CNN – Transfer Learning

Outline

- Introduction and Motivation to Transfer Learning (TL)
- Understanding Transfer Learning
- Transfer Learning Strategies
- TL in Deep Learning – CNN’s pre-trained models and Architectures
- Applications
- Advantages and Challenges

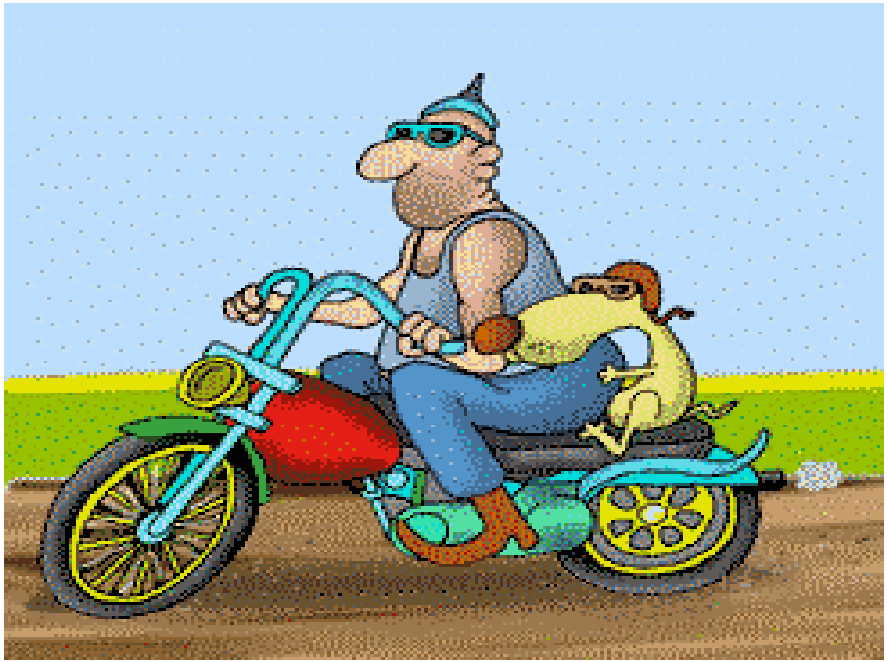
Transfer Learning

TRANSFER OF LEARNING

The application of skills, knowledge, and/or attitudes that were learned in one situation to another **learning** situation (Perkins, 1992)

Transfer Learning

Transfer Learning



Know how to ride a motorbike ➦ Learn how to ride a car

Introduction

- Human learners appear to have inherent ways to transfer knowledge between tasks.
- That is, we recognize and apply relevant knowledge from previous learning experience when we encounter new tasks.
- The more related a new task is to our previous experience, the more easily we can master it.
- Transfer learning involves the approach in which knowledge learned in one or more source tasks is transferred and used to improve the learning of a related target task.

Why Transfer Learning?

- Many deep neural networks trained on natural images exhibit a curious phenomenon in common: on the first layer they learn features similar to filters and color blobs.

- Such first-layer features appear not to specific to a particular dataset or task but are general in that they are applicable to many datasets and tasks.

- As finding these standard features on the first layer seems to occur regardless of the exact cost function and natural image dataset, we call these first-layer features general.

- For example, in a network with an N-dimensional softmax output layer that has been successfully trained towards a supervised classification objective, each output unit will be specific to a particular class. We thus call the last-layer features specific.

Motivation

- The key motivation, especially considering the context of deep learning is the fact that most models which solve complex problems need a whole lot of data, and getting vast amounts of labeled data for supervised models 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!

ImageNet Challenge

IMAGENET

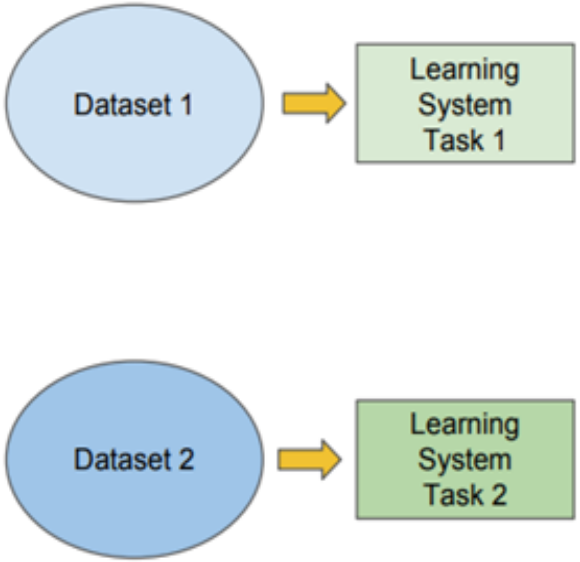
- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.

Motivation

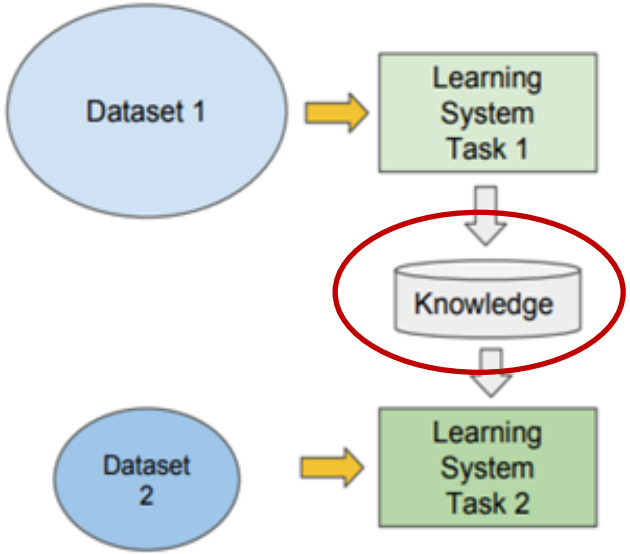
- However, getting such a dataset for every domain is tough.
- Besides, most deep learning models are very specialized to a particular domain or even a specific task.
- While these might be state-of-the-art models, with really high accuracy and beating all benchmarks, it would be only on very specific datasets and end up suffering a significant loss in performance when used in a new task which might still be similar to the one it was trained on.
- This forms the motivation for transfer learning, which goes beyond specific tasks and domains, and tries to see how to leverage knowledge from pre-trained models and use it to solve new problems!

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



Traditional ML/DL .vs. Transfer Learning

- **Traditional learning** is isolated and occurs purely based on specific tasks, datasets and training separate isolated models on them. No knowledge is retained which can be transferred from one model to another.

- In **Transfer learning**, you can leverage knowledge (features, weights etc) from previously trained models for training newer models and even tackle problems like having less data for the newer task!

- In the case of problems in the computer vision domain, certain low-level features, such as edges, shapes, corners and intensity, can be shared across tasks, and thus enable knowledge transfer among tasks! Also, as we have depicted in the earlier figure, knowledge from an existing task acts as an additional input when learning a new target task.

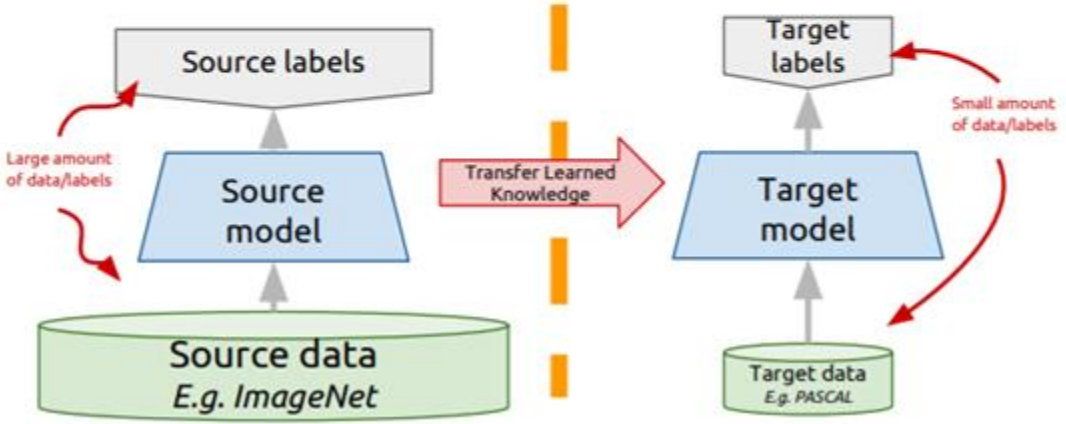
Transfer Learning in Deep Learning

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

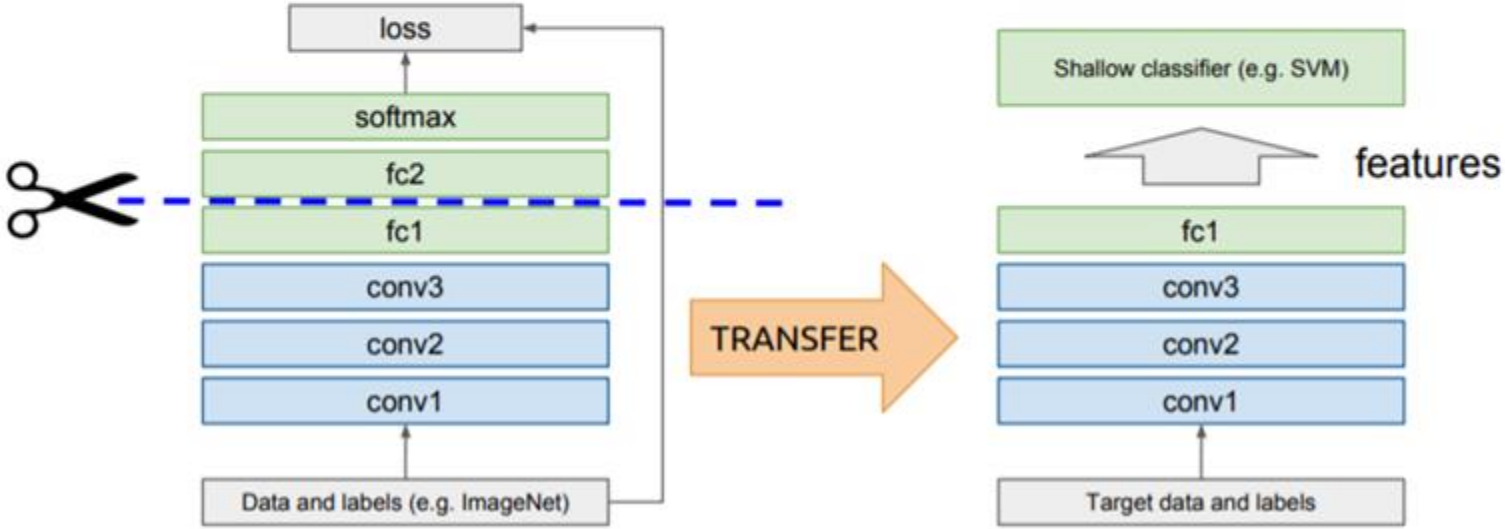


Deep Transfer Learning Strategies

Pre-trained models as feature extractors

Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that $D_S = D_T$



Pre-trained models as feature extractors

- Deep learning systems and models are layered architectures that learn different features at different layers (hierarchical representations of layered features).
- These layers are then finally connected to a last layer (usually a fully connected layer, in the case of supervised learning) to get the final output.
- This layered architecture allows us to utilize a pre-trained network (such as Inception V3 or VGG) without its final layer as a fixed feature extractor for other tasks.

Freeze or fine-tune?

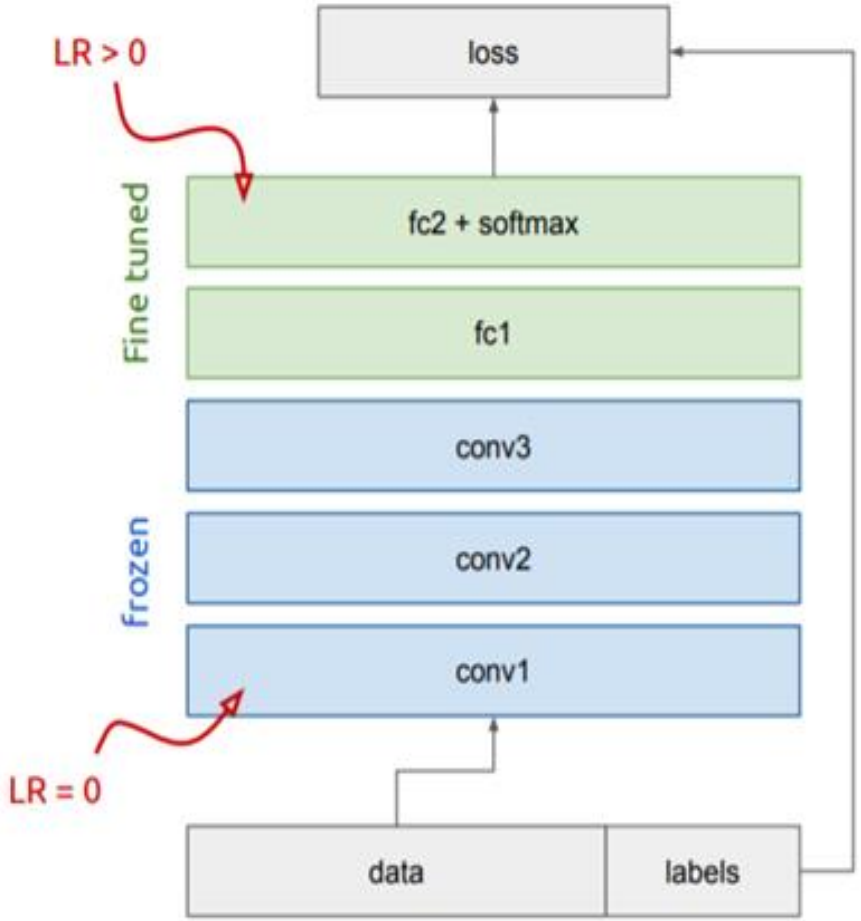
Bottom *n* layers can be frozen or fine tuned.

- **Frozen**: not updated during backprop
- **Fine-tuned**: updated during backprop

Which to do depends on target task:

- **Freeze**: target task labels are scarce, and we want to avoid overfitting
- **Fine-tune**: target task labels are more plentiful

In general, we can set learning rates to be different for each layer to find a tradeoff between freezing and fine tuning



Transfer Learning Scenarios

Depending on both the size of the new dataset and the similarity of the new dataset to the original dataset, the approach for using transfer learning will be different. Keeping in mind that ConvNet features are more generic in the early layers and more original-dataset specific in the later layers, **here are some common rules of thumb for navigating the four major scenarios:**

1. The *target* dataset is **small** and **similar** to the *base* training dataset.

   Since the target dataset is small, it is not a good idea to fine-tune the ConvNet due to the risk of overfitting. Since the *target* data is similar to the *base* data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, we:

   - Remove the fully connected layers near the end of the pretrained *base* ConvNet
   - Add a new fully connected layer that matches the number of classes in the *target* dataset
   - Randomize the weights of the new fully connected layer and freeze all the weights from the pre-trained network
   - Train the network to update the weights of the new fully connected layers

Transfer Learning Scenarios

2. The *target* dataset is **large** and **similar** to the *base* training dataset.

   Since the *target* dataset is large, we have more confidence that we won't overfit if we try to fine-tune through the full network. Therefore, we:

   - Remove the last fully connected layer and replace with the layer matching the number of classes in the *target* dataset
   - Randomly initialize the weights in the new fully connected layer
   - Initialize the rest of the weights using the pre-trained weights, i.e., unfreeze the layers of the pre-trained network
   - Retrain the entire neural network

Transfer Learning Scenarios

3. The *target* dataset is **small** and **different** from the *base* training dataset.

Since the data is small, overfitting is a concern. Hence, we train only the linear layers. But as the *target* dataset is very different from the *base* dataset, the higher level features in the ConvNet would not be of any relevance to the *target* dataset. So, the new network will only use the lower level features of the *base* ConvNet. To implement this scheme, we:

- Remove most of the pre-trained layers near the beginning of the ConvNet
- Add to the remaining pre-trained layers new fully connected layers that match the number of classes in the new dataset
- Randomize the weights of the new fully connected layers and freeze all the weights from the pre-trained network
- Train the network to update the weights of the new fully connected layers

Transfer Learning Scenarios

4. The *target* dataset is large and different from the *base* training dataset.

As the *target* dataset is large and different from the *base* dataset, we can train the ConvNet from scratch. However, in practice, it is beneficial to initialize the weights from the pre-trained network and fine-tune them as it might make the training faster. In this condition, the implementation is the same as in case 3.

Transfer Learning

Let us understand it using CNN

Transfer Learning

“You need a lot of a data if you want to train/use CNNs”

Not really

TL using CNN – Understand the data – Image Net

ImageNet

From Wikipedia, the free encyclopedia

The **ImageNet** project is a large visual database designed for use in visual object recognition software research. More than 14 million[1][2] images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided.[3] ImageNet contains more than 20,000 categories[2] with a typical category, such as "balloon" or "strawberry", consisting of several hundred images.[4] The database of annotations of third-party image URLs is freely available directly from ImageNet, though the actual images are not owned by ImageNet.[5] Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes. The challenge uses a "trimmed" list of one thousand non-overlapping classes.[6]

History of the database [edit]

AI researcher Fei-Fei Li began working on the idea for ImageNet in 2006. At a time when most AI research focused on models and algorithms, Li wanted to expand and improve the data available to train AI algorithms.[11] In 2007, Li met with Princeton professor Christiane Fellbaum, one of the creators of WordNet to discuss the project. As a result of this meeting, Li went on to build ImageNet starting from the word-database of WordNet and using many of its features.[12]

As an assistant professor at Princeton, Li assembled a team of researchers to work on the ImageNet project. They used Amazon Mechanical Turk to help with the classification of images.[12]

They presented their database for the first time as a poster at the 2009 Conference on Computer Vision and Pattern Recognition (CVPR) in Florida.[12][13][14]

Dataset [edit]

ImageNet crowdsources its annotation process. Image-level annotations indicate the presence or absence of an object class in an image, such as "there are tigers in this image" or "there are no tigers in this image". Object-level annotations provide a bounding box around the (visible part of the) indicated object. ImageNet uses a variant of the broad WordNet schema to categorize objects, augmented with 120 categories of dog breeds to showcase fine-grained classification.[6] One downside of WordNet use is the categories may be more "elevated" than would be optimal for ImageNet: "Most people are more interested in Lady Gaga or the iPod Mini than in this rare kind of diplodocus."[*clarification needed*] In 2012 ImageNet was the world's largest academic user of Mechanical Turk. The average worker identified 50 images per minute.[2]

Transfer Learning with CNNs

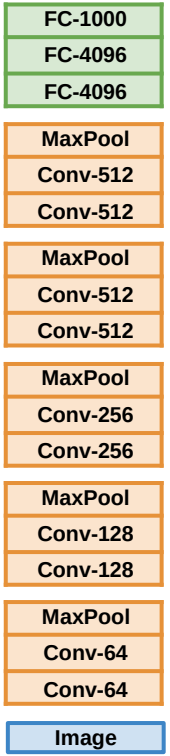
Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

1. Train on Imagenet

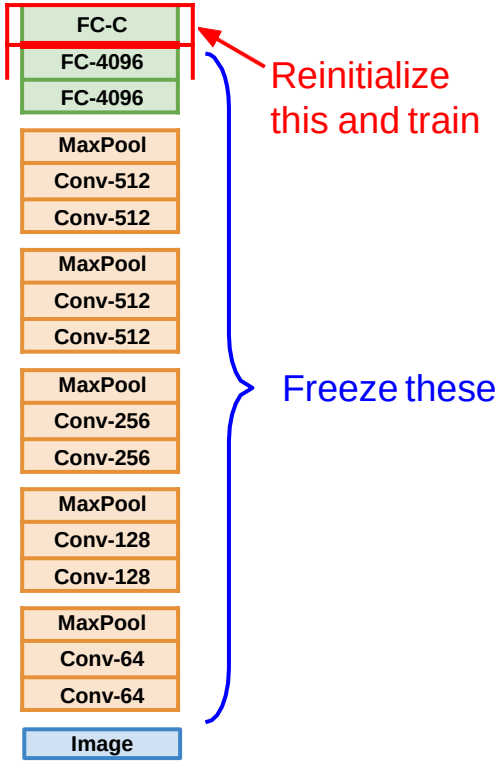
Transfer Learning with CNNs

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

1. Train on Imagenet



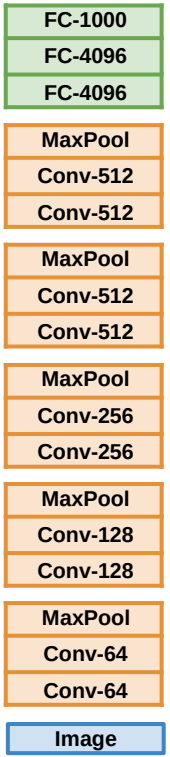
2. Small Dataset (C classes)



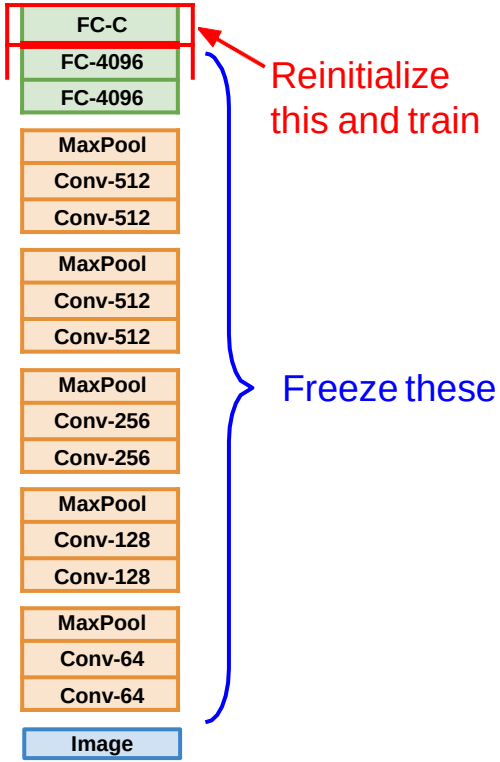
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

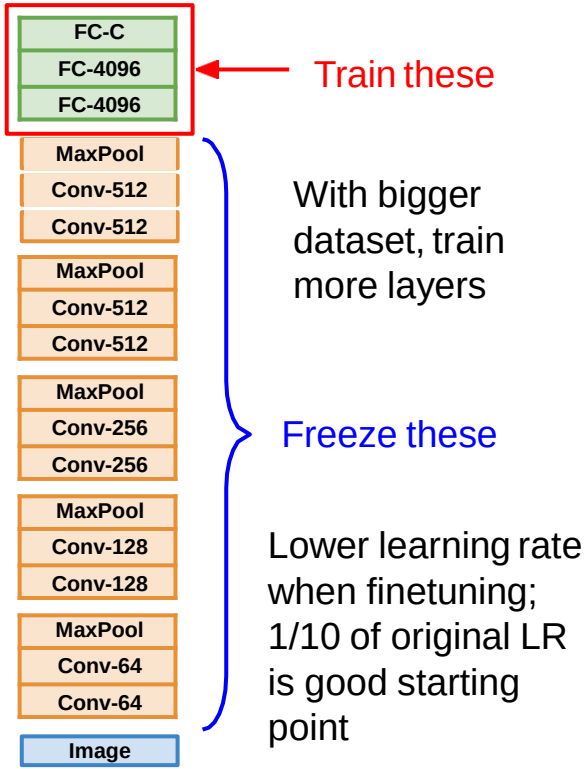
1. Train on Imagenet

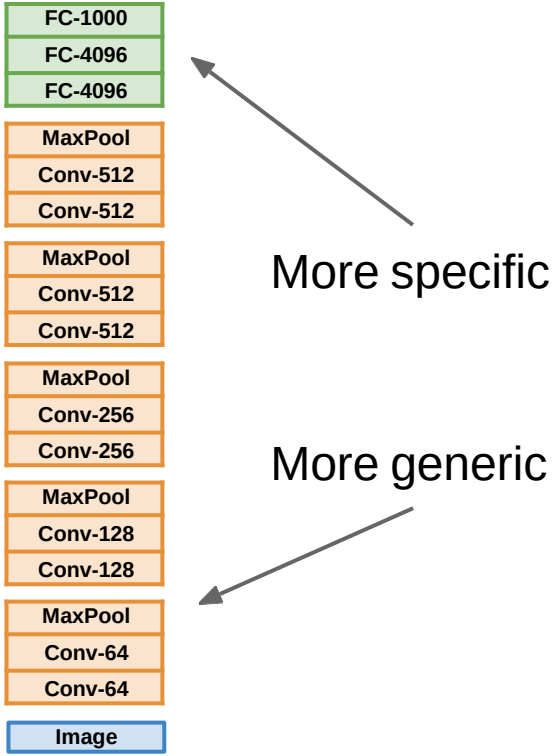


2. Small Dataset (C classes)

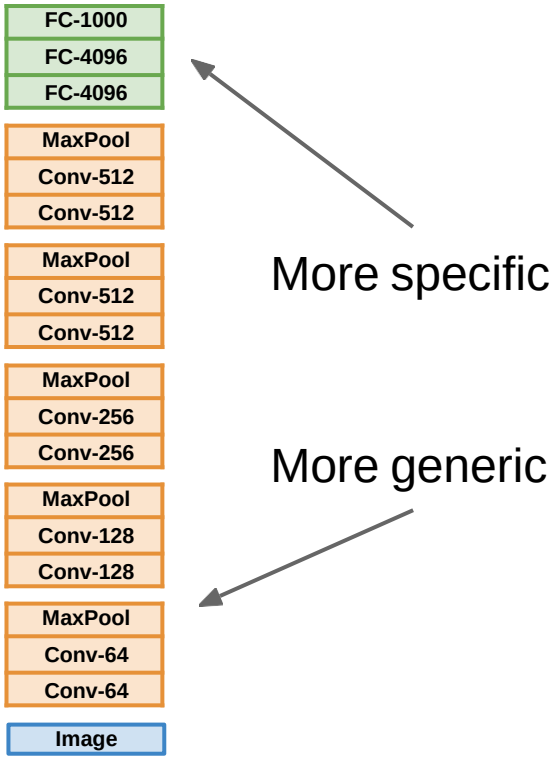


3. Bigger dataset





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



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Takeaway for your projects and beyond:

Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision

CNN Architectures

CNN Architectures (Pre-Trained Models)

Examples: (Available in Keras)

- Xception
- VGG16
- VGG19
- ResNet, ResNetV2
- InceptionV3
- InceptionResNetV2
- MobileNet, MobileNetV2
- DenseNet
- NASNet

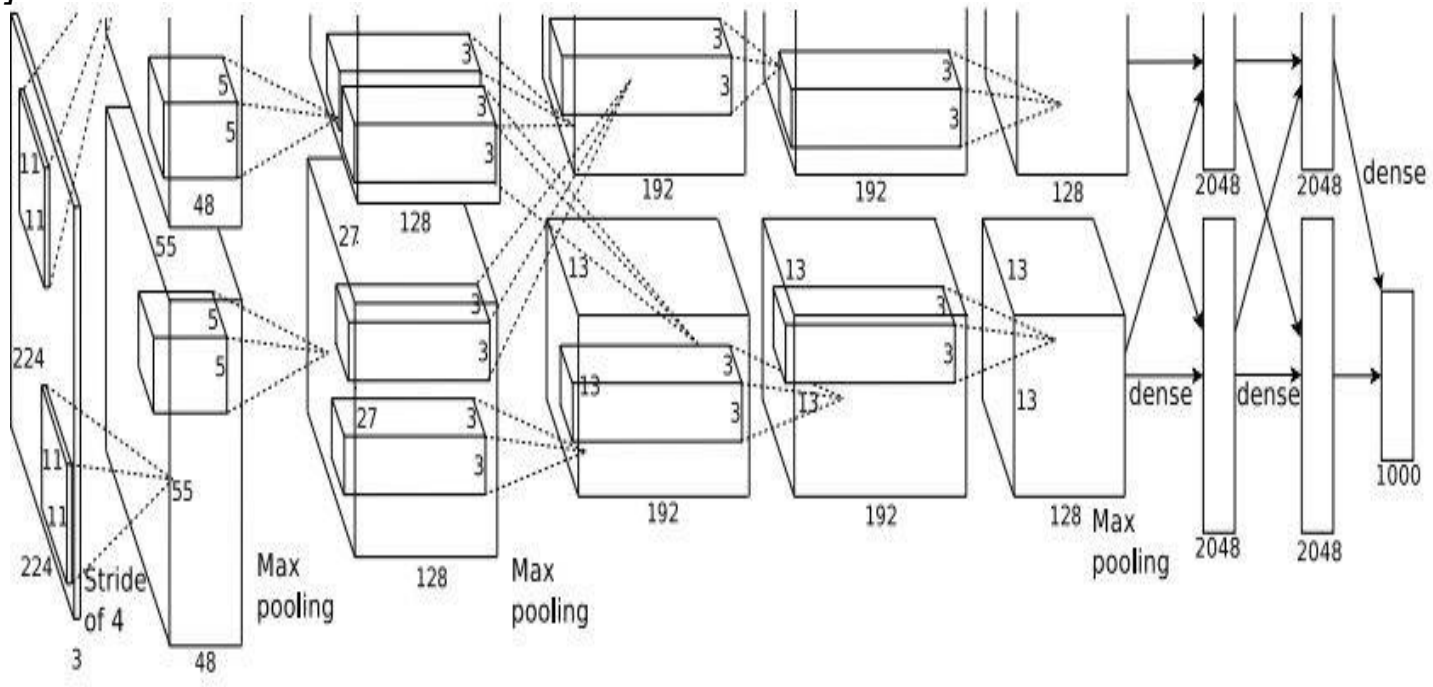
Find them @
https://keras.io/applications/

Case Study: AlexNet

[Krizhevsky et al. 2012]

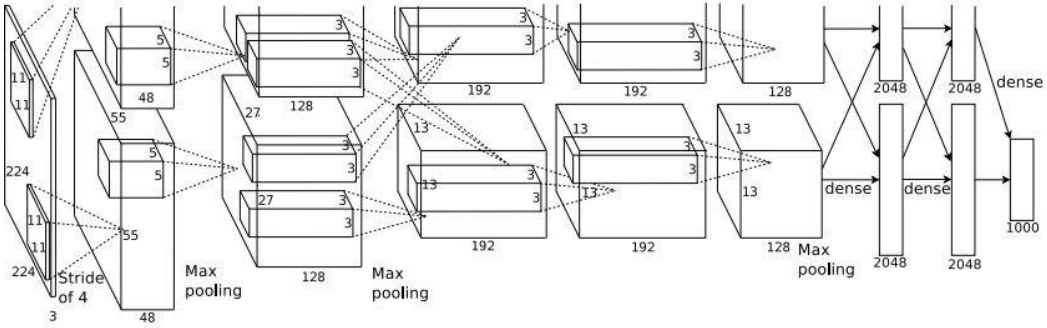
Architecture:

CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: **227x227x3** images

First layer (CONV1): 96 11x11 filters applied at stride 4

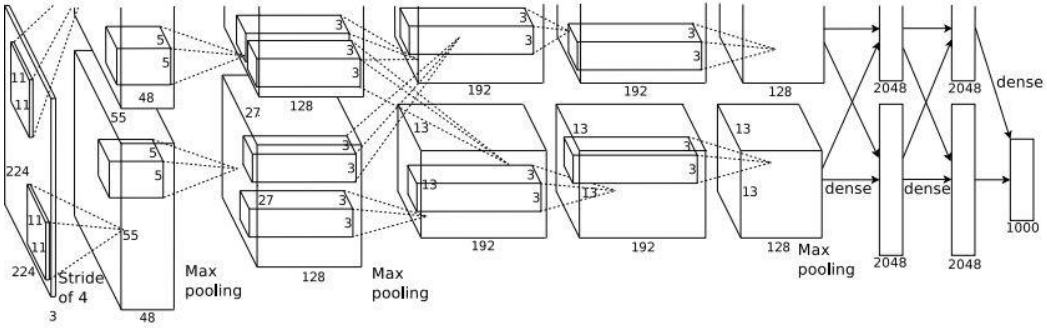
=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

(N – F)/S + 1

Case Study: AlexNet

[Krizhevsky et al. 2012]



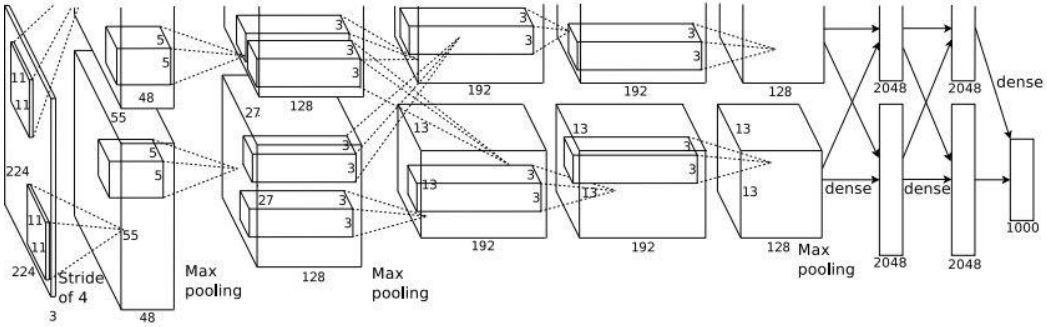
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]



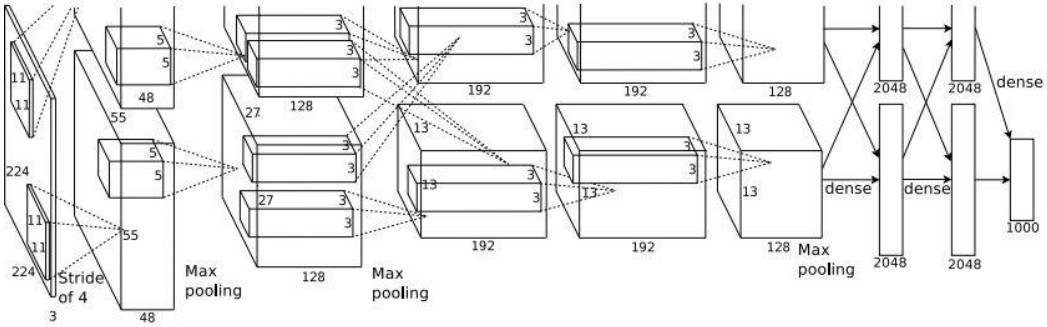
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

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

Case Study: AlexNet

[Krizhevsky et al. 2012]

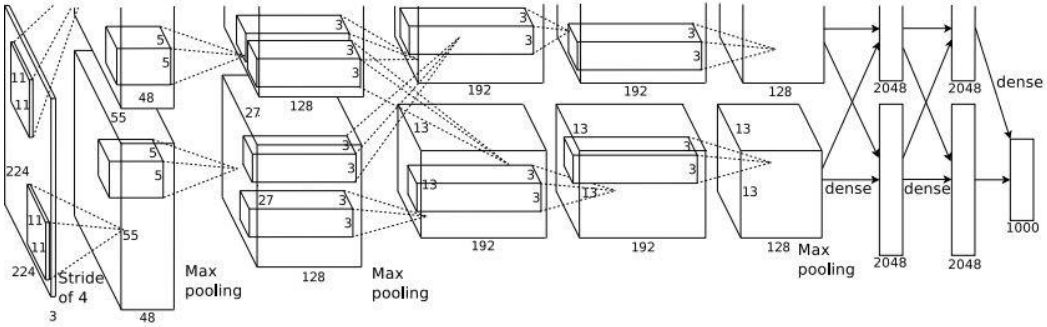


Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96

...

Case Study: AlexNet

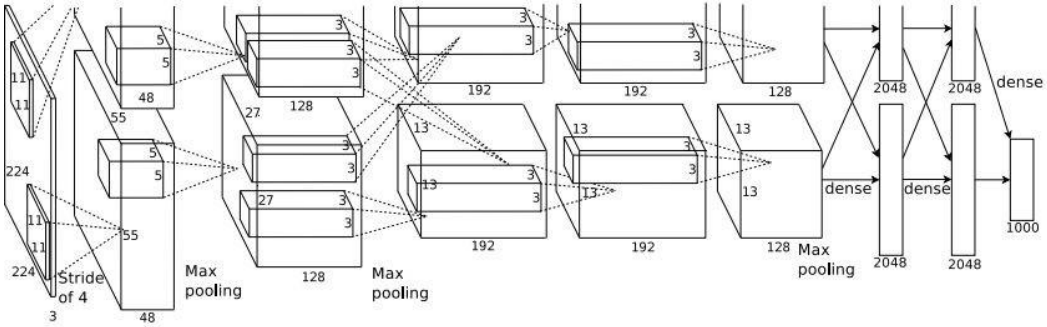
[Krizhevsky et al. 2012]



Full (simplified) AlexNet 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)

Case Study: AlexNet

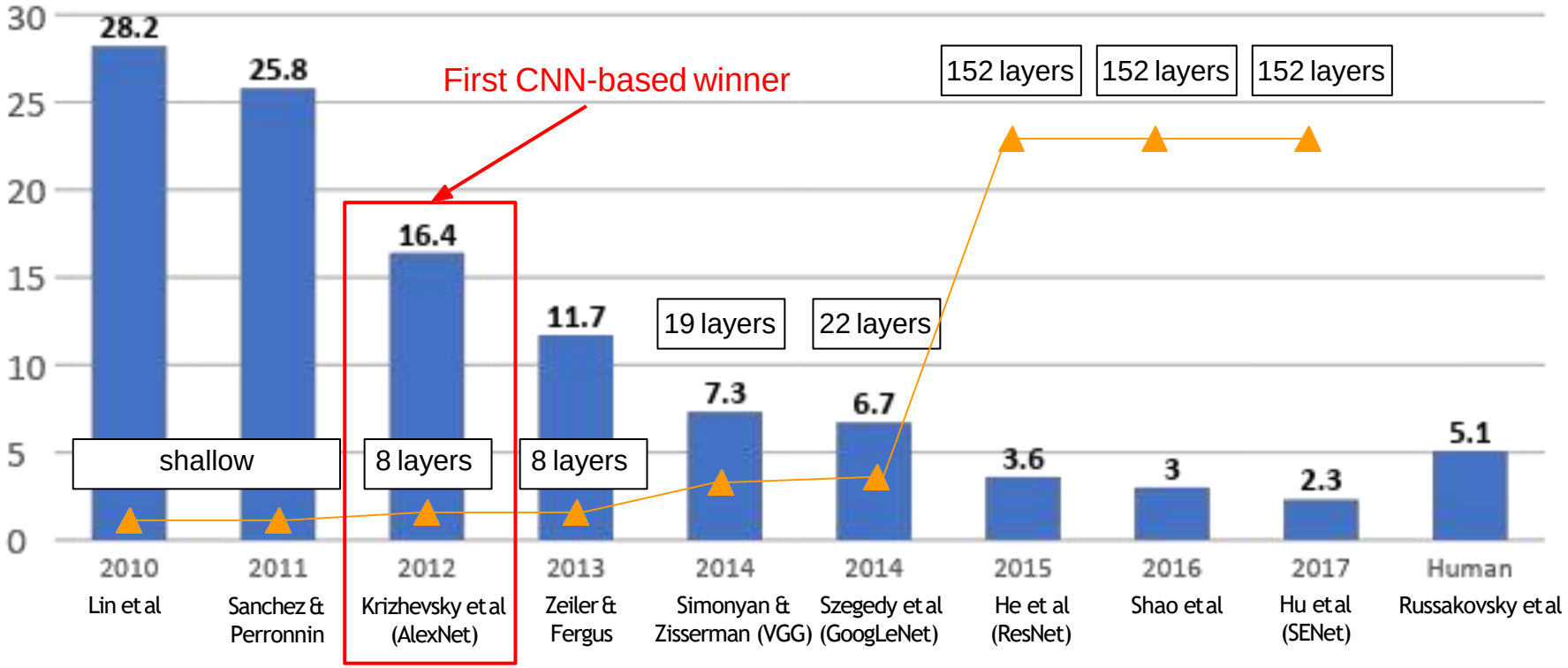
[Krizhevsky et al. 2012]



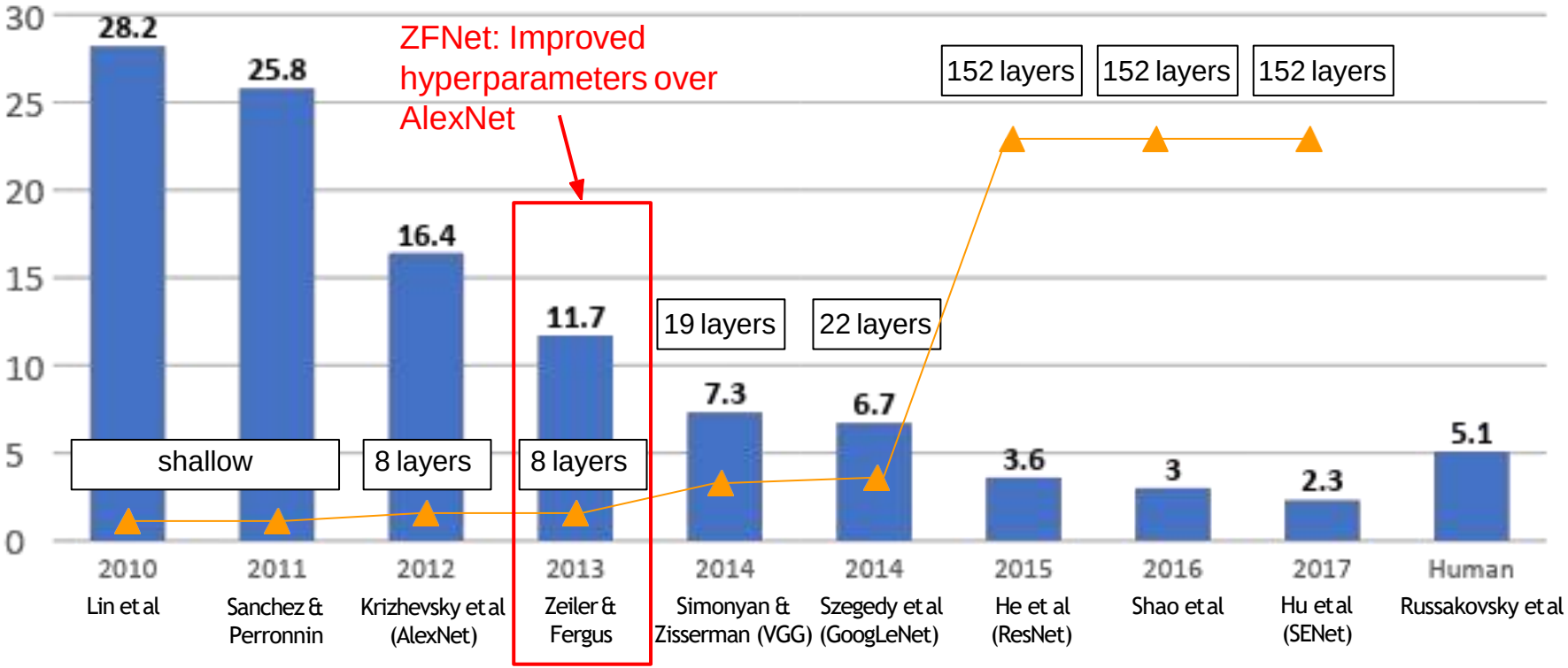
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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)

Details/Retrospectives:
-first use of ReLU
-used Norm layers (not common anymore)
-heavy data augmentation
-dropout 0.5
-batch size 128
-SGD Momentum 0.9
-Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

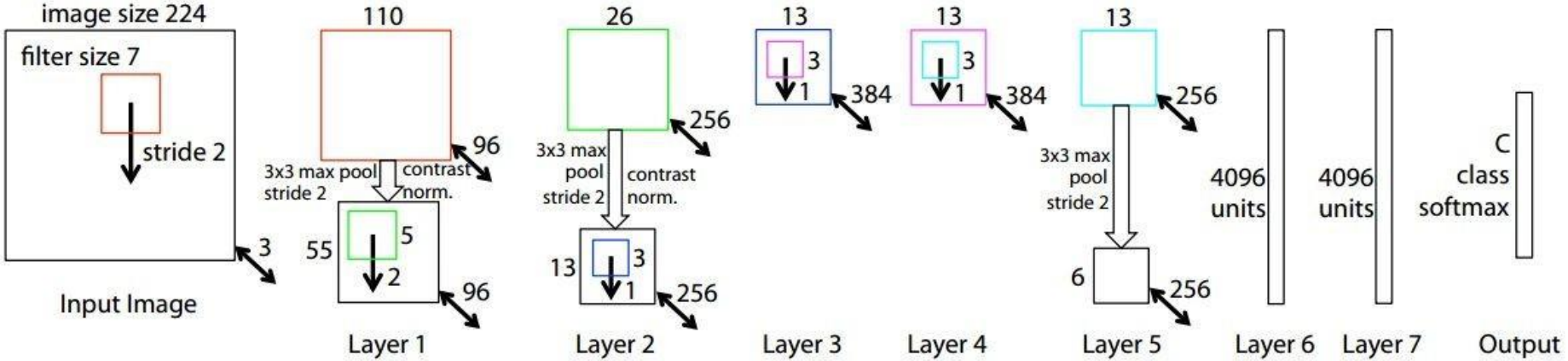


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ZFNet

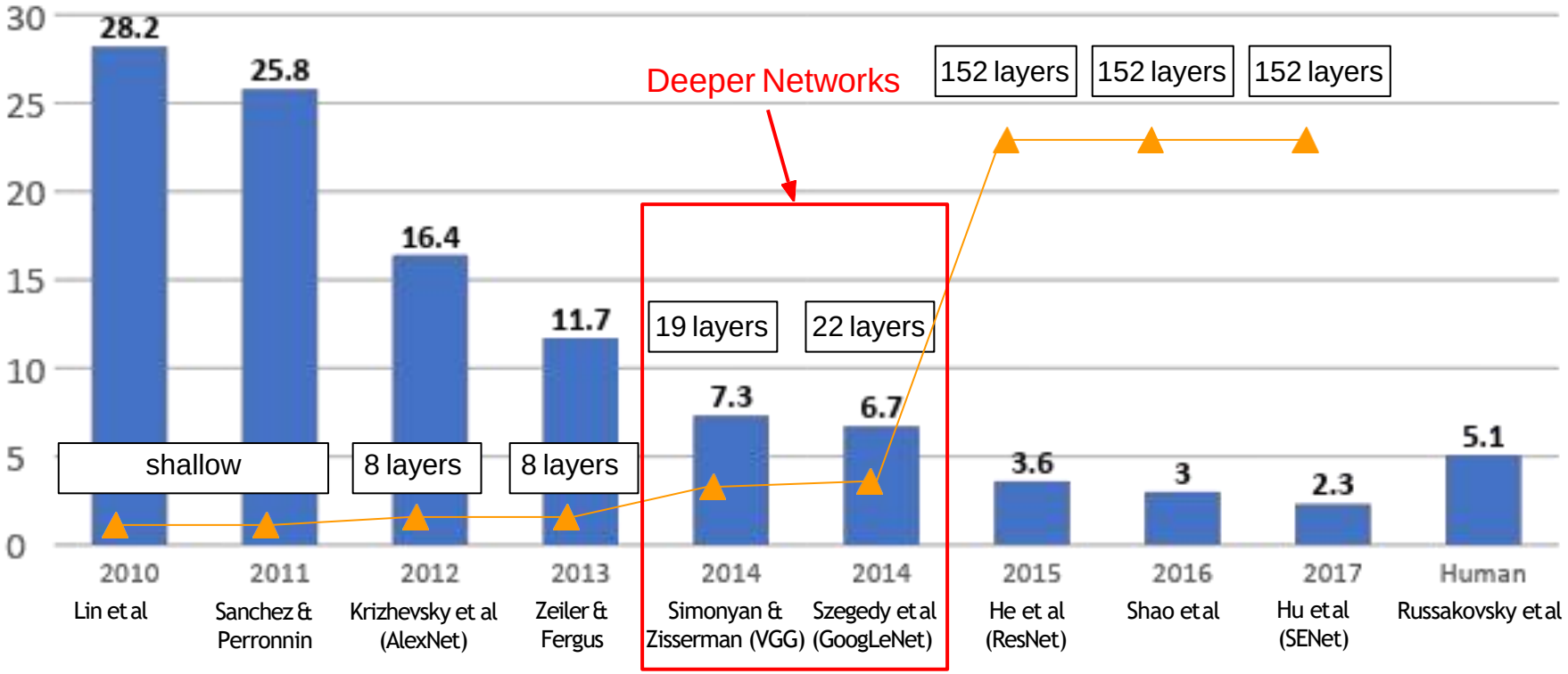
[Zeiler and Fergus, 2013]



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

ImageNet top 5 error: 16.4% -> 11.7%

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

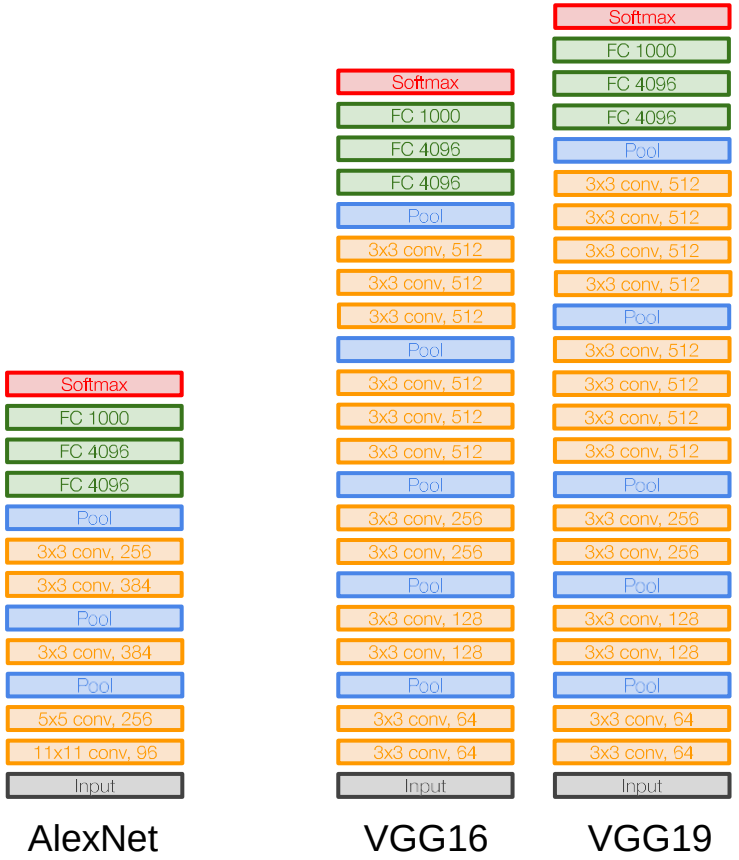
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 and 19 layers (VGG16, VGG19)

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



(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
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

Note:

Most memory is in early CONV

Most params are in late FC

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

(not counting biases)

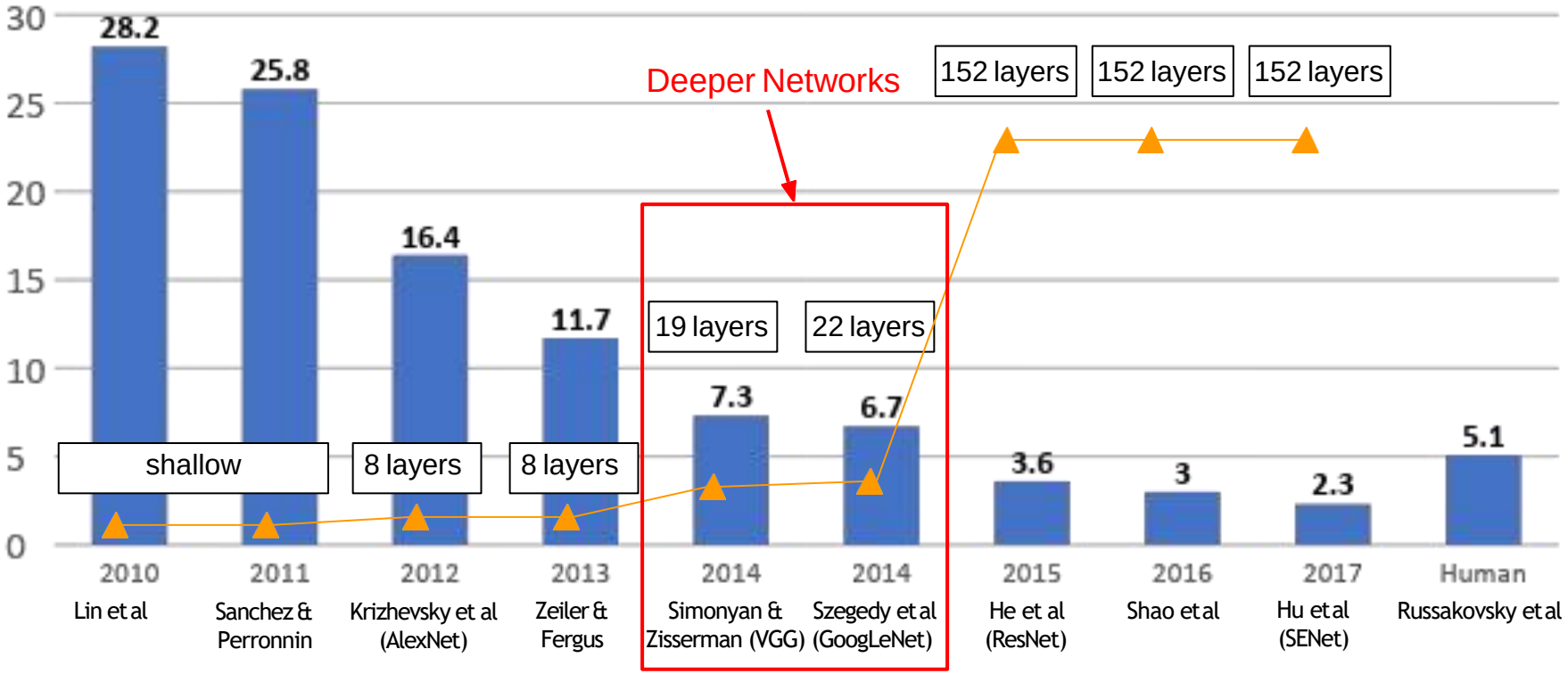
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M 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

Common names

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

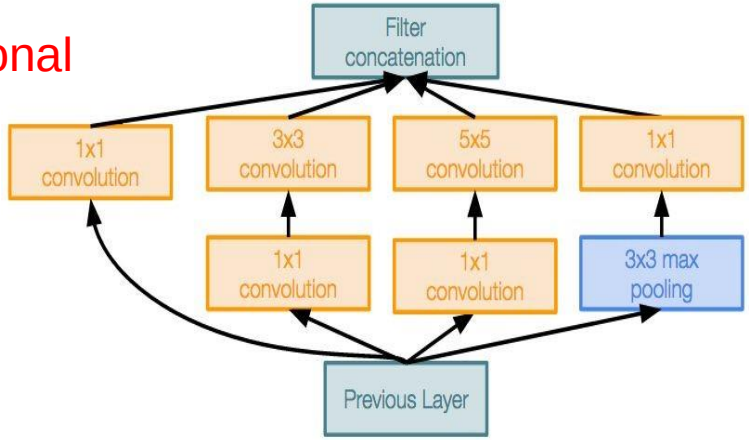


Case Study: GoogLeNet

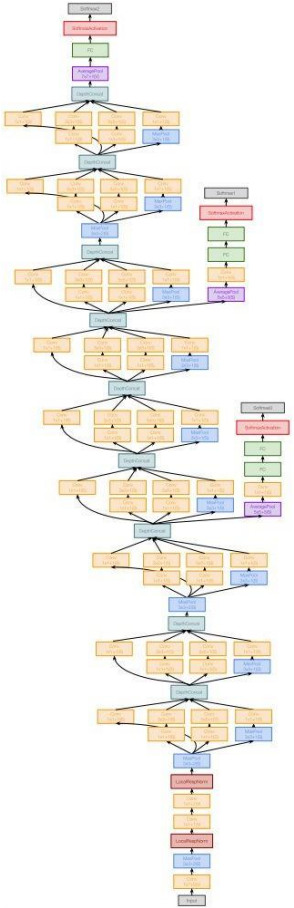
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC’14 classification winner
  (6.7% top 5 error)



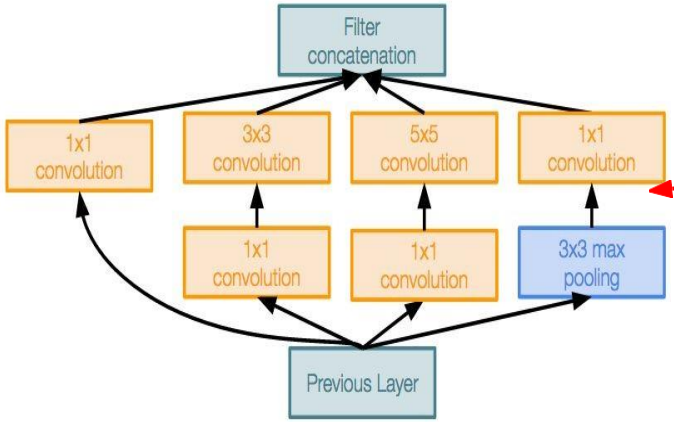
Inception module

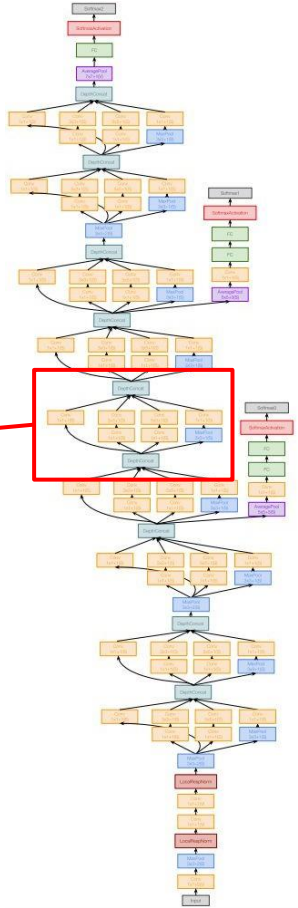


Case Study: GoogLeNet

[Szegedy et al., 2014]

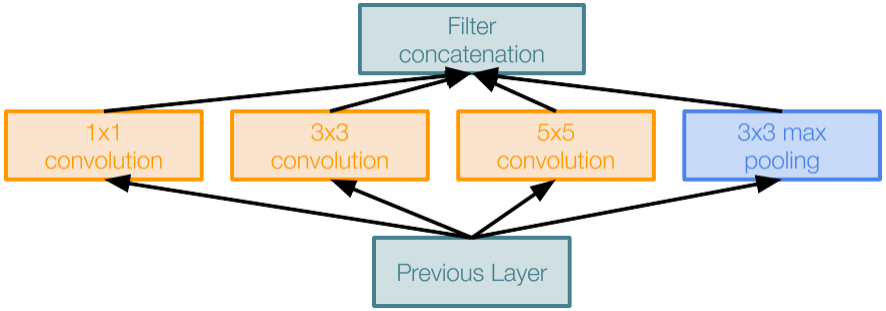
"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other





Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

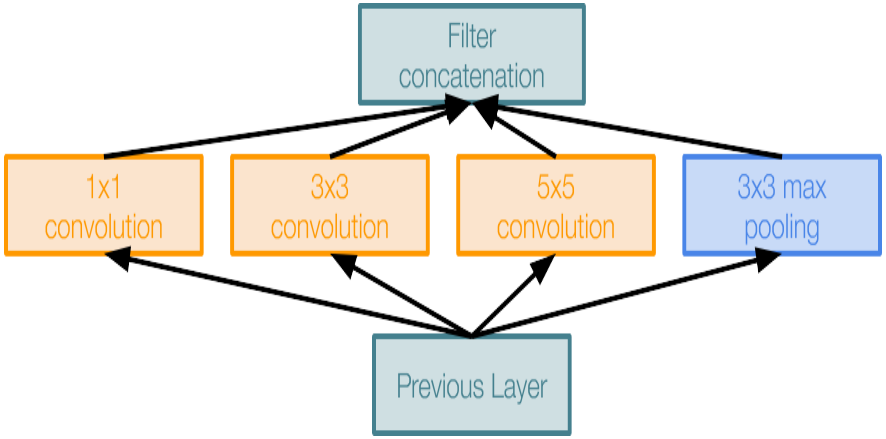
Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

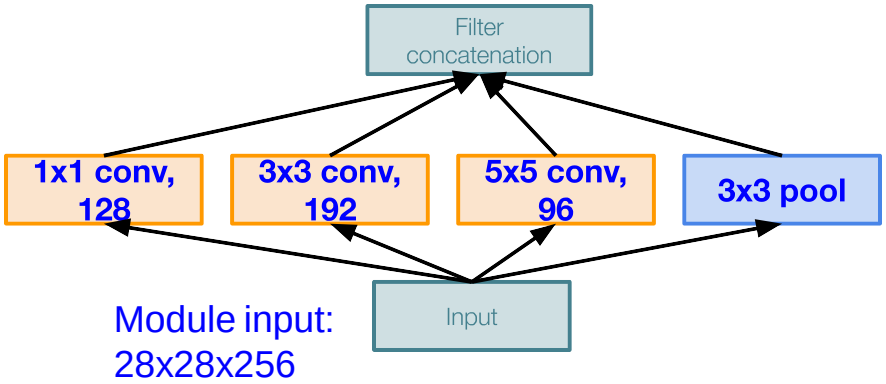
Q: What is the problem with this? [Hint: Computational complexity]

Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:



Naive Inception module

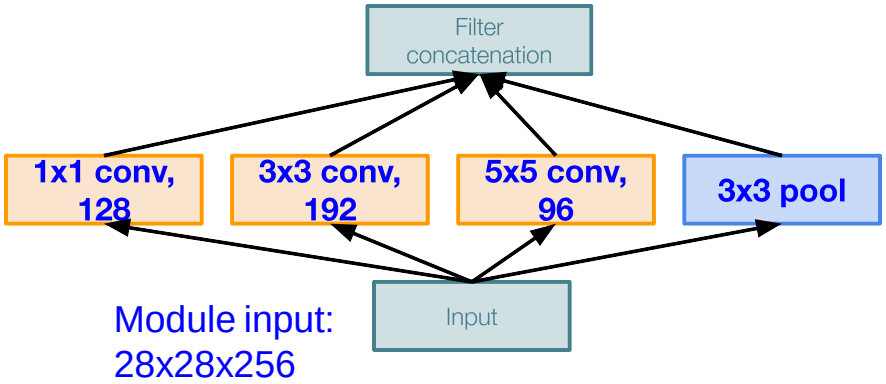
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q1: What is the output size of the 1x1 conv, with 128 filters?



Naive Inception module

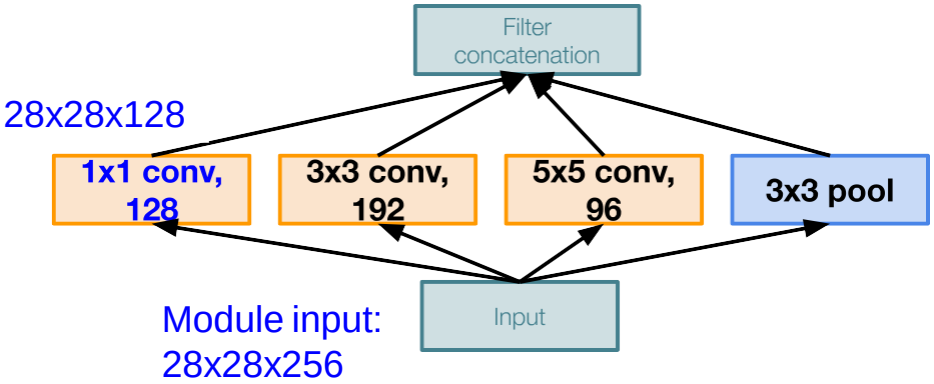
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q1: What is the output size of the 1x1 conv, with 128 filters?



Naive Inception module

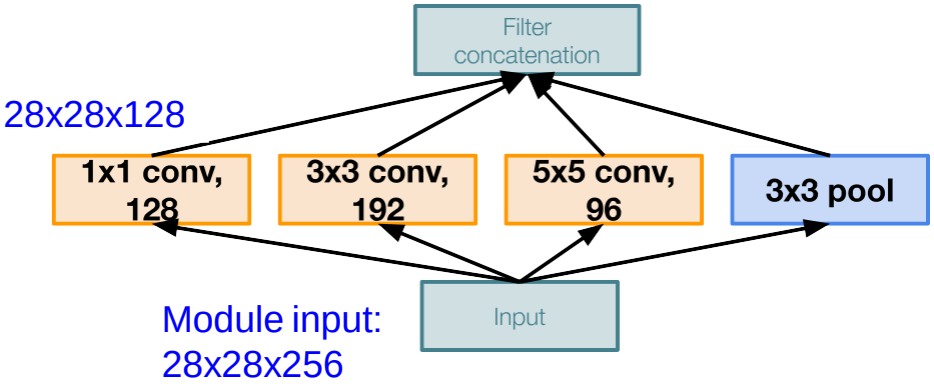
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q2: What are the output sizes of all different filter operations?



Naive Inception module

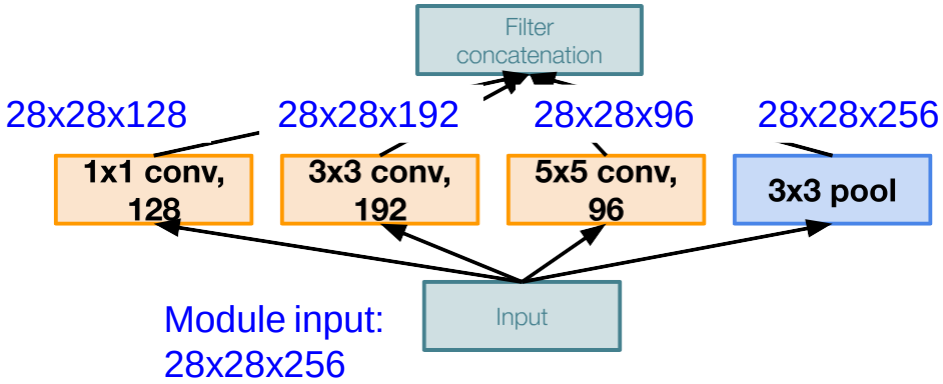
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q2: What are the output sizes of all different filter operations?



Naive Inception module

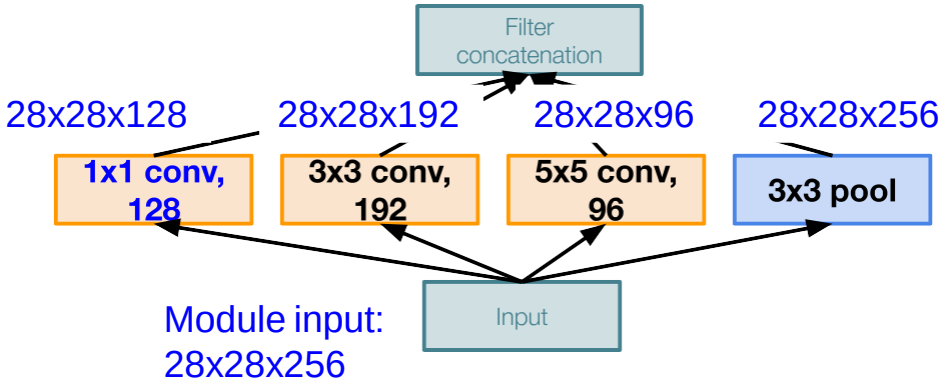
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q3:What is output size after filter concatenation?



Naive Inception module

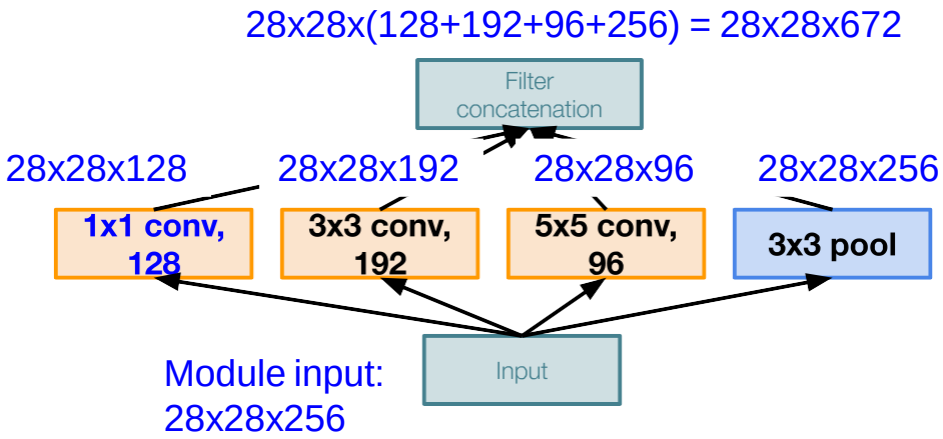
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q3:What is output size after filter concatenation?



Naive Inception module

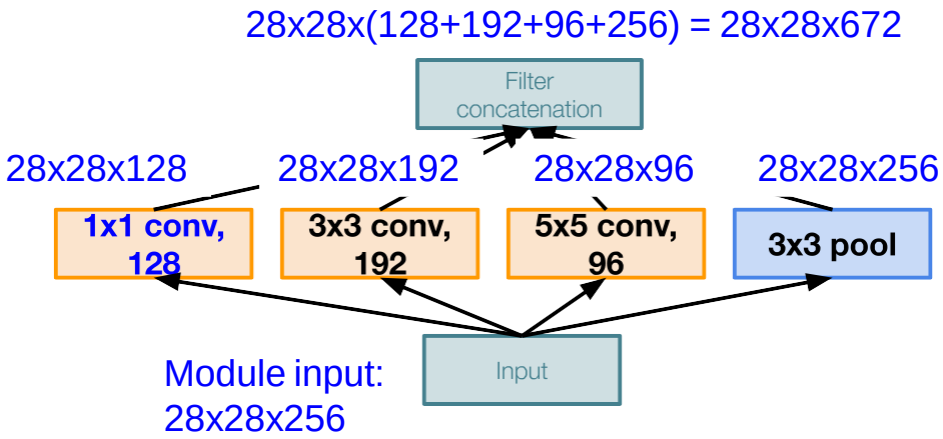
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q3:What is output size after filter concatenation?



Naive Inception module

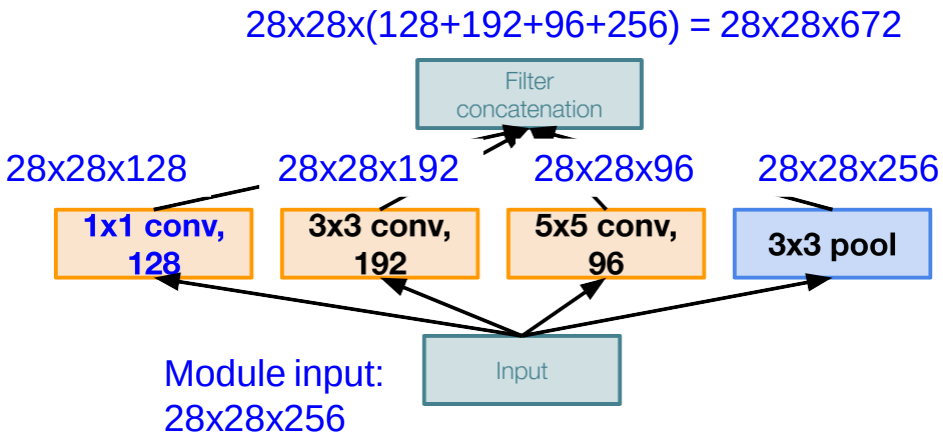
Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3:What is output size after filter concatenation?



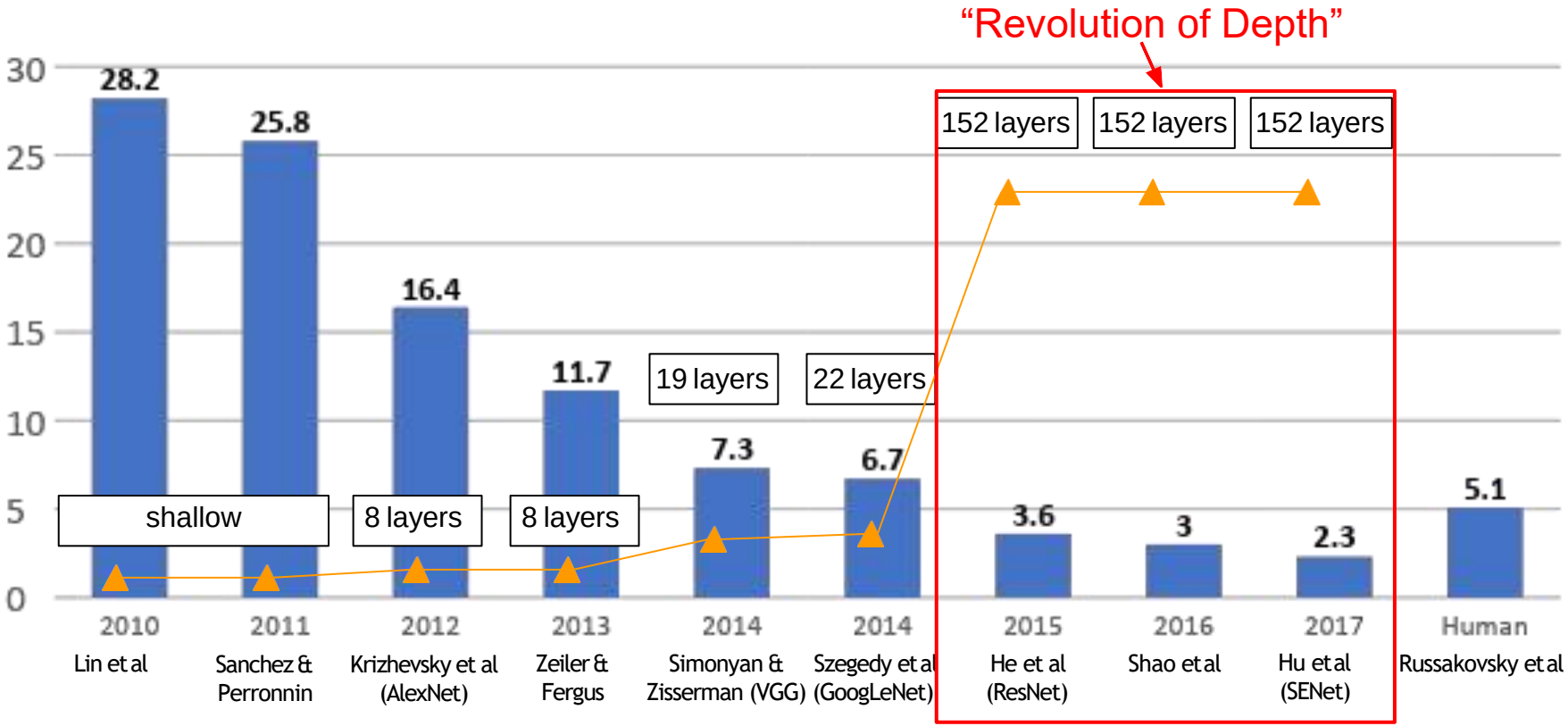
Naive Inception module

Q: What is the problem with this?
[Hint: Computational complexity]

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Very expensive compute

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

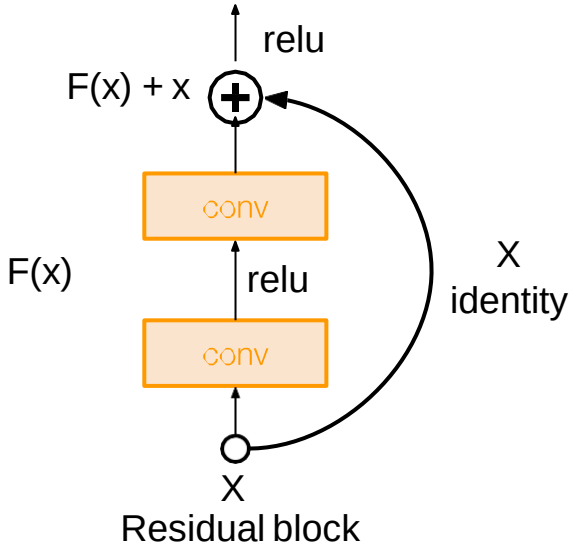


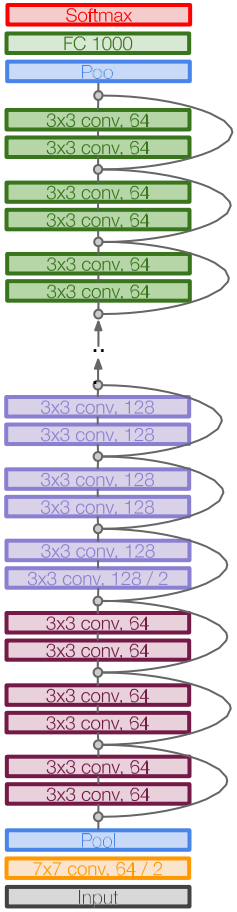
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- **152-layer model** for ImageNet
- ILSVRC'15 classification winner (**3.57%** top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

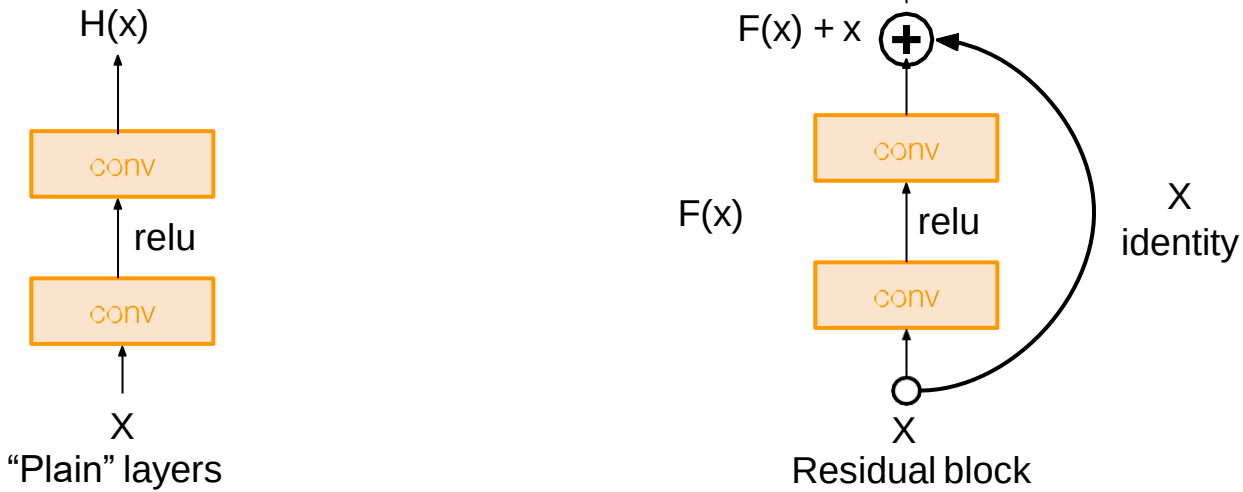




Case Study: ResNet

[He et al., 2015]

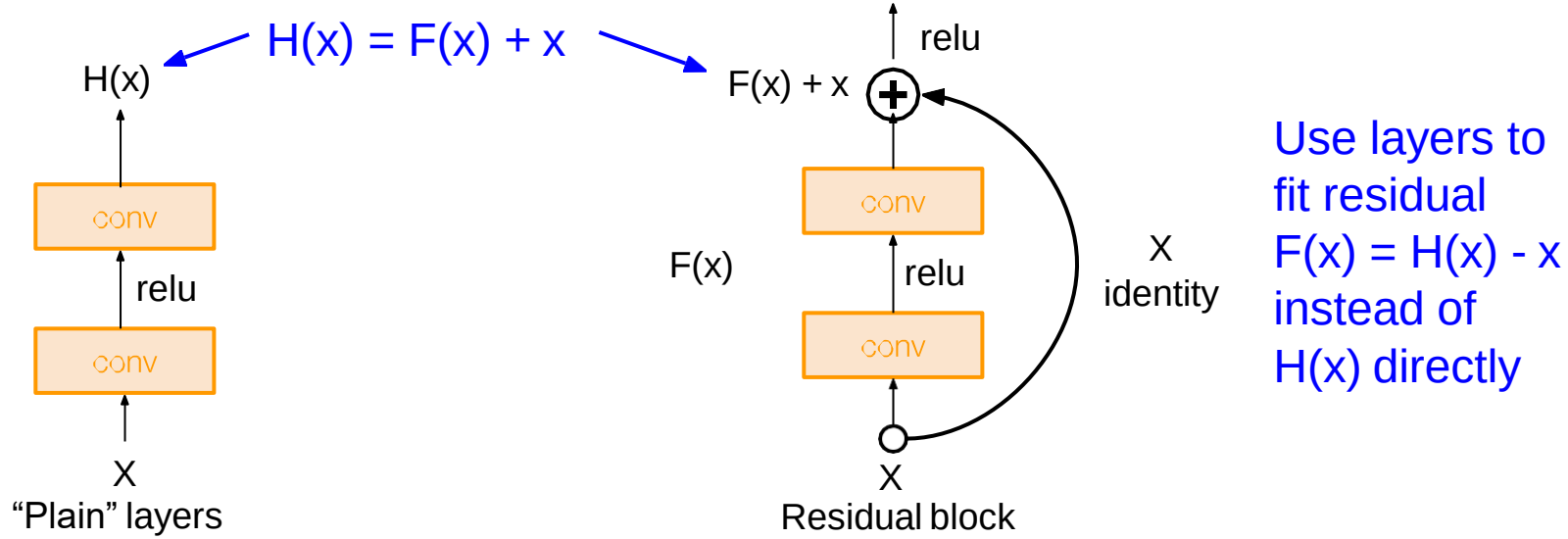
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



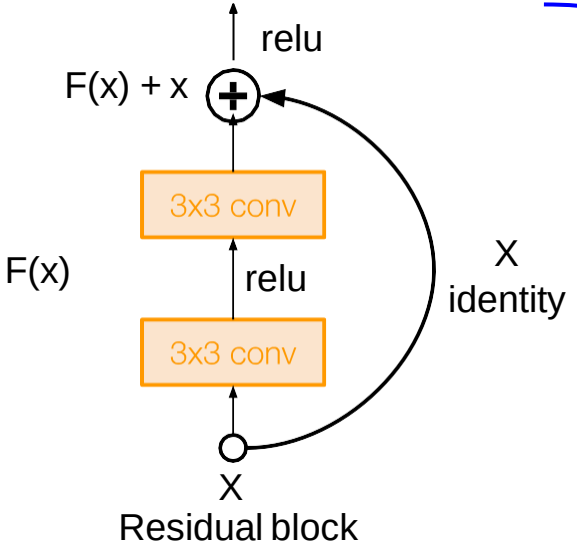
Use layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly

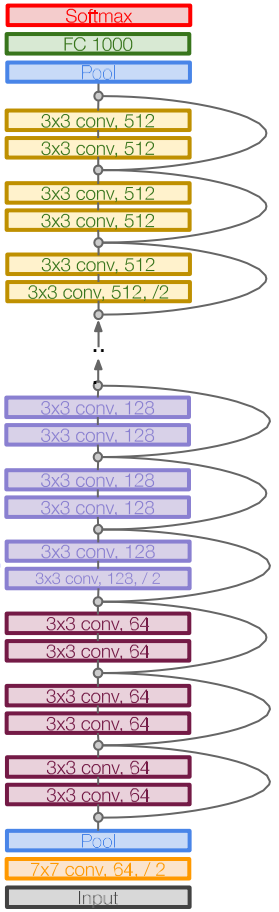
Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers





Case Study: ResNet

[He et al., 2015]

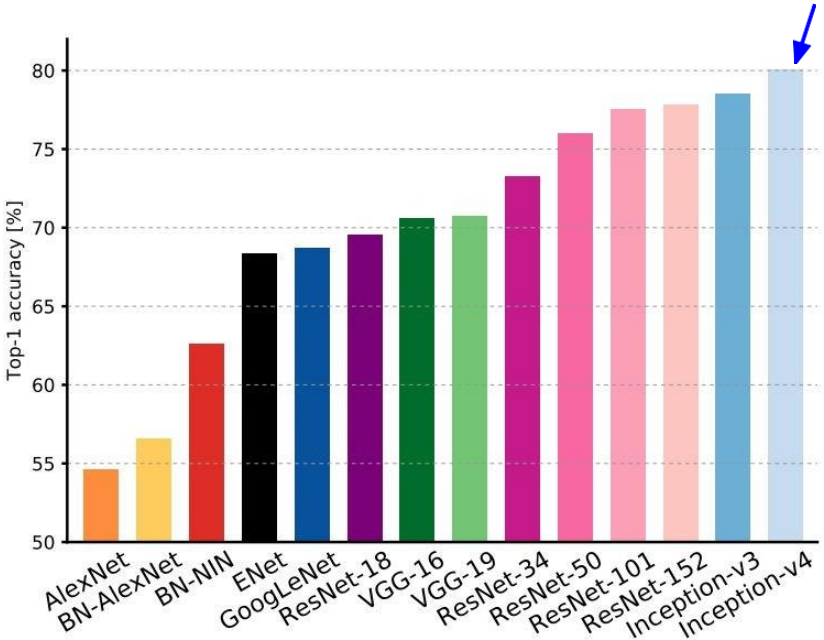
Experimental Results

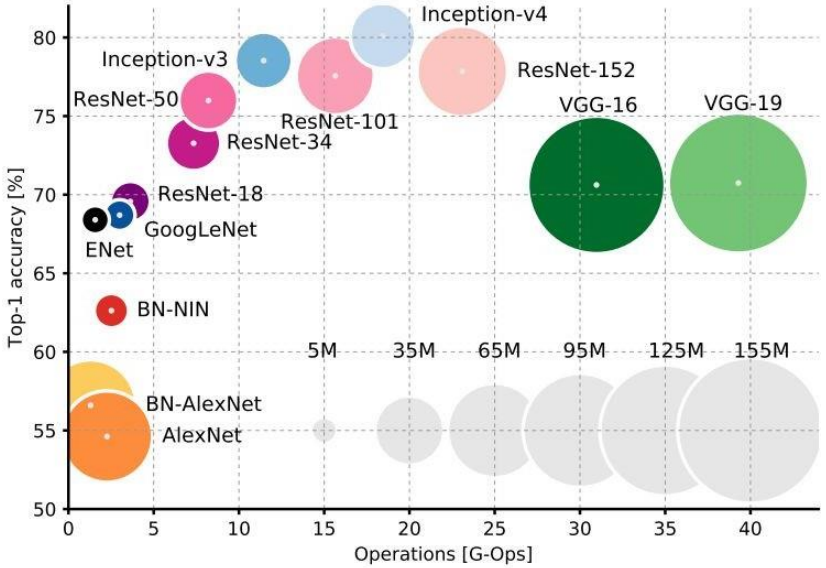
- Able to train very deep networks without degrading (152 layers on ImageNet, on Cifar)
- Deeper networks now achieve lowing training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than "human performance"! (Russakovsky 2014)

Comparing complexity...

Inception-v4: Resnet + Inception!

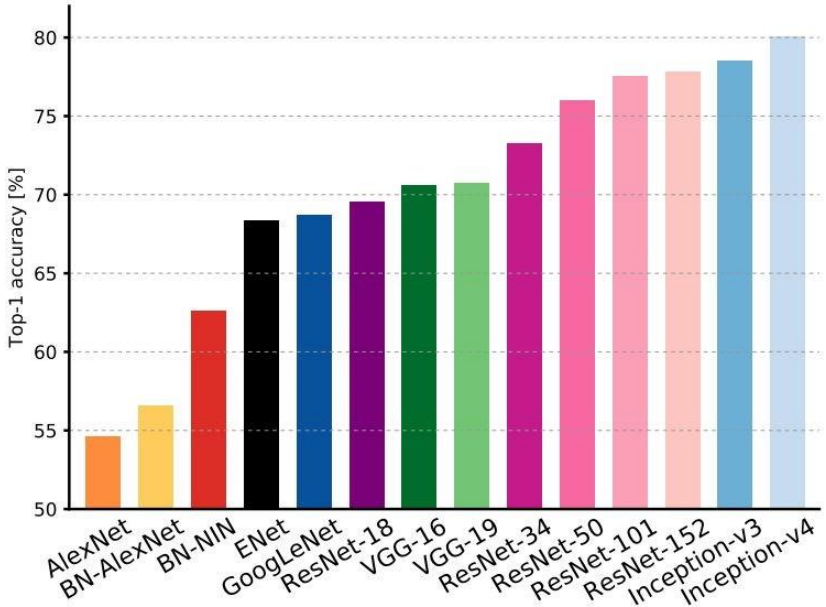


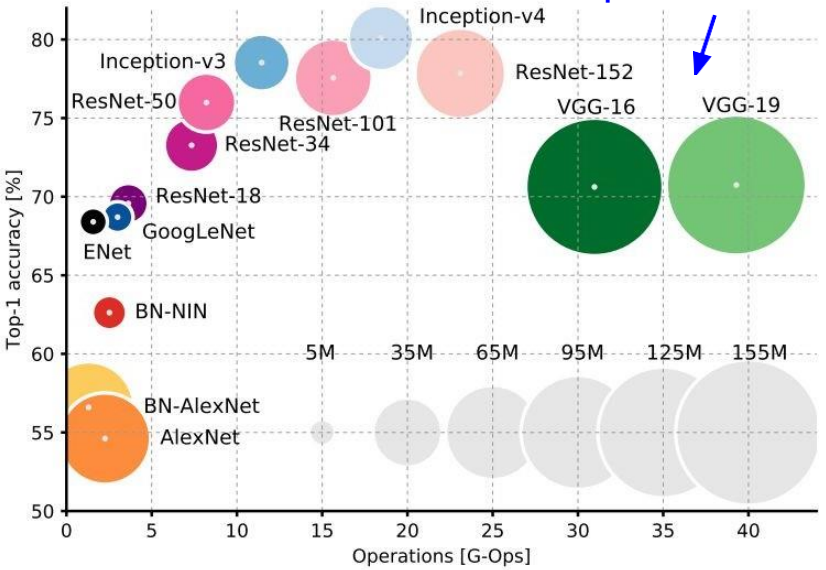


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

VGG: Highest memory, most operations

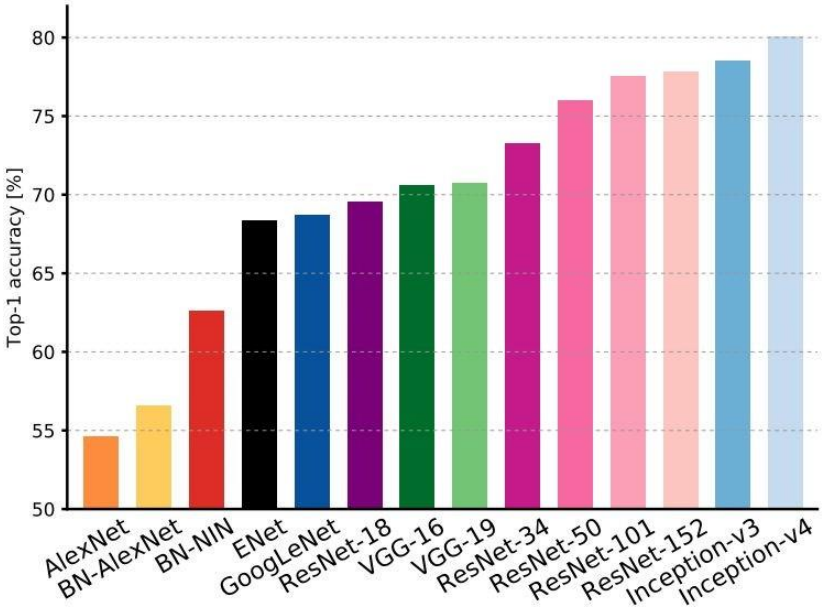


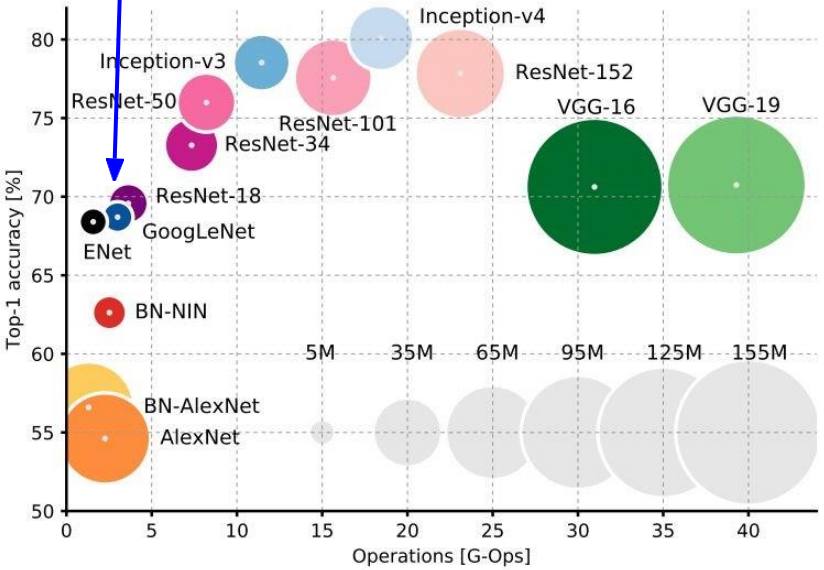


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

GoogLeNet:
most efficient

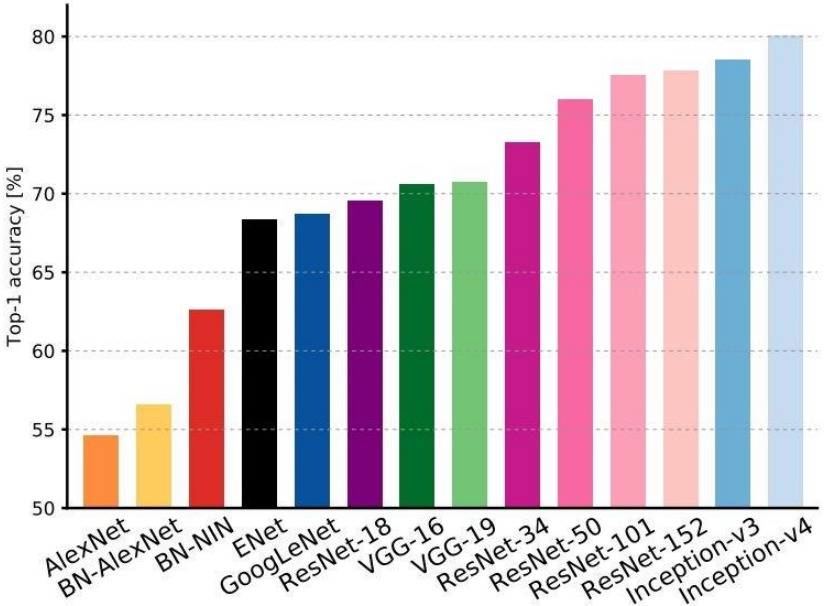


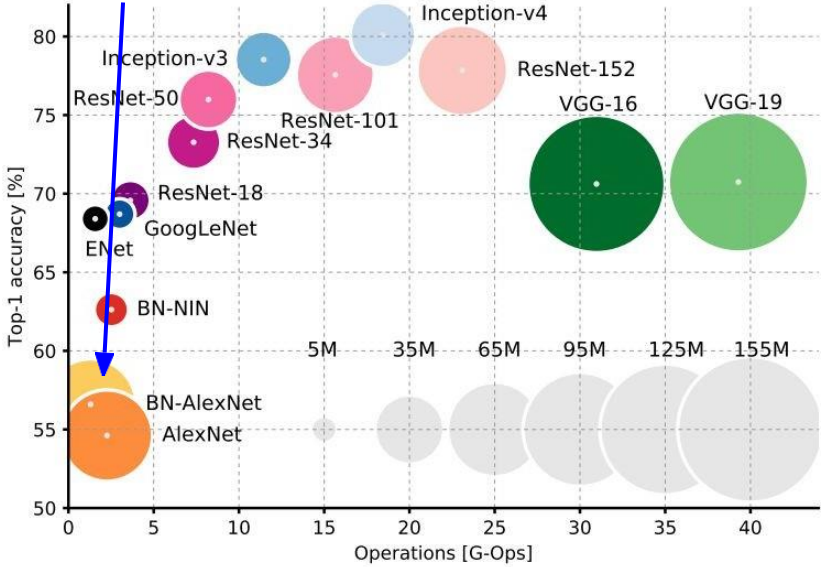


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

AlexNet:
Smaller compute, still memory heavy, lower accuracy

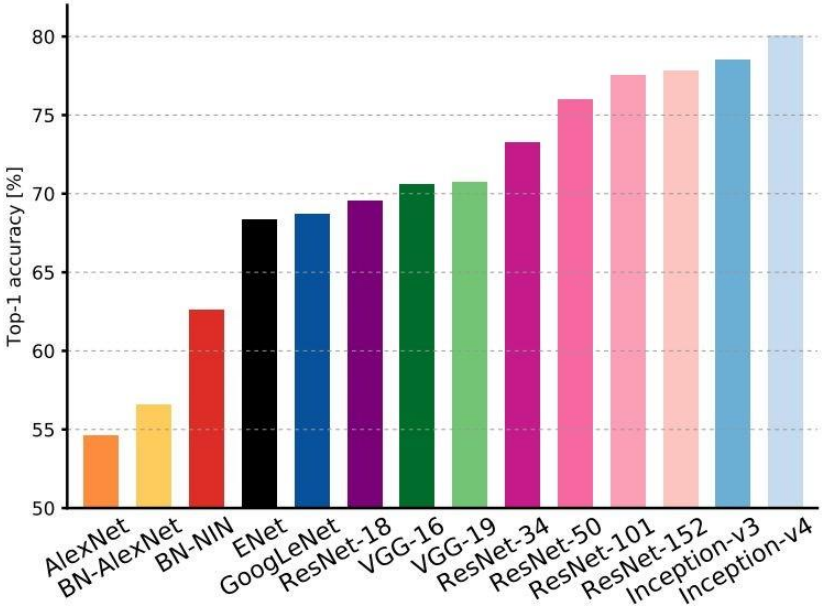


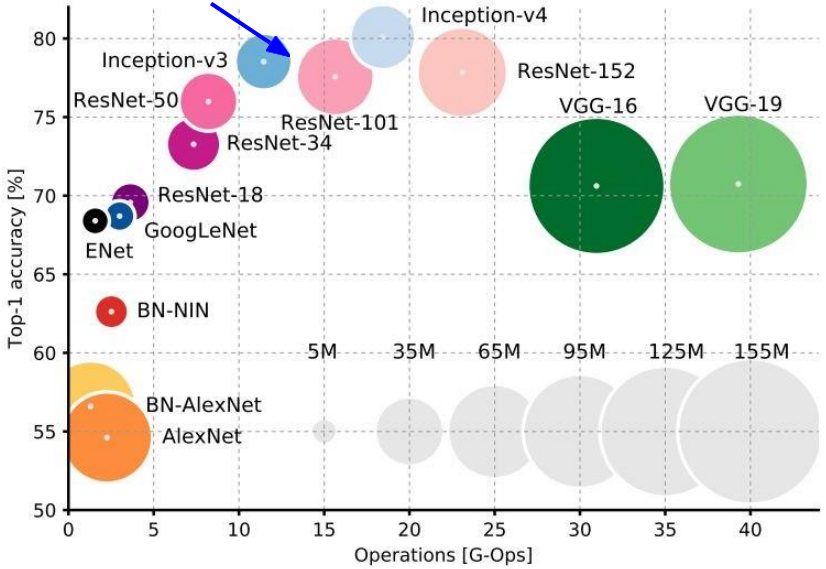


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

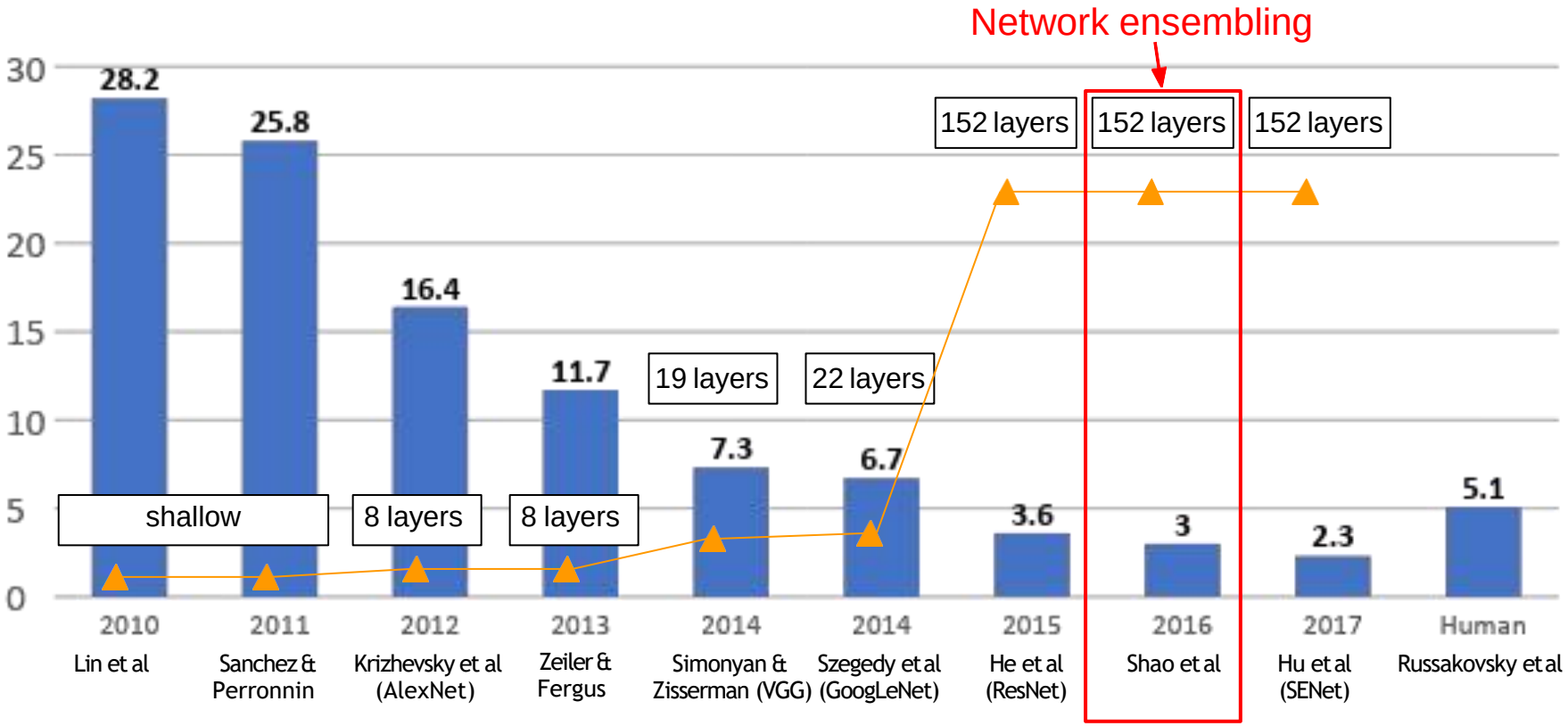
ResNet:
Moderate efficiency depending on model, highest accuracy





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Improving ResNets...

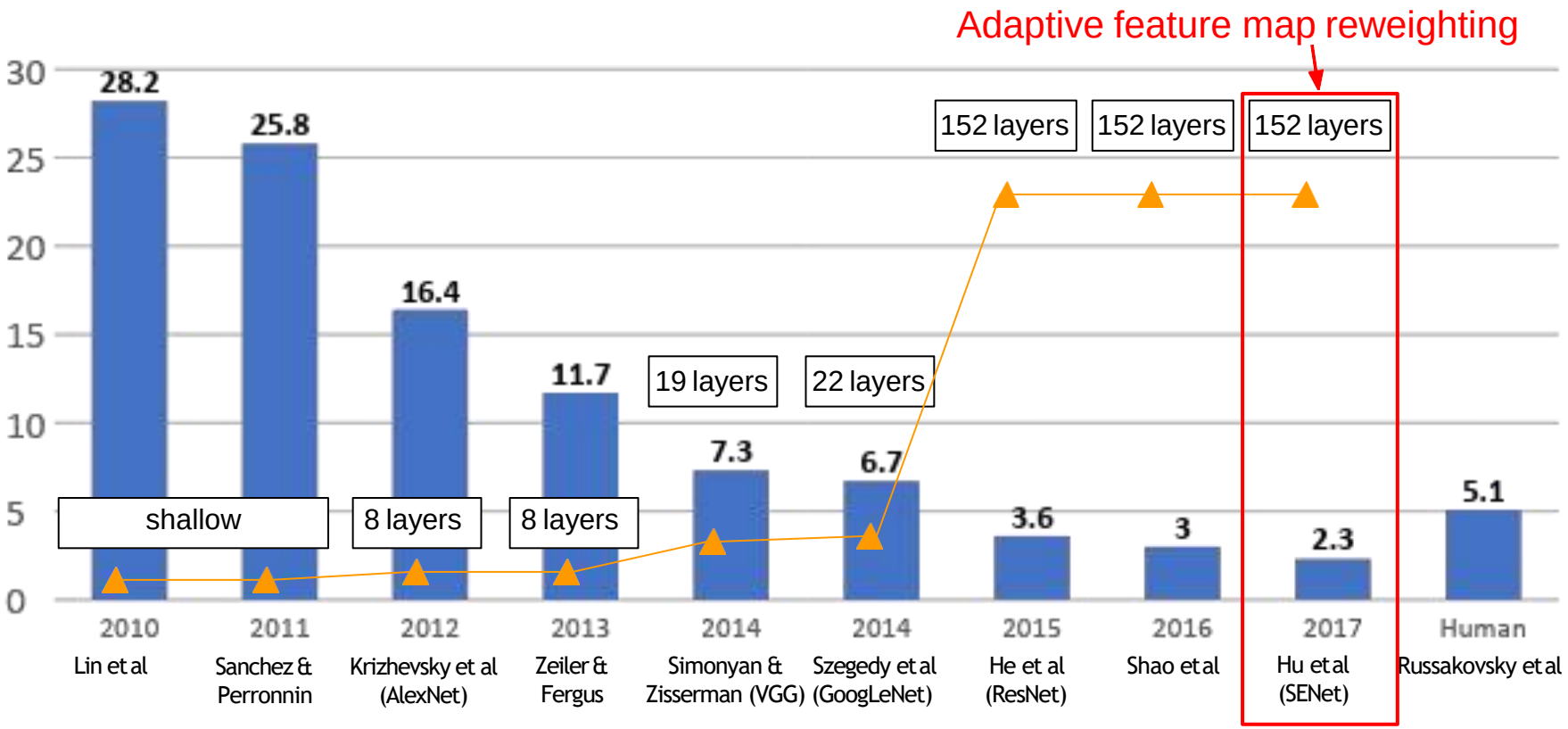
“Good Practices for Deep Feature Fusion”

[Shao et al. 2016]

- Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models
- ILSVRC’16 classification winner

	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion（Val.）	Fusion（Test）
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

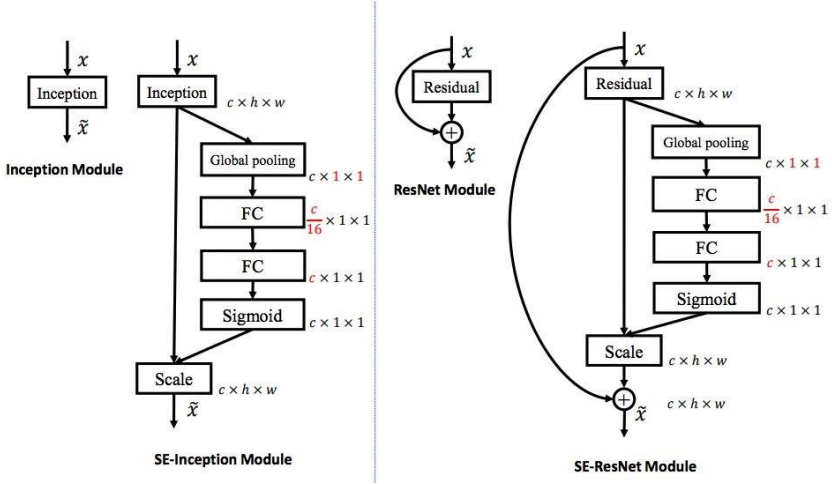


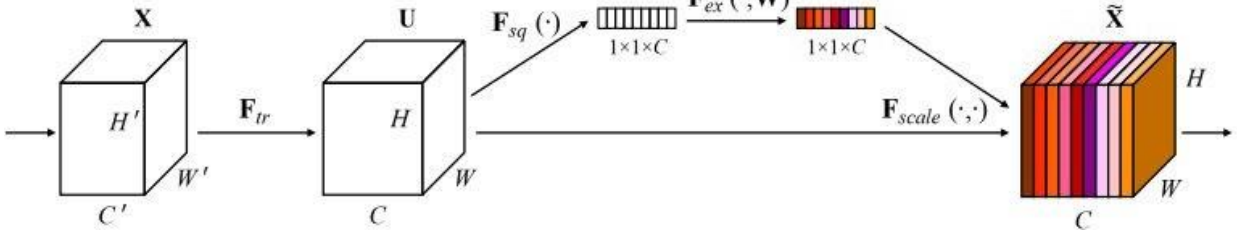
Improving ResNets...

Squeeze-and-Excitation Networks (SENet)

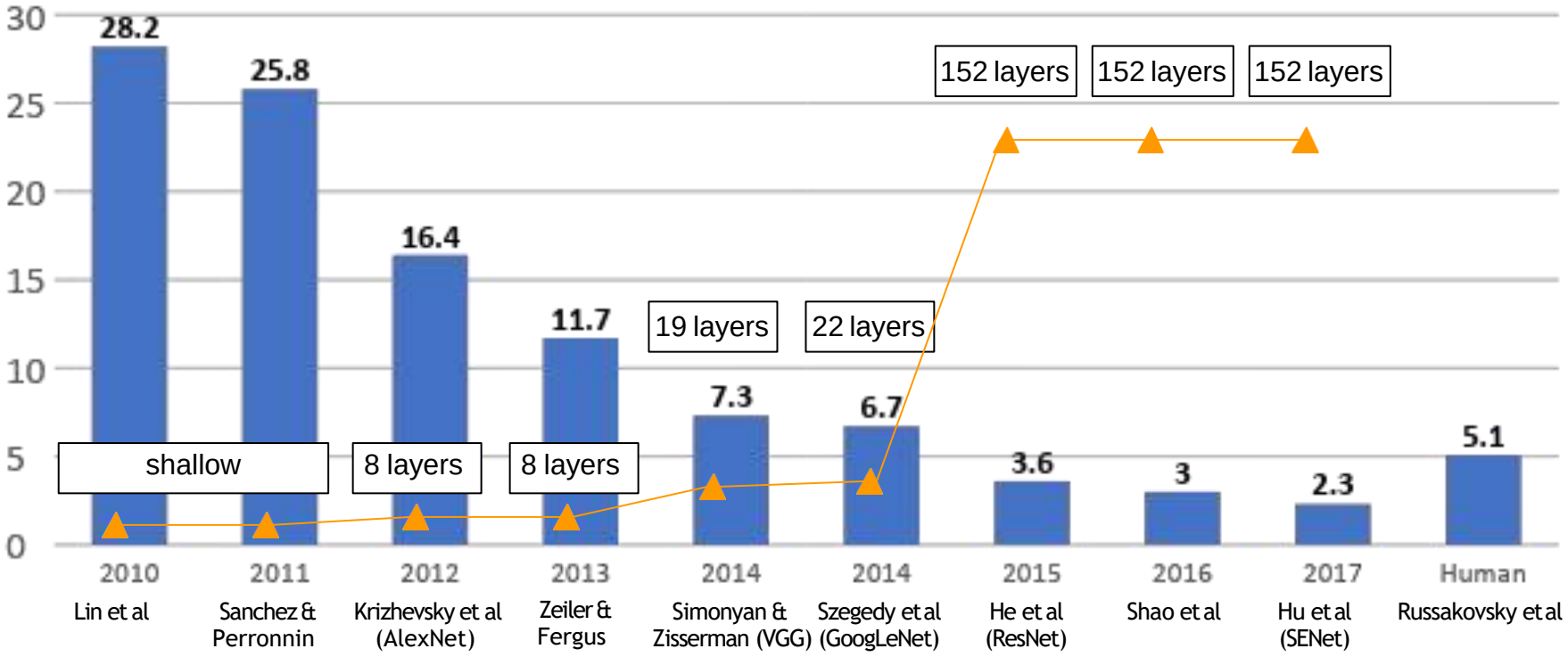
[Hu et al. 2017]

- Add a “feature recalibration” module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC’17 classification winner (using ResNeXt-152 as a base architecture)

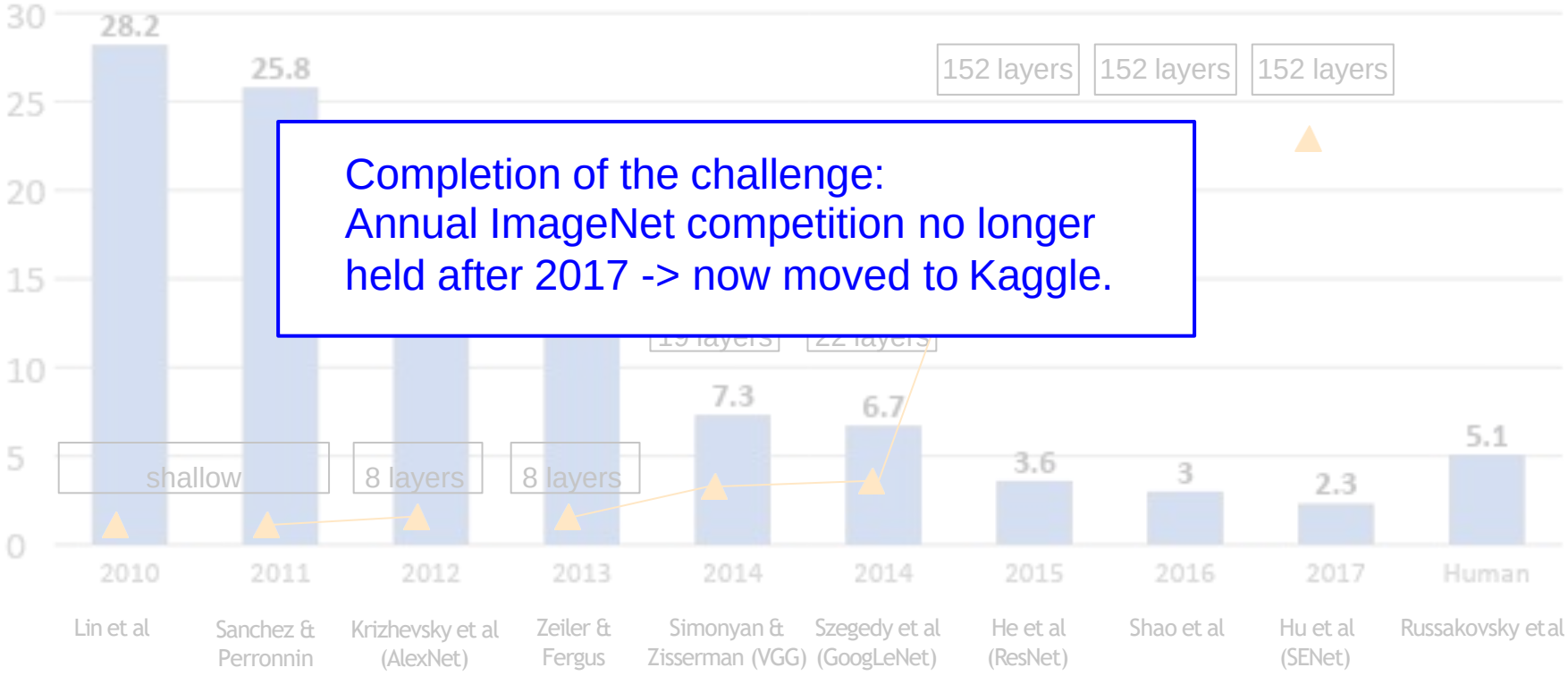




ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Completion of the challenge:
Annual ImageNet competition no longer held after 2017 -> now moved to Kaggle.

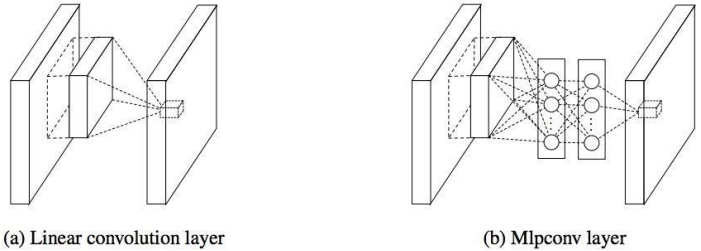
But research into CNN architectures is still flourishing

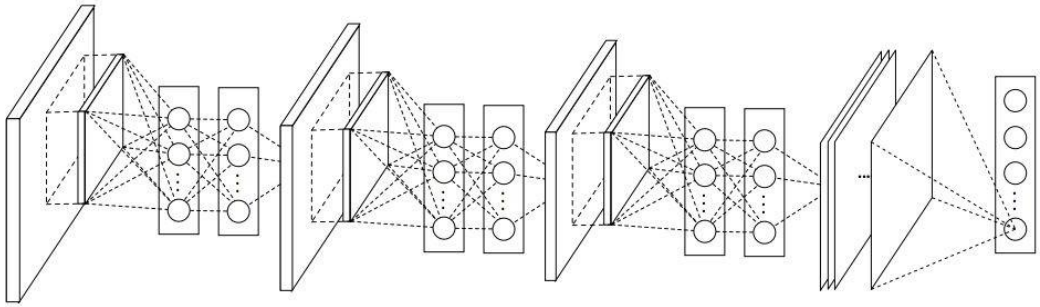
Of historical note...

Network in Network (NiN)

[Lin et al. 2014]

- Mlpconv layer with “micronetwork” within each conv layer to compute more abstract features for local patches
- Micronetwork uses multilayer perceptron
- Precursor to GoogLeNet and ResNet “bottleneck” layers
- Philosophical inspiration for GoogLeNet



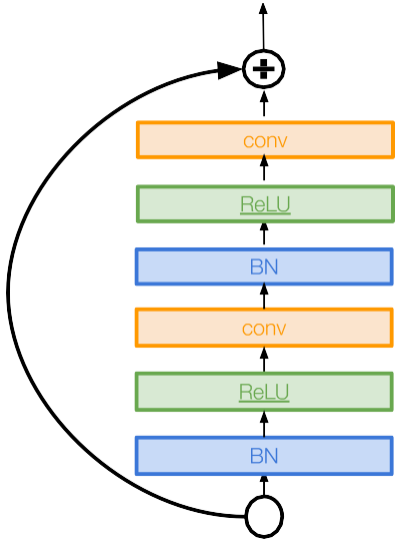


Improving ResNets...

Identity Mappings in Deep Residual Networks

[He et al. 2016]

- Improved ResNet block design from creators of ResNet
- Creates a more direct path for propagating information throughout network (moves activation to residual mapping pathway)
- Gives better performance

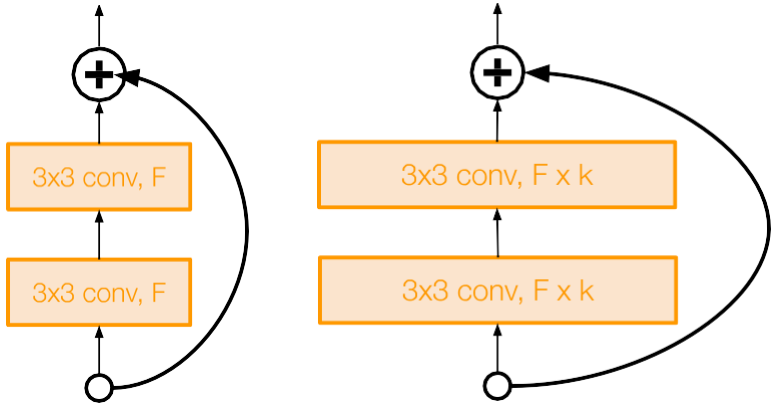


Improving ResNets...

Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- User wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)



Basic residual block

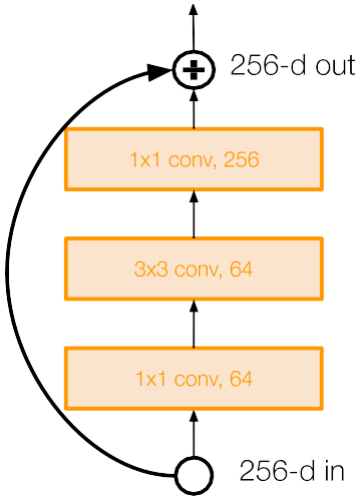
Wide residual block

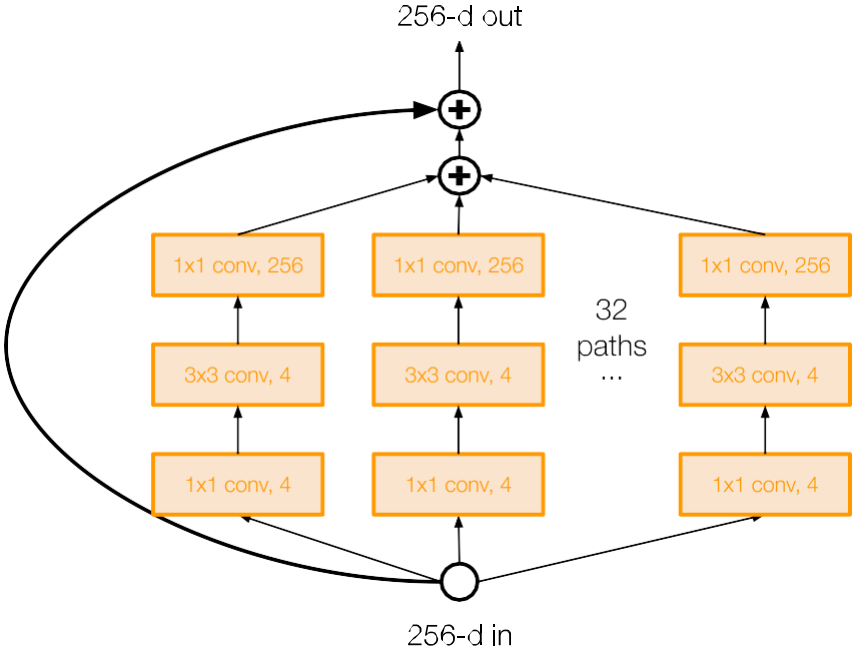
Improving ResNets...

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways (“cardinality”)
- Parallel pathways similar in spirit to Inception module



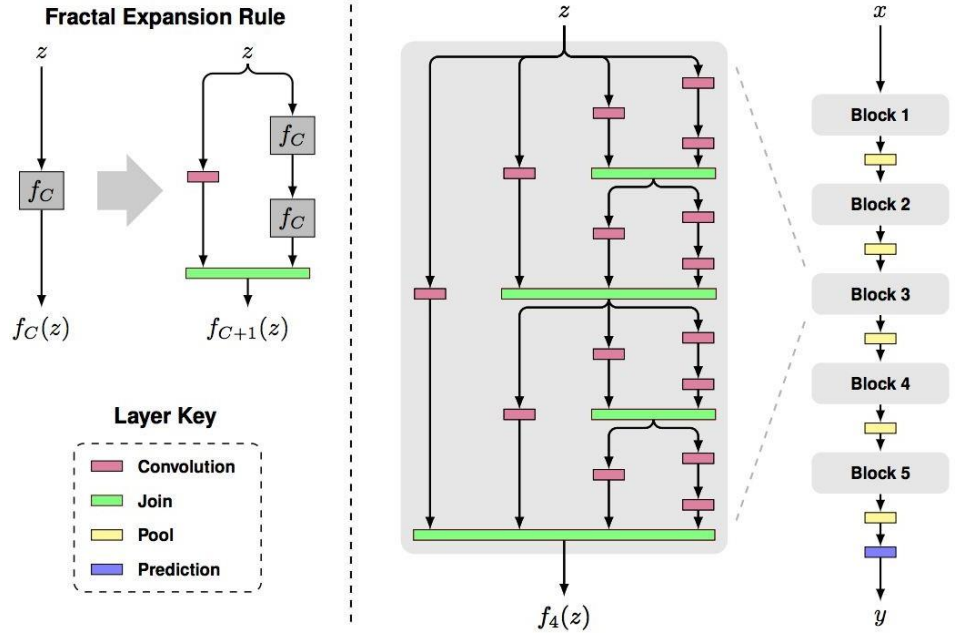


Other ideas...

FractalNet: Ultra-Deep Neural Networks without Residuals

[Larsson et al. 2017]

- Argues that key is transitioning effectively from shallow to deep and residual representations are not necessary
- Fractal architecture with both shallow and deep paths to output
- Trained with dropping out sub-paths
- Full network at test time



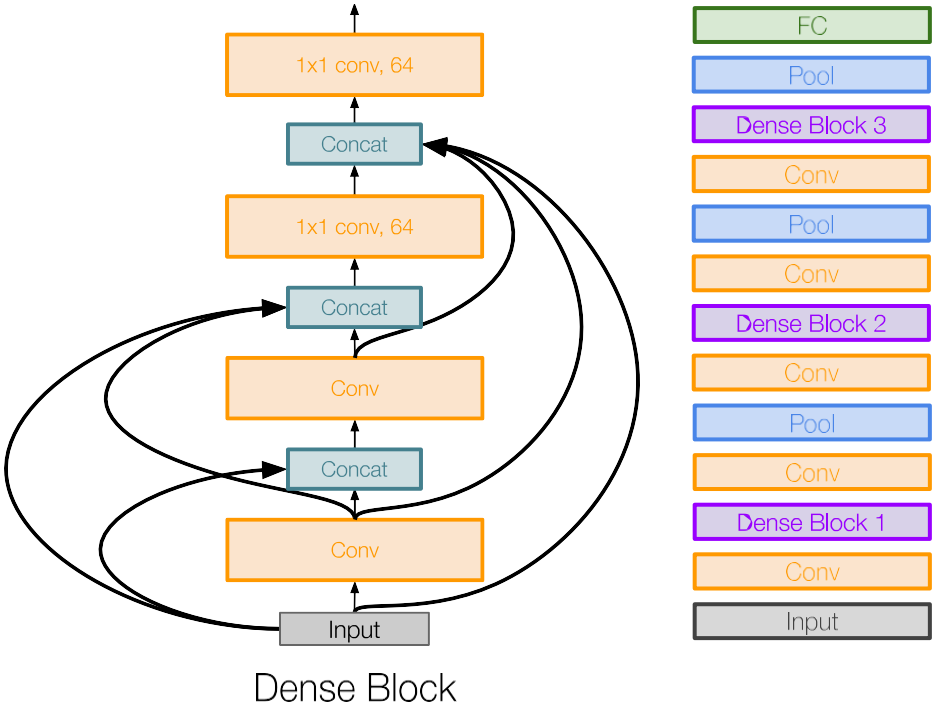
Figures copyright Larsson et al., 2017. Reproduced with permission.

Other ideas...

Densely Connected Convolutional Networks

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse

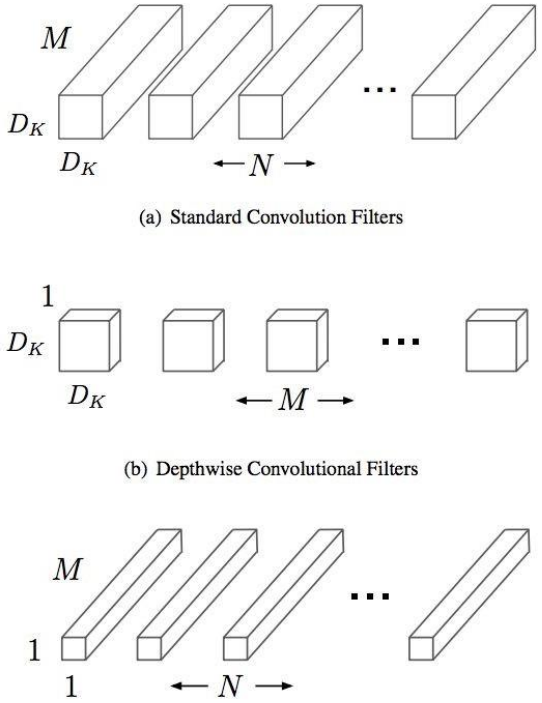


Efficient networks...

MobileNets: Efficient Convolutional Neural Networks for Mobile Applications

[Howard et al. 2017]

- Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1x1 convolution that is much more efficient
- Much more efficient, with little loss in accuracy
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)
- Other works in this space e.g. ShuffleNet (Zhang et al. 2017)

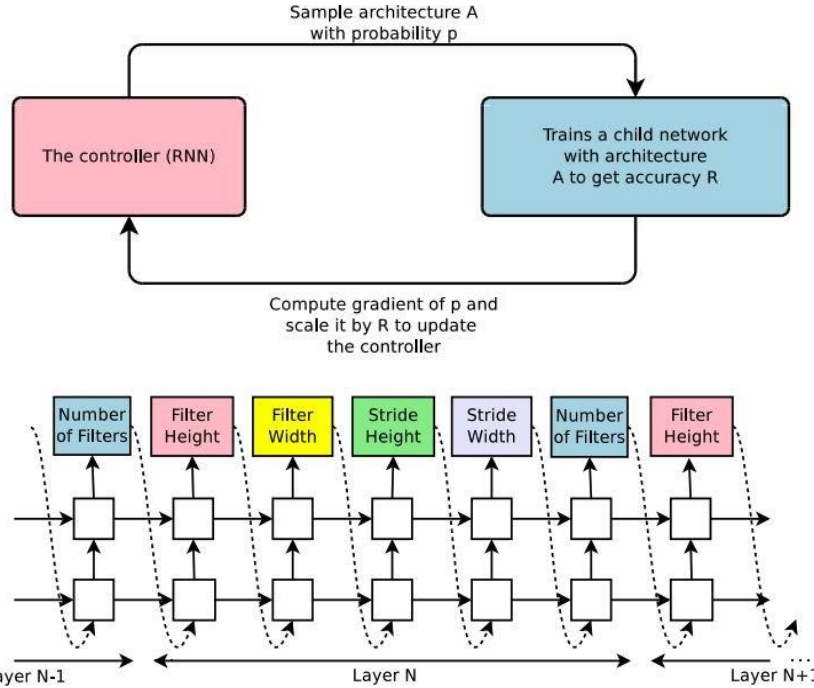


Meta-learning: Learning to learn network architectures...

Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- “Controller” network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  1) Sample an architecture from search space
  2) Train the architecture to get a “reward” R corresponding to accuracy
  3) Compute gradient of sample probability, and scale by R to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)



Summary: CNN Architectures

Case Studies

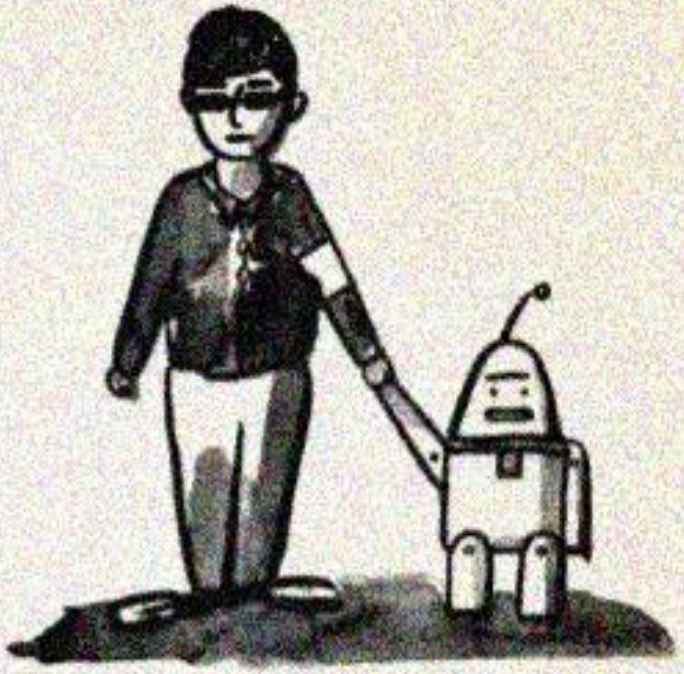
- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

- SENet
- NiN (Network in Network)
- Wide ResNet
- ResNeXT
- DenseNet
- FractalNet
- MobileNets
- NASNet

Summary: CNN Architectures

- Many popular architectures available in model zoos
- ResNet and SENet currently good defaults to use
- Networks have gotten increasingly deep over time
- Many other aspects of network architectures are also continuously being investigated and improved
- Even more recent trend towards meta-learning



THANK YOU!

QUESTIONS?