Deep Learning

Convolutional Neural Network Models

Dr. Mohamed Loey

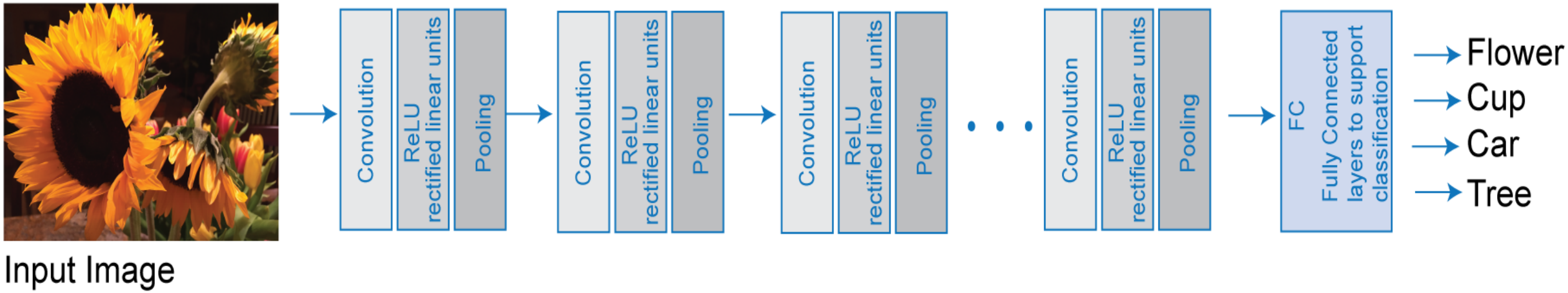
Lecturer, Faculty of Computers and Information
Benha University
Egypt

Table of Contents

Table of Contents

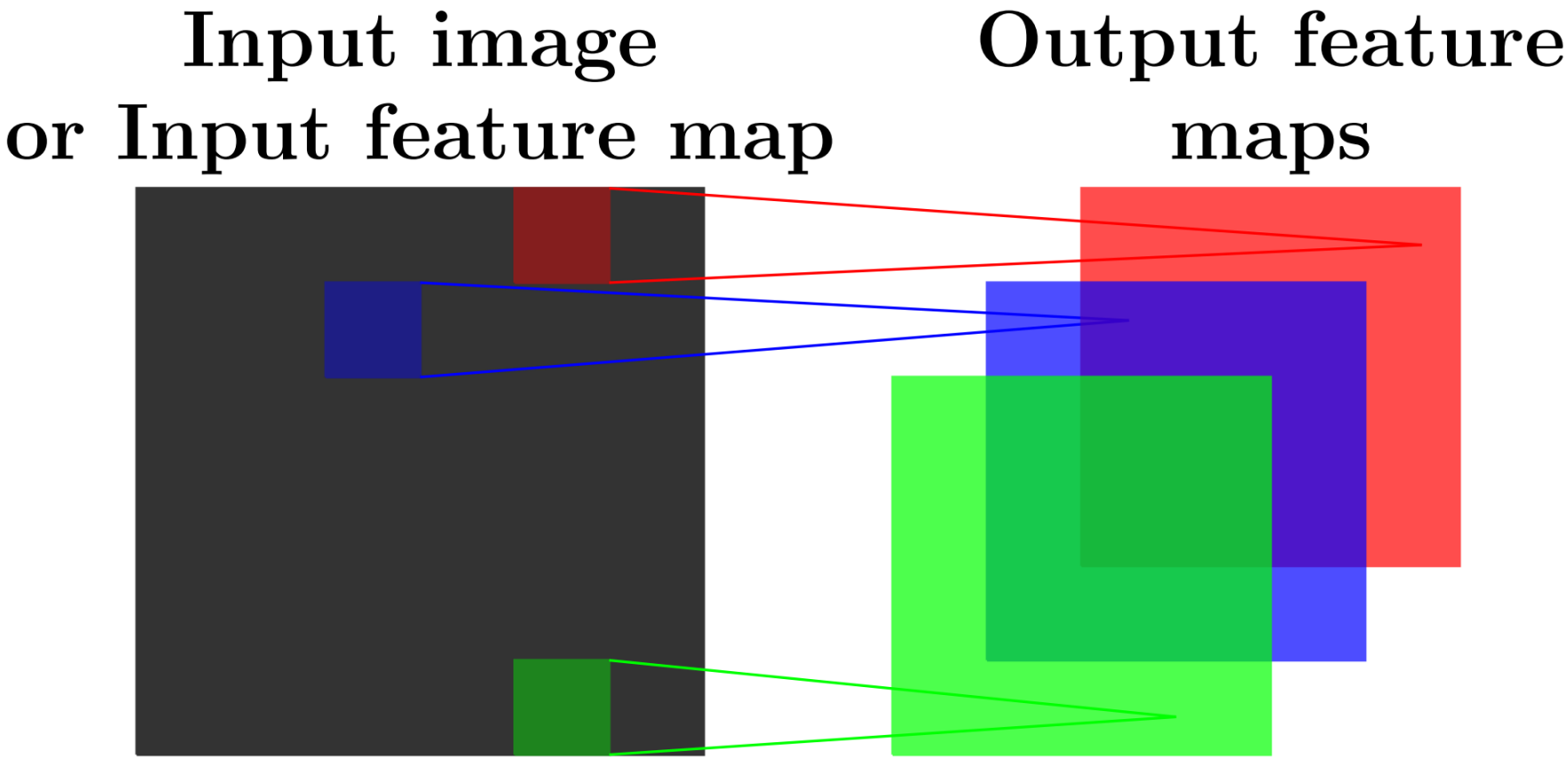
Convolutional Neural Network

- **Convolutional Neural Network (CNN)**is a multi-layer neural network
- **Convolutional Neural Network** is comprised of one or more **convolutional layers** (often with a **pooling layers**) and then followed by one or more **fully connected layers**.



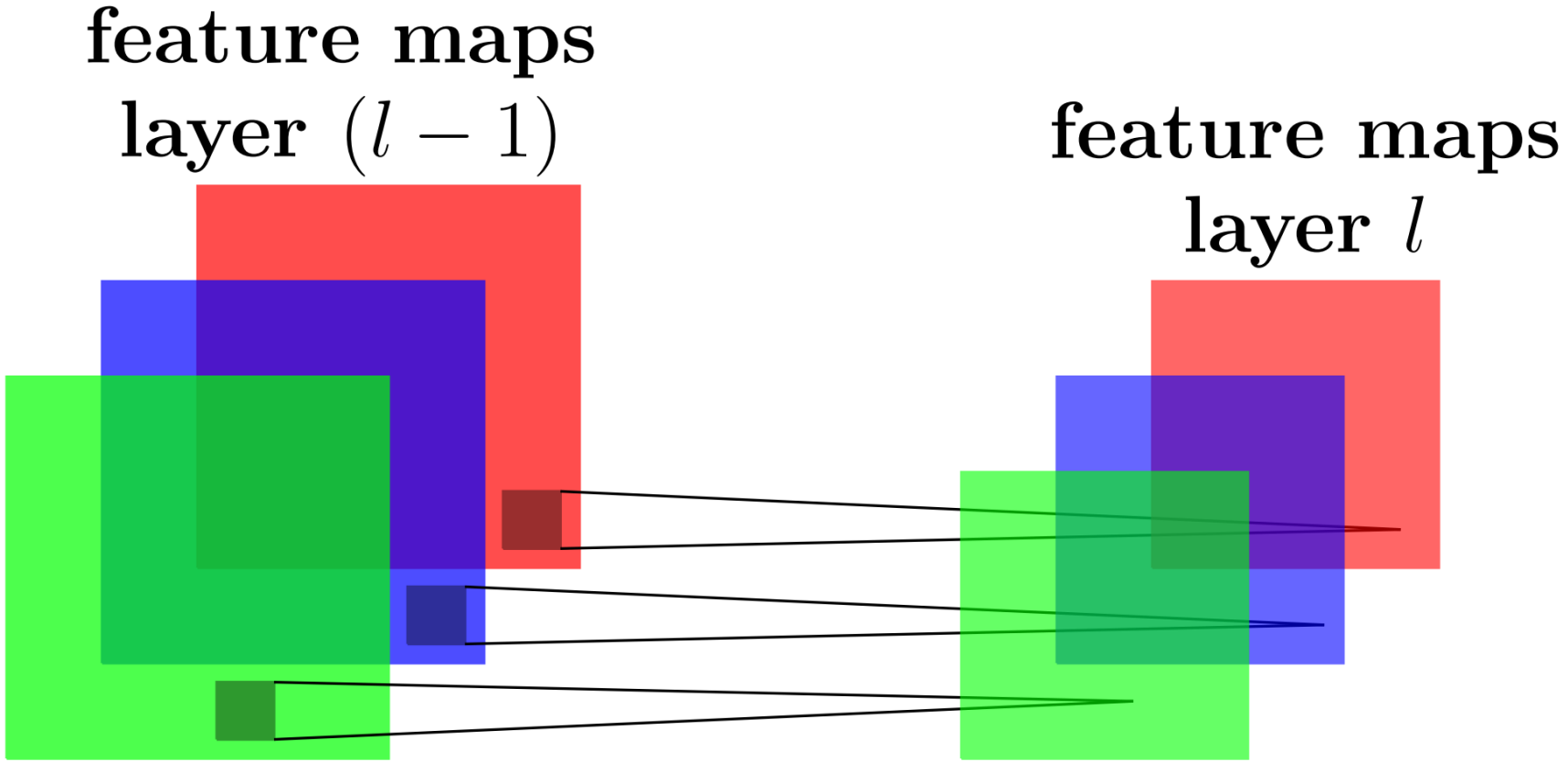
Convolutional Neural Network

- **Convolutional layer** acts as a feature extractor that extracts features of the inputs such as edges, corners , endpoints.



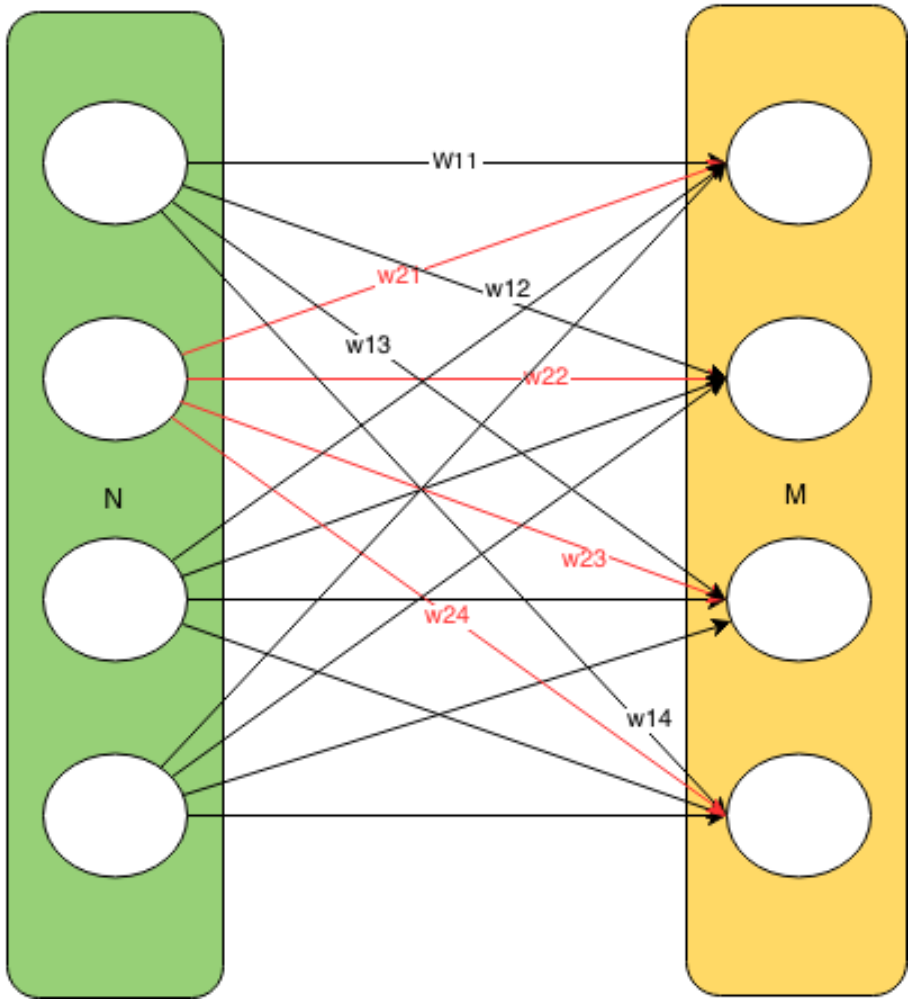
Convolutional Neural Network

- **Pooling layer** reduces the resolution of the image that reduce the precision of the translation (shift and distortion) effect.



Convolutional Neural Network

- **fully connected layer** have full connections to all activations in the previous layer.
- Fully connect layer act as classifier.



Convolutional Neural Network

- Output Image =

(((ImageSize+2*Padding)- KernalSize)/ Stride) +1

Convolutional Neural Network

- Conv 3x3 with stride=1,padding=0

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

6x6 Image

21	28	36	31
33	42	51	41
38	47	43	34
34	37	30	22

4x4

Convolutional Neural Network

- Conv 3x3 with stride=1,**padding=1**

0	0	0	0	0	0
0	1	5	3	9	0
0	4	4	7	5	0
0	6	7	5	6	0
0	6	5	3	1	0
0	0	0	0	0	0

4x4 Image

14	24	33	24
27	41	42	21
33	32	35	27
24	32	27	15

4x4

Convolutional Neural Network

- Conv 3x3 with **stride=2**,padding=0

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

7x7 Image

14	36	23
38	43	23
21	18	12

3x3

Convolutional Neural Network

- Conv 3x3 with **stride=2,padding=1**

0	0	0	0	0	0	0
0	1	2	1	1	1	0
0	1	1	5	3	9	0
0	2	4	4	7	5	0
0	3	6	7	5	6	0
0	1	6	5	3	1	0
0	0	0	0	0	0	0

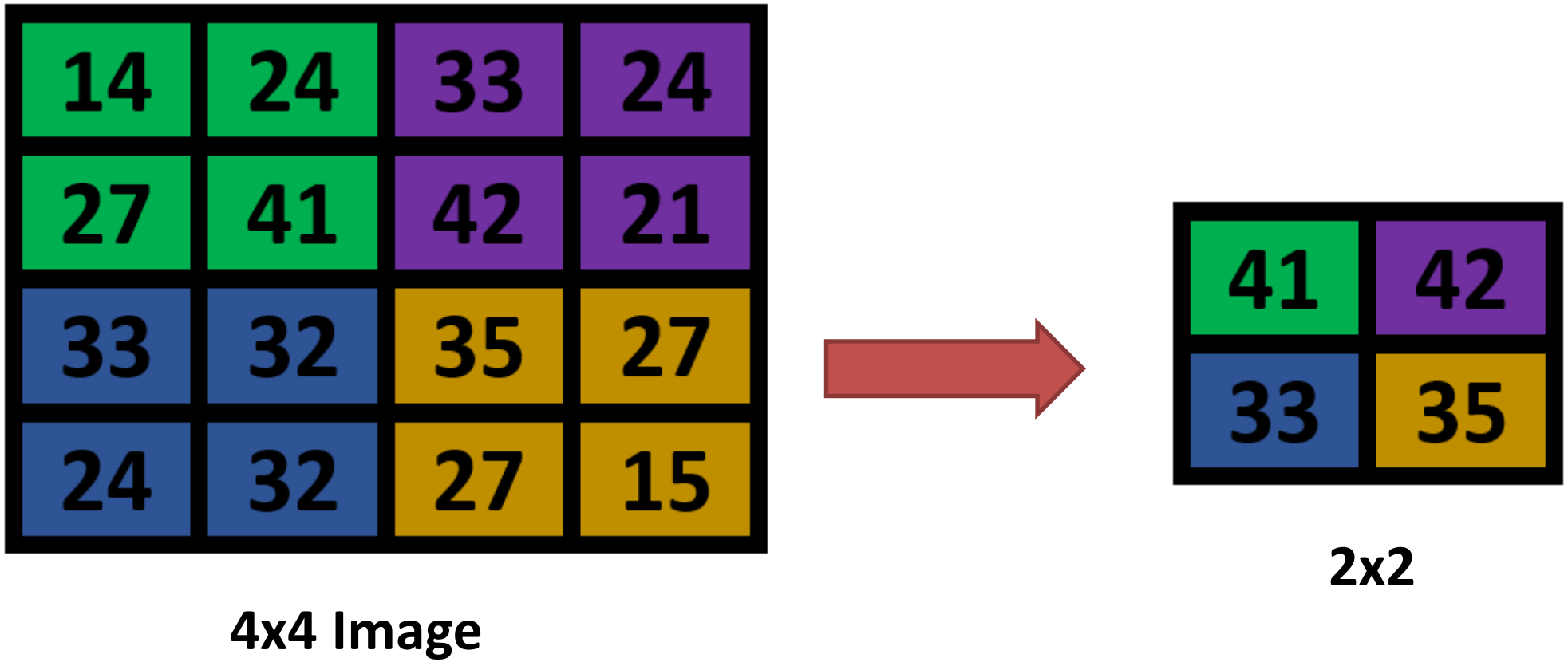
5x5 Image

5	13	14
17	42	35
16	32	15

3x3

Convolutional Neural Network

- MaxPooling 2x2 with **stride=2**



Convolutional Neural Network

- MaxPooling 3x3 with **stride=2**

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

7x7 Image

5	9	9
7	7	6
6	5	2

3x3

Table of Contents

ILSVRC

- **ImageNet Large Scale Visual Recognition Challenge** is image classification challenge to create model that can correctly classify an input image into 1,000 separate object categories.
- Models are trained on 1.2 million training images with another 50,000 images for validation and 150,000 images for testing

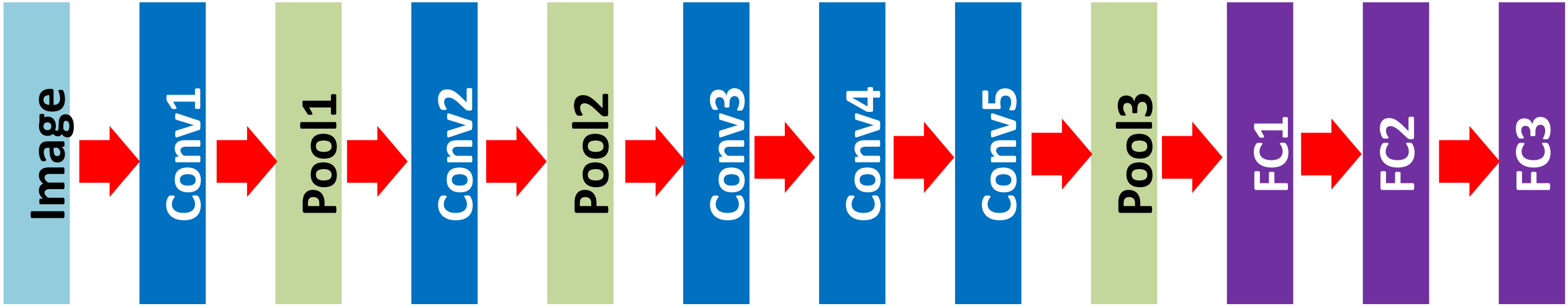
Table of Contents

AlexNet (2012)

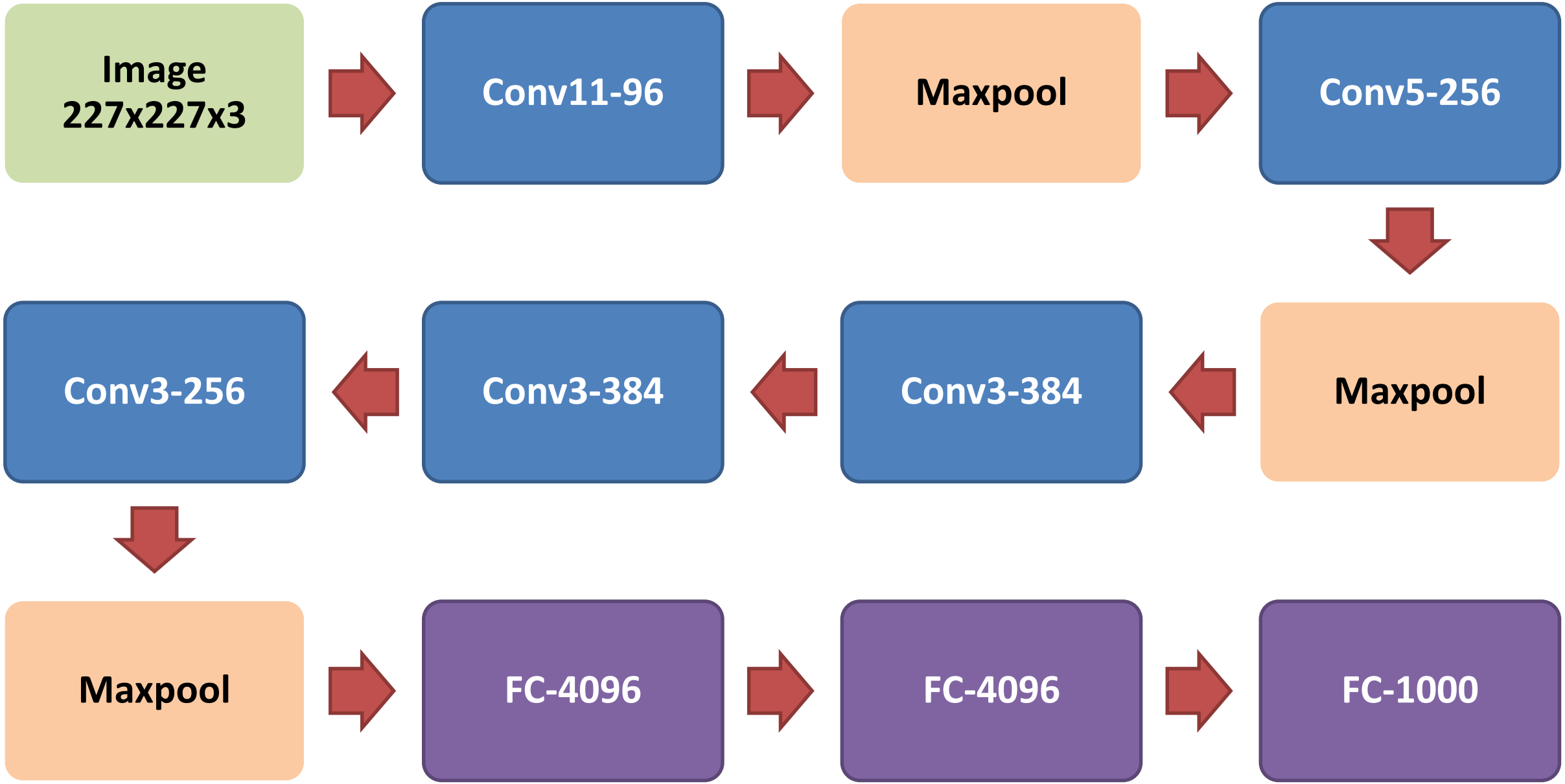
- **AlexNet** achieve on ILSVRC 2012 competition **15.3%** Top-5 error rate compare to 26.2% achieved by the second best entry.
- **AlexNet** using batch stochastic gradient descent on training, with specific values for momentum and weight decay.
- **AlexNet** implement dropout layers in order to combat the problem of overfitting to the training data.

AlexNet (2012)

- **AlexNet** has 8 layers without count pooling layers.
- **AlexNet** use ReLU for the nonlinearity functions
- **AlexNet** trained on two GTX 580 GPUs for **five to six days**

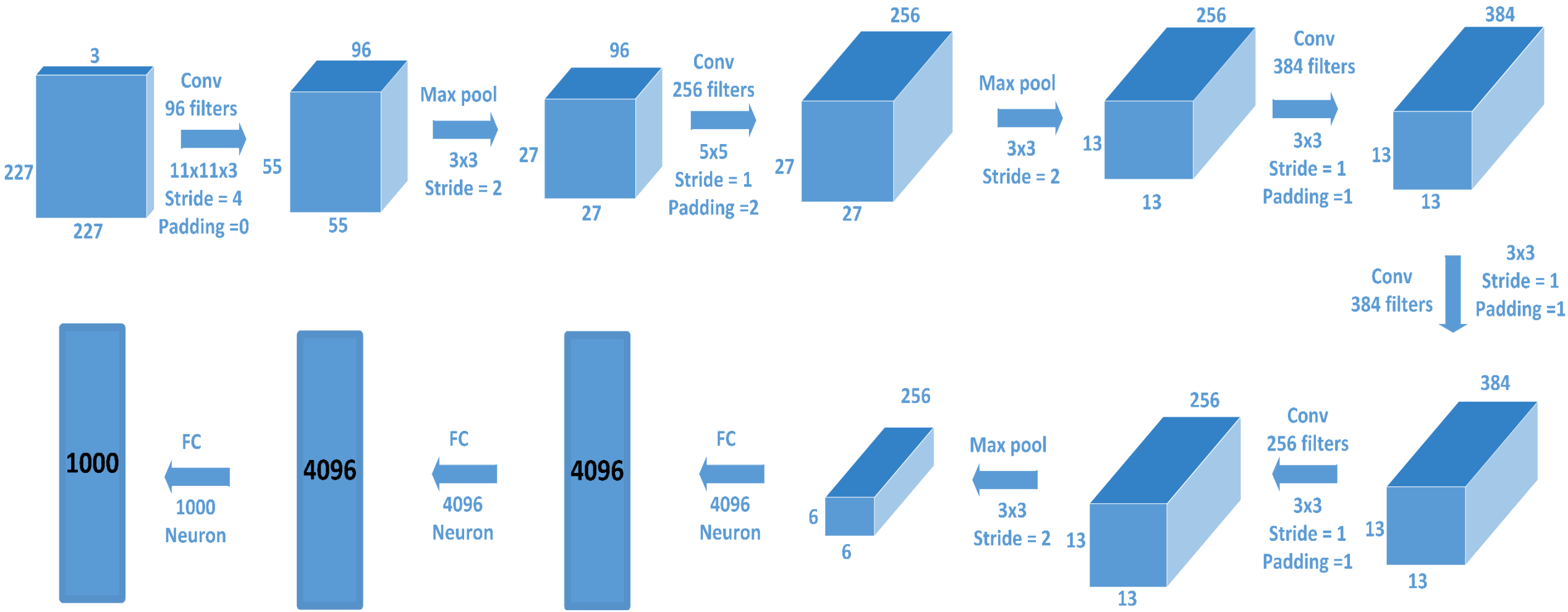


AlexNet (2012)



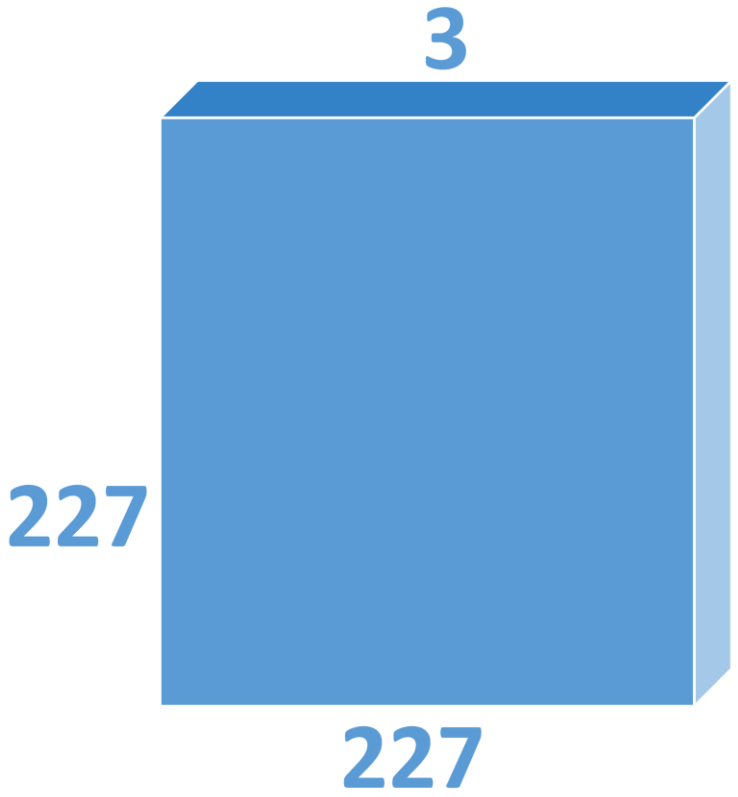
AlexNet (2012)

❑ AlexNet Model

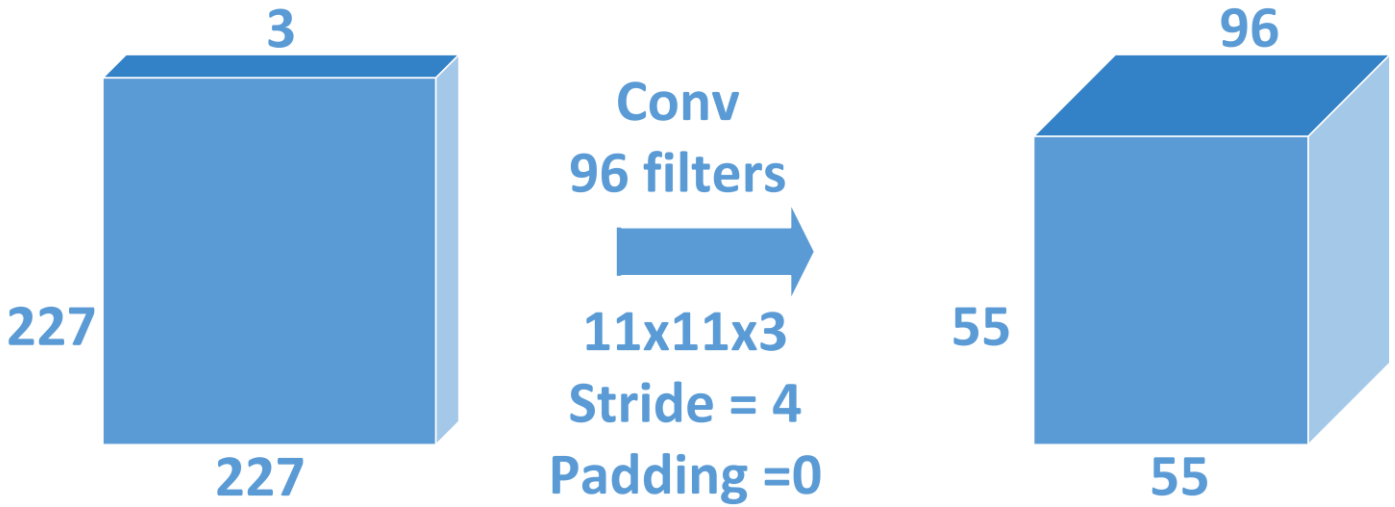


AlexNet (2012)

- **Layer 0: Input image**
- Size: 227 x 227 x 3
- Memory: 227 x 227 x 3

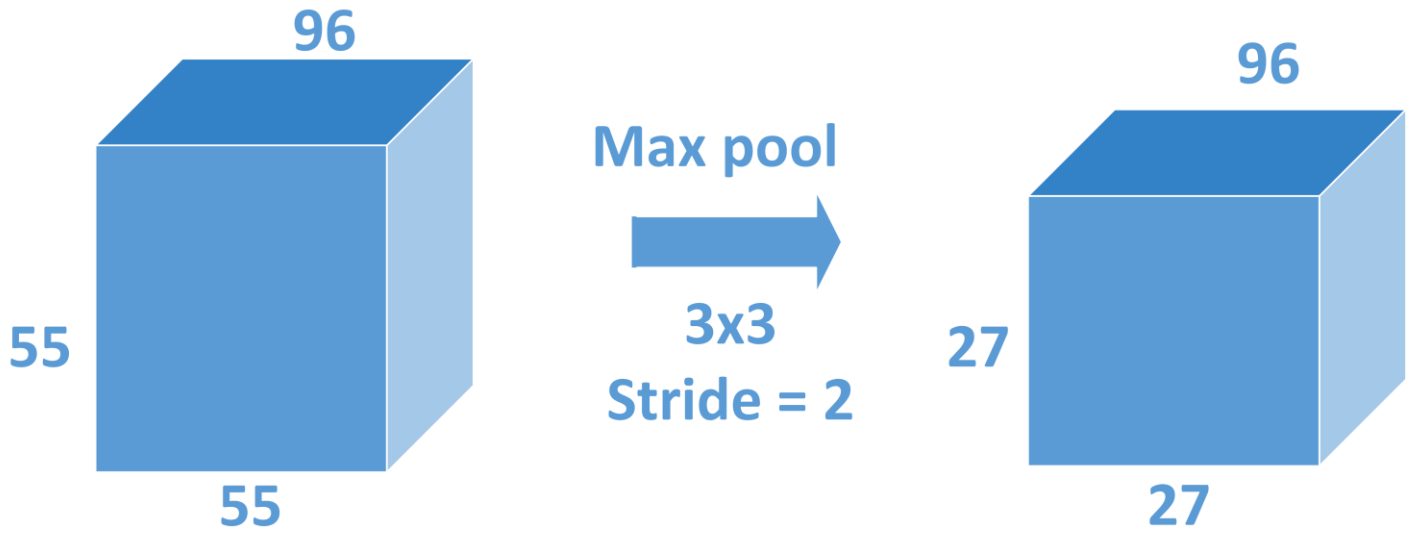


AlexNet (2012)



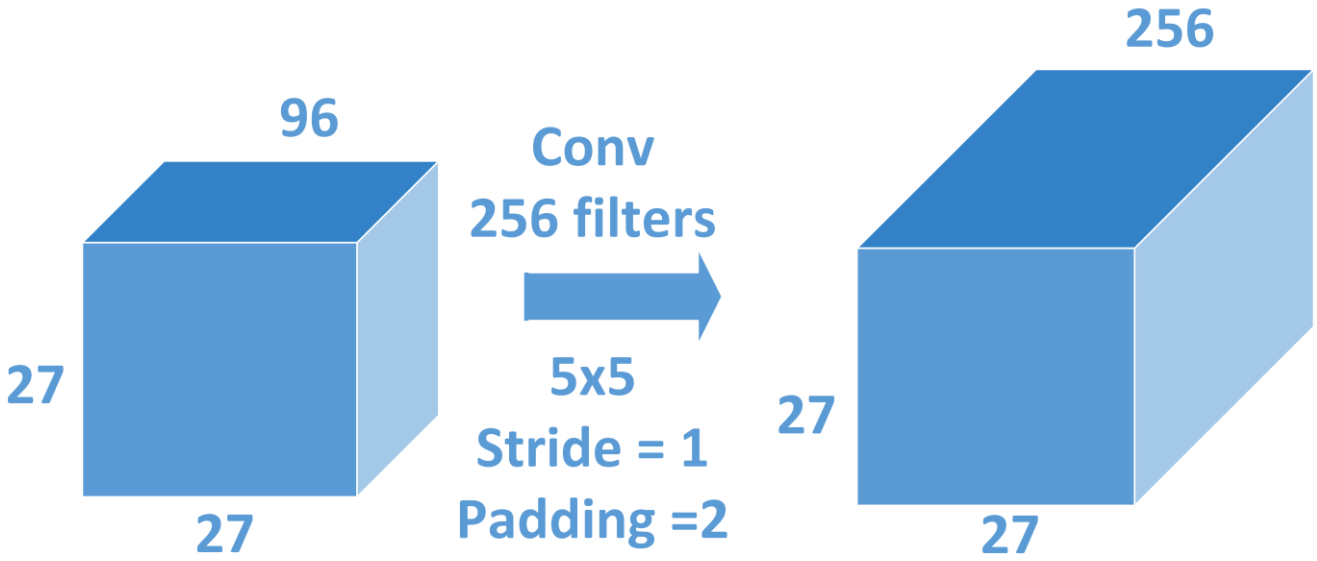
- **Layer 0: 227 x 227 x 3**
- **Layer 1: Convolution with 96 filters, size 11×11, stride 4, padding 0**
- **Outcome Size**= **55 x 55** x 96
- (227-11)/4 + 1 = **55** is size of outcome
- Memory: 55 x 55 x 96 x 3 (because of ReLU & LRN(Local Response Normalization))
- Weights (parameters) : 11 x 11 x 3 x 96

AlexNet (2012)



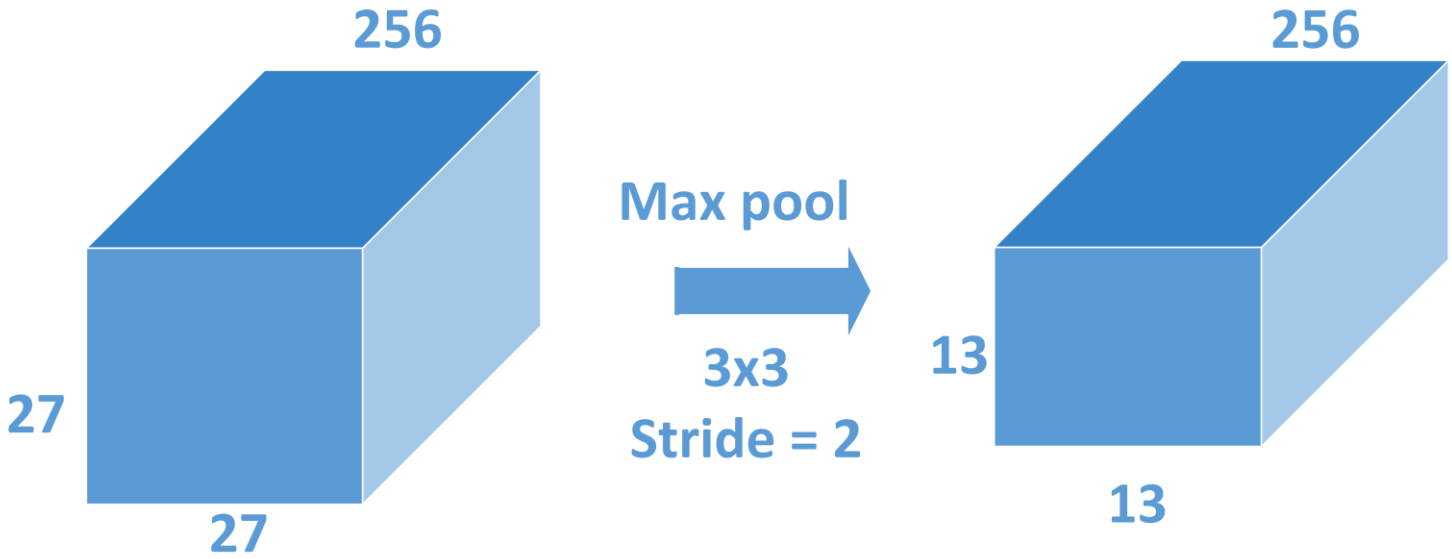
- **Layer 1: 55 x 55 x 96**
- **Layer 2: Max-Pooling with 3×3 filter, stride 2**
- **Outcome Size**= **27 x 27** x 96
- (55 – 3)/2 + 1 = **27** is size of outcome
- Memory: 27 x 27 x 96

AlexNet (2012)



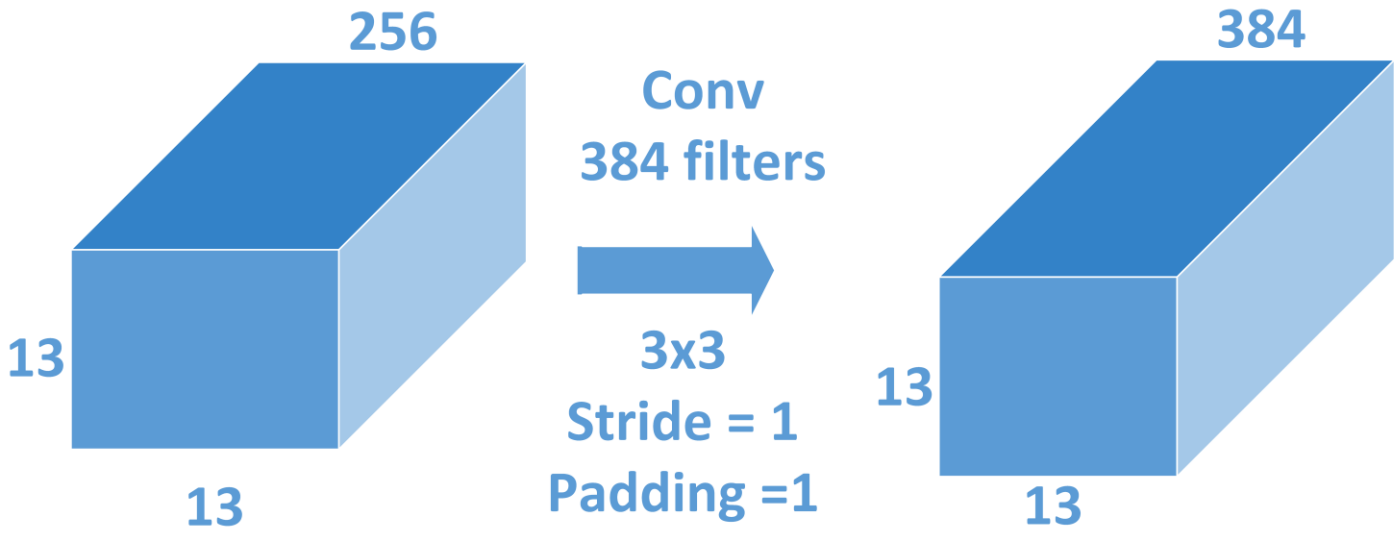
- **Layer 2: 27 x 27 x 96**
- **Layer 3: Convolution with 256 filters, size 5×5, stride 1, padding 2**
- **Outcome Size** = **27 x 27** x 256
- original size is restored because of padding
- Memory: 27 x 27 x 256 x 3 (because of ReLU and LRN)
- Weights: 5 x 5 x 96 x 256

AlexNet (2012)



- **Layer 3: 27 x 27 x 256**
- **Layer 4: Max-Pooling with 3×3 filter, stride 2**
- **Outcome Size** = **13 x 13** x 256
- (27 – 3)/2 + 1 = **13** is size of outcome
- Memory: 13 x 13 x 256

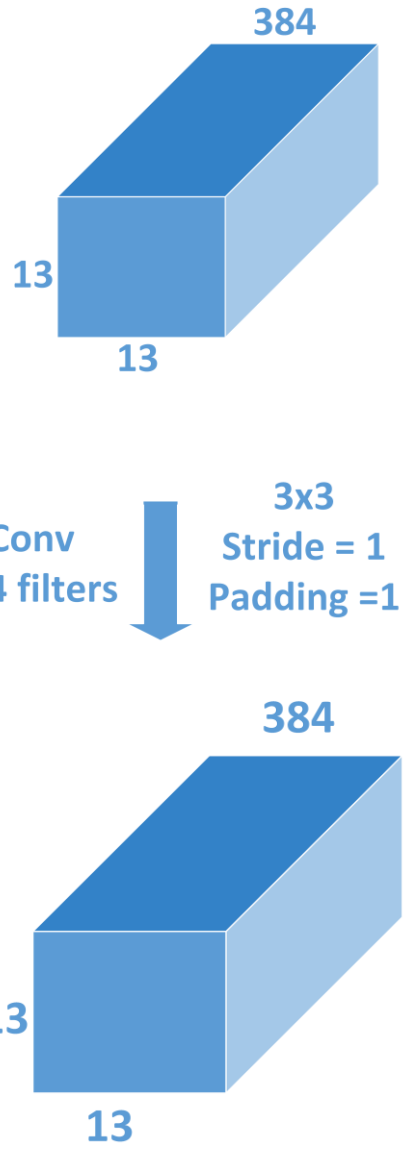
AlexNet (2012)



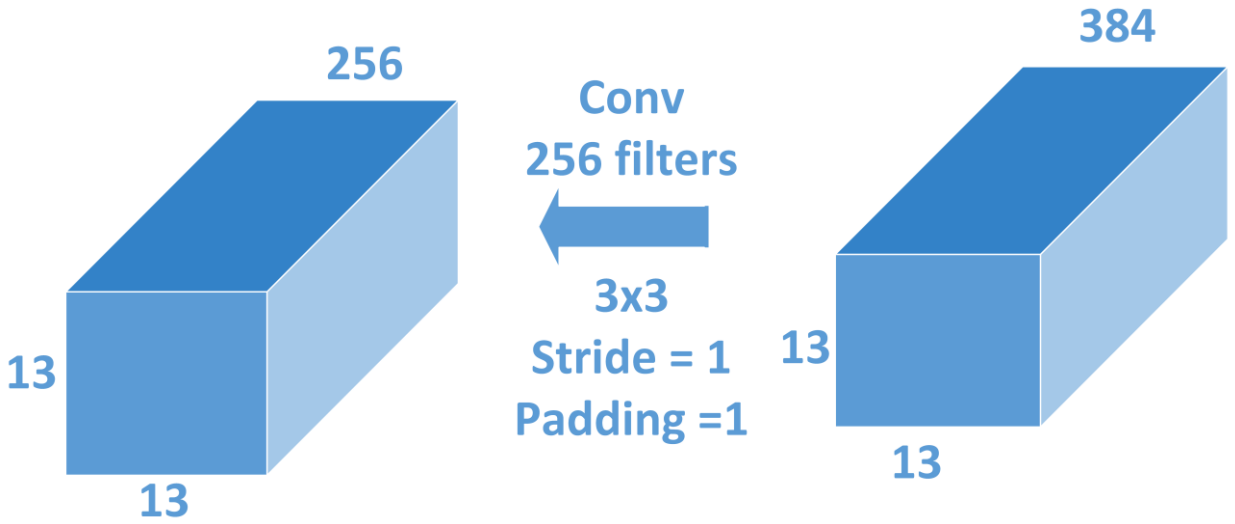
- **Layer 4: 13 x 13 x 256**
- **Layer 5: Convolution with 384 filters, size 3×3, stride 1, padding 1**
- **Outcome Size** = **13 x 13** x 384
- the original size is restored because of padding (13+2 -3)/1 +1 =13
- Memory: 13 x 13 x 384 x 2 (because of ReLU)
- Weights: 3 x 3 x 256 x 384

AlexNet (2012)

- **Layer 5: 13 x 13 x 384**
- **Layer 6: Convolution with 384 filters, size 3×3, stride 1, padding 1**
- **Outcome Size** = **13 x 13** x 384
- the original size is restored because of padding
- Memory: 13 x 13 x 384 x 2 (because of ReLU)
- Weights: 3 x 3 x 384 x 384

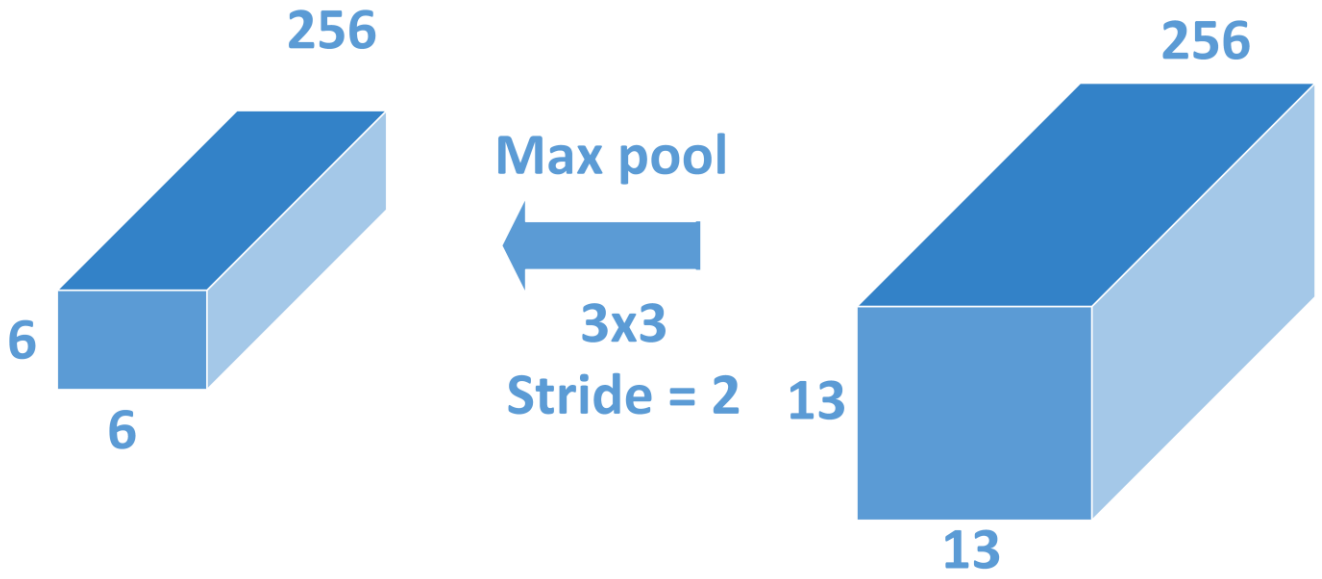


AlexNet (2012)



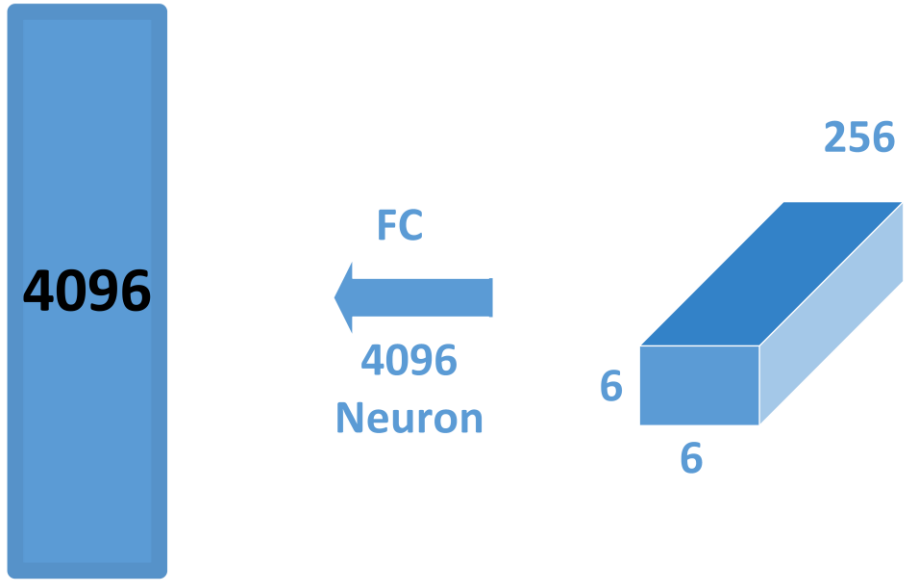
- **Layer 6: 13 x 13 x 384**
- **Layer 7: Convolution with 256 filters, size 3×3, stride 1, padding 1**
- **Outcome Size** = **13 x 13** x 256
- the original size is restored because of padding
- Memory: 13 x 13 x 256 x 2 (because of ReLU)
- Weights: 3 x 3 x 384 x 256

AlexNet (2012)



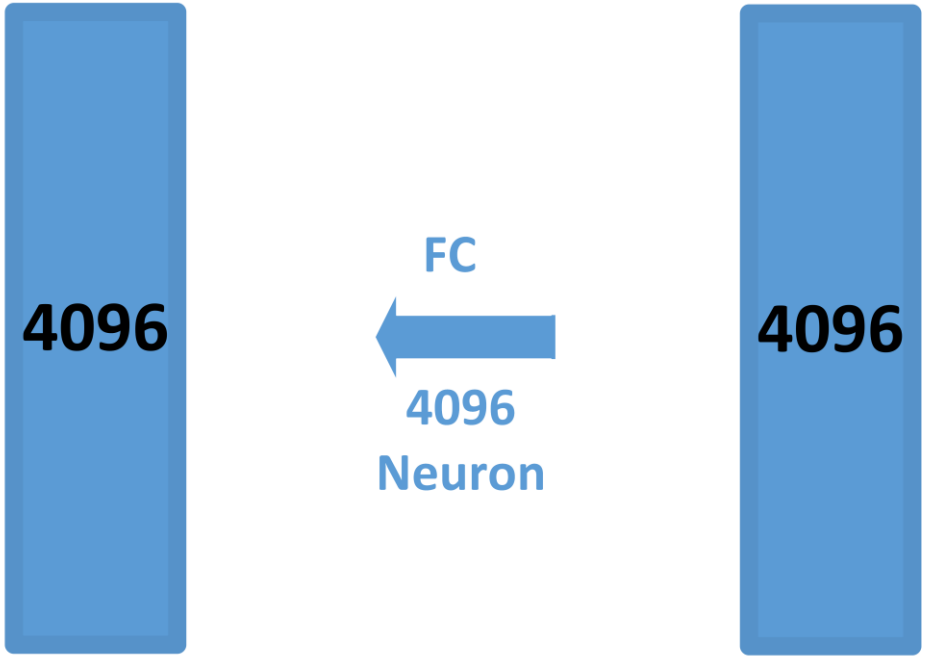
- **Layer 7: 13 x 13 x 256**
- **Layer 8: Max-Pooling with 3×3 filter, stride 2**
- **Outcome Size** = **6 x 6** x 256
- (13 – 3)/2 + 1 = 6 is size of outcome
- Memory: 6 x 6 x 256

AlexNet (2012)



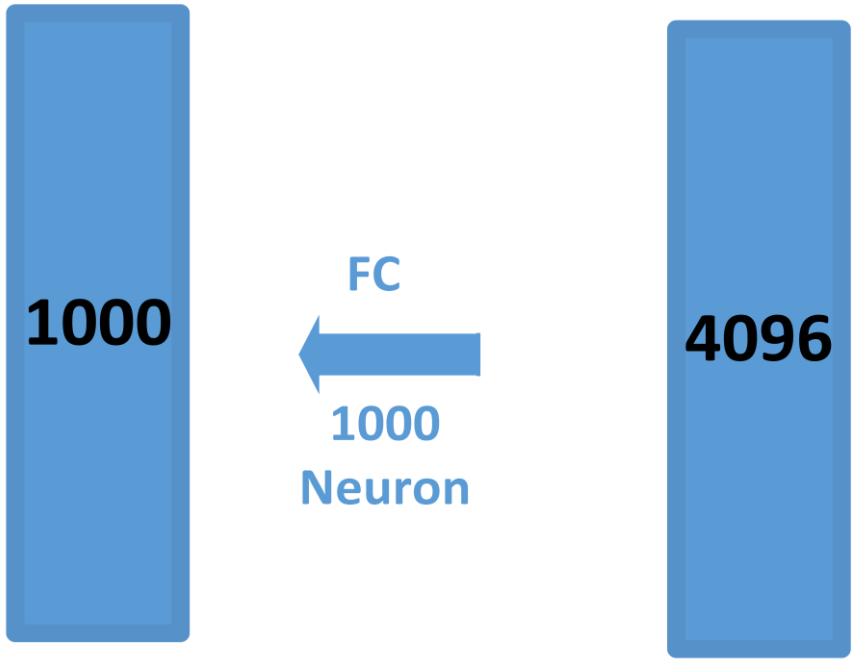
- **Layer 8: 6x6x256=9216 pixels are fed to FC**
- **Layer 9: Fully Connected with 4096 neuron**
- Memory: 4096 x 3 (because of ReLU and Dropout)
- Weights: 4096 x (6 x 6 x 256)

AlexNet (2012)



- **Layer 9: Fully Connected with 4096 neuron**
- **Layer 10: Fully Connected with 4096 neuron**
- Memory: 4096 x 3 (because of ReLU and Dropout)
- Weights: 4096 x 4096

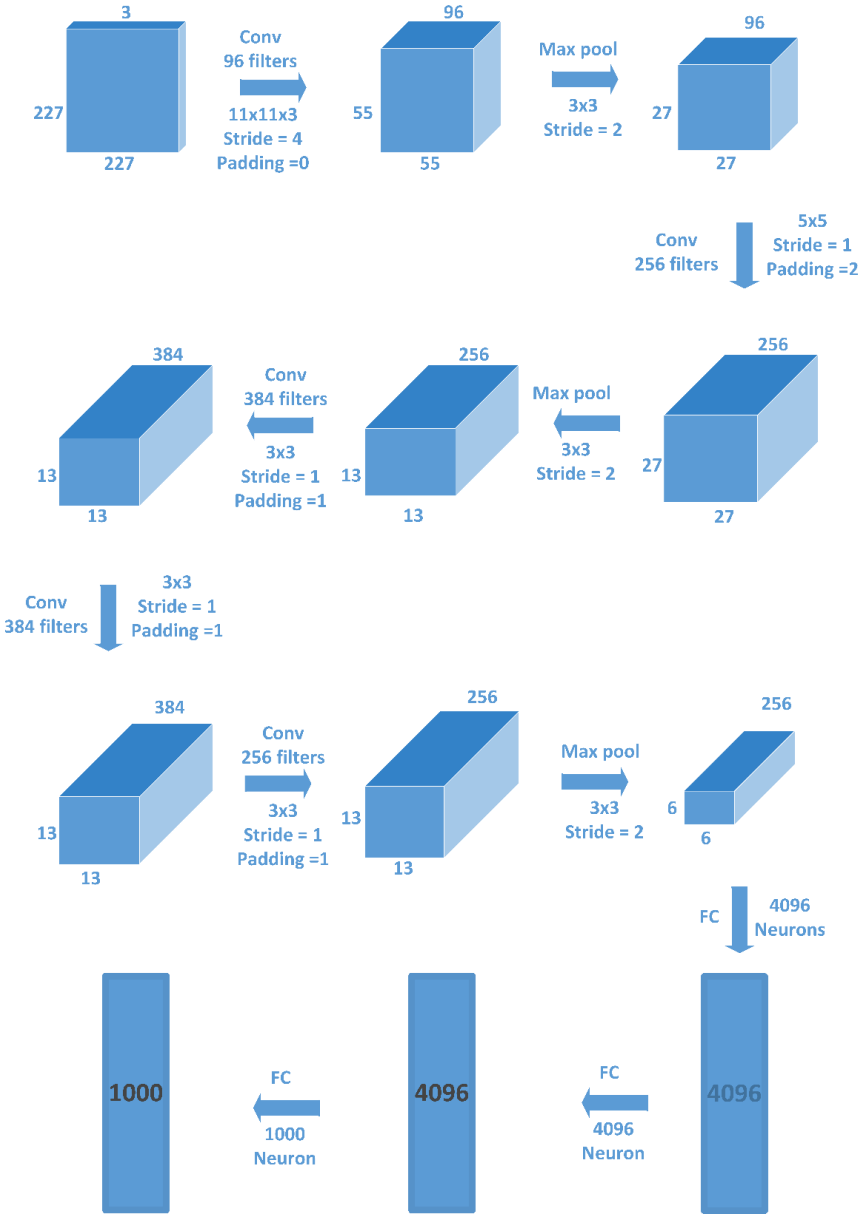
AlexNet (2012)



- **Layer 10: Fully Connected with 4096 neuron**
- **Layer 11: Fully Connected with 1000 neurons**
- Memory: 1000
- Weights: 4096 x 1000

AlexNet (2012)

- **Total (label and softmax not included)**
- Memory: 2.24 million
- Weights: 62.37 million



AlexNet (2012)

- first use of ReLU
- Alexnet used Norm layers
- Alexnet heavy used data augmentation
- Alexnet use dropout 0.5
- Alexnet batch size is 128
- Alexnet used SGD Momentum 0.9
- Alexnet used learning rate 1e-2, reduced by 10

AlexNet (2012)

[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

AlexNet (2012)

[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

AlexNet (2012)

- Implement **AlexNet** using **TFLearn**

```
from __future__ import division, print_function, absolute_import

import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.normalization import local_response_normalization
from tflearn.layers.estimator import regression

import tflearn.datasets.oxflower17 as oxflower17
X, Y = oxflower17.load_data(one_hot=True, resize_pics=(227, 227))
```

AlexNet (2012)

```
# Building 'AlexNet'
network = input_data(shape=[None, 227, 227, 3])
network = conv_2d(network, 96, 11, strides=4, activation='relu')
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = conv_2d(network, 256, 5, activation='relu')
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = conv_2d(network, 384, 3, activation='relu')
network = conv_2d(network, 384, 3, activation='relu')
network = conv_2d(network, 256, 3, activation='relu')
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = fully_connected(network, 4096, activation='tanh')
```

AlexNet (2012)

```
network = dropout(network, 0.5)
network = fully_connected(network, 4096, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 17, activation='softmax')
network = regression(network, optimizer='momentum',
                     loss='categorical_crossentropy',
                     learning_rate=0.001)


# Training
model = tflearn.DNN(network, checkpoint_path='model_alexnet',
                    max_checkpoints=1, tensorboard_verbose=2)
model.fit(X, Y, n_epoch=1000, validation_set=0.1, shuffle=True,
          show_metric=True, batch_size=64, snapshot_step=200,
          snapshot_epoch=False, run_id='alexnet_oxflowers17')
```

Table of Contents

ZFNet (2013)

- **ZFNet** the winner of the competition ILSVRC 2013 with **14.8%** Top-5 error rate
- **ZFNet** built by Matthew Zeiler and Rob Fergus

- **ZFNet** has the same global architecture as Alexnet, that is to say 5 convolutional layers, two fully connected layers and an output softmax one. The differences are for example better sized convolutional kernels.

ZFNet (2013)

- **ZFNet** used filters of size 7x7 and a decreased stride value, instead of using 11x11 sized filters in the first layer (which is what AlexNet implemented).

- **ZFNet** trained on a GTX 580 GPU for **twelve days.**
- Developed a visualization technique named Deconvolutional Network "deconvnet" because it maps features to pixels.

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

Table of Contents

VGGNet (2014)

- **Keep it deep. Keep it simple.**
- **VGGNet** the runner up of the competition ILSVRC 2014 with **7.3%** Top-5 error rate.
- **VGGNet** use of only 3x3 sized filters is quite different from AlexNet's 11x11 filters in the first layer and ZFNet's 7x7 filters.
- two 3x3 conv layers have an effective receptive field of 5x5
- Three 3x3 conv layers have an effective receptive field of 7x7
- **VGGNet** trained on 4 Nvidia Titan Black GPUs for **two to three weeks**

VGGNet (2014)

- Interesting to notice that the number of filters doubles after each maxpool layer. This reinforces the idea of shrinking spatial dimensions, but growing depth.
- VGGNet used scale jittering as one data augmentation technique during training
- VGGNet used ReLU layers after each conv layer and trained with batch gradient descent

VGGNet (2014)

Image

Conv Conv Pool

Conv Conv Pool

Conv Conv Conv Pool

Conv Conv Conv Pool

Conv Conv Conv Pool

FC FC FC

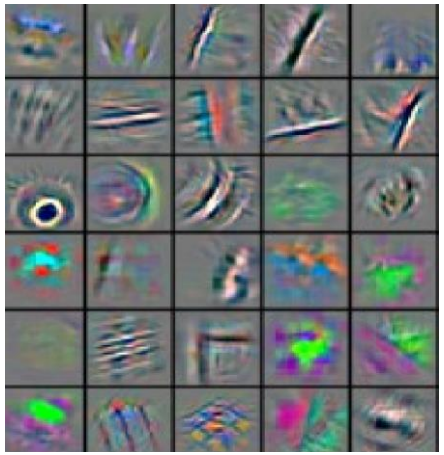
Image

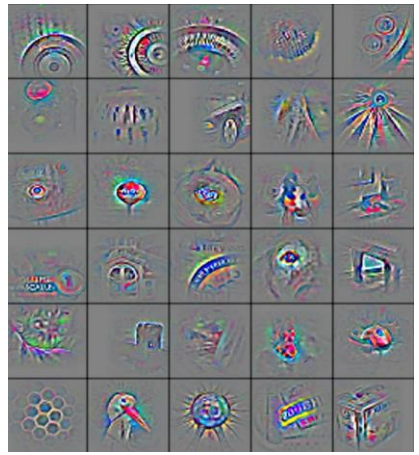
Low Level Feature

Mid Level Feature

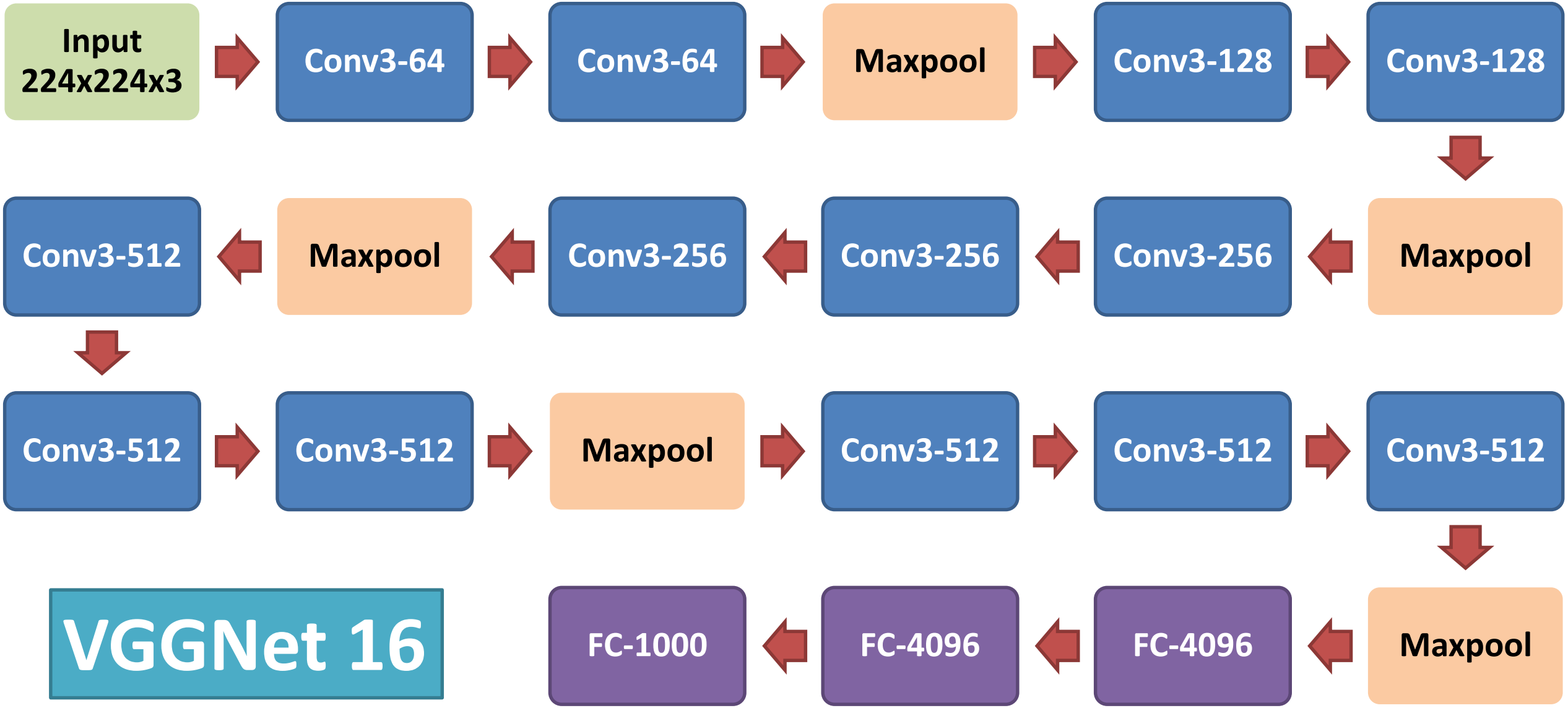
High Level Feature

Classifier

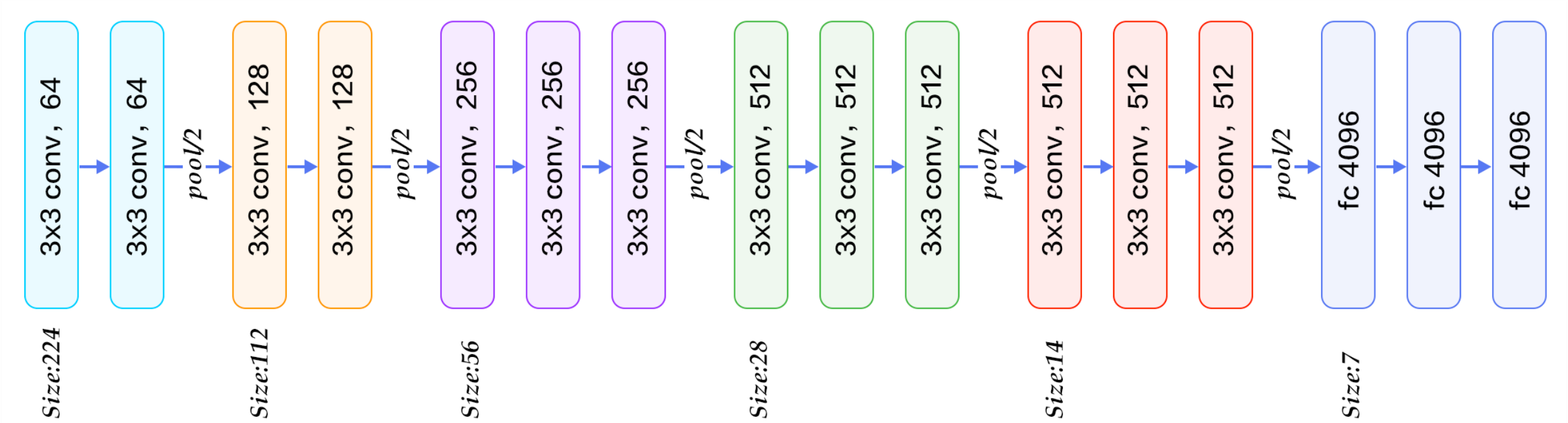




VGGNet (2014)

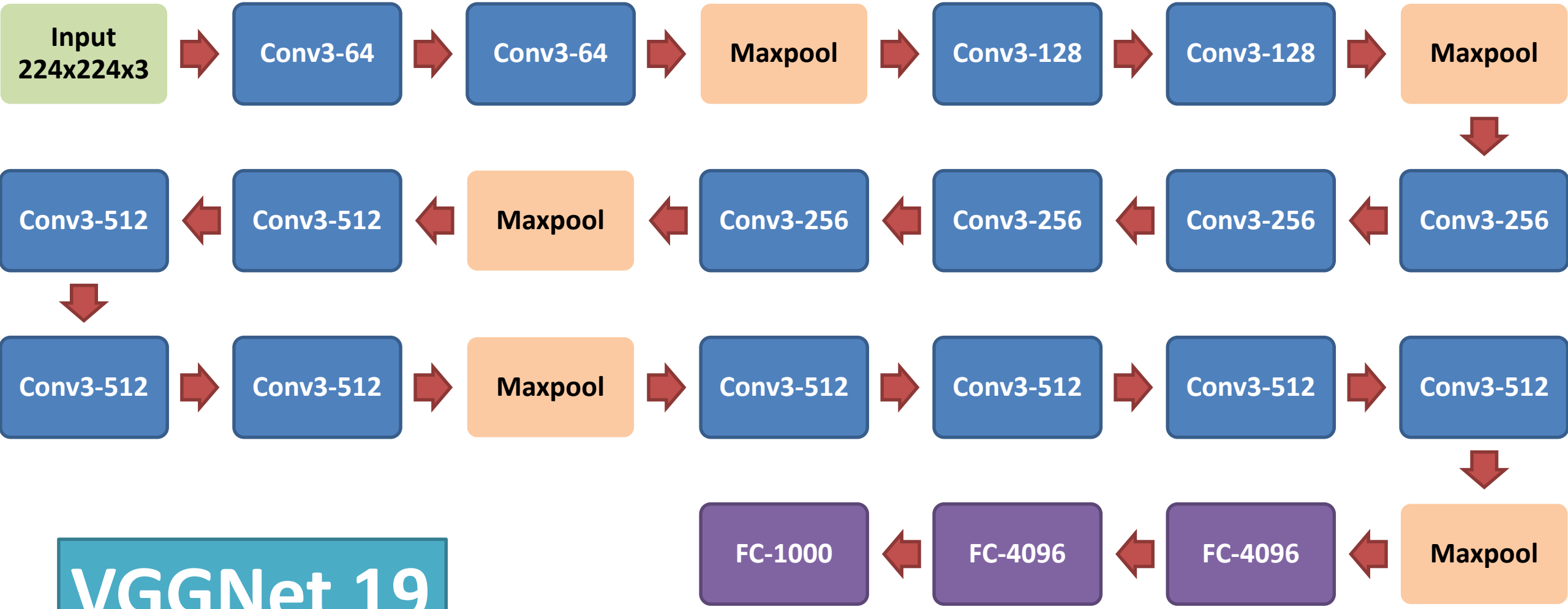

Input 224x224x3
Conv3-64
Conv3-64
Maxpool
Conv3-128
Conv3-128
Maxpool
Conv3-256
Conv3-256
Conv3-256
Maxpool
Conv3-512
Conv3-512
Conv3-512
Maxpool
Conv3-512
Conv3-512
Conv3-512
Maxpool
FC-4096
FC-4096
FC-1000
VGGNet 16

VGGNet (2014)



VGGNet 16

VGGNet (2014)



VGGNet 19

VGGNet (2014)

- Implement **VGGNet16** using **TFLearn**

```
from __future__ import division, print_function, absolute_import

import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.estimator import regression

# Data loading and preprocessing
import tflearn.datasets.oxflower17 as oxflower17
X, Y = oxflower17.load_data(one_hot=True)

# Building 'VGG Network'
network = input_data(shape=[None, 224, 224, 3])
```

VGGNet (2014)

```
network = conv_2d(network, 64, 3, activation='relu')
network = conv_2d(network, 64, 3, activation='relu')
network = max_pool_2d(network, 2, strides=2)

network = conv_2d(network, 128, 3, activation='relu')
network = conv_2d(network, 128, 3, activation='relu')
network = max_pool_2d(network, 2, strides=2)

network = conv_2d(network, 256, 3, activation='relu')
network = conv_2d(network, 256, 3, activation='relu')
network = conv_2d(network, 256, 3, activation='relu')
network = max_pool_2d(network, 2, strides=2)
```

VGGNet (2014)

```
network = conv_2d(network, 512, 3, activation='relu')
network = conv_2d(network, 512, 3, activation='relu')
network = conv_2d(network, 512, 3, activation='relu')
network = max_pool_2d(network, 2, strides=2)

network = conv_2d(network, 512, 3, activation='relu')
network = conv_2d(network, 512, 3, activation='relu')
network = conv_2d(network, 512, 3, activation='relu')
network = max_pool_2d(network, 2, strides=2)

network = fully_connected(network, 4096, activation='relu')
network = dropout(network, 0.5)
network = fully_connected(network, 4096, activation='relu')
network = dropout(network, 0.5)
```

VGGNet (2014)

```
network = fully_connected(network, 17, activation='softmax')

network = regression(network, optimizer='rmsprop',
                     loss='categorical_crossentropy',
                     learning_rate=0.0001)

# Training
model = tflearn.DNN(network, checkpoint_path='model_vgg',
                    max_checkpoints=1, tensorboard_verbose=0)
model.fit(X, Y, n_epoch=500, shuffle=True,
          show_metric=True, batch_size=32, snapshot_step=500,
          snapshot_epoch=False, run_id='vgg_oxflowers17')
```

Table of Contents

GoogLeNet (2014)

- **GoogleNet** is the winner of the competition ILSVRC 2014 with **6.7%** Top-5 error rate.
- **GoogleNet** Trained on "a few high-end GPUs **with in a week**"
- **GoogleNet** uses 12x fewer parameters than AlexNet
- **GoogleNet** use an average pool instead of fully connected layers, to go from a 7x7x1024 volume to a 1x1x1024 volume. This saves a huge number of parameters.

GoogLeNet (2014)

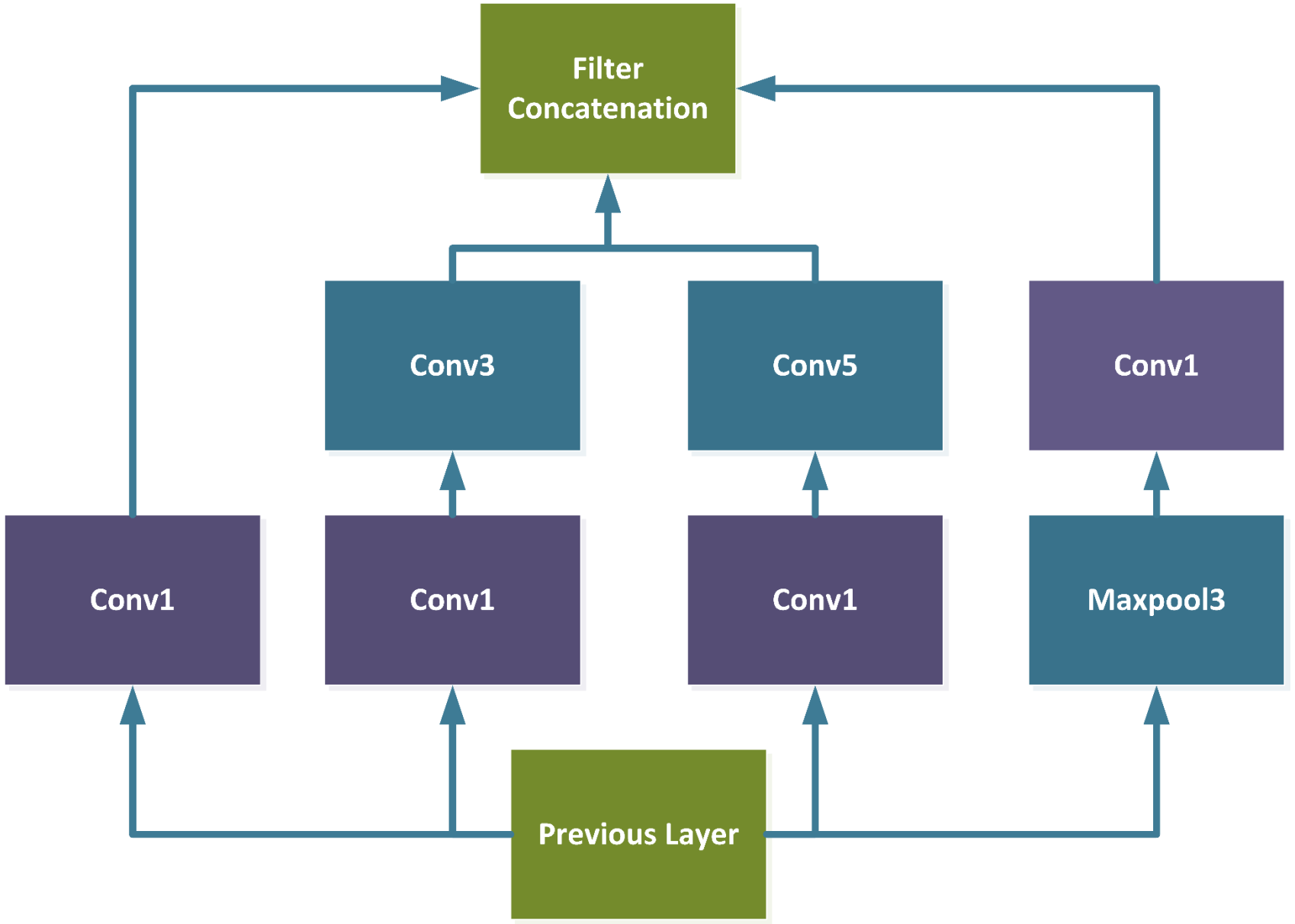
- **GoogleNet** used 9 Inception modules in the whole architecture
- This 1x1 convolutions (bottleneck convolutions) allow to control/reduce the depth dimension which greatly reduces the number of used parameters due to removal of redundancy of correlated filters.
- **GoogleNet** has 22 Layers deep network

GoogLeNet (2014)

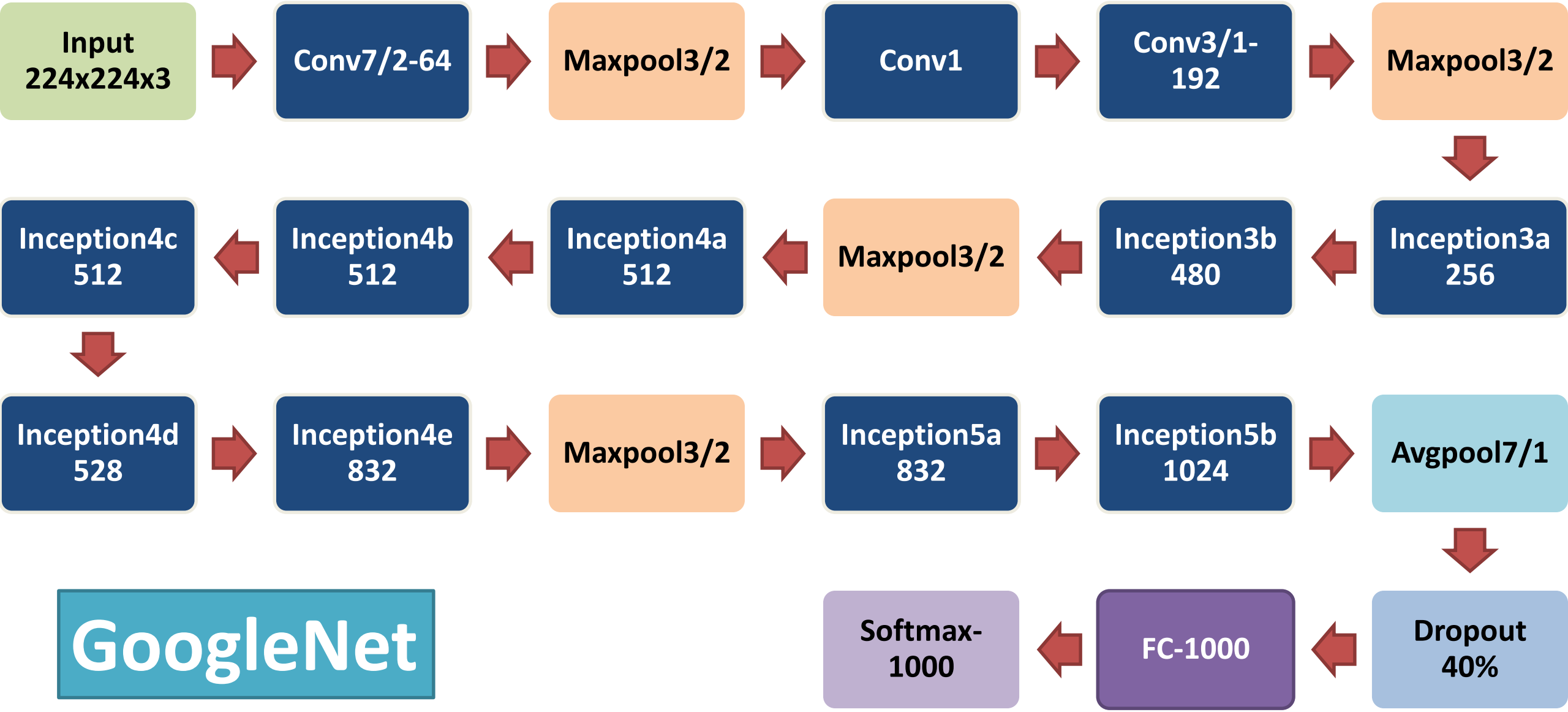
- **GoogleNet** use an average pool instead of using FC-Layer, to go from a 7x7x1024 volume to a 1x1x1024 volume. This saves a huge number of parameters.
- **GoogleNet** use inexpensive Conv1 to compute reduction before the expensive Conv3 and Conv5
- Conv1 follow by Relu to reduce overfitting

GoogLeNet (2014)

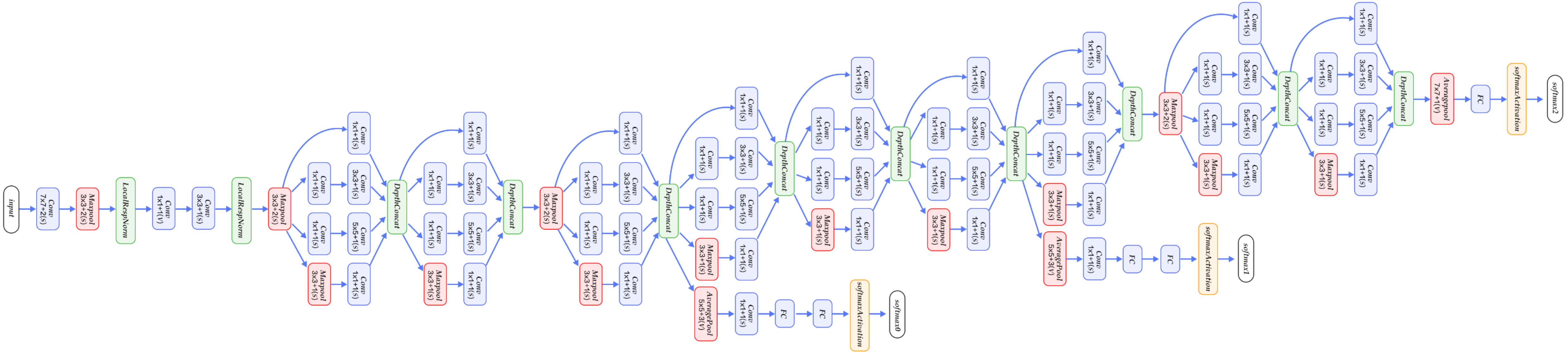
- Inception module



GoogLeNet (2014)



GoogLeNet (2014)



GoogLeNet (2014)

Type	Size/ Stride	Output	Depth	Conv1	# Conv3	Conv3	# Conv5	Conv5	Pool	Param	Ops
Conv	7x7/2	112x112x64	**1**	**-**	-	**-**	-	**-**	**-**	2.7K	34M
Maxpool	3x3/2	56x56x64	**0**	**-**	-	**-**	-	**-**	**-**	-	-
Conv	3x3/1	56x56x192	**2**	**-**	64	**192**	-	**-**	**-**	112K	360M
Maxpool	3x3/2	28x28x192	**0**	**-**	-	**-**	-	**-**	**-**	-	-
Inception 3a	-	28x28x**256**	**2**	**64**	96	**128**	16	**32**	**32**	159K	128M
Inception 3b	-	28x28x**480**	**2**	**128**	128	**192**	32	**96**	**64**	380K	304M
Maxpool	3x3/2	14x14x480	**0**	**-**	-	**-**	-	**-**	**-**	-	-
Inception 4a	-	14x14x**512**	**2**	**192**	96	**208**	16	**48**	**64**	364K	73M
Inception 4b	-	14x14x**512**	**2**	**160**	112	**224**	24	**64**	**64**	437K	88M
Inception 4c	-	14x14x**512**	**2**	**128**	128	**256**	24	**64**	**64**	463K	100M
Inception 4d	-	14x14x**528**	**2**	**112**	144	**288**	32	**64**	**64**	580K	119M

GoogLeNet (2014)

Type	Size/ Stride	Output	Depth	Conv1	# Conv3	Conv3	# Conv5	Conv5	Pool	Param	Ops
Inception 4e	-	14x14x**832**	**2**	**256**	160	**320**	32	**128**	**128**	840K	170M
Maxpool	3x3/2	7x7x832	**0**	-	-	-	-	-	-	-	-
Inception 5a	-	7x7x**832**	**2**	**256**	160	**320**	32	**128**	**128**	1072K	54M
Inception 5b	-	7x7x**1024**	**2**	**384**	192	**384**	48	**128**	**128**	1388K	71M
Avgpool	7x7/1	1x1x1024	**0**	-	-	-	-	-	-	-	-
Dropout .4	-	1x1x1024	**0**	-	-	-	-	-	-	-	-
Linear	-	1x1x1024	**1**	-	-	-	-	-	-	1000K	1M
Softmax	-	1x1x1024	**0**	-	-	-	-	-	-	-	-

Total Layers	**22**

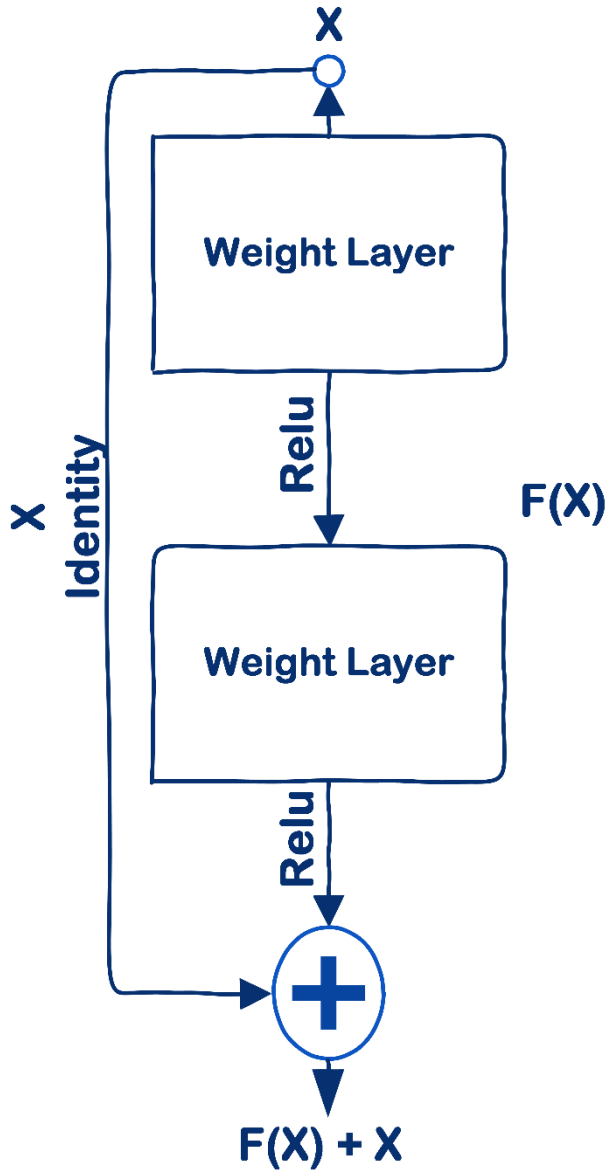
Table of Contents

ResNet (2015)

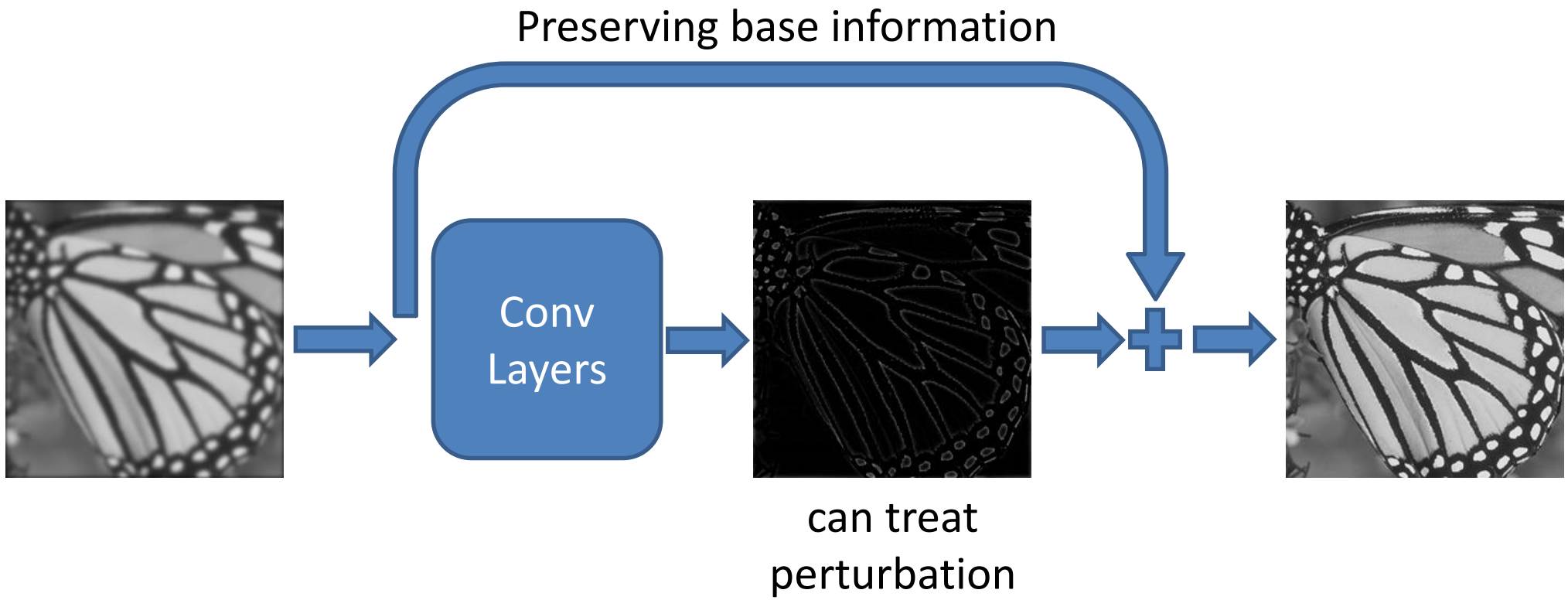
- **ResNet** the winner of the competition ILSVRC 2015 with **3.6%** Top-5 error rate.
- **ResNet** mainly inspired by the philosophy of VGGNet.
- **ResNet** proposed a residual learning approach to ease the difficulty of training deeper networks. Based on the design ideas of Batch Normalization (BN), small convolutional kernels.
- **ResNet** is a new 152 layer network architecture.
- **ResNet** Trained on an 8 GPU machine for **two to three weeks**

ResNet (2015)

- Residual network
- Keys:
  - ✓ No max pooling
  - ✓ No hidden fc
  - ✓ No dropout
  - ✓ Basic design (VGG-style)
  - ✓ All 3x3 conv (almost)
  - ✓ Batch normalization

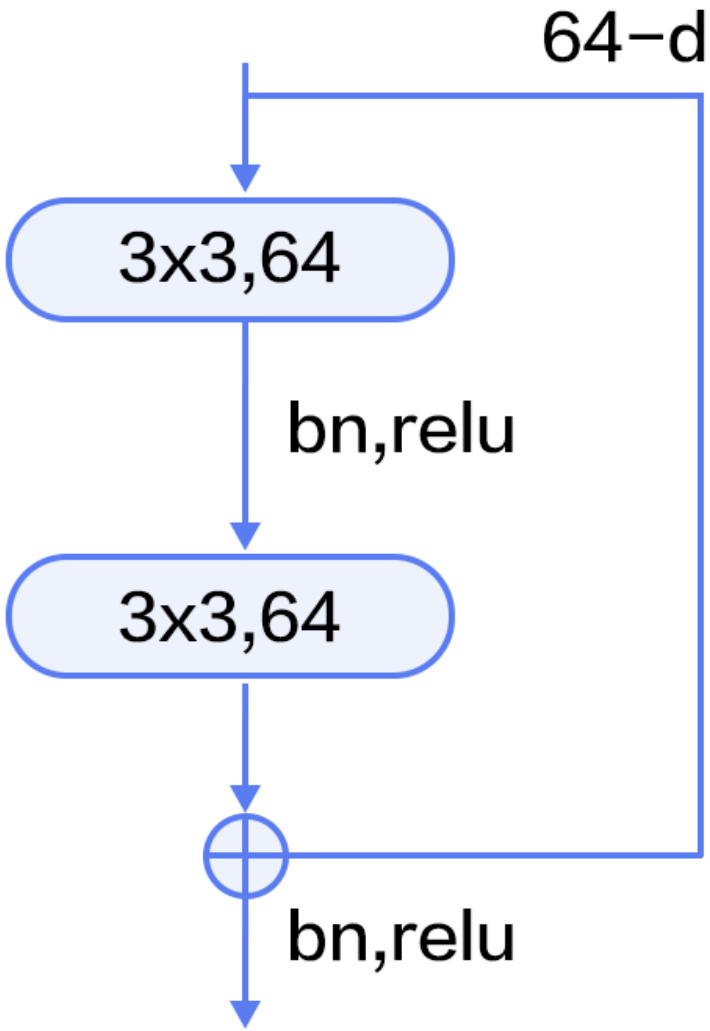


ResNet (2015)


Preserving base information
Conv Layers
can treat perturbation

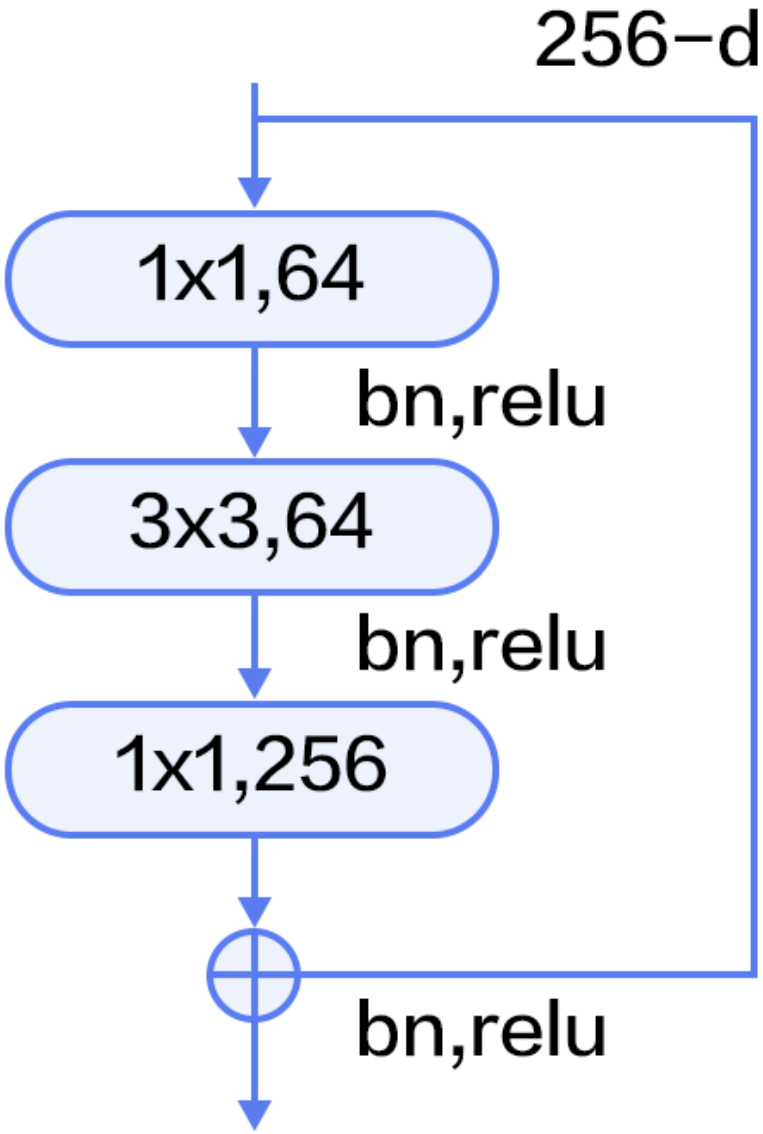
ResNet (2015)

- Residual block



ResNet (2015)

- **Residual Bottleneck** consist of a 1×1 layer for reducing dimension, a 3×3 layer, and a 1×1 layer for restoring dimension.



ResNet (2015)

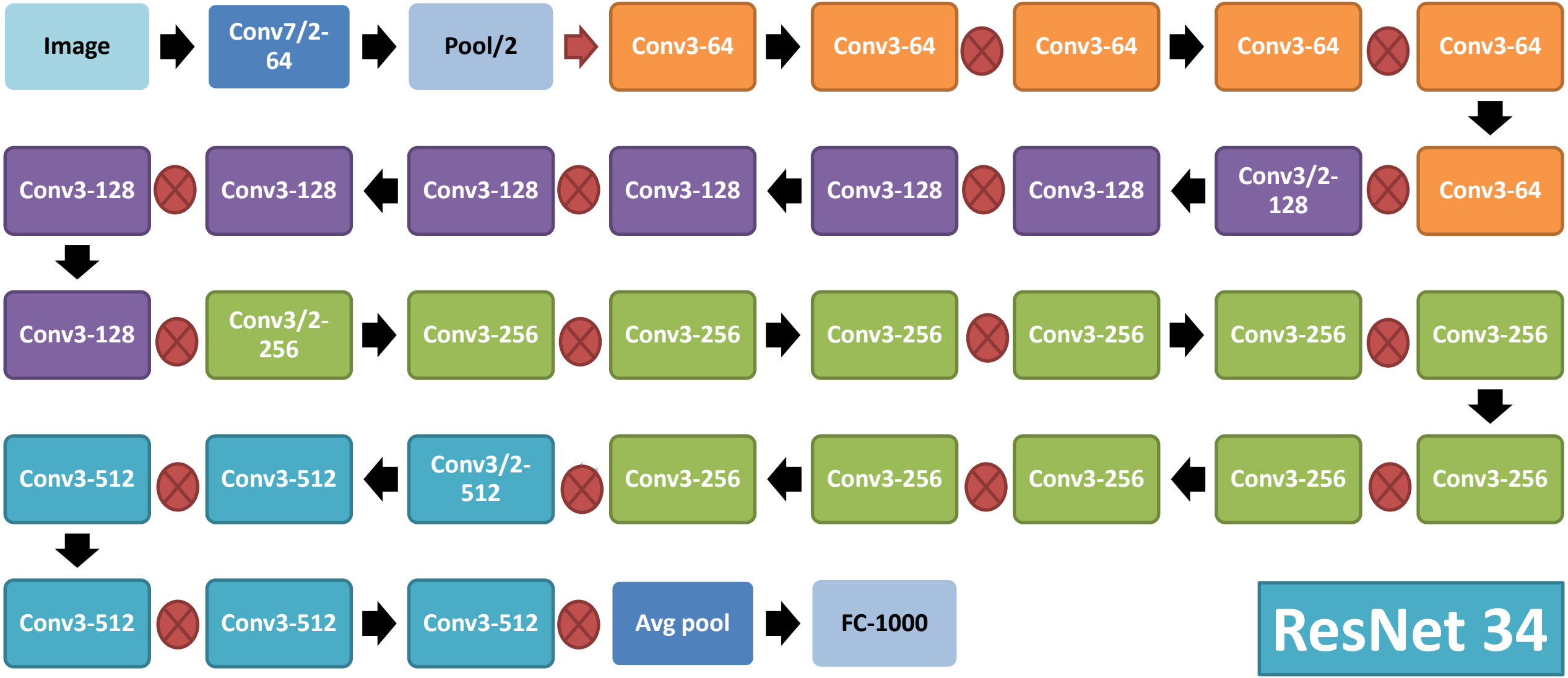


Image
Conv7/2-64
Pool/2
Conv3-64
Conv3-64
Conv3-64
Conv3-64
Conv3-64
Conv3-64
Conv3/2-128
Conv3-128
Conv3-128
Conv3-128
Conv3-128
Conv3-128
Conv3-128
Conv3-128
Conv3/2-256
Conv3-256
Conv3-256
Conv3-256
Conv3-256
Conv3-256
Conv3-256
Conv3-256
Conv3/2-512
Conv3-512
Conv3-512
Conv3-512
Conv3-512
Conv3-512
Avg pool
FC-1000
ResNet 34

ResNet (2015)

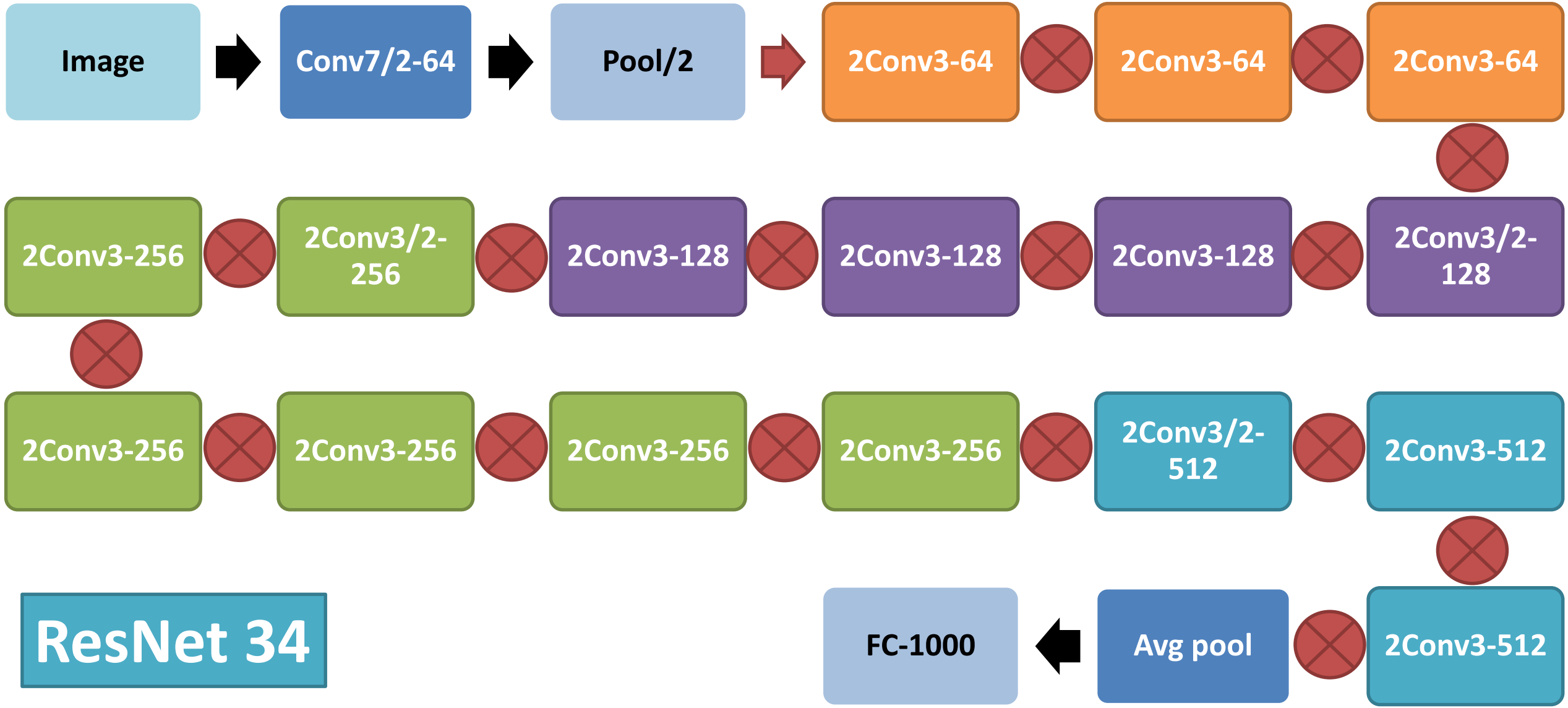
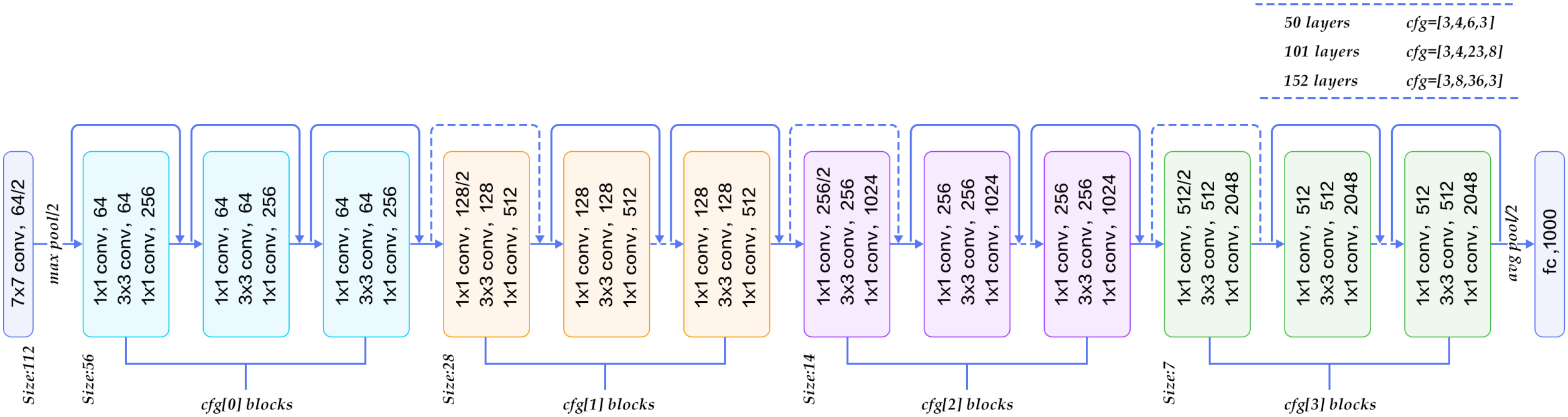


Image
Conv7/2-64
Pool/2
2Conv3-64
2Conv3-64
2Conv3-64
2Conv3/2-128
2Conv3-128
2Conv3-128
2Conv3-128
2Conv3/2-256
2Conv3-256
2Conv3-256
2Conv3-256
2Conv3-256
2Conv3-256
2Conv3/2-512
2Conv3-512
2Conv3-512
Avg pool
FC-1000
ResNet 34

ResNet (2015)

❑ ResNet Model



ResNet (2015)

Layer	Output	18-Layer	34-Layer	50-Layer	101-Layer	152-Layer
Conv-1	112x112	7x7/2-64				
Conv-2	56x56	3x3 Maxpooling/2				
		$2x\begin{pmatrix}3x3,64\\3x3,64\end{pmatrix}$	$3x\begin{pmatrix}3x3,64\\3x3,64\end{pmatrix}$	$3x\begin{pmatrix}1x1,64\\3x3x64\\1x1x256\end{pmatrix}$	$3x\begin{pmatrix}1x1,64\\3x3x64\\1x1x256\end{pmatrix}$	$3x\begin{pmatrix}1x1,64\\3x3x64\\1x1x256\end{pmatrix}$
Conv-3	28x28	$2x\begin{pmatrix}3x3,128\\3x3,128\end{pmatrix}$	$4x\begin{pmatrix}3x3,128\\3x3,128\end{pmatrix}$	$4x\begin{pmatrix}1x1,128\\3x3x128\\1x1x512\end{pmatrix}$	$4x\begin{pmatrix}1x1,128\\3x3x128\\1x1x512\end{pmatrix}$	$8x\begin{pmatrix}1x1,128\\3x3x128\\1x1x512\end{pmatrix}$
Conv-4	14x14	$2x\begin{pmatrix}3x3,256\\3x3,256\end{pmatrix}$	$6x\begin{pmatrix}3x3,256\\3x3,256\end{pmatrix}$	$6x\begin{pmatrix}1x1,256\\3x3x256\\1x1x1024\end{pmatrix}$	$23x\begin{pmatrix}1x1,256\\3x3x256\\1x1x1024\end{pmatrix}$	$36x\begin{pmatrix}1x1,256\\3x3x256\\1x1x1024\end{pmatrix}$
Conv-5	7x7	$2x\begin{pmatrix}3x3,512\\3x3,512\end{pmatrix}$	$3x\begin{pmatrix}3x3,512\\3x3,512\end{pmatrix}$	$3x\begin{pmatrix}1x1,512\\3x3x512\\1x1x2048\end{pmatrix}$	$3x\begin{pmatrix}1x1,512\\3x3x512\\1x1x2048\end{pmatrix}$	$3x\begin{pmatrix}1x1,512\\3x3x512\\1x1x2048\end{pmatrix}$
	1x1	Avgpool-FC1000-Softmax				
Flops		$1.8x10^9$	$3.6x10^9$	$3.8x10^9$	$7.6x10^9$	$11.3x10^9$

ResNet (2015)

- Implement **ResNet** using **TFLearn**

```
from __future__ import division, print_function, absolute_import

import tflearn

# Residual blocks
# 32 layers: n=5, 56 layers: n=9, 110 layers: n=18
n = 5

# Data loading
from tflearn.datasets import cifar10
(X, Y), (testX, testY) = cifar10.load_data()
Y = tflearn.data_utils.to_categorical(Y, 10)
testY = tflearn.data_utils.to_categorical(testY, 10)
```

ResNet (2015)

```
from tflearn.datasets import cifar10
(X, Y), (testX, testY) = cifar10.load_data()
Y = tflearn.data_utils.to_categorical(Y, 10)
testY = tflearn.data_utils.to_categorical(testY, 10)

# Real-time data preprocessing
img_prep = tflearn.ImagePreprocessing()
img_prep.add_featurewise_zero_center(per_channel=True)

# Real-time data augmentation
img_aug = tflearn.ImageAugmentation()
img_aug.add_random_flip_leftright()
img_aug.add_random_crop([32, 32], padding=4)
```

ResNet (2015)

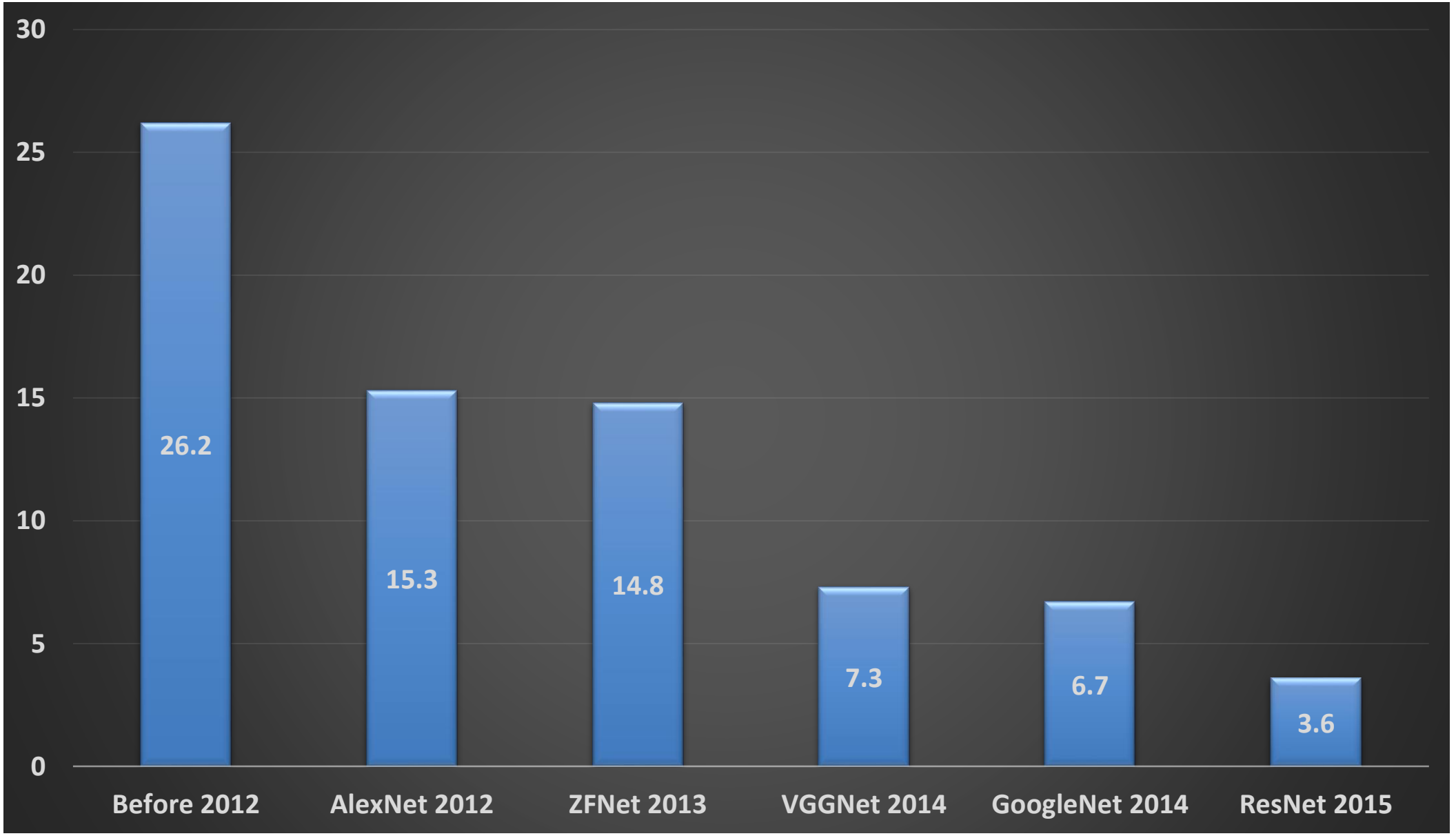
```
# Building Residual Network
net = tflearn.input_data(shape=[None, 32, 32, 3],
                         data_preprocessing=img_prep,
                         data_augmentation=img_aug)
net = tflearn.conv_2d(net, 16, 3, regularizer='L2', weight_decay=0.0001)
net = tflearn.residual_block(net, n, 16)
net = tflearn.residual_block(net, 1, 32, downsample=True)
net = tflearn.residual_block(net, n-1, 32)
net = tflearn.residual_block(net, 1, 64, downsample=True)
net = tflearn.residual_block(net, n-1, 64)
net = tflearn.batch_normalization(net)
net = tflearn.activation(net, 'relu')
net = tflearn.global_avg_pool(net)
```

ResNet (2015)

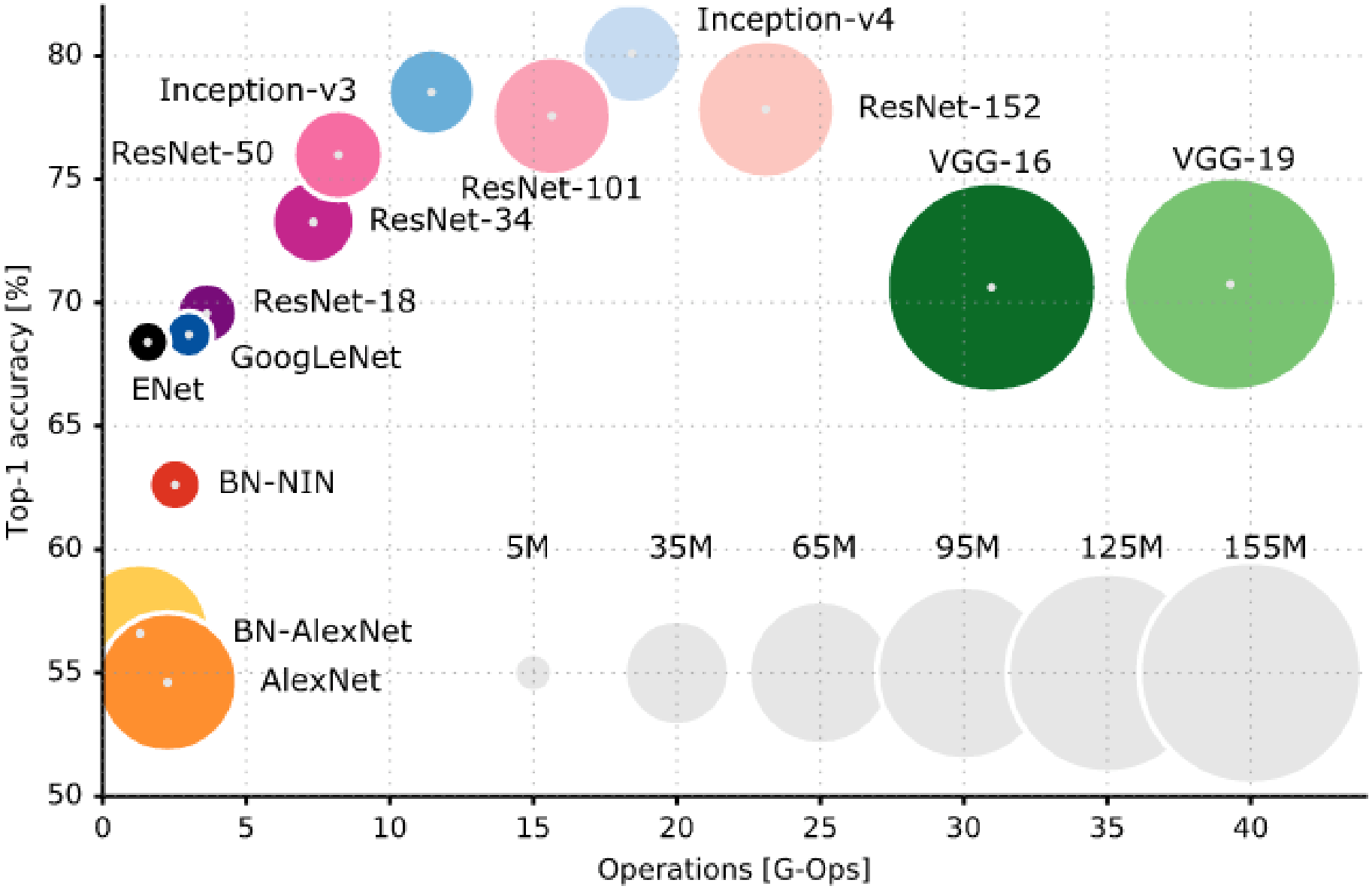
```
# Regression
net = tflearn.fully_connected(net, 10, activation='softmax')
mom = tflearn.Momentum(0.1, lr_decay=0.1, decay_step=32000, staircase=True)
net = tflearn.regression(net, optimizer=mom,
                         loss='categorical_crossentropy')
# Training
model = tflearn.DNN(net, checkpoint_path='model_resnet_cifar10',
                    max_checkpoints=10, tensorboard_verbose=0,
                    clip_gradients=0.)

model.fit(X, Y, n_epoch=200, validation_set=(testX, testY),
          snapshot_epoch=False, snapshot_step=500,
          show_metric=True, batch_size=128, shuffle=True,
          run_id='resnet_cifar10')
```

Table of Contents

Top-5 error rates on ILSVRC image classification



CNN Models


Inception-v4
Inception-v3
ResNet-152
ResNet-50
ResNet-101
VGG-16
VGG-19
ResNet-34
ResNet-18
GoogLeNet
ENet
BN-NIN
5M
35M
65M
95M
125M
155M
BN-AlexNet
AlexNet
Top-1 accuracy [%]
80
75
70
65
60
55
50
0
5
10
15
20
25
30
35
40
Operations [G-Ops]

Contact Me


facebook.com/mloey
mohamedloey@gmail.com
twitter.com/mloey


linkedin.com/in/mloey
BENHA UNIVERSITY
mloey@fci.bu.edu.eg
mloey.github.io


THANKS FOR
YOUR TIME