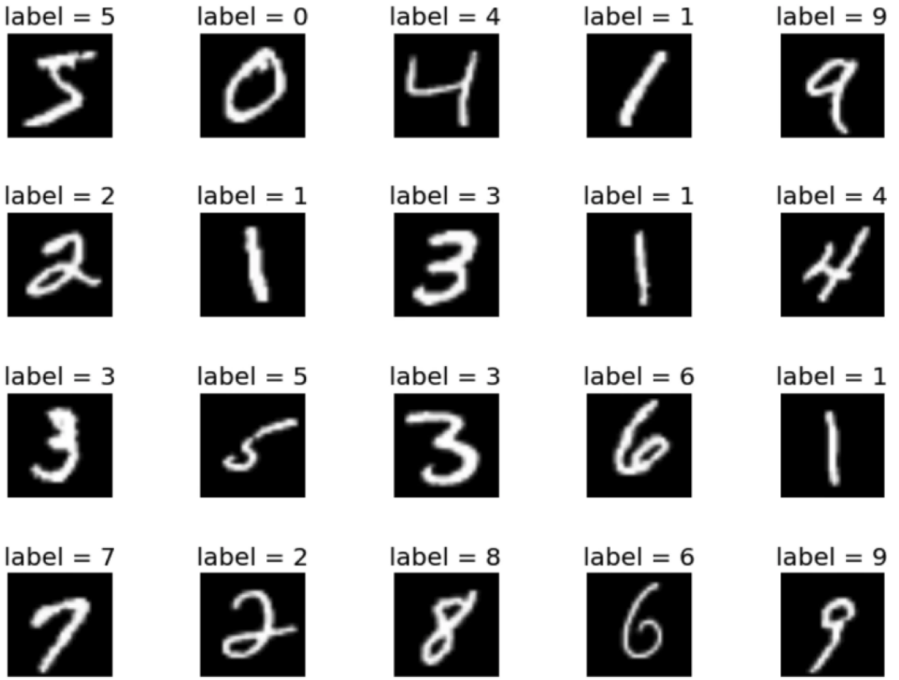
Deep Learning

ABNANE Ibtissam

Transfer Learning

Good deep learning models are dependednt on:

- ❑ Model architecture
  - Number/size of filters
  - Number of convolution layers
- ❑ Quality of training
  - Time
  - Good data
  - Computational power



Transfer learning: Taking the knowledge resulted from the completion of one task, and implement it to achieve a new task

Transfer Learning

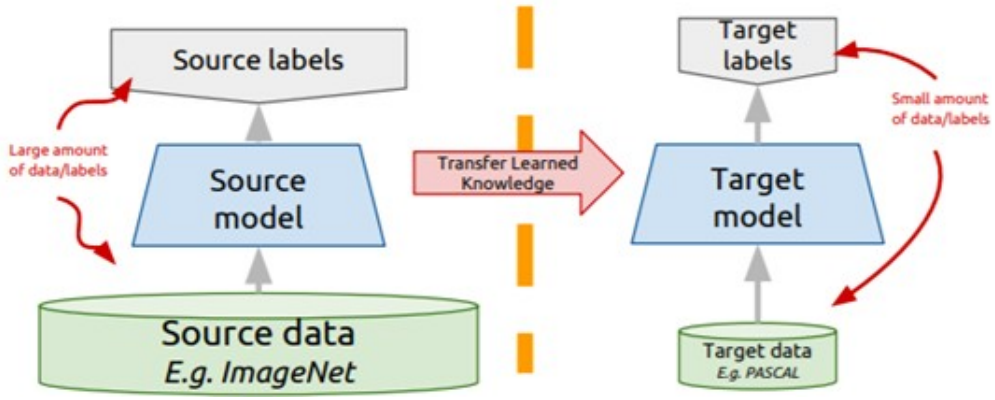
Transfer learning: idea

Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations:

- Same domain, different task
- Different domain, same task



Transfer Learning

Example: Suppose you are a camera surveillance company, hired by the government to install cameras in the city to identify parked cars.
Goal: Identify cars that have overstayed the parking limit and deserve a ticket, during 9 am to 5pm
Solution: Implement a neural network that takes camera footage as input and identity cars (parked in the day light)
Due to the success of the application, the government want to extend the time period to night also:
Option 1: start from scratch and build a new models trained on both images during day and night
Option 2: use your pretrained model, adjust it on night images=> additional training on night images

Transfer learning

Scenario: Improving Skin Cancer Detection

Imagine a medical research team working on a project to improve the early detection of skin cancer, a critical and life-saving task. The team faces a challenge: they have a limited dataset of skin lesion images from their local hospital, which isn't sufficient for training a robust deep learning model.

Problem: The team needs to create a skin cancer detection system that can accurately classify skin lesions as malignant or benign, but they lack the necessary labeled data.

Solution: Transfer Learning overcome data limitations and improve the accuracy of their skin cancer detection model.

Transfer learning

Scenario: Personalized Healthcare with Limited Medical Data

Imagine a small rural clinic in a remote area that lacks access to advanced medical expertise and technology. The clinic serves a diverse population with a range of medical conditions, but they have limited patient data for making accurate diagnoses and treatment decisions.

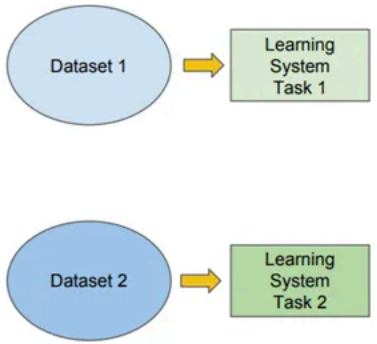
Problem: The clinic's healthcare practitioners face challenges in providing personalized medical care due to the scarcity of patient data and limited access to specialized medical knowledge.

Solution: Transfer Learning => empowers a rural clinic to provide personalized healthcare services despite data limitations.

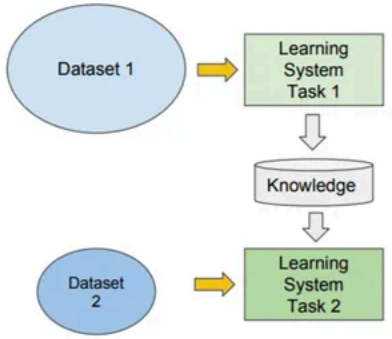
Transfer Learning

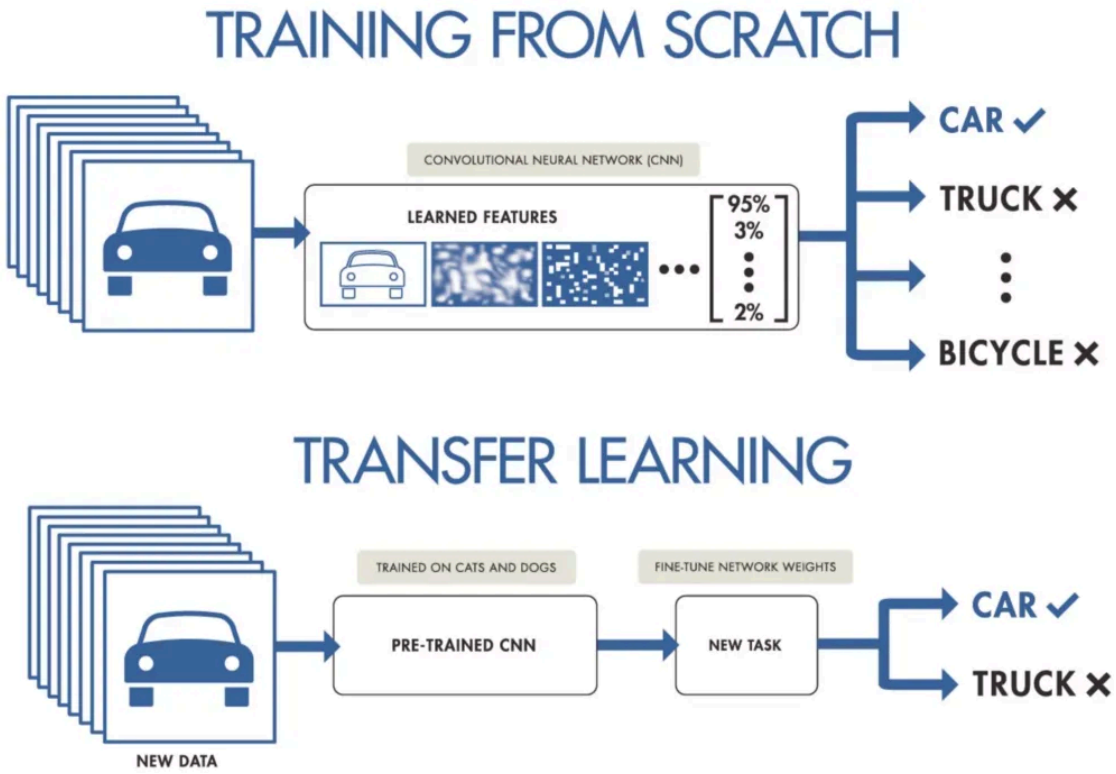
Traditional ML vs Transfer Learning

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data



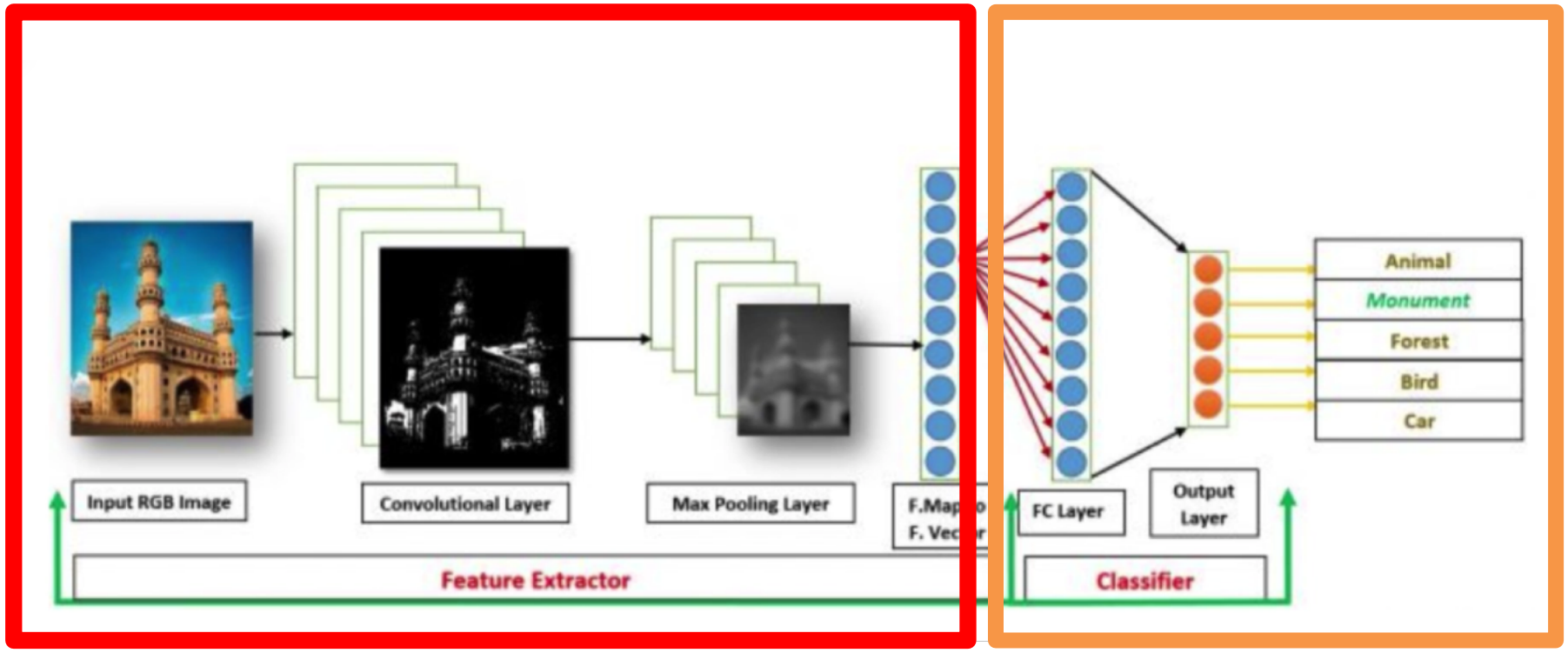


Transfer Learning

Ant/bee classification

- Not enough labelled data (400 images)
- Effective pretrained model already exists

Transfer Learning



Freeze **Re-train**

Transfer Learning

When:

- **Limited Data Availability:** Instead of training a deep learning model from scratch, you can start with a pre-trained model on a large dataset and fine-tune it on your smaller dataset. This leverages the knowledge already captured in the pre-trained model.
- **Similar Tasks:** Transfer learning is most effective when the source task (the one the pre-trained model was trained on) is related to the target task (your specific problem).
- **Reduced Training Time:** Can significantly reduce training time because you're building on pre-existing knowledge.
- **Resource Constraints:** If you don't have access to large-scale computing resources, transfer learning can be a more practical option than training a model from scratch.

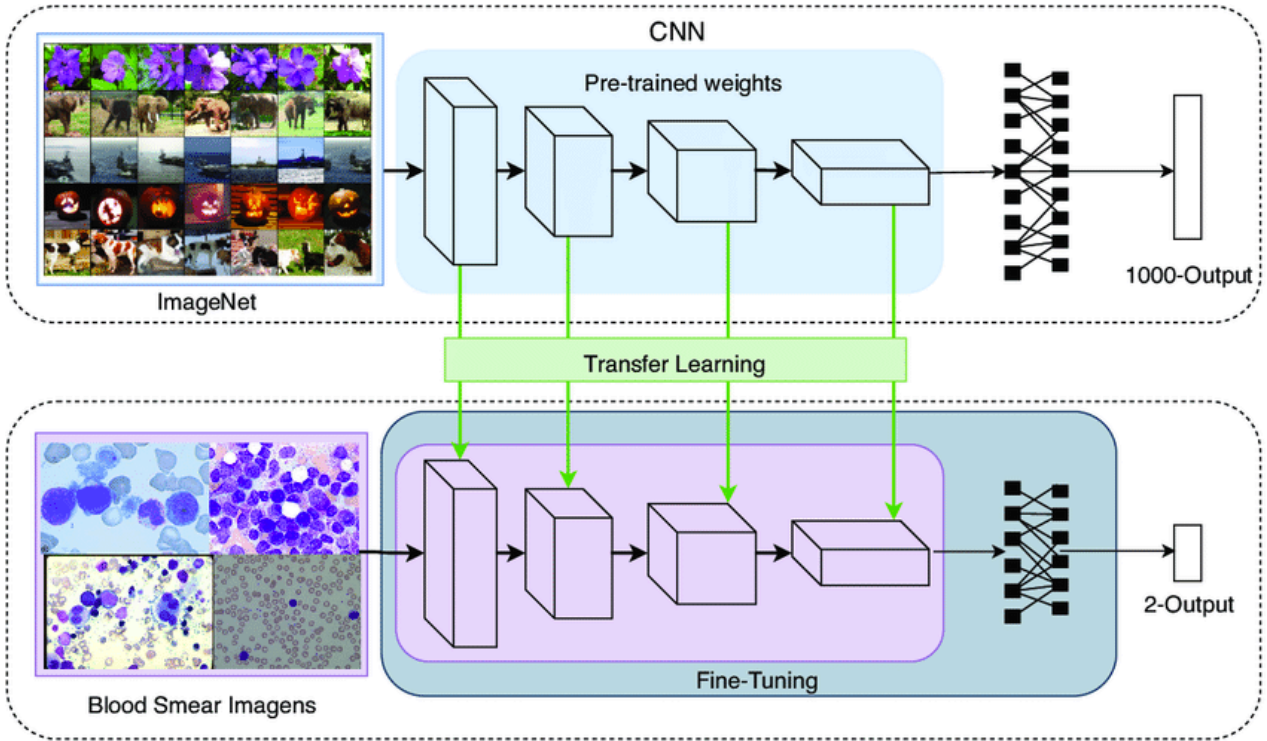
Transfer Learning

- Broad concept that encompasses various techniques for utilizing knowledge from a pre-trained model in a new task.
- Involves taking a model that has been trained on a **source task** and applying it to a **related target task.** The primary idea is to transfer knowledge learned from the source task to the target task, which can help **improve** performance or **reduce the amount of data** required for training
- Typically replaces the final layers (the task-specific output layers) of the pre-trained model with new layers that match the target task. These new layers are randomly initialized.
- The weights of the pre-trained model are often **frozen**, meaning they are not updated during training on the target task. Only the weights of the new layers are trained on the target data.
- **Transfer learning is commonly used when the source and target tasks are related but not identical, and when the target dataset is relatively small.**

Transfer Learning

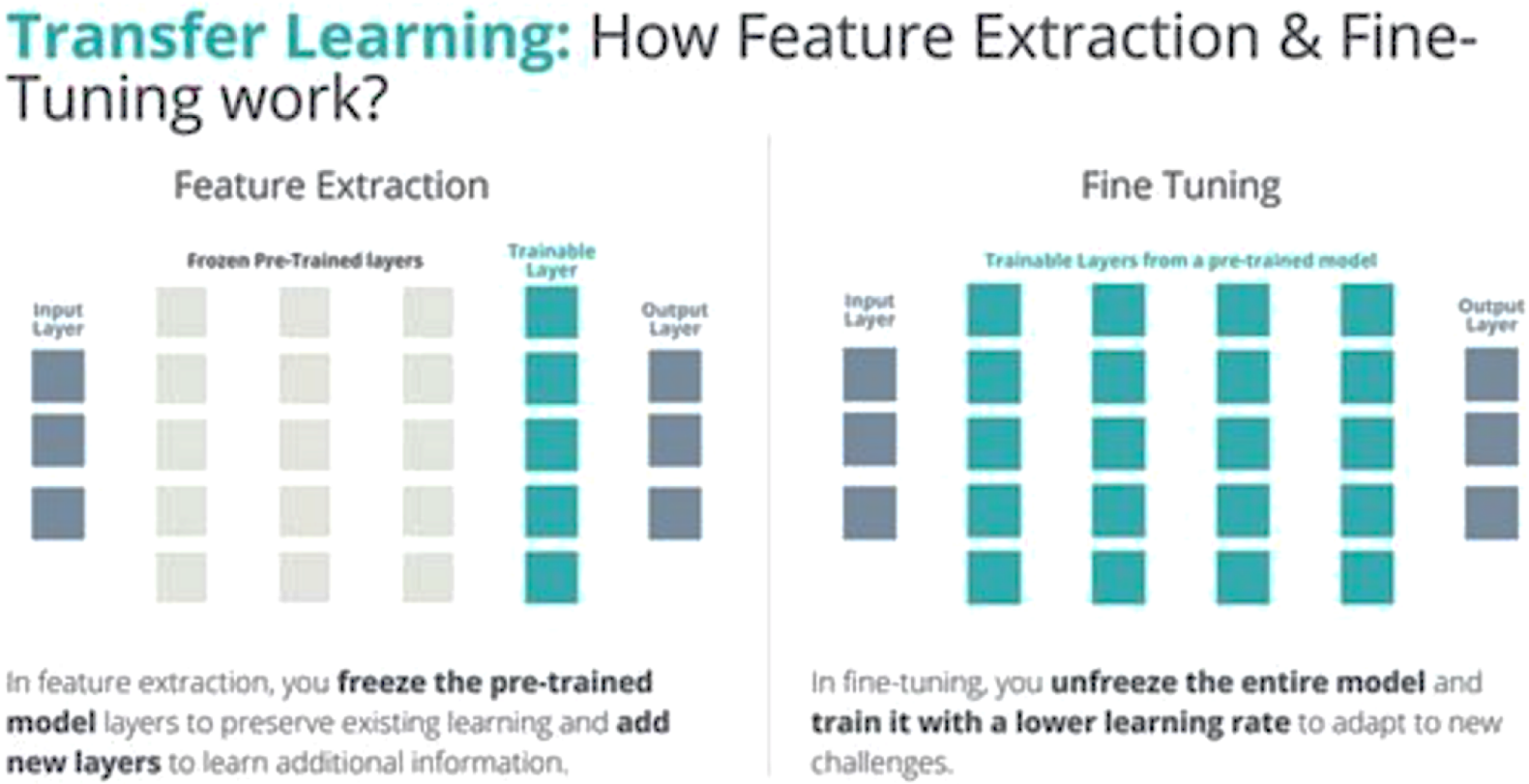
Fine tuning

- Fine-tuning a techniques within transfer learning where not only the final layers but also some earlier layers of the pre-trained model are modified to adapt it to the target task.
- While transfer learning freezes all the pre-trained layers and only trains the new layers, fine-tuning goes a step further by allowing the pre-trained layers to be updated



Transfer Learning

Fine tuning



Transfer learning

Scenario	Use Fine-Tuning	Freeze Layers (Feature Extraction)
Similar source and target tasks	**Yes**	Sometimes
Moderate to sufficient target data	**Yes**	Sometimes (especially with limited data)
Task requires task-specific features	**Yes**	Sometimes (if not needed)
Complex target task	**Yes**	Sometimes (depends on task complexity)
Limited target data	Sometimes (if enough data)	**Yes**
Dissimilar source and target tasks	Sometimes (if some features transfer)	**Yes**
Limited computational resources	Sometimes (if resources are available)	**Yes**
Privacy and security concerns with the pre-trained model	Sometimes (if no sensitive data)	**Yes**

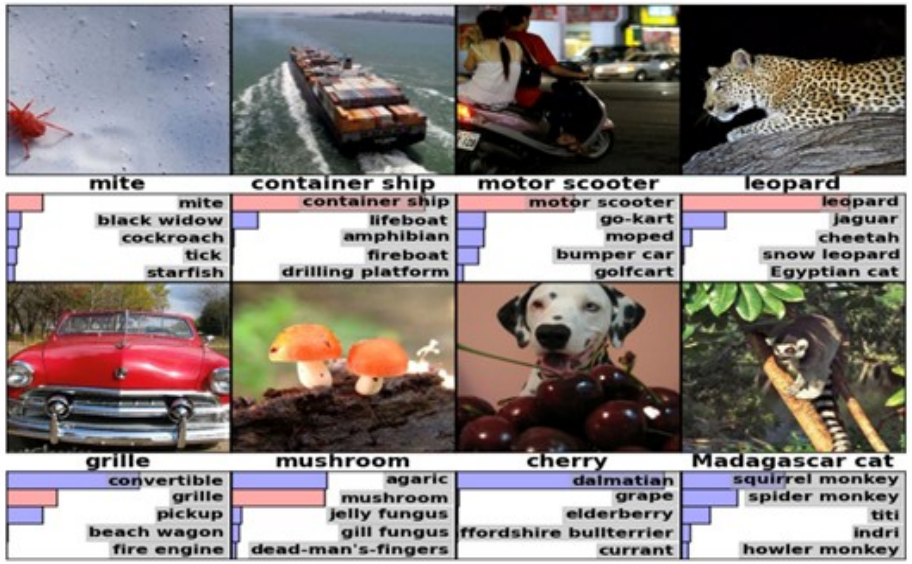
Transfer Learning

ImageNet Large Scale Visual Recognition Challenge (ILSVRC), was an annual computer vision competition that aimed to evaluate the performance of computer vision algorithms on a large-scale image dataset. The challenge was initiated in 2010 and ran until 2017.

ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.



Transfer Learning

Main task: image classification.

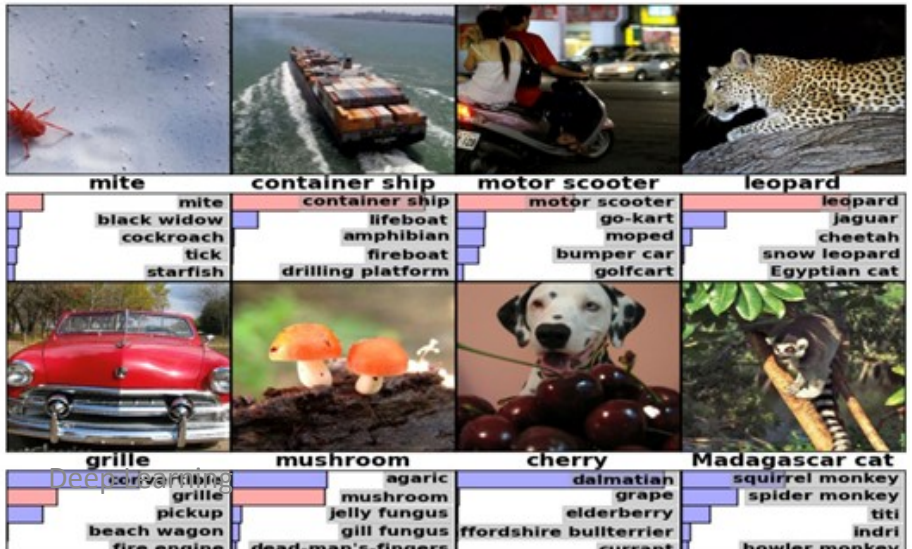
Participants were provided with a dataset called ImageNet, which contained millions of labeled images across thousands of categories.

The goal was to develop algorithms that could accurately classify images into the correct categories. Initially, it focused on 1,000 object categories.

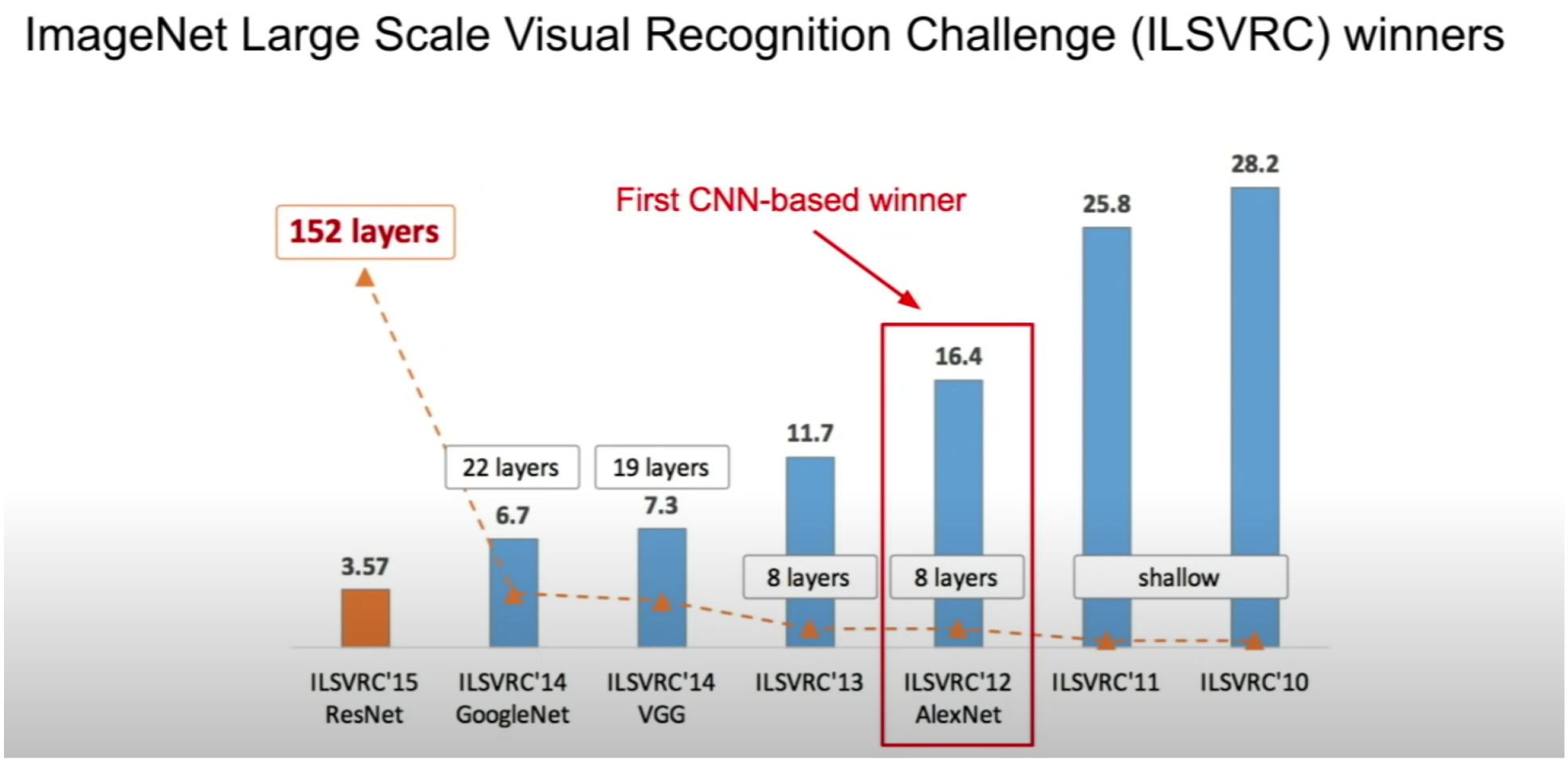
ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.



ImageNet Challenge



ImageNet

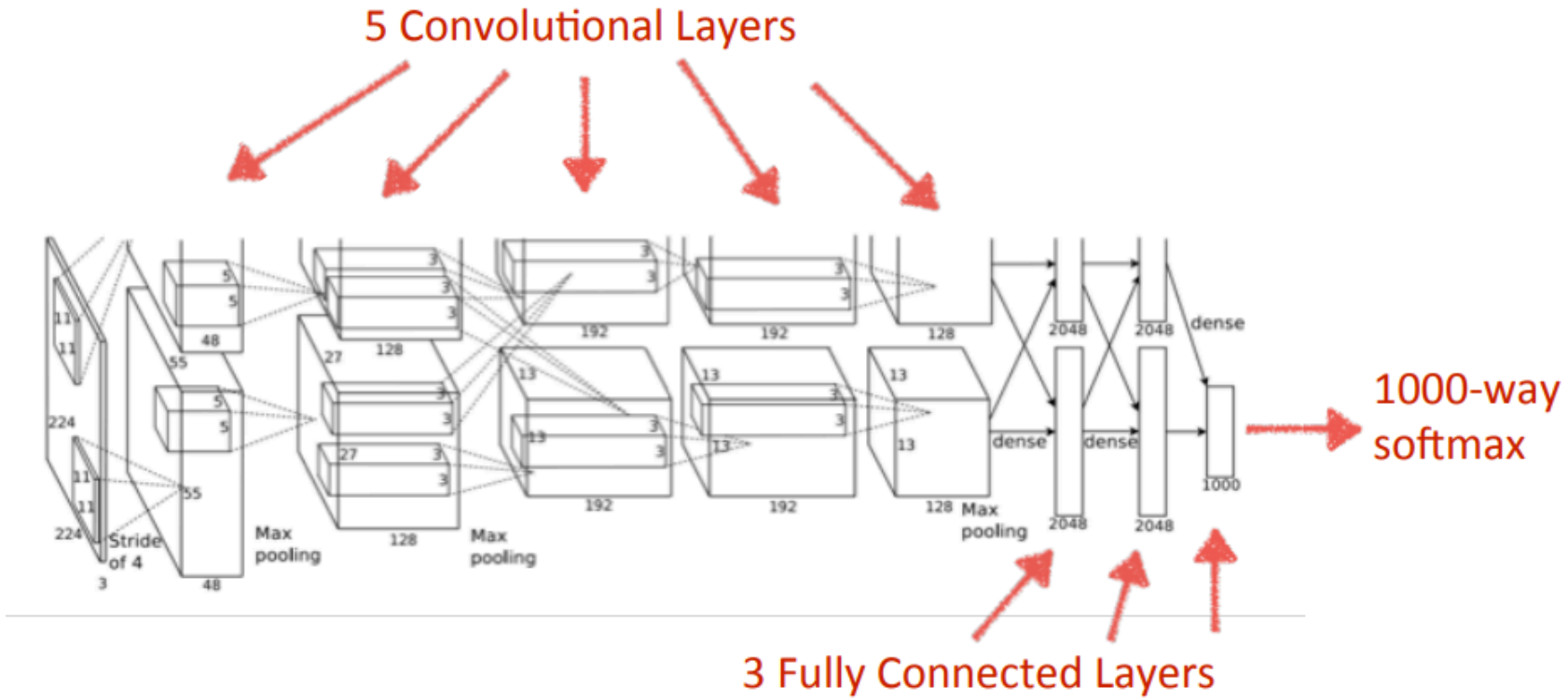
In the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), "top-1" and "top-5" errors are metrics used to evaluate the performance of image classification models.

Top-1 Error: Measures the percentage of test images for which the correct category (the single most likely category predicted by the model) is not the same as the ground truth label. In other words, it calculates the proportion of images where the model's top prediction is incorrect.

Top-5 Error: Measures the percentage of test images for which the correct category is not among the top five most likely categories predicted by the model. In other words, it calculates the proportion of images where the correct label is not found within the model's top five predictions.

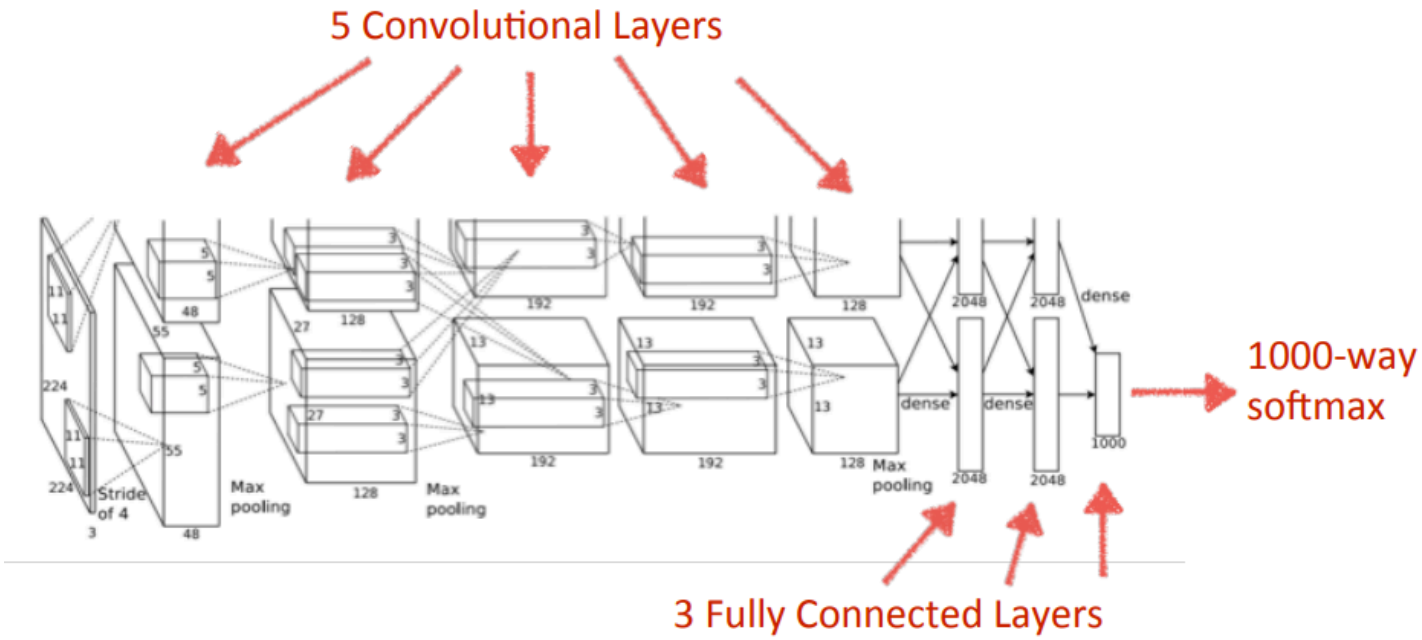
AlexNet

- 2012: one of the first deep convolutional neural networks (CNNs) to demonstrate the effectiveness of deep architectures for image classification.
- Top 5 test error rate of 15.4% (2nd **26,2 %**)

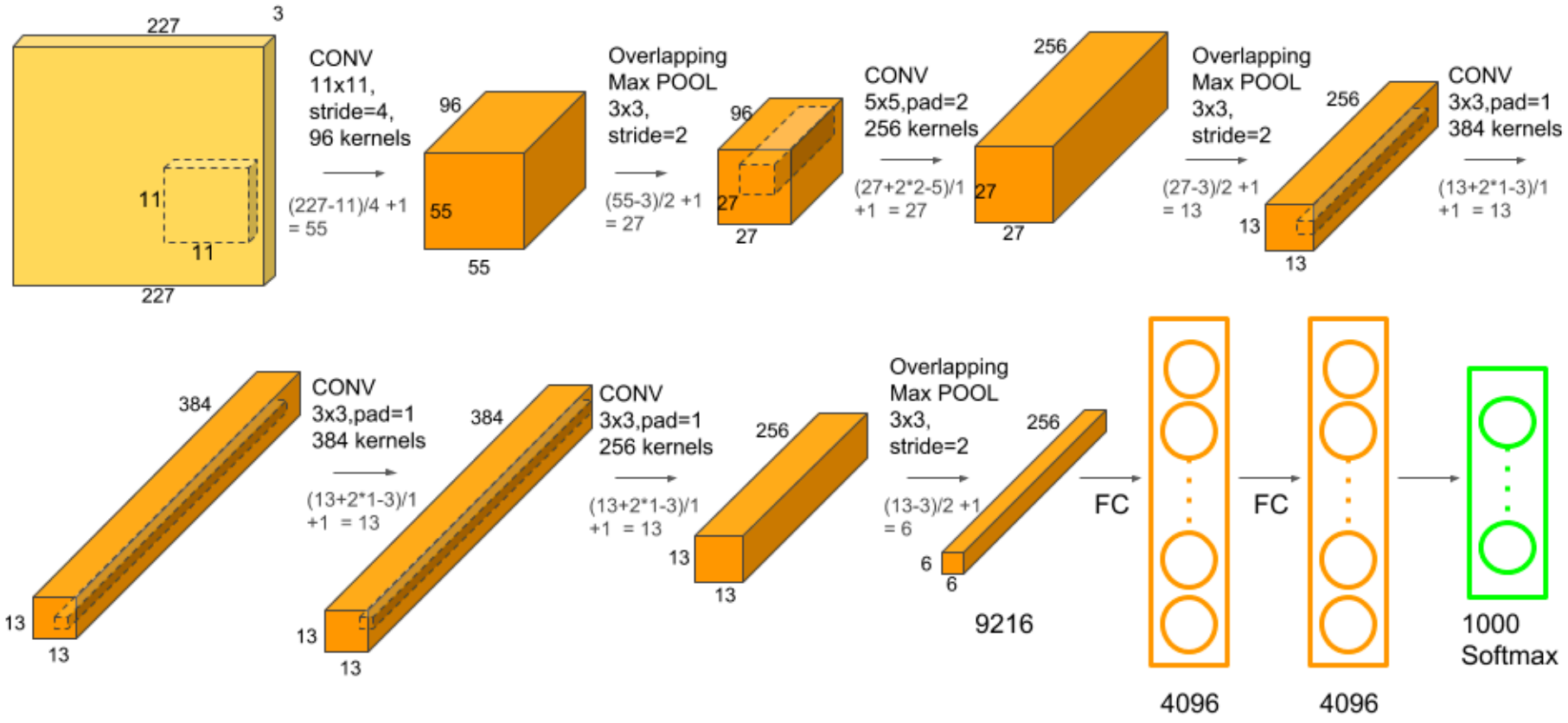


AlexNet

- Used ReLU for non-linearity
- Used data augmentation techniques
- Implemented dropout layers to prevent overfitting
- Trained on two GTX 580 GPUs for five to six days

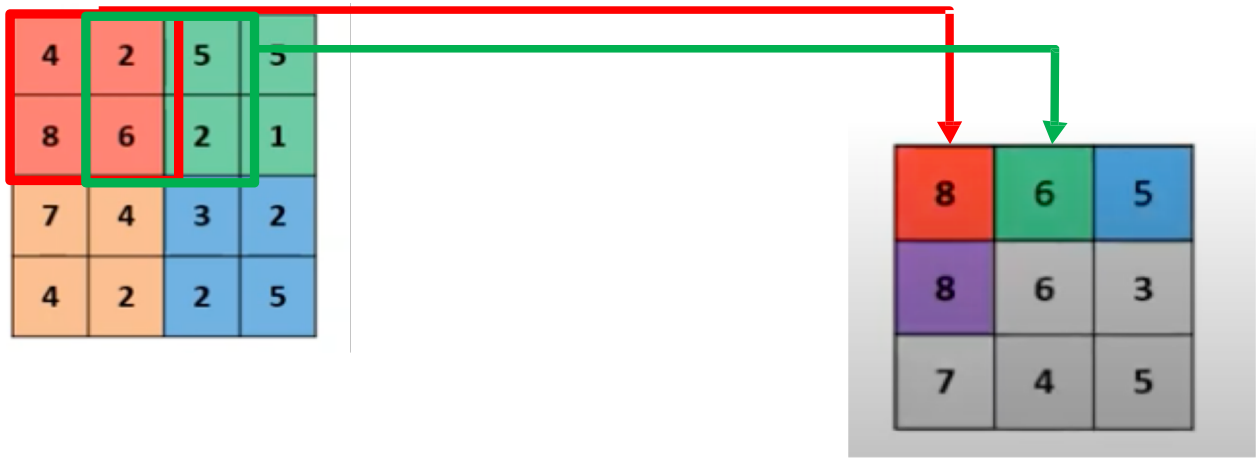


AlexNet


227
3
11
11
227
CONV
11x11,
stride=4,
96 kernels
(227-11)/4 +1
= 55
96
55
55
Overlapping
Max POOL
3x3,
stride=2
(55-3)/2 +1
= 27
96
27
27
CONV
5x5,pad=2
256 kernels
(27+2*2-5)/1
+1 = 27
256
27
27
Overlapping
Max POOL
3x3,
stride=2
(27-3)/2 +1
= 13
256
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
256 kernels
(13+2*1-3)/1
+1 = 13
256
13
13
Overlapping
Max POOL
3x3,
stride=2
(13-3)/2 +1
= 6
256
6
6
9216
FC
4096
FC
4096
1000
Softmax

AlexNet

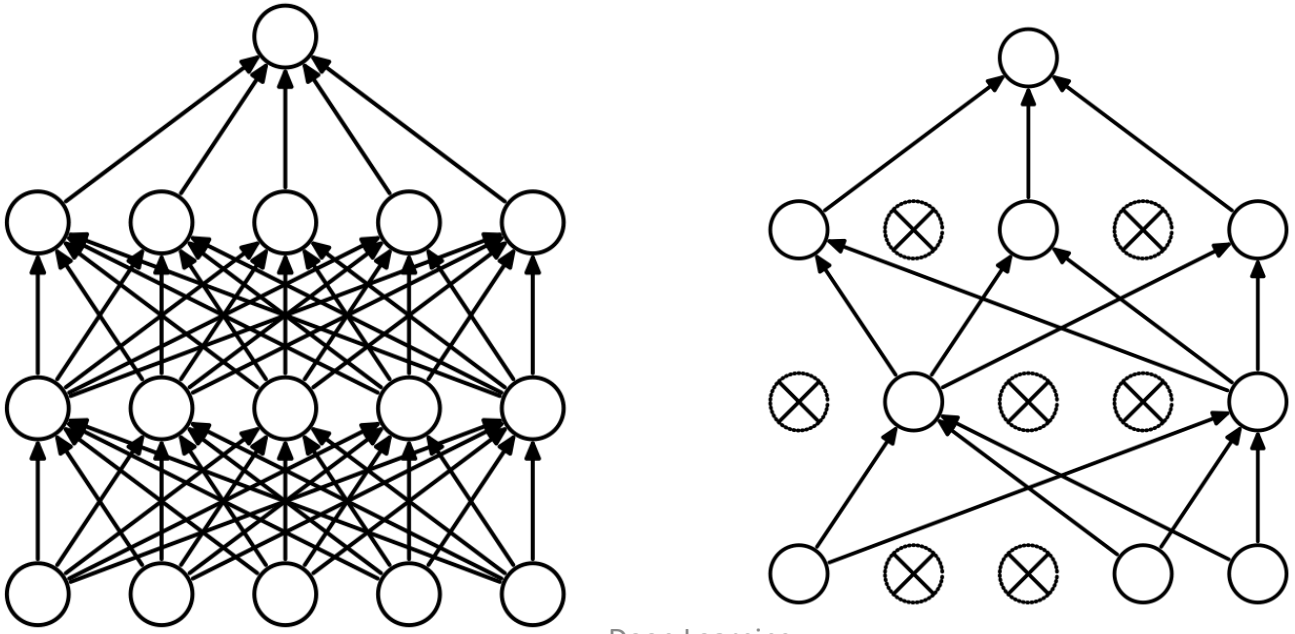
Overlapping pooling:



AlexNet

Dropout

- Dropout is a regularization technique commonly used in deep learning, to prevent overfitting.
- Overfitting occurs when a model performs well on the training data but fails to generalize effectively to unseen data.



Transfer learning

Dropout

During Training: dropout randomly **deactivates** (sets to zero) a fraction of neurons in a neural network layer on each forward and backward pass. These deactivated neurons do not contribute to the computations for that particular iteration. The fraction of neurons that are deactivated is a **hyperparameter**, typically set between 0.2 and 0.5.

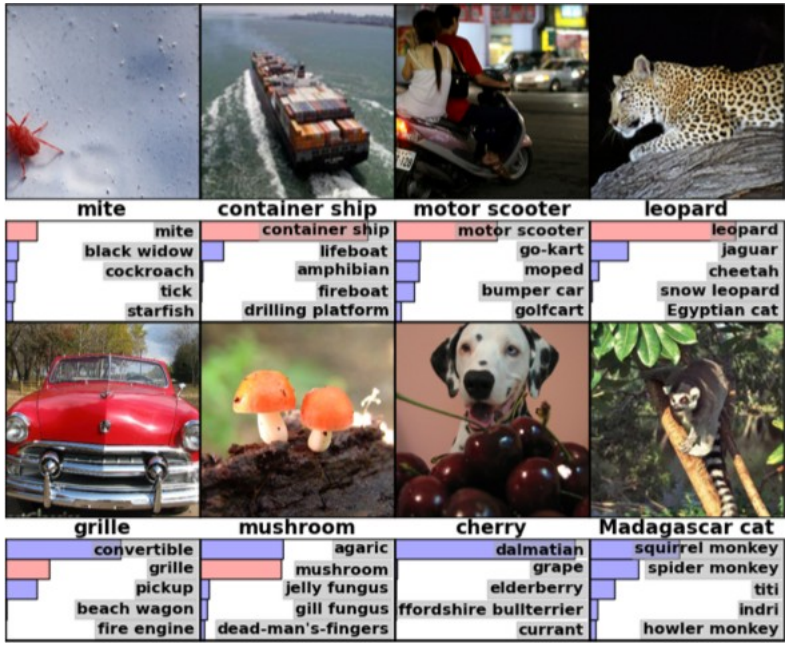
Random Deactivation: The key idea behind dropout is that it prevents individual neurons from becoming overly specialized or dependent on specific input features.

Testing Phase: dropout is turned off, and all neurons are active. The predictions are made using the full network.

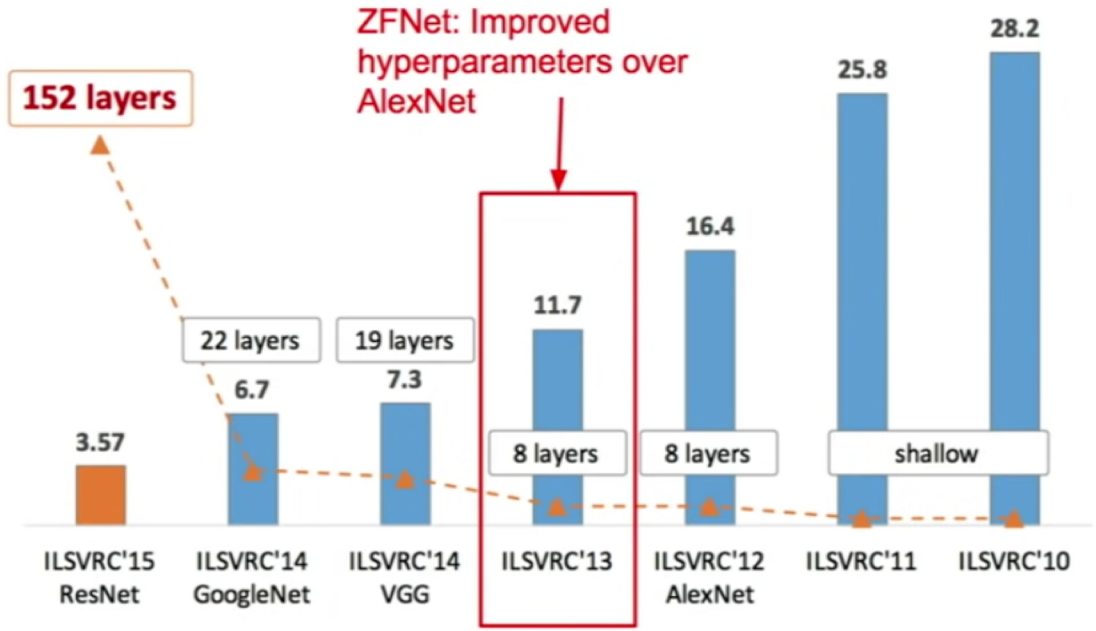
AlexNet

The Results.

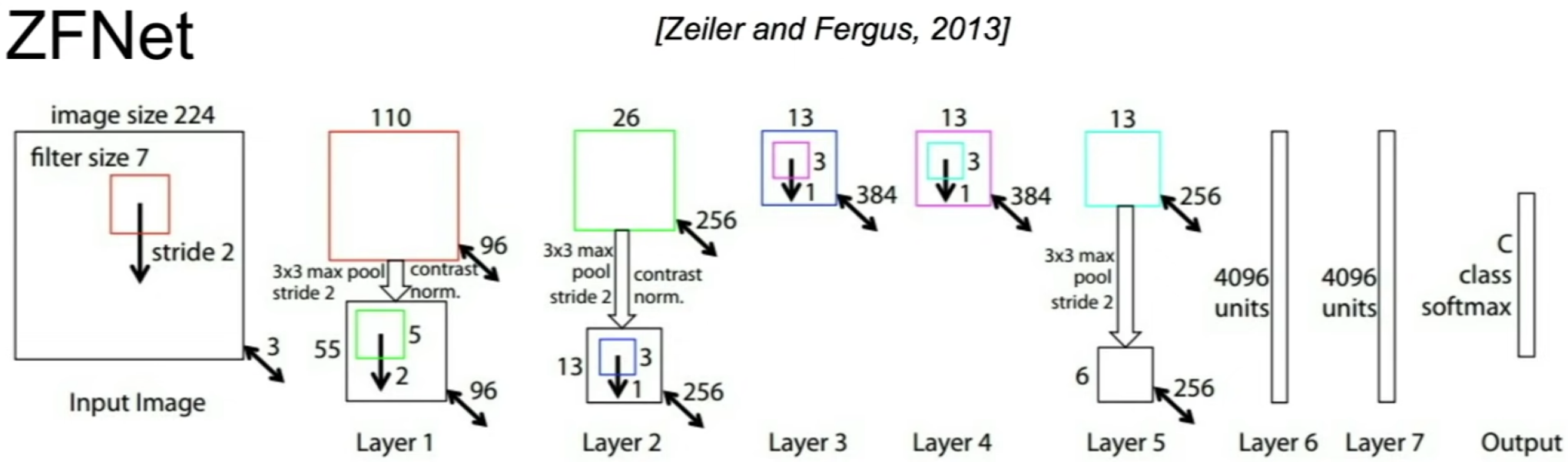
- On the 2010 version of the ImageNet competition, the best model achieved **47.1% top-1 error and 28.2% top-5 error.**
- AlexNet vastly outpaced this with a **37.5% top-1 error and a 17.0% top-5 error.**
- AlexNet **won the 2012 ImageNet competition** with a top-5 error rate of 15.3%, compared to the second place top-5 error rate of 26.2%.



ZFNet


ZFNet: Improved hyperparameters over AlexNet
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

ZFNet



TODO: remake figure

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

Parameters

How to calculate parameters for a CNN?

```
1 my_input = Input(shape = (10, 10, 1))
2 my_output = Convolution2D(1, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 model = Model(my_input, my_output)
```

```
1 model.summary()
```

- **Black and white image**
- **10*10 image**
- **1 convolutional filter**
- **3*3 filter**
- **The output => 8*8 image**

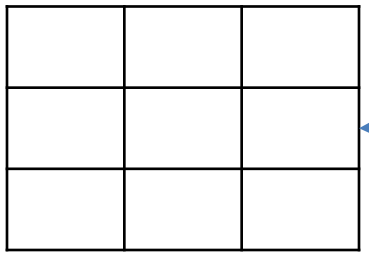
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_17 (InputLayer)        [(None, 10, 10, 1)]       0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 8, 8, 1)           10
_________________________________________________________________
activation_19 (Activation)   (None, 8, 8, 1)           0
=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
_________________________________________________________________
```

Parameters

```
1 my_input = Input(shape = (10, 10, 1))
2 my_output = Convolution2D(1, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 model = Model(my_input, my_output)
```

```
1 model.summary()
```

- Black and white image (Grayscale)
- 10*10 image
- 1 convolutional filter (3*3)
- The output => 8*8 image



3*3 filter + bias = 10

```
Layer (type)                 Output Shape              Param #
=================================================================
input_17 (InputLayer)        [(None, 10, 10, 1)]       0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 8, 8, 1)           10
_________________________________________________________________
activation_19 (Activation)   (None, 8, 8, 1)           0
=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
_________________________________________________________________
```

Parameters

Example 2

```
1 my_input = Input(shape = (10, 10, 1))
2 my_output = Convolution2D(5, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(2, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

- **Black and white image (Grayscale)**
- **10*10 image**
- **5 convolutional filters (3*3)**

Layer (type)	Output Shape	Param #
input_19 (InputLayer)	[(None, 10, 10, 1)]	0
conv2d_22 (Conv2D)	(None, 8, 8, 5)	50
activation_22 (Activation)	(None, 8, 8, 5)	0
conv2d_23 (Conv2D)	(None, 6, 6, 2)	92
activation_23 (Activation)	(None, 6, 6, 2)	0

Total params: 142
Trainable params: 142
Non-trainable params: 0

Parameters

Example 2

```
1 my_input = Input(shape = (10, 10, 1))
2 my_output = Convolution2D(5, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(2, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

- **Black and white image (Grayscale)**
- **10*10 image**
- **5 convolutional filters (3*3)**
- **Output?**

Layer (type)	Output Shape	Param #
input_19 (InputLayer)	[(None, 10, 10, 1)]	0
conv2d_22 (Conv2D)	(None, 8, 8, 5)	50
activation_22 (Activation)	(None, 8, 8, 5)	0
conv2d_23 (Conv2D)	(None, 6, 6, 2)	92
activation_23 (Activation)	(None, 6, 6, 2)	0

Total params: 142
Trainable params: 142
Non-trainable params: 0

Parameters

Example 2

```
1 my_input = Input(shape = (10, 10, 1))
2 my_output = Convolution2D(5, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(2, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

- Black and white image (Grayscale)
- 10*10 image
- 5 convolutional filters (3*3)
- Output?

ayer (type)	Output Shape	Param #
nput_19 (InputLayer)	[(None, 10, 10, 1)]	0
conv2d_22 (Conv2D)	(None, 8, 8, 5)	50
activation_22 (Activation)	(None, 8, 8, 5)	0
conv2d_23 (Conv2D)	(None, 6, 6, 2)	92
activation_23 (Activation)	(None, 6, 6, 2)	0

Total params: 142
Trainable params: 142
Non-trainable params: 0

10*10 image

5 filters (3*3)

5 images (8*8)

Each filter has 10 parameters associated => 50 parameters

Parameters

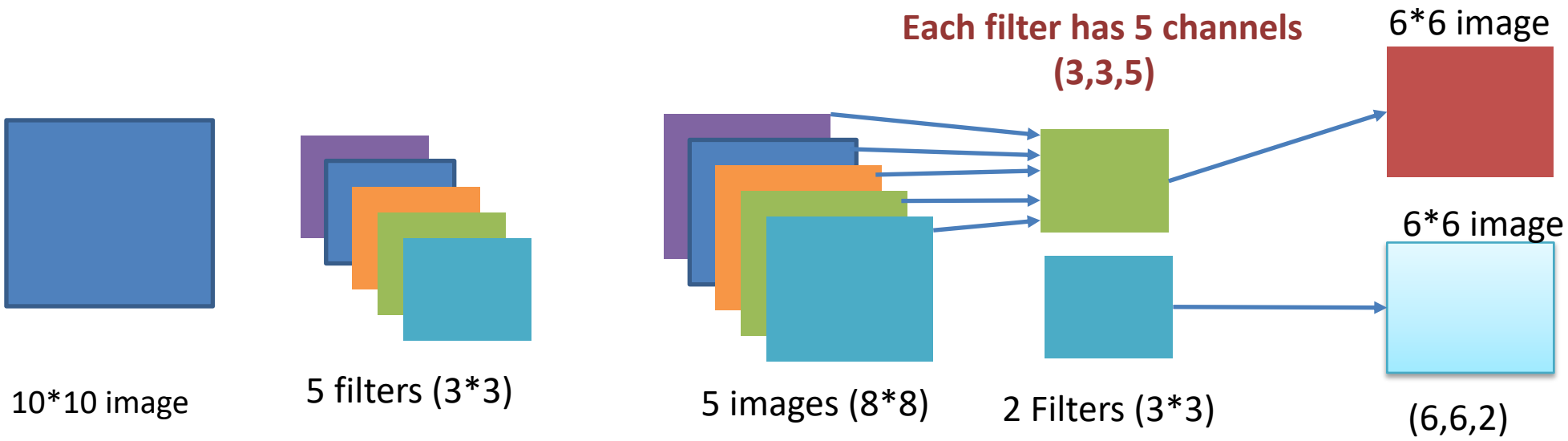
```
1 my_input = Input(shape = (10, 10, 1))
2 my_output = Convolution2D(5, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(2, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input_19 (InputLayer)        [(None, 10, 10, 1)]       0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 8, 8, 5)           50
_________________________________________________________________
activation_22 (Activation)   (None, 8, 8, 5)           0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 6, 6, 2)           92
_________________________________________________________________
activation_23 (Activation)   (None, 6, 6, 2)           0
=================================================================
Total params: 142
Trainable params: 142
Non-trainable params: 0
_________________________________________________________________
```

Input

- **Five 8*8 images (8*8*5)**
- **2 convolutional filters (3*3)**

Output?



Parameters

```
1 my_input = Input(shape = (10, 10, 1))
2 my_output = Convolution2D(5, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(2, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

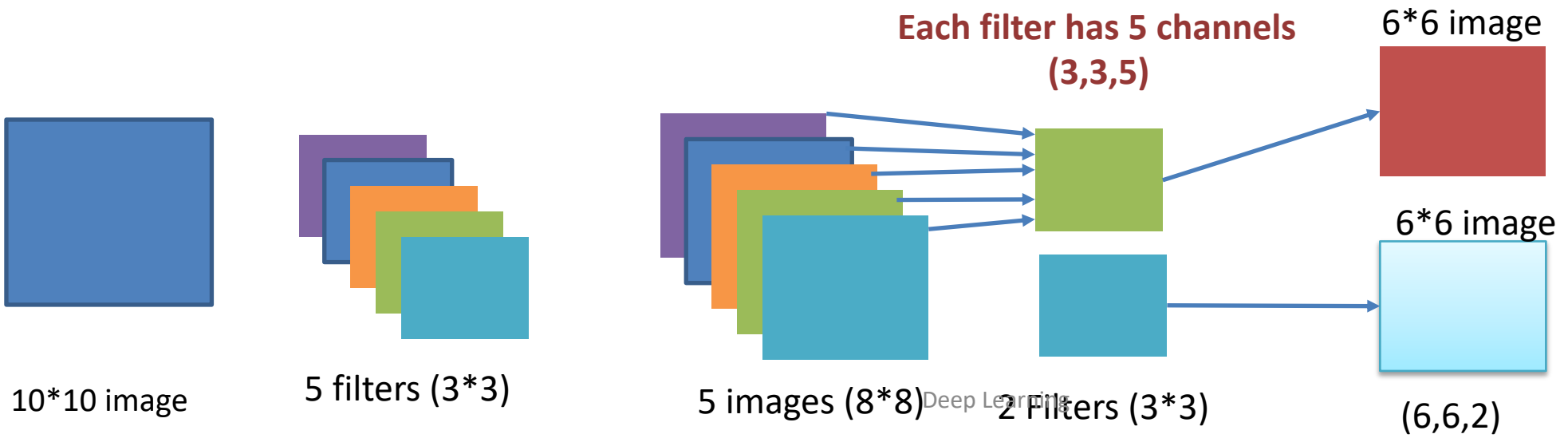
Layer (type)	Output Shape	Param #
input_19 (InputLayer)	[(None, 10, 10, 1)]	0
conv2d_22 (Conv2D)	(None, 8, 8, 5)	50
activation_22 (Activation)	(None, 8, 8, 5)	0
conv2d_23 (Conv2D)	(None, 6, 6, 2)	92
activation_23 (Activation)	(None, 6, 6, 2)	0

Total params: 142
Trainable params: 142
Non-trainable params: 0

Input

- **Five 8*8 images (8*8*5)**
- **2 convolutional filters (3*3)**

Output?



For each filter:
3*3*5 =45
We add the bias => 46

Two filters= **92**

Parameters

```
1 my_input = Input(shape = (100, 100, 3))
2 my_output = Convolution2D(8, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(1, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

Output?

- image of 100*100*3 (colored image)
- 8 convolutional filters of 3*3

$\Rightarrow$ 98*98*3 (number of filters)

$\Rightarrow$ How much parameters we have?

Parameters

```
1 my_input = Input(shape = (100, 100, 3))
2 my_output = Convolution2D(8, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(1, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

Output?

- image of 100*100*3 (colored image)
- 8 convolutional filters of 3*3

⇒ 98*98*3 (number of filters)

⇒ How much parameters we have?
⇒ The filters adapt to the image depth => the filters are 3*3*3 filters
⇒ 27 parameters + bias
⇒ 28
⇒ 28*8 => 224

Parameters

```
1 my_input = Input(shape = (100, 100, 3))
2 my_output = Convolution2D(8, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(1, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

Output?

- image of 100*100*3 (colored image)
- 8 convolutional filters of 3*3

⇒ 98*98*3 (number of filters)

⇒ How much parameters we have?

⇒ **The filters matchs the image depth** => 3*3*3 filters

⇒ 27 parameters + bias

⇒ 28

⇒ 28*8 => 224

Layer (type)	Output Shape	Param #
input_20 (InputLayer)	[(None, 100, 100, 3)]	0
conv2d_24 (Conv2D)	(None, 98, 98, 8)	224
activation_24 (Activation)	(None, 98, 98, 8)	0
conv2d_25 (Conv2D)	(None, 96, 96, 1)	73
activation_25 (Activation)	(None, 96, 96, 1)	0

Total params: 297
Trainable params: 297
Non-trainable params: 0

Parameters

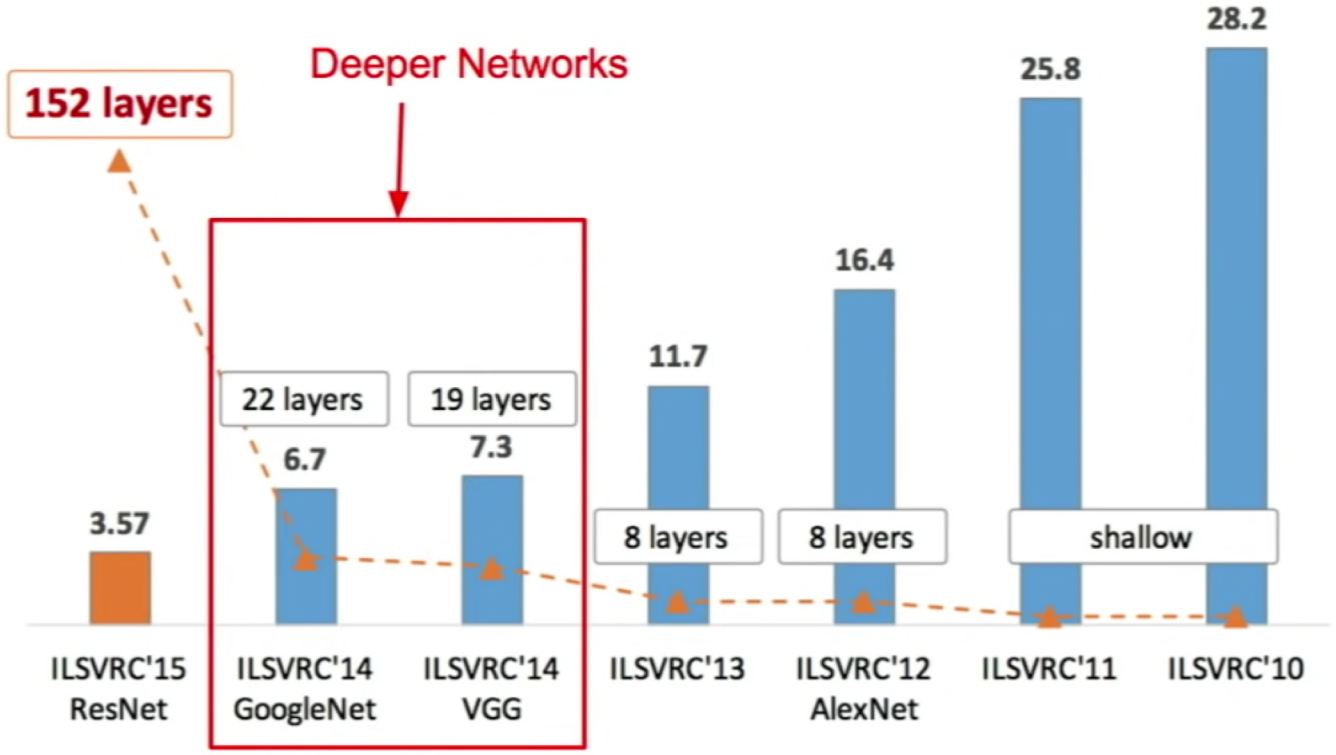
```
1 my_input = Input(shape = (100, 100, 3))
2 my_output = Convolution2D(8, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(1, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

How did we get 73?

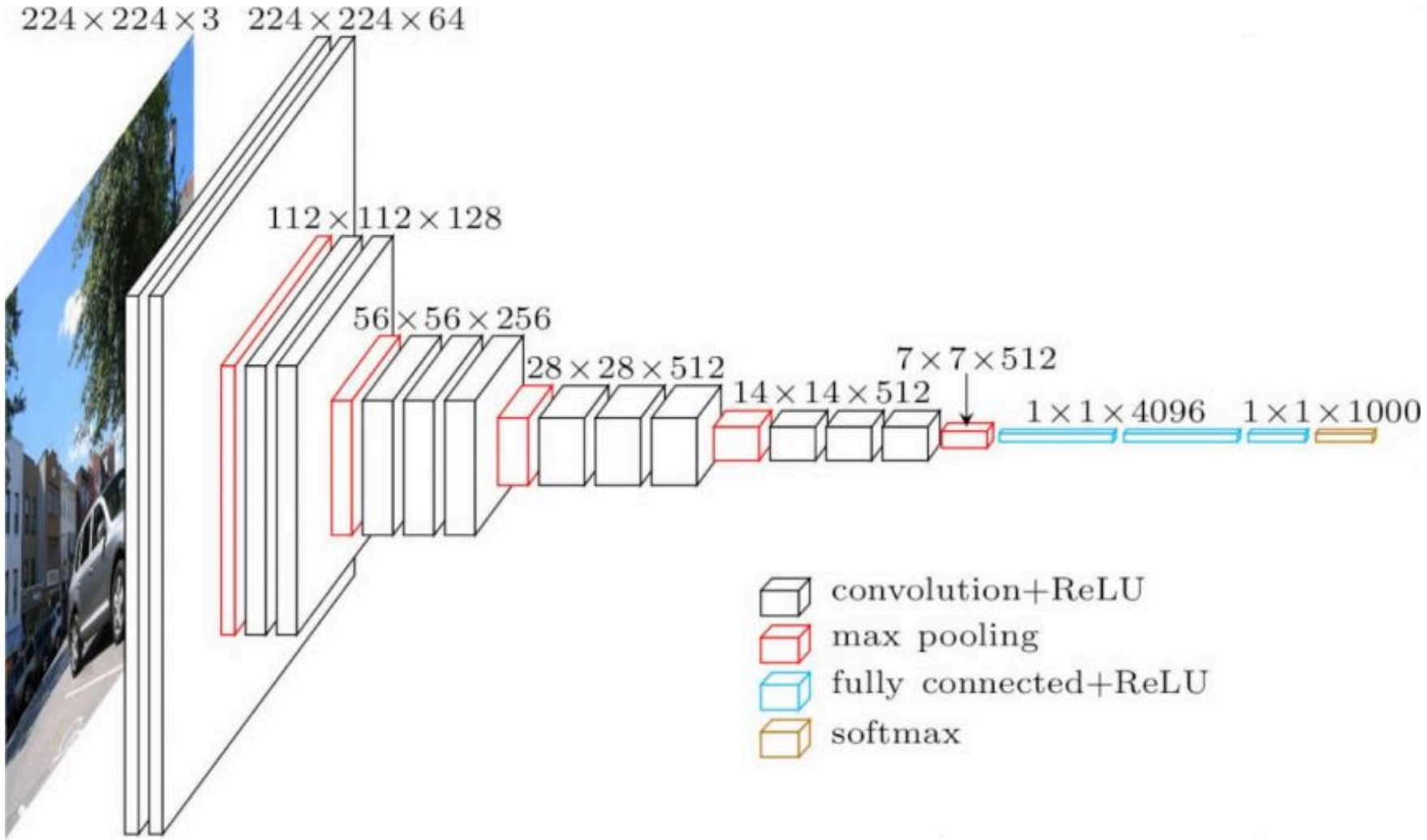
Layer (type)	Output Shape	Param #
input_20 (InputLayer)	[(None, 100, 100, 3)]	0
conv2d_24 (Conv2D)	(None, 98, 98, 8)	224
activation_24 (Activation)	(None, 98, 98, 8)	0
conv2d_25 (Conv2D)	(None, 96, 96, 1)	73
activation_25 (Activation)	(None, 96, 96, 1)	0

Total params: 297
Trainable params: 297
Non-trainable params: 0

Deeper Networks



VGG


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

VGG

Case Study: VGGNet

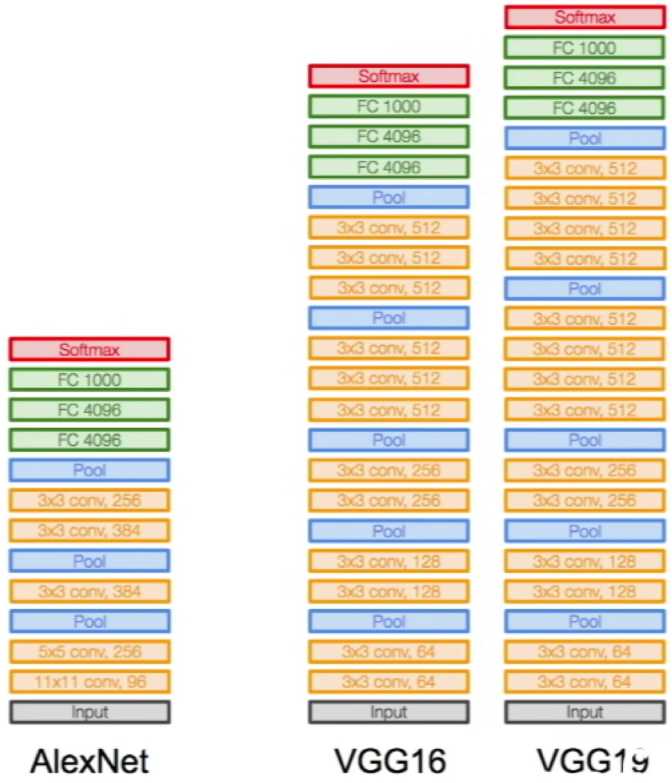
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

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

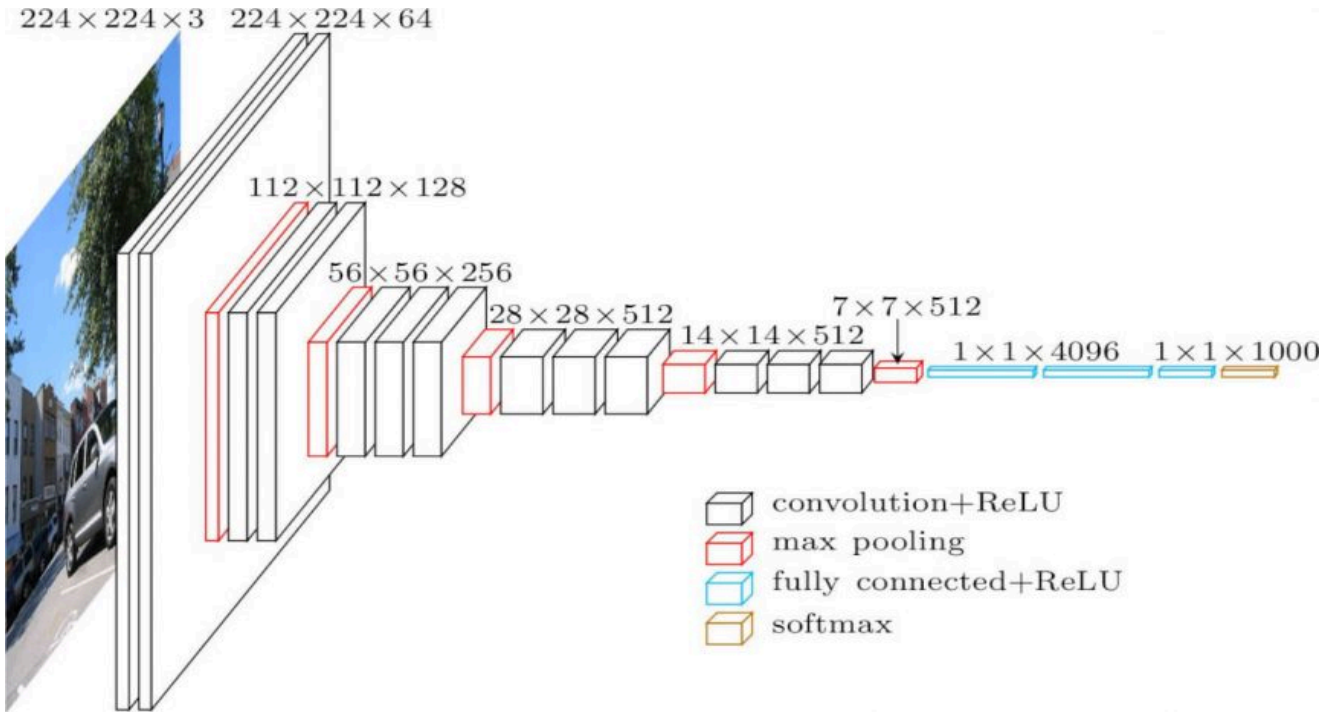
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

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



VGG

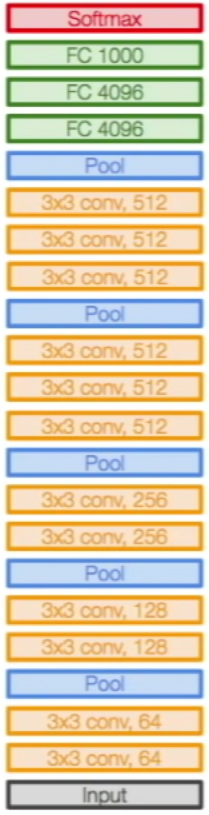
- Simplicity and depth (16/19 layers)
- 7.3% error rate
- The use of only 3*3 sized filters
- As the spatial size of the input volumes at each layer increases, the depth of the volume increases (increased number of filters)
- Trained on 4 Nvidia Titan BlackGPUs for Two weeks



VGG

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

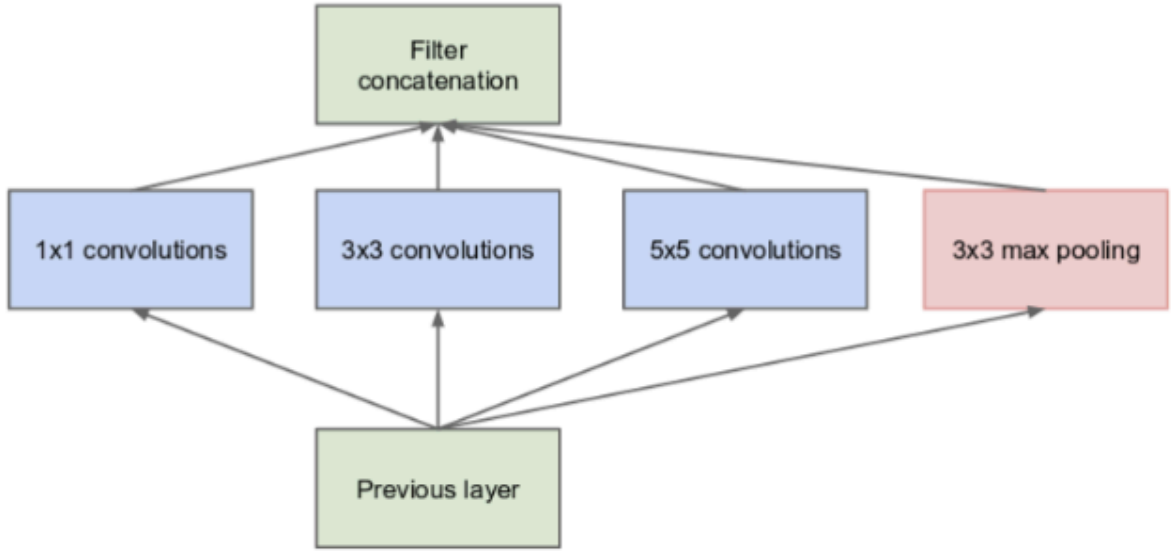
TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



VGG16

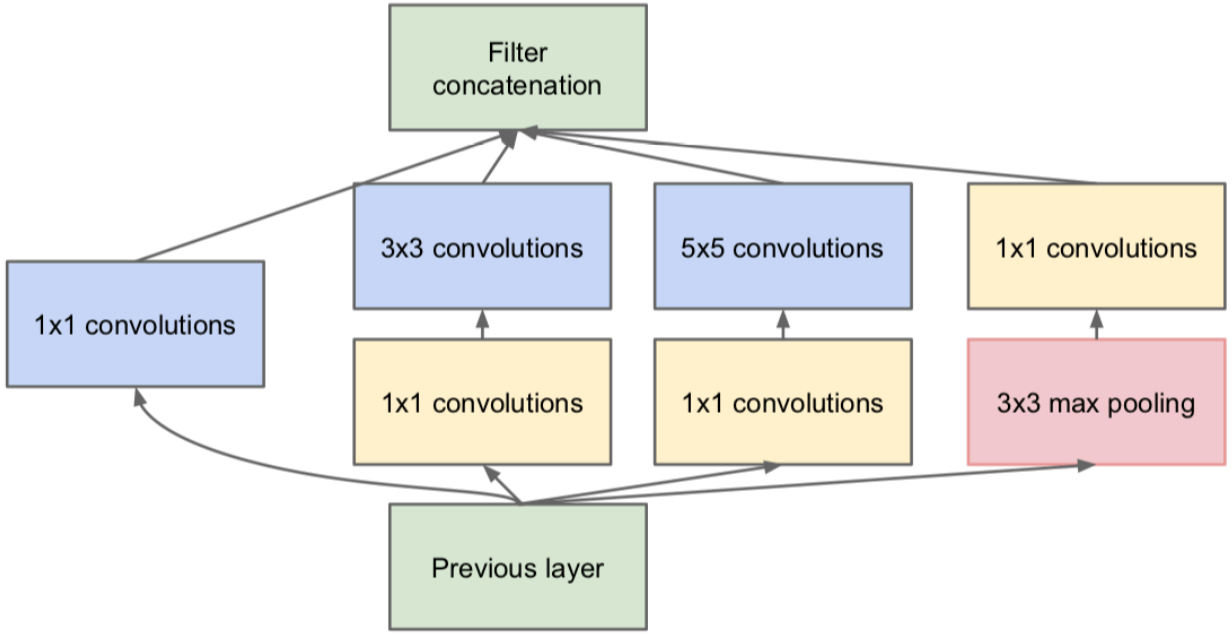
GoogleNet

- Top 5 error rate of 6.7%
- Key point: parallel convolutions with output concatennation
- 100 layers in total
- No use of fully connected layers
- Uses 12x fewer parameters than AlexNet (**only 5 million parameters**)



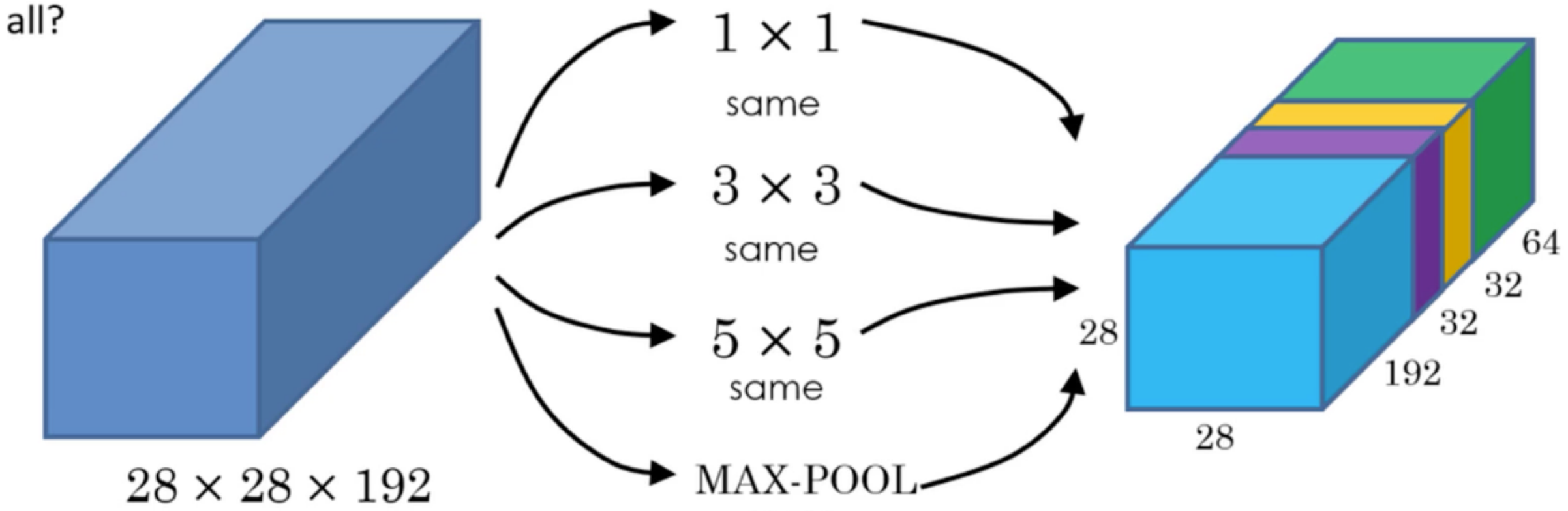
GoogleNet

Do you want to use 1*1 filter, or 3*3, or 5*5 ? Use a pooling layer? Or use them all together?



GoogleNet

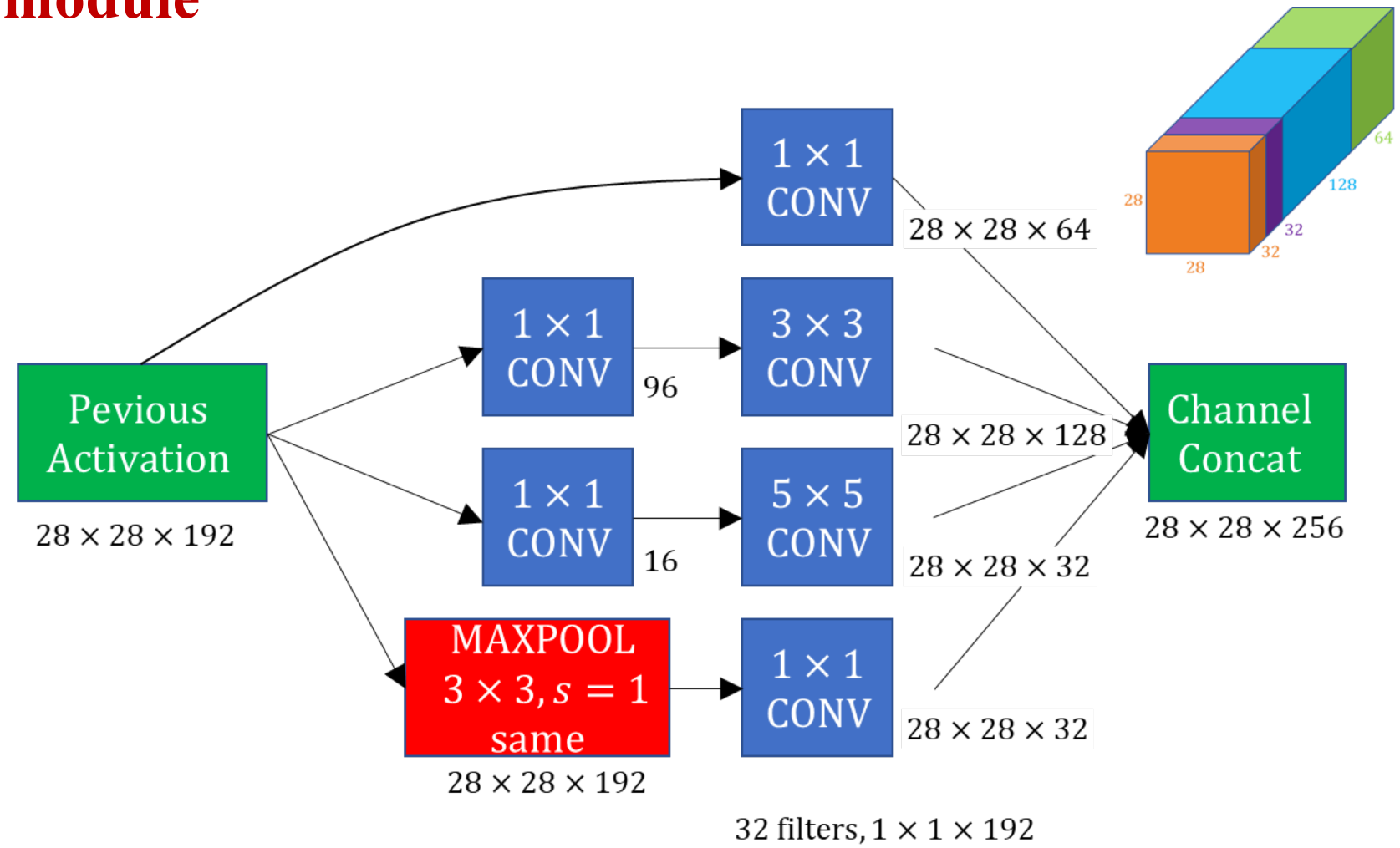
Do you want to use 1*1 filter, or 3*3, or 5*5 ? Use a pooling layer? Or use them all?



Same size=> padding =true

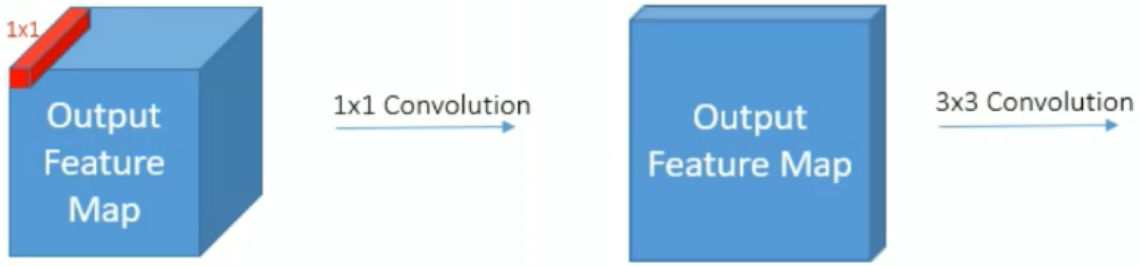
GoogleNet

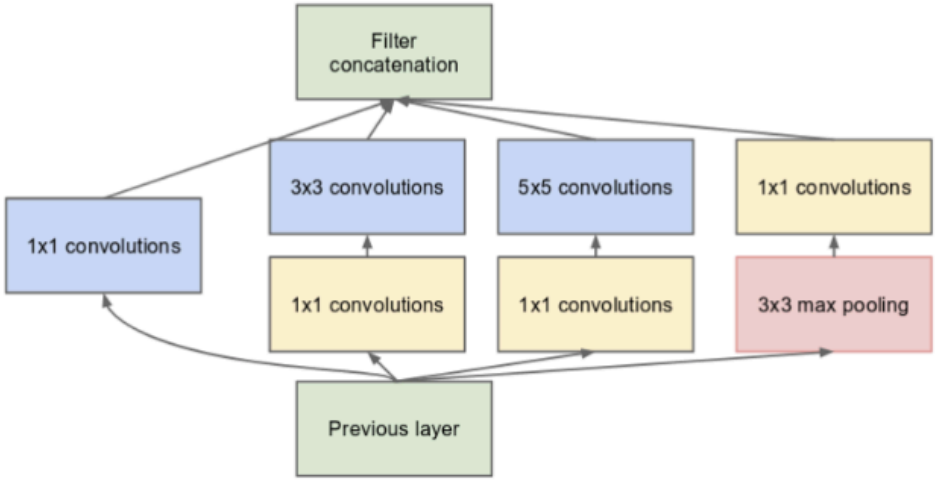
Inception module



GoogleNet

Reducing the depth while maintaining the size





The 1x1 convolutions are a method for **dimensionality reduction.**

Example: Consider an input volume of 100x100x60
Applying **30 filters** of 1x1 convolution would allow you to reduce the volume to 100x100x30 →The later convolution operations (such as 3x3 and 5x5 convolutions) won't have as large of a volume to deal with.

GoogleNet

The GoogleNet Architecture is 22 layers deep, with 27 pooling layers included. There are **9 inception modules** stacked linearly in total. The ends of the inception modules are connected to **the global average pooling layer**

Convolution
Pooling
Softmax
Other