Transfer Learning

Introduction to Deep learning

Agenda

1. What is Transfer learning (TL) ?
2. Comparison between normal training and TL
3. AlexNet
4. VGGNet
5. GoogleNet
6. ResNet

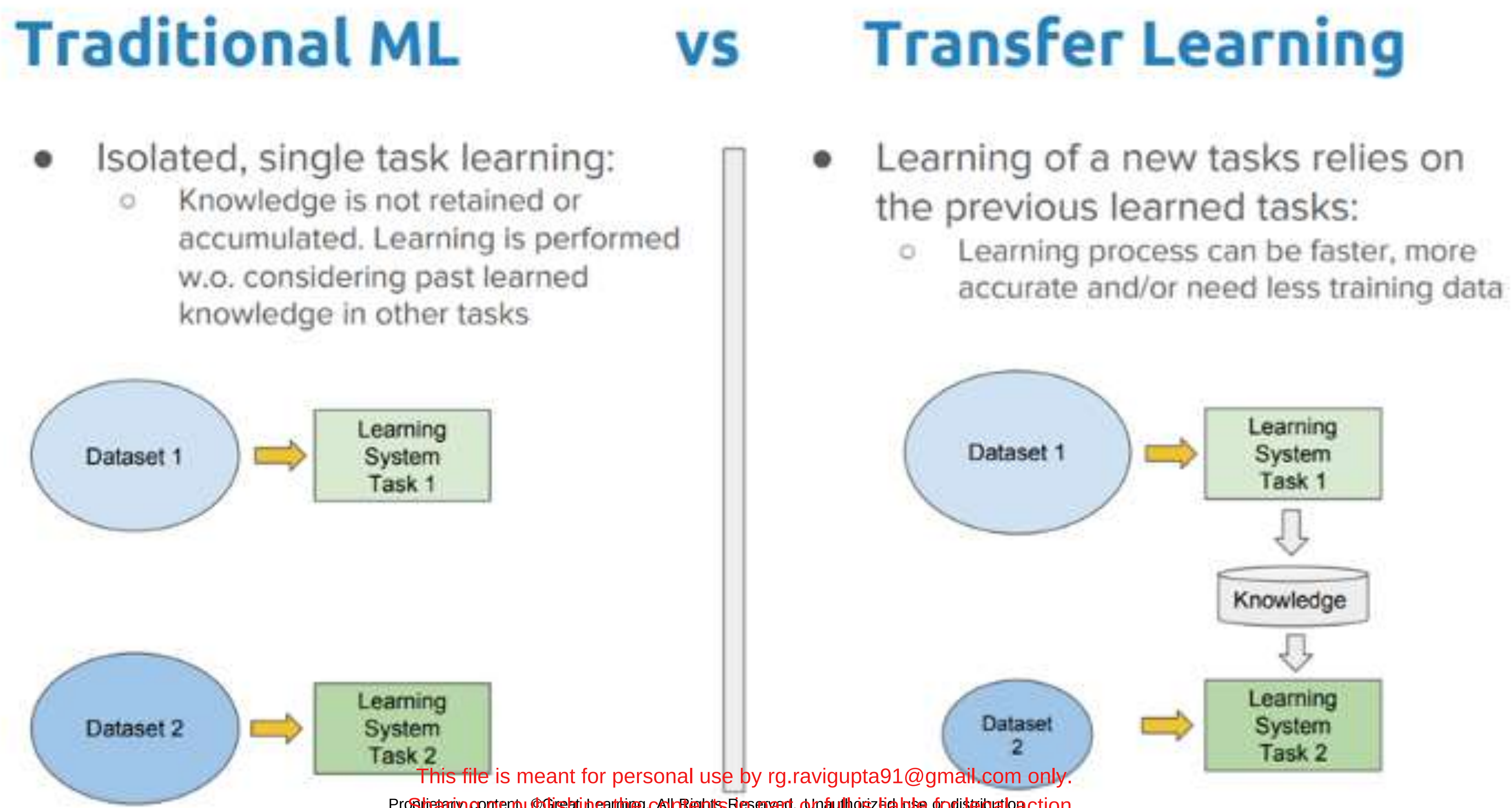
Introduction to Transfer Learning

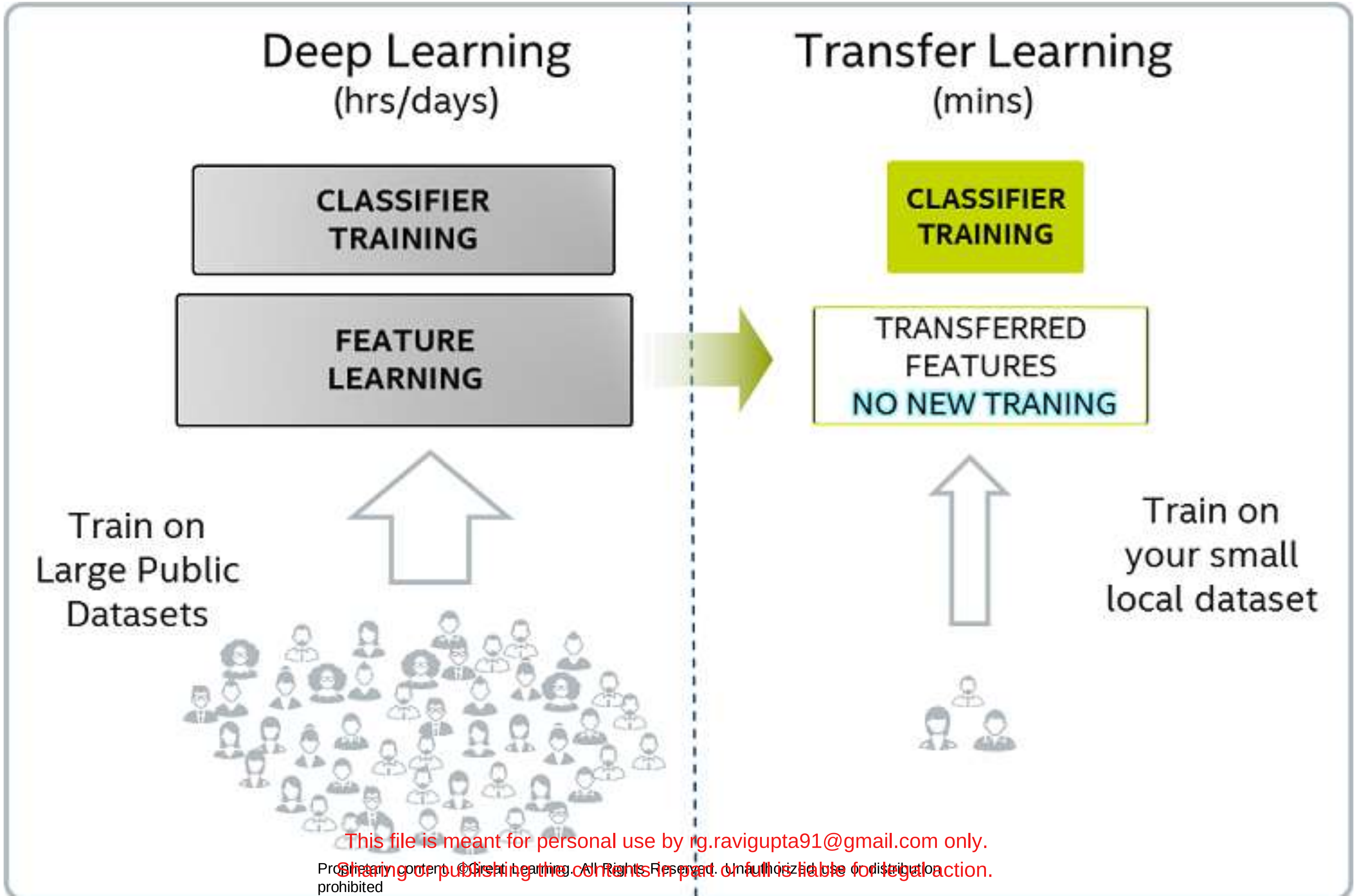
- Conventional machine learning and deep learning algorithms, so far, have been traditionally designed to work in isolation. These algorithms are trained to solve specific tasks.
- The models have to be rebuilt from scratch once the feature-space distribution changes.
- Transfer learning is the idea of overcoming the isolated learning paradigm and utilizing knowledge acquired for one task to solve related ones.
- After supervised learning — Transfer Learning will be the next driver of ML commercial success - Andrew NG
- Considering the context of deep learning is the fact that most models which solve complex problems need
  - a whole lot of data
  - vast amounts of labeled data

Introduction to Transfer Learning

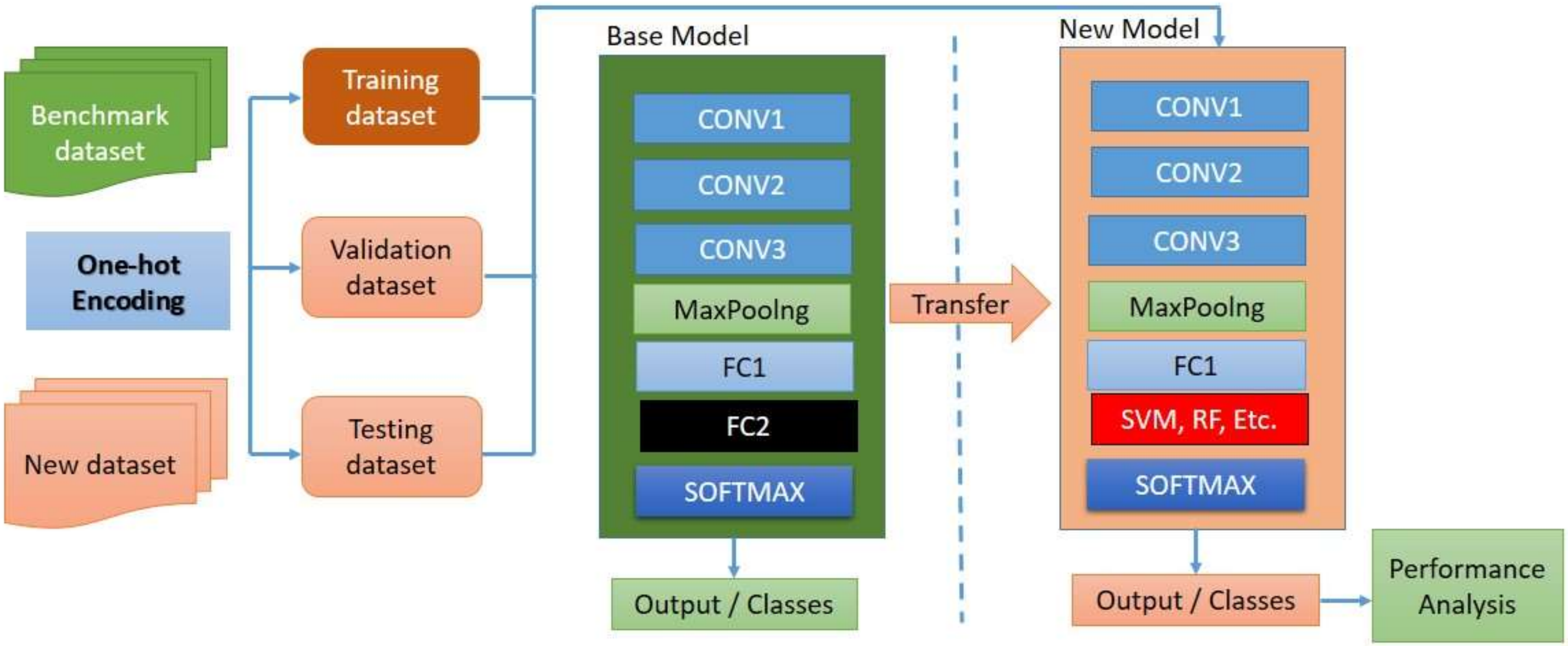
For supervised models getting labelled data can be really difficult, considering the time and effort it takes to label data points. A simple example would be the ImageNet dataset, which has millions of images pertaining to different categories, thanks to years of hard work starting at Stanford!

Introduction to Transfer Learning




Deep Learning
(hrs/days)
CLASSIFIER TRAINING
FEATURE LEARNING
Train on Large Public Datasets
Transfer Learning
(mins)
CLASSIFIER TRAINING
TRANSFERRED FEATURES
NO NEW TRANING
Train on your small local dataset

Transfer Learning: Base Model and the New Model



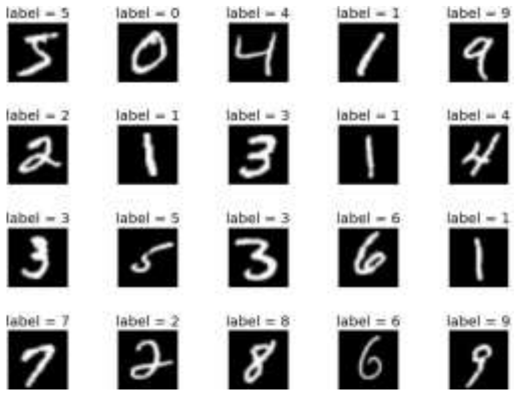
© Source: www.AmitRay.Com Fight Against COVID-19 AI Projects - Dr. Amit Ray

A Few CNN Case Studies

Case Studies

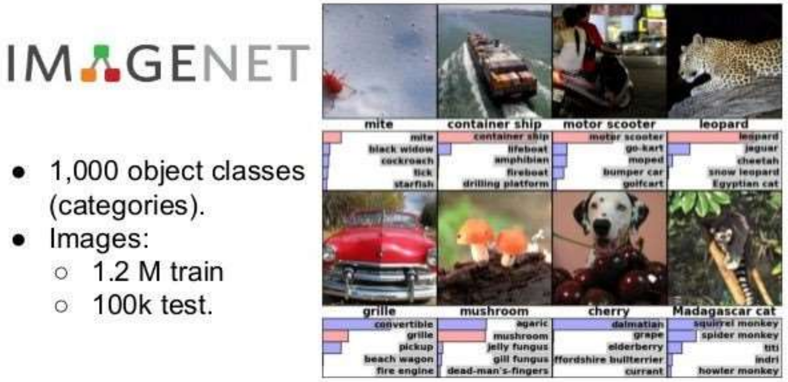
## 1.	Hand Written Digit Classification (LeNet - 1998)

input: a small single channel image
output: 10 outputs corresponding to the 10 digits 0-9.
60,000 training images,10,000 test images



## 2.	Image Net Classification – Annual world cup for CV

input: colored image
output:1000 outputs corresponding to the 1000 object classes in the dataset
1.2 M training images and 100,000 test images



Case Studies

CNNs on MNIST

1. LeNet (1998)
    - 10 way neural network classifier
    - Handwritten digits as an input
    - Tolerant of various transformations like rotation and scale
    - Was used by banks to recognize handwritten numbers on digitized checks
    - 4 weight layers

Case Studies

CNNs on ImageNet

1. AlexNet (2012)
   - First CNN to successfully be able classify ImageNet images
   - Improved benchmark performance (top-5) on this image dataset from 26% to 15%
   - 7 layers deep

2. ZF Net (2013)
   - Reduced the top-5 error rate to 11.2%
   - No major contributions
   - Also 7 layers deep

3. VGGNet (2014)
   - Simple and elegant
   - Reduced the top-5 error rate 7.2%
   - Did not win the competition, GoogleNet did!
   - 6 layers deep

Case Studies

CNNs on ImageNet

4. GoogleNet (2014)
   - 2014 imagenet winner with top-5 error rate of 6.7%
   - Used inception modules
   - 22 layers deep and used side cost functions

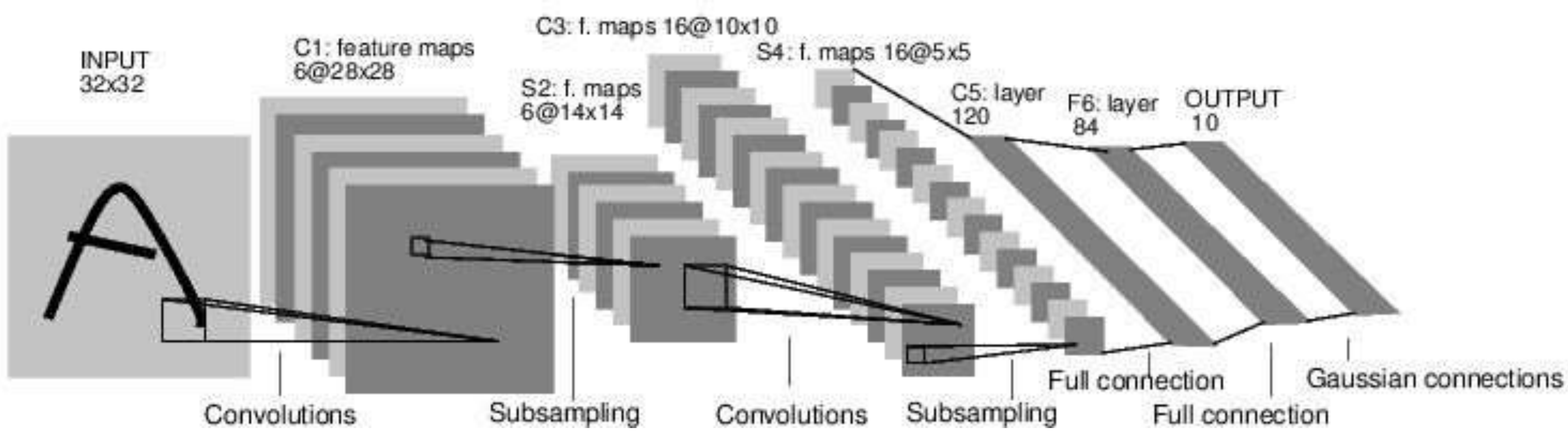
5. ResNet (2015)
   - 2015 imagenet winner with top-5 error rate of 3.57
   - First truly deep network with152 weight layers

6. CUImage (2016)
   - 2016 imagenet winner with top-5 error rate of 2.99
   - Ensemble approach, not very interesting

7. SENet (2017)
   - 2016 ImageNet winner with top-5 error rate of 2.251
   - Work by Momenta
   - The last ImageNet challenge!

Case Study: LeNet-5

[LeCun et al., 1998]



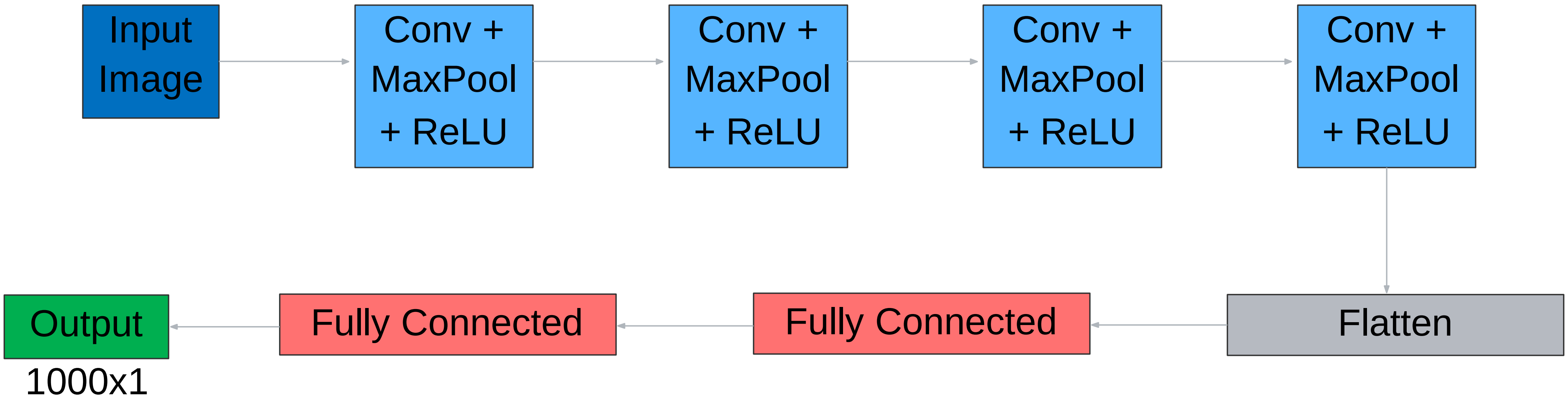
- Conv filters were 5x5, applied at stride 1
- Subsampling (Pooling) layers were 2x2 applied at stride 2
- Architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Source: Gradient Based Learning Applied to Document Recognition, LeCun et al. (1998)

Case Study: AlexNet

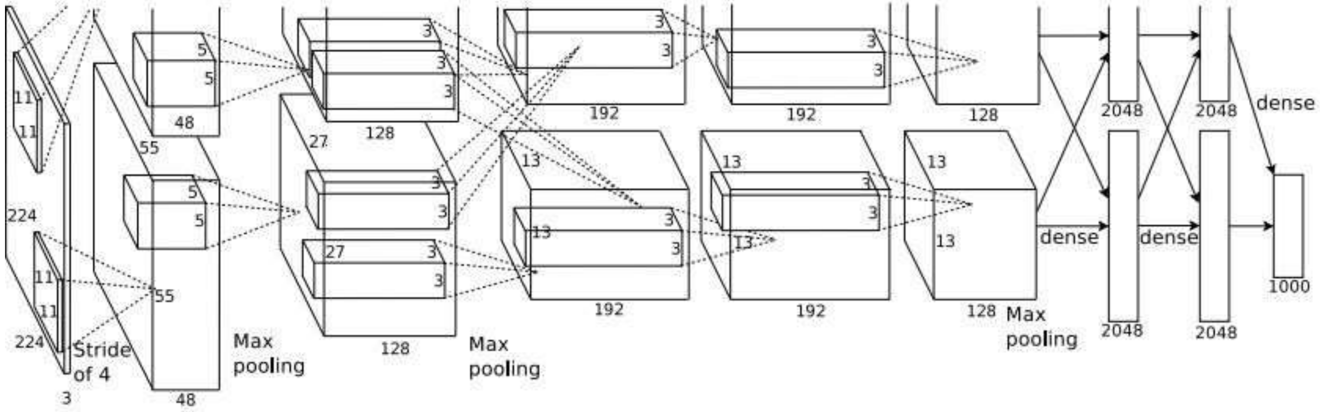
[Krizhevsky et al. 2012]

96



Case Study: AlexNet

[Krizhevsky et al. 2012]

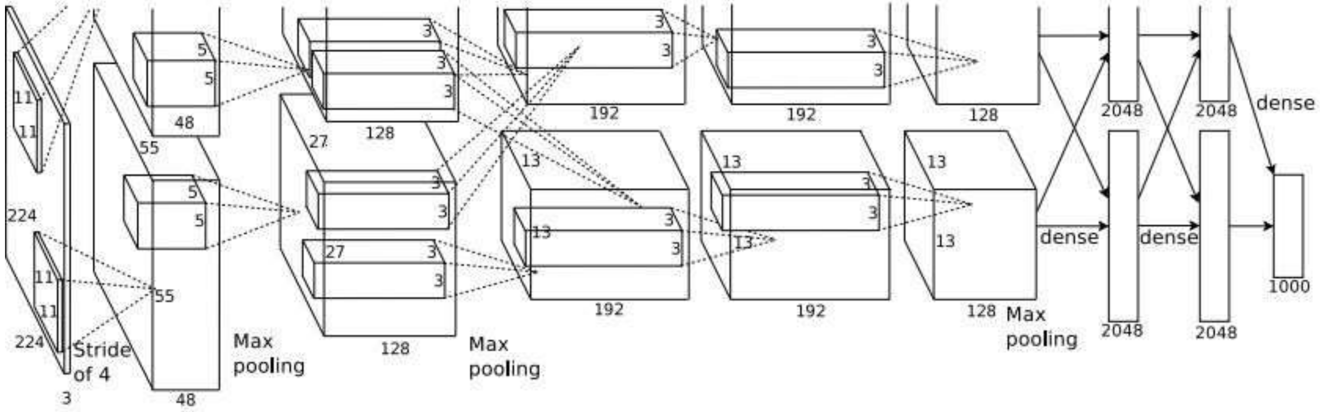


- **Input**: 227x227x3 images
- **First layer** (CONV1): 96 11x11 filters applied at stride 4
- Output volume size? (Hint: (227-11)/4+1)

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

Case Study: AlexNet

[Krizhevsky et al. 2012]

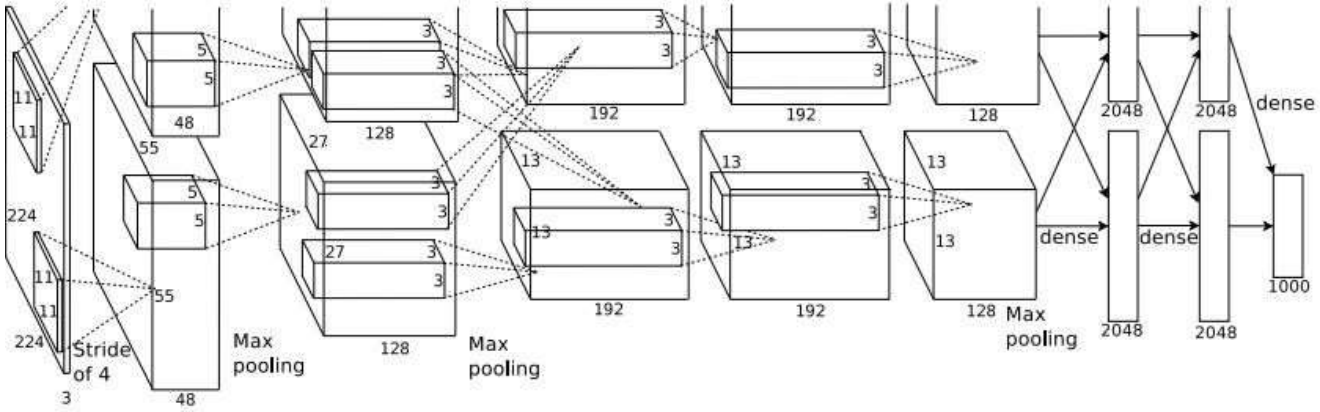


- **Input**: 227x227x3 images
- **First layer** (CONV1): 96 11x11 filters applied at stride 4
- **Output volume size**: (227-11)/4+1 = 55 for each H and W, so 55x55x96

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

Case Study: AlexNet

[Krizhevsky et al. 2012]

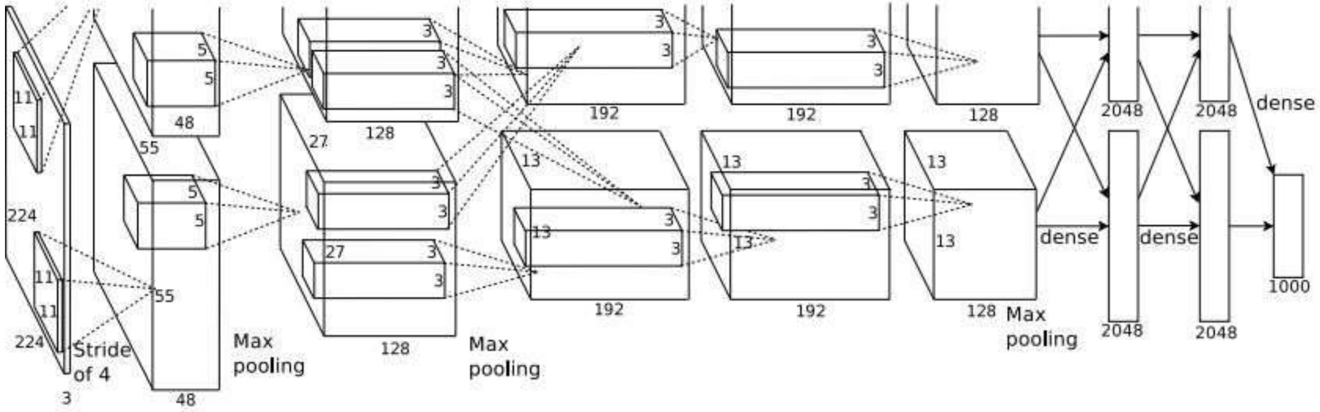


- **Input**: 227x227x3 images
- **First layer** (CONV1): 96 11x11 filters applied at stride 4
- **Output volume size**: 55x55x96
- Total number of parameters?

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

Case Study: AlexNet

[Krizhevsky et al. 2012]

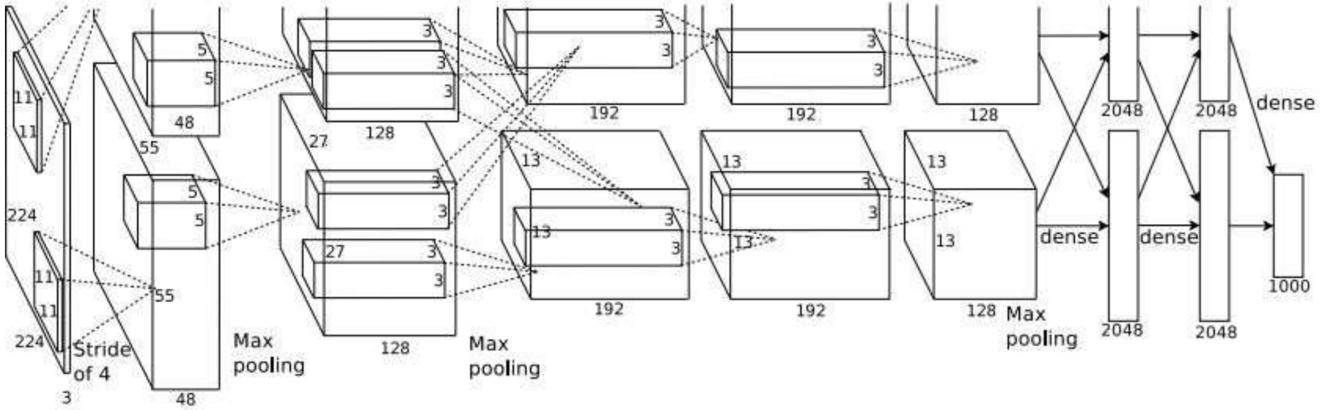


- **Input**: 227x227x3 images
- **First layer** (CONV1): 96 11x11 filters applied at stride 4
- **Output volume size**: 55x55x96
- **Total number of parameters**: (11*11*3)*96 = **35K**

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

Case Study: AlexNet

[Krizhevsky et al. 2012]



- **Input**: 227x227x3 images
- **After** CONV1: 55x55x96

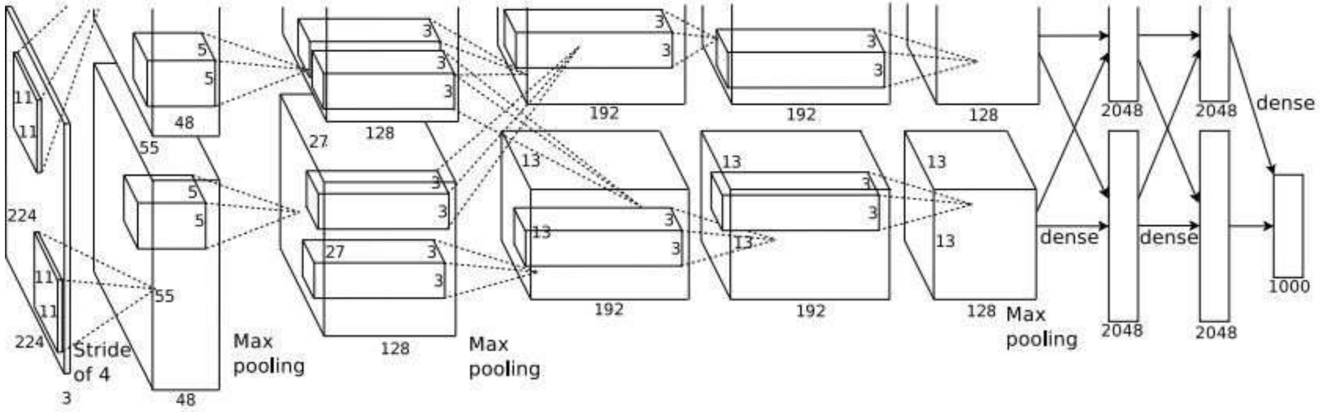
- **Second layer (POOL1): 3x3 filters applied**

- **What is the output volume size? (Hint: (55-3)/2+1 = 27)**

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

Case Study: AlexNet

[Krizhevsky et al. 2012]



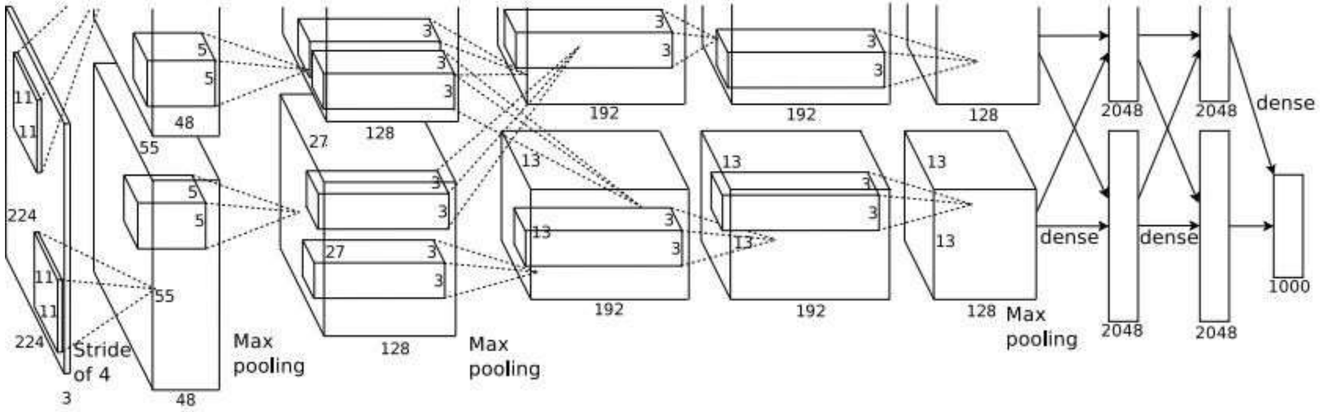
- **Input**: 227x227x3 images
- **After** CONV1: 55x55x96

- **Second layer** (POOL1): 3x3 filters applied applied at stride 2
- Output volume: 27x27x96

- What is the number of parameters?

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

Case Study: AlexNet
[Krizhevsky et al. 2012]



- **Input**: 227x227x3 images
- **After** CONV1: 55x55x96

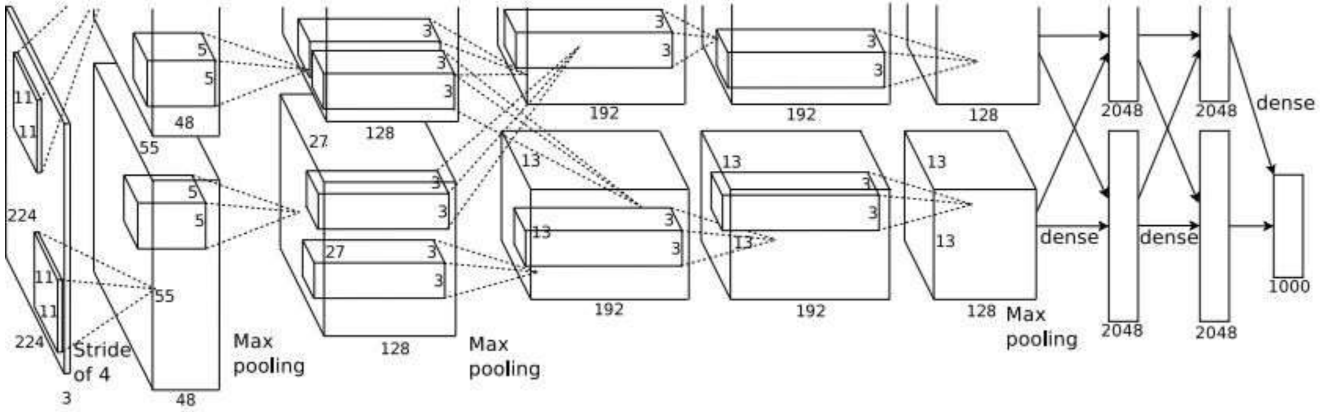
- **Second layer** (POOL1): 3x3 filters applied applied at stride 2
- Output volume: 27x27x96

- What is the number of parameters: **0!**

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

Case Study: AlexNet

[Krizhevsky et al. 2012]

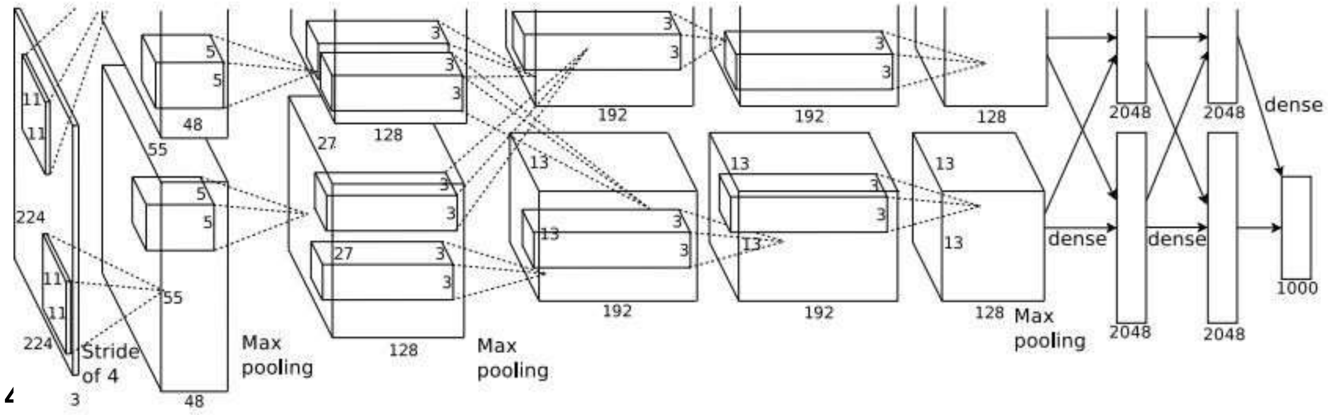


- **Input**: 227x227x3 images
- **After** CONV1: 55x55x96
- **After** POOL1: 27x27x96

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

Case Study: AlexNet

[Krizhevsky et al. 2012]



Architecture:

[227x227x3] **INPUT**

[55x55x96] **CONV1:** 96 11x11 filters at stride 4

0 [27x27x96] **MAX POOL1:** 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2:** 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2:** Normalization layer

[13x13x384] **CONV3:** 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4:** 384 3x3 filters at stride 1, pad 1

[13x13x256] **CONV5:** 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3:** 3x3 filters at stride 2

Finishing with:

[4096] **FC6:** 4096 neurons

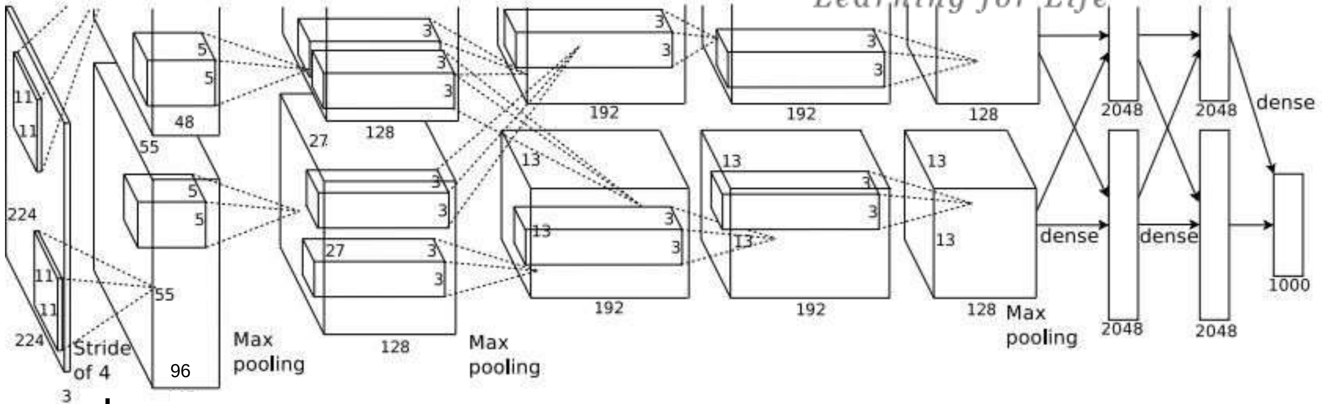
[4096] **FC7:** 4096 neurons

[1000] **FC8:** 1000 neurons (class scores)

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

Case Study: AlexNet

[Krizhevsky et al. 2012]



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

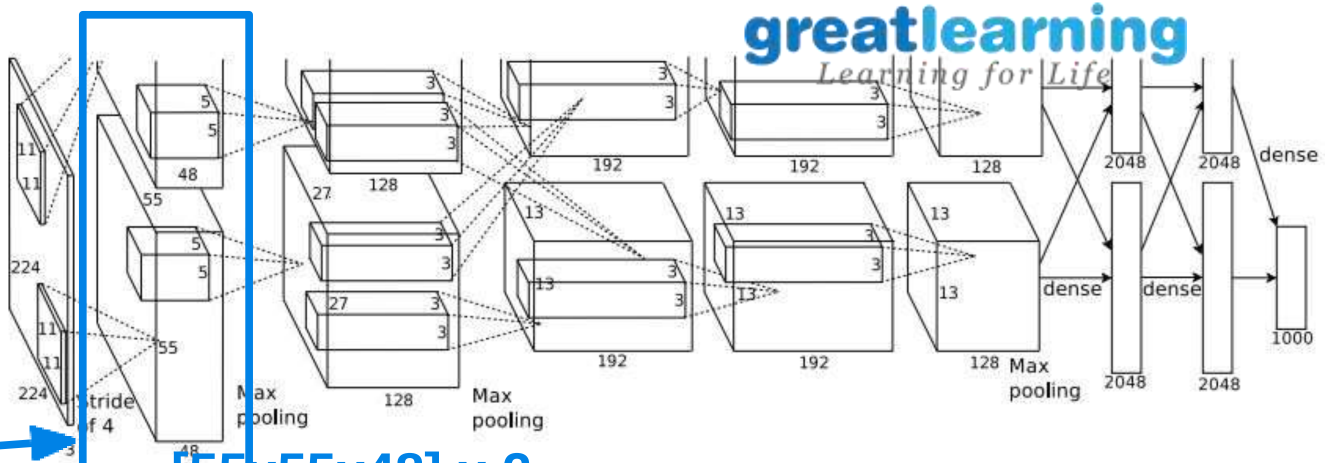
Salient points:

- Popularized use of ReLU in Vision
- Used Norm layers (not common anymore)
- Heavy data augmentation
- Dropout 0.5 in only last few fully-connected
- Batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10
- Manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: ***18.2% improved to 15.4%***

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

Case Study: AlexNet

[Krizhevsky et al. 2012]



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

Historical Note:

Trained on GTX580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the feature maps on each GPU.

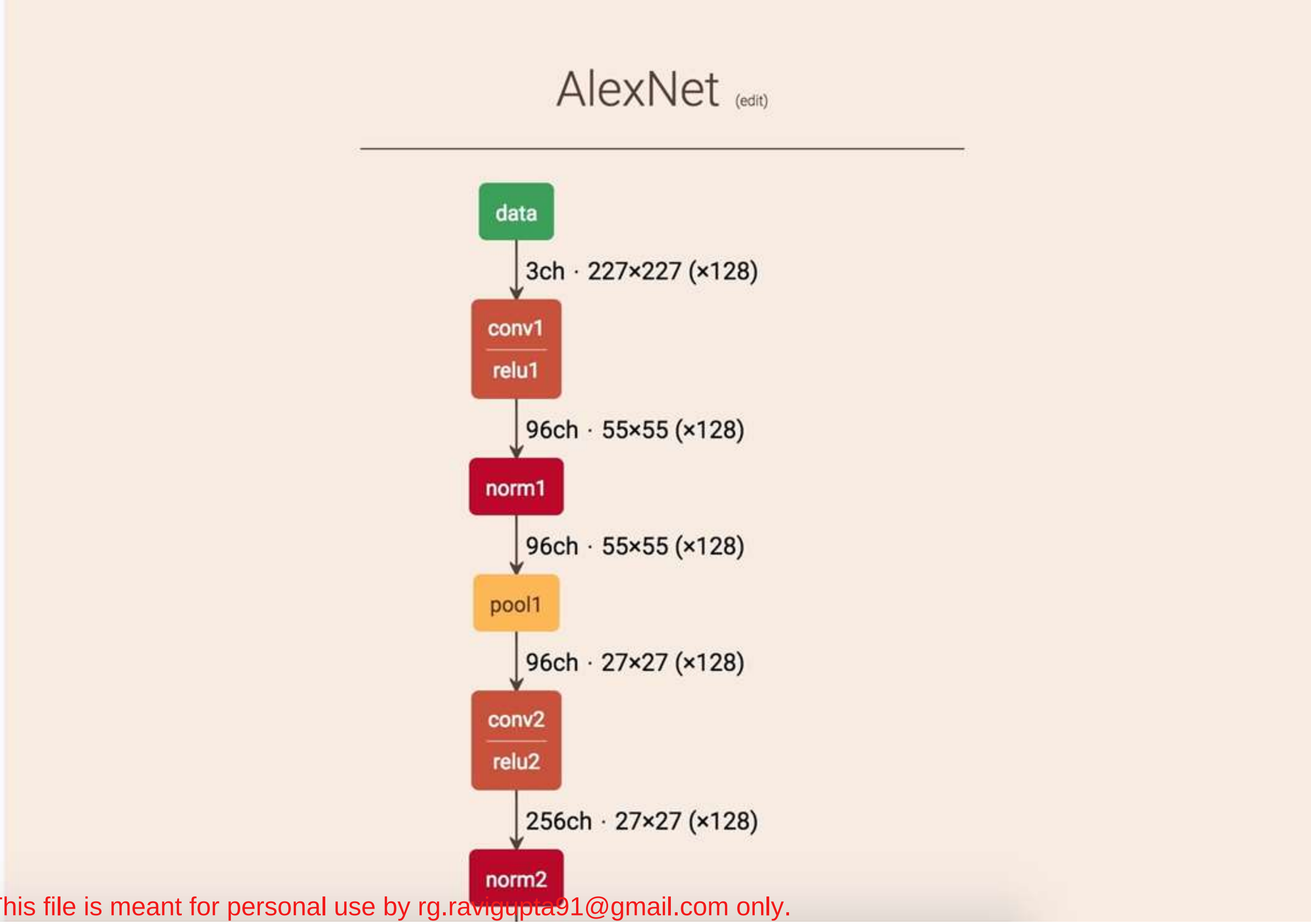
[1000] **FC8:** 1000 neurons (class scores)

Sourced with permission from: ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

A tool to analyze deep networks

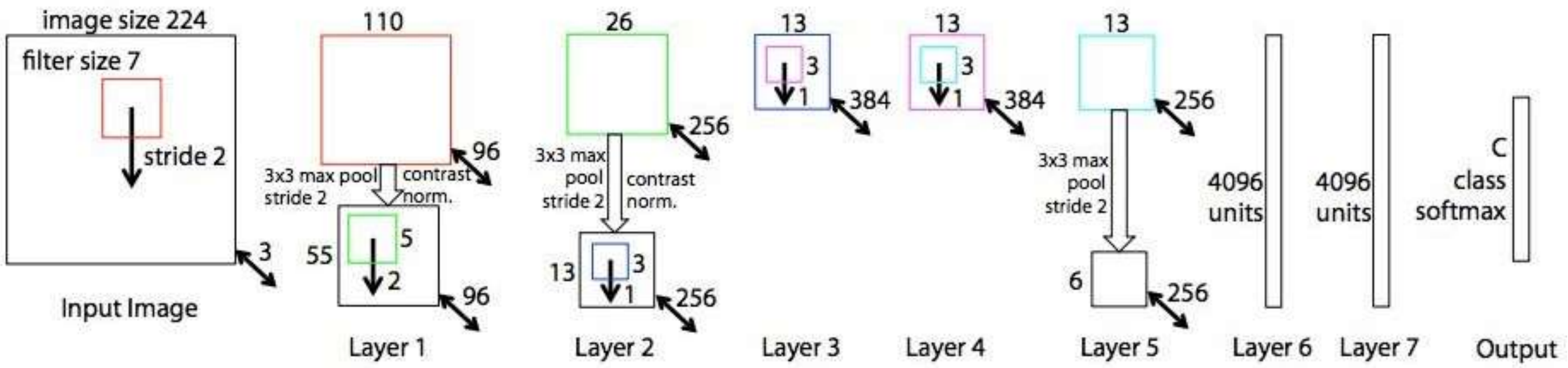
http://dgschwend.github.io/netscope/#/editor

```
name: "AlexNet"
layer {
  name: "data"
  type: "Data"
  top: "data"
  input_param {
      shape: {
          dim: 128
          dim: 3
          dim: 227
          dim: 227
      }
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
```



Case Study - ZFNet

[Zeiler and Fergus, 2013]



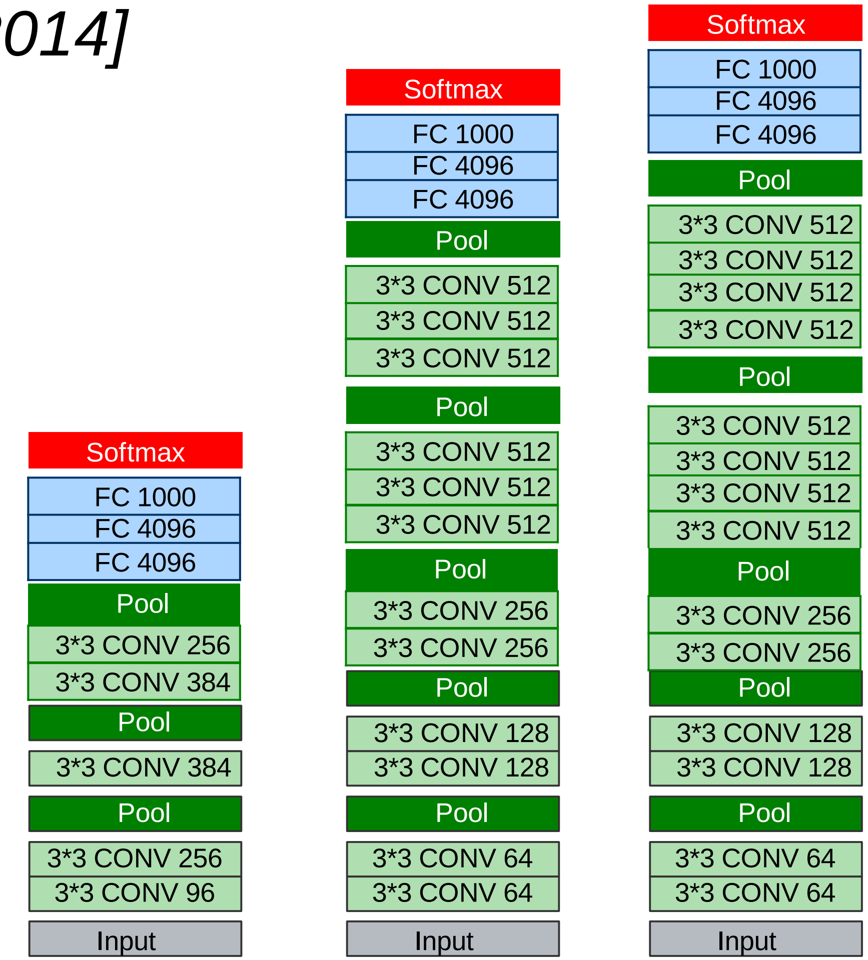
Similar to AlexNet with the following differences:
CONV1: (7x7 stride 2) instead of (11x11 stride 4)
CONV3,4,5: 512, 1024, 512 filters instead of 384, 384, 256 respectively

Reduced top 5 error on ImageNet
From **15.4%** To **14.8%**
Later brought down to 11.2%

Sourced with permission from 'Visualizing and Understanding Convolutional Networks' by Zeiler, Fergus (2013)

Case Study: VGGNet

[Simonyan and Zisserman, 2014]



AlexNet **VGG 16** **VGG 19**

This model used:

- **Smaller filters**
  But
- **Deeper networks**

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

Why use smaller filters? (3x3 conv)

Answer: Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 but deeper, more non-linearities and fewer parameters.

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

This model used:

- **Smaller filters**

  But
- **Deeper networks**

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

Improved from 11.2% top 5 error
in ILSVRC 2013
To **7.3% top 5 error**

And yet, this model did not win!

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					

Best performing model

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

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

Sourced with permission from: 'Very deep large-scale image recognition, Simonyan & Zisserman (2015)

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

INPUT: [224x224x3]
CONV3-64: [224x224x64]
CONV3-64: [224x224x64]
POOL2: [112x112x64]
CONV3-128: [112x112x128]
CONV3-128: [112x112x128]
POOL2: [56x56x128]
CONV3-256: [56x56x256]
CONV3-256: [56x56x256]
CONV3-256: [56x56x256]
POOL2: [28x28x256]
CONV3-512: [28x28x512]
CONV3-512: [28x28x512]
CONV3-512: [28x28x512]
POOL2: [14x14x512]
CONV3-512: [14x14x512]
CONV3-512: [14x14x512]
CONV3-512: [14x14x512]
POOL2: [7x7x512]
FC: [1x1x4096]
FC: [1x1x4096]
FC: [1x1x1000]

MEMORY

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

INPUT: [224x224x3]	224*224*3=150K
CONV3-64: [224x224x64]	224*224*64=3.2M
CONV3-64: [224x224x64]	224*224*64=3.2M
POOL2: [112x112x64]	112*112*64=800K
CONV3-128: [112x112x128]	112*112*128=1.6M
CONV3-128: [112x112x128]	112*112*128=1.6M
POOL2: [56x56x128]	56*56*128=400K
CONV3-256: [56x56x256]	56*56*256=800K
CONV3-256: [56x56x256]	56*56*256=800K
CONV3-256: [56x56x256]	56*56*256=800K
POOL2: [28x28x256]	28*28*256=200K
CONV3-512: [28x28x512]	28*28*512=400K
CONV3-512: [28x28x512]	28*28*512=400K
CONV3-512: [28x28x512]	28*28*512=400K
POOL2: [14x14x512]	14*14*512=100K
CONV3-512: [14x14x512]	14*14*512=100K
CONV3-512: [14x14x512]	14*14*512=100K
CONV3-512: [14x14x512]	14*14*512=100K
POOL2: [7x7x512]	7*7*512=25K
FC: [1x1x4096]	4096
FC: [1x1x4096]	4096
FC: [1x1x1000]	1000

Total memory:

24M * 4 bytes ~= 93MB/image

Only for forward. What if we include backward?

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

total parameters!

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

MEMORY

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

INPUT: [224x224x3]
CONV3-64: [224x224x64]
CONV3-64: [224x224x64]
POOL2: [112x112x64]
CONV3-128: [112x112x128] CONV3-128: [112x112x128]
POOL2: [56x56x128]
CONV3-256: [56x56x256]
CONV3-256: [56x56x256]
CONV3-256: [56x56x256]
POOL2: [28x28x256]
CONV3-512: [28x28x512]
CONV3-512: [28x28x512]
CONV3-512: [28x28x512]
POOL2: [14x14x512]
CONV3-512: [14x14x512]
CONV3-512: [14x14x512]
CONV3-512: [14x14x512]
POOL2: [7x7x512]
FC: [1x1x4096]
FC: [1x1x4096]
FC: [1x1x1000]

224*224*3=150K
224*224*64=3.2M
224*224*64=3.2M
112*112*64=800K
112*112*128=1.6 M
112*112*128=1.6 M
56*56*128=400K
56*56*256=800K
56*56*256=800K
56*56*256=800K
28*28*256=200K
28*28*512=400K
28*28*512=400K
28*28*512=400K
14*14*512=100K
14*14*512=100K
14*14*512=100K
14*14*512=100K
7*7*512=25K
4096
4096
1000

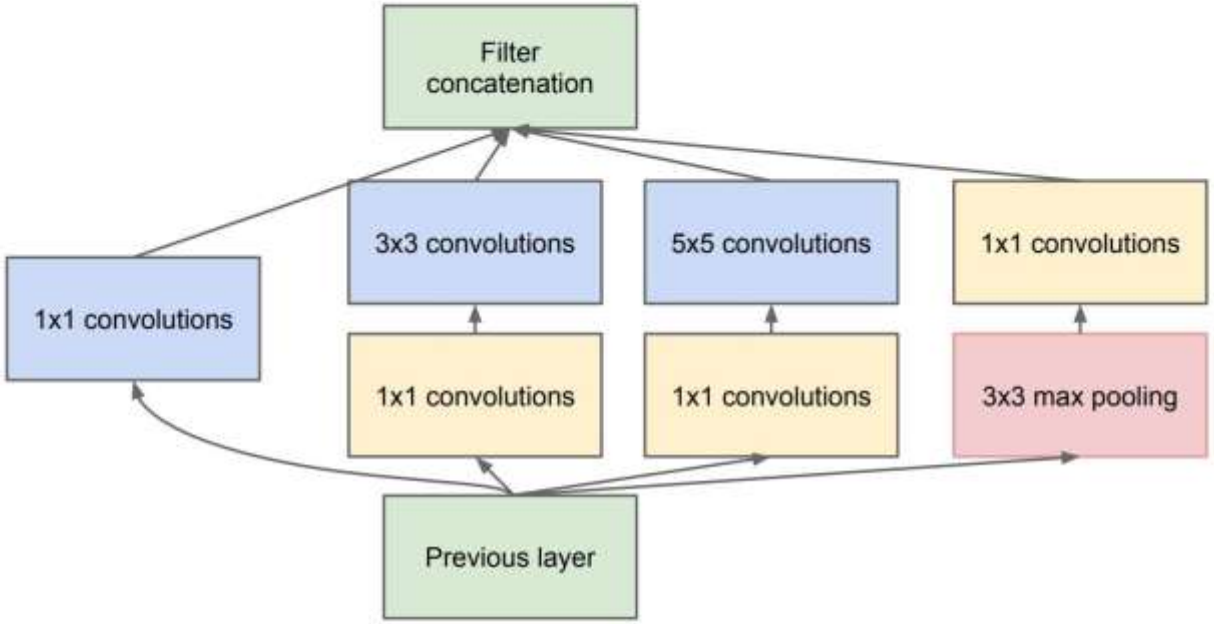
Most memory in early CONV layers

ConvNet Configuration			
B	C	D	19
13 weight layers	16 weight layers	16 weight layers	
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

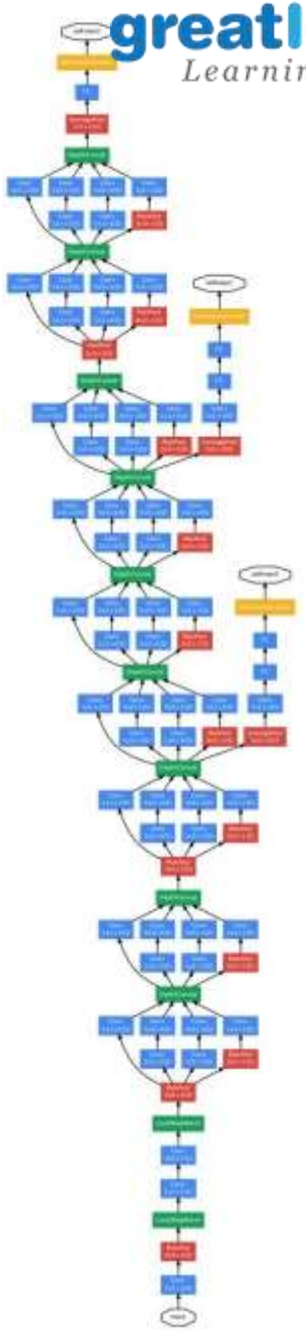
	MEMORY	PARAMETERS
INPUT: [224x224x3]	224*224*3=150K	0
CONV3-64: [224x224x64]	224*224*64=3.2M	(3*3*3)*64 = 1,728
CONV3-64: [224x224x64]	224*224*64=3.2M	(3*3*64)*64 = 36,864
POOL2: [112x112x64]	112*112*64=800K	0
CONV3-128: [112x112x128]	112*112*128=1.6M	(3*3*64)*128 = 73,728
CONV3-128: [112x112x128]	112*112*128=1.6M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

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module – with dimension reductions



The full GoogLeNet

Winner of ILSVRC 2014 with **6.7%** top 5 error

Sourced with permission from: 'Going Deeper with Convolutions', Szegedy et al. (2014)

Case Study: GoogLeNet

[Szegedy et al., 2014]

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								

This model has only 5 million parameters! (Removes FC layers completely)

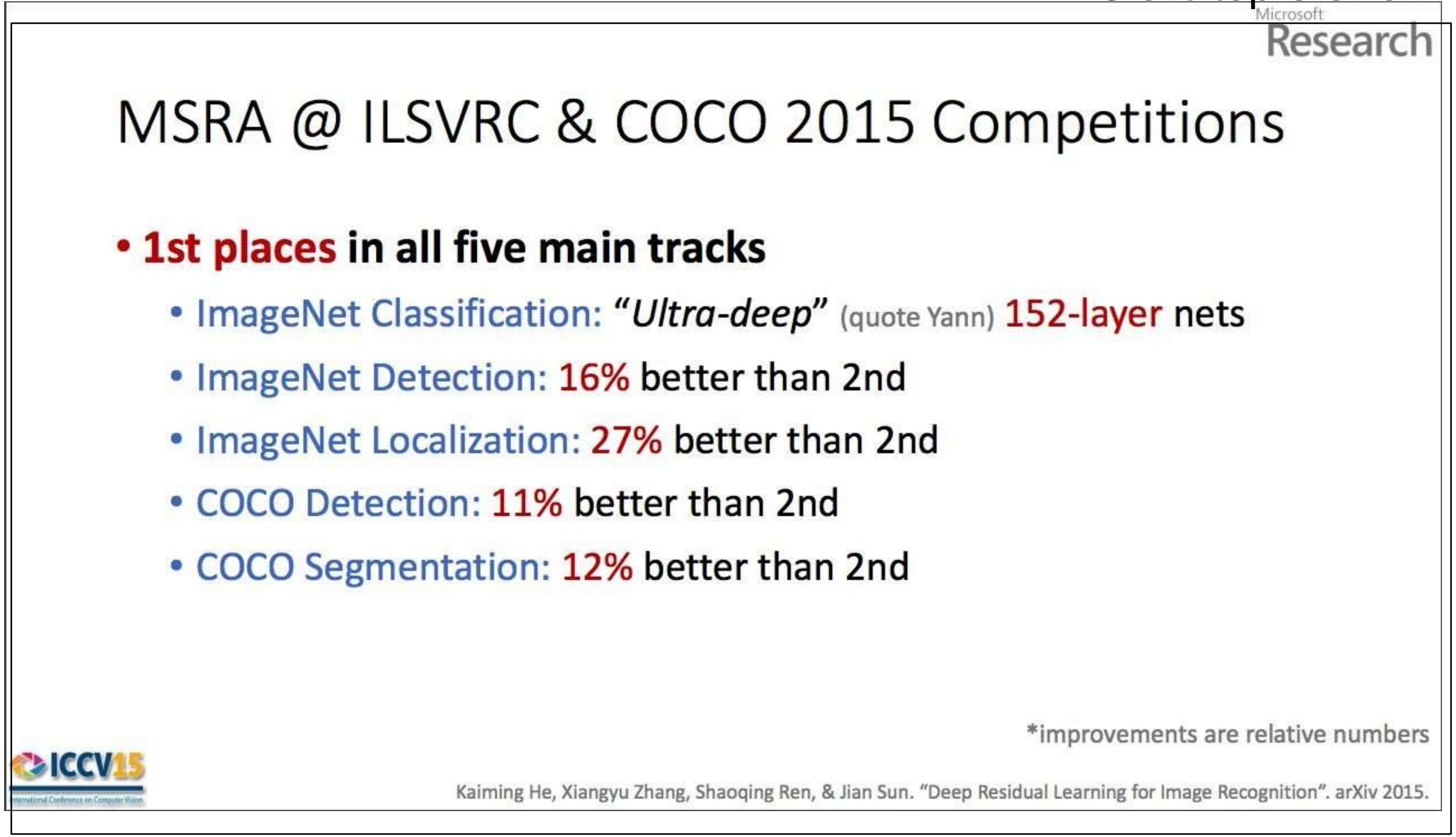
Compared to AlexNet, this model has: 12X less params | 2x more compute | 6.67% top-5 error rate vs. 16.4%

Sourced with permission from: 'Going Deeper with Convolutions', Szegedy et al. (2014)

Case Study: ResNet

[He et al., 2015]

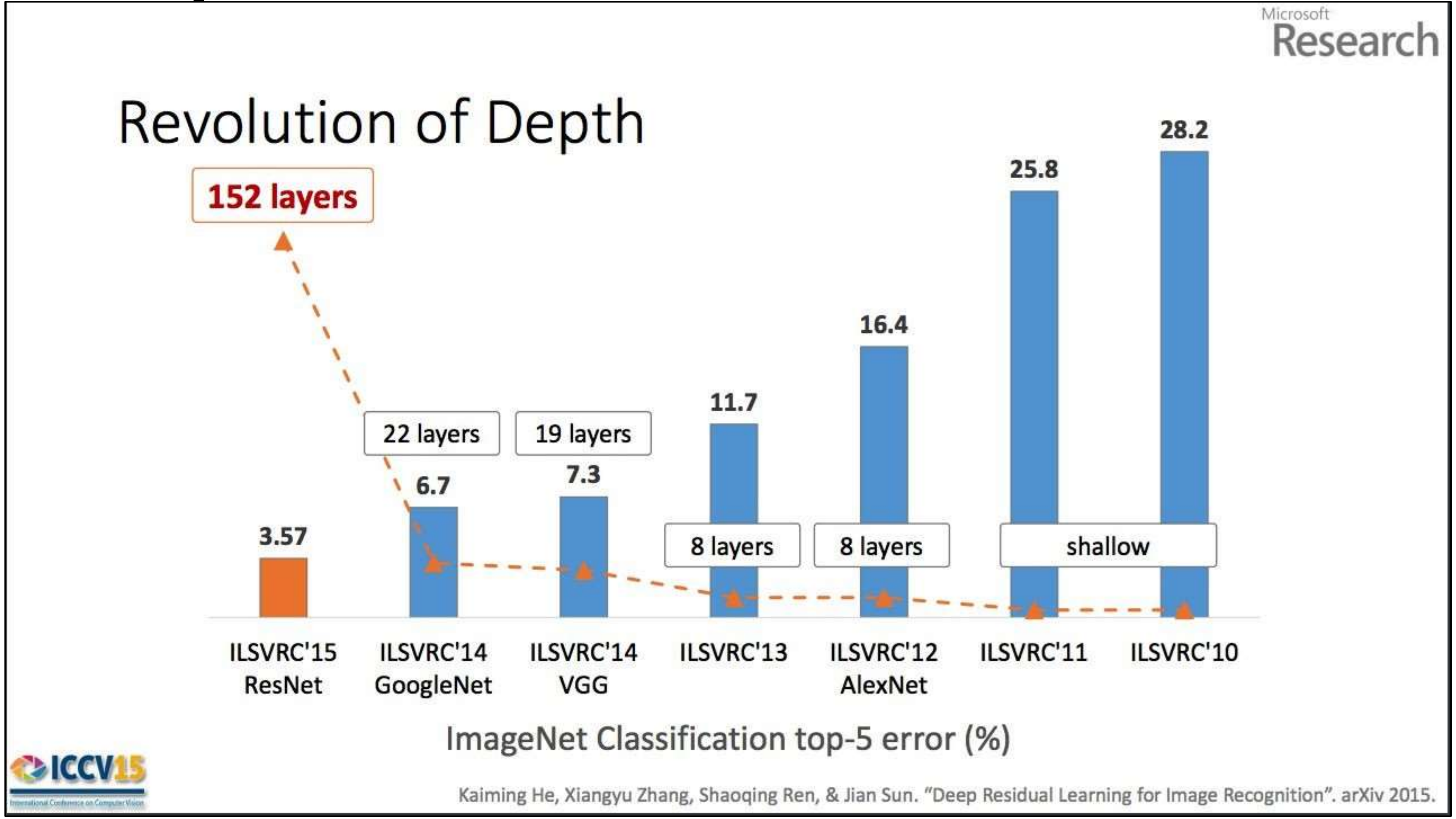
Winner of ILSVRC 2015
3.6% top-5 error!



Sourced with permission from: Deep Residual Learning for Image Recognition, Kaiming He (2015)

Case Study: ResNet

[He et al., 2015]

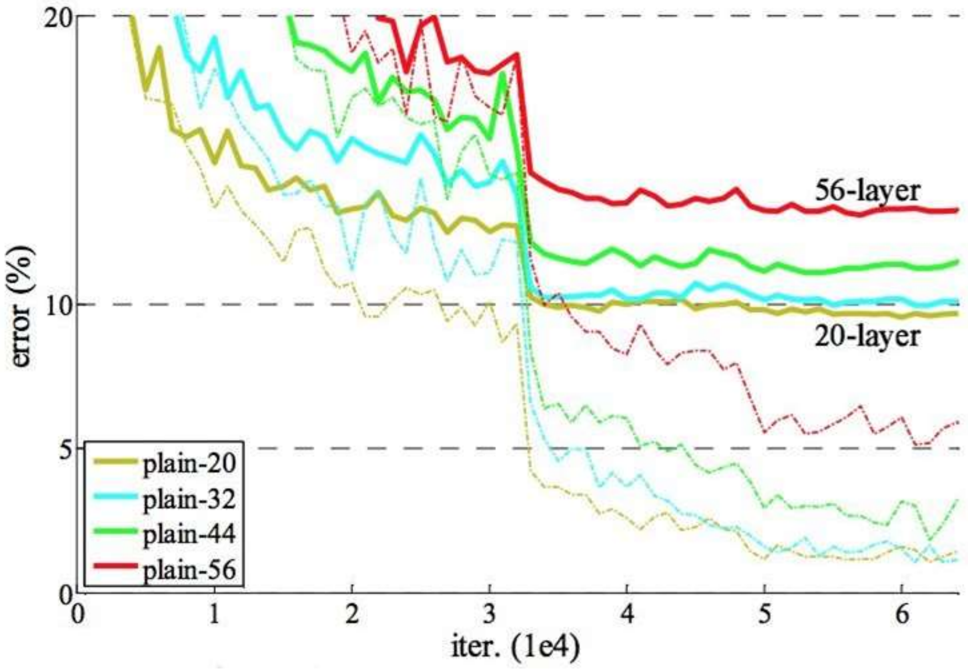


Sourced with permission from: Deep Residual Learning for Image Recognition, Kaiming He (2015)

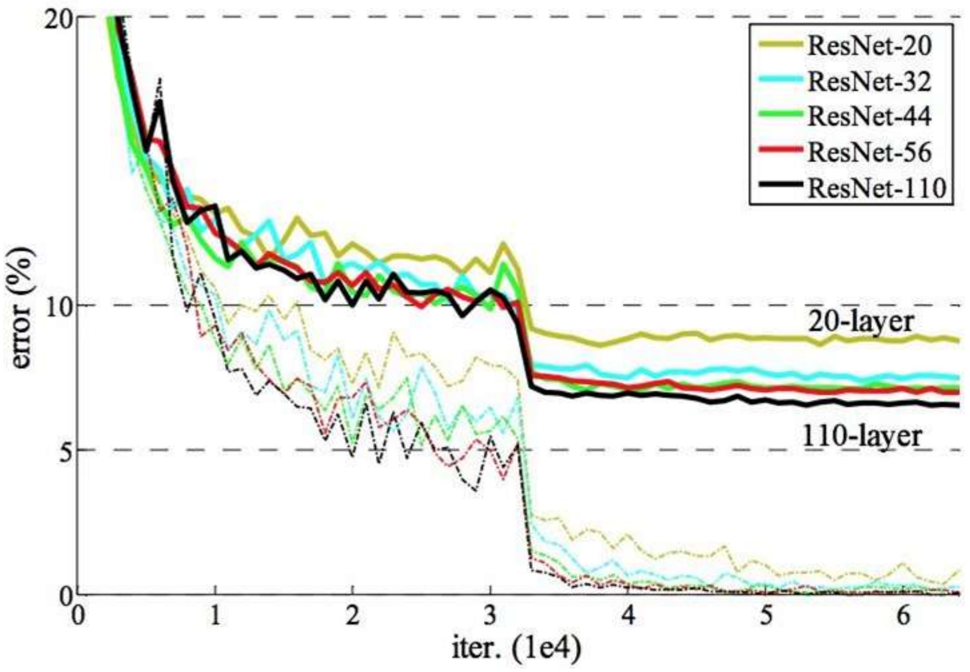
Case Study: ResNet

[He et al., 2015]

Experiments on CIFAR-10



Plain Networks

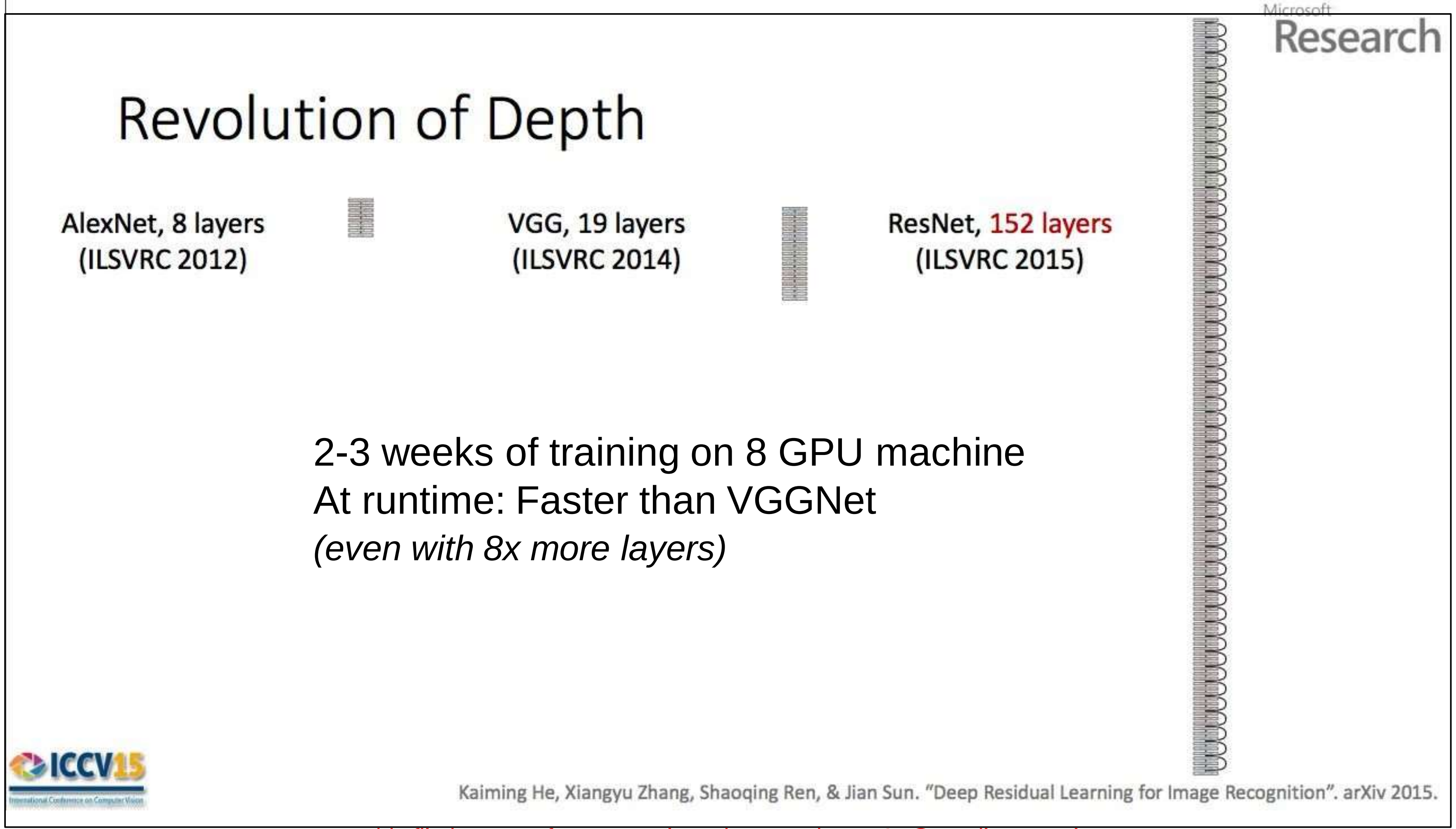


ResNets

Sourced with permission from: Deep Residual Learning for Image Recognition, Kaiming He (2015)

Case Study: ResNet

[He et al., 2015]



2-3 weeks of training on 8 GPU machine

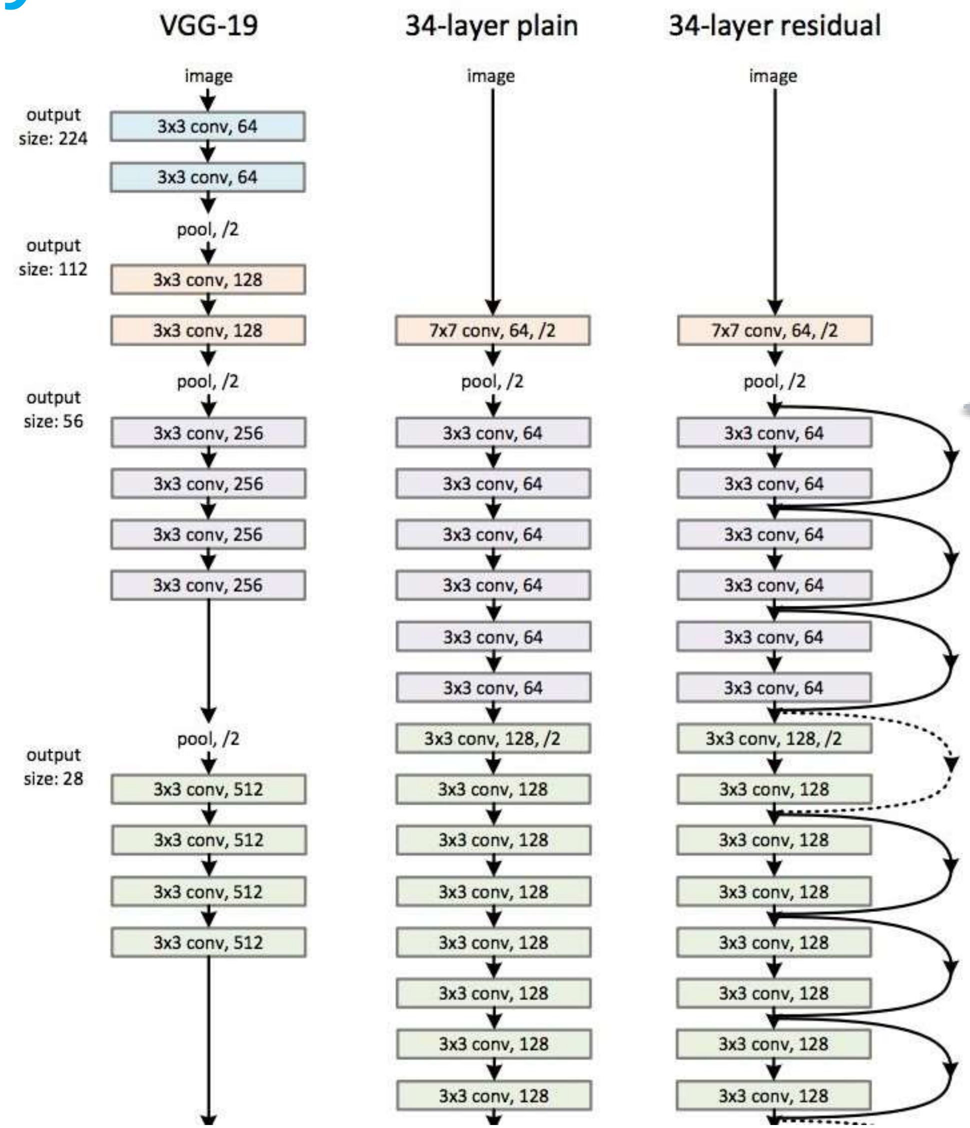
At runtime: Faster than VGGNet

(even with 8x more layers)

Sourced with permission from: Deep Residual Learning for Image Recognition, Kaiming He (2015)

Case Study: ResNet

[He et al., 2015]



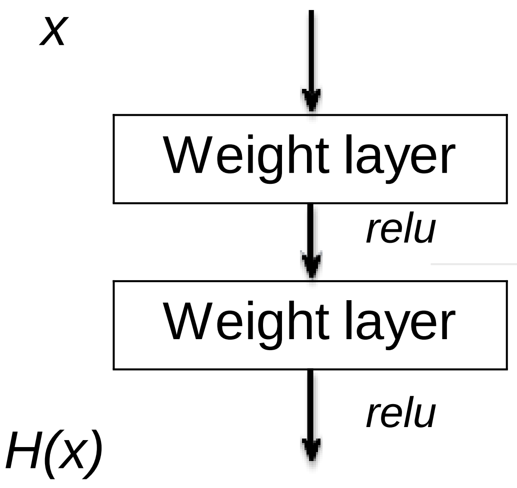
224x224x3

56 x 56
spatial dimension

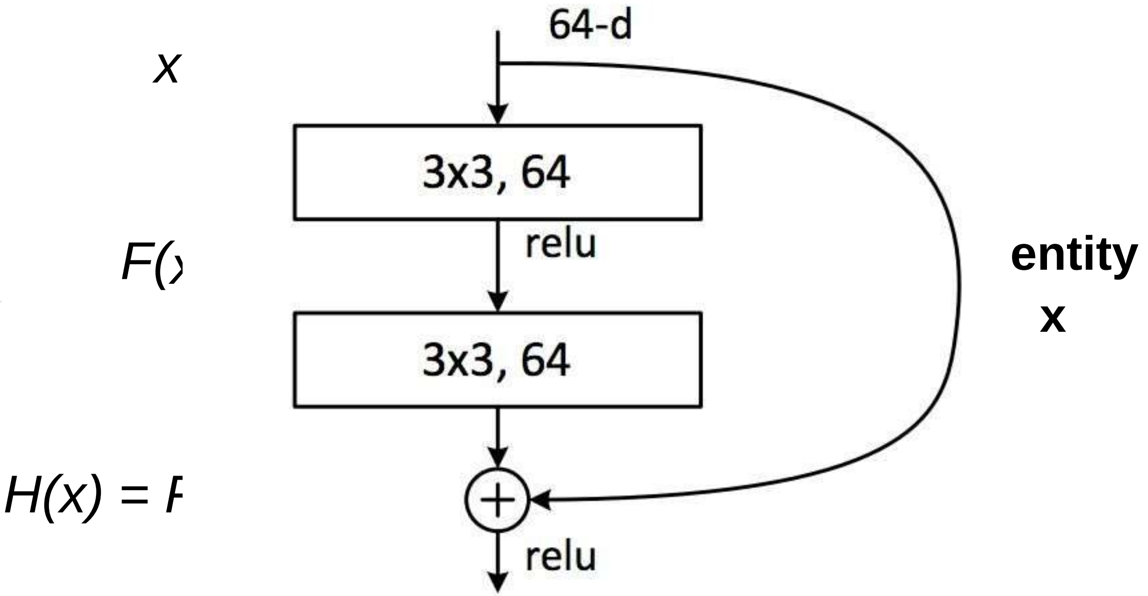
Sourced with permission from: Deep Residual Learning for Image Recognition, Kaiming He (2015)

Case Study: ResNet

[He et al., 2015]



Plain Network



ResNet

Sourced with permission from: Deep Residual Learning for Image Recognition, Kaiming He (2015)

Case Study: ResNet

[He et al., 2015]

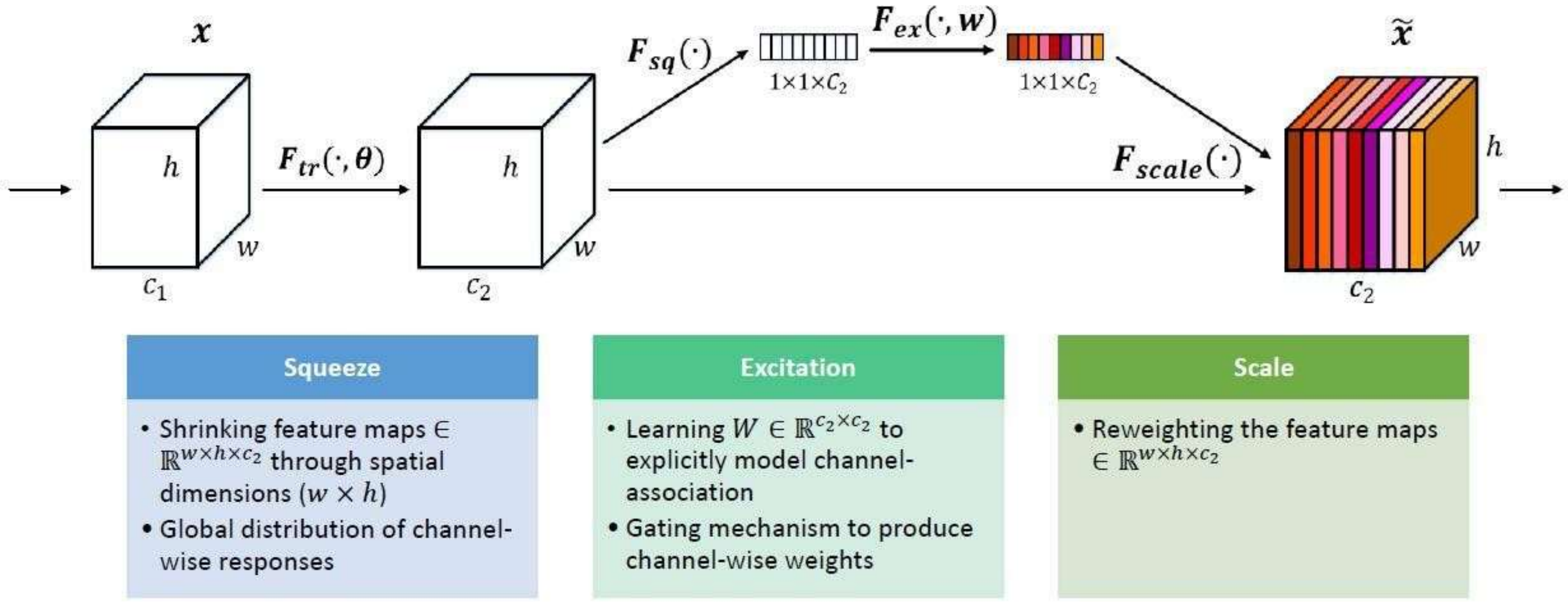
- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

ILVRC 2016

- CUImage was the winner with the ensemble approach.
- Classification error is down to 3.0% from 3.6% last year.
- Pretty boring, best model is just an ensemble
- https://www.reddit.com/r/MachineLearning/comments/54jiyy/large__scale_visual_recognition_challenge_2016/
- http://image-net.org/challenges/LSVRC/2016/results#loc

ILVRC 2017, Squeeze & Excitation Network

- Squeeze and Excitation block that can be added to a Conv Layer
- Add parameters to each channel of a convolutional block so that the network can adaptively adjust the weighting of each feature map

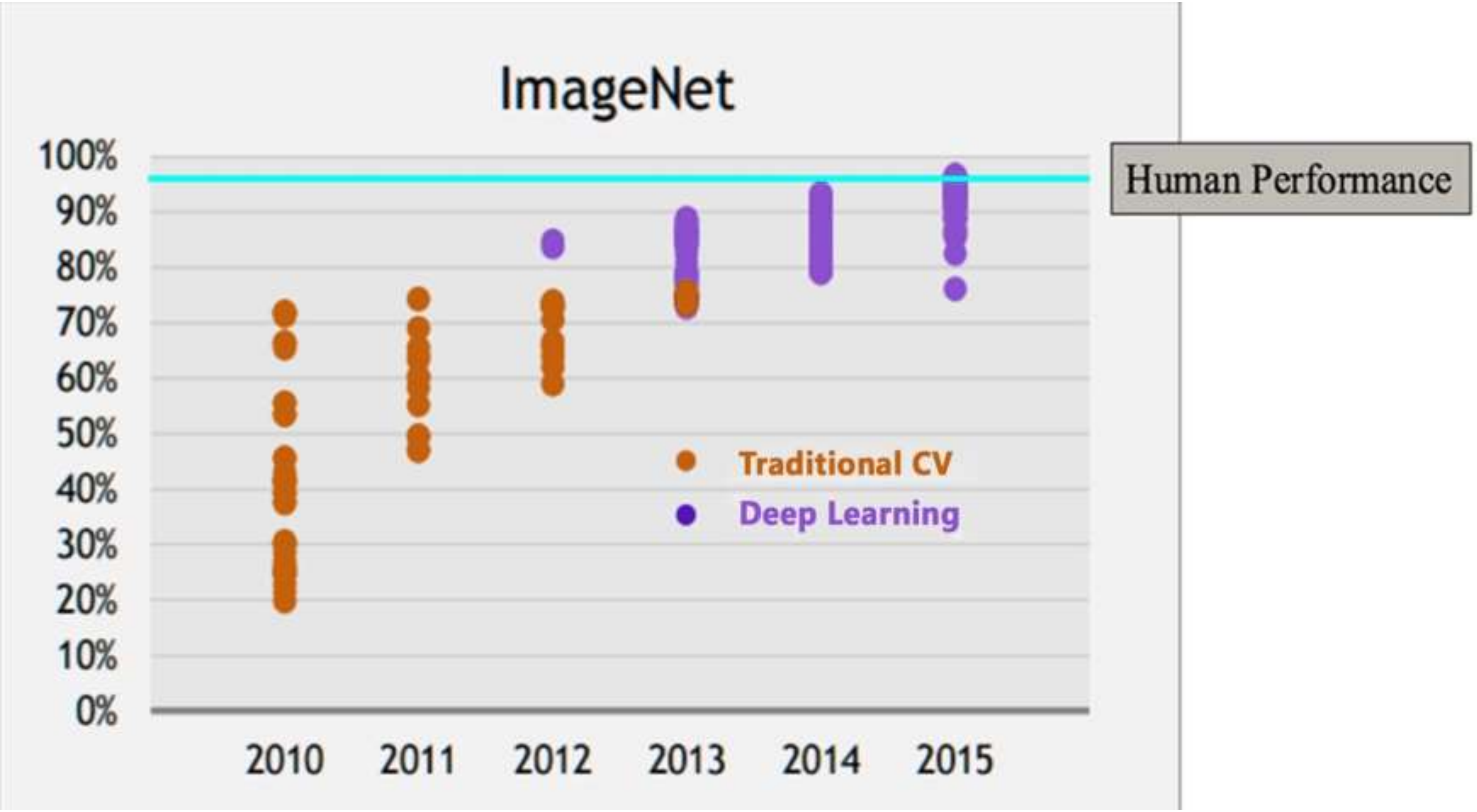


Sourced with permission from: Squeeze and Excitation Networks, ILSVRC 2017 presentation, Jie Hu et al. (2017)

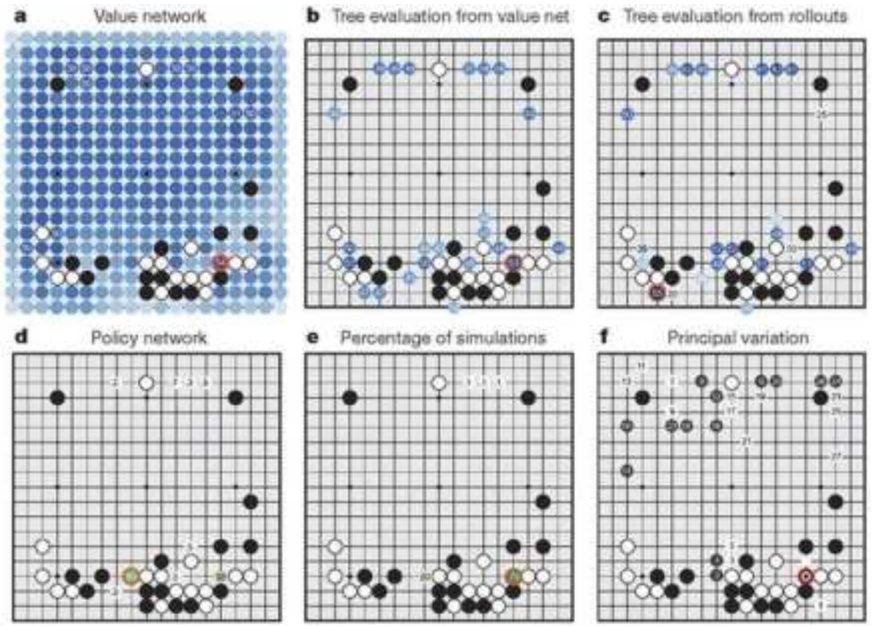
ILVRC 2017, Squeeze & Excitation Network

- Winning entry comprised a small ensemble of SENets that employed a standard multi-scale and multi-crop fusion strategy
- **2.251%** top-5 error on the test set
- Nearly 25% improvement on the winning entry of 2016 (2.99% top-5 error)
- One of the high-performing networks is constructed by integrating SE blocks with a modified ResNeXt

Why ConvNets?



Case Study: DeepMind’s AlphaGo



Images Source: ‘Mastering the game of Go without human knowledge’, Nature, David Silver et al. (2017)

Case Study: DeepMind's AlphaGo

The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

Policy network:

INPUT: [19x19x48]
CONV1: 192 5x5 filters , stride 1, pad 2 [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 [19x19] *(probability map of promising moves)*

Excerpt Source: 'Mastering the game of Go without human knowledge', Nature, David Silver et al. (2017)

Summary

- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like:

  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K-SOFTMAX

  where N is usually up to ~5, M is large, 0 <= K <= 2.
- But recent advances such as ResNet/GoogLeNet challenge this paradigm

Data needs for ConvNets

"ConvNets need a lot of data to train"?

This is a myth

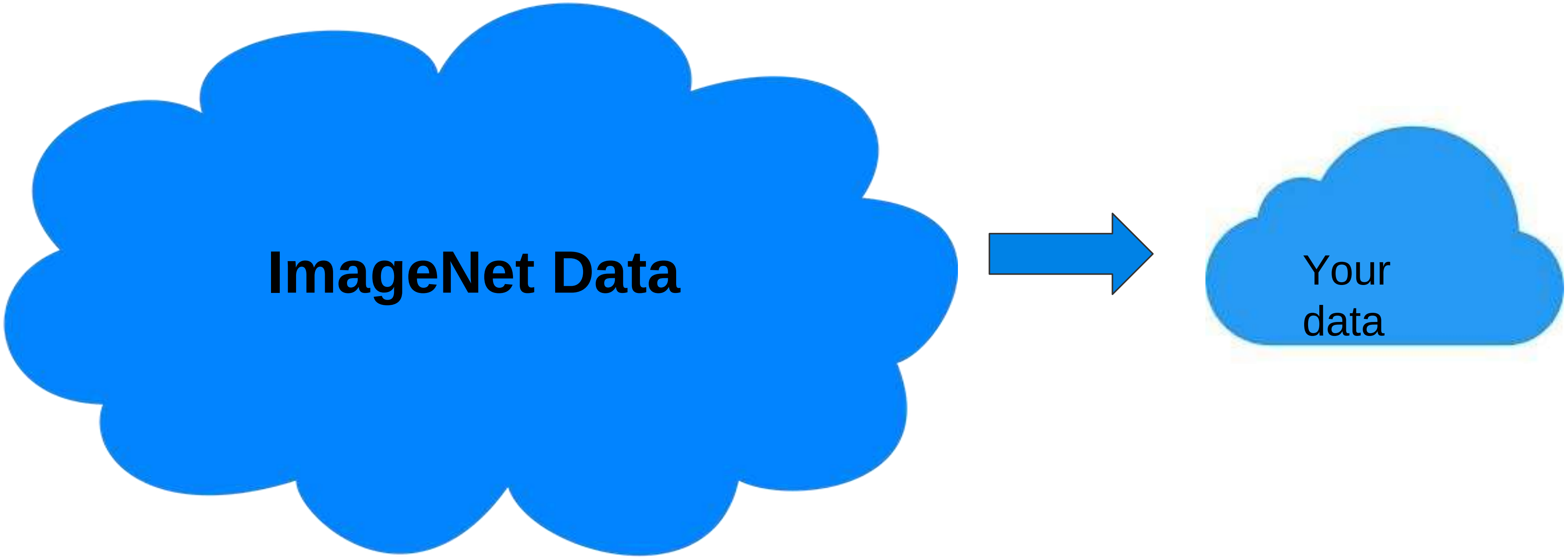
Finetuning

ConvNets usually not trained from scratch

Data needs for ConvNets

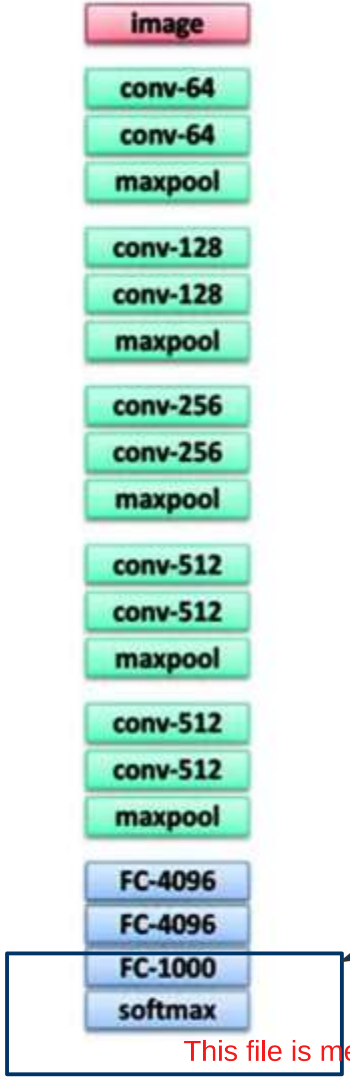
Train once on massive data like ImageNet

Fine tune the network using your own (much smaller) data



Transfer Learning with CNNs

1.
Train on ImageNet



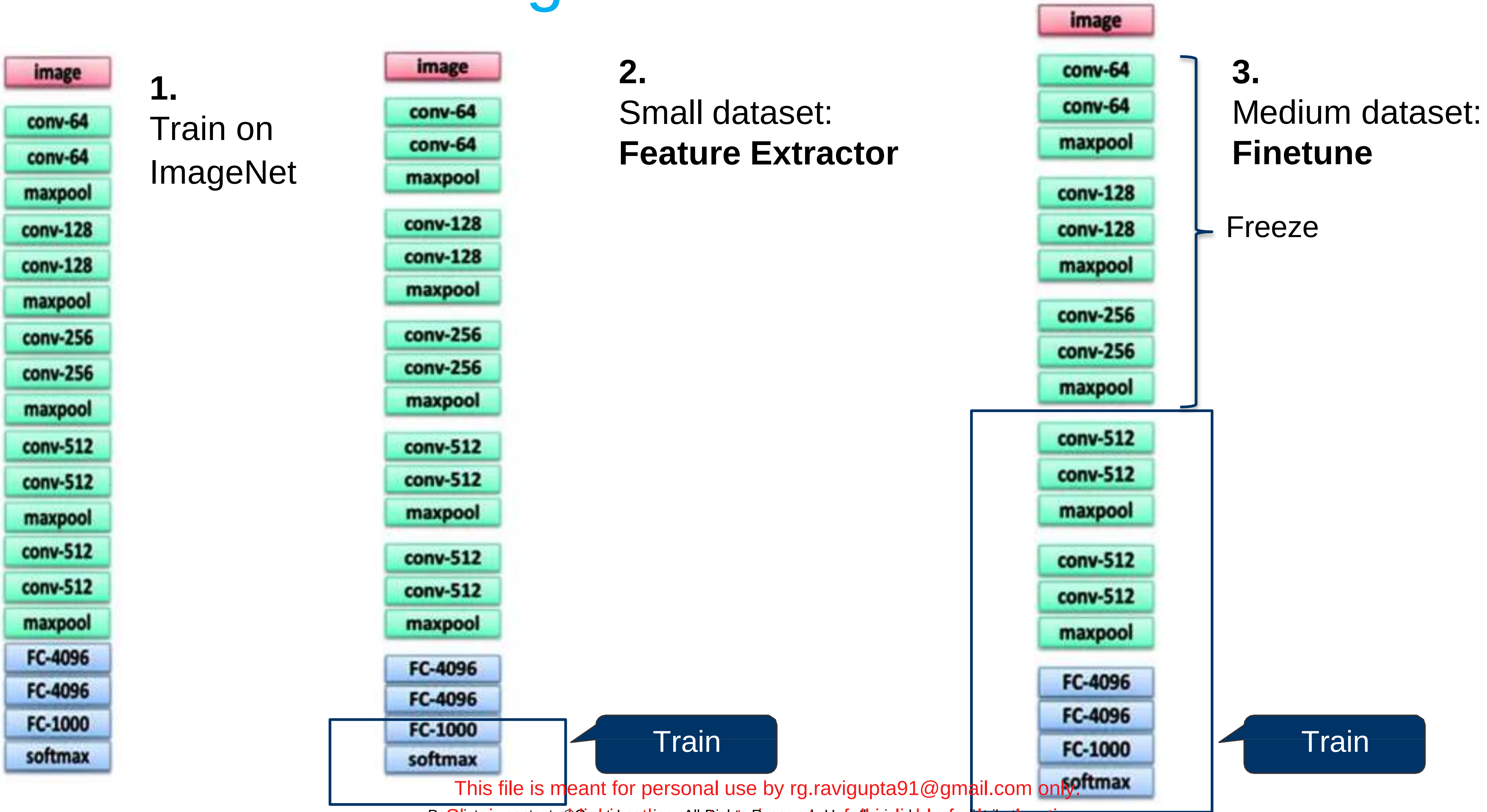
2.
If you have small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

Swap softmax layer at end

3.
If dataset is medium sized, **"finetune"**. Use the old weights as initialization, train the full network or only some of the higher layers

Retrain bigger portion of network

Transfer Learning with CNNs



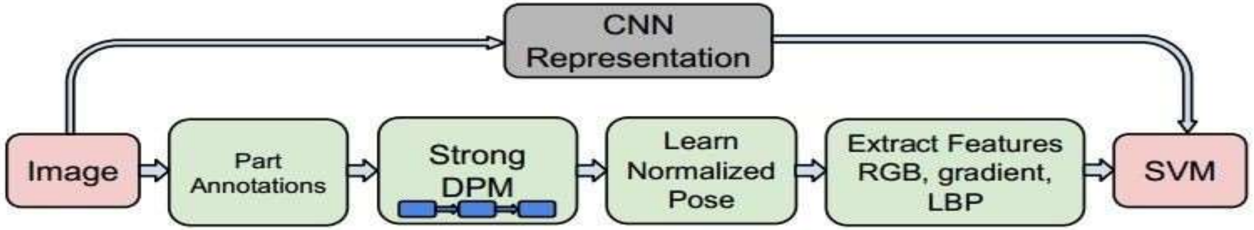
Transfer Learning with CNNs

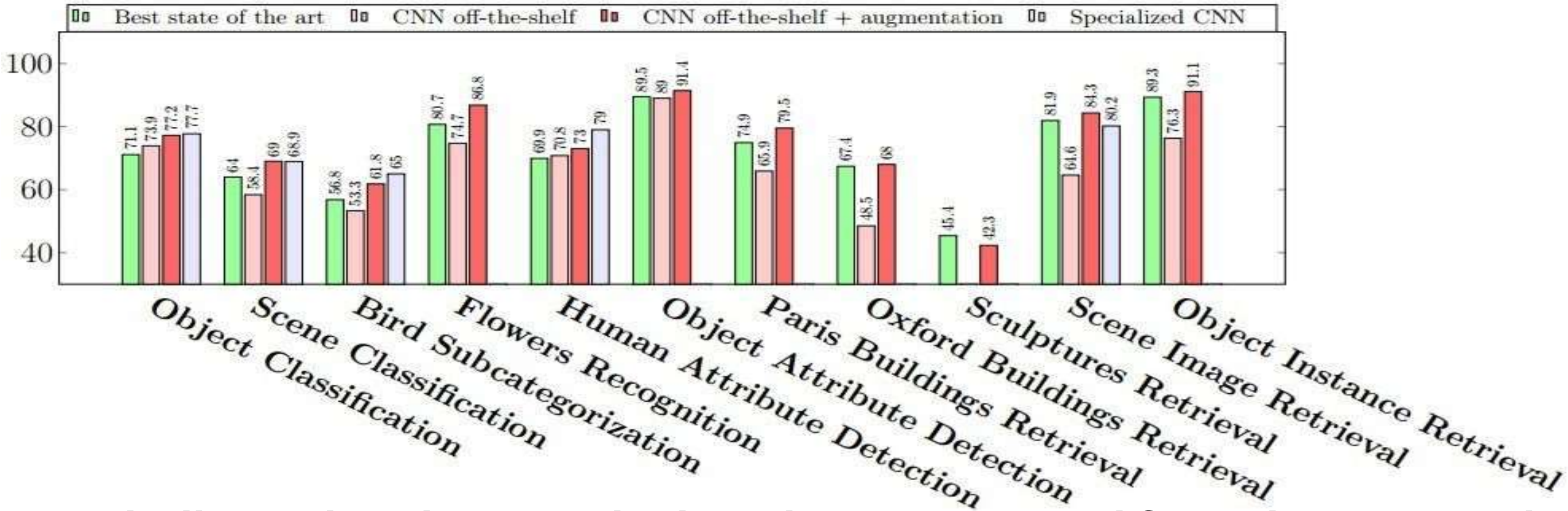
Rule of thumb:

- Use only ~1/10th of the original learning rate in finetuning top layer
- And ~1/100th in intermediate layers

CNN Features off-the-shelf

[Razavian et al, 2014]





“Recent results indicate that the generic descriptors extracted from the convolutional neural networks are very powerful.”

Source: ‘CNN Features off-the-shelf: An Astounding Baseline for Recognition’, Razavian et al. (2014)

Deep Convolutional Activation for Generic Visual Recognition

[Donahue, Jia et al., 2013]

	$DeCAF_6$	$DeCAF_7$
LogReg	**40.94 ± 0.3**	40.84 ± 0.3
SVM	39.36 ± 0.3	40.66 ± 0.3
Xiao et al. (2010)	38.0	

Source: ‘DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition, Donahue, Jia, et al., (2013)

more generic

more specific

	very similar dataset	**very different dataset**
very little data	?	?
quite a lot of data	?	?

more generic

more specific

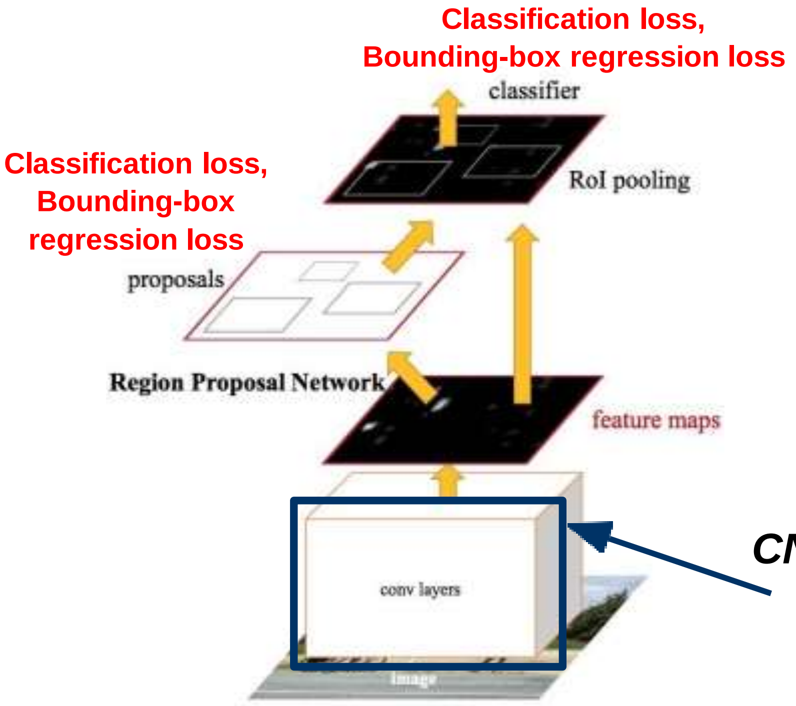
	very similar dataset	very different dataset
very little data	On the Top layer use Linear Classifier	?
quite a lot of data	Finetuning of few layers	?

more generic

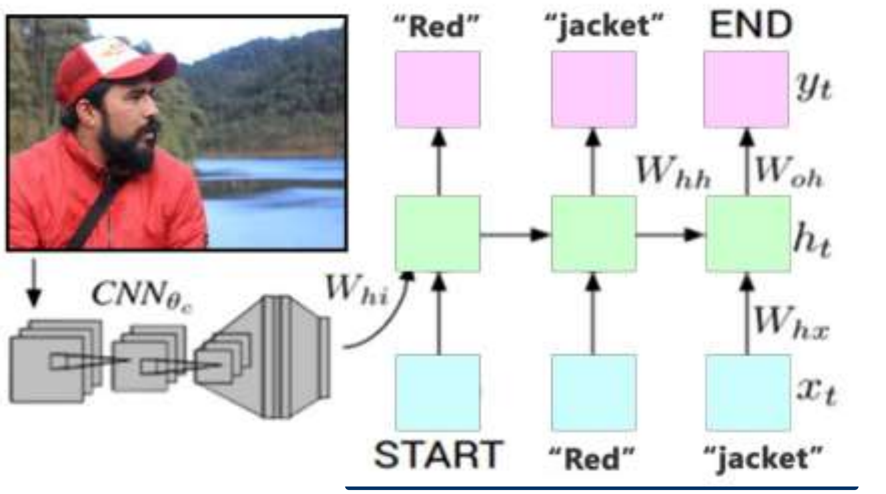
more specific

	very different dataset	very similar dataset
very little data	On the Top layer use Linear Classifier	Try linear classifier from different stages
quite a lot of data	Finetuning of few layers	Finetune a larger number of layers

Transfer learning with CNNs is common



CNN pretrained on ImageNet



Word vectors pretrained from word2vec

Object Detection
Faster R-CNN

The Image Captioning problem
CNN + RNN

Sources: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, Ren, He et al. (2016)

E.g. Caffe Model Zoo: Lots of pretrained ConvNets

https://github.com/BVLC/caffe/wiki/Model-Zoo

https://github.com/szagoruyko/loadcaffe

Model Zoo

ELM edited this page 21 days ago · 56 revisions

Check out the model zoo documentation for details.

To acquire a model:

1. download the model gist by `./scripts/download_model_from_gist.sh <gist_id> <dirname>` to load the model metadata, architecture, solver configuration, and so on. (`<dirname>` is optional and defaults to caffe/models).
2. download the model weights by `./scripts/download_model_binary.py <model_dir>` where `<model_dir>` is the gist directory from the first step.

or visit the model zoo documentation for complete instructions.

Berkeley-trained models

- Finetuning on Flickr Style: same as provided in `models/`, but listed here as a Gist for an example.
- BVLC GoogleNet: `models/bvlc_googlenet`

Network in Network model

The Network in Network model is described in the following ICLR-2014 paper:

```
Network In Network
M. Lin, Q. Chen, S. Yan
International Conference on Learning Representations, 2014 (arXiv:1409.1556)
```

please cite the paper if you use the models.

Models:

- NIN-Imagenet: a small(29MB) model for imagenet, yet performs slightly better than AlexNet, and fast to train. (Note: a more caffe-compatible version with correct convolutional weights shape: https://drive.google.com/folderview?id=0B0IedYUunOQINEFtUi1QNWVhVVU&usp=drive_web)
- NIN-CIFAR10: NIN model on CIFAR10, originally published in the paper Network in Network. The error rate of this model is 10.4% on CIFAR10.

Models from the BMVC-2014 paper "Return of the Devil in the Details: Delving Deep into Convolutional Nets"

The models are trained on the ILSVRC-2012 dataset. The details can be found on the project page or in the following BMVC-2014 paper:

```
Return of the Devil in the Details: Delving Deep into Convolutional Nets
K. Chatfield, K. Simonyan, A. Vedaldi, A. Zisserman
British Machine Vision Conference, 2014 (arXiv ref. cs1405.3531)
```

Please cite the paper if you use the models.

Models:

- VGG_CNN_S: 13.1% top-5 error on ILSVRC-2012-val
- VGG_CNN_M: 13.7% top-5 error on ILSVRC-2012-val
- VGG_CNN_M_2048: 13.5% top-5 error on ILSVRC-2012-val
- VGG_CNN_M_1024: 13.7% top-5 error on ILSVRC-2012-val
- VGG_CNN_M_128: 15.6% top-5 error on ILSVRC-2012-val
- VGG_CNN_F: 16.7% top-5 error on ILSVRC-2012-val

Models used by the VGG team in ILSVRC-2014

Places-CNN model from MIT.

GoogLeNet GPU implementation from Princeton.

Fully Convolutional Semantic Segmentation Models (FCN-Xs)

CaffeNet fine-tuned for Oxford flowers dataset

CNN Models for Salient Object Subitizing.

Deep Learning of Binary Hash Codes for Fast Image Retrieval

Places_CNDS_models on Scene Recognition

Models for Age and Gender Classification.

GoogLeNet_cars on car model classification

Holistically-Nested Edge Detection

Translating Videos to Natural Language

VGG Face CNN descriptor

Yearbook Photo Dating

CCNN: Constrained Convolutional Neural Networks for Weakly Supervised Segmentation

...

Thank you!