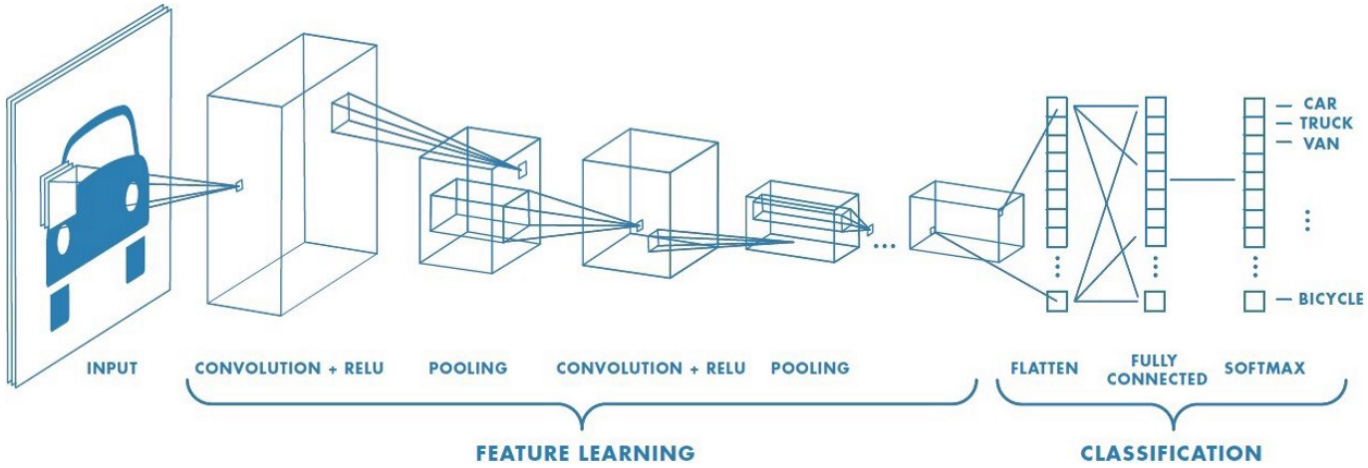
Deep Learning Architectures

Dr. Mehul Parikh
L. D. College of Engineering

Outline

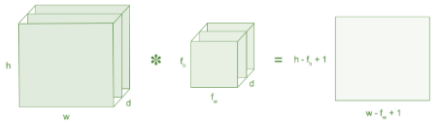
- Basics of CNN
- CNN Architectures
  - LeNet
  - AlexNet
  - ZFNet
  - VGGNet
  - GoogLeNet (Inception)
  - ResNet
  - DenseNet
- RNN
  - Vanilla RNN
  - LSTM
  - Bidirectional RNN
  - GRU

CNN - Basics



- Convolution layer
- Rectified Linear Unit (ReLU) layer
- Pooling layer
- Flatten layer
- Fully Connected (FC) layer
- Output layer

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**



Convolution Layer

2	4	9	1	4
2	1	4	4	6
1	1	2	9	2
7	3	5	1	3
2	3	4	8	5

X

1	2	3
-4	7	4
2	-5	1

Filter / Kernel

=

51		

Feature

2*1 + 4*2 + 9*3 + 2*(-4) + 1*7 + 4*4 + 1*2 + 1*(-5) + 2*1 = 51

2	4	9	1	4
2	1	4	4	6
1	1	2	9	2
7	3	5	1	3
2	3	4	8	5

Image

X

1	2	3
-4	7	4
2	-5	1

Filter / Kernel

=

51	66	

Feature

4*1 + 9*2 + 1*3 + 1*(-4) + 4*7 + 4*4 + 1*2 + 2*(-5) + 9*1 = 66

2	4	9	1	4
2	1	4	4	6
1	1	2	9	2
7	3	5	1	3
2	3	4	8	5

Image

X

1	2	3
-4	7	4
2	-5	1

Filter / Kernel

=

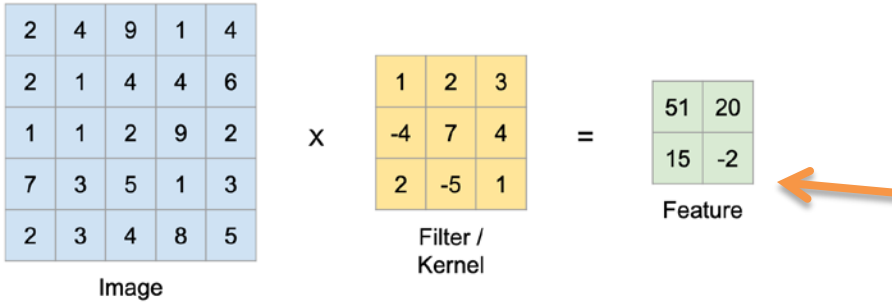
51	66	20
31	49	101
15	53	-2

Feature

2*1 + 9*2 + 2*3 + 5*(-4) + 1*7 + 3*4 + 4*2 + 8*(-5) + 5*1 = -2

Feature Size, Stride

Feature size = ((Image size − Kernel size) / Stride) + 1

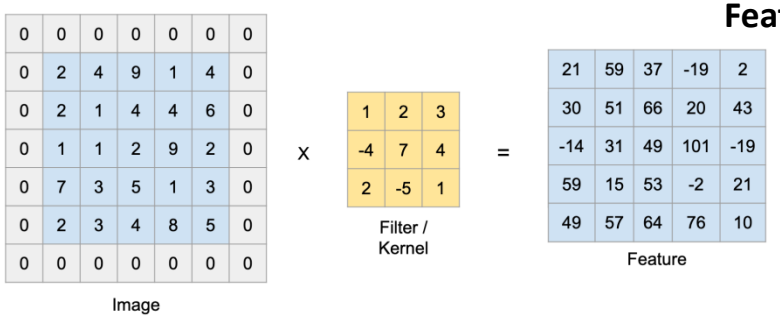


Feature size = ((5 − 3) / 1) + 1 = 3, Stride = 1

Feature size = ((5 − 3) / 2) + 1 = 2, Stride = 2

With Padding

Feature size = ((Image size + 2 * Padding size − Kernel size) / Stride)+1



Feature size = ((5 + 2 * 1 − 3) / 1) + 1= 5

Pooling Layer

21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

Convolved Feature

59			

Max Pooled Feature

21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

Convolved Feature

59	66		

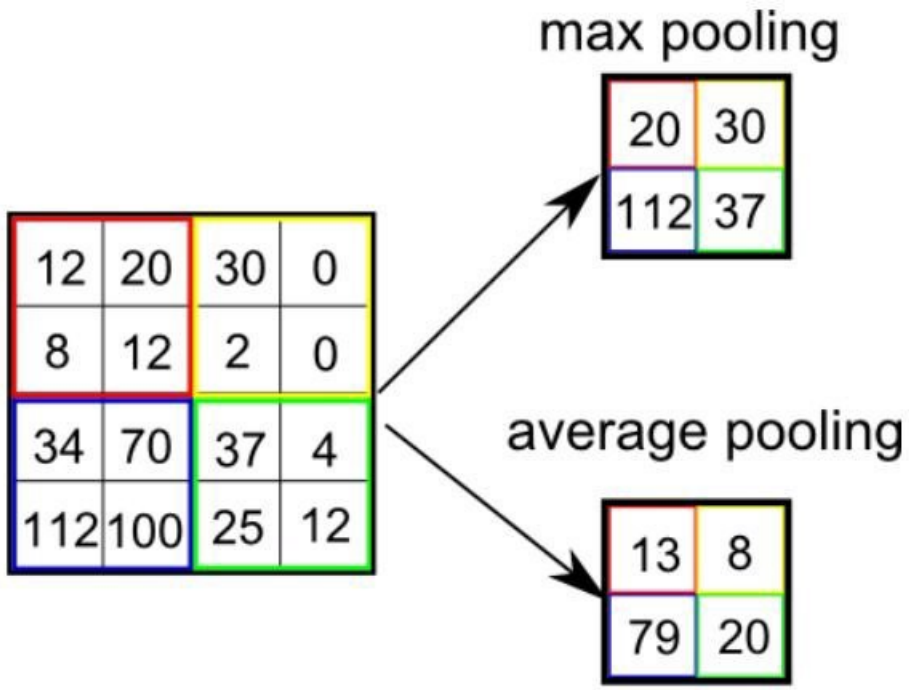
Max Pooled Feature

21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

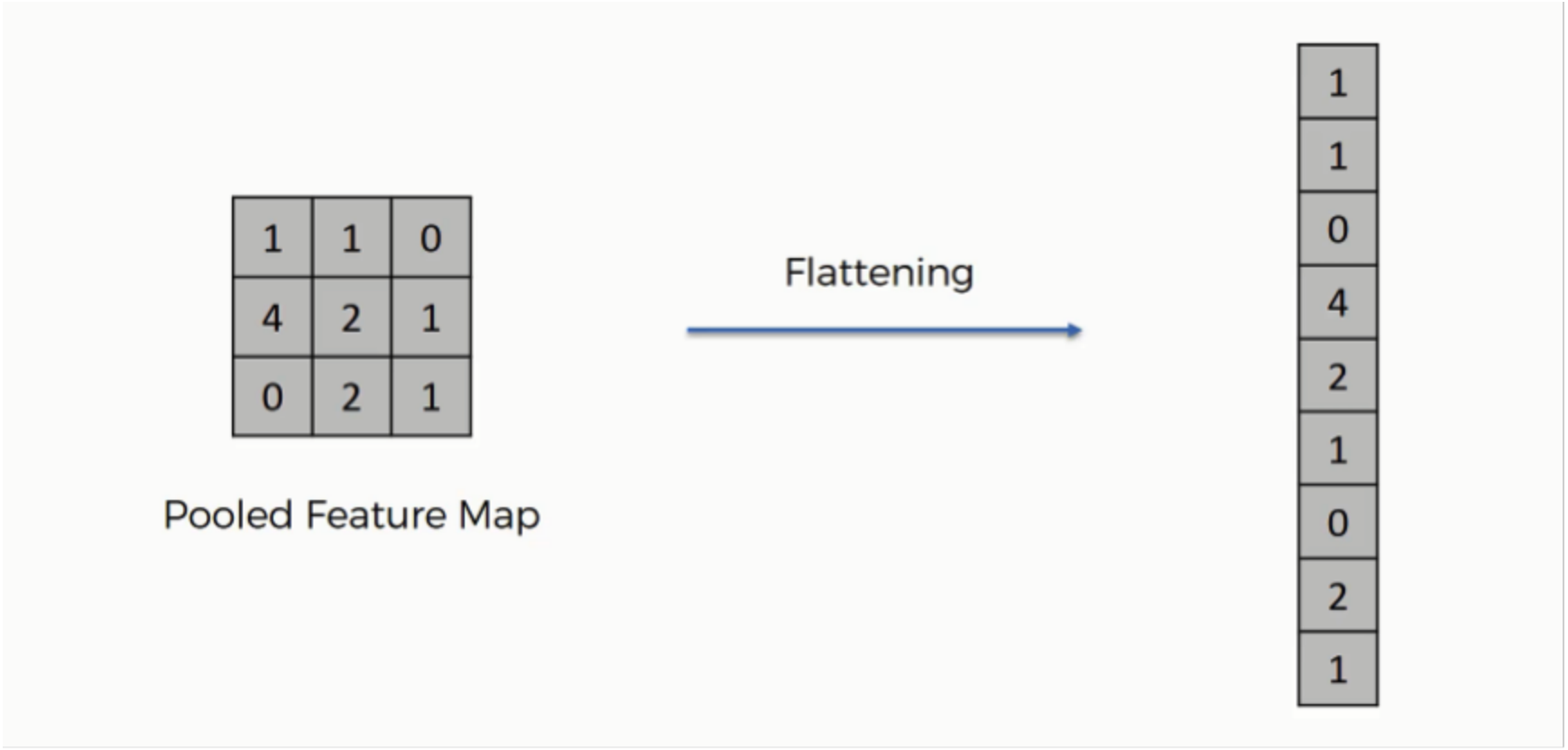
Convolved Feature

59	66	66	43
51	66	101	101
59	53	101	101
59	64	76	76

Max Pooled Feature



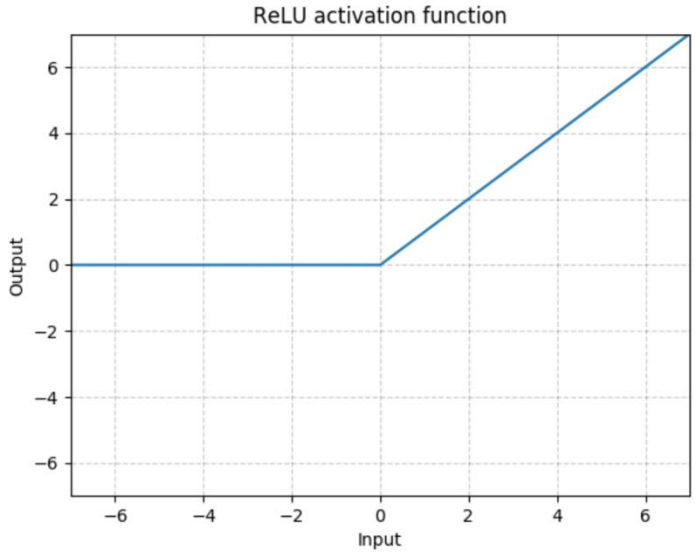
Flattening Layer

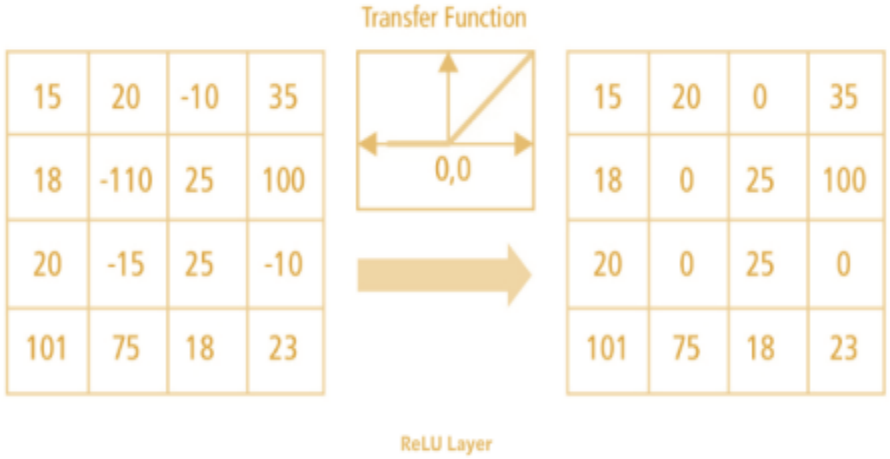


Activation Function

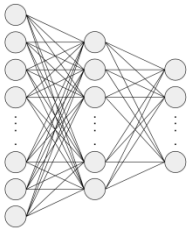
RelU (Rectified Linear Unit)

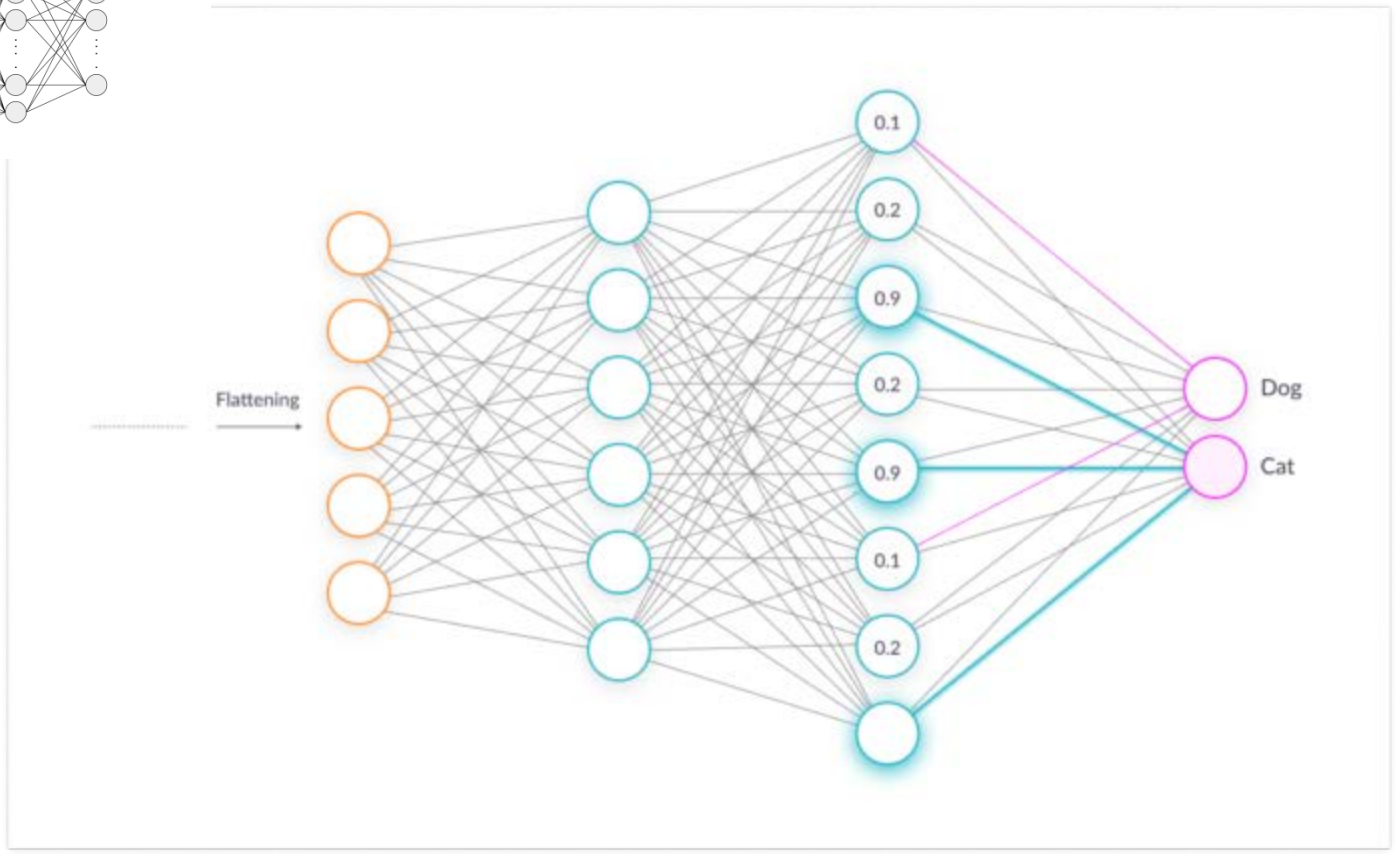
ReLU(x)=max(0,x)



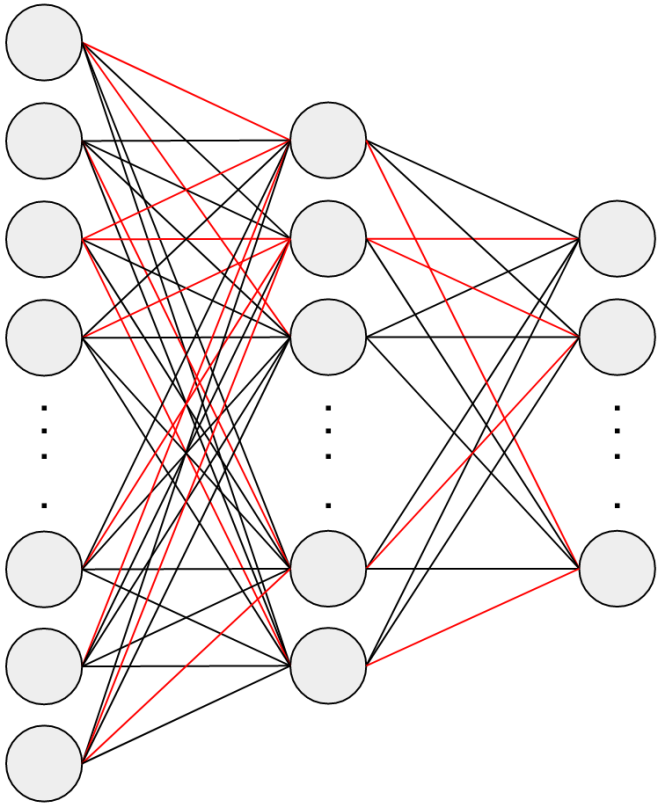


Fully Connected Layers

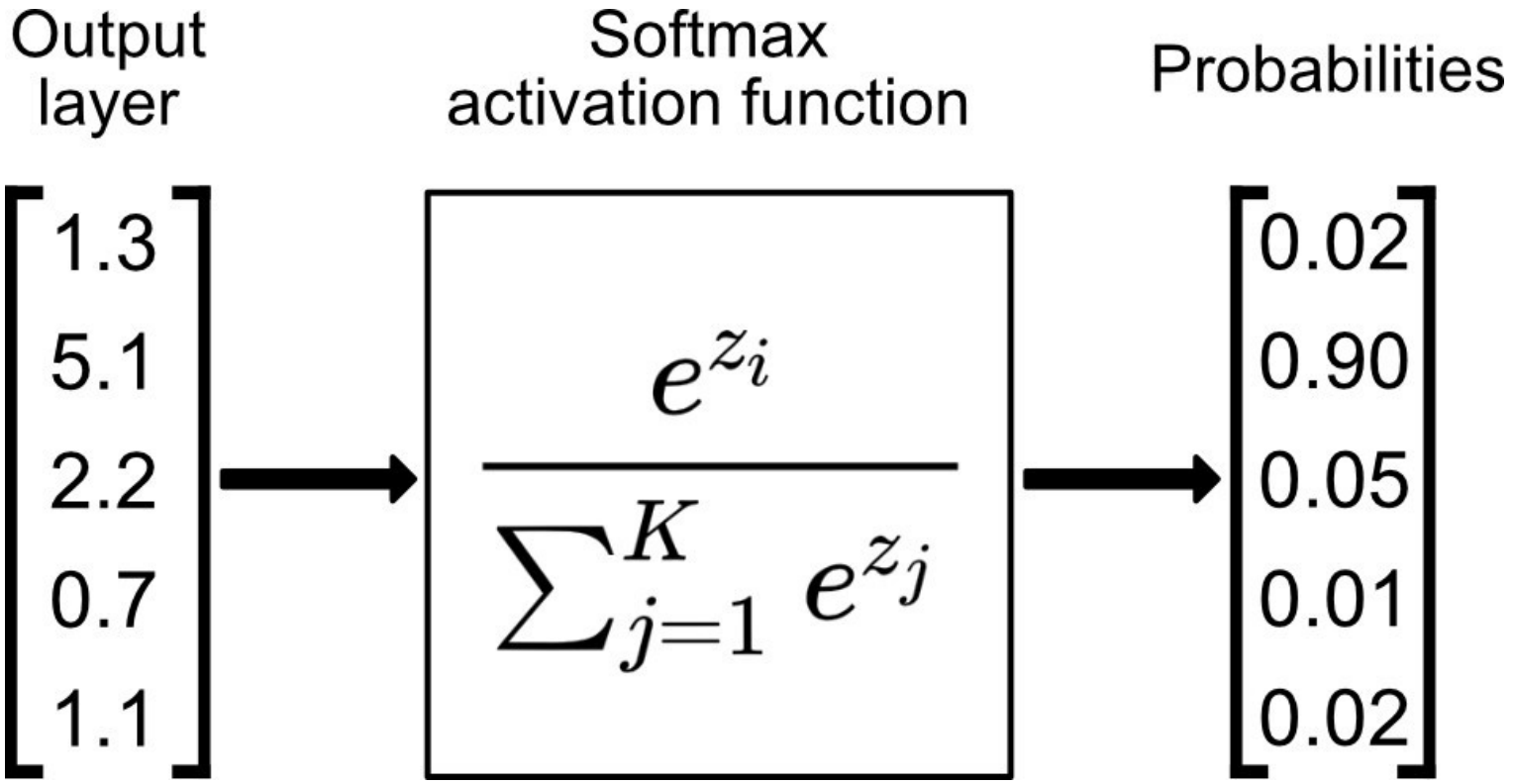




Drop Out Layers



Output Layer



Loss Function – Cross Entropy

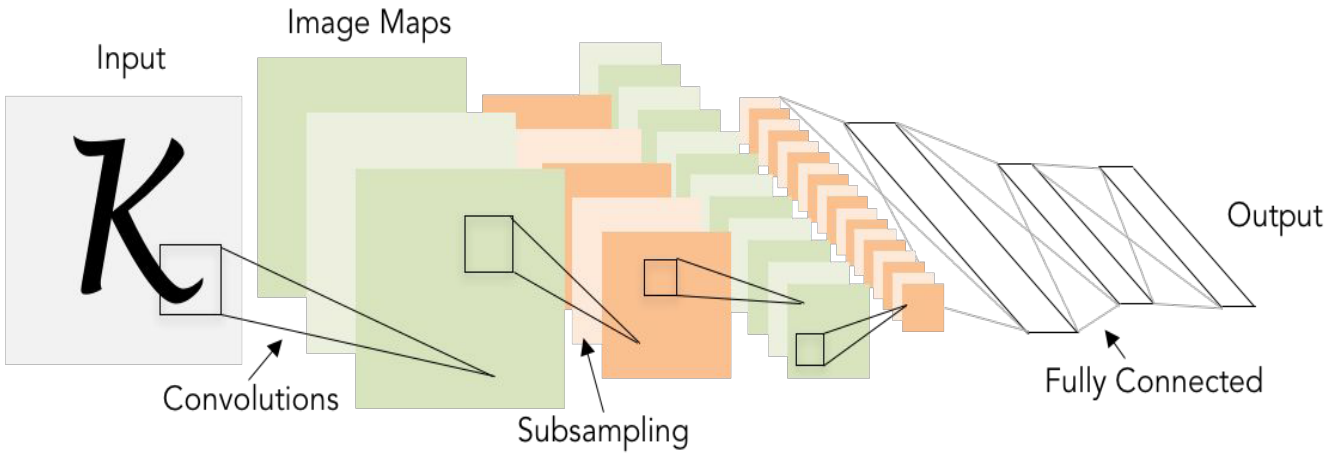
The loss can be described as:

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

CNN Architectures

- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogLeNet (Inception)
- ResNet
- DenseNet

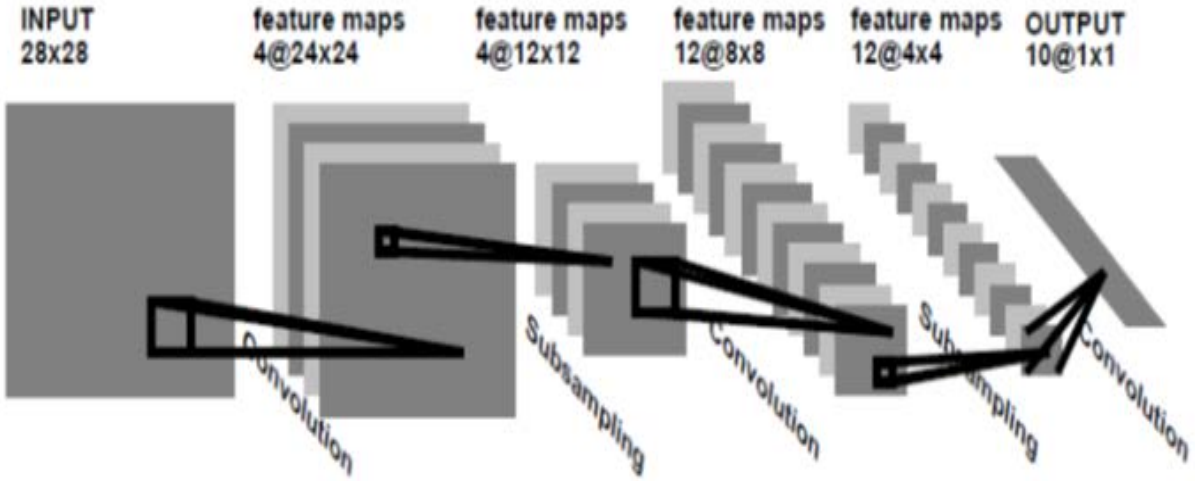
LeNet



Main Ideas

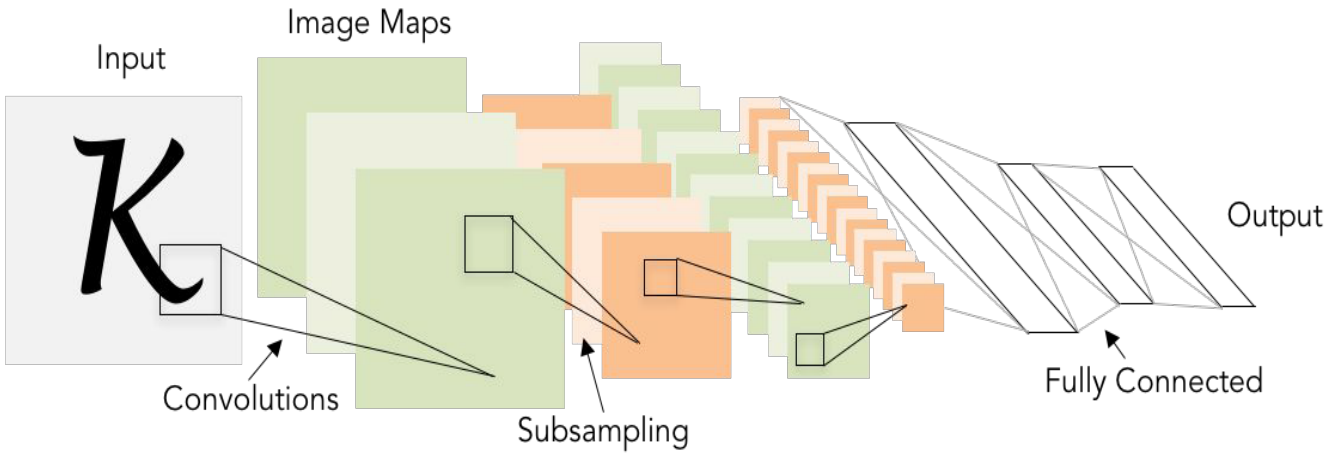
convolution, local receptive fields, shared weights, spacial subsampling

LeNet-1



- **28×28 input image >**
- **Four 24×24 feature maps convolutional layer (5×5 size) >**
- **Average Pooling layers (2×2 size) >**
- **Eight 12×12 feature maps convolutional layer (5×5 size) >**
- **Average Pooling layers (2×2 size) >**
- **Directly fully connected to the output**

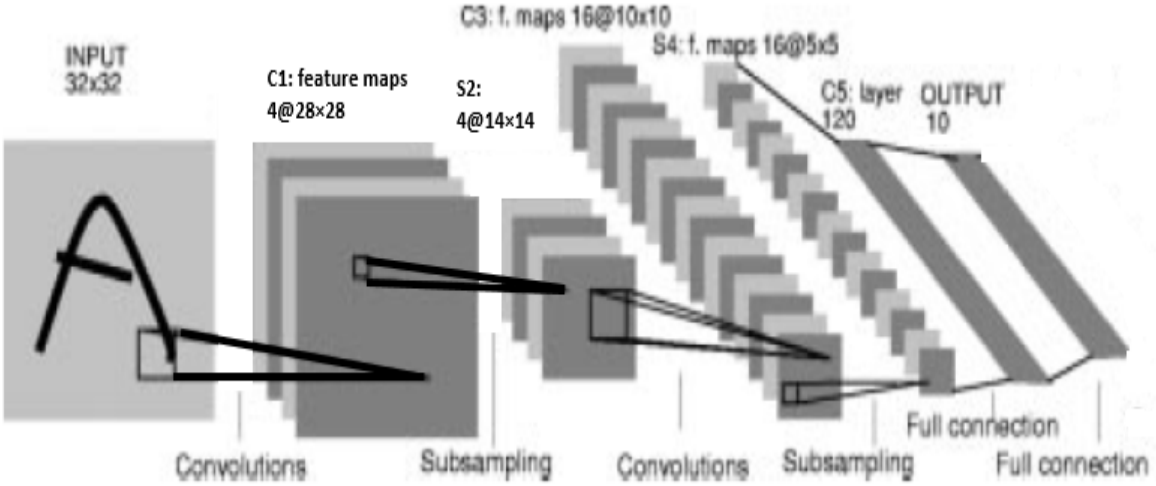
LeNet



Main Ideas

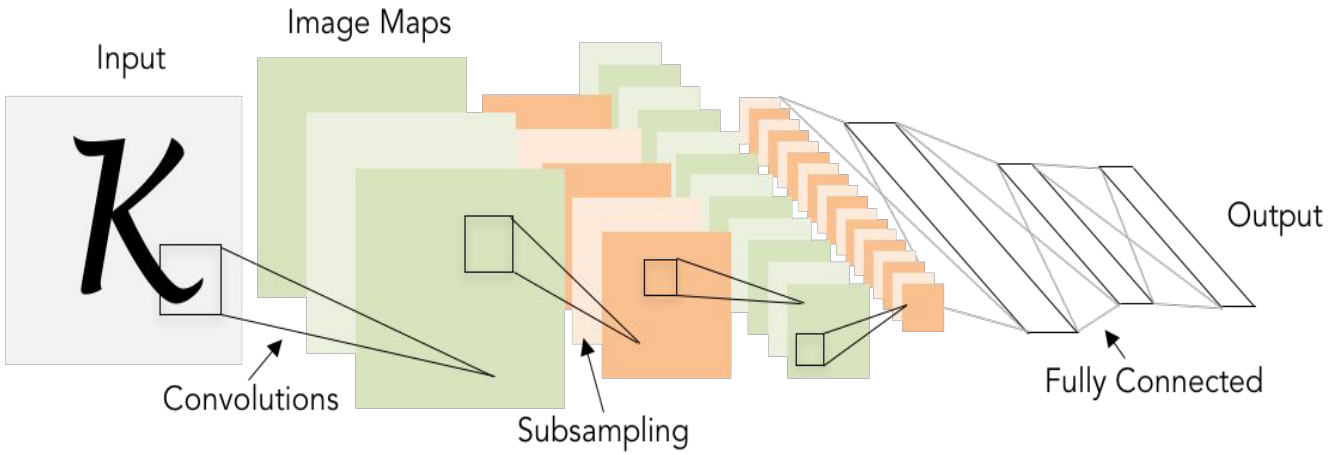
convolution, local receptive fields, shared weights, spacial subsampling

LeNet-4



- **32×32 input image >**
- **Four 28×28 feature maps convolutional layer (5×5 size) >**
- **Average Pooling layers (2×2 size) >**
- **Sixteen 10×10 feature maps convolutional layer (5×5 size) >**
- **Average Pooling layers (2×2 size) >**
- **Fully connected to 120 neurons >**
- **Fully connected to 10 outputs**

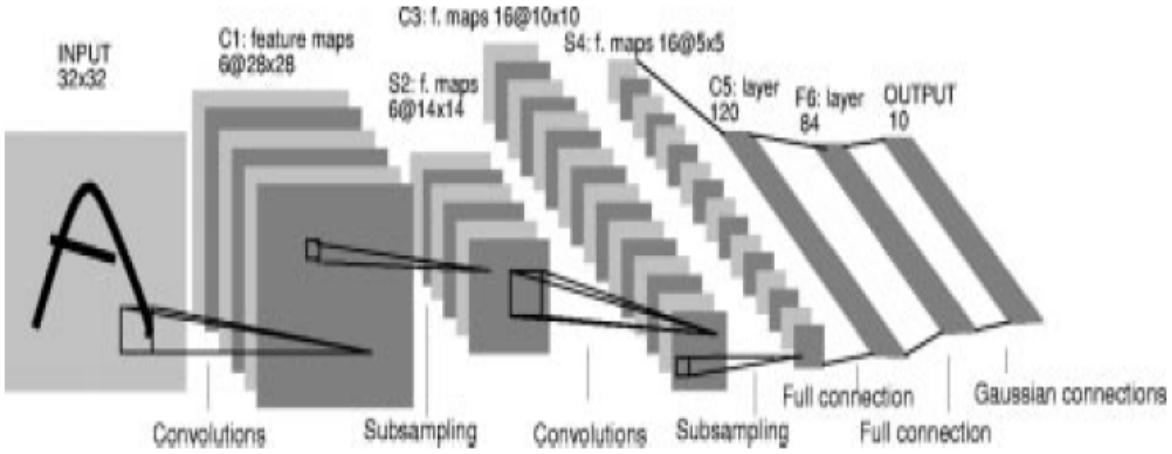
LeNet



Main Ideas

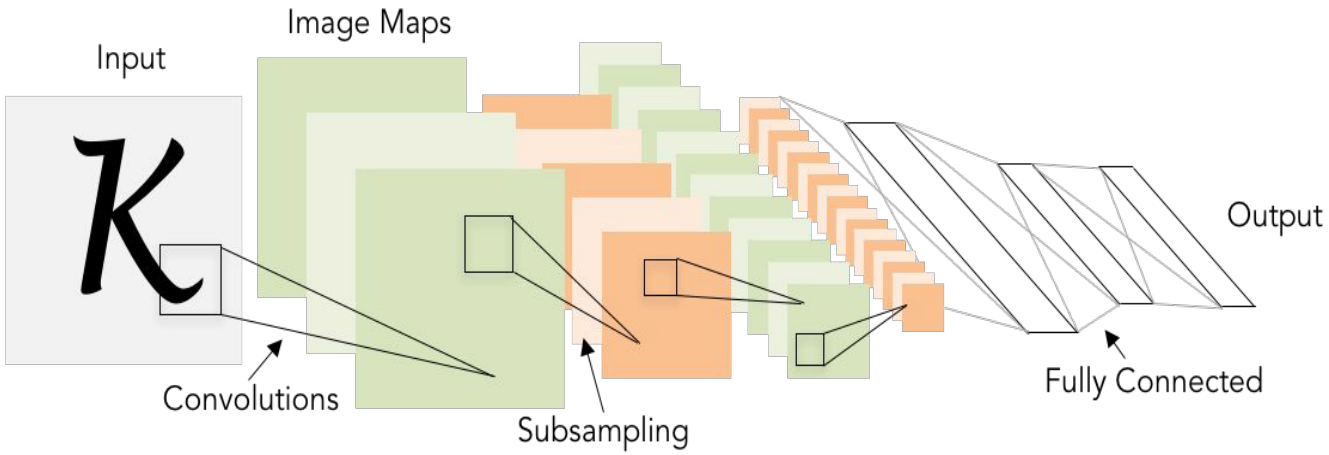
convolution, local receptive fields, shared weights, spacial subsampling

LeNet-5



- **32×32 input image >**
- **Six 28×28 feature maps convolutional layer (5×5 size) >**
- **Average Pooling layers (2×2 size) >**
- **Sixteen 10×10 feature maps convolutional layer (5×5 size) >**
- **Average Pooling layers (2×2 size) >**
- **Fully connected to 120 neurons >**
- **Fully connected to 84 neurons >**
- **Fully connected to 10 outputs**

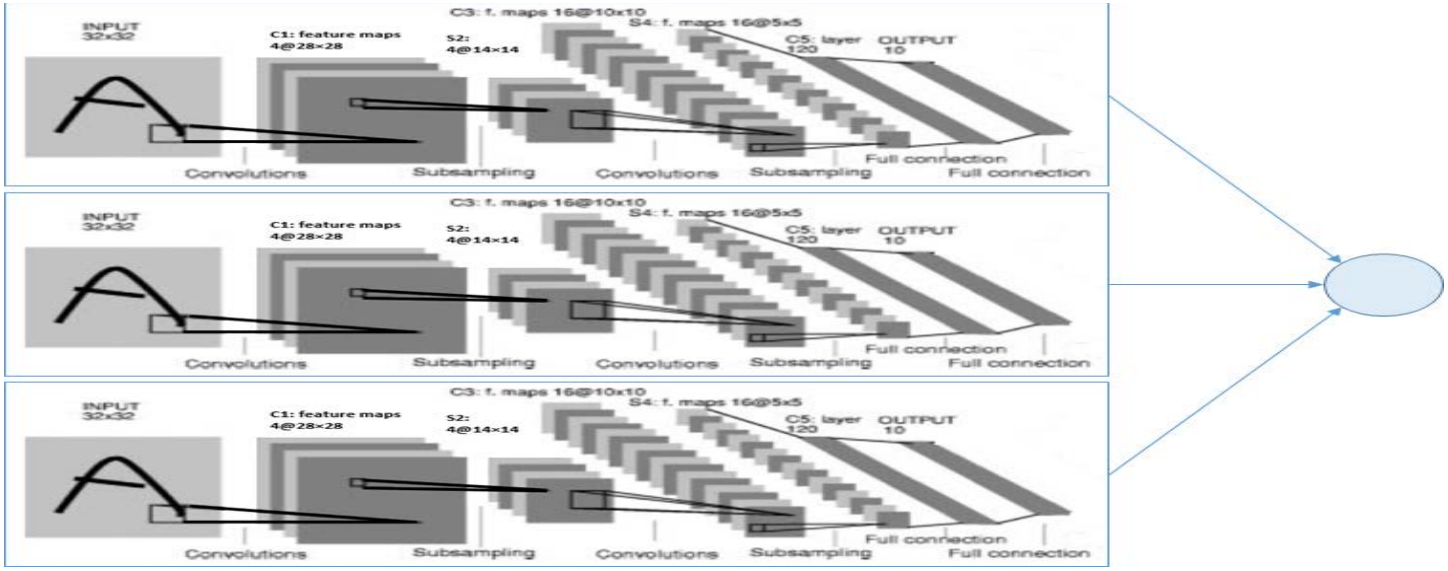
LeNet



Main Ideas

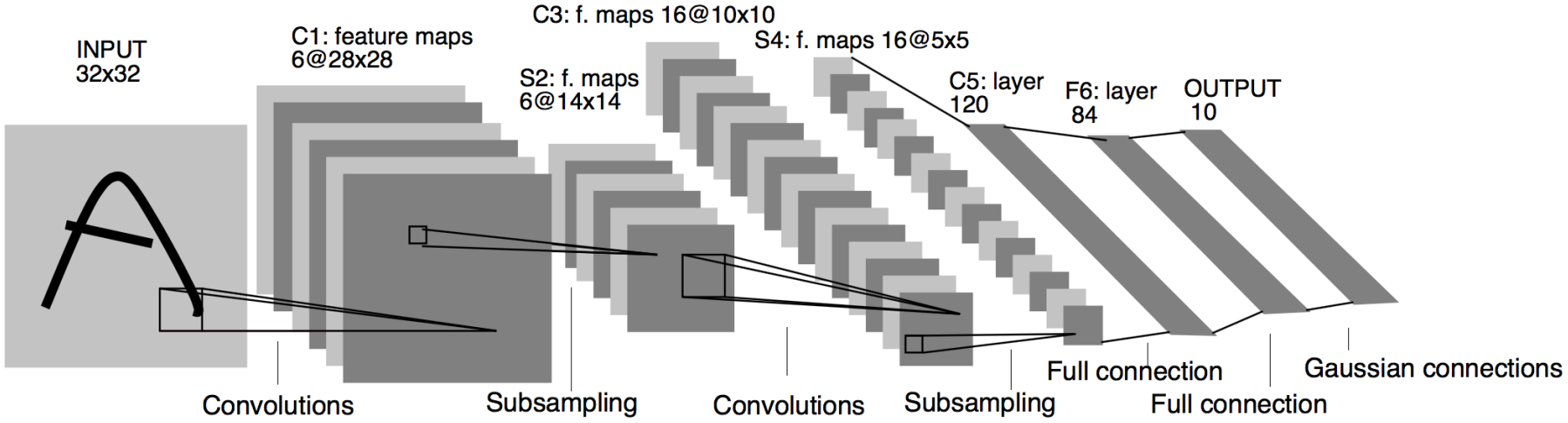
convolution, local receptive fields, shared weights, spacial subsampling

Boosted LeNet-4



combine the results from several weak classifiers to get a more accurate results

LeNet-5

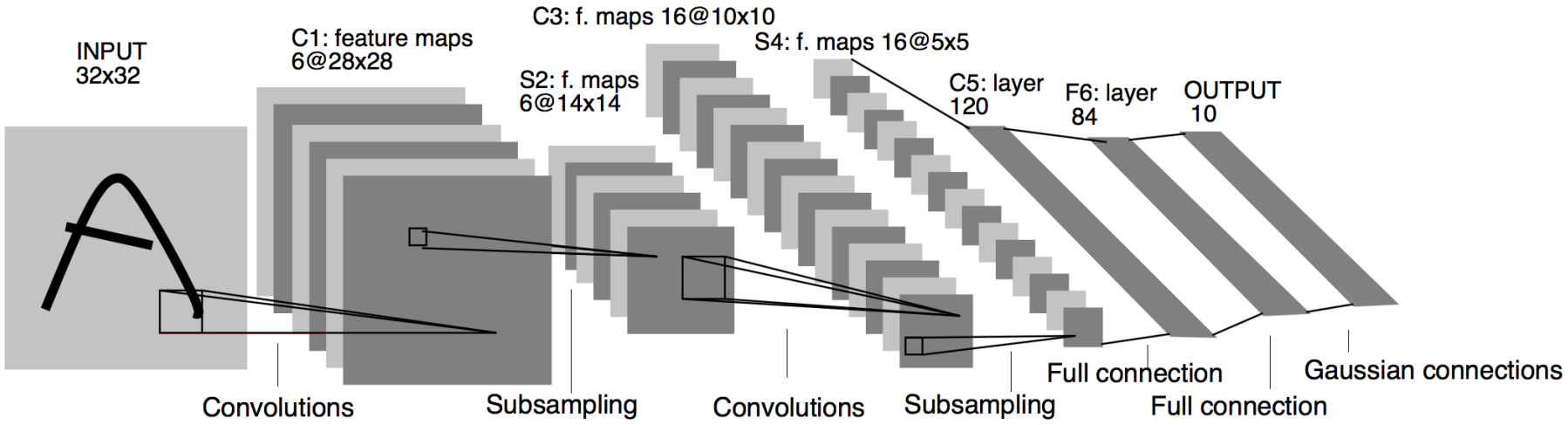


LeNet-5 architecture as published in the 1998, paper by LeCun et al.

LeNet-5 layers:

- **Convolution #1. Input = 32x32x1. Output = 28x28x6 conv2d**
- **SubSampling #1. Input = 28x28x6. Output = 14x14x6. SubSampling is simply Average Pooling so we use avg_pool**
- **Convolution #2. Input = 14x14x6. Output = 10x10x16 conv2d**
- **SubSampling #2. Input = 10x10x16. Output = 5x5x16 avg_pool**
- **Fully Connected #1. Input = 5x5x16. Output = 120**
- **Fully Connected #2. Input = 120. Output = 84**
- **Output 10**

LeNet-5

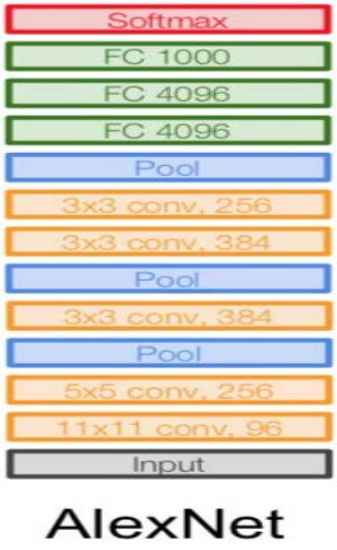


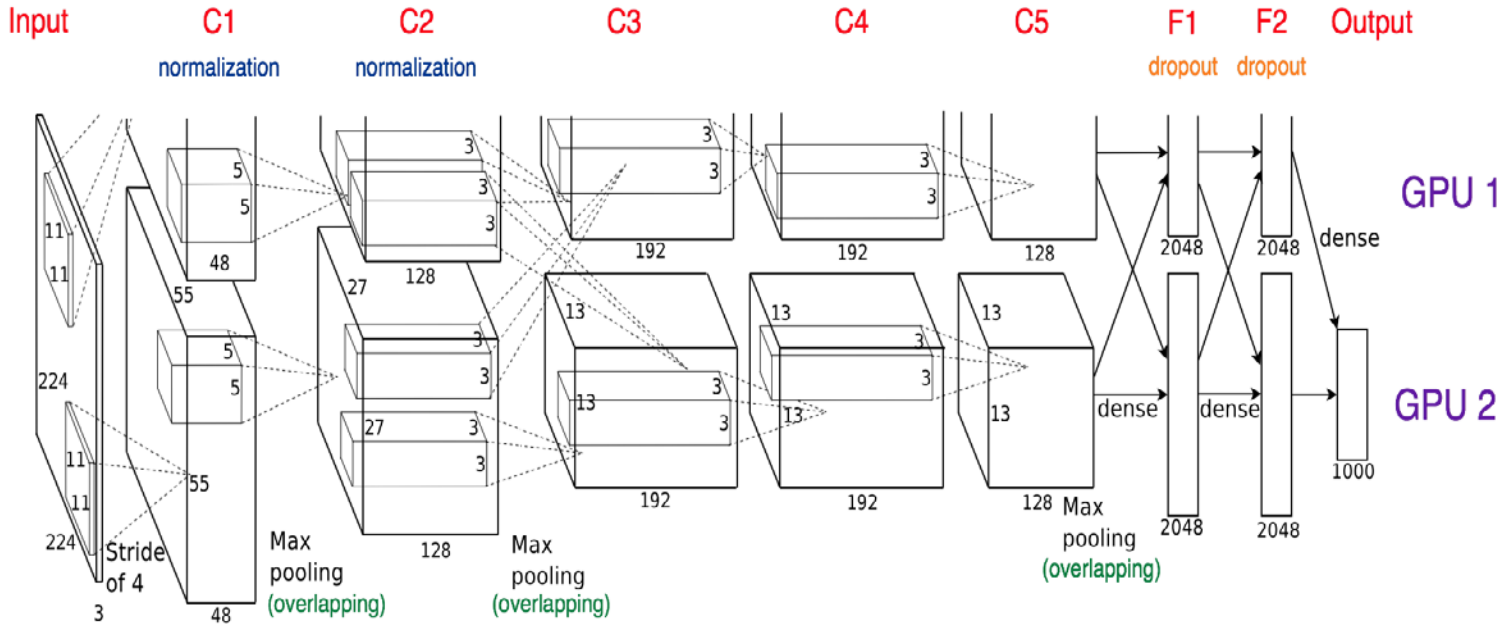
LeNet-5 architecture as published in the 1998, paper by LeCun et al.

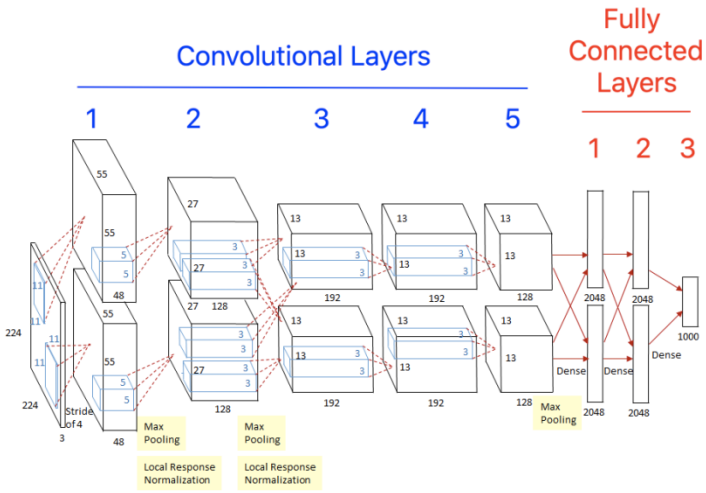
LeNet CNN Structure

Layer	Layer Type	Feature Maps	Kernel/Filter or Units	Input / Feature Map size	Trainable parameters	Connections	Strides	Activation Function
Input	Image	-	-	32x32	-	-	-	-
C1	Convolution	6	5x5	28x28	156	122,304		Hyperbolic tangent (tanh)
S2	Sub Sampling	6	2x2	14x14	12	5,880		Sigmoid
C3	Convolution	16	5x5	10x10	1,516	151,600		Hyperbolic tangent (tanh)
S4	Sub Sampling	16	2x2	5x5	32	2,000		Sigmoid
C5	Convolution	120	5x5	1x1	48,120			Hyperbolic tangent (tanh)
F6	Fully Connected	-	-	84	10,164			Hyperbolic tangent (tanh)
Output	Fully Connected	-	-	10				Softmax

AlexNet







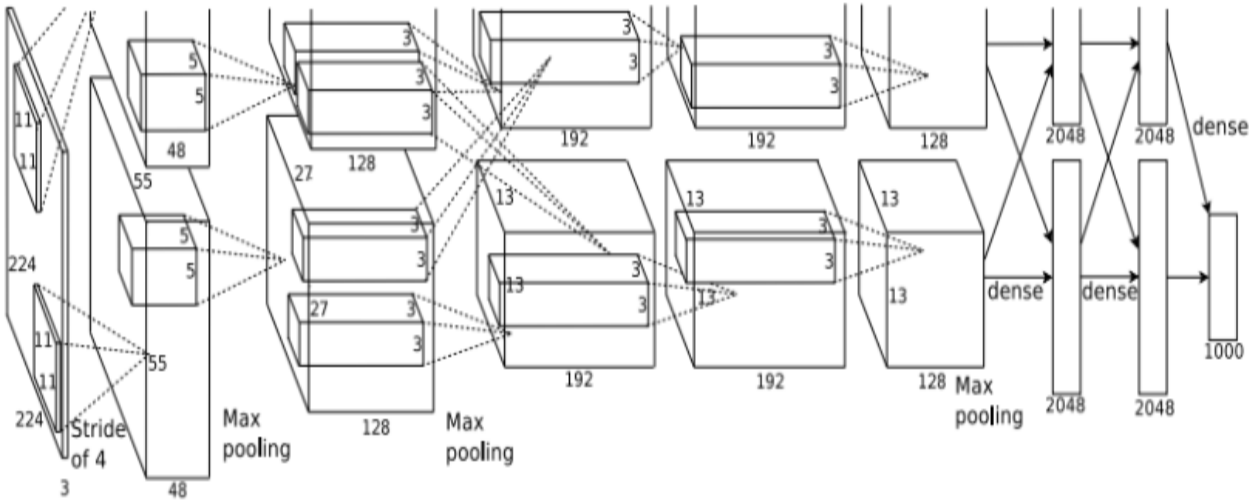
AlexNet contains 8 layers.

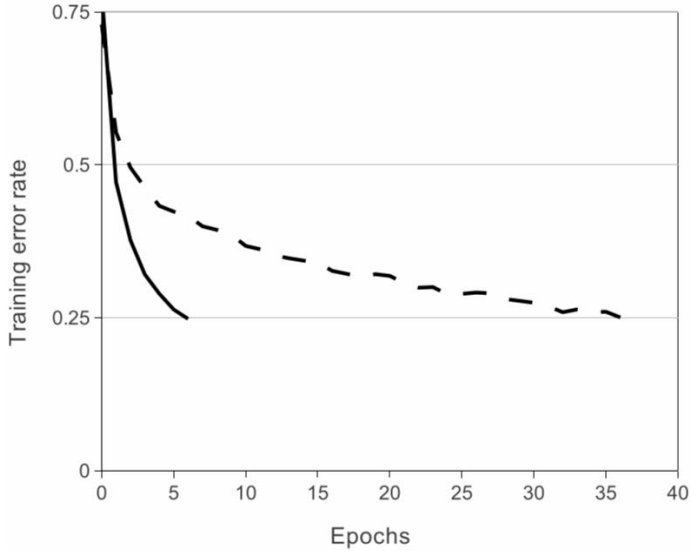
Five convolutional and

Three fully-connected

- **First use of ReLU**
- **Used Norm layers**
- **Dropout 0.5**
- **Batch size 128**
- **SGD Momentum 0.9**
- **Learning rate 0.01,**
- **L2 weight decay 5e-4**
- **7 CNN ensemble: 18.2%**

AlexNet

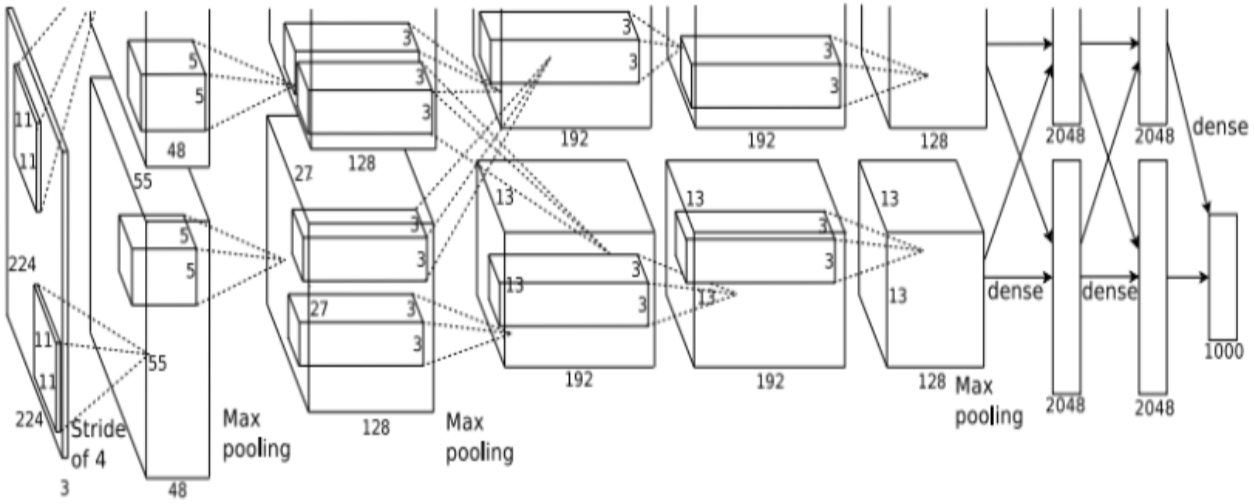




Main ideas

- **ReLU nonlinearity,**
- **training on multiple GPUs,**
- **local response normalization,**
- **overlapping pooling,**
- **data augmentation,**
- **dropout**

AlexNet



Main ideas

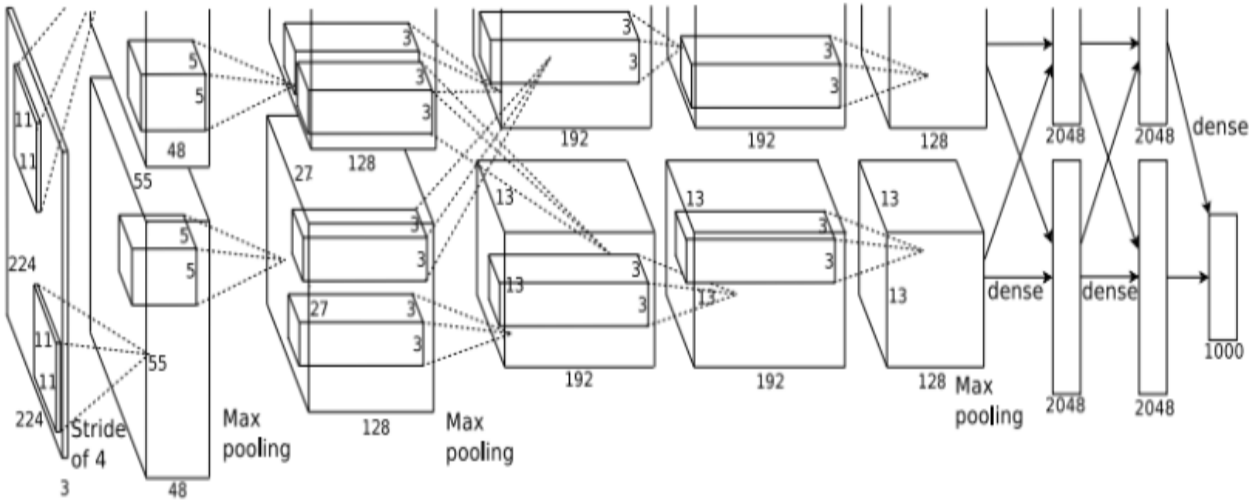
- **ReLU nonlinearity,**
- **training on multiple GPUs,**
- **local response normalization,**
- **overlapping pooling,**
- **data augmentation,**
- **dropout**

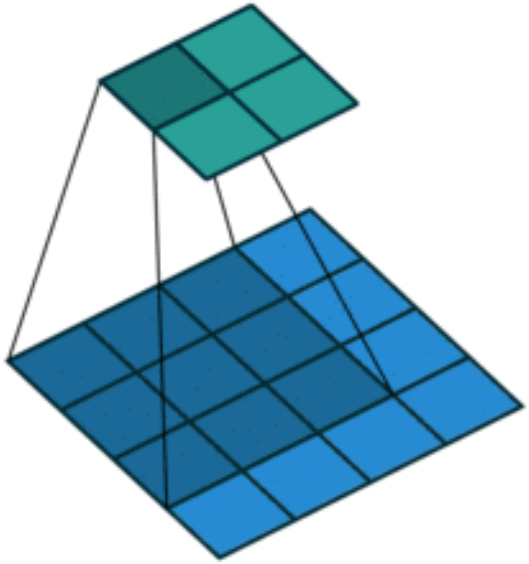
$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

$k = 2$
$n = 5$
$\alpha = 10^{-4}$
$\beta = 0.75$

i	Activity	Normalized
2	4	1.74
3	16	6.97
4	100	43.26
5	0	0.00
6	17	7.34
7	10	5.78
8	16	9.25

AlexNet

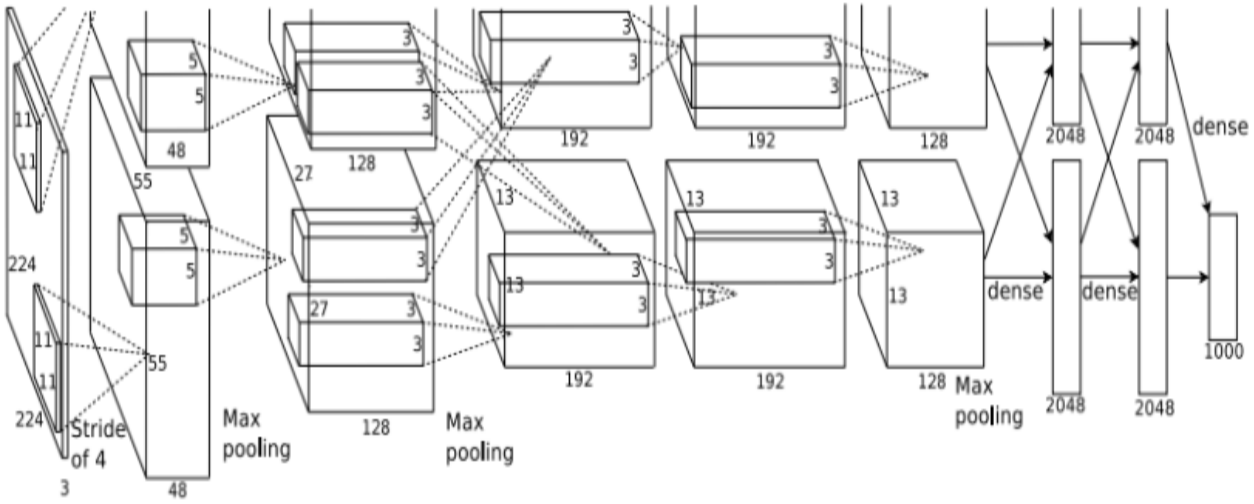




Main ideas

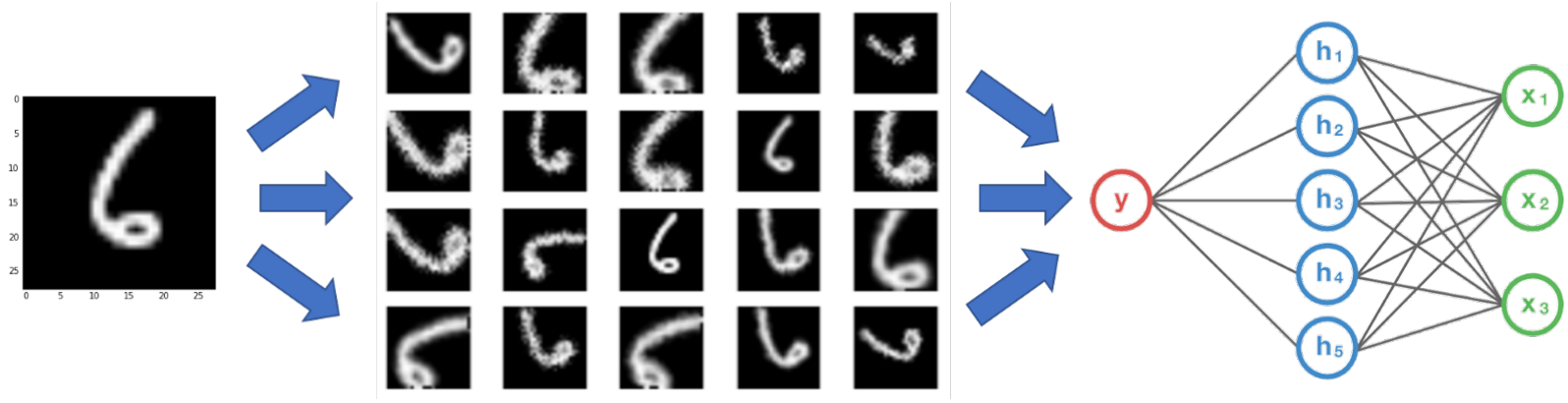
- **ReLU nonlinearity,**
- **training on multiple GPUs,**
- **local response normalization,**
- **overlapping pooling,**
- **data augmentation,**
- **dropout**

AlexNet

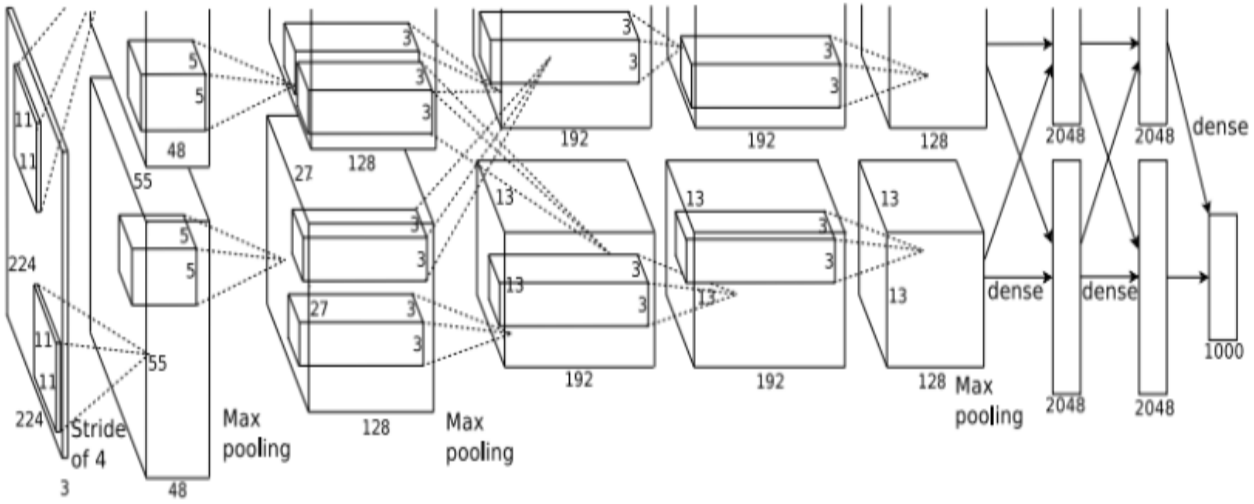


Main ideas

- **ReLU nonlinearity,**
- **training on multiple GPUs,**
- **local response normalization,**
- **overlapping pooling,**
- **data augmentation,**
- **dropout**

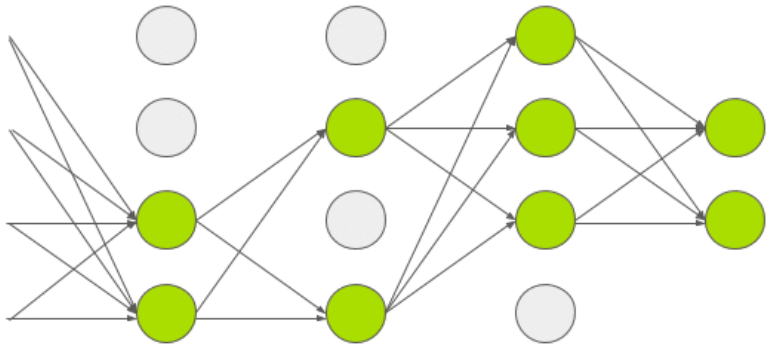


AlexNet

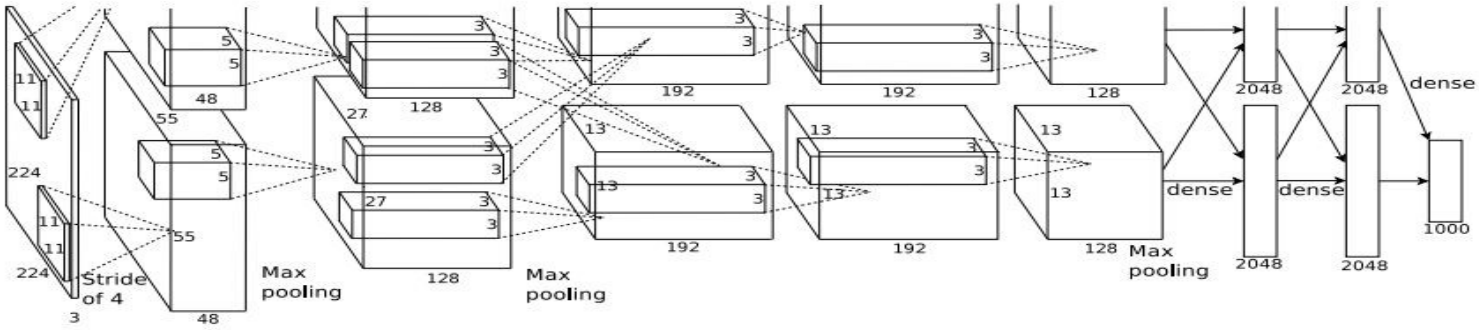


Main ideas

- **ReLU nonlinearity,**
- **training on multiple GPUs,**
- **local response normalization,**
- **overlapping pooling,**
- **data augmentation,**
- **dropout**



AlexNet



CONV1
MAX POOL1
[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)

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

AlexNet

#	layer name	output size	#filters	filter size	stride	padding	#parameters
1	INPUT	227x227x3	-	-	-	-	-
2	CONV1	55x55x96	96	11x11	4	-	(11*11*3+1)*96=349...
3	MAX POOL1	27x27x96	-	3x3	2	-	-
4	NORM1	27x27x96	-	-	-	-	-
5	CONV2	27x27x256	256	5x5	1	2	(5*5*96+1)*256=614...
6	MAX POOL2	13x13x256	-	-	-	-	-
7	NORM2	13x13x256	-	-	-	-	-
8	CONV3	13x13x384	384	3x3	1	1	(3*3*256+1)*384=88...
9	CONV4	13x13x384	384	3x3	1	1	(3*3*384+1)*384=13...
10	CONV5	13x13x256	256	3x3	1	1	(3*3*384+1)*256=88...
11	MAX POOL3	6x6x256	-	3x3	2	-	-
12	FC6	4096	-	-	-	-	256*6*6*4096=3774...
13	FC7	4096	-	-	-	-	4096*4096=16777216
14	FC8	1000	-	-	-	-	4096*1000=4096000
15							

ZFNet

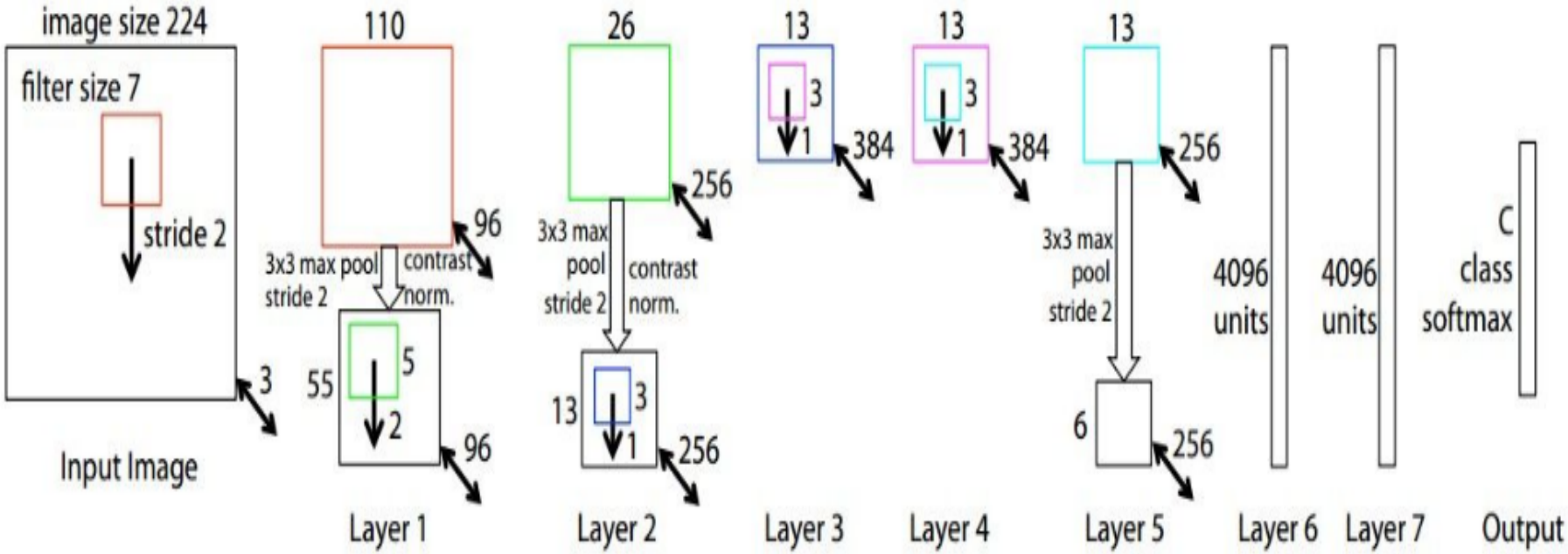
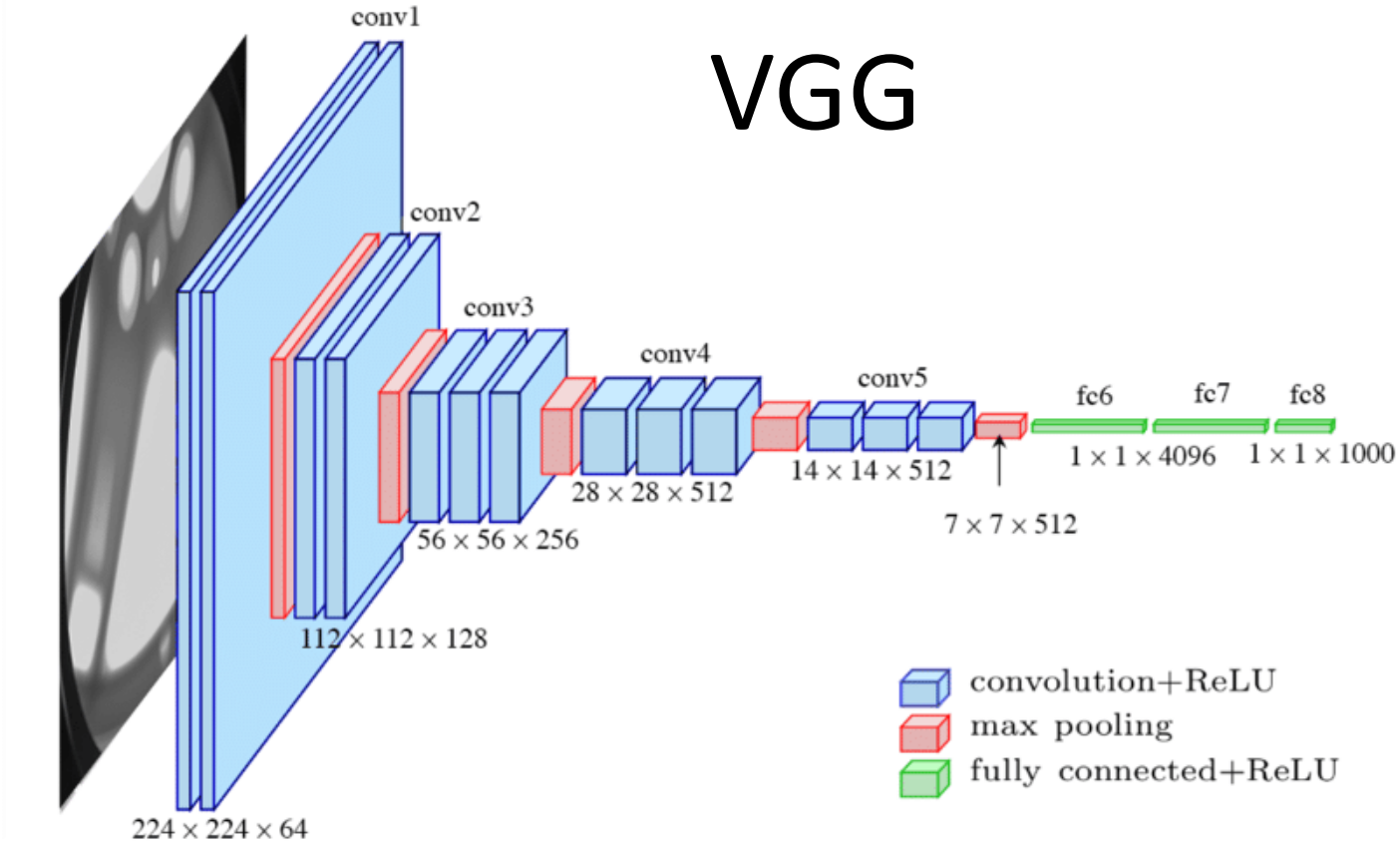


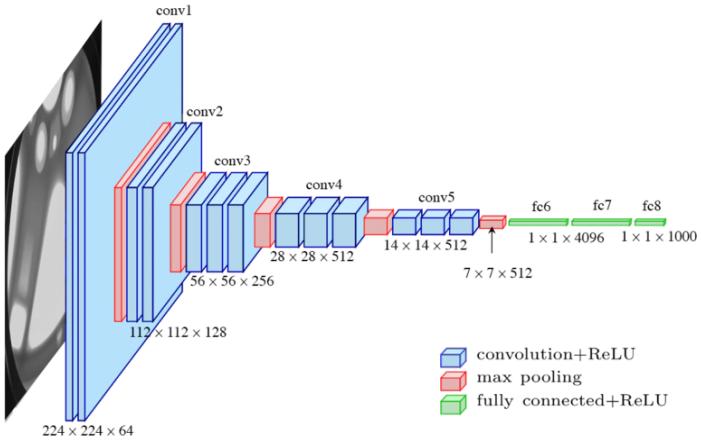
image size 224
filter size 7
stride 2
3
Input Image
110
96
3x3 max pool
stride 2
contrast
norm.
55
5
2
96
Layer 1
26
256
3x3 max
pool
stride 2
contrast
norm.
13
3
1
256
Layer 2
13
3
1
384
Layer 3
13
3
1
384
Layer 4
13
256
3x3 max
pool
stride 2
6
256
Layer 5
4096
units
Layer 6
4096
units
Layer 7
C
class
softmax
Output

VGG



- Developed by Simonyan and Zisserman for ILSVRC 2014 competition.
- VGGNet consists of 16 convolutional layers with only 3x3 kernels
- The design opted by the authors are similar to Alexnet i.e. increase the number of the features map or convolution as the depth of the network increases
- The network comprises of 138 million parameters.

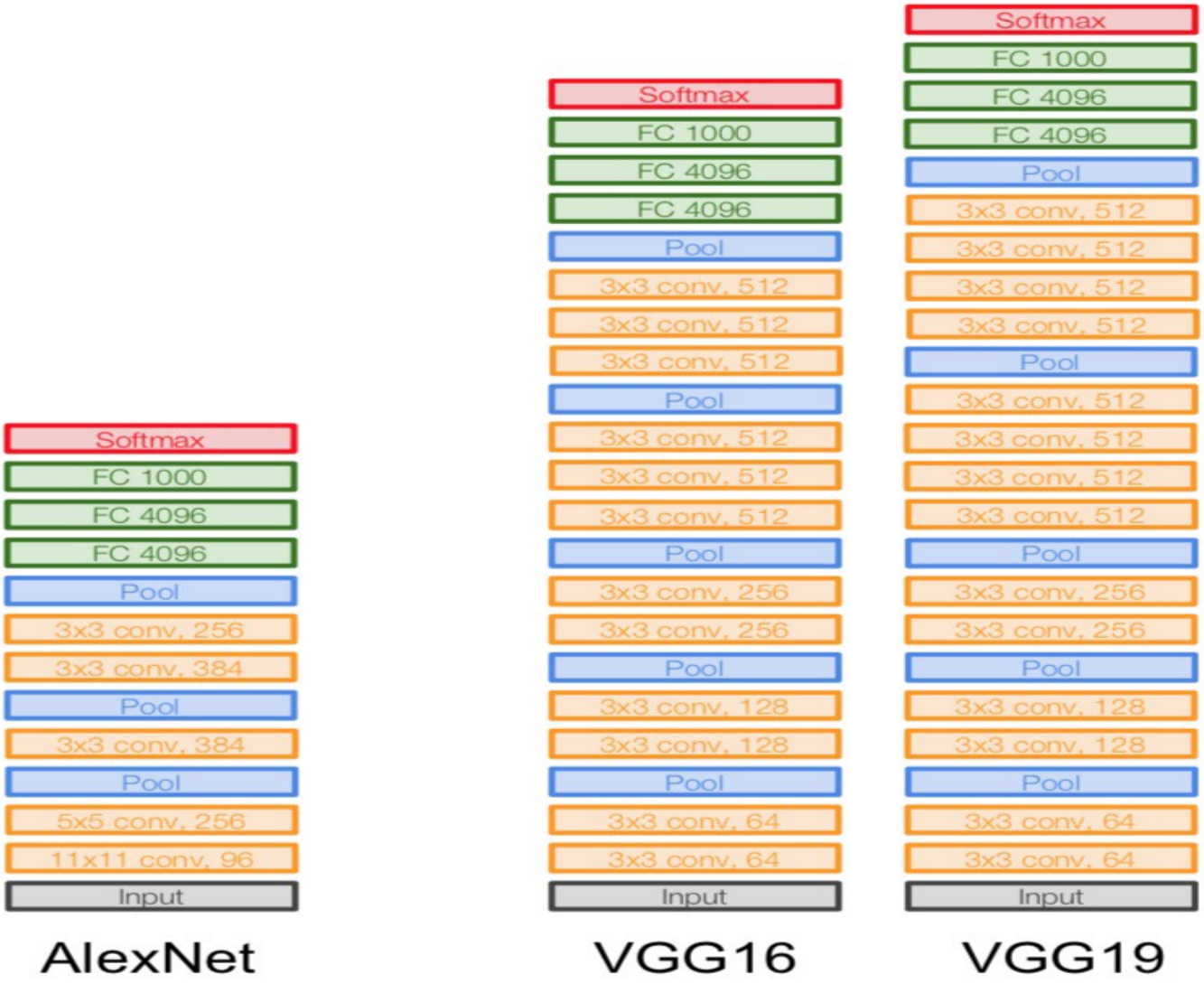
VGG



It is also called the OxfordNet model, named after the Visual Geometry Group from Oxford.
Number 16 refers that it has a total of 16 layers that has some weights.
It Only has Conv and pooling layers in it.
always use a 3 x 3 Kernel for convolution.
2x2 size of the max pool.
has a total of about 138 million parameters.
Trained on ImageNet data
It has an accuracy of 92.7%.
it has one more version of it Vgg 19, a total of 19 layers with weights.

VGG-16

VGG


Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 256
3x3 conv, 384
Pool
3x3 conv, 384
Pool
5x5 conv, 256
11x11 conv, 96
Input
AlexNet
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
VGG16
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
VGG19

VGG

Convolution Layer No.	Convolution filters, Strides and Padding	Convolution O/P dimension	Max-pooling Strides and Padding	Max-Pool O/P dimension
1 & 2	convolution layer of 64 channel of 3x3 kernel with padding 1, stride 1	224x224x64	Max pool stride=2, size 2x2	112x112x64
3 & 4	convolution layer of 128 channel of 3x3 kernel	112x112x128	Max pool stride=2, size 2x2	56x56x128
5, 6, 7	convolution layer of 256 channel of 3x3 kernel	56x56x256	Max pool stride=2, size 2x2	28x28x256
8, 9, 10	Convolution layer of 512 channel of 3x3 kernel	28x28x512	Max pool stride=2, size 2x2	14x14x512
11, 12, 13	Convolution layer of 512 channel of 3x3 kernel	14x14x512	Max pool stride=2, size 2x2	7x7x512

VGG

#	layer name	output size	#filters	filter size	stride	padding	#parameters
1	INPUT	224x224x3	-	-	-	-	-
2	CONV1-1	224x224x64	64	3x3	1	1	(3*3*3)*64=1728
3	CONV1-2	224x224x64	64	3x3	1	1	(3*3*64)*64=36,864
4	MAX POOL1	112x112x64	-	2x2	2	-	-
5	CONV2-1	112x112x...	128	3x3	1	1	(3*3*64)*128=73,728
6	CONV2-2	112x112x...	128	3x3	1	1	(3*3*128)*128=147,456
7	MAX POOL2	56x56x128	-	2x2	2	-	-
8	CONV3-1	56x56x256	256	3x3	1	1	(3*3*128)*256=294,912
9	CONV3-2	56x56x256	256	3x3	1	1	(3*3*256)*256=589,824
10	CONV3-3	56x56x256	256	3x3	1	1	(3*3*256)*256=589,824
11	MAX POOL3	28x28x256	-	2x2	2	-	-
12	CONV4-1	28x28x512	512	3x3	1	1	(3*3*256)*512=117,648
13	CONV4-2	28x28x512	512	3x3	1	1	(3*3*512)*512=2,359,...
14	CONV4-3	28x28x512	512	3x3	1	1	(3*3*512)*512=2,359,...
15	MAX POOL4	14x14x512	-	2x2	2	-	-
16	CONV5-1	14x14x512	512	3x3	1	1	(3*3*512)*512=2,359,...
17	CONV5-2	14x14x512	512	3x3	1	1	(3*3*512)*512=2,359,...
18	CONV5-3	14x14x512	512	3x3	1	1	(3*3*512)*512=2,359,...
19	MAX POOL5	7x7x512	-	2x2	2		-
20	FC6	4096	-	-	-	-	7*7*512*4096=102,7...
21	FC7	4096	-	-	-	-	4096*4096=16,777,216
22	FC8	1000	-	-	-	-	4096*1000=4,096,000

VGG

(not counting biases)

memory: 224*224*3=150K params: 0

memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728

memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864

memory: 112*112*64=800K params: 0

memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728

memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456

memory: 56*56*128=400K params: 0

memory: 56*56*256=800K params: (3*3*128)*256 = 294,912

memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

memory: 28*28*256=200K params: 0

memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648

memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

memory: 14*14*512=100K params: 0

memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

memory: 7*7*512=25K params: 0

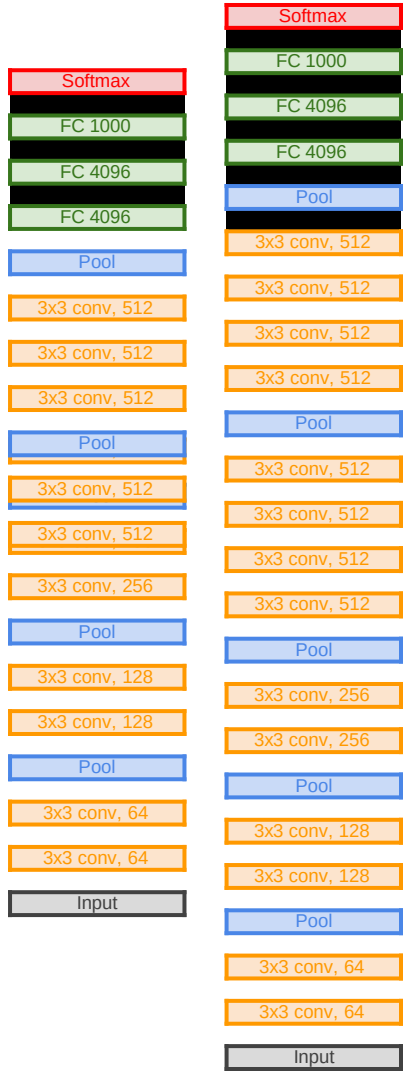
memory: 4096 params: 7*7*512*4096 = 102,760,448

memory: 4096 params: 4096*4096 = 16,777,216

memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image

TOTAL params: 138M parameters



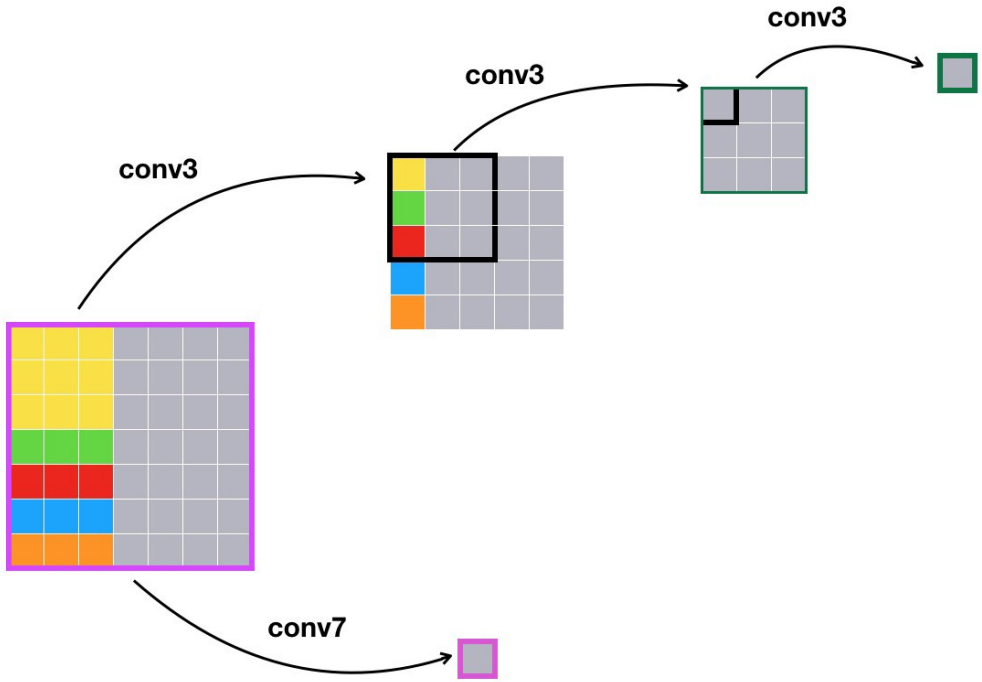
VGG

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

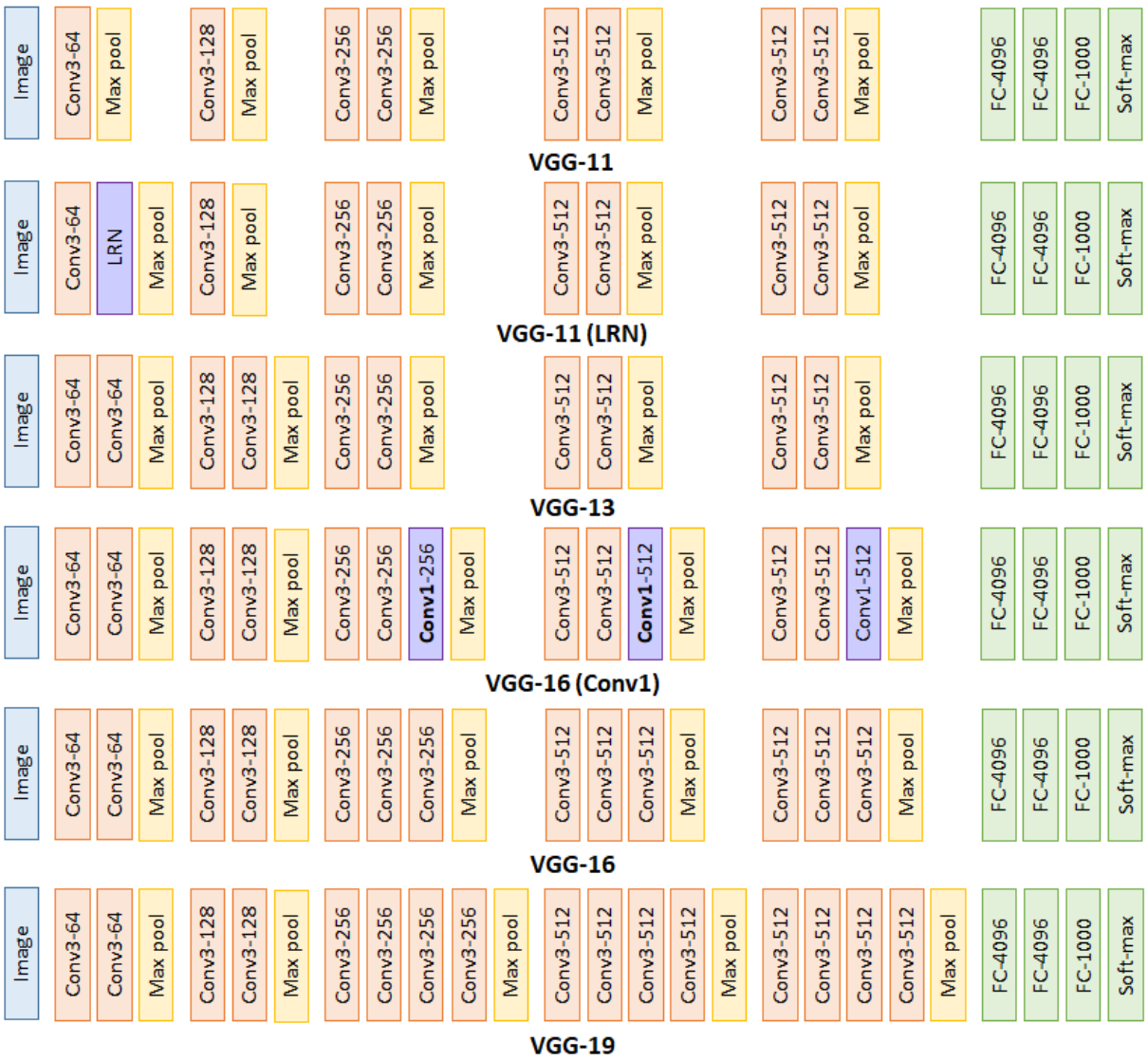
Table 2: **Number of parameters** (in millions).

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

VGG

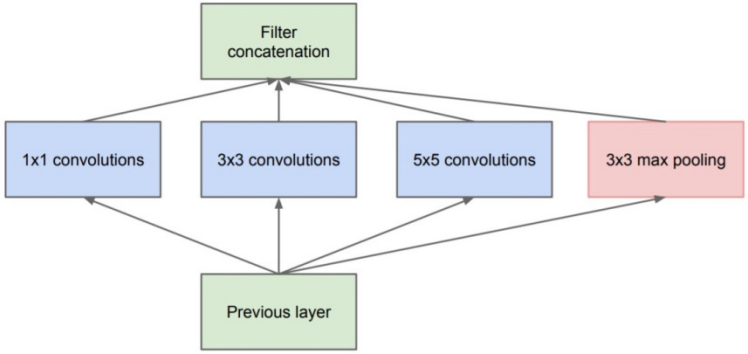


VGG

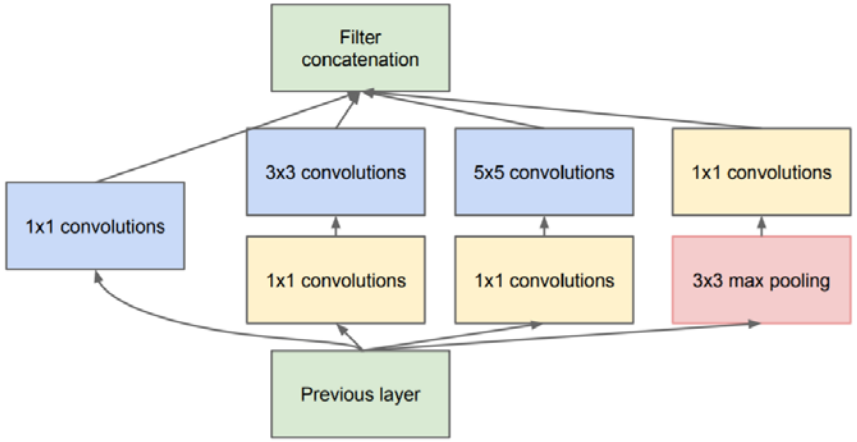


	Number of Parameters (millions)	Top-5 Error Rate (%)
VGG-11	133	10.4
VGG-11 (LRN)	133	10.5
VGG-13	133	9.9
VGG-16 (Conv1)	134	9.4
VGG-16	138	8.8
VGG-19	144	9.0

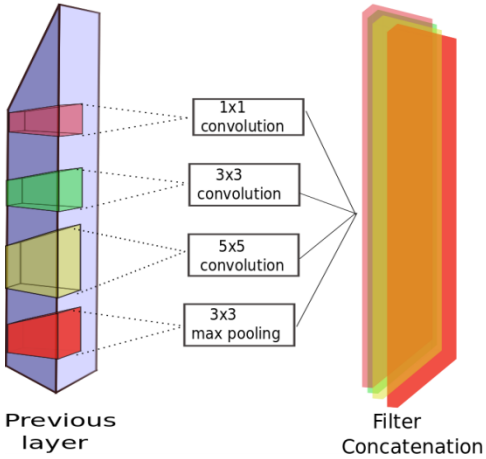
GoogLeNet

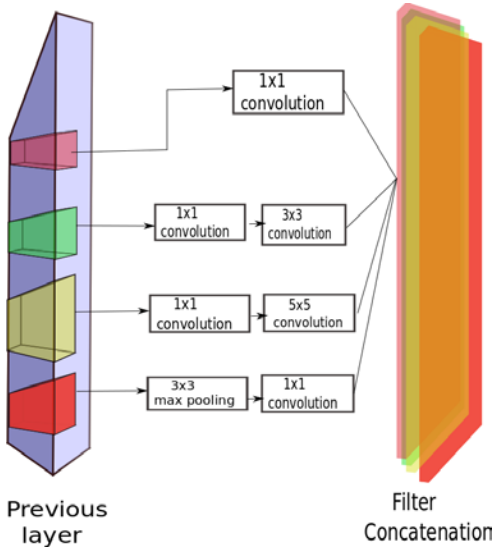


(a) Inception module, naïve version

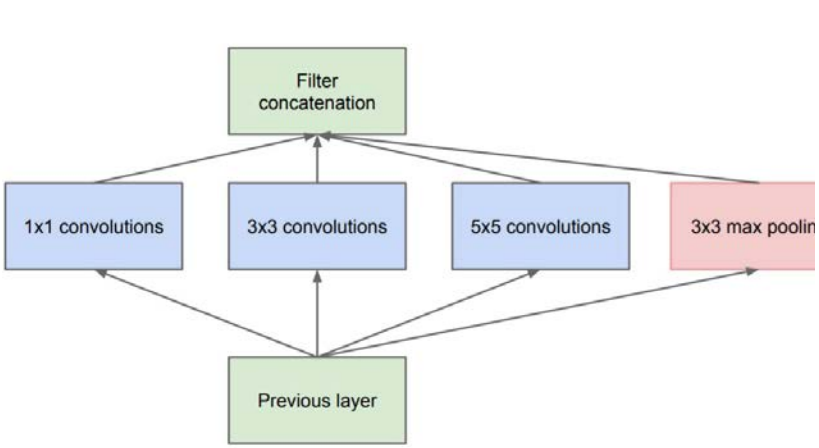


(b) Inception module with dimension reductions

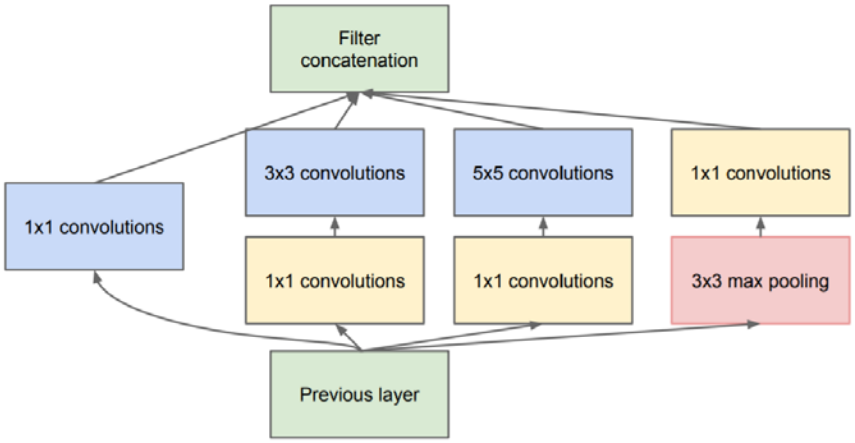




GoogLeNet



(a) Inception module, naïve version



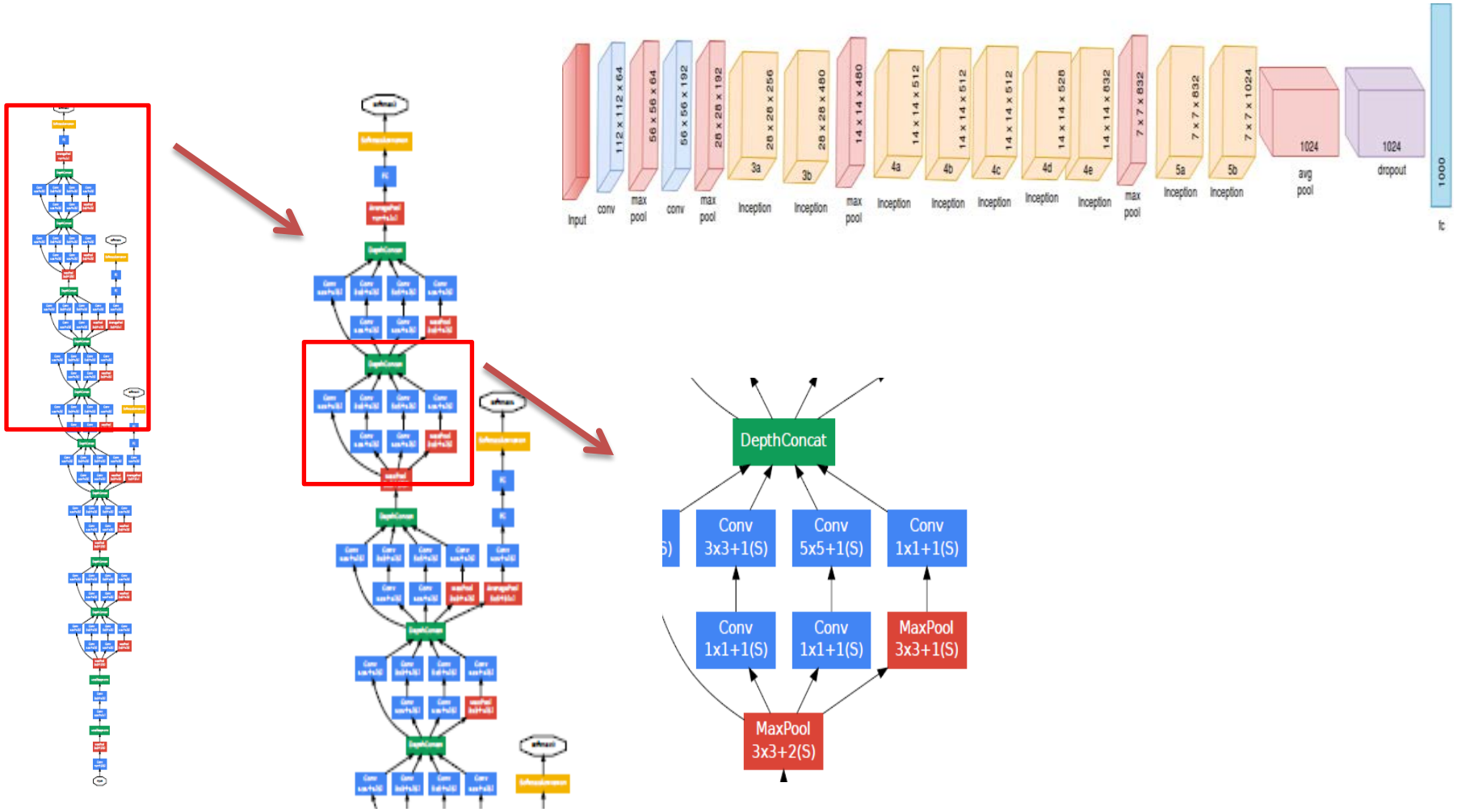
(b) Inception module with dimension reductions

dimension reductions — *1×1 convolutions used for computing reductions before the expensive 3×3 and 5×5 convolutions*
projections — *1×1 convolutions used for shielding a large number of input filters of the last stage to the next after max-pooling*

GoogLeNet

- **An average pooling layer with 5×5 filter size and stride 3, resulting in a 4×4×512 output for the (4a), and 4×4×528 for the (4d) stage.**
- **A 1×1 convolution with 128 filters for dimension reduction and rectified linear activation.**
- **A fully connected layer with 1024 units and rectified linear activation.**
- **A dropout layer with a 70% ratio of dropped outputs.**
- **A linear layer with softmax loss as the classifier (predicting the same 1000 classes as the main classifier, but removed at inference time).**

GoogleNet



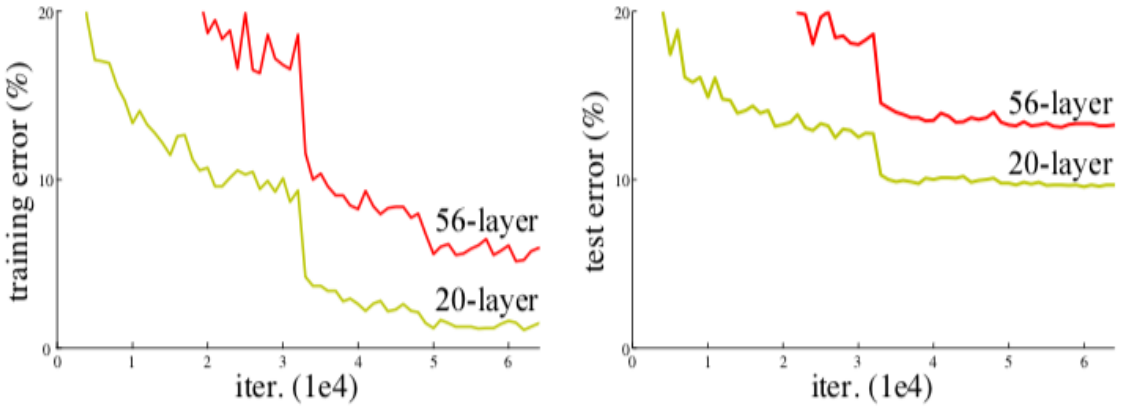
C. Szegedy et al., Going deeper with convolutions, CVPR 2015

GoogLeNet

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

ResNet

- Introduced by He et al. at the ILSVRC 2015 competition.
- They observed that *With network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly.*
- The network comprises of novel approach pathway called skip connection.
- These connection provide alternate pathway for data and gradients to flow and thus making training possible.
- This connectivity pattern aids in training network with 152 layers while being less complex than VGG.



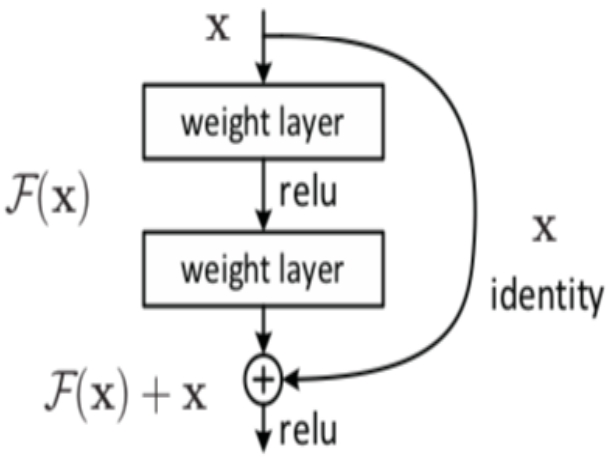
Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error

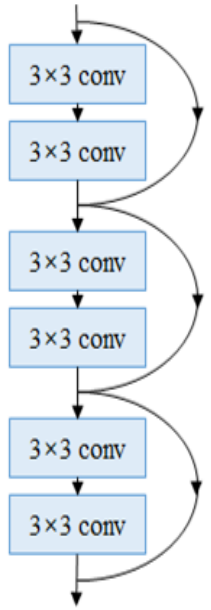
Image Courtesy: He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778).

ResNet

Core Idea of Residual Networks

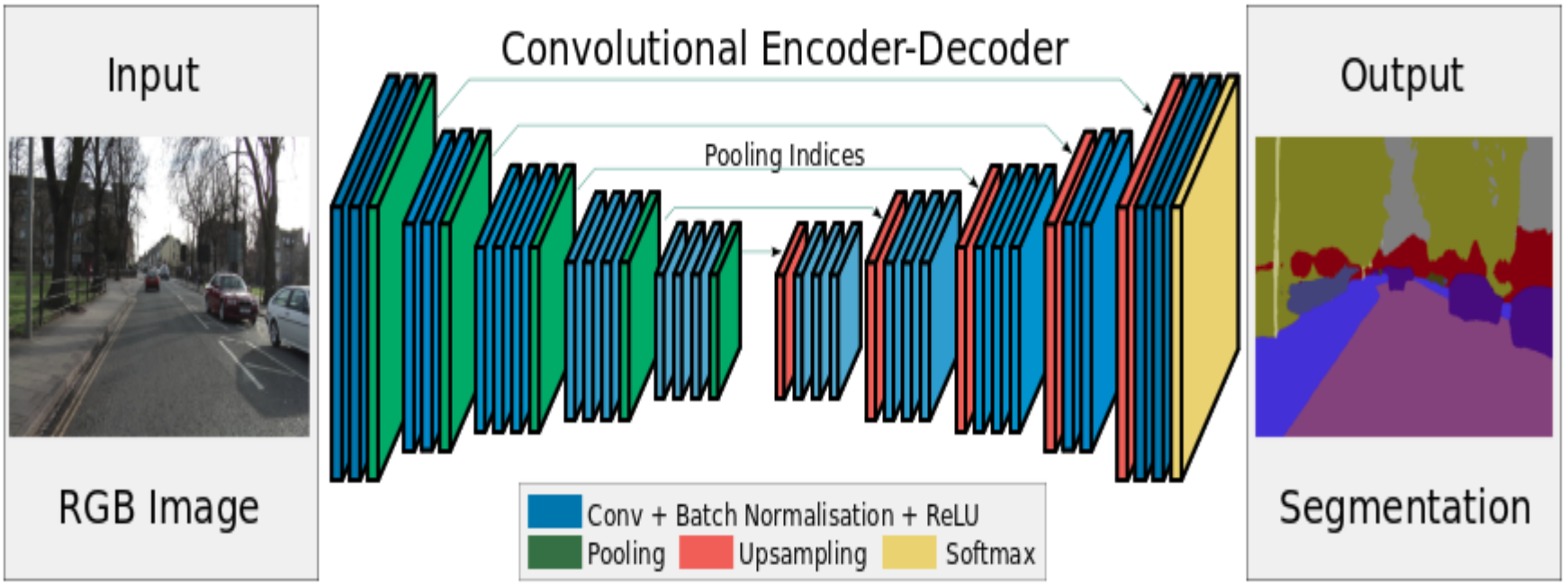
- A deeper network can be made from a shallow network.
- This is achieved by copying weights in shallow network and setting other layers in the deeper network to be identity mapping.
- This formulation indicates that the deeper model should not produce higher training error than the shallower counterpart.





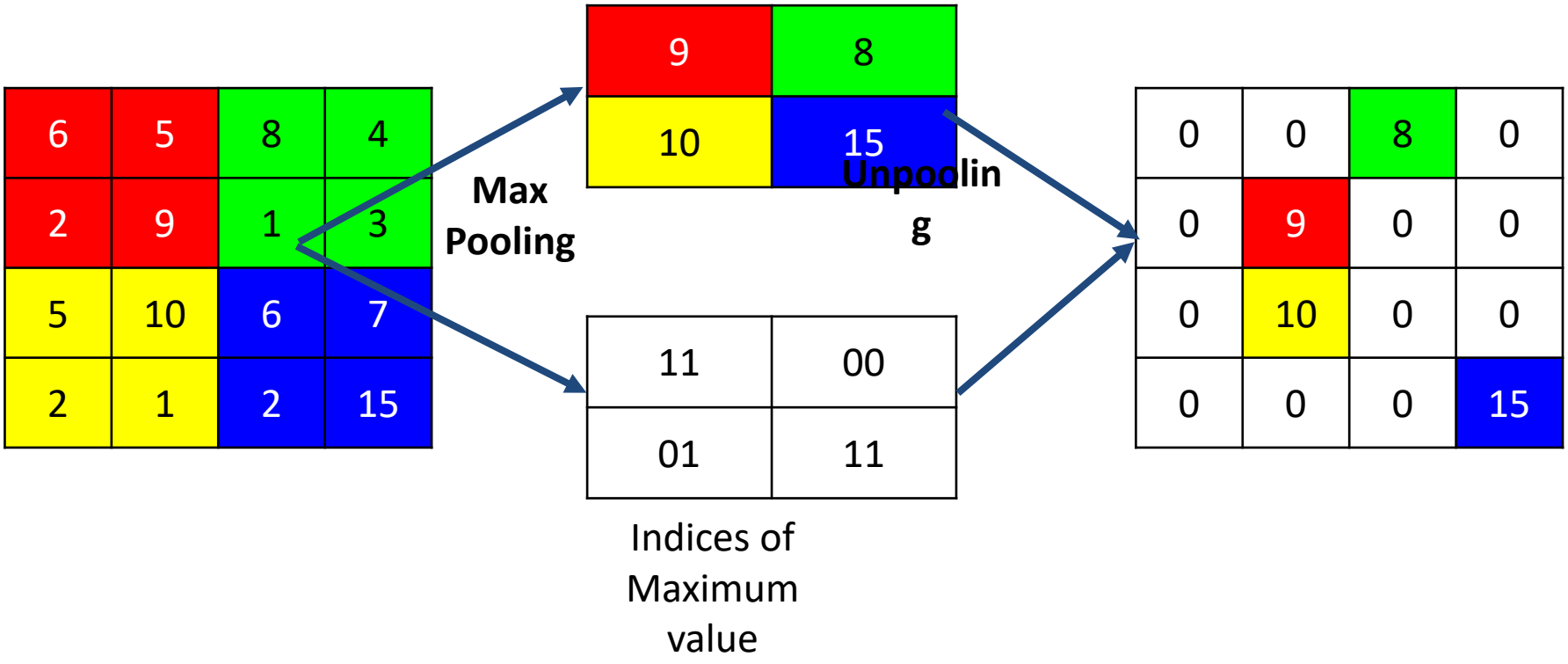
SegNet

1. Eliminates the need for learning to up-sample
2. Keeps high frequency details intact by using unpooling method

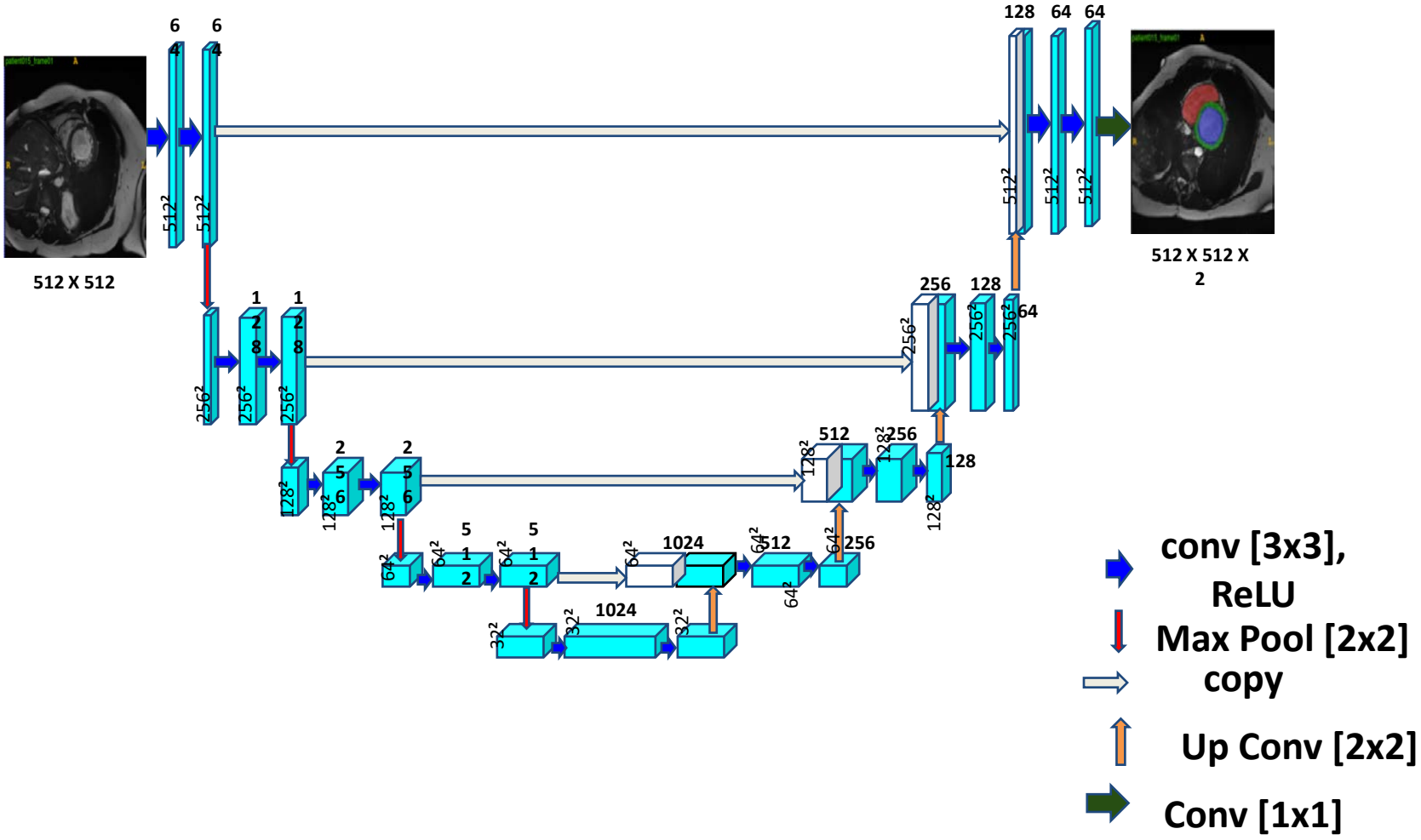


Badrinarayanan et al., "Segnet: A deep convolutional encoder-decoder architecture for image segmentation.", CVPR, 2015

Unpooling


6
5
8
4
2
9
1
3
5
10
6
7
2
1
2
15
Max Pooling
9
8
10
15
11
00
01
11
Indices of Maximum value
Unpoolin g
0
0
8
0
0
9
0
0
0
10
0
0
0
0
0
15

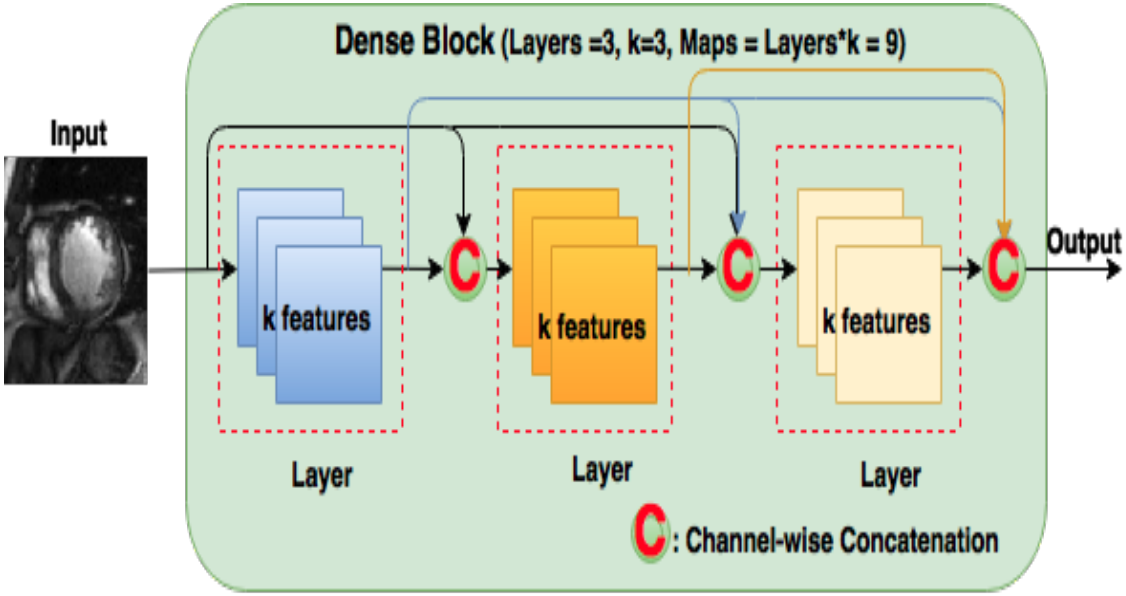
U-NET

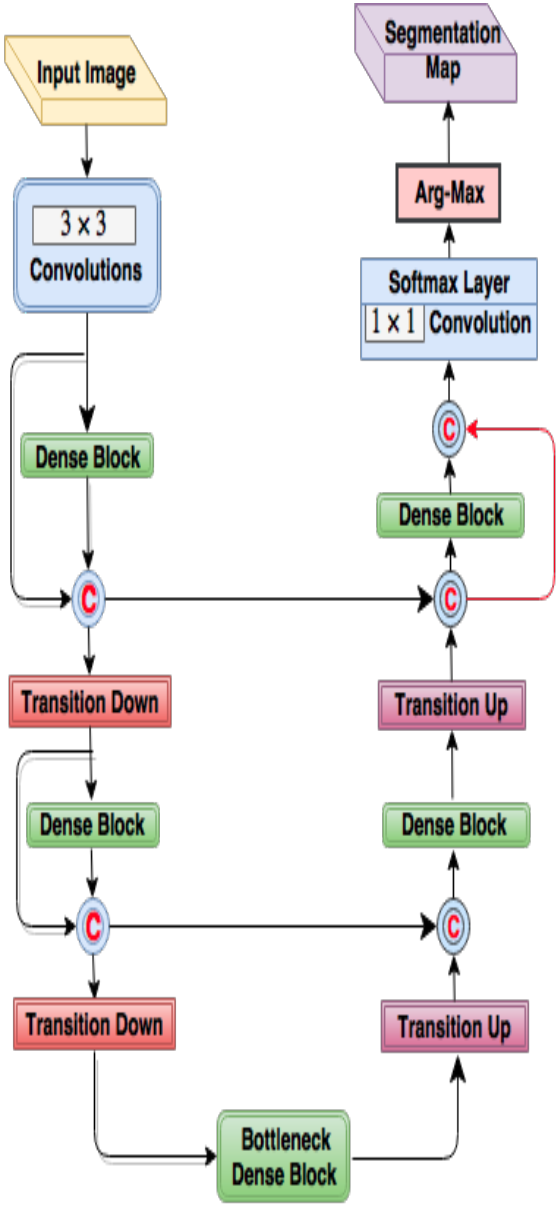


Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

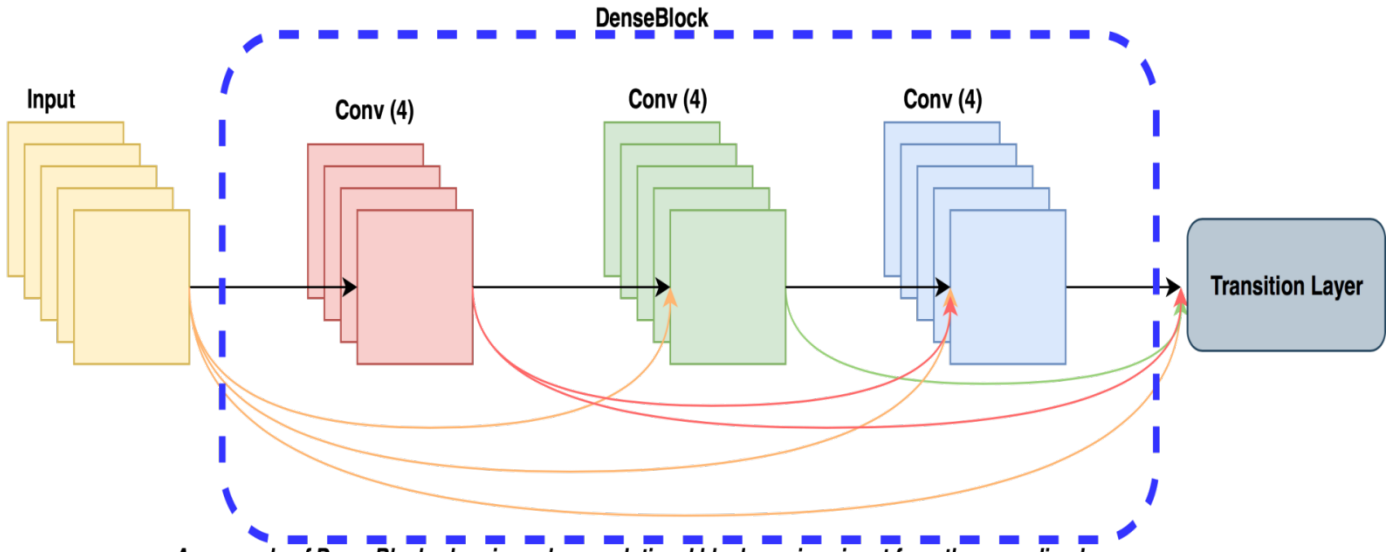
Fully Convolutional DenseNets

1. Encoder path of the network is based on DenseNets
2. Each layer is directly connected to all other layers
3. Extensive feature reuse and parameter efficient





DenseNet



- **As CNNs become deep, they are harder to train due to vanising gradient – Addressed by ResNet**
- **Key Observation – Creating short paths from early layers to later layers helps train deep networks**
- **Connect all layers directly with each other – Network with L layers will have L connections. Here there will be L(L+1)/2 connections**

G. Huang, Z. Liu, L. v. d. Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 2261-2269.

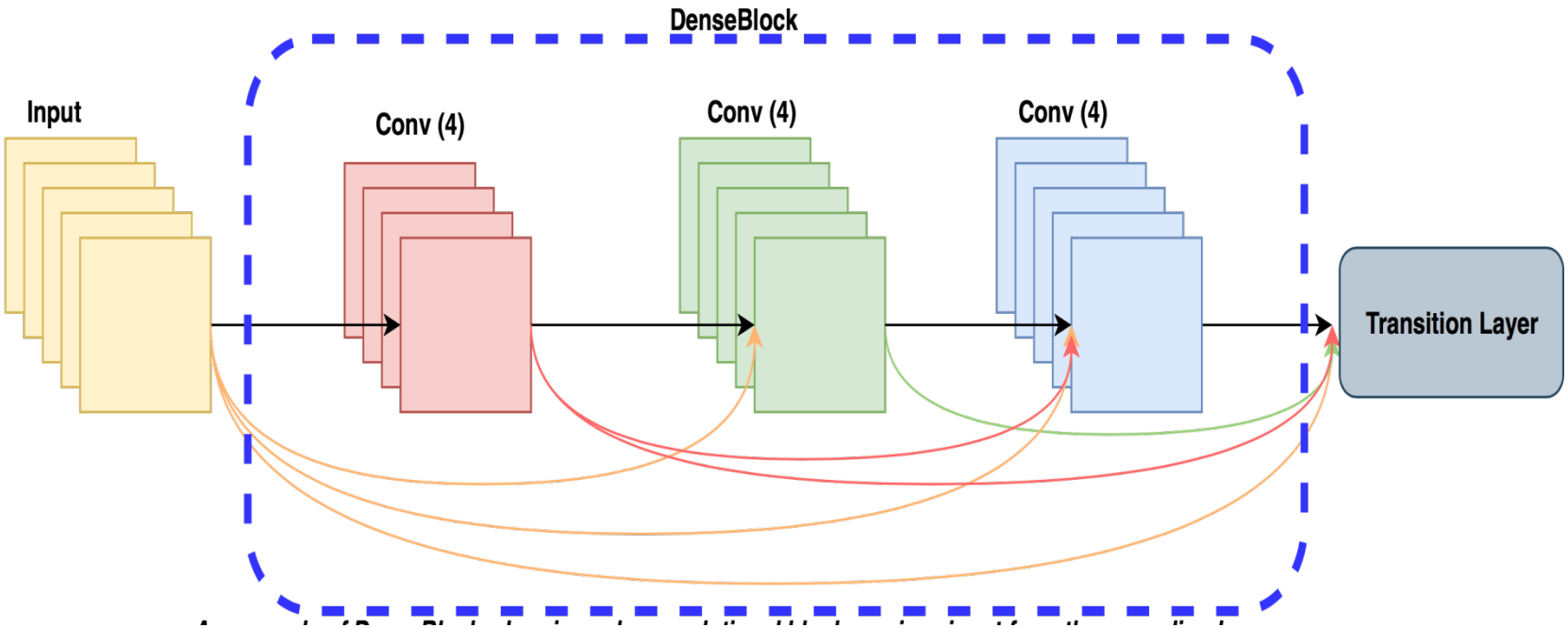
DenseNet

Advantages

- Parameter efficiency – Every layer adds only a limited number of parameters- for e.g. only about 12 kernels are learnt per layer
- Implicit deep supervision- Improved flow of gradient through the network- Feature maps in all layers have direct access to the loss function and its gradient.

G. Huang, Z. Liu, L. v. d. Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 2261-2269.

DenseNet



- Growth Rate
- Dense Connectivity
- Transition Layers
- Composite Function

G. Huang, Z. Liu, L. v. d. Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 2261-2269.

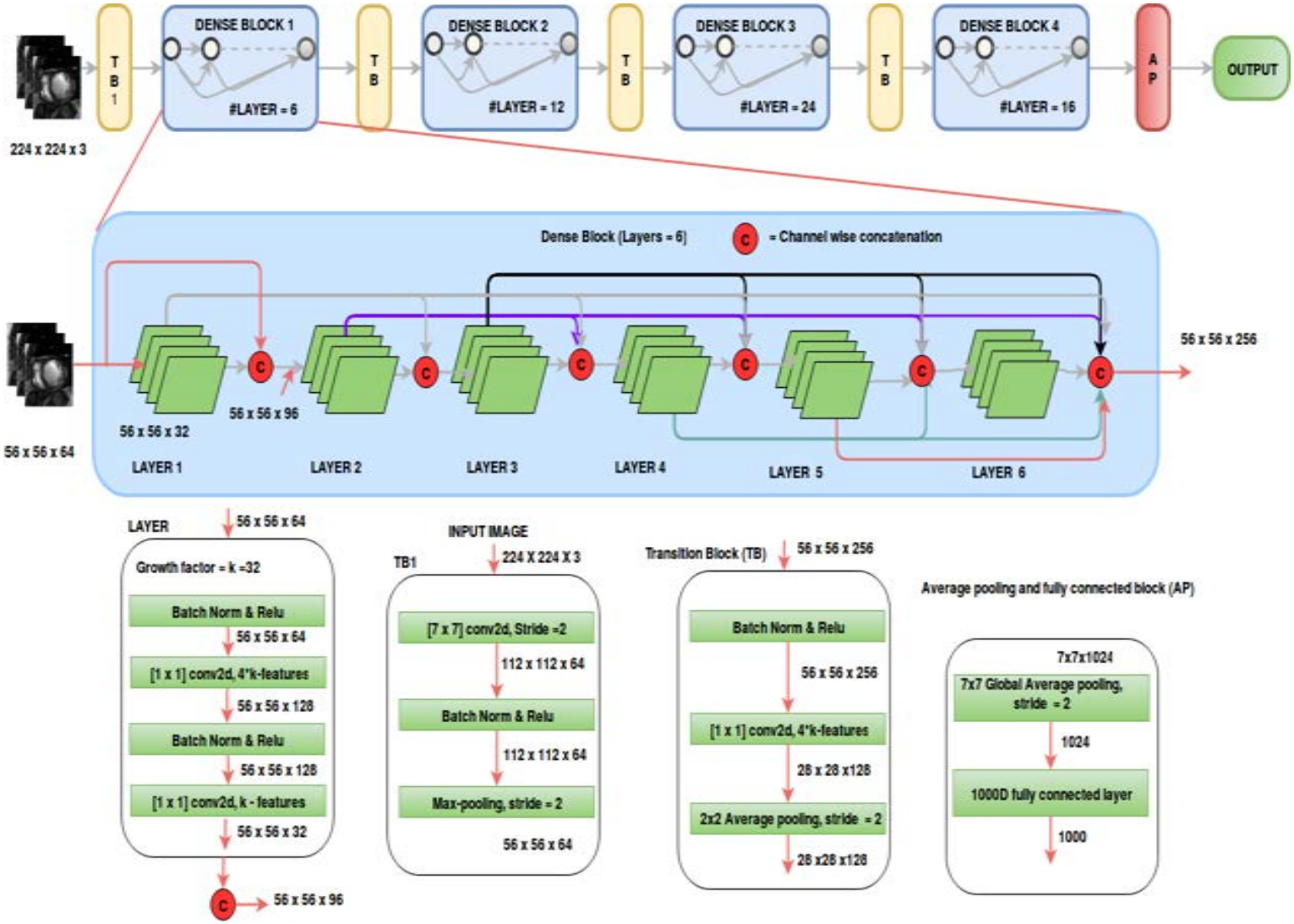
DenseNet

- Each Layer outputs k –feature maps, k is the growth factor
- Bottleneck Layer of 1x1 followed by 3x3 convolutions- 1x1 convolutions output 4k feature maps
- Initial conv Layer outputs 4k Feature maps, for ImageNet the initial conv layer outputs 2k feature maps

DenseNet for ImageNet Challenge

Layers	Output size	DenseNet-121,k=32
Convolution	112x112	
Pooling	56x56	
Dense Block 1	56x56	**[1x1->3x3]x6**
Transition layer 1	56x56-> 28x28	
Dense Block 2	28x28	**[1x1->3x3]x12**
Transition Layer 2	28x28-> 14x14	
Dense Block 3	14x14	**1x1->3x3]x 24**
Transition Layer 3	14x14->7x7	
Dense Block 4	7x7	**1x1->3x3]x16**
Classification Layer	1x1	**7x7 GAP-> 1000 Fully Connected softmax**

DenseNet for ImageNet Challenge

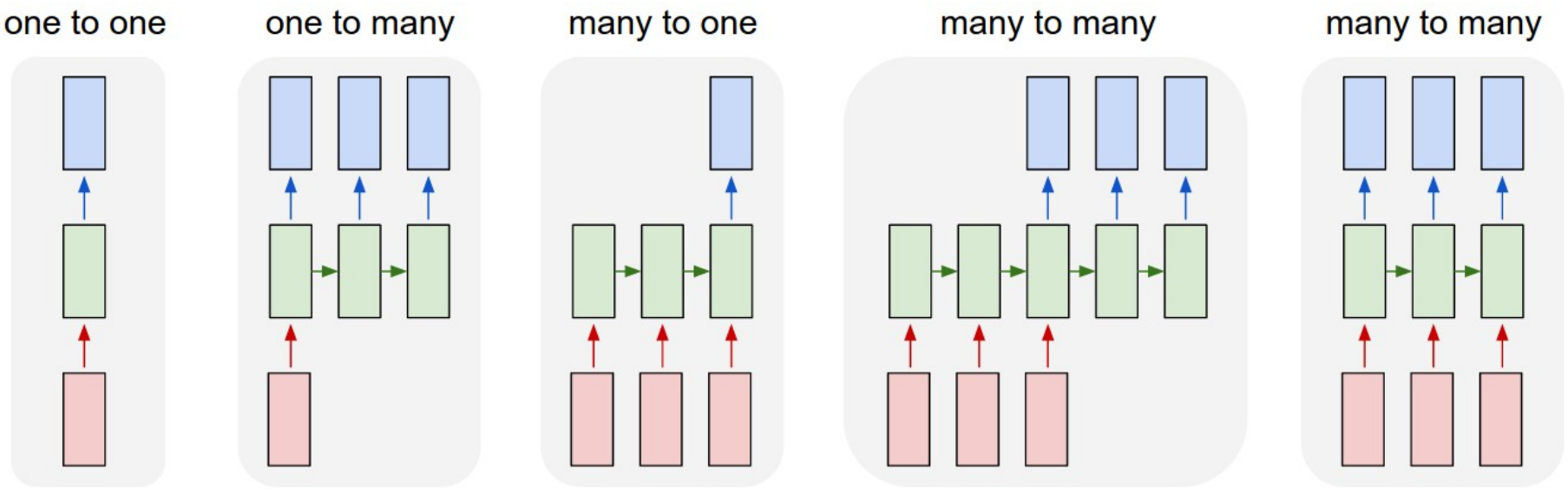


G. Huang, Z. Liu, L. v. d. Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 2261-2269.

RNN Architectures

- Vanilla RNN
- LSTM
- Bidirectional RNN
- GRU

RNN



Vanilla RNN

Image Captioning
Image → Sequence of Words

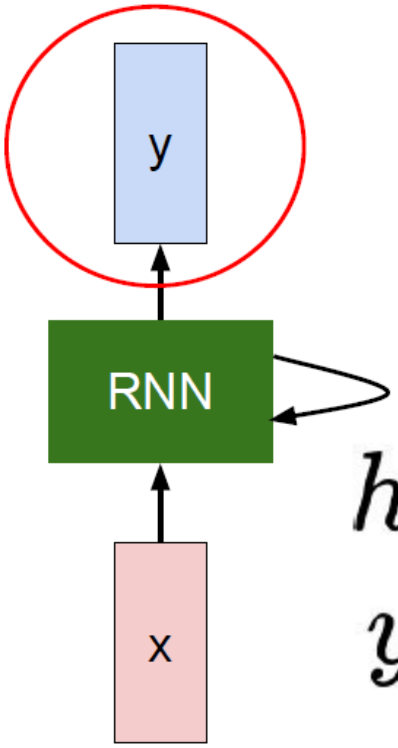
Sentiment Analysis
Sequence of Words → Sentiment

Machine Translation
Seq of Words → Seq of Words

Frame level
Video Classification

RNN

process a sequence of vectors **x by** applying a **recurrence formula at every time step**



$$h_t = f_w(h_{t-1}, x)$$

New State

Some Function *F* with parameters w

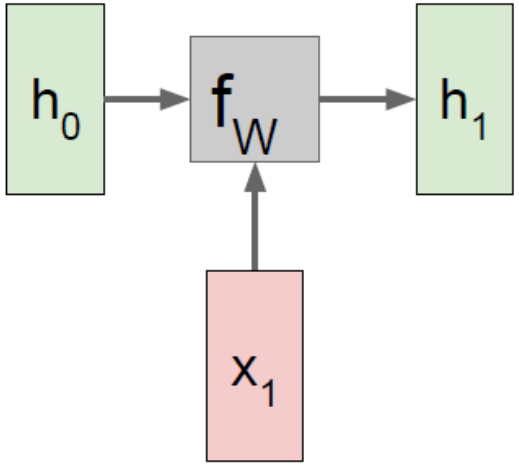
Old State

Input vector at some time step

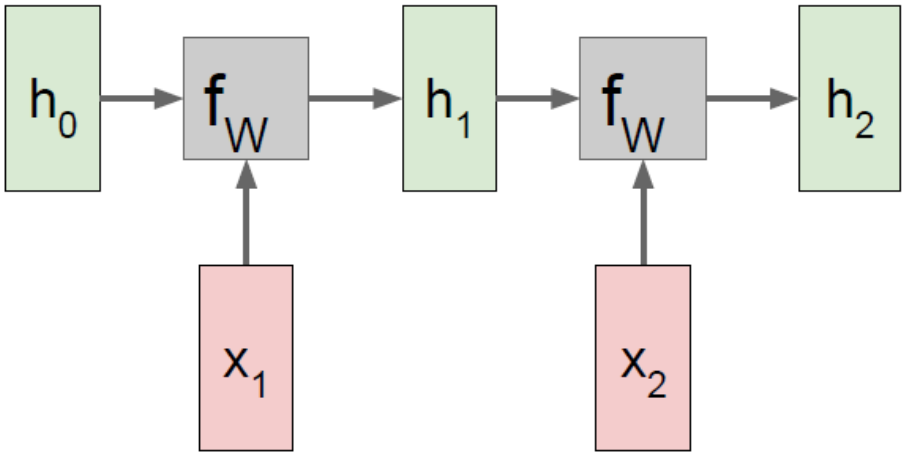
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

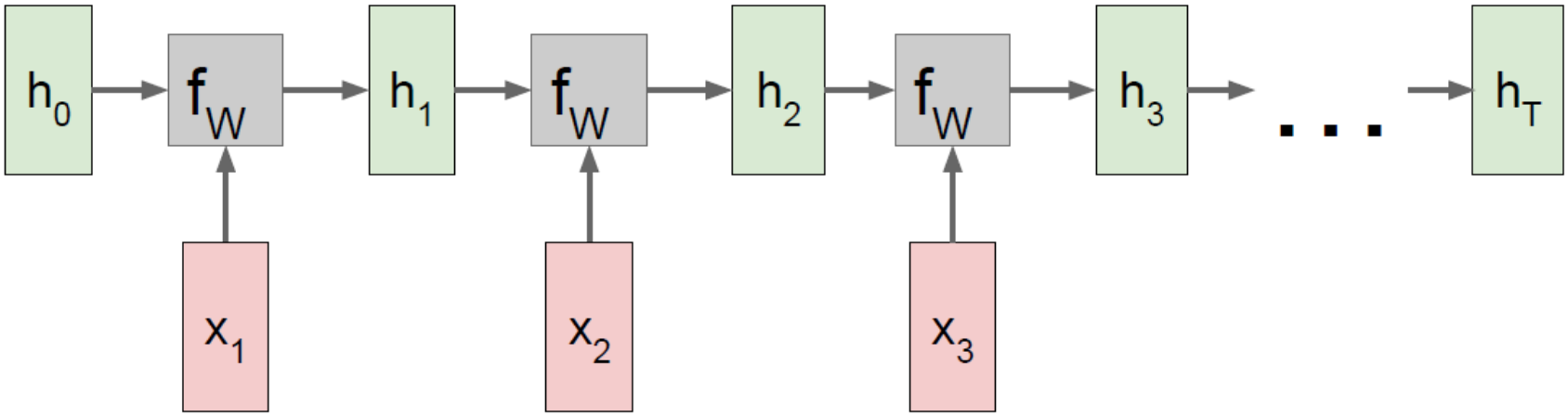
RNN: Computational Graph



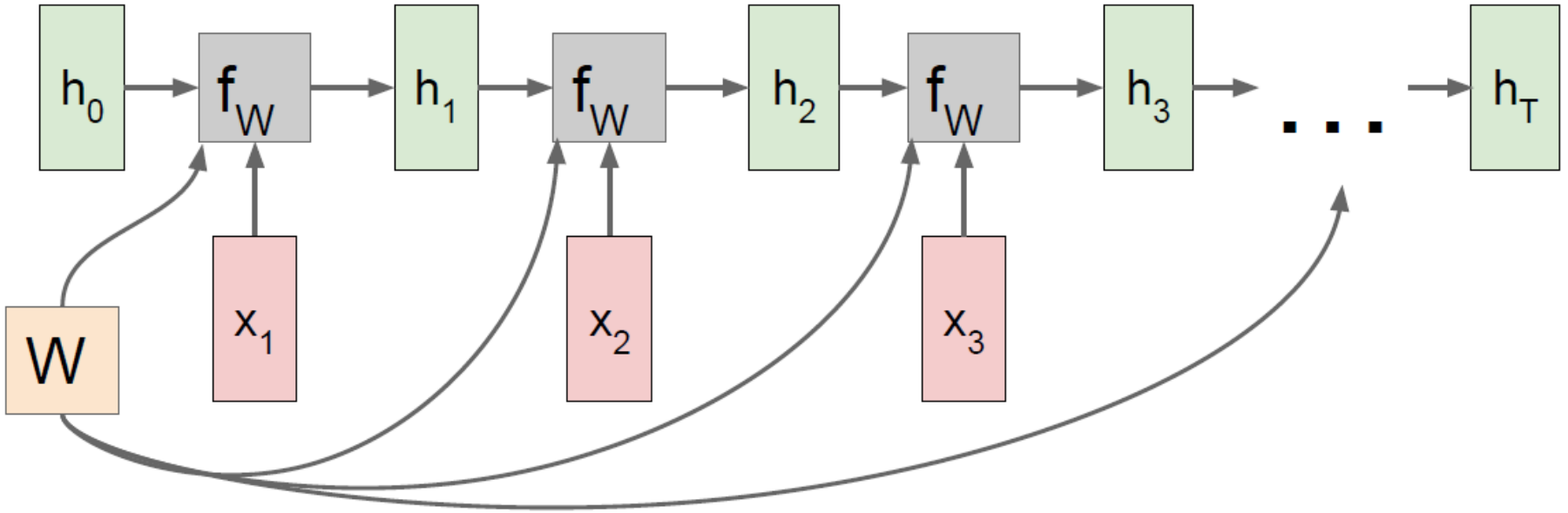
RNN: Computational Graph



RNN: Computational Graph

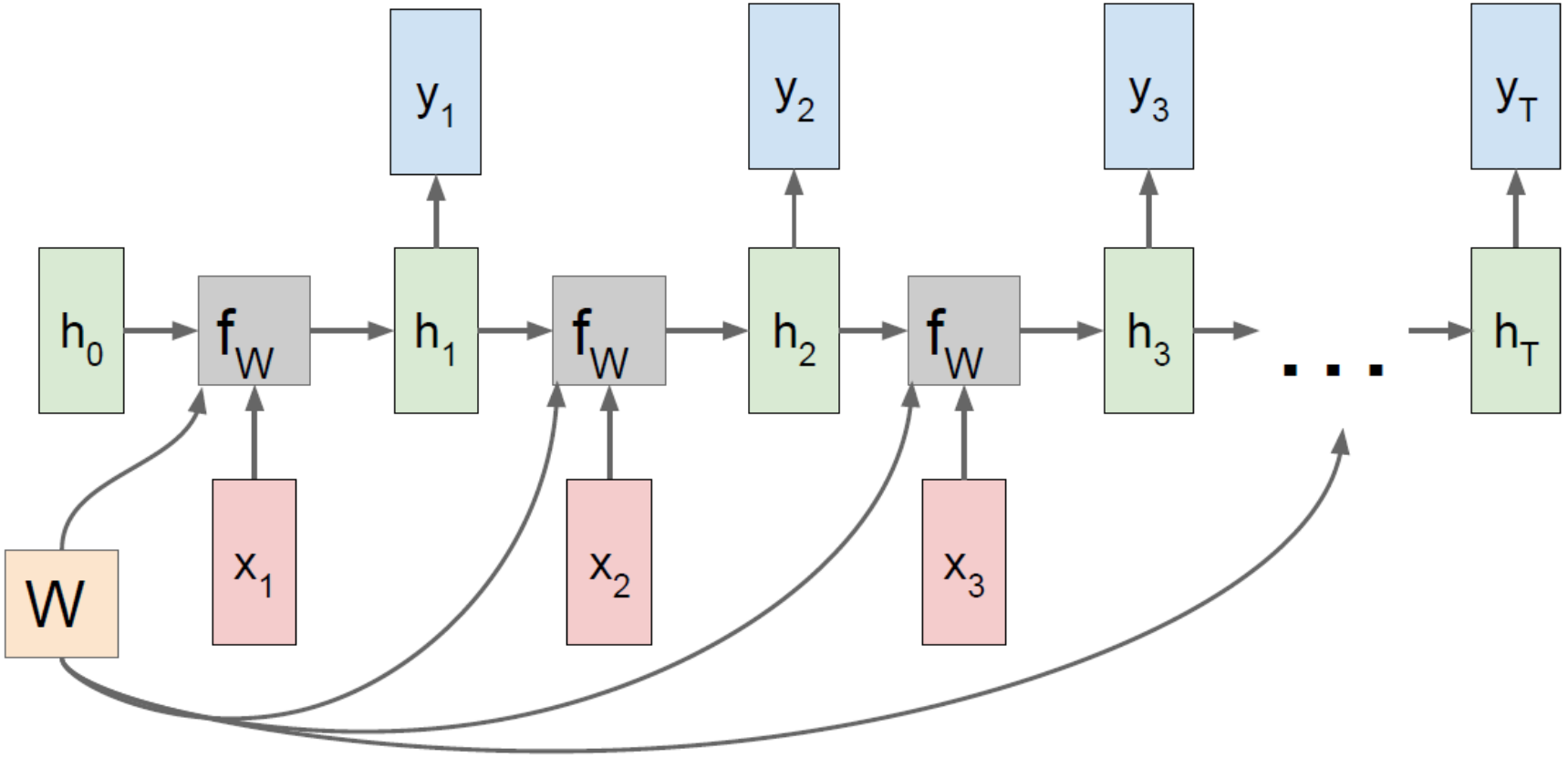


RNN: Computational Graph



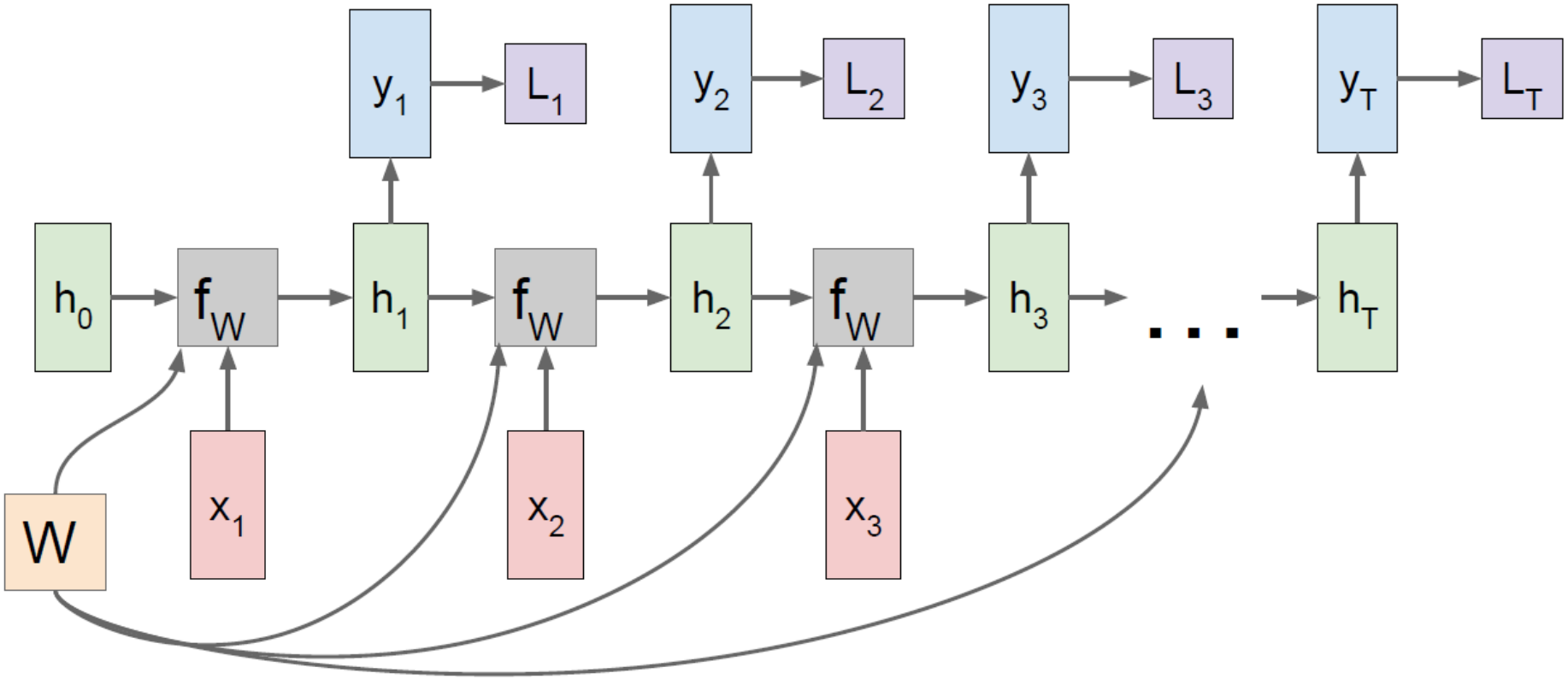
RNN: Computational Graph

Many to Many



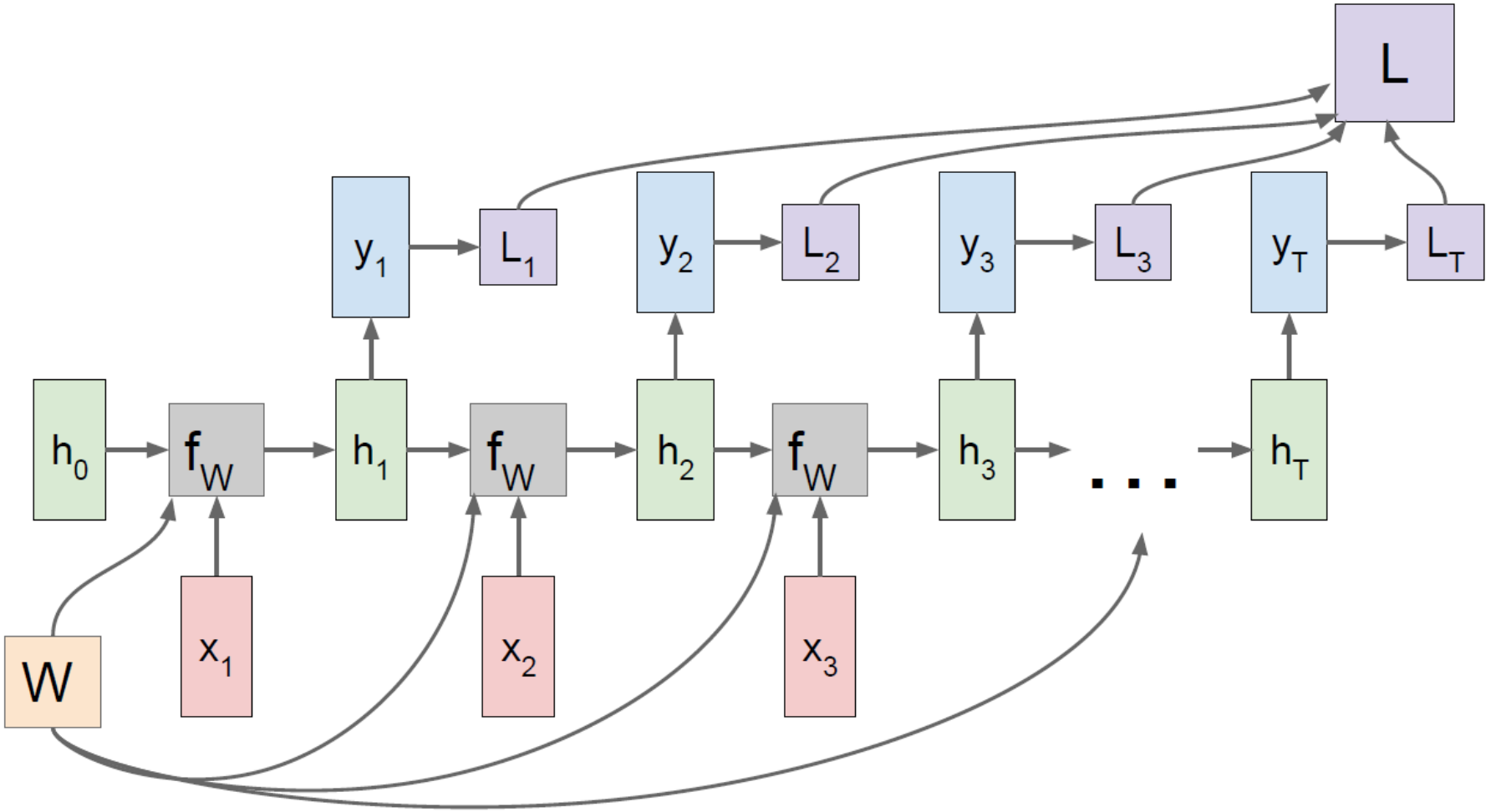
RNN: Computational Graph

Many to Many



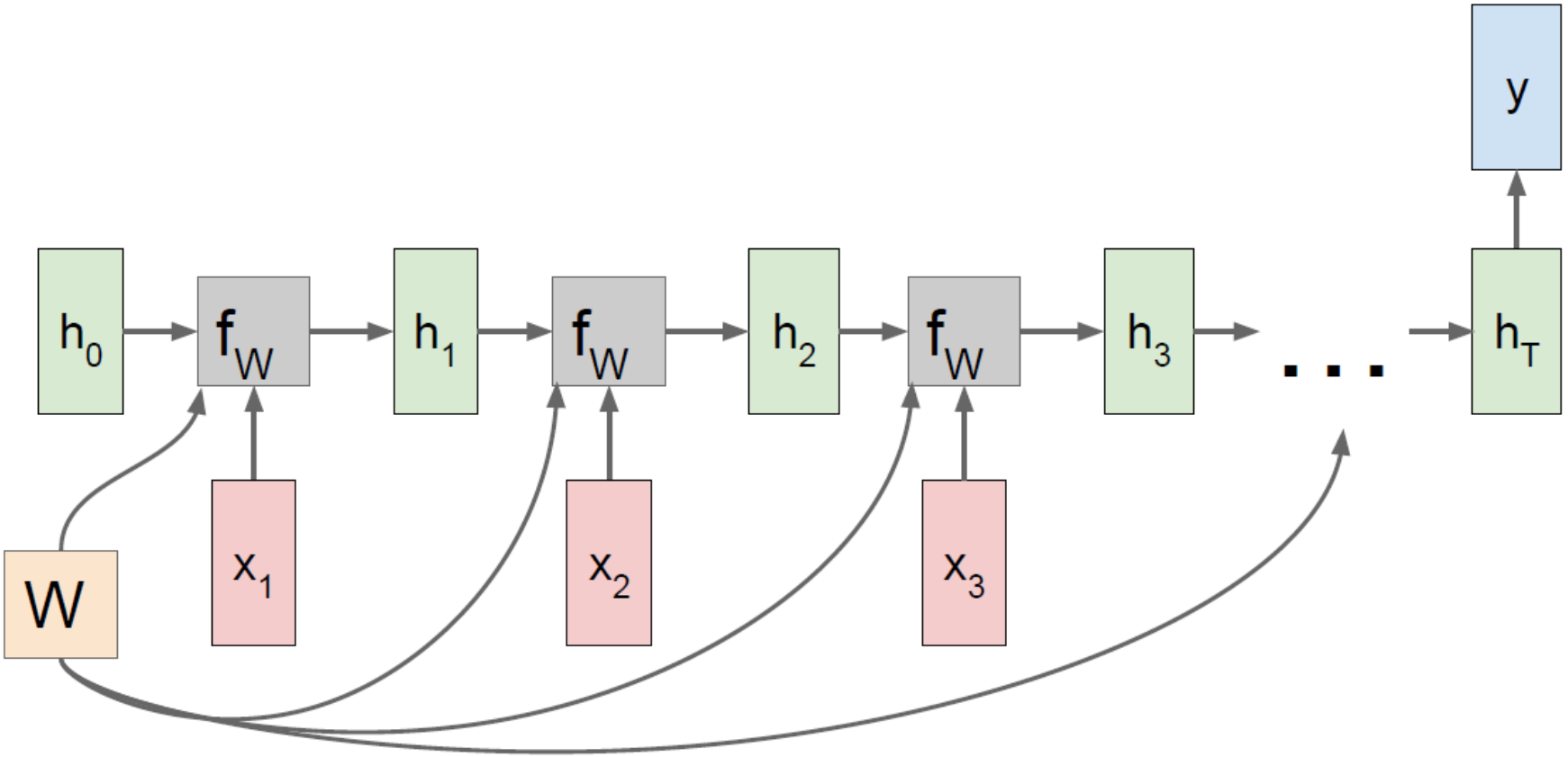
RNN: Computational Graph

Many to Many



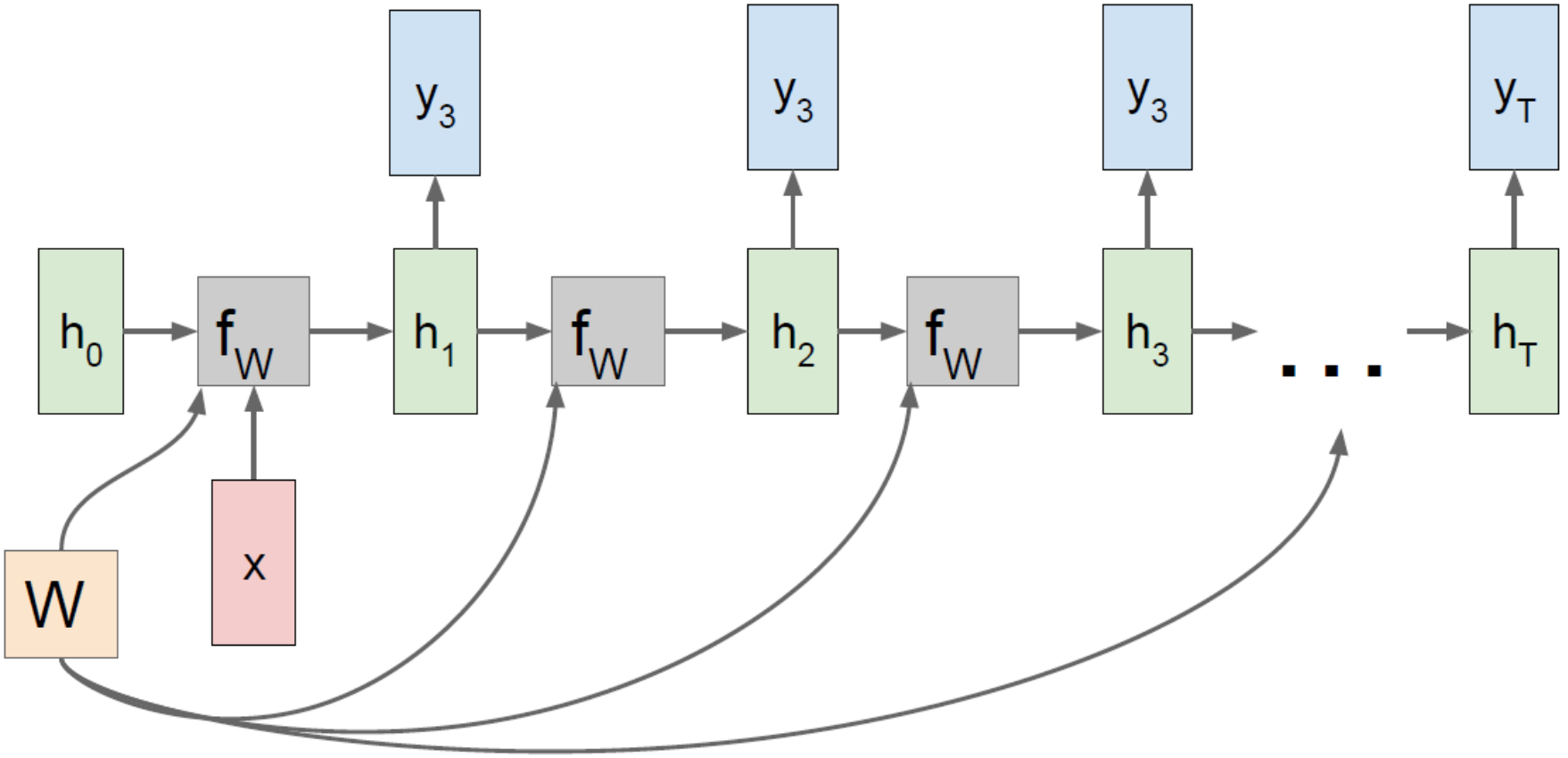
RNN: Computational Graph

Many to One

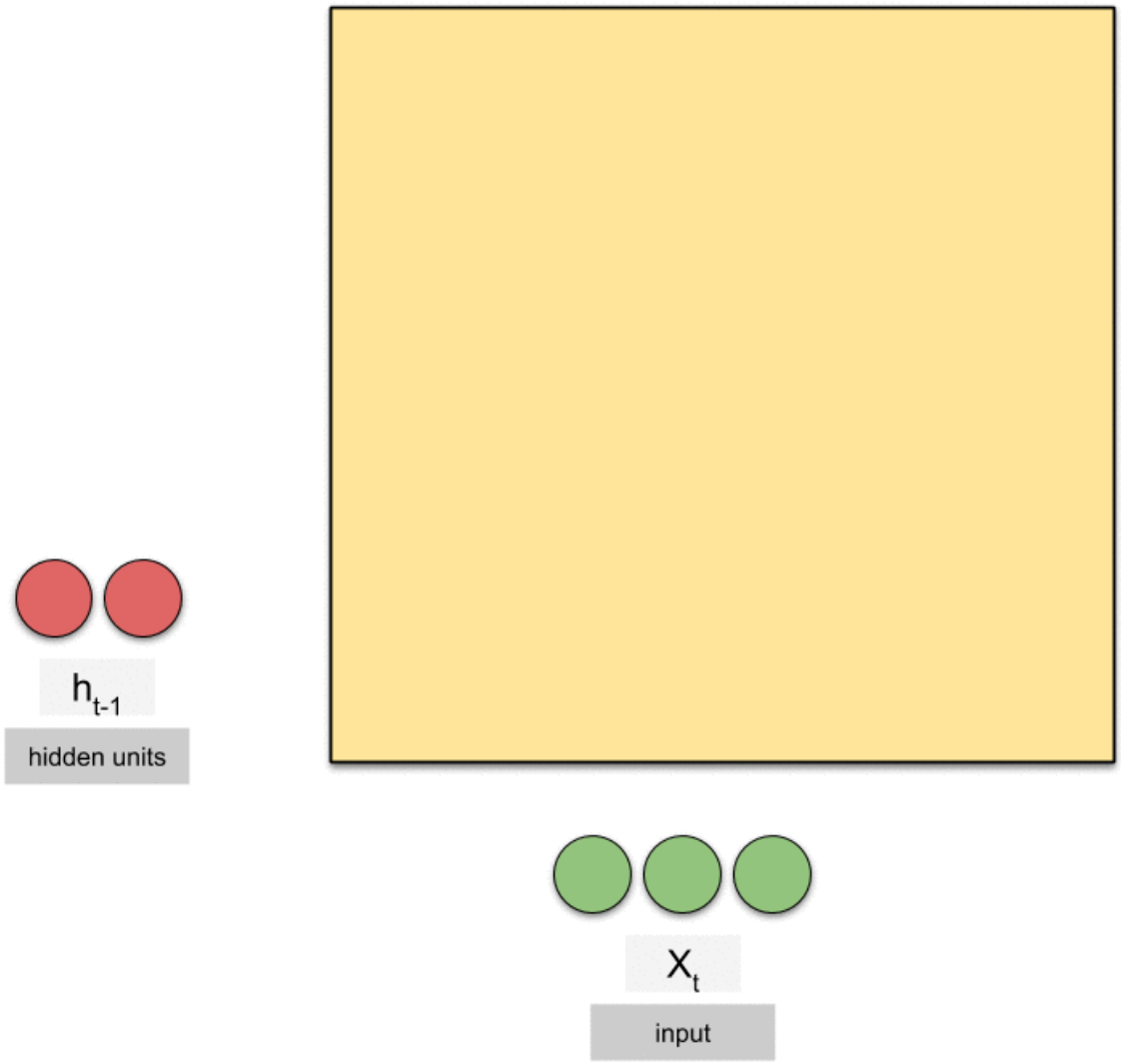


RNN: Computational Graph

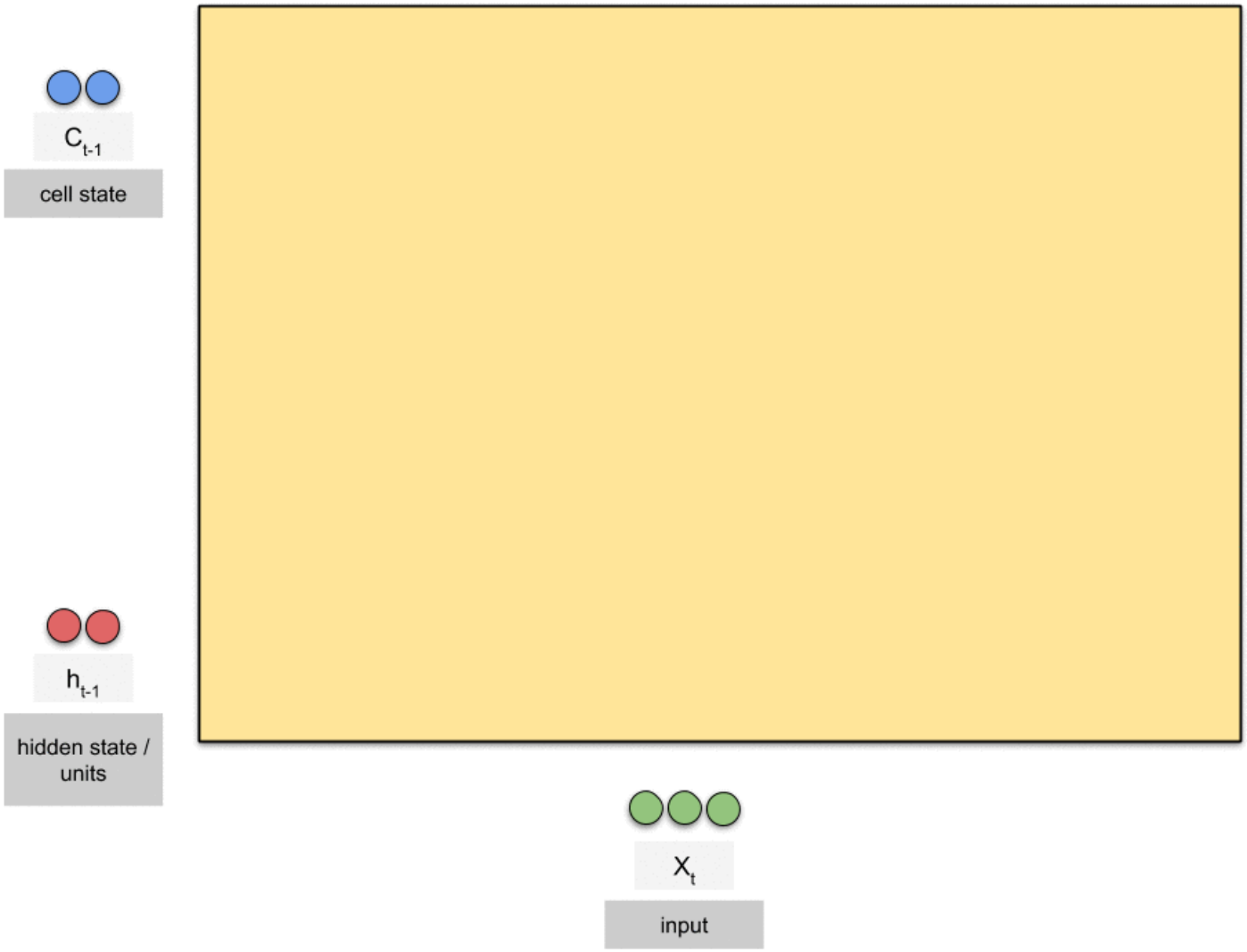
One to Many



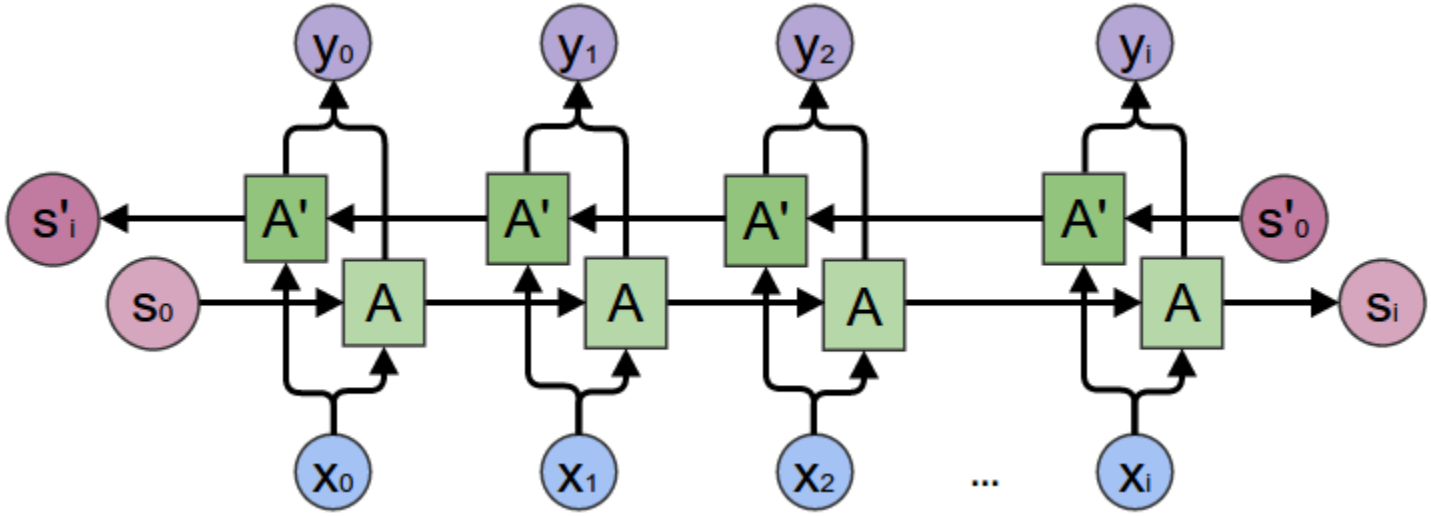
Vanilla RNN



LSTM



Bidirectional RNN



GRU