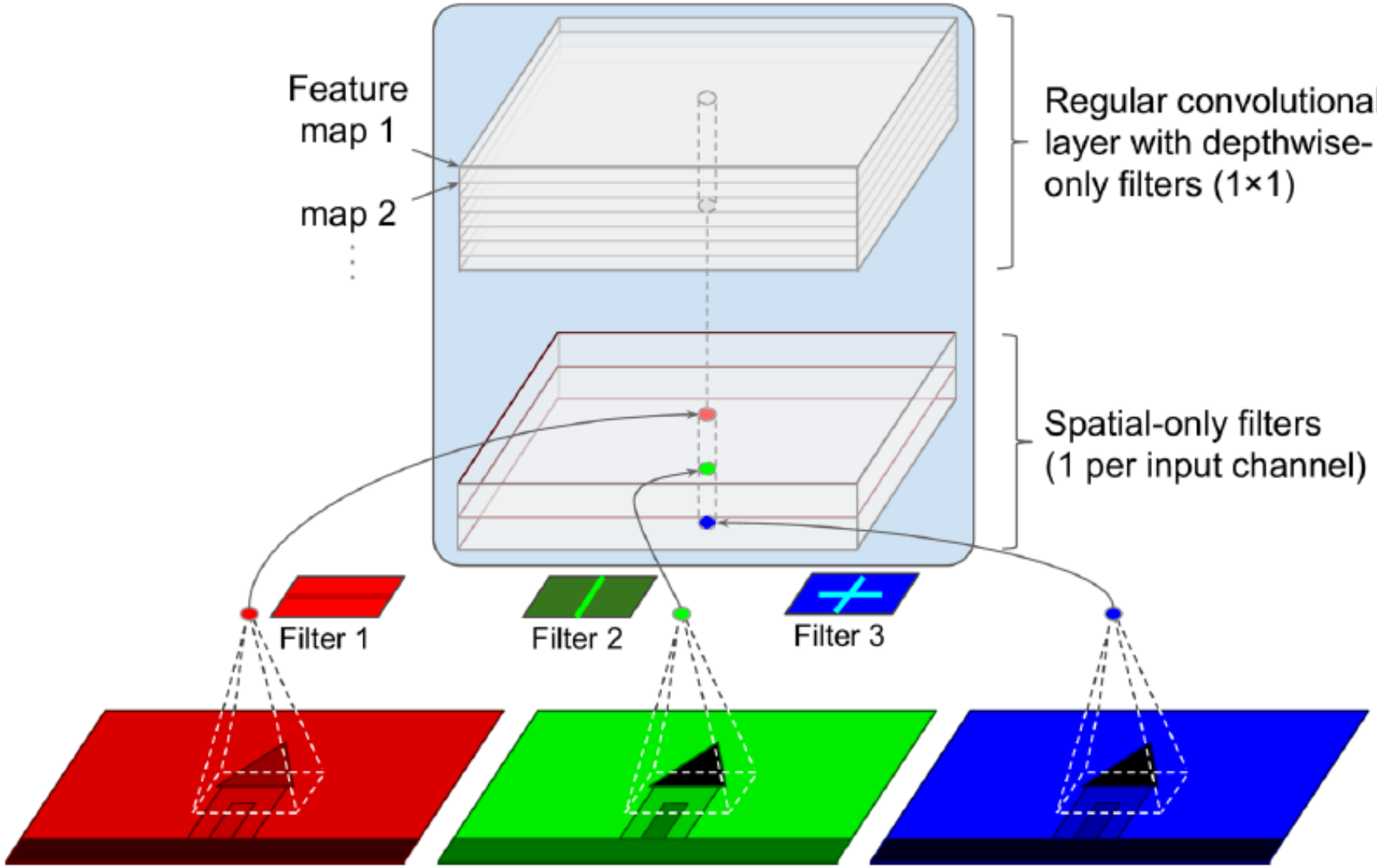
ResNet architecture



Skip connection

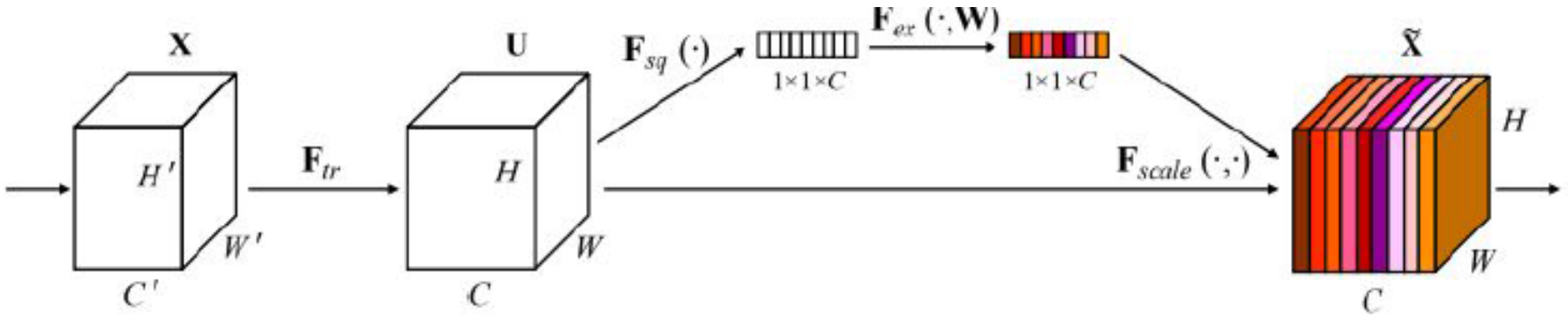


Extreme Inception (Xception) 2016

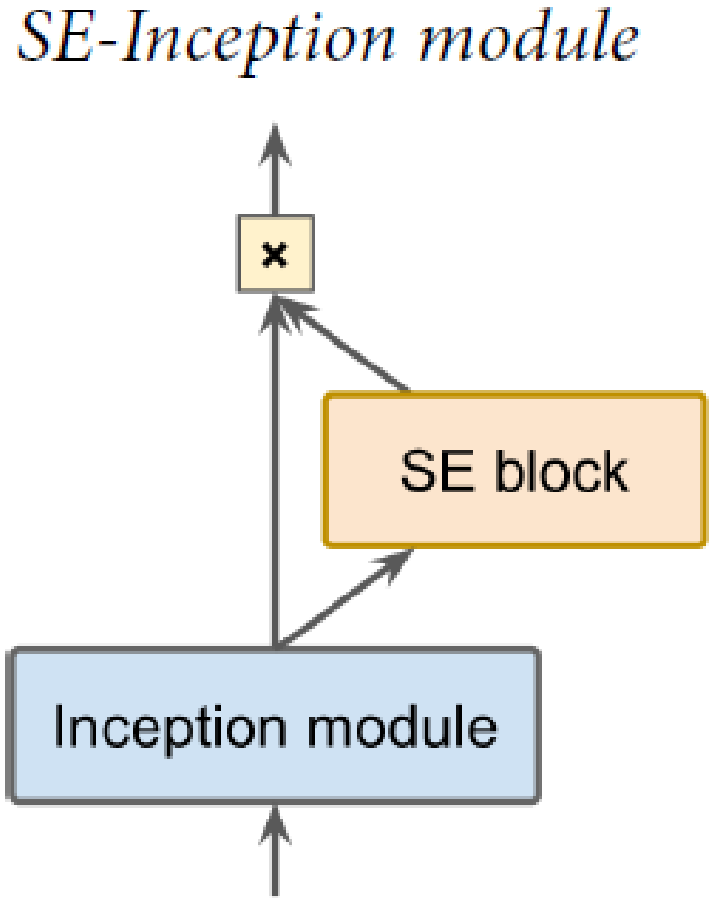


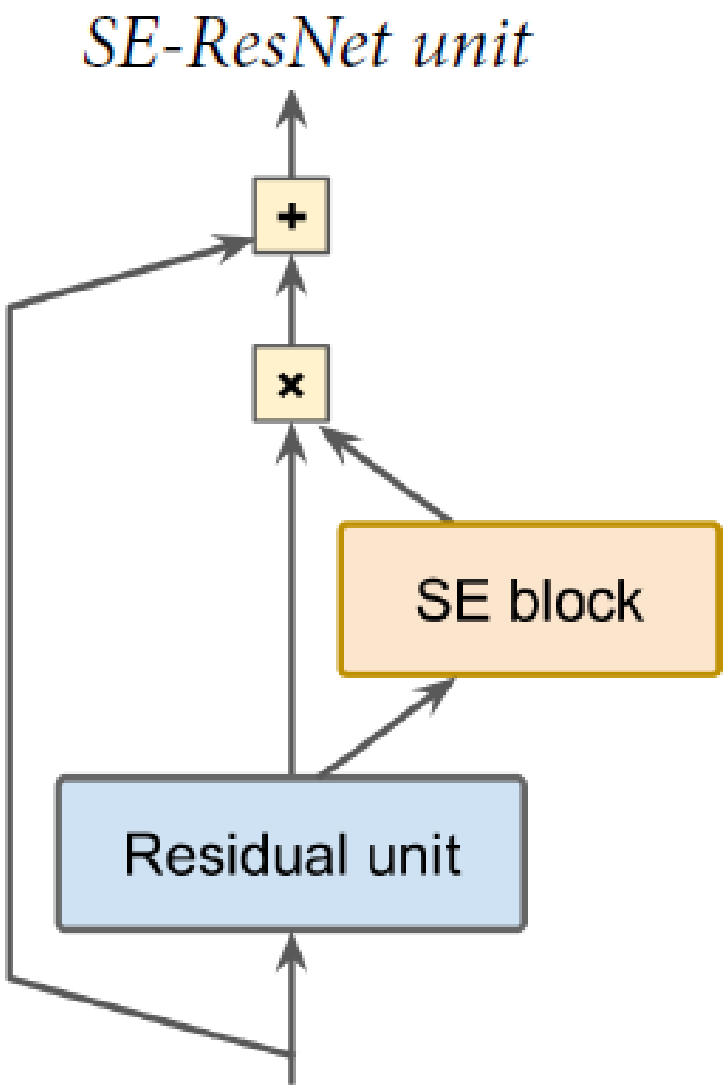
Squeeze&Excitation Network(SENet) 2017

Add a “feature recalibration” module that learns to adaptively reweight feature maps
Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
ILSVRC’17 classification winner (using ResNeXt-152 as a base architecture)

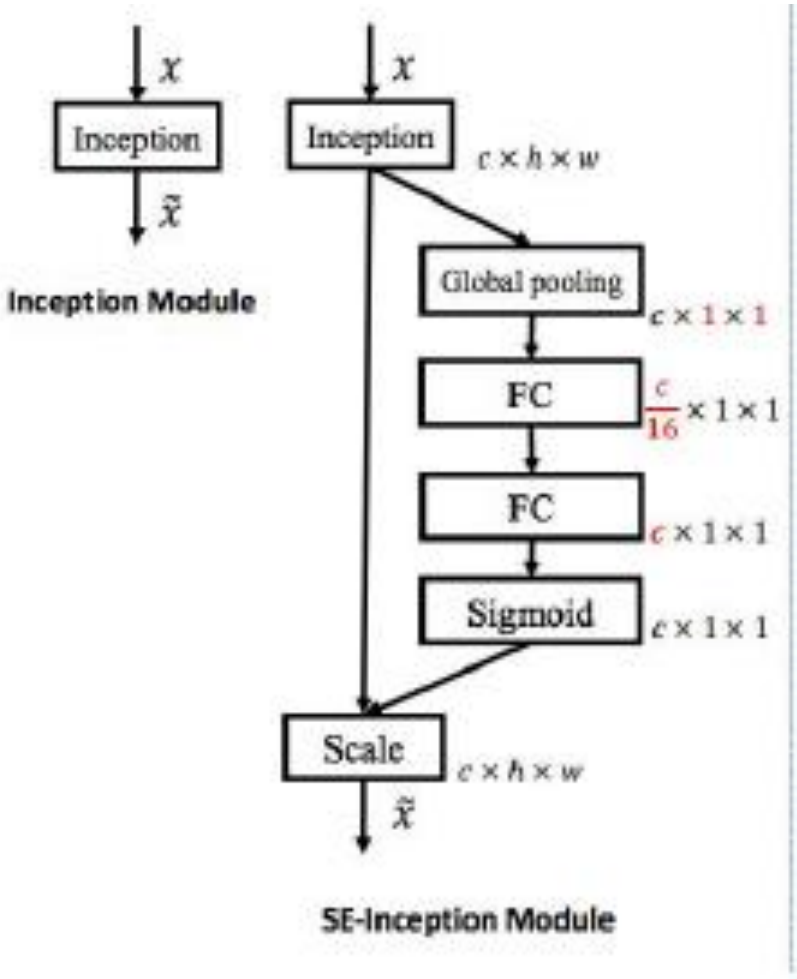


Squeeze&Excitation Network(SENet) 2017

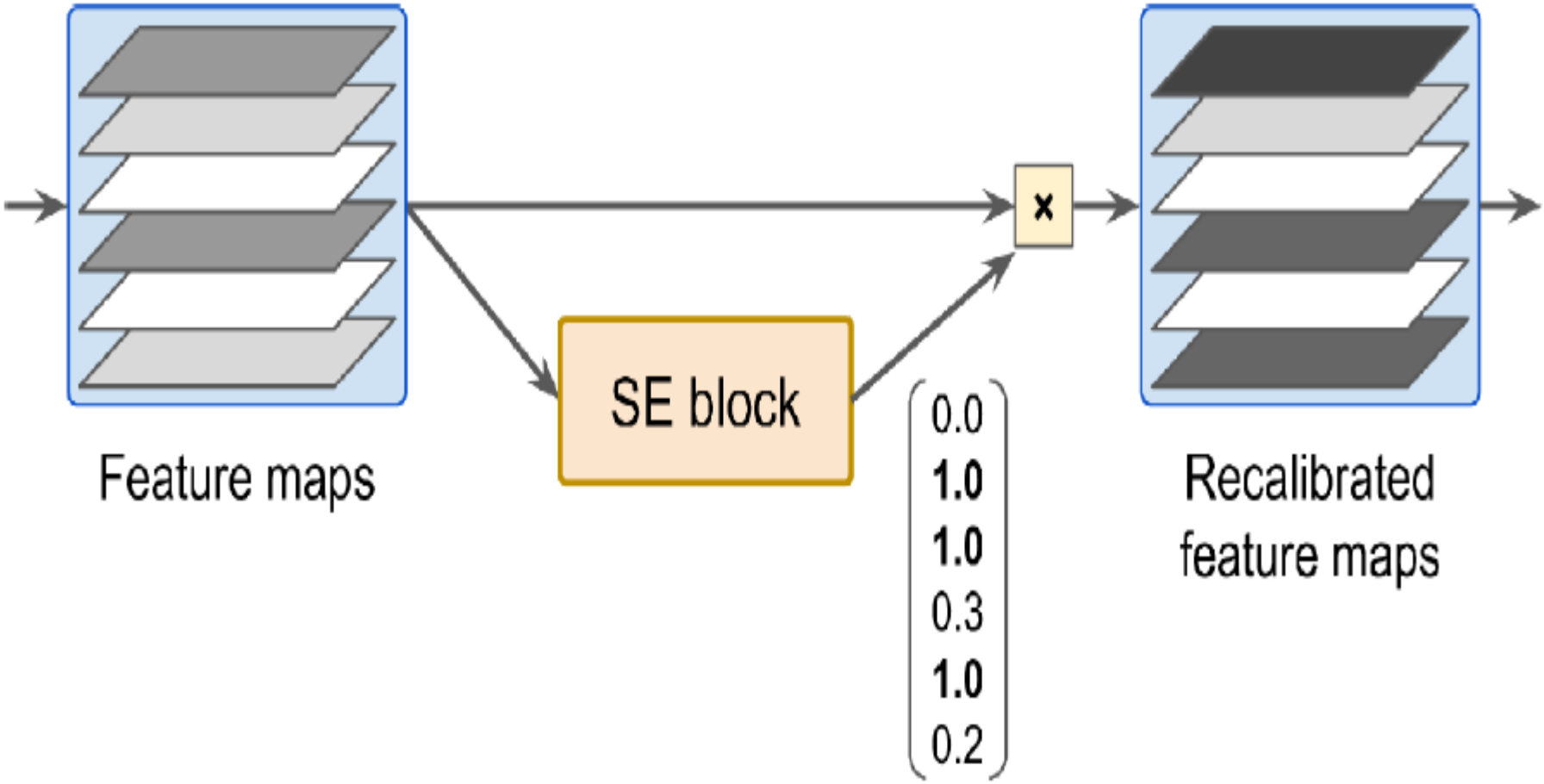




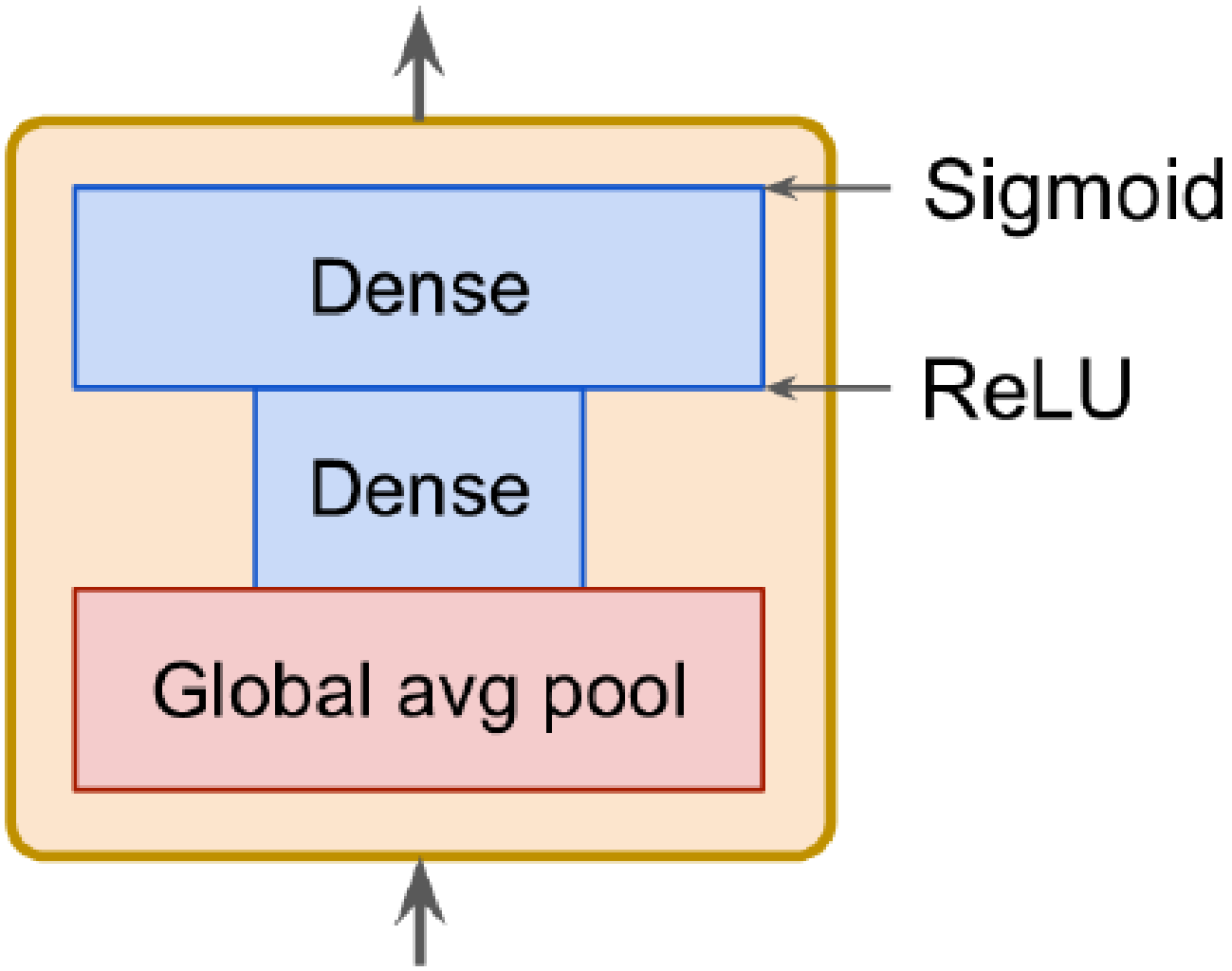
Squeeze&Excitation Network(SENet) 2017



SE block for feature map recalibration

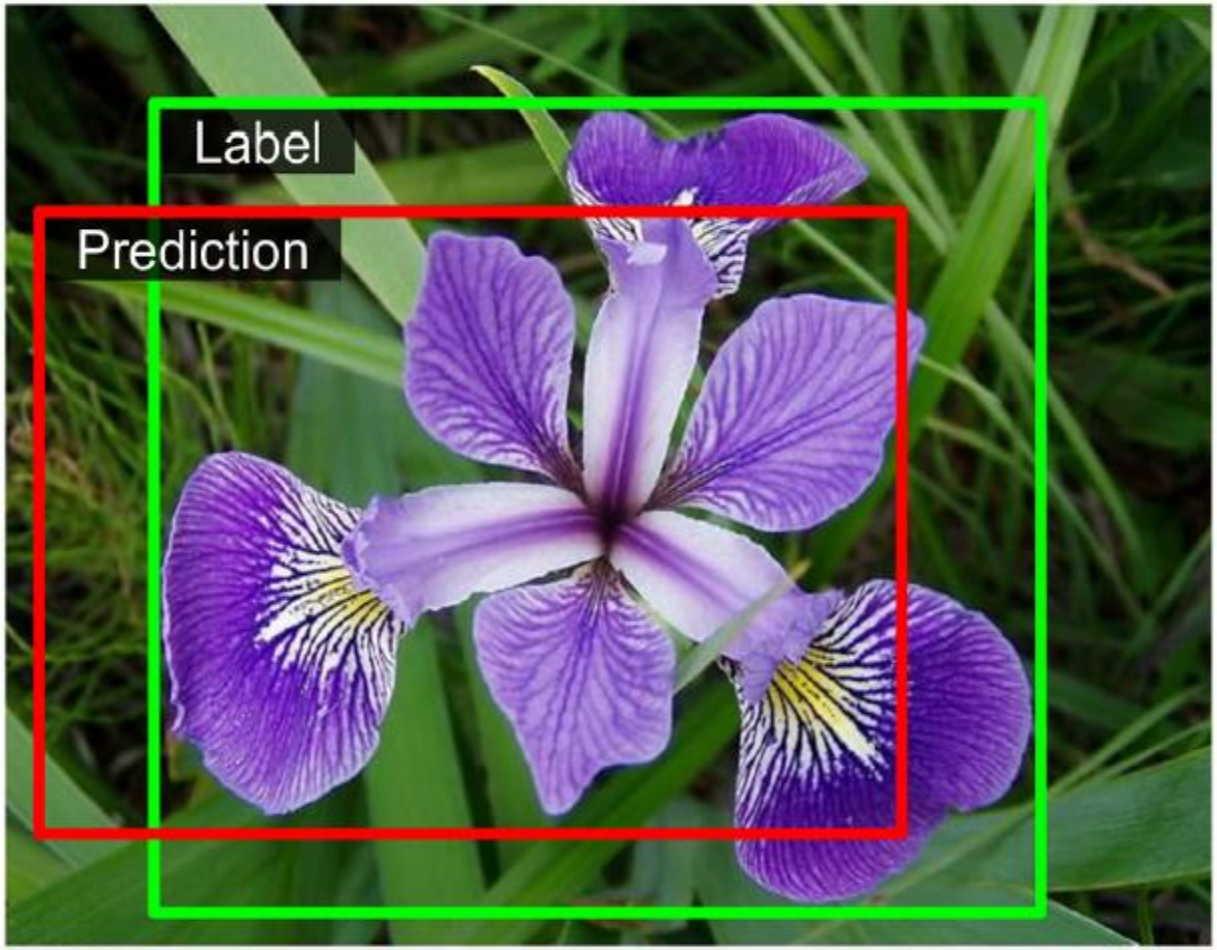


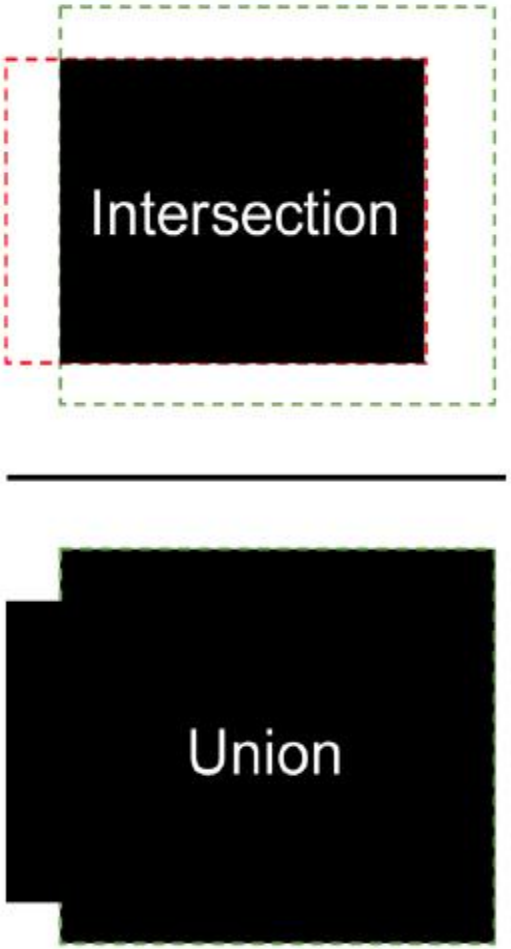
SE block architecture



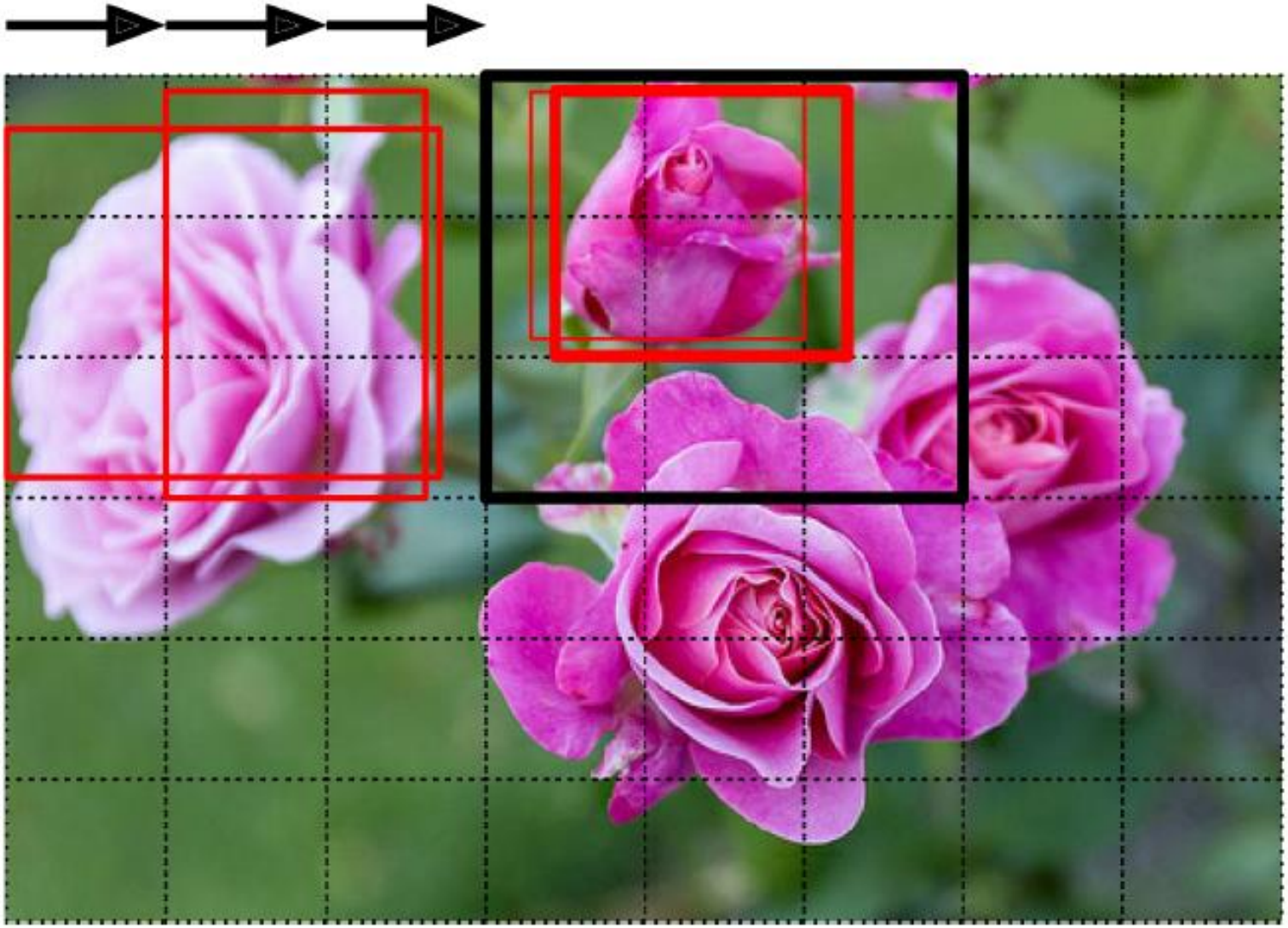
CNN for Classification and Localization

Intersection over Union (IoU) metric for bounding boxes

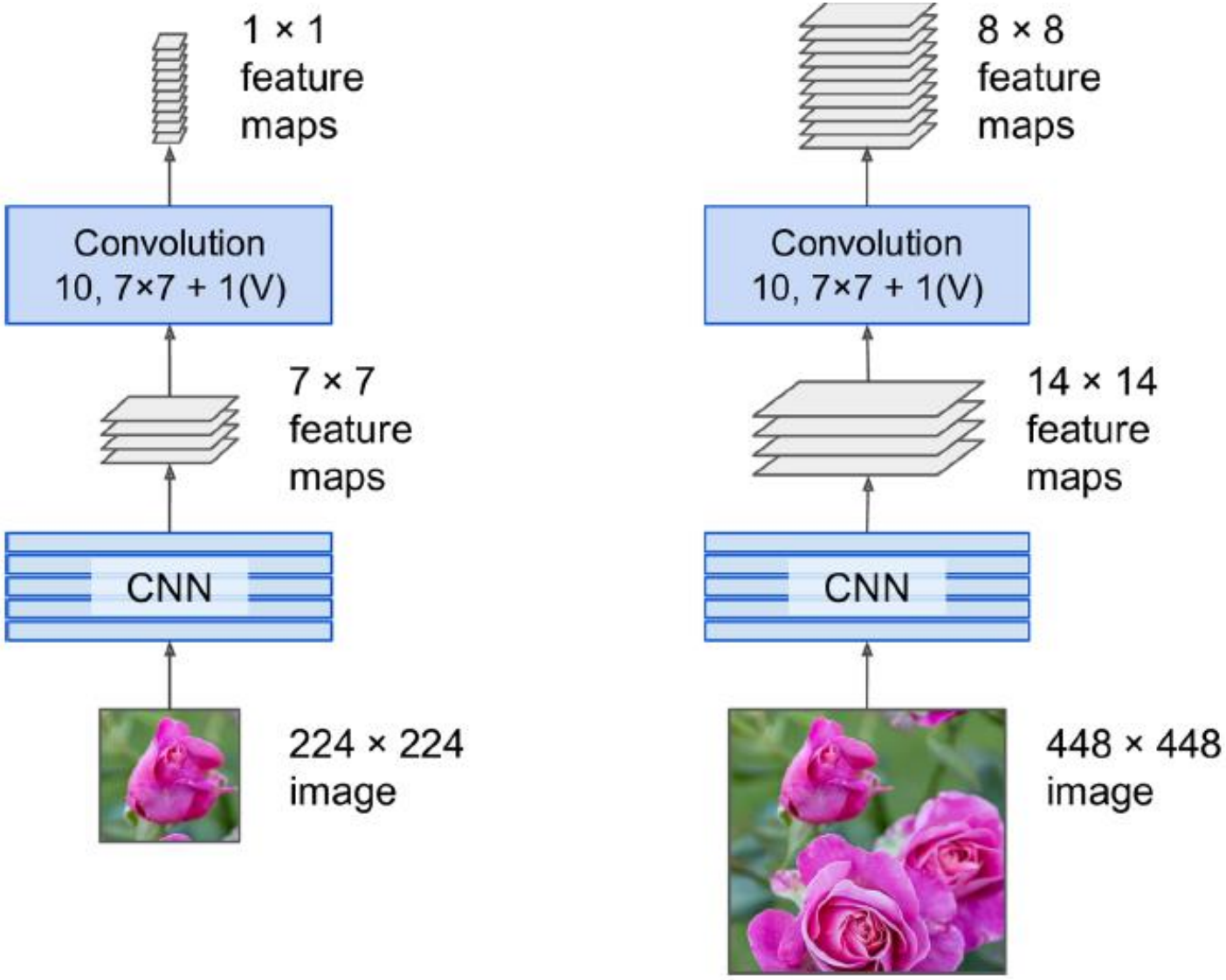




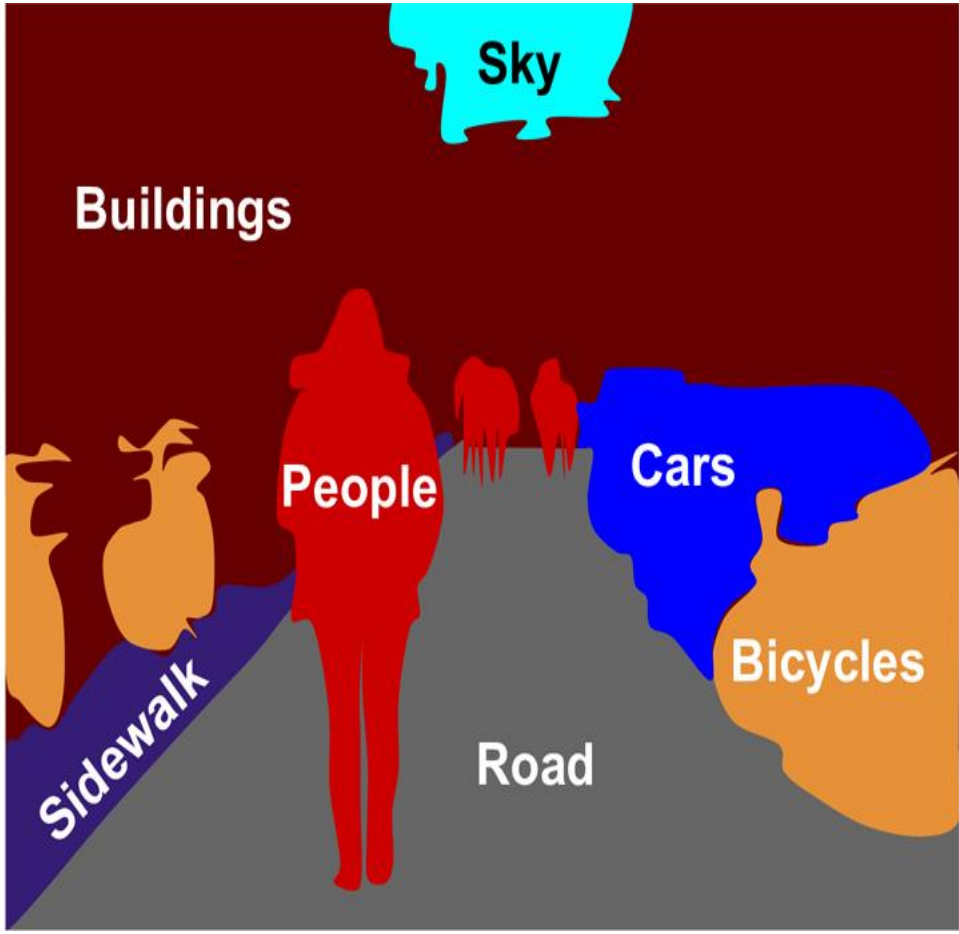
Detecting multiple objects by sliding a CNN



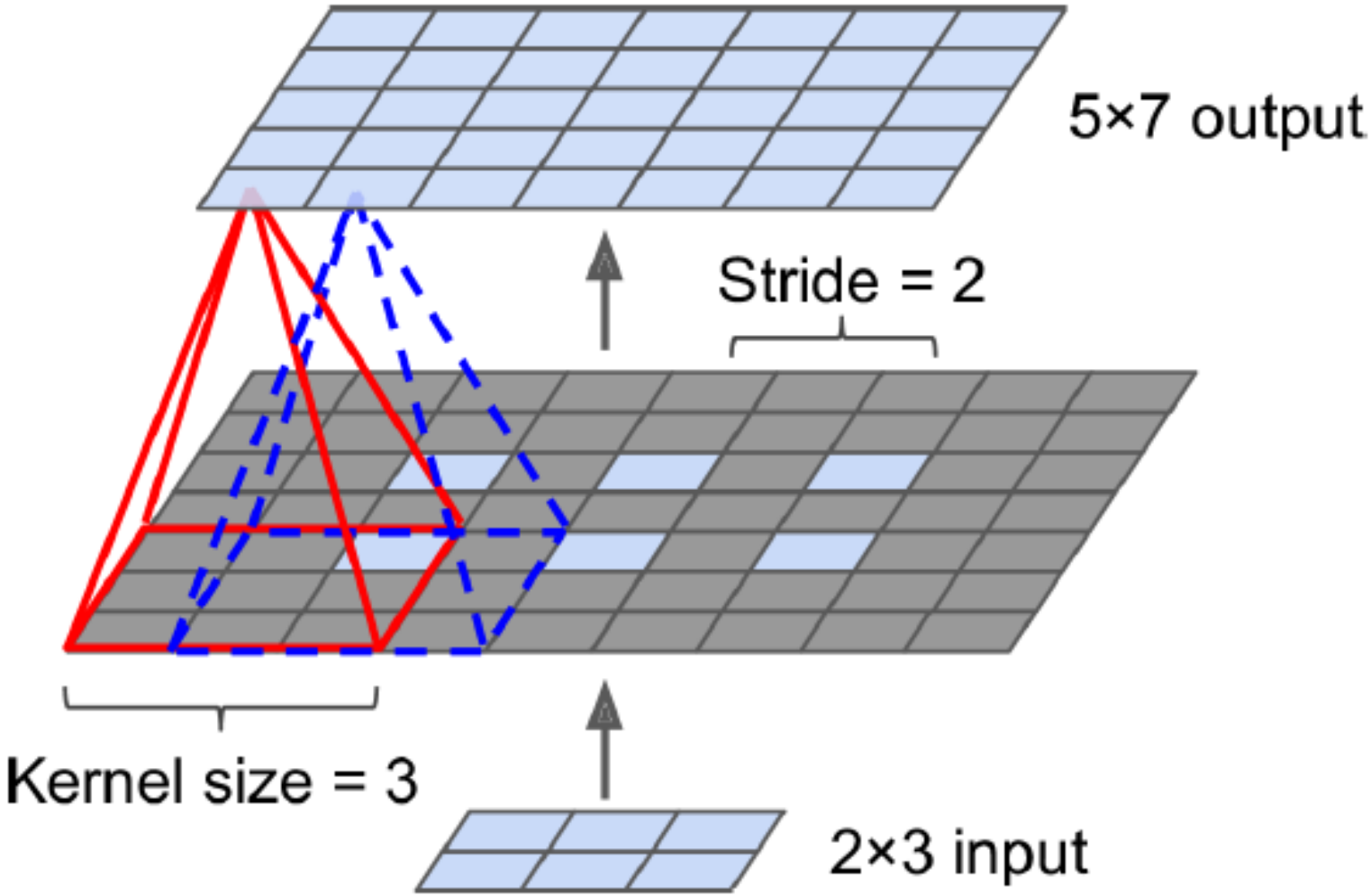
Fully Convolutional Networks



Semantic Segmentation



sampling using a transposed convolutional lay



Skip layers recover spatial resolution