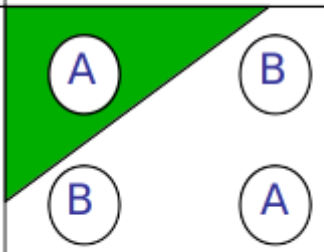
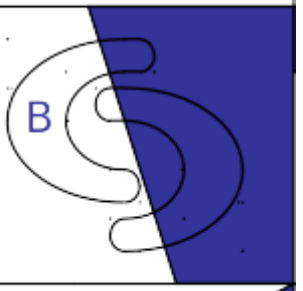
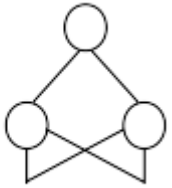
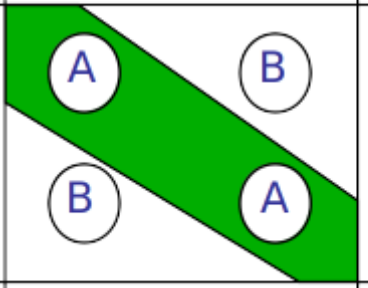
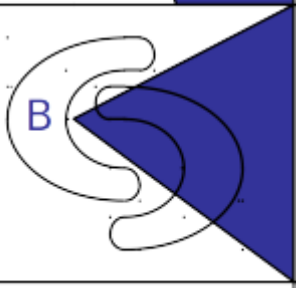
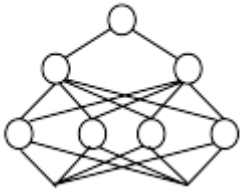
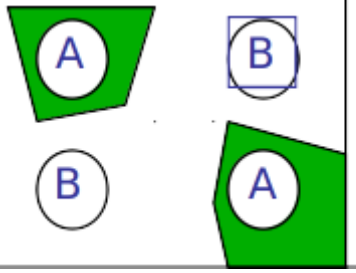
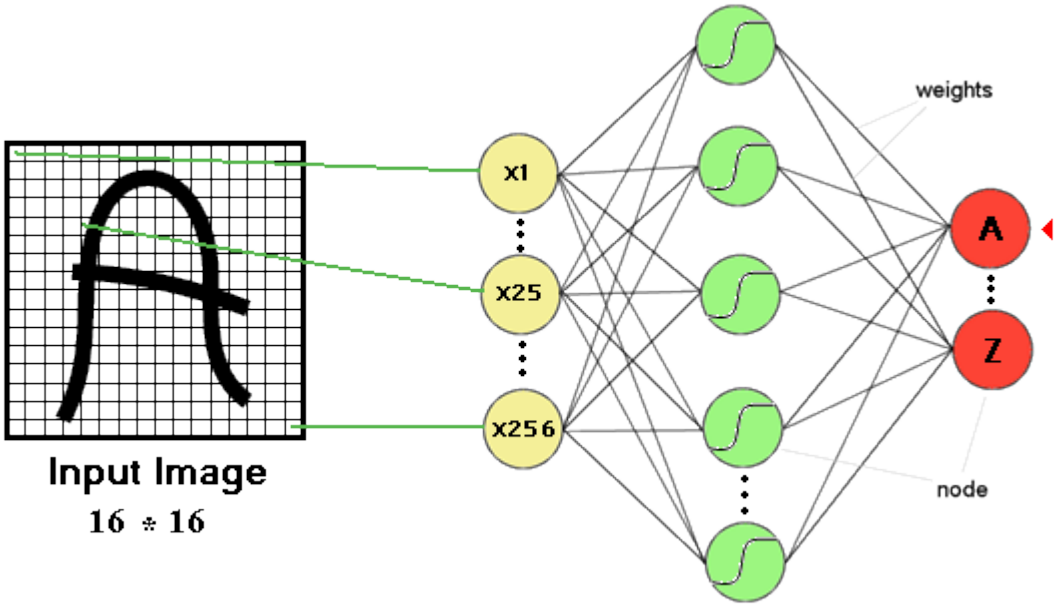
Behavior of multilayer neural networks

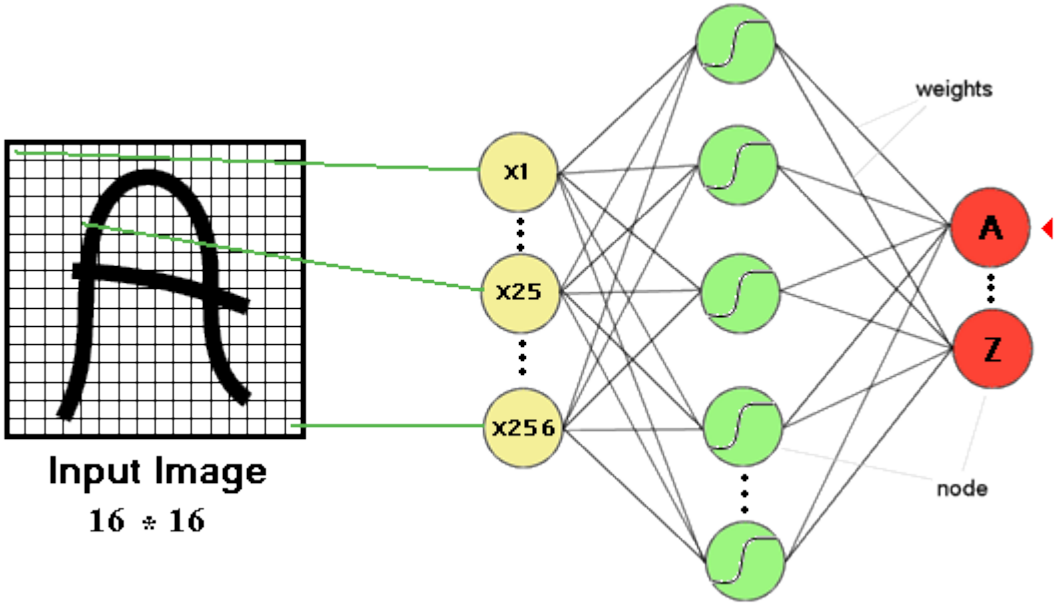
Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*
Single-Layer 	*Half Plane Bounded By Hyper plane*		
Two-Layer 	*Convex Open Or Closed Regions*		
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)		

MLP for Image classification



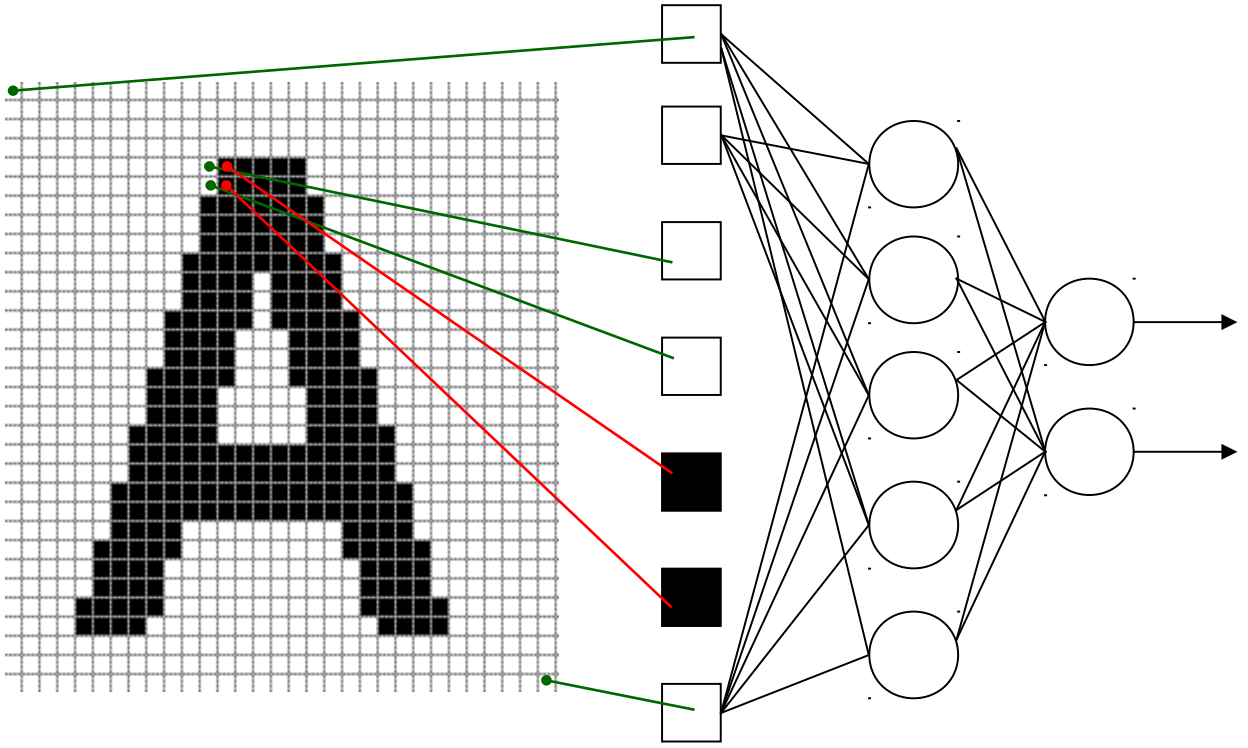
Drawbacks of previous neural networks

The number of trainable parameters becomes extremely large



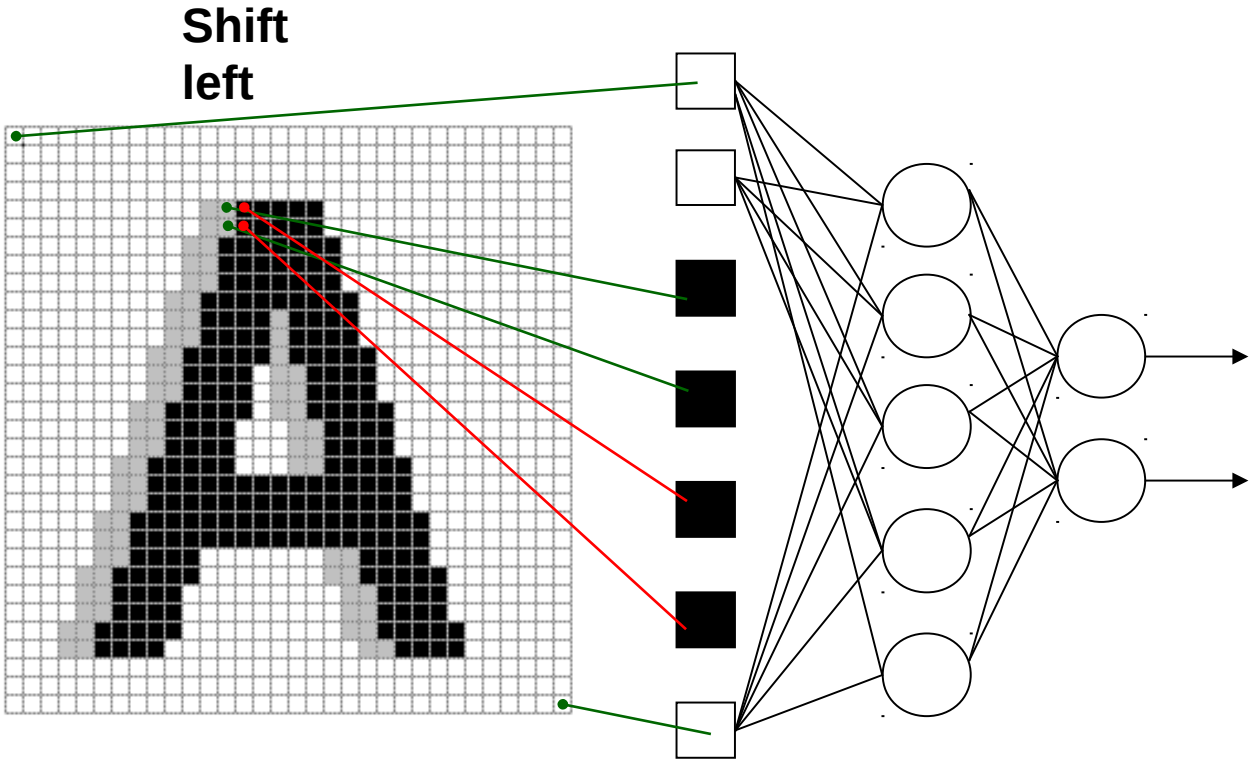
Drawbacks of previous neural networks

Little or no invariance to shifting, scaling, and other forms of distortion



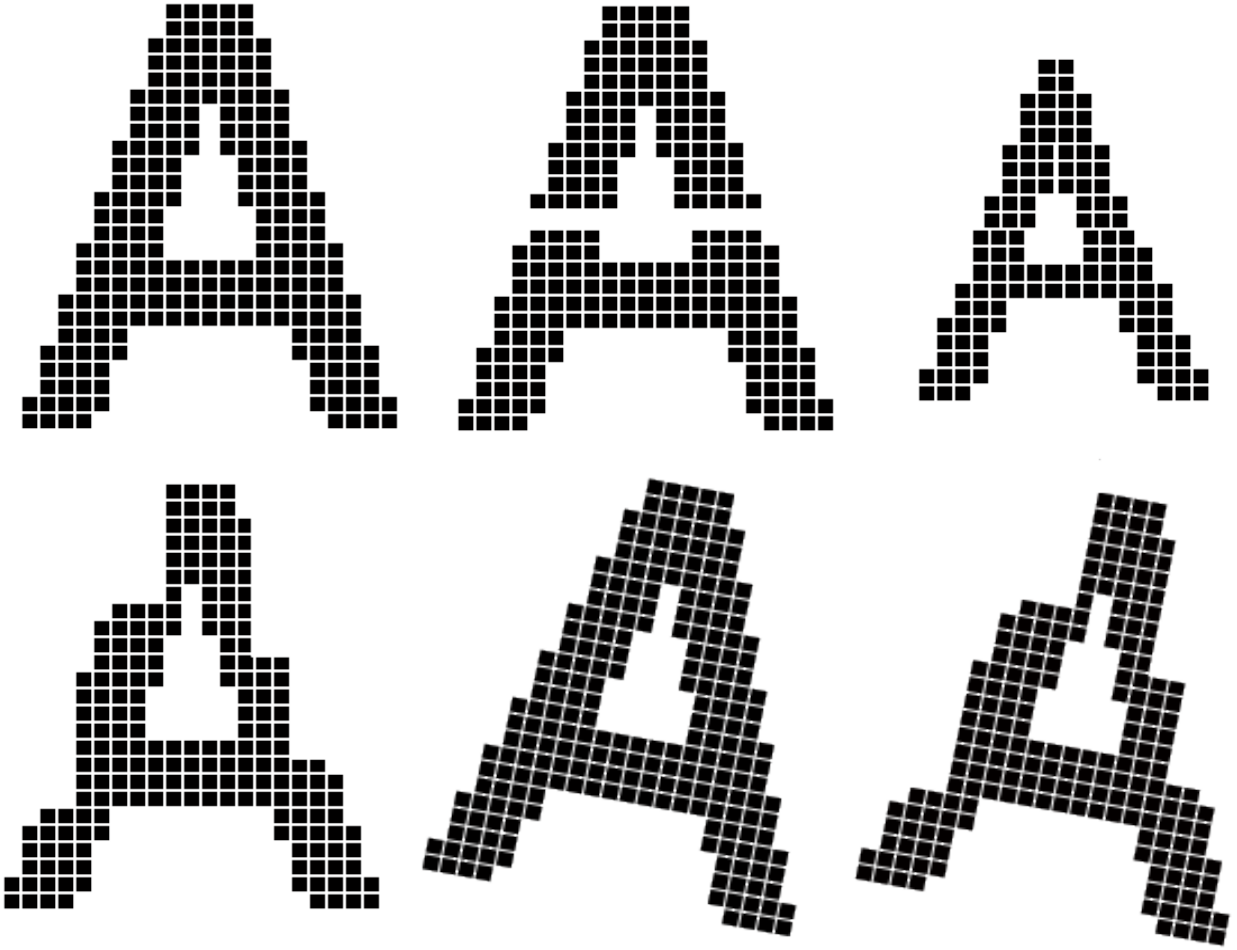
Drawbacks of previous neural networks

Little or no invariance to shifting, scaling, and other forms of distortion



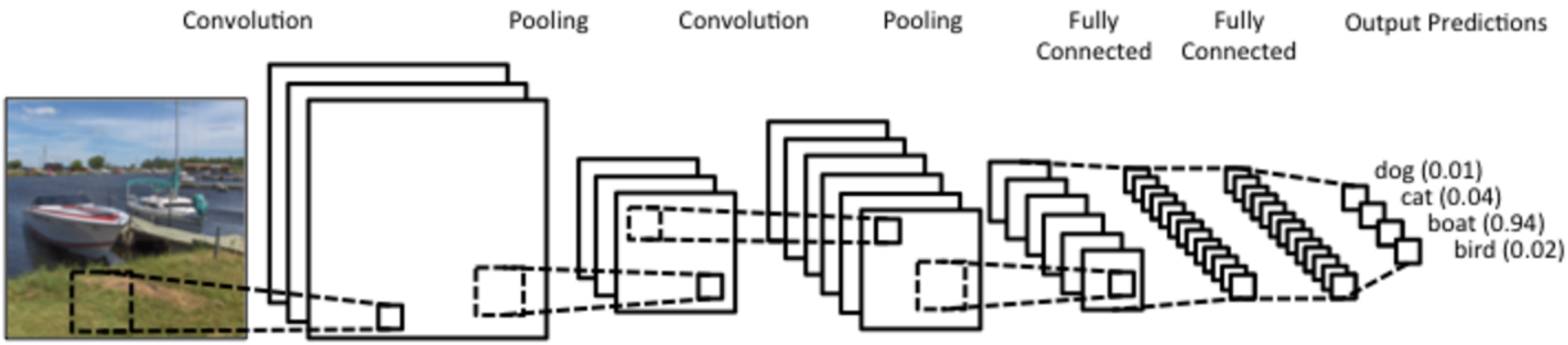
Drawbacks of previous neural networks

scaling, and other forms of distortion



Convolutional Neural Networks

Filtering	is done by	Convolution
Scaling	is done by	Max/Mean Pooling
Final Classification	is done by	Fully connected layers



Convolution

- Here, convolution is defined as follows:

z=Relu (x.w+bi), x: A Patch of input signal, w : Filter, bi: bias

Differences with signal & systems convolution:

- Bias is added
- The result is passed from an activation function (Relu)

x (Chosen Patch)

1	0	-1			
1	-2	0			
0	1	1			

a*b=(6*6)

w (Filter Window)

2	3	-9
0	5	0
0	6	0

0.5 bias

c*l=(3*3)

Activation Function Relu/ tanh

Z (unit in Feature map)

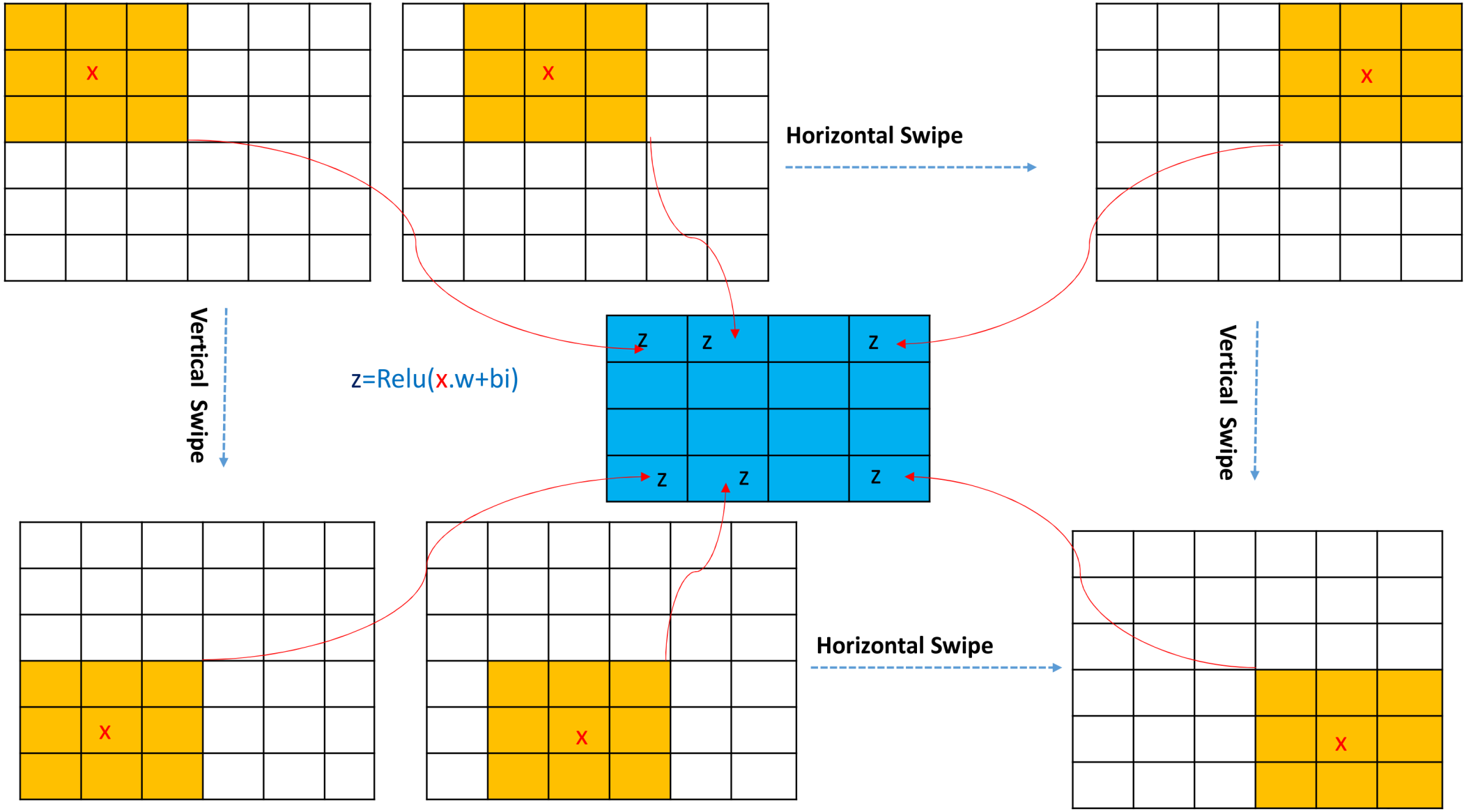
7.5			

Feature Map

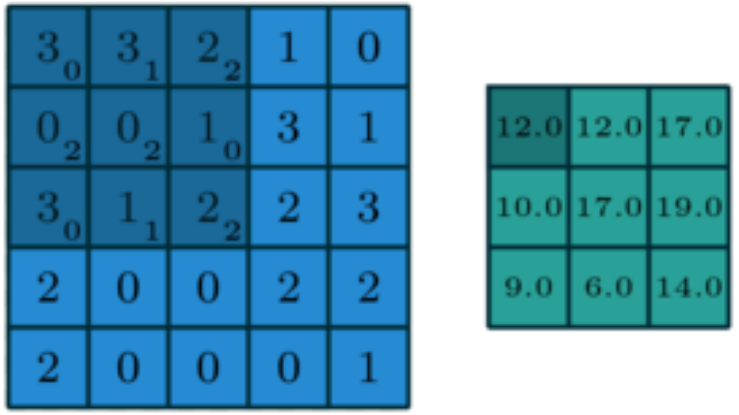
(a-c+1)*(b-l+1)=4*4

z=Relu(1*2+0*3+(-1)*4+...+1*6+1*0+0.5)

Convolution



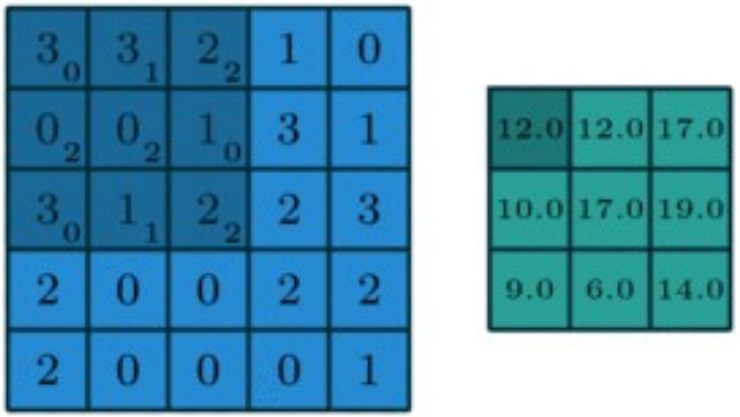
Convolution



- *x* is a 3×3 chunk (dark area) of the image (blue array)
- Each output neuron is parametrized with the 3×3 weight matrix *w* (small numbers)
- The activation obtained by sliding the 3×3 window and computing:

$$z(x) = relu(\mathbf{w}^T x + b)$$

Convolution



- *x* is a 3×3 chunk (dark area) of the image (blue array)
- Each output neuron is parametrized with the 3×3 weight matrix *w* (small numbers)
- The activation obtained by sliding the 3×3 window and computing:

$$z(x) = relu(\mathbf{w}^T x + b)$$

•Motivations

Local connectivity

- A neuron depends only on a few local input neurons
- Translation invariance

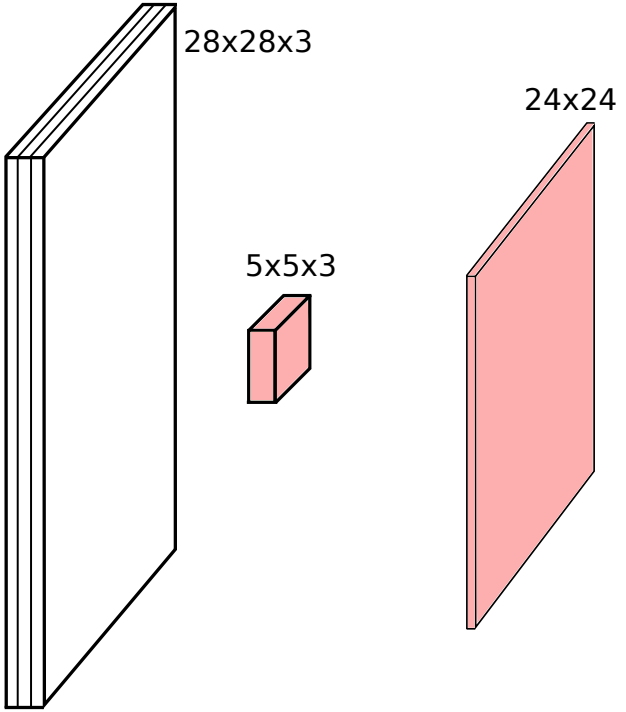
Comparison to Fully connected

- Parameter sharing: reduce overfitting
- Make use of spatial structure: strong prior for vision!

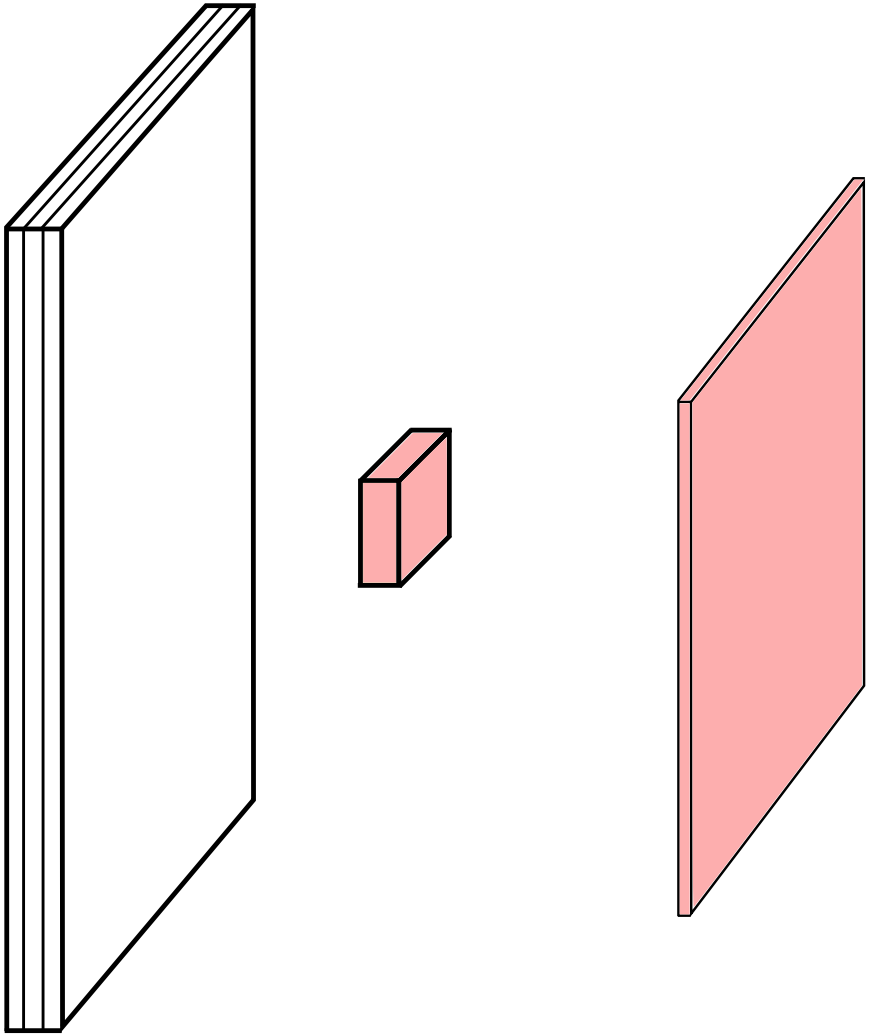
Channels

- Colored image = tensor of shape (height, width, channels)

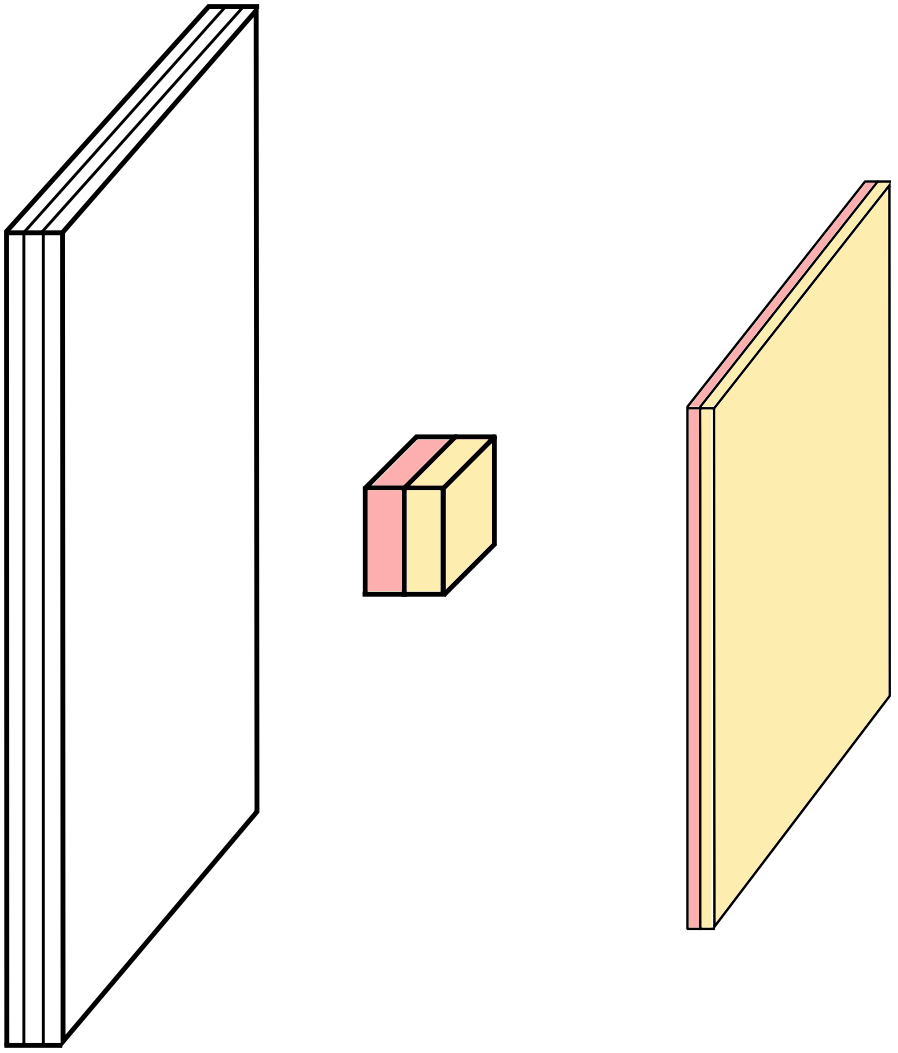
- Convolutions are usually computed for each channel and summed:



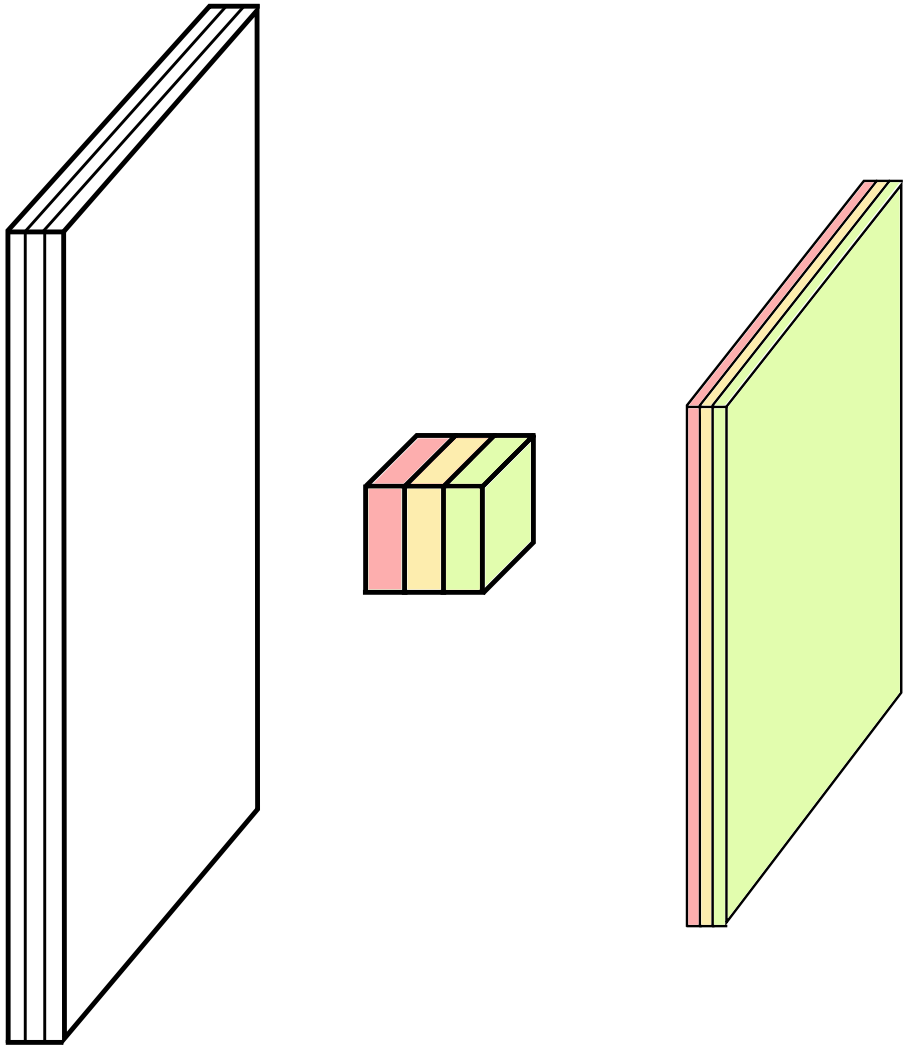
Multiple convolutions



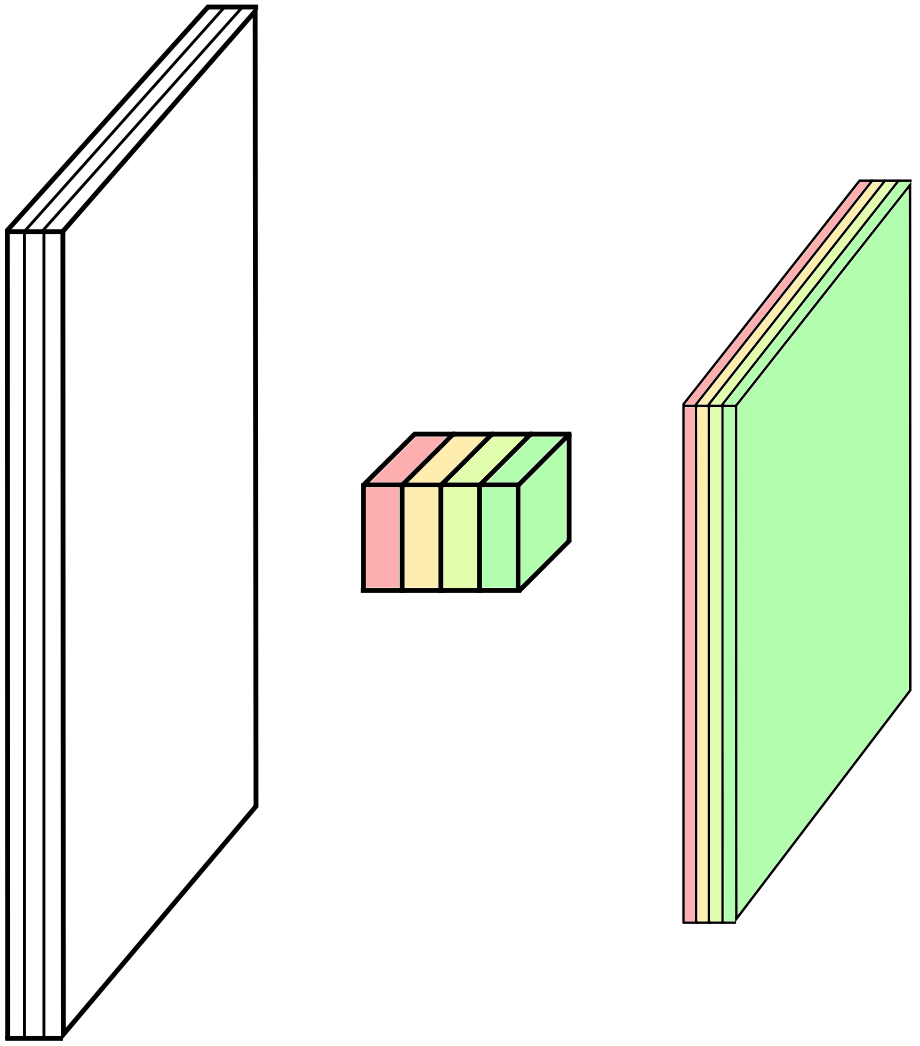
Multiple convolutions



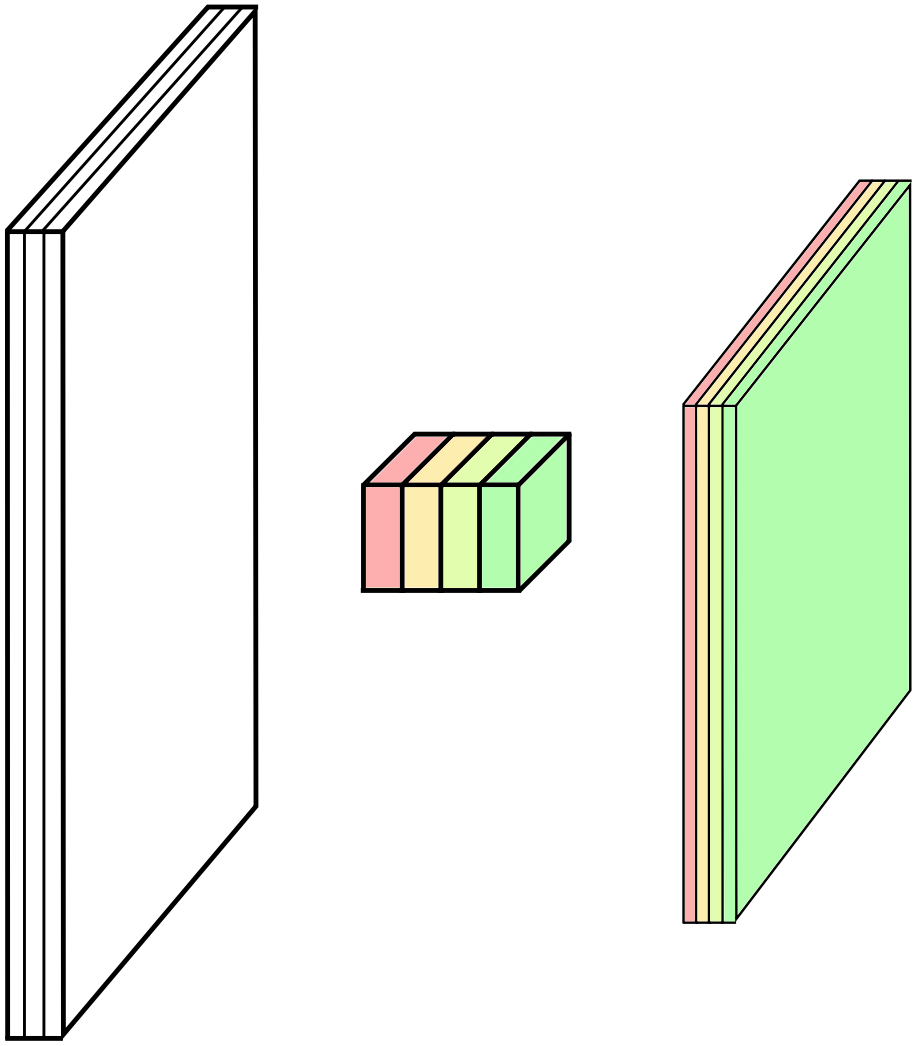
Multiple convolutions



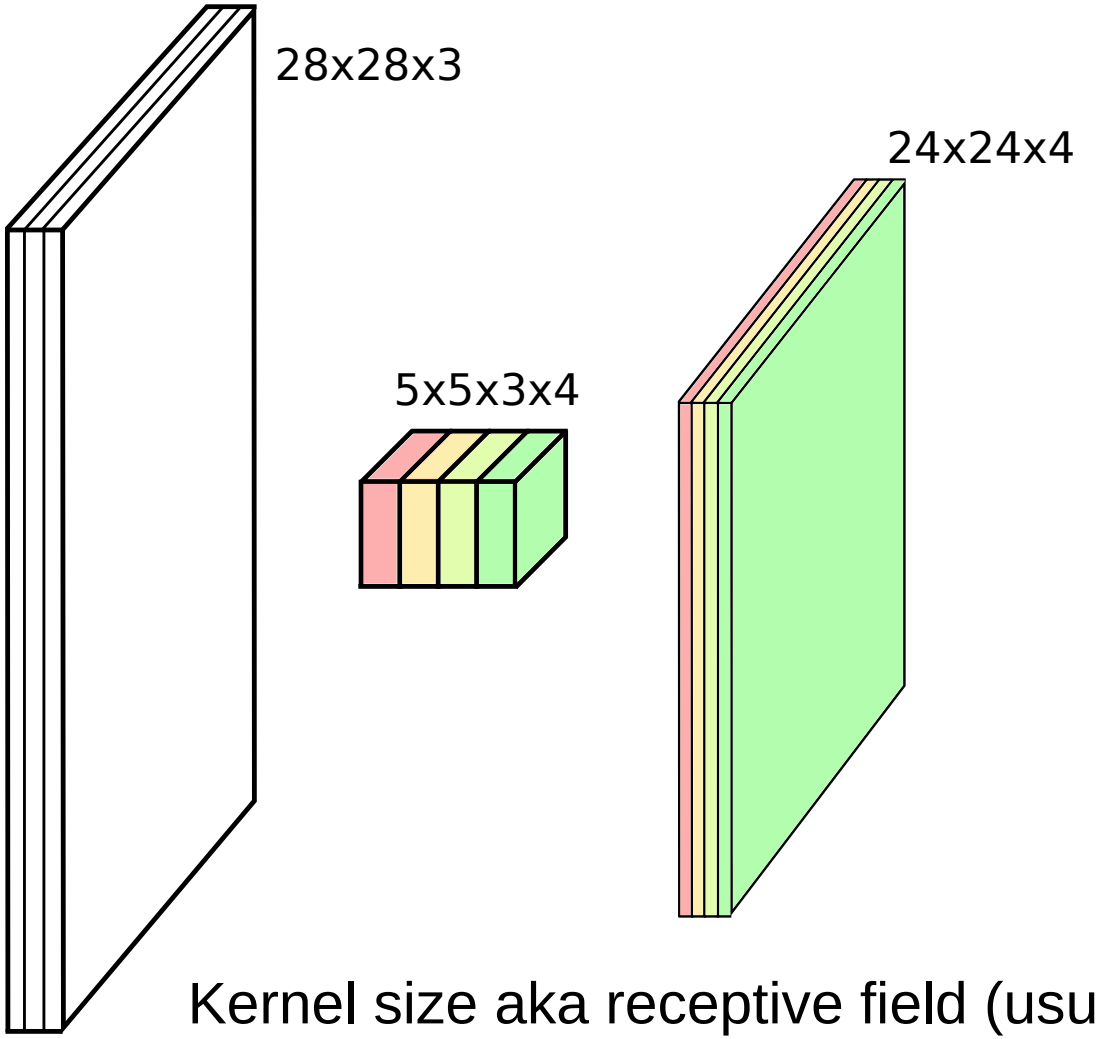
Multiple convolutions



Multiple convolutions



Multiple convolutions

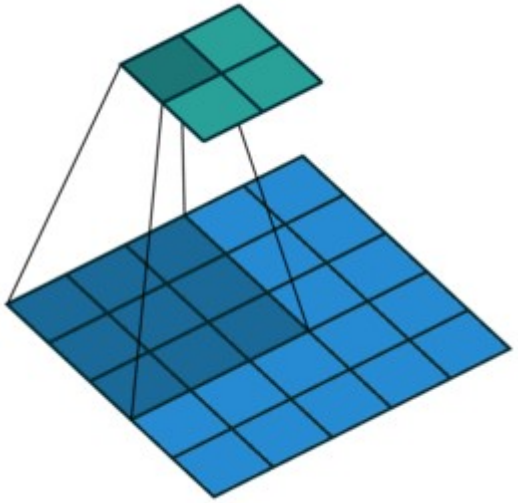


Kernel size aka receptive field (usually 1, 3, 5, 7, 11)

Output dimension: length - kernel_size + 1

Strides

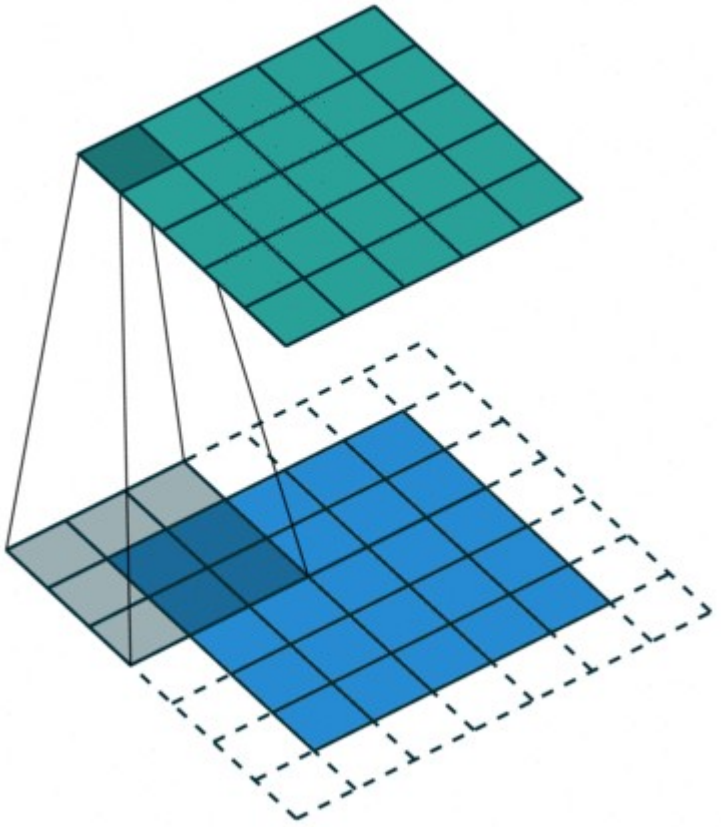
- Strides: increment step size for the convolution operator
- Reduces the size of the output map



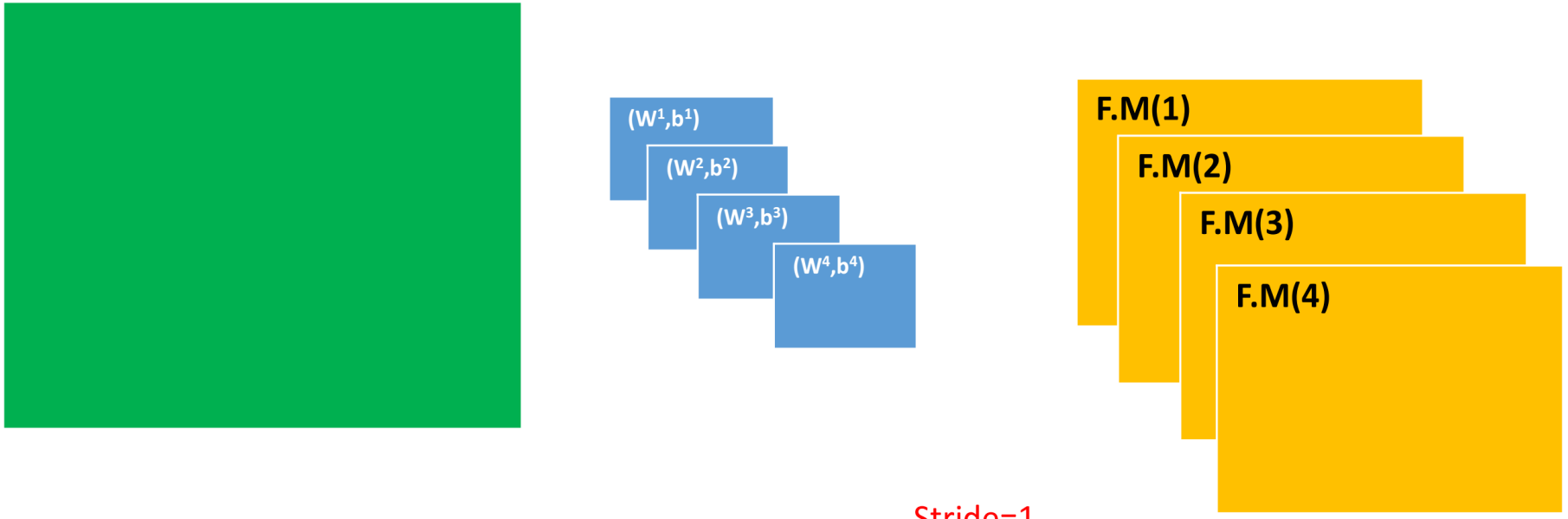
Example with kernel size 3×33×3 and a stride of 22 (image in blue)

Padding

- Padding: artificially fill borders of image
- Useful to keep spatial dimension constant across filters
- Useful with strides and large receptive fields
- Usually: fill with 0s



More than one Feature Map



Input Image(a*b)

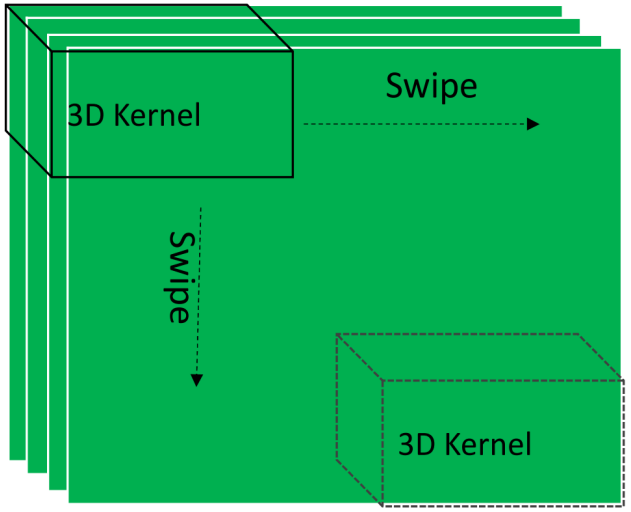
4 Filters (c*l)

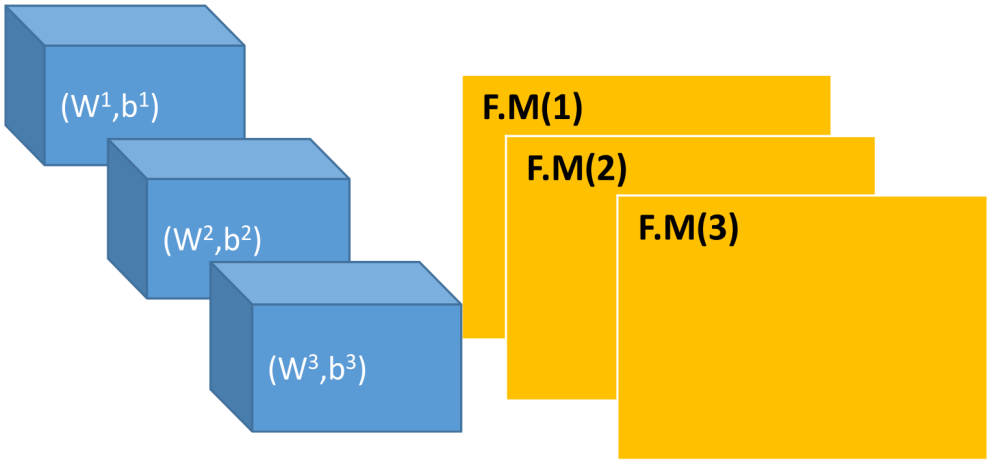
4 Feature Maps (a-c+1*b-l+1)

•Every Feature map contains an independent extracted frequented sub-pattern.

3D Convolution

1. For Colored Images
2. For 2nd, 3rd, …. Convolution layers





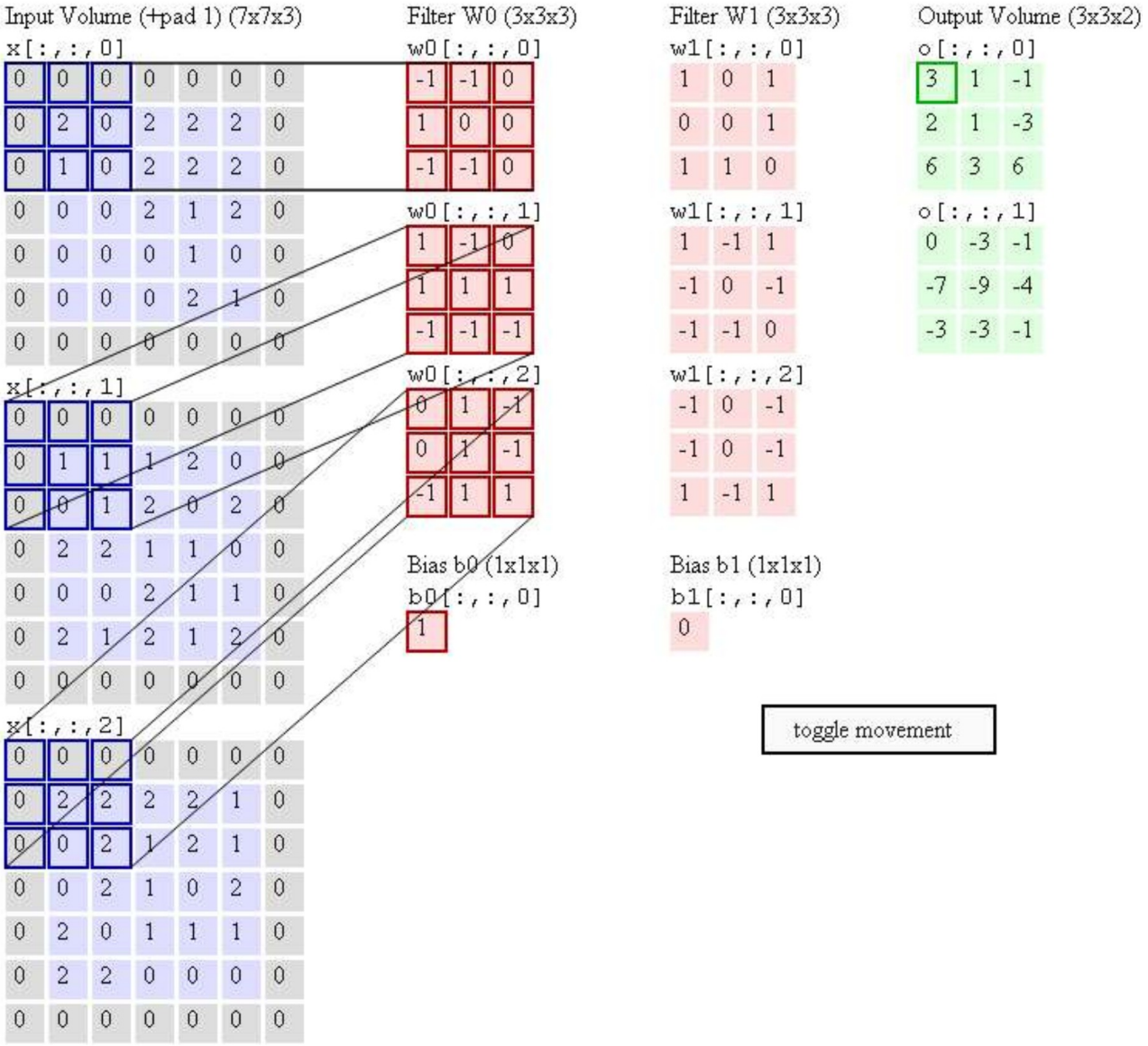
Image(a*b*4)

3 3D-Filters (c*l*4)+ 3 biases

Stride=1

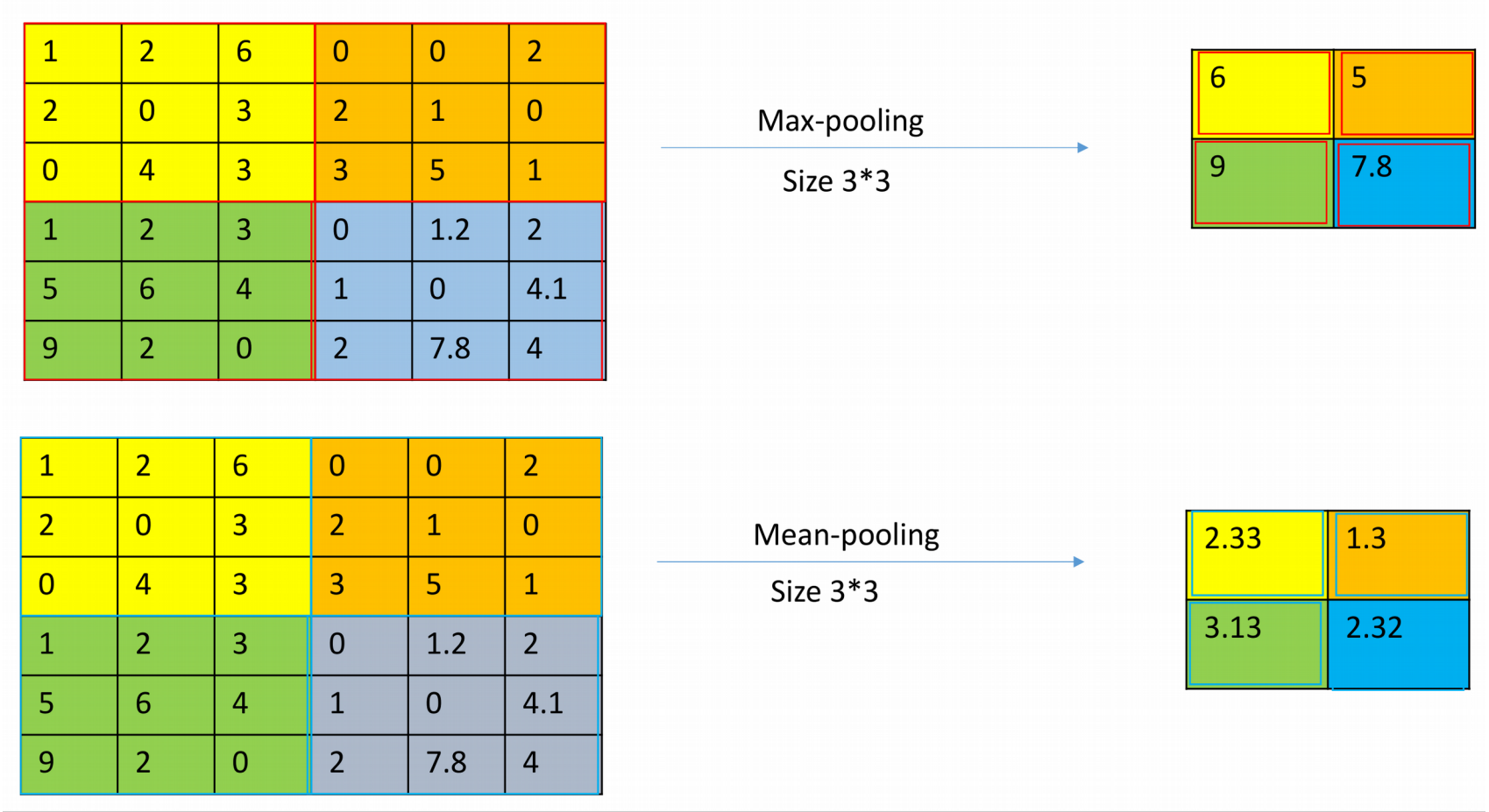
3 Feature Maps (a-c+1*b-l+1)

An illustrative example of 3D convolution swiping patches are done with Stride=2



Pooling

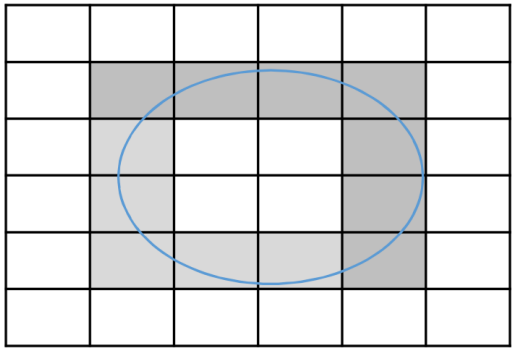
Max-pooling , Mean-pooling
(Dimension-Reduction(subsampling) & scaling)

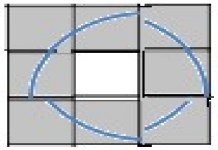


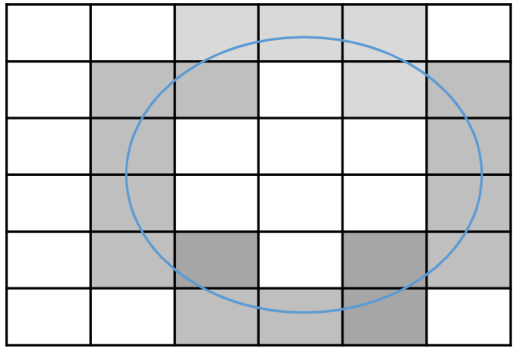
Pooling

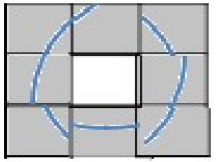
Scaling by Max-Pooling

Pooling Size 3*3

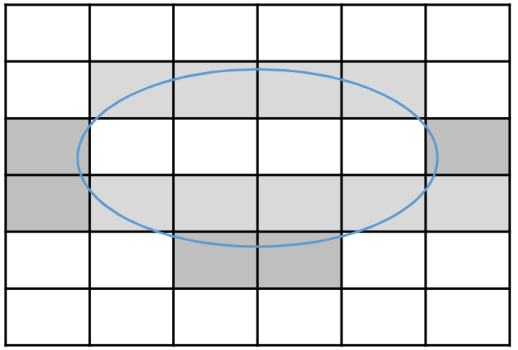


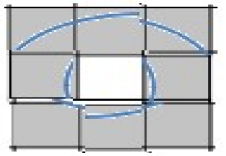


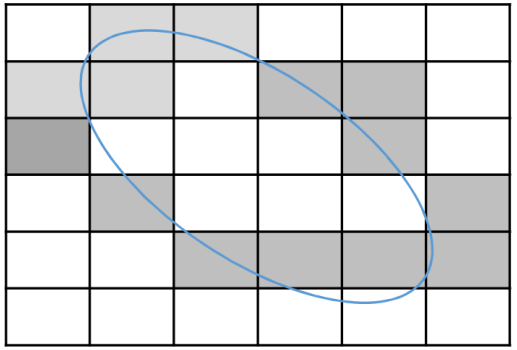


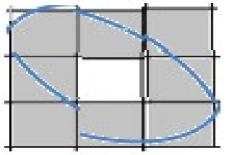


Pooling Size 3*3









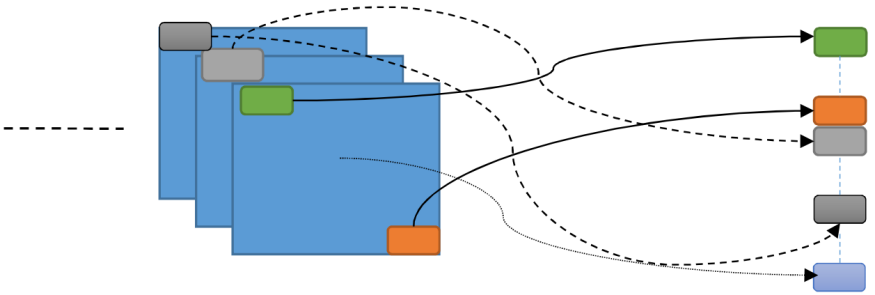
Correlation coefficient (0.3 → 0.6)

About Convolution and Pooling layers

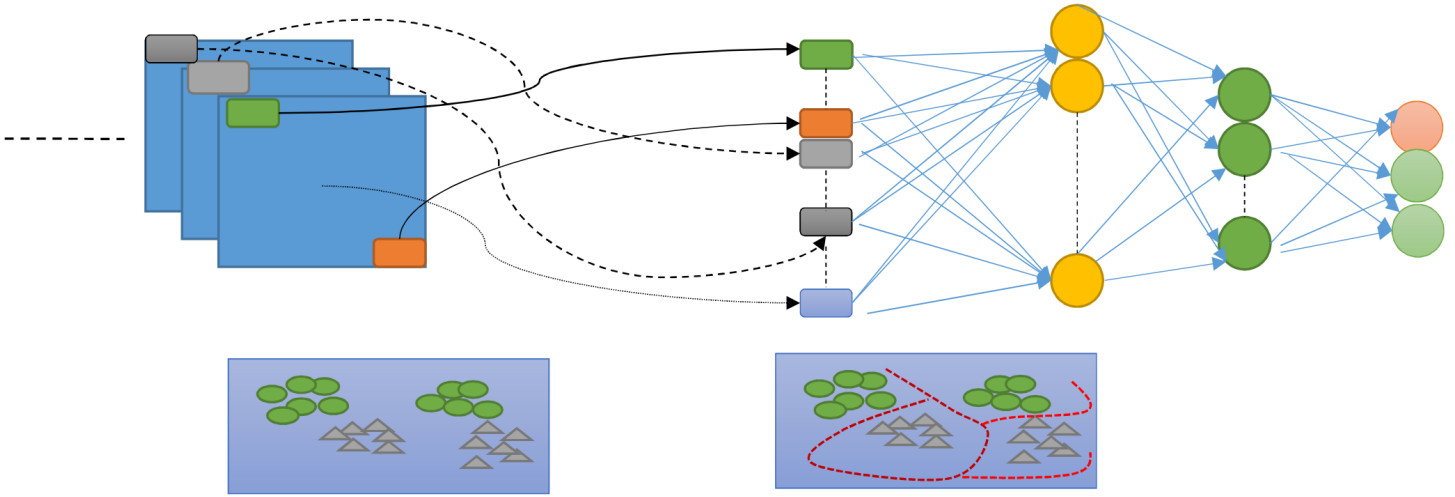
- Each convolution Layer (in a certain kernel size) remove infrequent sub-patterns (Disturbances) and extract frequent sub-patterns (Features).
- At each convolution Layer several feature maps may be generated . Each feature map extract an independent frequent sub-pattern.
- Each Pooling Layer reduces the dimension of the feature maps by sub-sampling. After each pooling operation, all features points which make a common frequent sub-pattern (from different input patterns) come nearer together: their scales will be more similar together . Also, one can say that after applying a pooling layer, the extracted features will be more invariant to size, position and perspective of the input pattern.
- Repeating the Convolution and Pooling layers allows (1) to filter disturbances from lower to upper size and concurrently (2) to extract each feature with a certain scale.
- Sub-Patterns which appear in the first convolution layer are the main edges of the inserted pattern; In next layers, sub-patterns become like corners and smaller subskeletons and finally in the last convolution layer they form the main sub-skeletons (effective features)

Fully Connected Layers

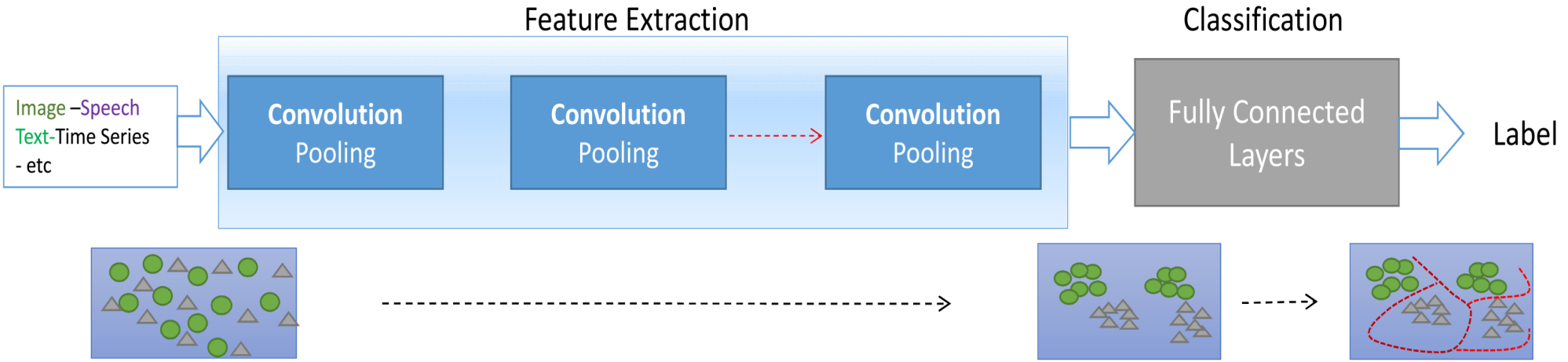
1. Units of last (Convolution Pooling) layer are arranged in a line :

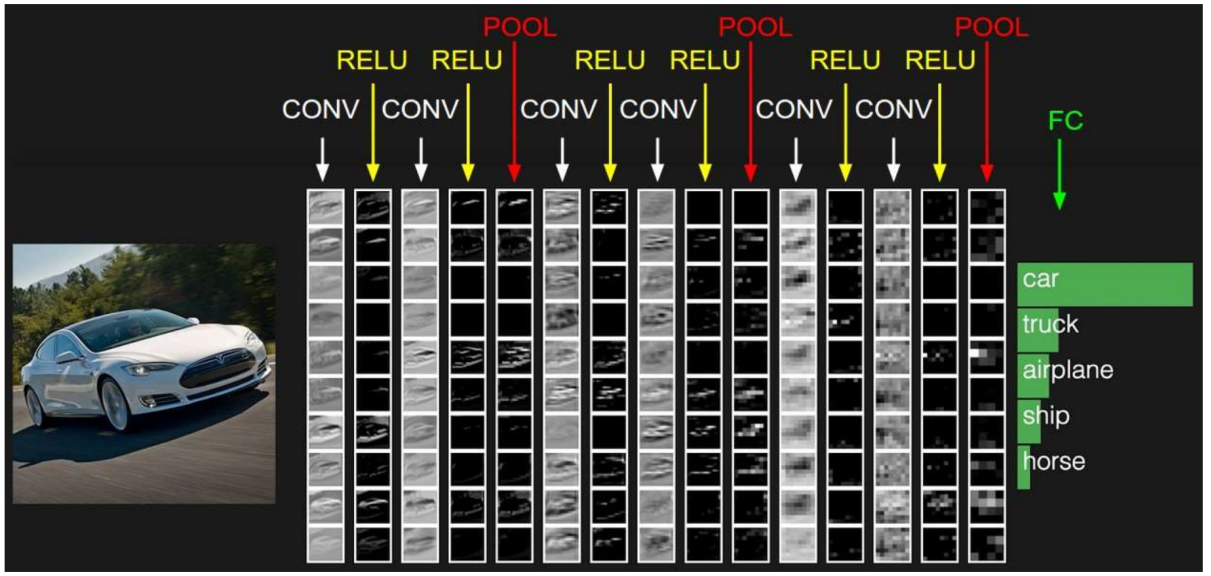


2. Then, one or two fully connected layers are defined and Finally, the output visible layer of CNN are defined.



CNN Architecture





* CNNs can be utilized for regression applications, too. It is enough to (1) extract features in conv./pooling layers and (2) use fully connected layers for mapping purpose.

Backpropagation in CNN

Forward pass

As you observed the forward pass of the convolutional layer can be expressed as

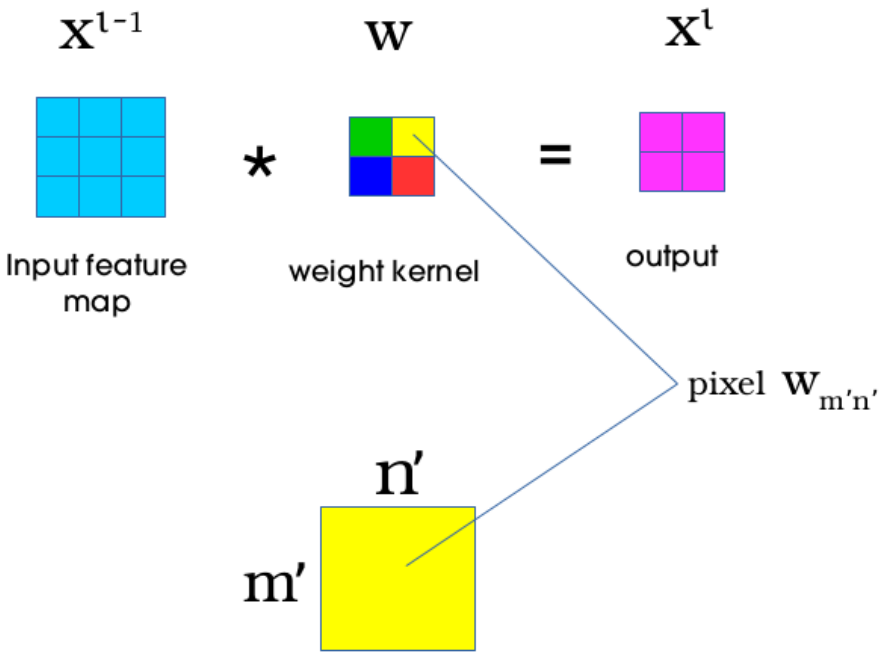
$$x_{i,j}^{l} = \sum_m \sum_n w_{m,n}^{l} o_{i+m,j+n}^{l-1} + b_{i,j}^{l}$$

Backpropagation

Assuming you are using the mean squared error (MSE) defined as

$$E = \frac{1}{2} \sum_p (t_p - y_p)^2,$$

Backpropagation in Conv layers



$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \frac{\partial E}{\partial x^l_{i,j}} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}}$$

$$= \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \delta^l_{i,j} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}}$$

$x^l_{i,j}$ is equivalent to $\sum_m \sum_n w^l_{m,n} o^{l-1}_{i+m,j+n} + b^l$

$$\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \frac{\partial}{\partial w^l_{m',n'}} \left(\sum_m \sum_n w^l_{m,n} o^{l-1}_{i+m,j+n} + b^l \right)$$

Convolution between the input feature map of dimension H×W and the weight kernel of dimension k1×k2 produces an output feature map of size (H−k1+1) by (W−k2+1).

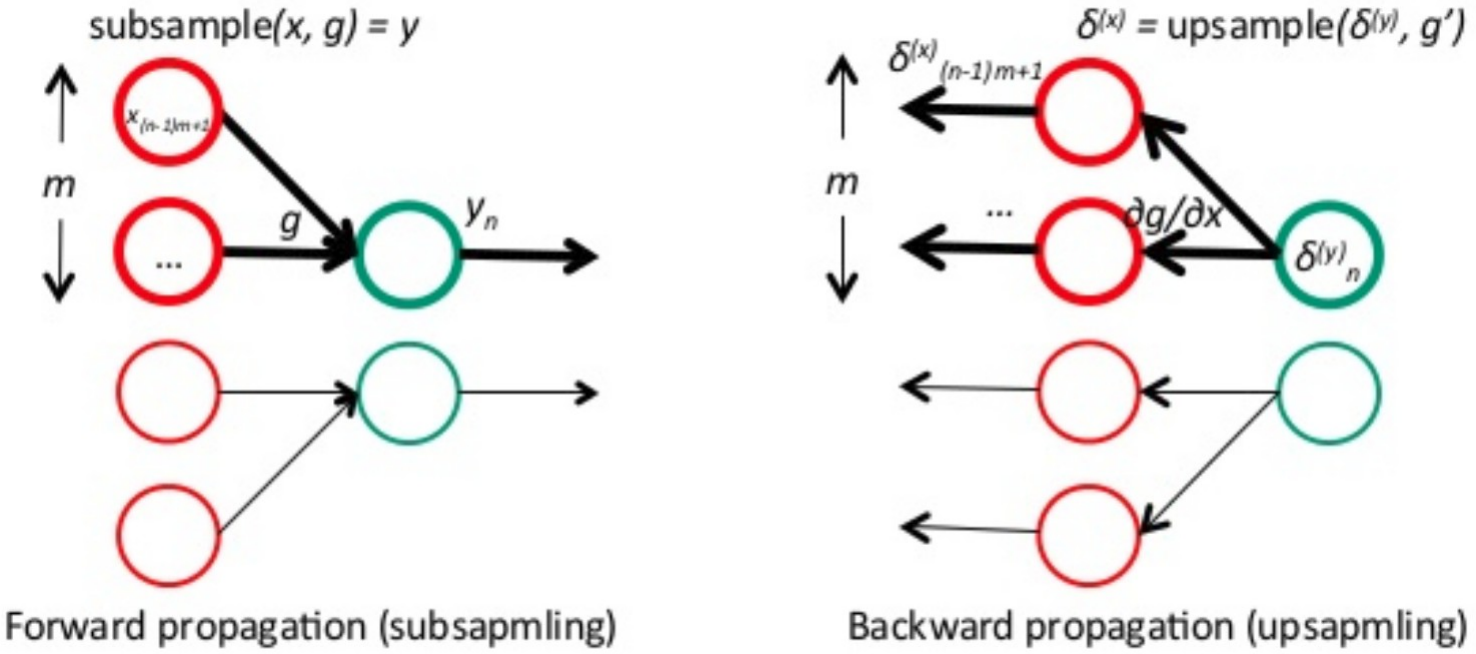
Backpropagation in Conv layers

$$
\begin{aligned}
\frac{\partial x_{i,j}^{l}}{\partial w_{m',n'}^{l}} &= \frac{\partial}{\partial w_{m',n'}^{l}} \left(w_{0,0}^{l} o_{i+0,j+0}^{l-1} + \cdots + w_{m',n'}^{l} o_{i+m',j+n'}^{l-1} + \cdots + b^{l} \right) \\
&= \frac{\partial}{\partial w_{m',n'}^{l}} \left(w_{m',n'}^{l} o_{i+m',j+n'}^{l-1} \right) \\
&= o_{i+m',j+n'}^{l-1}
\end{aligned}
$$

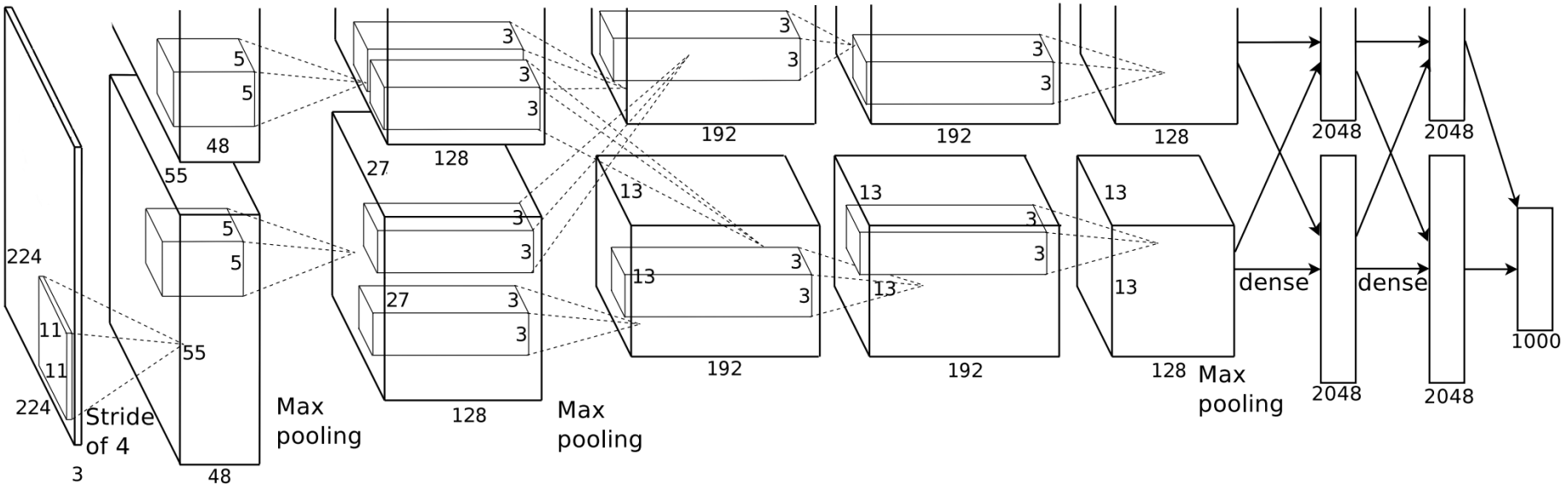
$$
\frac{\partial E}{\partial w_{m',n'}^{l}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^{l} o_{i+m',j+n'}^{l-1}
$$

Backpropagation in pooling layers

- Pooling neurons backpropagate the delta through upsampling mechanism
  - In max pooling, the unit which was the max in forward pass receives all the error at backward pass



AlexNet



- The network was made up of 5 conv layers, max-pooling layers, dropout layers, and 3 fully connected layers.
- The network they designed was trained on ImageNet dataset and was used for classification with 1000 possible categories.

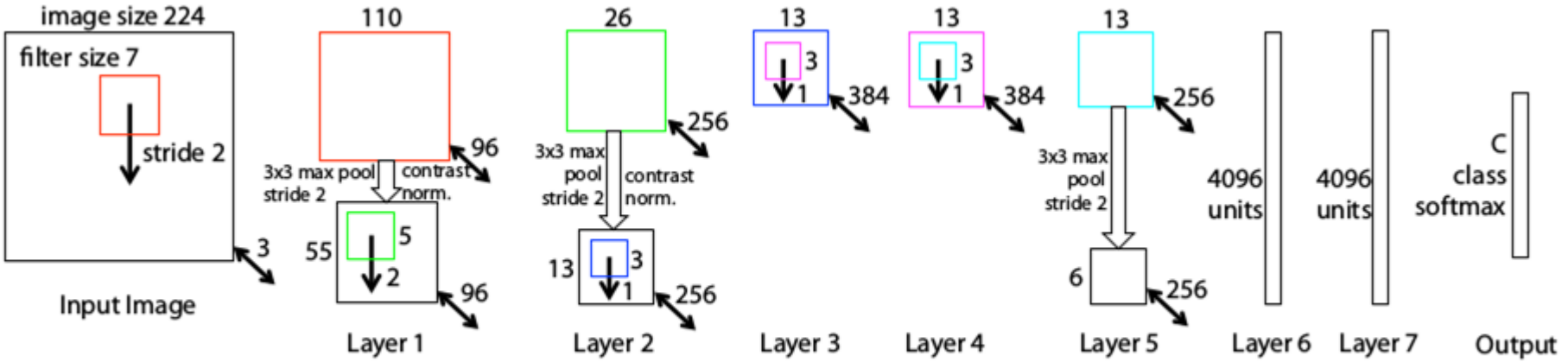
AlexNet

Main Points:

- Trained the network on ImageNet data, which contained over 15 million annotated images from a total of over 22,000 categories.
- Used ReLU for the nonlinearity functions (Found to decrease training time as ReLUs are several times faster than the conventional tanh function).
- Used data augmentation techniques that consisted of image translations, horizontal reflections, and patch extractions.
- Implemented dropout layers in order to combat the problem of overfitting to the training data.
- Trained the model using batch stochastic gradient descent, with specific values for momentum and weight decay.
- Trained on two GTX 580 GPUs for five to six days.
- Achieved a top 5 test error rate of 15.4%. The next best entry achieved an error of 26.2%, which was an astounding improvement that pretty much shocked the computer vision community.

ZF Net

- This model achieved an 11.2% error rate



ZF Net Architecture

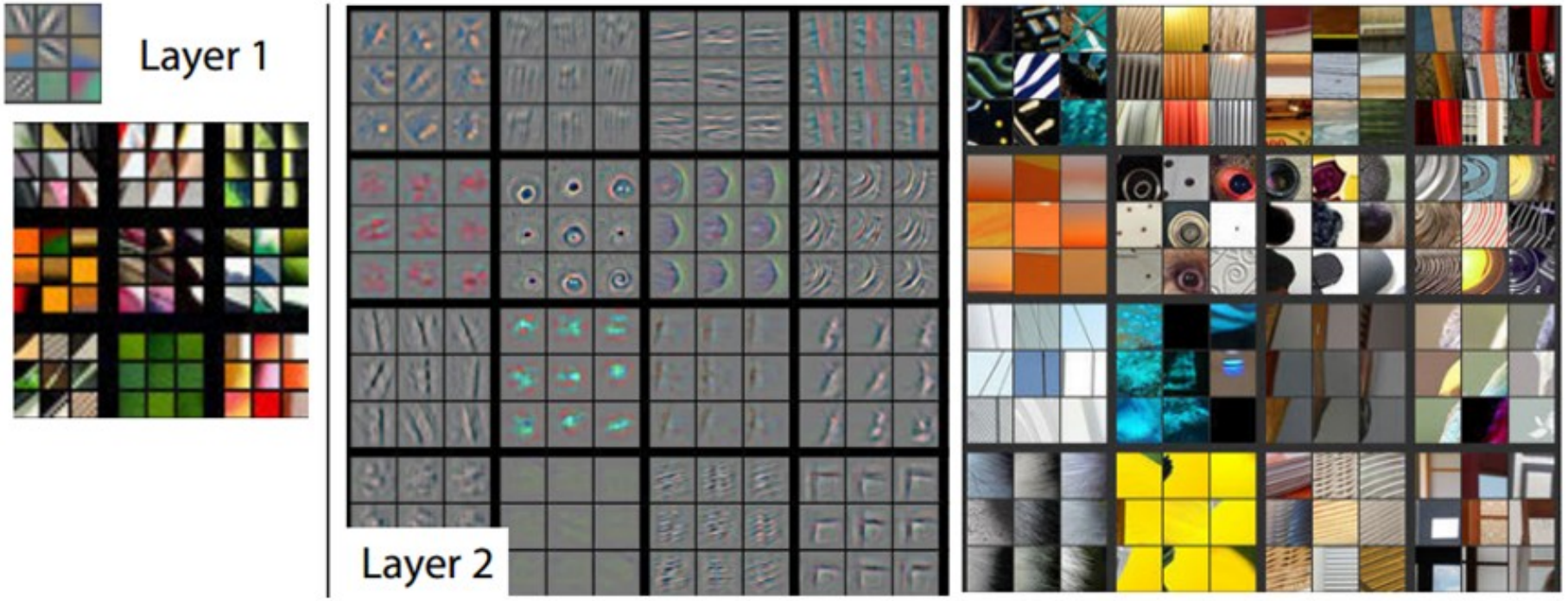
ZF Net

Main Points

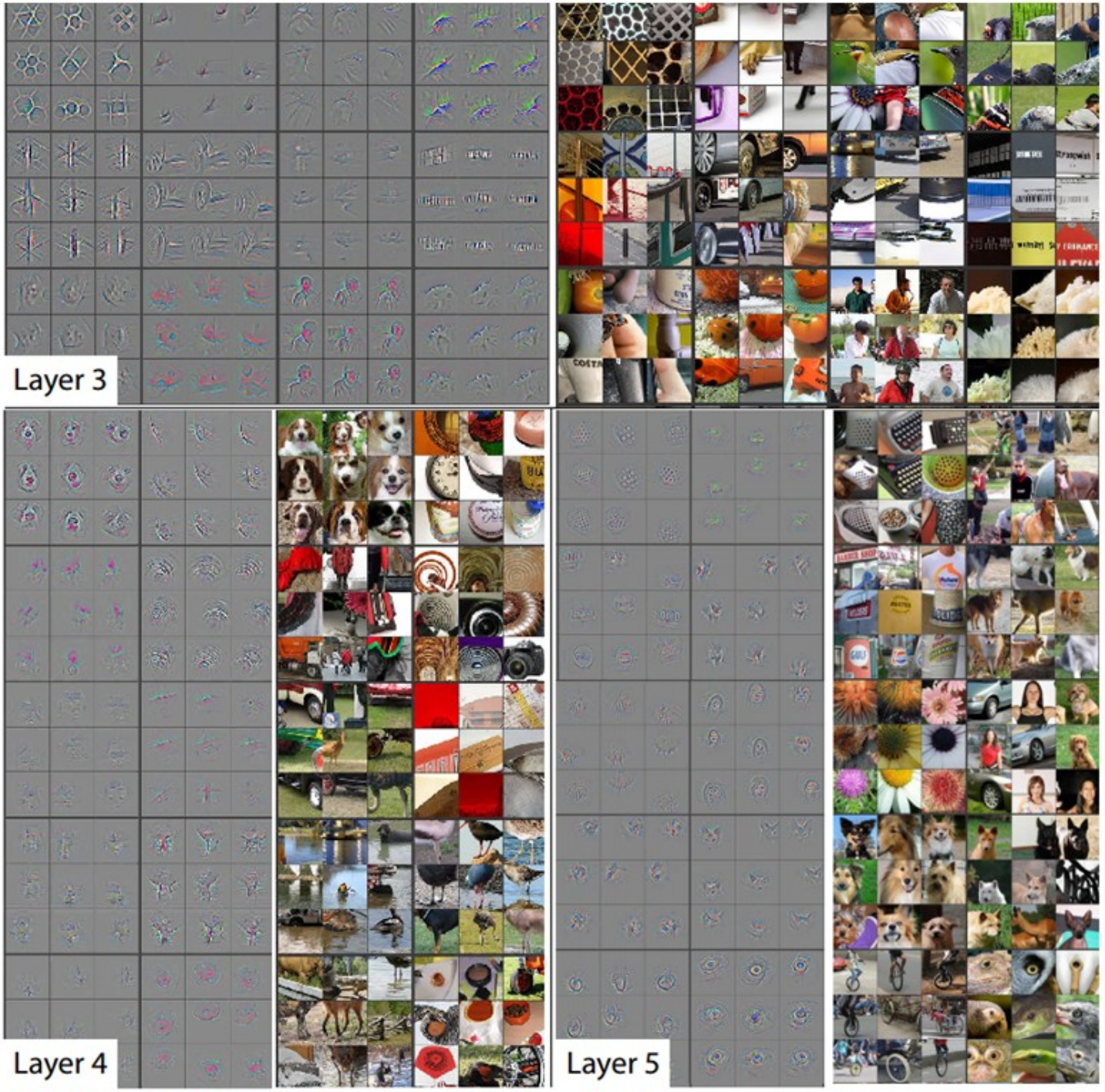
- Very similar architecture to AlexNet.
- AlexNet trained on 15 million images, while ZF Net trained on only 1.3 million images.
- Instead of using 11x11 sized filters in the first layer (same as AlexNet), ZF Net used filters of size 7x7 and a decreased stride value. The reasoning behind this modification is that a smaller filter size in the first conv layer helps retain a lot of original pixel information in the input volume. A filtering of size 11x11 proved to be skipping a lot of relevant information, especially as this is the first conv layer.
- As the network grows, we also see a rise in the number of filters used.
- Used ReLUs for their activation functions, cross-entropy loss for the error function, and trained using batch stochastic gradient descent.
- Trained on a GTX 580 GPU for twelve days.
- Developed a visualization technique named Deconvolutional Network, which helps to examine different feature activations and their relation to the input space.

DeConvNet

- The basic idea behind how this works is that at every layer of the trained CNN, you attach a “deconvnet” which has a path back to the image pixels.



Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labled Layer 2, we have representations of the 16 different filters (on the left)



Visualizations of Layers 3, 4, and 5

VGG Net

That's what a model created in 2014 best utilized with its 7.3% error rate.

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 imag)					
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					

The 6 different architecures of VGG Net. Configuration D produced the best results

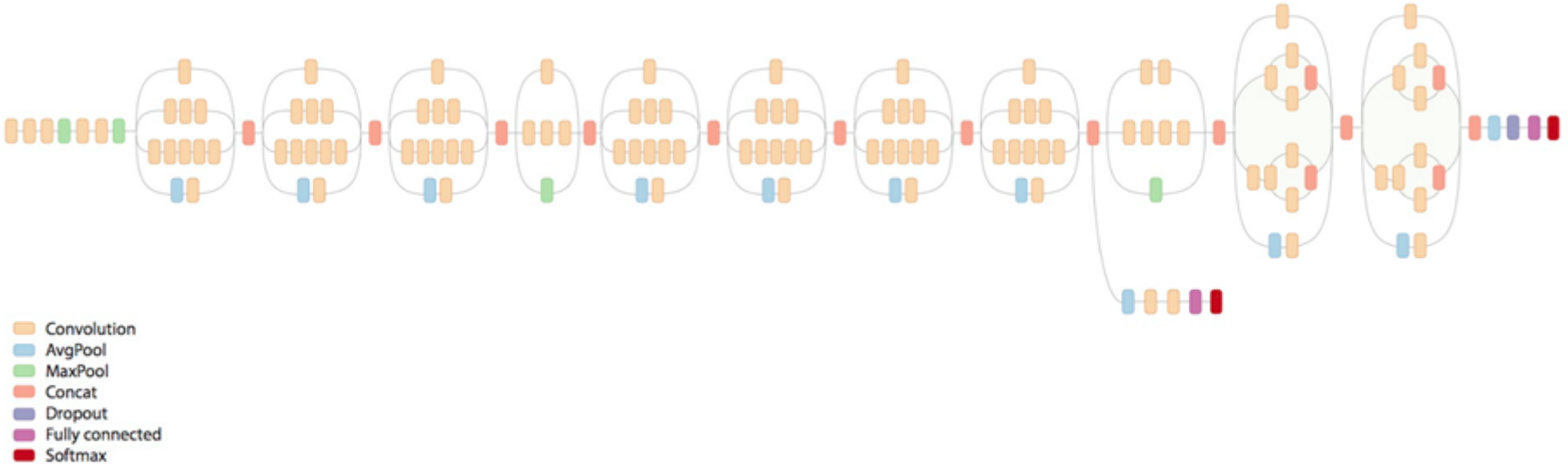
VGG

Main points:

- The use of only 3x3 sized filters is quite different from AlexNet's 11x11 filters in the first layer and ZF Net's 7x7 filters. The authors' reasoning is that the combination of two 3x3 conv layers has an effective receptive field of 5x5. This in turn simulates a larger filter while keeping the benefits of smaller filter sizes. One of the benefits is a decrease in the number of parameters. Also, with two conv layers, we're able to use two ReLU layers instead of one.
- 3 conv layers back to back have an effective receptive field of 7x7.
- As the spatial size of the input volumes at each layer decrease (result of the conv and pool layers), the depth of the volumes increase due to the increased number of filters as you go down the network.
- Interesting to notice that the number of filters doubles after each maxpool layer. This reinforces the idea of shrinking spatial dimensions, but growing depth.
- Worked well on both image classification and localization tasks. The authors used a form of localization as regression (see page 10 of the paper for all details).
- Used scale jittering as one data augmentation technique during training.

GoogleNet

- GoogLeNet is a 22 layer CNN and was the winner of ILSVRC 2014 with a top 5 error rate of 6.7%.

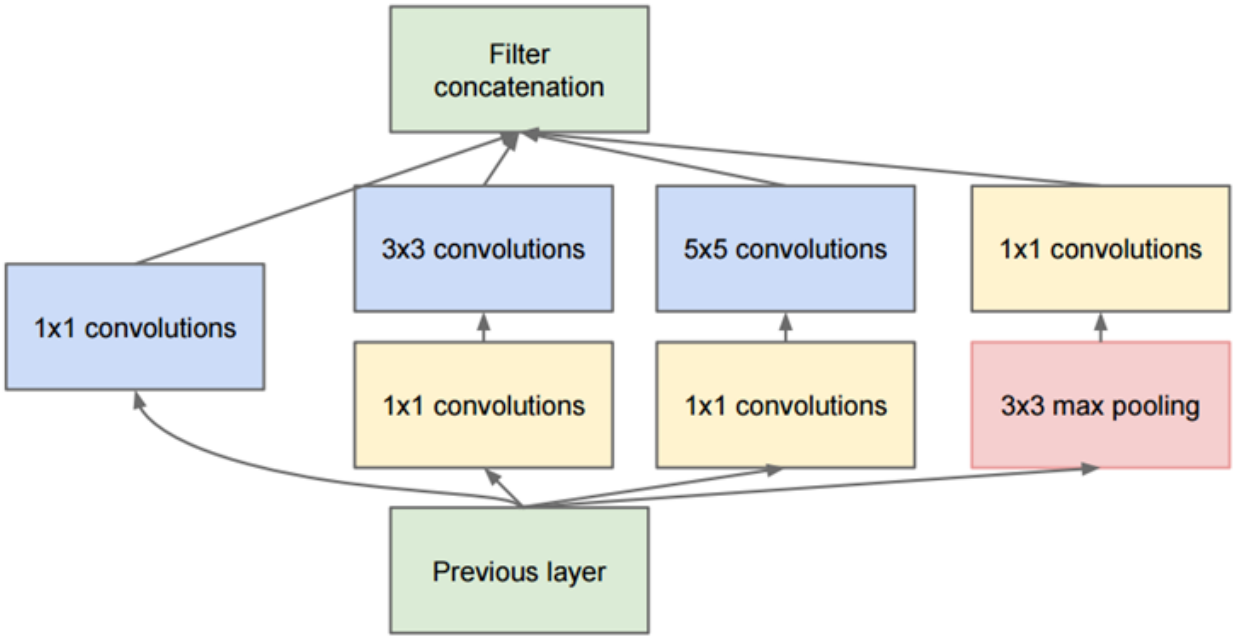


Another view of GoogLeNet's architecture.

- Each box is called an Inception module.

GoogleNet

- The network in network conv is able to extract information about the very fine grain details in the volume, while the 5x5 filter is able to cover a large receptive field of the input, and thus able to extract its information as well.
- The 1x1 convolutions (or network in network layer) provide a method of dimensionality reduction. For example, let's say you had an input volume of 100x100x60 (This isn't necessarily the dimensions of the image, just the input to any layer of the network).



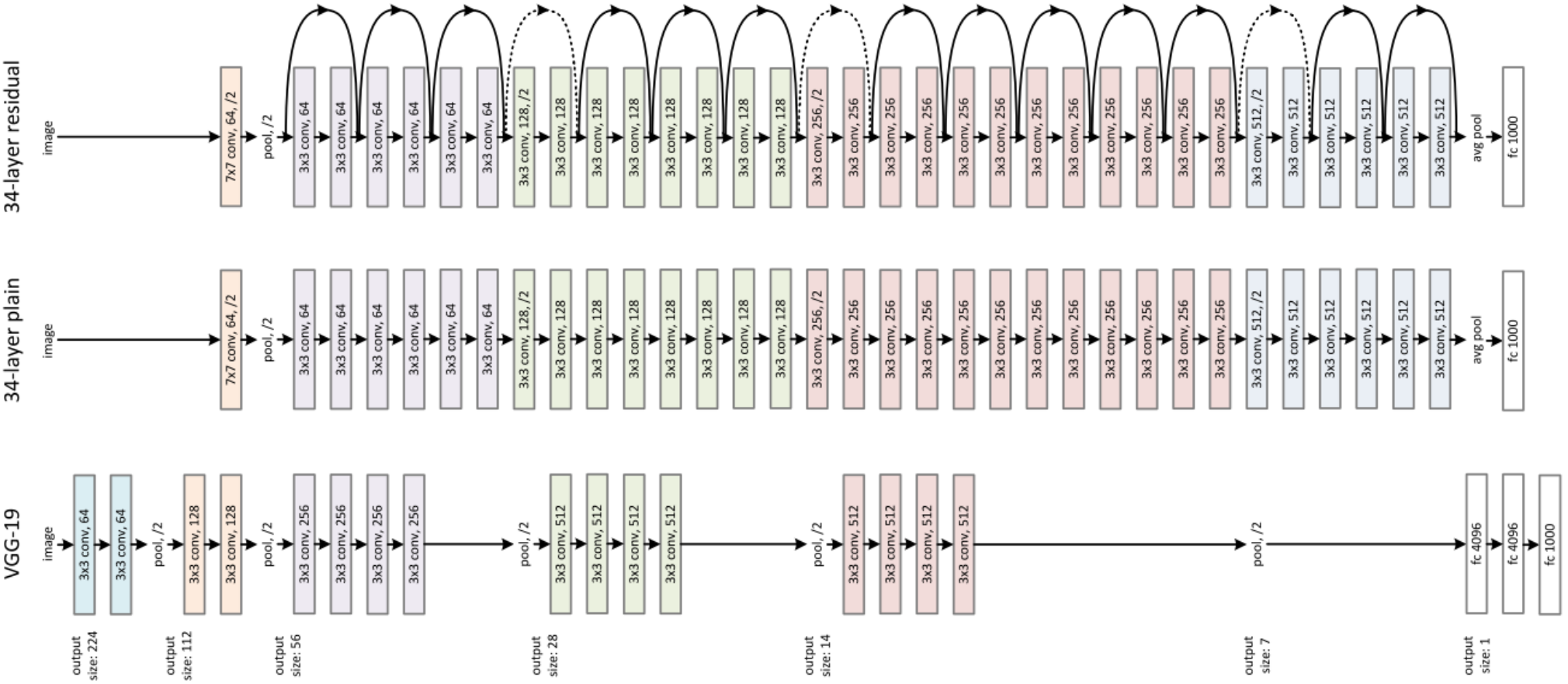
Full Inception module

GoogleNet

Main Points

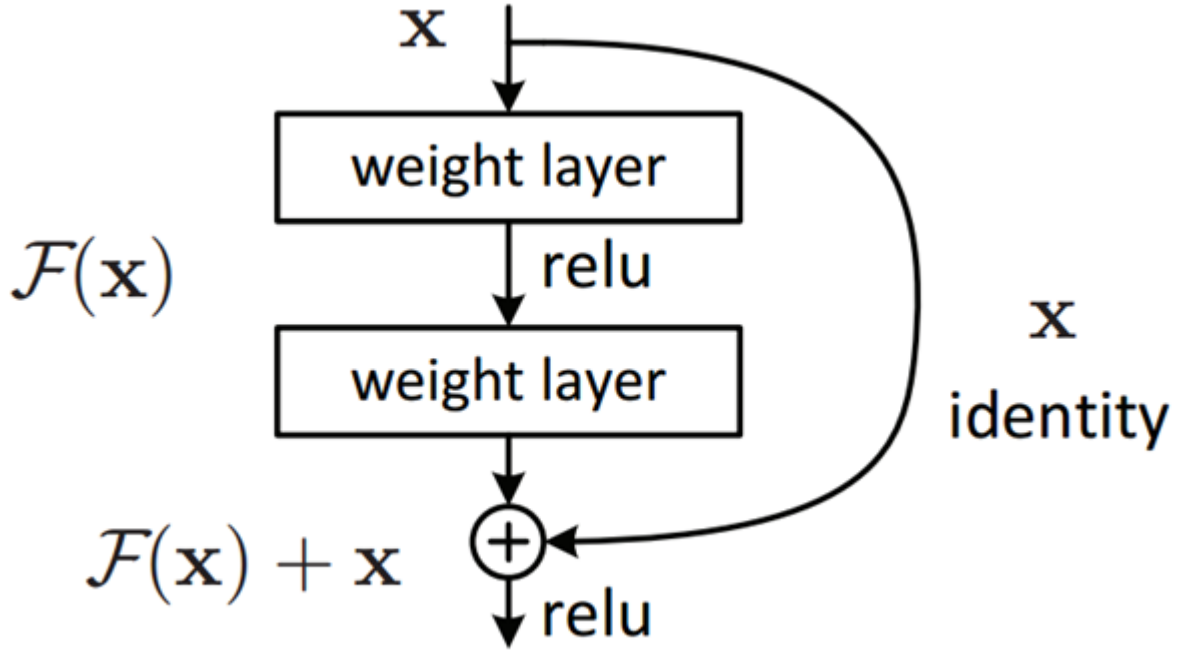
- Used 9 Inception modules in the whole architecture, with over 100 layers in total! Now that is deep…
- No use of fully connected layers! They use an average pool instead, to go from a 7x7x1024 volume to a 1x1x1024 volume. This saves a huge number of parameters.
- Uses 12x fewer parameters than AlexNet.
- During testing, multiple crops of the same image were created, fed into the network, and the softmax probabilities were averaged to give us the final solution.
- Utilized concepts from R-CNN (a paper we'll discuss later) for their detection model.
- There are updated versions to the Inception module (Versions 6 and 7).
- Trained on "a few high-end GPUs within a week".

ResNet (Residual Network)



Residual Block

- So, instead of just computing the transformation (straight from x to F(x)), we're computing the term that you have to add, F(x), to your input, x.
- Basically, the mini module shown below is computing a "delta" or a slight change to the original input x to get a slightly altered representation



A residual block

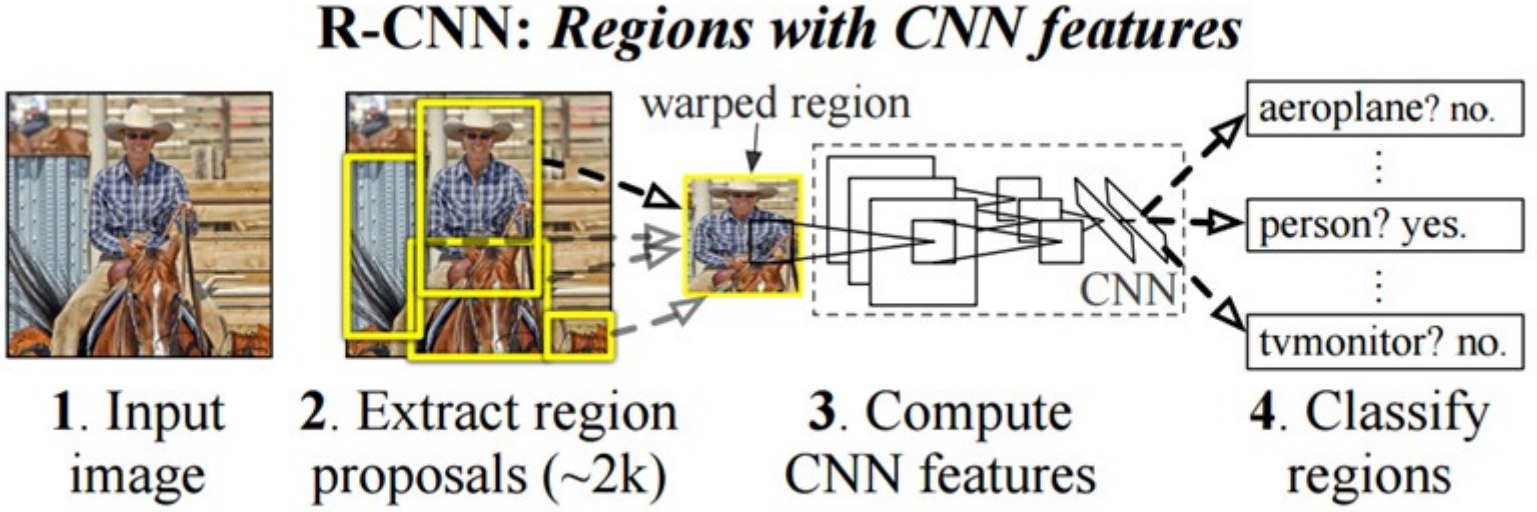
ResNet

Main Points

- “Ultra-deep” – Yann LeCun.
- 152 layers…
- Interesting note that after only the first 2 layers, the spatial size gets compressed from an input volume of 224x224 to a 56x56 volume.
- Authors claim that a naïve increase of layers in plain nets result in higher training and test error.
- The group tried a 1202-layer network, but got a lower test accuracy, presumably due to overfitting.
- Trained on an 8 GPU machine for two to three weeks.
- 3.6% error rate on ImageNet

Region Based CNNs (R-CNN)

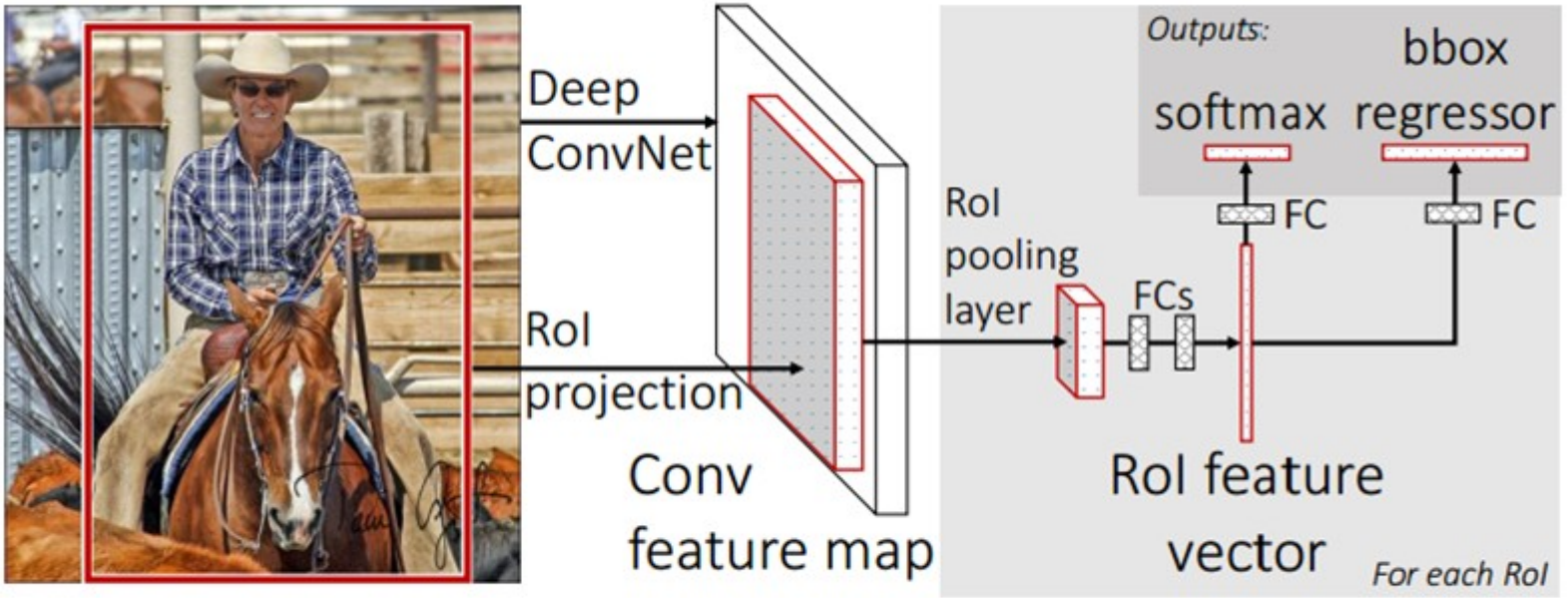
- The purpose of R-CNNs is to solve the problem of object detection. Given a certain image, we want to be able to draw bounding boxes over all of the objects. The process can be split into two general components, the region proposal (Selective Search) step and the classification step (Deep CNN).



R-CNN workflow

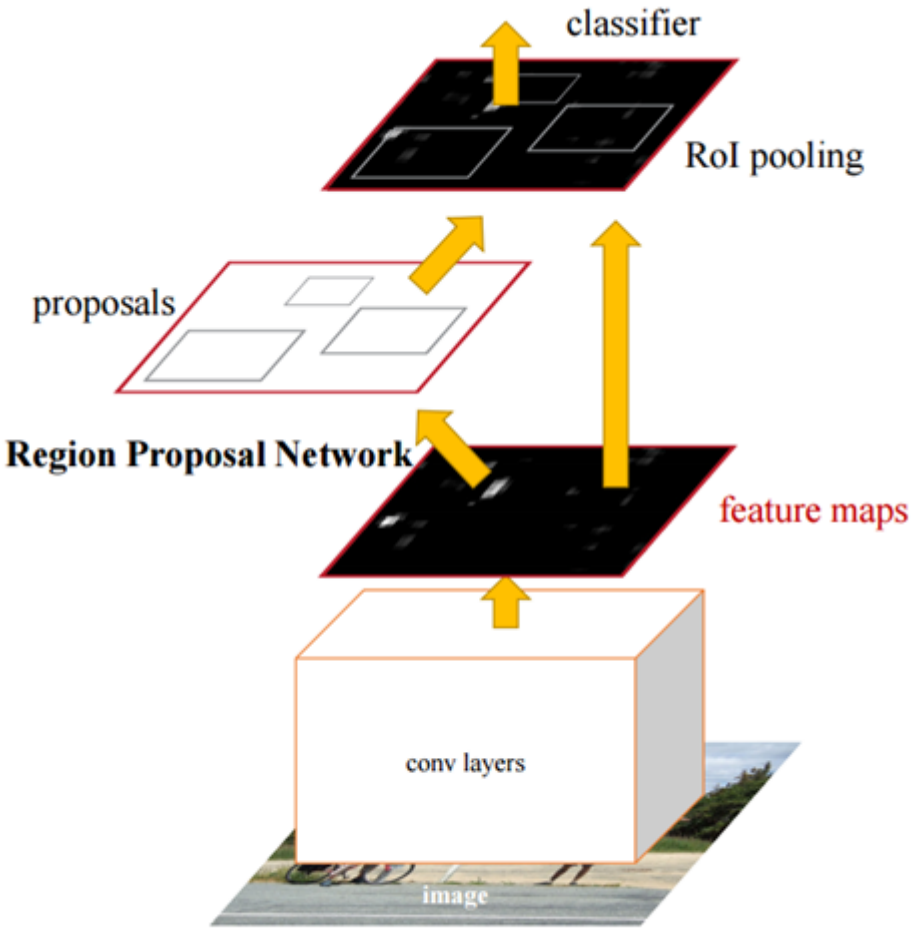
Selective Search

Fast R-CNN, Faster R-CNN



Fast R-CNN workflow

Faster R-CNN



Faster R-CNN workflow

Generative Adversarial Networks (GAN)