FACULTY OF MATHEMATICS,
PHYSICS AND INFORMATICS
Comenius University
Bratislava

Neural Networks for Computer Vision

Lecture 8: CNN Architectures

Ing. Viktor Kocur, PhD., RNDr. Zuzana Černeková, PhD.

9.11.2021

Contents

Acknowledgment

Some of the slides are directly adopted from slides for CS231n[1] course at Stanford University!

[1] Fei-Fei Li, Ranjay Krishna, and Danfei Xu. *Stanford CS231n lecture slides*. `http://cs231n.stanford.edu/slides/`

Recap - CNNs

Main building blocks of a CNN

- Convolutional layers
- Pooling layers
- Fully-connected layers
- Activations

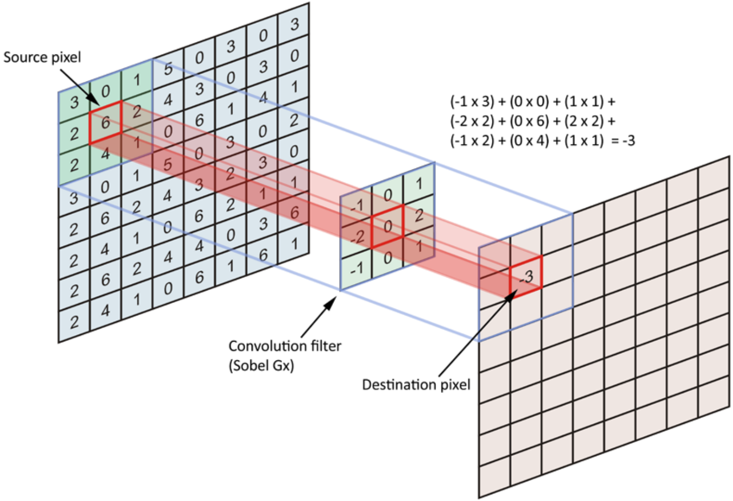
Recap - CNNs

Main building blocks of a CNN

- Convolutional layers
- Pooling layers
- Fully-connected layers
- Activations

There are many potential combinations! We will now discuss historically most important architectures and what can we learn from them!

Recap - Convolution

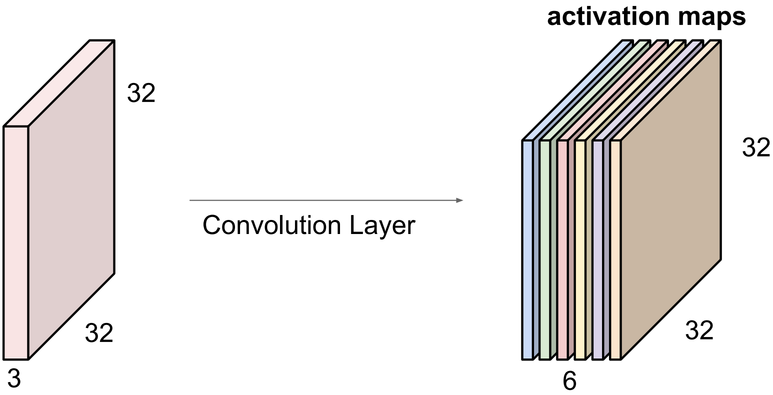


For one dimension of the input image (height or width independetly) after applying convolution we get the following formula for the dimension of the output:

$$N_{out} = \frac{N_{in} - F + 2P}{S} + 1, \tag{1}$$

where N_{out} is the output size, N_{in} is the input size, F is the size of the kernel, P is the padding and S is the stride.

Recap - Convolution



Each conv filter outputs a “slice” in the activation

Recap - Parameters

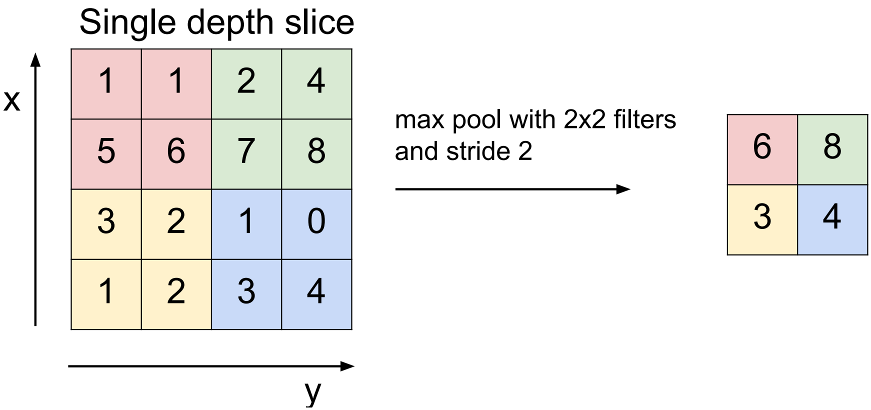
For a convolution with C_{in} input channels, C_{out} output channels, kernel size $K_h \times K_w$ the number of parameters is:

$$P_{conv} = (C_{in} \cdot K_w \cdot K_h + 1) \cdot C_{out} \tag{2}$$

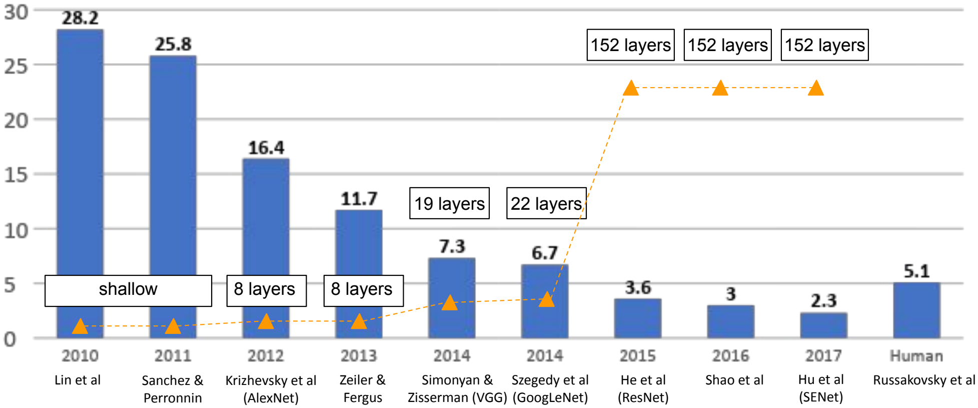
For a fully-connected layer with C_{in} input channels/neurons and C_{out} output channels is:

$$P_{fc} = (C_{in} + 1) * C_{out} \tag{3}$$

Recap - Pooling



ILSVRC Winners



AlexNet

AlexNet[2] won the ILSVRC 2012 and started CNN revolution!

- Based on earlier work - LeNet[3]
- Relied on heavy data augmentation
- Training on two GPUs
- Used ReLU activations
- Batch size: 128
- SGD momentum + manual learning rate changes
- L2 regularization
- Ensemble of 7 CNNs to reduce error

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105

[3] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition." In: *Neural computation* 1.4 (1989), pp. 541–551

AlexNet vs LeNet

LeNet

Image: 28 (height) × 28 (width) × 1 (channel)
↓
Convolution with 5×5 kernel+2padding:28×28×6
↓ sigmoid
Pool with 2×2 average kernel+2 stride:14×14×6
↓
Convolution with 5×5 kernel (no pad):10×10×16
↓ sigmoid
Pool with 2×2 average kernel+2 stride: 5×5×16
↓ flatten
Dense: 120 fully connected neurons
↓ sigmoid
Dense: 84 fully connected neurons
↓ sigmoid
Dense: 10 fully connected neurons
↓
Output: 1 of 10 classes

AlexNet

Image: 224 (height) × 224 (width) × 3 (channels)
↓
Convolution with 11×11 kernel+4stride:54×54×96
↓ ReLu
Pool with 3×3 max. kernel+2 stride: 26×26×96
↓
Convolution with 5×5 kernel+2 pad:26×26×256
↓ ReLu
Pool with 3×3 max.kernel+2stride:12×12×256
↓
Convolution with 3×3 kernel+1 pad:12×12×384
↓ ReLu
Convolution with 3×3 kernel+1 pad:12×12×384
↓ ReLu
Convolution with 3×3 kernel+1 pad:12×12×256
↓ ReLu
Pool with 3×3 max.kernel+2stride:5×5×256
↓ flatten
Dense: 4096 fully connected neurons
↓ ReLu, dropout p=0.5
Dense: 4096 fully connected neurons
↓ ReLu, dropout p=0.5
Dense: 1000 fully connected neurons
↓
Output: 1 of 1000 classes

ZFNet

ZFNet[4] won the challenge next year with some modifications:

- 7×7 stride 2 conv instead of 11×11 stride 4 in first layer
- More channels

[4] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks." In: *European conference on computer vision*. Springer. 2014, pp. 818–833

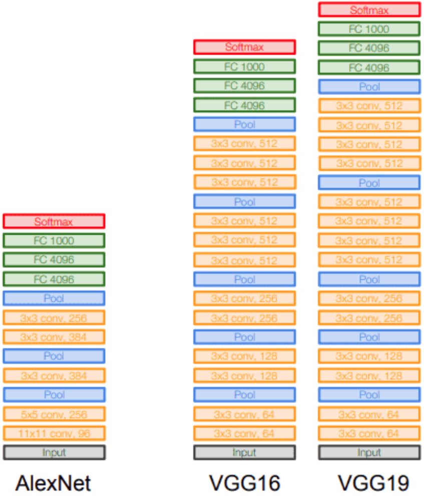
VGG

The next significant advance were VGG[5] networks. The main differences were:

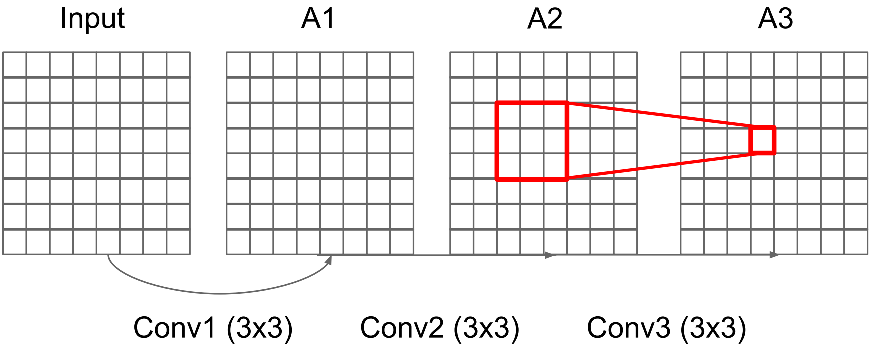
- Using only 3×3 convolutions
- Deeper networks are better!
- Trained early layers first without later ones then joined them together.

[5] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014)

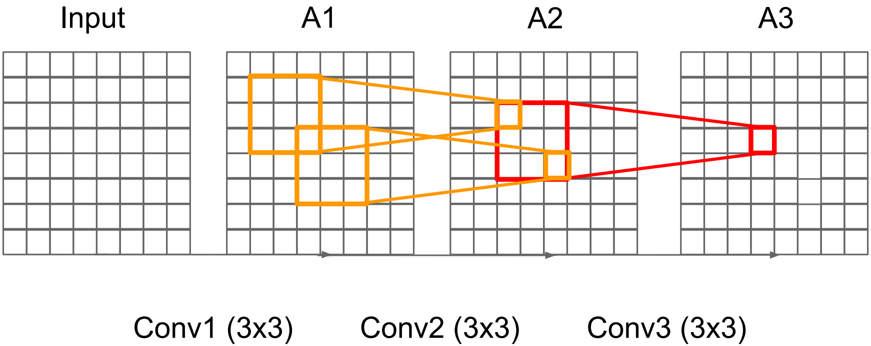
VGG


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

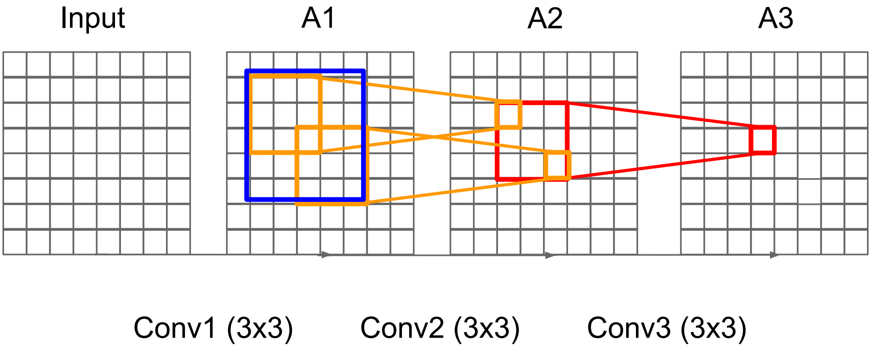
Stacking convolutions with smaller kernels



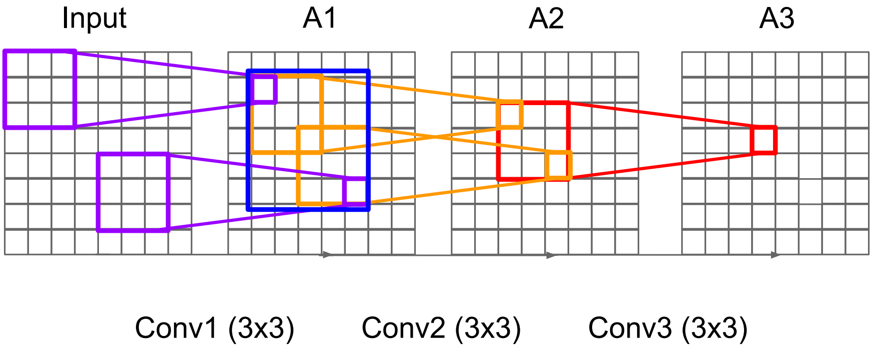
Stacking convolutions with smaller kernels



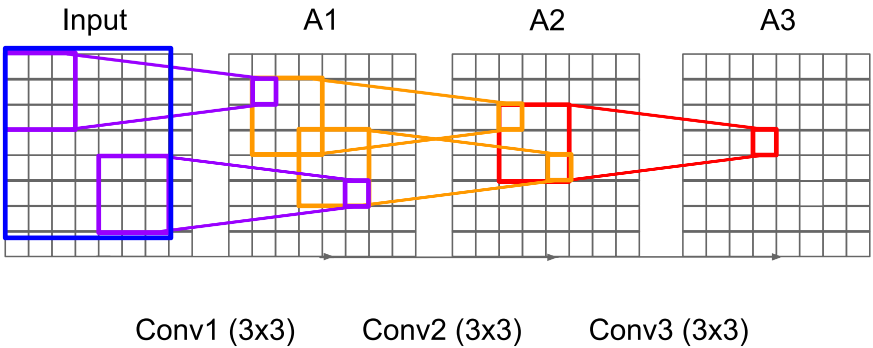
Stacking convolutions with smaller kernels



Stacking convolutions with smaller kernels



Stacking convolutions with smaller kernels



VGG parameters and memory

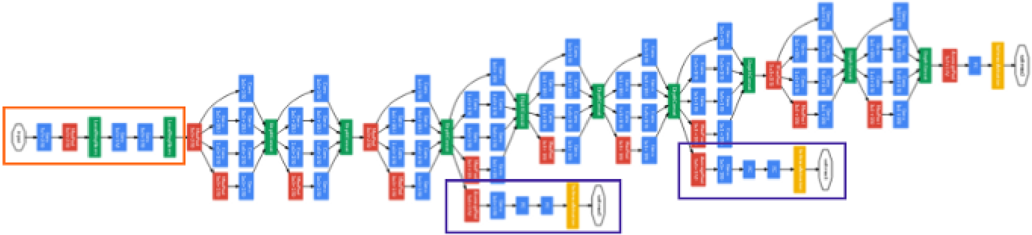
(not counting biases)

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 (for a forward pass)
TOTAL params: 138M parameters

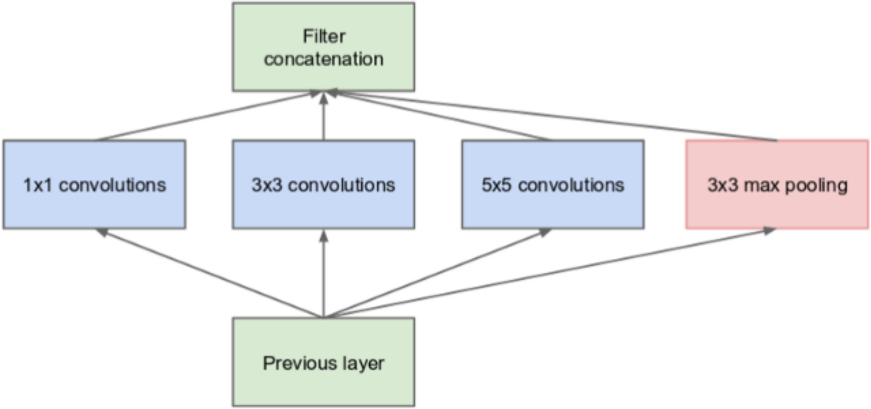
VGG16

The winner of ILSVRC in 2014 was Googlenet, a.k.a Inception V1[6]

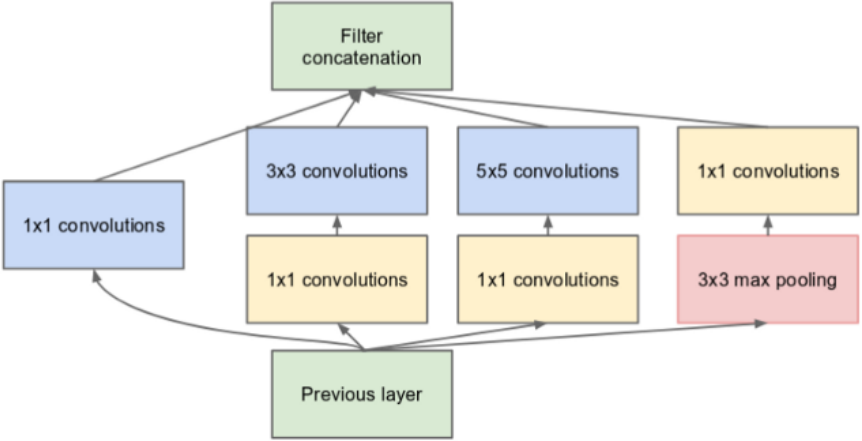


[6] Christian Szegedy et al. "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9

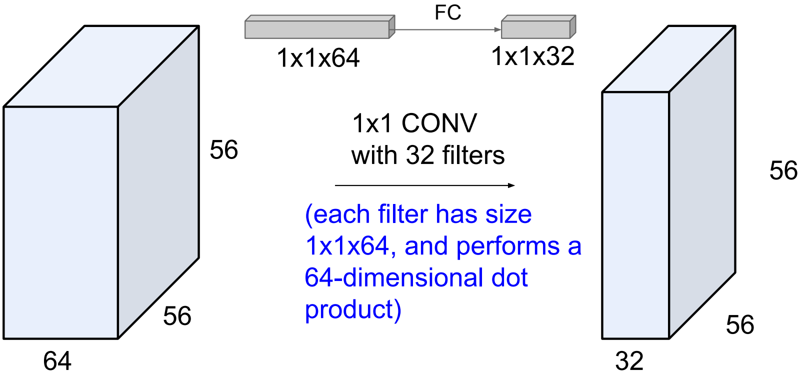
Inception blocks



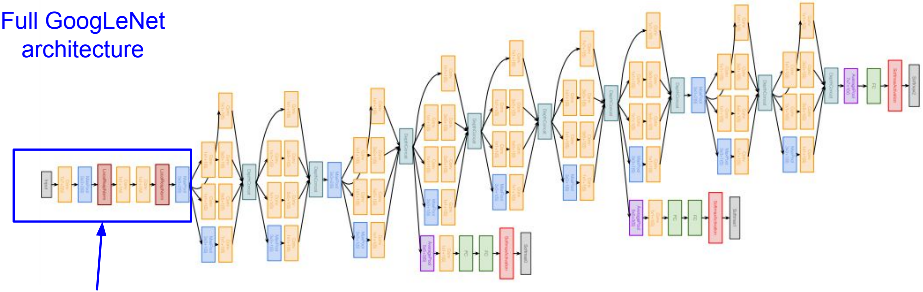
Inception blocks



1 × 1 convolution

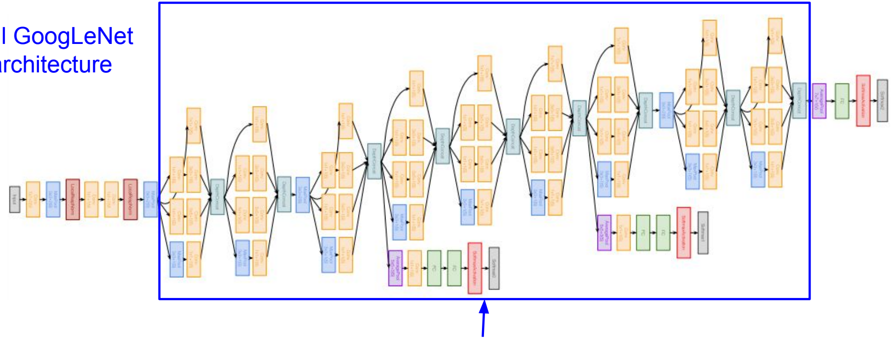


Full GoogLeNet architecture



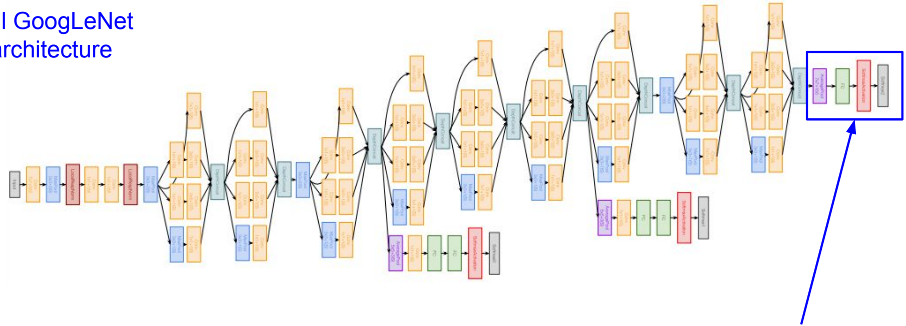
Stem Network:
Conv-Pool-
2x Conv-Pool

Full GoogLeNet architecture



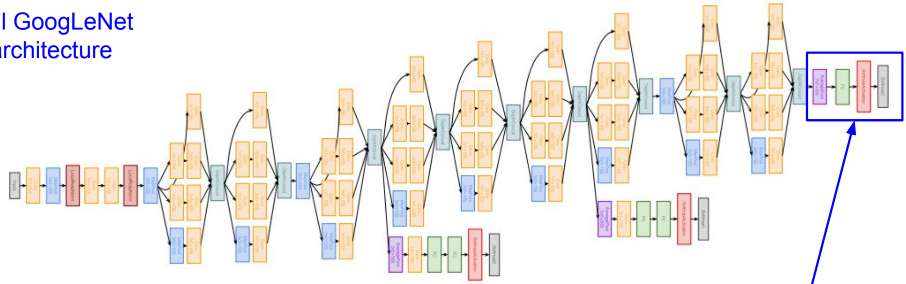
Stacked Inception Modules

Full GoogLeNet architecture



Classifier output

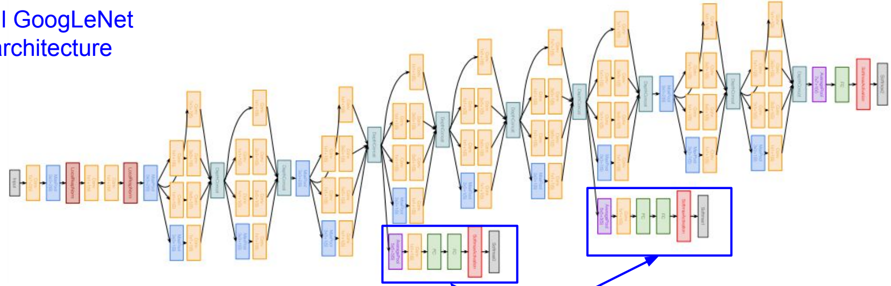
Full GoogLeNet
architecture



Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

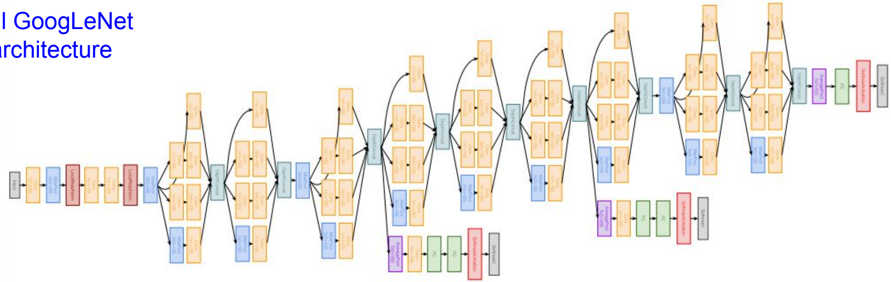
Classifier output

Full GoogLeNet architecture



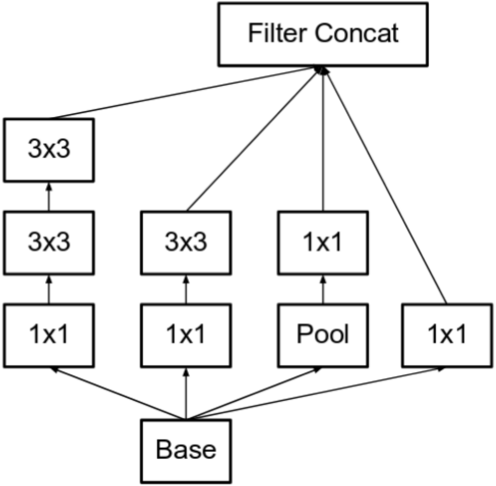
Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

Full GoogLeNet architecture

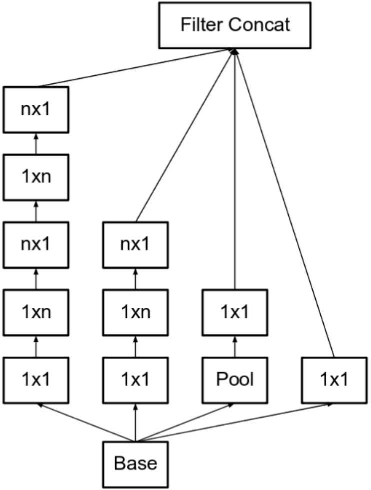


22 total layers with weights
(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

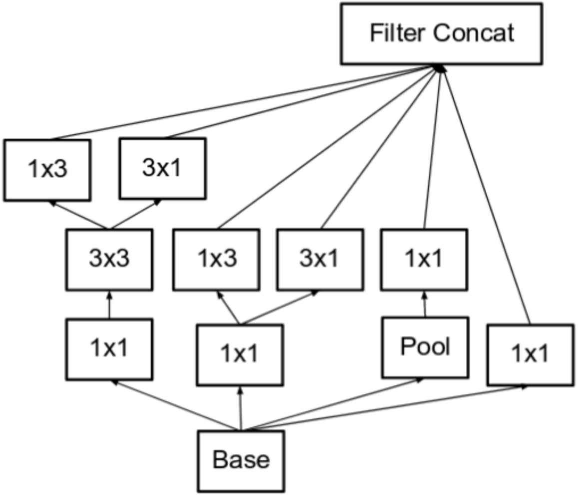
Inception v2 blocks - I



Inception v2 blocks - II



Inception v2 blocks - III



Vanishing/exploding gradients

The problem of vanishing/exploding gradients prevents training naive deep networks. So far we saw two solutions:

- VGG - train the early layers by themselves, add more later
- Inception - use auxillary training layers

Training deep networks

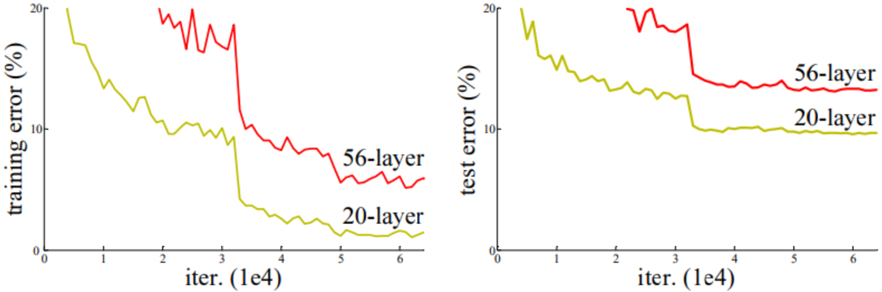
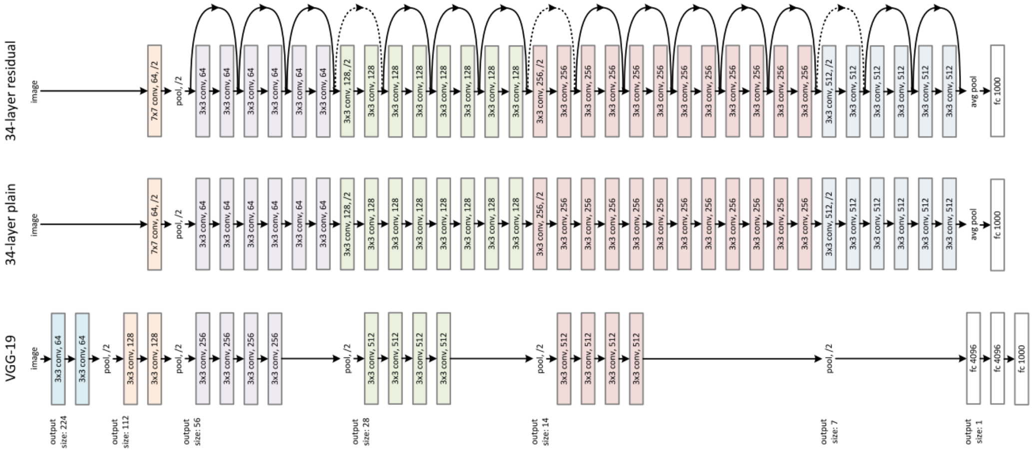
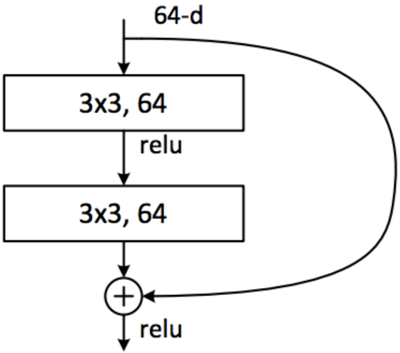


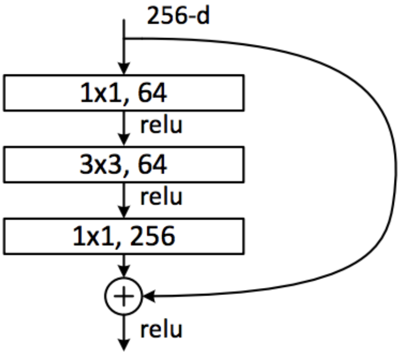
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Introducing residual/skip connections



Residual connection





Residual networks paper showed that a network of arbitrary depths can be trained! (With diminishing results)

[7] Kaiming He et al. “Deep residual learning for image recognition.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778

ResNets

Residual networks paper showed that a network of arbitrary depths can be trained! (With diminishing results)

Even 6 years later ResNets are still a good choice for a backbone architecture especially when prototyping!

[7] Kaiming He et al. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778

Residual networks paper showed that a network of arbitrary depths can be trained! (With diminishing results)

Even 6 years later ResNets are still a good choice for a backbone architecture especially when prototyping!

The original paper[7] is one of the most cited papers in CV.

[7] Kaiming He et al. “Deep residual learning for image recognition.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778

Resnet variants and successors

- Inception-ResNet[8] - Combination of Inception style blocks and residual connections
- ResNet v2[9] - Switch around the order of operations within a block
- Wide ResNets[10] - Having more channels is better than going deeper
- ResNext[11] - Multiple pathways, similar to Inception
- Squeeze-and-ExcitationNetwork[12] - Added recalibration layer - Won ILSVRC 2017

[8] Christian Szegedy et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." In: *Thirty-first AAAI conference on artificial intelligence*. 2017

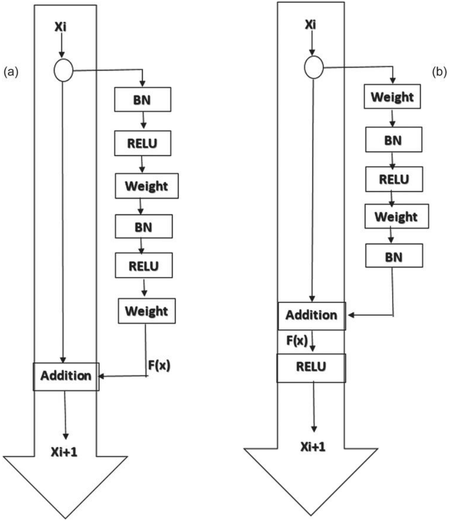
[9] Kaiming He et al. "Identity mappings in deep residual networks." In: *European conference on computer vision*. Springer. 2016, pp. 630–645

[10] Sergey Zagoruyko and Nikos Komodakis. "Wide residual networks." In: *arXiv preprint arXiv:1605.07146* (2016)

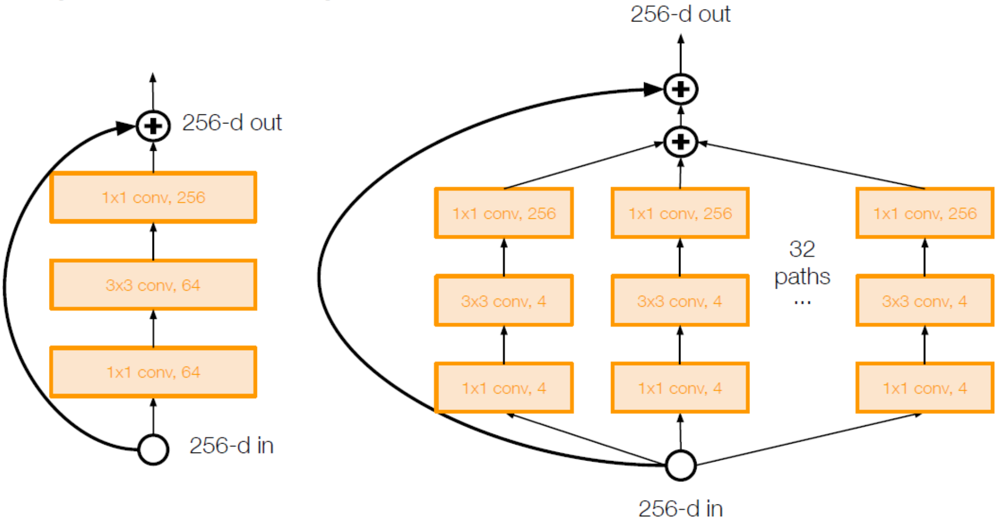
[11] Saining Xie et al. "Aggregated residual transformations for deep neural networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500

[12] Jie Hu, Li Shen, and Gang Sun. "Squeeze-and-excitation networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141

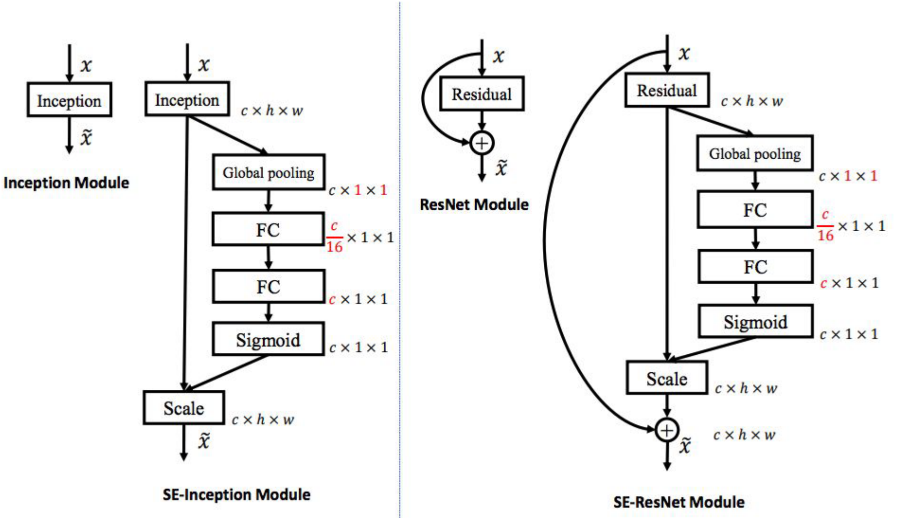
ResNet v2



(a)
Xi
BN
RELU
Weight
BN
RELU
Weight
F(x)
Addition
Xi+1
(b)
Xi
Weight
BN
RELU
Weight
BN
Addition
F(x)
RELU
Xi+1

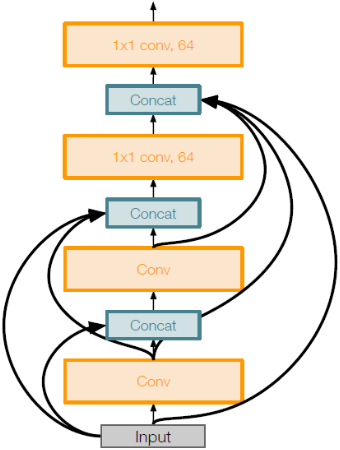

256-d out
1x1 conv, 256
3x3 conv, 64
1x1 conv, 64
256-d in
256-d out
1x1 conv, 256
1x1 conv, 256
1x1 conv, 256
32
paths
...
3x3 conv, 4
3x3 conv, 4
3x3 conv, 4
1x1 conv, 4
1x1 conv, 4
1x1 conv, 4
256-d in

SENet

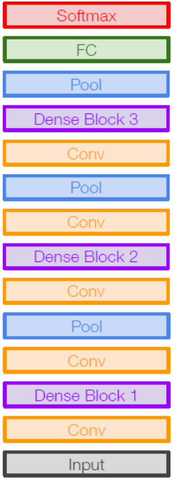


x
Inception
x̃
Inception Module
x
Inception
c × h × w
Global pooling
c × 1 × 1
FC
c/16 × 1 × 1
FC
c × 1 × 1
Sigmoid
c × 1 × 1
Scale
c × h × w
x̃
SE-Inception Module
x
Residual
x̃
ResNet Module
x
Residual
c × h × w
Global pooling
c × 1 × 1
FC
c/16 × 1 × 1
FC
c × 1 × 1
Sigmoid
c × 1 × 1
Scale
c × h × w
c × h × w
x̃
SE-ResNet Module

DenseNet



Dense Block



Neural Architecture Search

So far the networks were usually designed by hand. It is possible to treat this as an optimization problem in the space of architectures. This is however a difficult task a very influential paper[13] used some tricks to get it working:

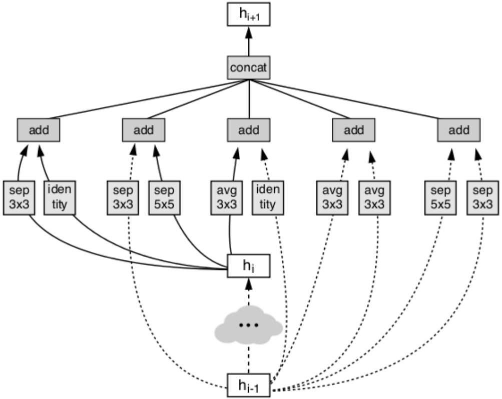
- Consider a network composed of two types of blocks and only optimize the block structure
- Optimize on smaller dataset and transfer to larger one later by stacking multiple blocks

Since then there have been some other approaches to NAS such as ENAS[14] where the optimization starts with a large computational graph and tries to find the optimal subgraph.

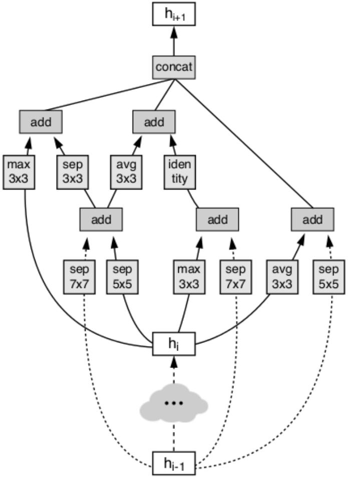
[13] Barret Zoph et al. "Learning transferable architectures for scalable image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710

[14] Hieu Pham et al. "Efficient neural architecture search via parameters sharing." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4095–4104

NASNet



Normal Cell



Reduction Cell

Efficient architectures

So far the architectures were mostly designed to optimize for accuracy. However it is possible to optimize for efficiency as well. There are multiple approaches.

- Depthwise Separable Convolutions - Introduced in MobileNet
  - Introduced in MobileNet[15]
  - Also used in Xception[16] - Inception-like architecture
- ShuffleNet[17] - Group pointwise convolutions with group shuffling
- NAS - MNASNet[18] optimized for accuracy + computational speed on mobile devices

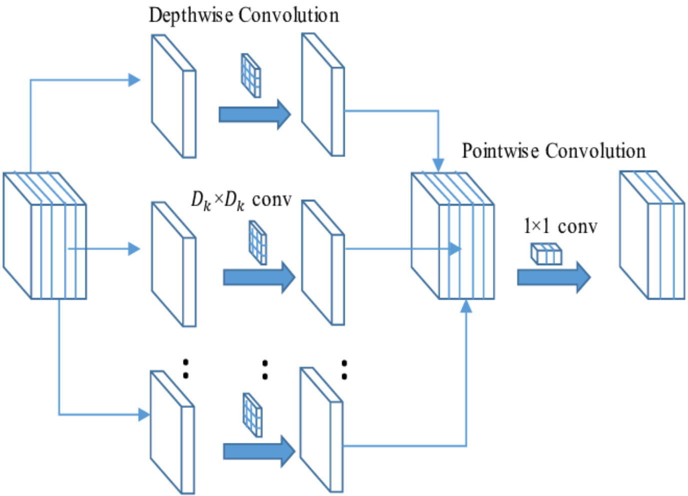
[15] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." In: *arXiv preprint arXiv:1704.04861* (2017)

[16] François Chollet. "Xception: Deep learning with depthwise separable convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258

[17] Xiangyu Zhang et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6848–6856

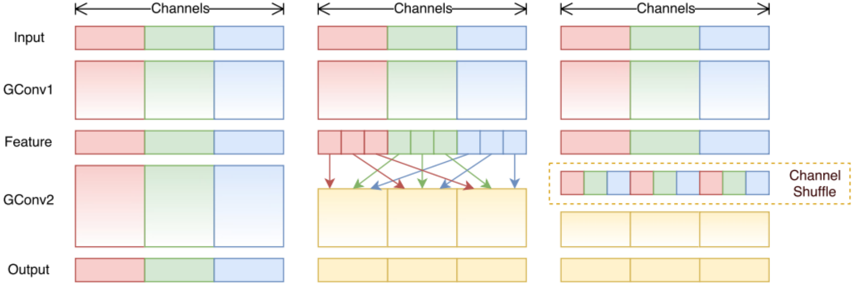
[18] Mingxing Tan et al. "Mnasnet: Platform-aware neural architecture search for mobile." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2820–2828

MobileNet



Depthwise Convolution
$D_k \times D_k$ conv
Pointwise Convolution
1×1 conv

ShuffleNet


Channels
Channels
Channels
Input
GConv1
Feature
GConv2
Output
Channel
Shuffle

EfficientNets

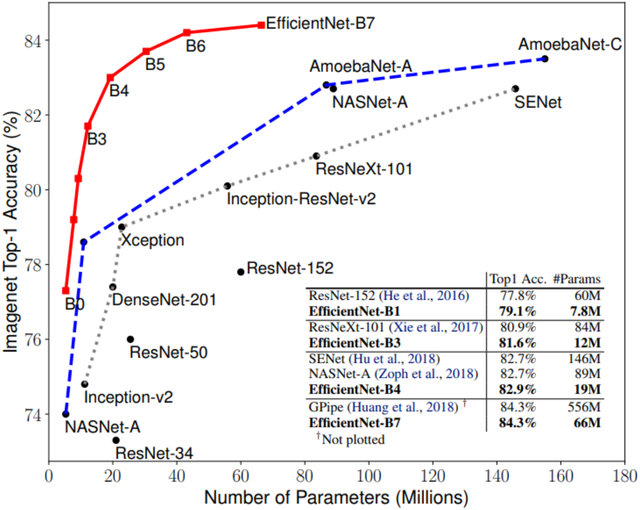
Most networks can be scaled roughly in three dimensions:

- Depth - number of layers
- Width - number of feature channels
- Input size - dimensions of input image

In the EfficientNet[19] paper authors propose a mechanism that scales the network along the three dimensions to ensure better performance. They also use NAS to find optimal architecture and propose several models EfficientNet-B0 to EfficientNet-B7.

[19] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6105–6114

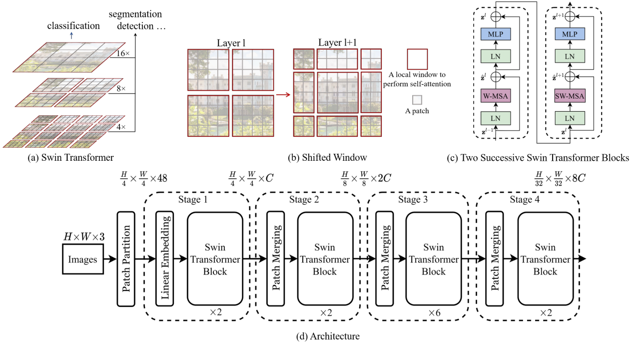
EfficientNets



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

† Not plotted

Transformers



(a) Swin Transformer (b) Shifted Window (c) Two Successive Swin Transformer Blocks

(d) Architecture

The latest trend is to abandon CNNs and instead use transformers even for CV Tasks! Still a developing field so using a CNN is still a safer bet but that might change very soon. More on this in the next lecture.

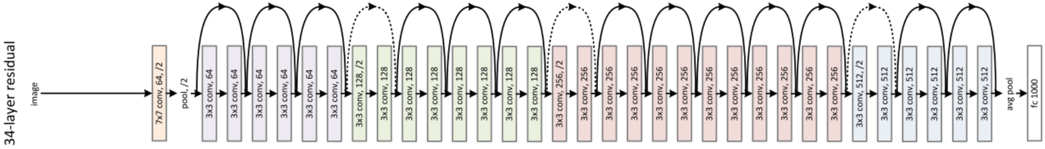
Transfer Learning

Training deep neural nets requires lots of annotated data! If our dataset is small we might have a problem.

Transfer Learning

Training deep neural nets requires lots of annotated data! If our dataset is small we might have a problem.

Solution: Pretrain the network on bigger dataset (of similar data) and fine-tune on our dataset.



If your own dataset is very small you can freeze the convolutional layers and keep train only the final FC layer.

Transfer Learning as Initialization

In practice we use pre-trained weights even if our dataset is large! This removes issues with parameter initialization. It often works even if data is from different domain!

Most common architectures can be initialized in both TF and Pytorch with a single line of code. Check the model ZOO's:

- `https://pytorch.org/vision/stable/models.html`
- `https://keras.io/api/applications/`

How to choose the best architecture

Some tips for selecting an architecture for your project:

- Do not create your own - use existing architectures
- If possible use models directly from frameworks model zoo
- If you need/want to make modifications - find code on GitHub
- ResNets are usually a good baseline - easy to compare to other approaches
- Other than that go for what achieves best ImageNet acc for your computational constraints
- Do not use VGG or AlexNet
- Be mindful of the image input size, network depth (next slide)

Selecting model

- For early prototypes use small models (ResNet18) with small input size
- If everything works try a larger model and input size
- Image input size should not be larger than receptive field of network
- You can sometimes increase/decrease receptive field by modifying some elements of the network (e.g. stem in ResNets)
- If you operate under some computational-constraints (e.g. realtime video processing on edge device) keep that in mind and go for smaller nets with smaller input sizes!

ResNet - receptive fields

layer	resnet18	resnet34	resnet50	resnet101
conv1	7	7	7	7
maxpool	11	11	11	11
layer1	43	59	35	35
layer2	99	179	91	91
layer3	211	547	267	811
layer4	435	899	427	971

Due to use of different blocks ResNet50 has smaller receptive field than ResNet18!