Course Notes: Deep Learning for Visual Computing

Peter Wonka

August 30, 2021

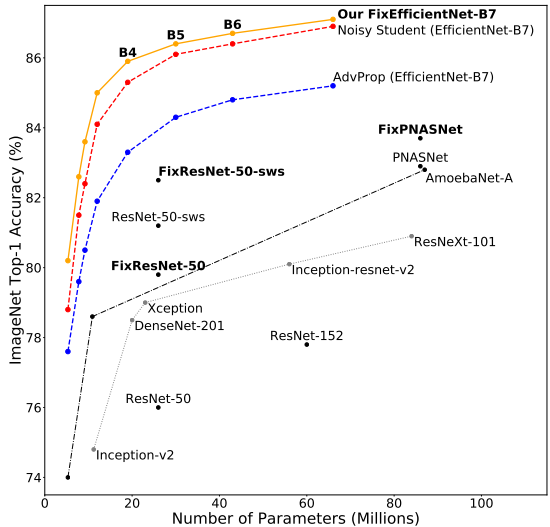
Contents

1 Classification Architectures

1.1 Metrics to summarize an architecture

- Depth: number of layers
- **Number** of learnable / trainable **network parameters**
- Memory consumption during inference / training
- Number of computations during inference / training
- Speed of inference on a particular hardware architecture
- **Task performance**:
  - Top-5 or Top-1 imagenet classification accuracy

1.2 Preview of Architectures



1.3 How to count layers?

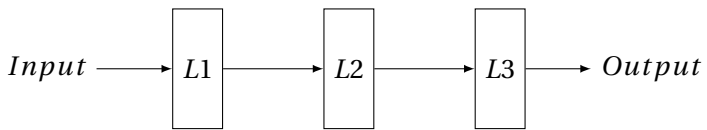
- Idea: only count layers with parameters (inconsitently applied)
- Fully connected and convolutional layers count
- pooling layers may or may not count (different conventions)
- Input layer does not count
- Non-linear activation layers do not count
- Normalization layers (e.g. batch-norm), Dropout, ... do not count
- Example:
  - VGG-16 has 16 layers + many other layers that do not count
  - ResNet-50 has 50 layers + many others that do not count

1.4 Printing network architecture in PyTorch

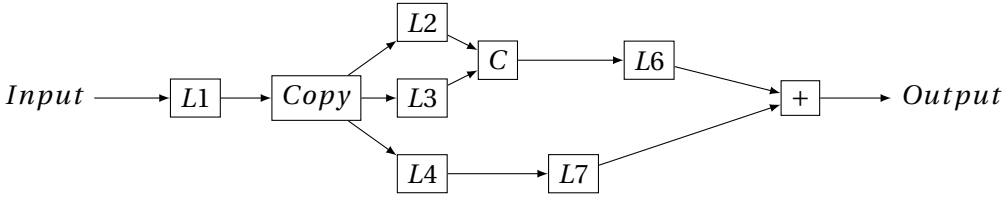
```
from torchvision import models
model = models.AlexNet()
print(model)
```

1.5 Building blocks for advanced computational graphs / networks

- Simple vs. Complex **Network Topology**
  - Simple networks have a chain topology (LeNet, AlexNet, VGG)



- Complex networks use additional components
  - Adding tensors
  - Tensor concatenation
  - Tensor Copying


Input
L1
Copy
L2
L3
L4
C
L6
L7
+
Output

1.6 Adding Tensors

- Input: tensors $\boldsymbol{X}$ and $\boldsymbol{Y} \in \mathbb{R}^{W \times H \times C}$
  - Tensors need to have the same dimensions
- Output: tensor $\boldsymbol{Z} \in \mathbb{R}^{W \times H \times C}$

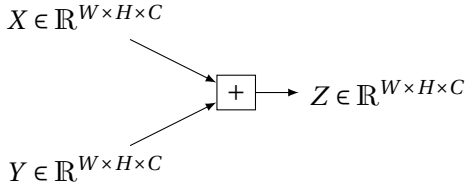
$$\boldsymbol{Z} = \boldsymbol{X} + \boldsymbol{Y} \tag{1.1}$$

- Addition of tensors is defined elementwise:

$$\boldsymbol{Z}(w, h, z) = \boldsymbol{X}(w, h, z) + \boldsymbol{Y}(w, h, z) \tag{1.2}$$

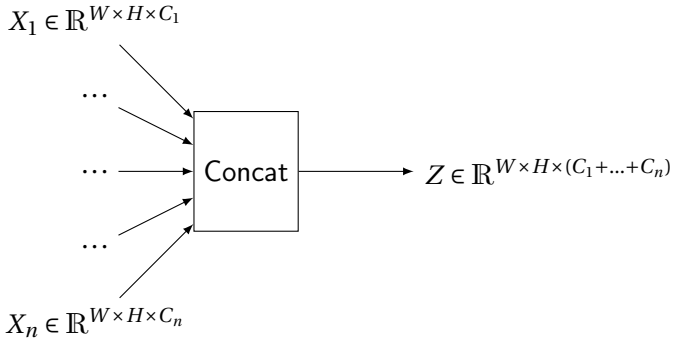
$$\tag{1.3}$$

  - $1 \leq w \leq W, 1 \leq h \leq H, 1 \leq c \leq C$
- Straightforward extension from vector and matrix addition


$X \in \mathbb{R}^{W \times H \times C}$
+
$Z \in \mathbb{R}^{W \times H \times C}$
$Y \in \mathbb{R}^{W \times H \times C}$

1.7 Concatenating Tensors

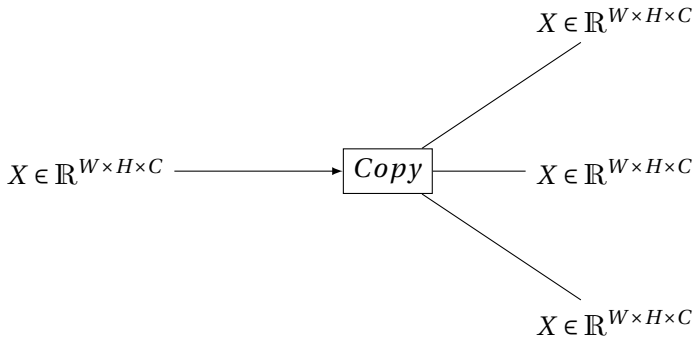
- Input: n tensors
  - tensor $\boldsymbol{X}_1 \in \mathbb{R}^{W \times H \times C_1}$
  - ...
  - tensor $\boldsymbol{X}_n \in \mathbb{R}^{W \times H \times C_n}$
- Output:
  - tensor $\boldsymbol{Y} \in \mathbb{R}^{W \times H \times (C_1 + ... + C_n)}$
- Channels are concatenated



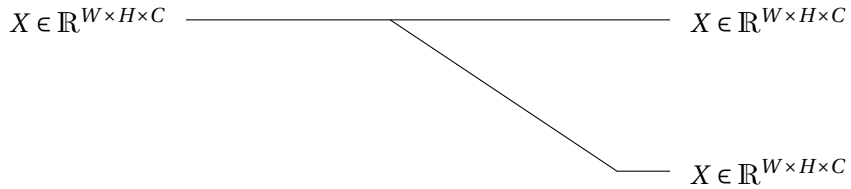
- Generally not drawn as separate operation
  - drawn as multiple inputs to a CONV or FC layer (e.g. DenseNet)

1.8 Copying Tensors

- Input: one tensor
- Output: multiple copies of the same tensor



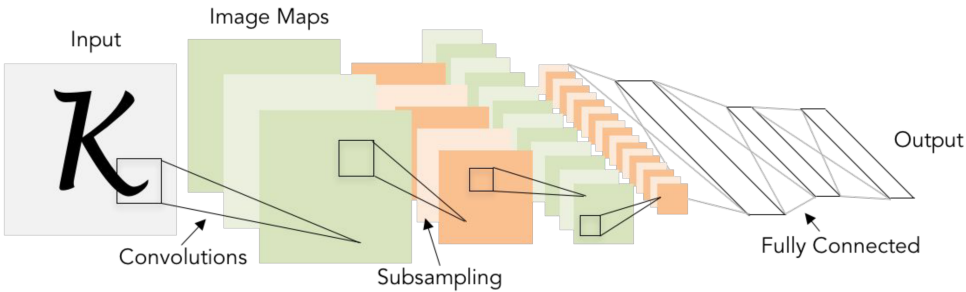
- Alternate drawing 1 (unlabeled branch)



- Alternate drawing 2 (just multiple arrows from CONV or FC layer)

1.9 LeNet-5

- Literature:
  - LeNet-5 webpage
  - LeCun et al., Gradient-based learning applied to document recognition
- **Historical importance**, early successful ConvNet



- Architecture Details:
  - Note: second CONV layer has filters that do not cover all channels to save memory / computation.
  - Non-linear activations after pooling layers

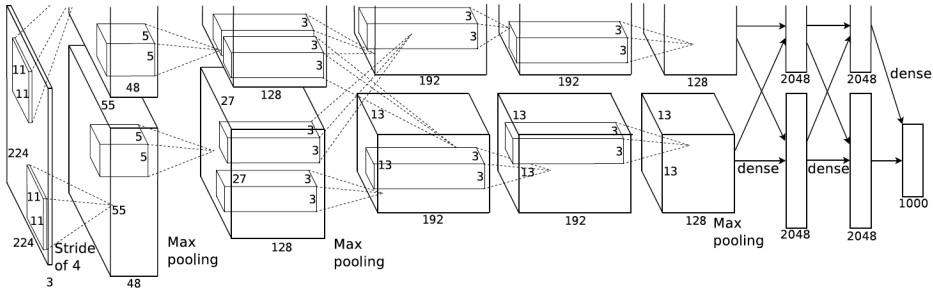
Type	#Filters	Spatial Res.	Specification	Activation	Output
Input		$32 \times 32 \times 1$	image (grayscale)		$32 \times 32 \times 1$
CONV	6	$5 \times 5 \times 1$	stride 1, padding 0	tanh	$28 \times 28 \times 6$
POOL		2×2	stride 2, average pooling	tanh	$14 \times 14 \times 6$
CONV	16	$5 \times 5 \times 6$	stride 1, padding 0	tanh	$10 \times 10 \times 16$
POOL		2×2	stride 2, average pooling	tanh	$5 \times 5 \times 16$
FC / CONV	120	$5 \times 5 \times 16$	can be seen as FC or CONV	tanh	120
FC	84			tanh	84
FC	10			Softmax	10

1.9.1 AlexNet

- Literature:
  - Krizhevsky et al. 2012, ImageNet Classification with Deep Convolutional Neural Networks
- **Significant Impact**:
  - Convinced many people about the strength of deep learning
- Paper ideas
  - (mainly make sensible engineering choices):
  - training on GPUs in parallel
  - use ReLU for faster training
  - use dropout

1.9.2 AlexNet Details

- use of Norm layers not common anymore
- dropout 0.5
- heavy data augmentation
- batch size 128
- SGD with momentum (0.9)
- Learning rate 0.01 reduced by 10 manually when val accuracy plateaus
- L2 weight decay $5e-4$
- 7 CNN ensemble: $18,2\% \rightarrow 15.4\%$
- Complex parallelization (not common anymore)




```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2,
        2))
    (1): ReLU(inplace)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
        ceil_mode=False)
```

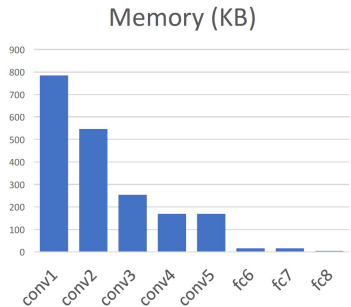
```
6       (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2,
            2))
7       (4): ReLU(inplace)
8       (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
            ceil_mode=False)
9       (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1,
            1))
10      (7): ReLU(inplace)
11      (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
            1))
12      (9): ReLU(inplace)
13      (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
             1))
14      (11): ReLU(inplace)
15      (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
            ceil_mode=False)
16    )
17    (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
18    (classifier): Sequential(
19      (0): Dropout(p=0.5)
```

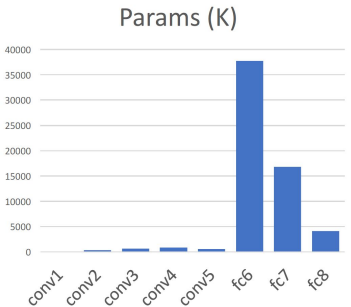
```
20      (1): Linear(in_features=9216, out_features=4096, bias=True)
21      (2): ReLU(inplace)
22      (3): Dropout(p=0.5)
23      (4): Linear(in_features=4096, out_features=4096, bias=True)
24      (5): ReLU(inplace)
25      (6): Linear(in_features=4096, out_features=1000, bias=True)
26    )
27  )
```

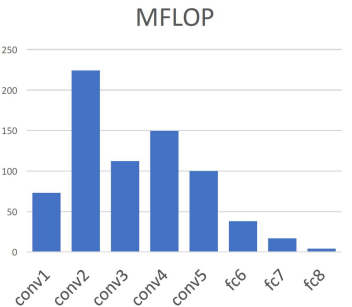
1.9.3 AlexNet Table

	Input Size		Layer				Output				
Layer	C	W / H	Filters	Kernel	Stride	Pad	C	H/W	Memory(KB)	Params(K)	Flops(M)
CONV1	3	227	64	11	4	2	64	56	784	23	73
POOL1	64	56		3	2	0	64	27	182	0	0.4
CONV2	64	27	192	5	1	2	192	27	547	307	224
POOL2	192	27		3	2	0	192	13	127	0	0
CONV3	192	13	384	3	1	1	384	13	254	664	112
CONV4	384	13	256	3	1	1	256	13	169	885	145
CONV5	256	13	256	3	1	1	256	13	169	590	100
POOL5	256	13		3	2	0	256	6	36	0	0
Flatten	256	6							36	0	0
FC6	9216		4096						16	37,749	38
FC7	4096		4096						16	16,777	17
FC8	4096		1000						4	4,096	4

1.9.4 AlexNet Statistics

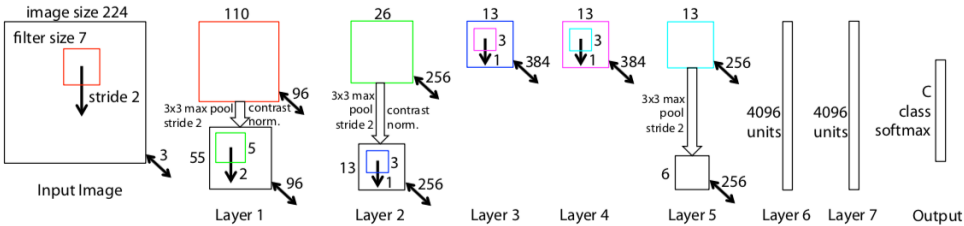






1.10 ZFNET

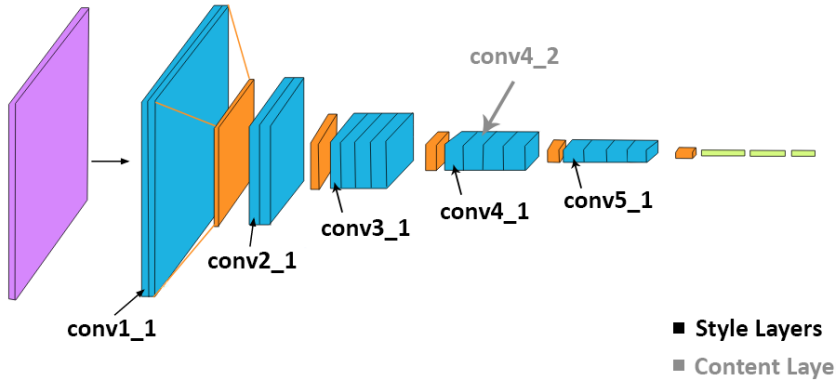
- Literature: Zeiler and Fergus, Visualizing and Understanding Convolutional Networks
  - Paper is more about the visualization
- Modified AlexNet to reduce top-5 error: $16.4\% \rightarrow 11.7\%$
  - Change from (11×11 stride 4) filters to (7×7 stride 2)
  - increase the number of filters / channels for CONV and FC layers by a factor of 2



1.11 VGG

- Literature: Karen Simonyan, Andrew Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition
  - not very deep by current standards
- Design principles:
  - Only use 3×3 convolutional layers
  - All max pool are 2×2 stride 2
  - After pooling, double the number of channels

1.11.1 Architecture Overview



- input: purple
- conv layers + relu: blue

- max pooling: orange
- fully connected layers: green
- final layer is softmax
- conv layers are organized into blocks that have the same resolution to assign names:
  - convX_Y means conv layer Y in block X
- conv layers use small filters (all filters are 3×3)
  - one configuration also has 1×1 filter
- Five max-pooling layers with a 2×2 pixel window, and stride 2.

1.11.2 Details in Numbers

ConvNet configurations (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

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**

Number of parameters (in millions).

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

1.11.3 Details

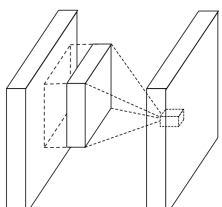
- Subtract mean RGB value from each pixel as pre-process
- mini-batch gradient descent with momentum as optimization algorithm
- batch size is 256
- Regularization: weight decay and dropout (for the first two fully connected layers)
- Learning rate schedule: initially set to 10^{-2}, decreased by a factor of 10 when the validation set accuracy stopped improving.
  - learning rate was decreased 3 times
- trained for 370K iterations / 74 epochs
- for testing they convert the fully connected layer to convolutional layers to test on larger images

1.11.4 Discussion

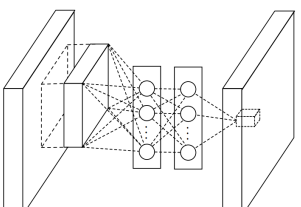
- Very popular as pre-trained encoder
- Very popular for feature extraction (texture synthesis, style transfer, ...)
- Fully connected layers at the end are probably overkill (lot of parameters, little improvement)

1.12 Network in Network

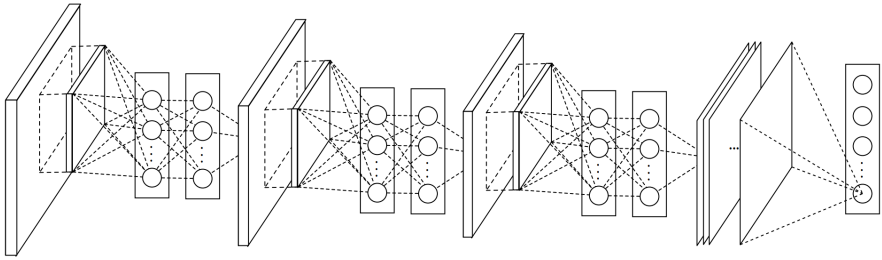
- Literature: Lin et al., Network In Network
- Idea: Use a mini network instead of a convolutional filter
  - conv layer + 2 local FC layers
  - basically equivalent to conv layer + two 1×1 CONV layers



(a) Linear convolution layer

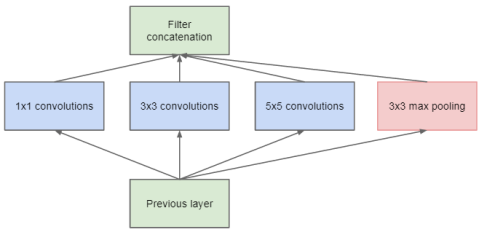


(b) Mlpconv layer

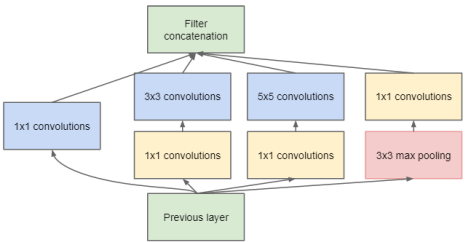


1.13 Inception V1.0

- Literature: Szegedy et al., Going Deeper with Convolutions
- Idea:
  - Build a good building block / module
  - Stack the module

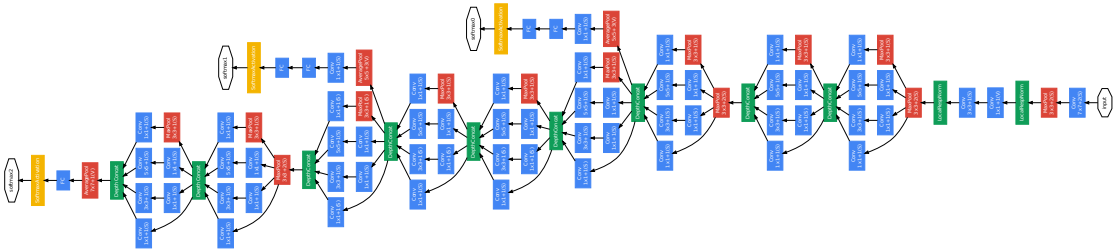


(a) Inception module, naïve version



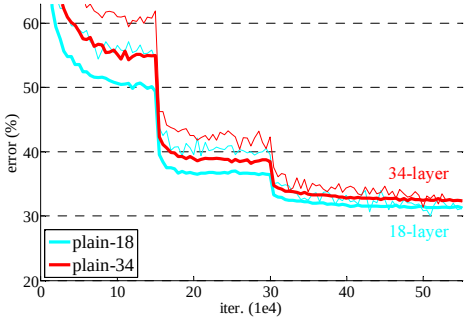
(b) Inception module with dimension reductions

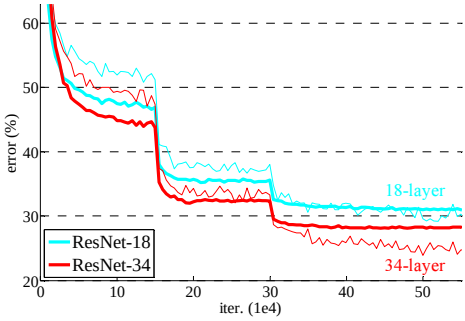
- GoogLeNet
  - One particular architecture using the inception module
  - Google-Net but capitalized to pay homage to LeNet
  - Combine inception blocks with pooling layers
  - Uses three classifiers in the middle to help training (two branches + one at the end)



1.14 ResNet

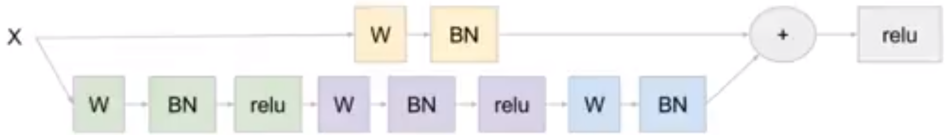
- Literature:
  - He et al., Deep Residual Learning for Image Recognition
  - youtube
- Idea:
  - Deeper networks should do at least as good as shallower networks
    - worst case extra layers should do nothing, i.e. learn the identity function
  - Deeper networks can have worse performance than shallower networks in practice
    - Maybe too difficult to learn the identity function (do nothing) and improve?
  - Use shortcuts to make learning identity function easier
    - Network layers that learn the difference to the identity function


error (%)
iter. (1e4)
plain-18
plain-34
34-layer
18-layer


error (%)
iter. (1e4)
ResNet-18
ResNet-34
18-layer
34-layer

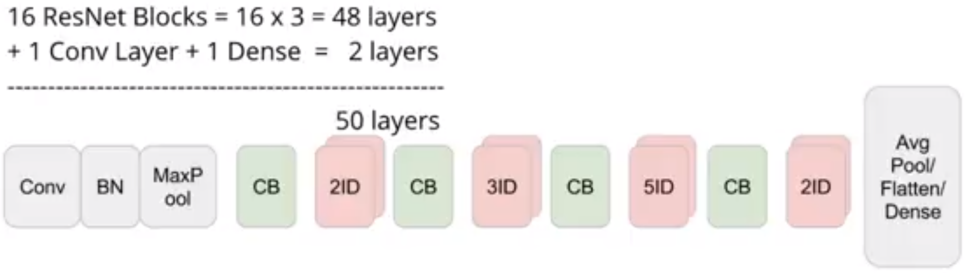
1.14.1 ResNet Blocks

- Term: Identity Block
- X is a tensor
- W is a CONV or FC layer
- BN is a batch normalization layer
- $+$ means tensors are added component wise
- Terms:
  - main path / main brach is the branch with most of the computation (bottom)
  - shortcut branch is the branch bypassing the computation (top)



- Term: CONV Block
- Problem: convolutions that change the output size make tensors on two branches incompatible
- Solution: requires additional computation on the shortcut branch to match tensor shapes

1.14.2 ResNet Architecture Example

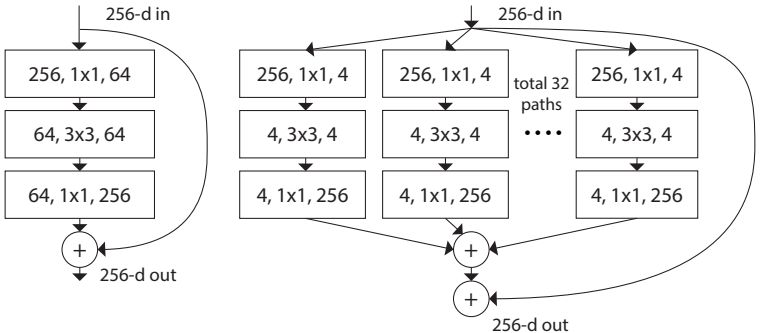


1.14.3 Details

- ResNet has many different versions
  - ResNet-18, ResNet-50, ResNet-152, ...
  - ResNets with > 1000 layers have been built
- Winner of 2015 ImageNet challenge
- Batchnormalization after each convolution, before nonlinear activation function
- Batch size of 256
- Learning rate starts at 0.1 and is divided by 10 when the error plateaus
- Does **not use dropout**
- Test-time augmentation: 10-crop testing

1.15 ResNeXt

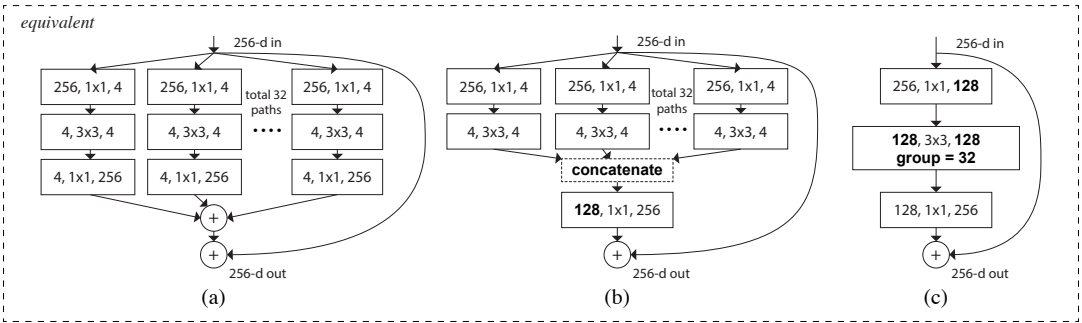
- Literature: Xie et al.,Aggregated Residual Transformations for Deep Neural Networks



- **Left**: A block of ResNet
- **Right**: A block of ResNeXt with cardinality = 32

- roughly the same complexity.

- A layer is shown as (# in channels, filter size, # out channels).



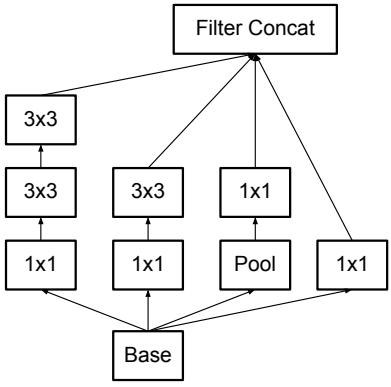
- Multiple equivalent ways to view a ResNext Block

1.16 Inception V2.0 and v3.0

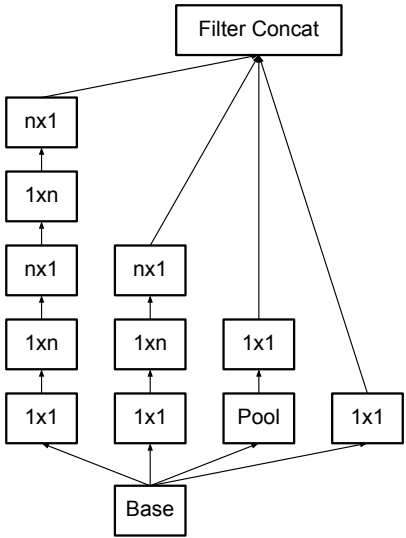
- Literature:
  - Szegedy et al., Rethinking the Inception Architecture for Computer Vision
  - Inception v2.0 and v3.0 introduced in the same paper
  - They are similar and we introduce the ideas together

1.16.1 New Inception blocks

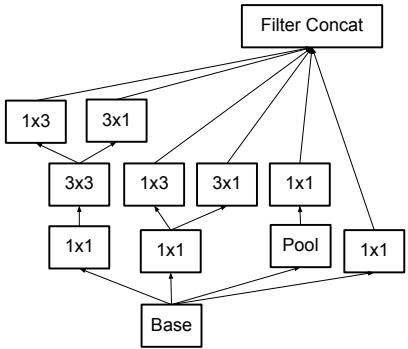
- Ideas for new blocks
- Factor 5×5 convolutions into two 3×3 convolutions
  - Uses fewer parameters
  - Should there be a non-linear layer between the two 3×3 convolutions?
    - Experiments suggest yes


Filter Concat
3x3
3x3
3x3
1x1
1x1
1x1
Pool
1x1
Base

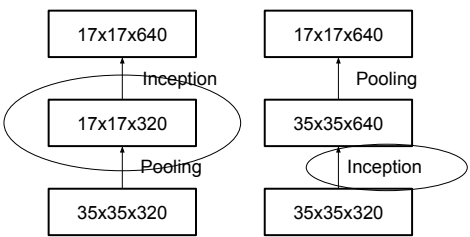
- Factor $k \times k$ convolutions into a $k \times 1$ and a $1 \times k$ convolution (cheaper)

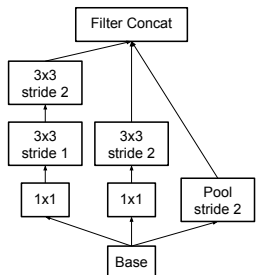


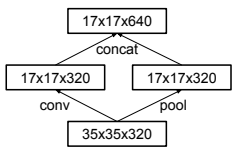
- The filter banks in the module were expanded (made wider instead of deeper)



- Downsampling block
  - concern about the amount of information flowing in the network.
  - downsampling via pooling leads to the loss of a lot of information (data in the tensors)
  - cannot simply be recovered by increasing the data later







1.16.2 Inception Architecture Details

- RMSProp Optimizer
- Factorized 5×5 and 7×7 convolutions
- BatchNorm in the Auxillary Classifiers
- Label Smoothing
- Table Notes:
  - Convolution with 0-padding is marked
  - 0-padding is also used inside Inception modules that do not reduce the spatial resolution

type	**patch size/stride** or remarks	**input size**
conv	$3\times3/2$	$299\times299\times3$
conv	$3\times3/1$	$149\times149\times32$
conv padded	$3\times3/1$	$147\times147\times32$
pool	$3\times3/2$	$147\times147\times64$
conv	$3\times3/1$	$73\times73\times64$
conv	$3\times3/2$	$71\times71\times80$
conv	$3\times3/1$	$35\times35\times192$
3×Inception	Block I	$35\times35\times288$
5×Inception	Block 2	$17\times17\times768$
2×Inception	Block 3	$8\times8\times1280$
pool	8×8	$8\times8\times2048$
linear	logits	$1\times1\times2048$
softmax	classifier	$1\times1\times1000$

1.17 Inception V4.0 and Inception ResNet

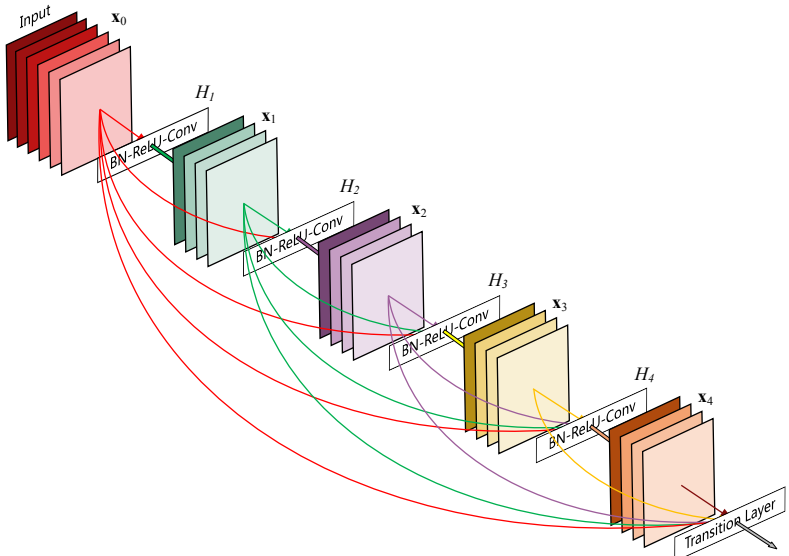
- Literature: Szegedy et al., Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning
- Inception v4.0 and Inception ResNet are introduced in the same paper
- See the paper for architecture figures

1.18 DenseNet

- Literature:
  - Huang et al., Densely Connected Convolutional Networks
  - CVPR talk
- Idea:
  - Simple networks use the output of one layer as the input of the next layer. (L layers have L connections)
  - DenseNet: Each layer takes all previous layer's outputs as input (L layers have $L(L+1)/2$ connections)
- Advantages (claimed in the paper):
  - reduce the vanishing-gradient problem
  - strengthen feature propagation
  - encourage feature reuse
  - reduce the number of parameters

- Difference to ResNet:
  - Tensors are combined by concatenation not elementwise addition

1.18.1 DenseNet Block



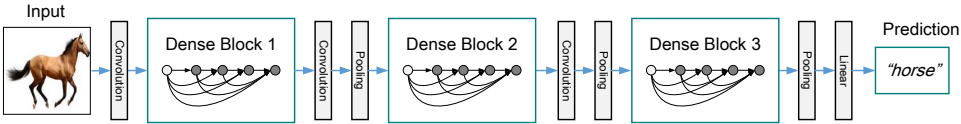
1.18.2 DenseNet Details

- Growth Rate k:
  - Growth rate k is the number of channels output by a convolutional later:
  - If a dense block has a tensor with k_0 channels as input, the l^{th} layer has a $k_0 + (l-1)k$ input channels
  - DenseNet uses narrow layers, e.g. $k = 12$
- Convolutional Layer:
  - batch normalization
  - ReLU
  - 3×3 convolution
- Transition Layers:
  - Transition layer reduces the spatial resolution of tensors
  - batch normalization layer
  - 1×1 convolutional layer

- 2×2 average pooling layer

- Bottleneck Layers
  - Use BN-ReLU-Conv(1×1)-BN-ReLU-Conv(3×3) instead of BN-ReLU-Conv(3×3)

1.18.3 DenseNet Architecture Example



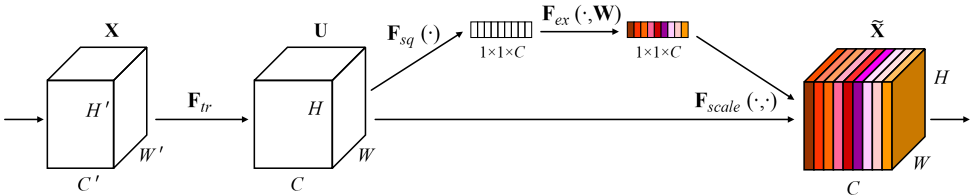
1.19 NASNet

- Idea:
  - Use genetic algorithms, stochatic search, reinforcement learning to find a suitable architecture
  - Will be discussed in separate chapter

1.20 SENet

- Literature: Hu et al., Squeeze-and-Excitation Networks
- Idea:
  - Add a block that computes a per channel scaling
  - All values in one channel are multiplied with the same scalar
  - Requires computation of C scalars to scale the channels
  - Can be done after an operation, such as convolution

1.21 SENet Buildung Block



- F_{tr} is an arbitrary transformation in the network, e.g. a CONV layer
- Squeeze and Excite block is a post-process consisting of three steps
- F_{sq} computes the average value per channel (one scalar per channel)
  - output $1 \times 1 \times C$ tensor (vector) $\mathbf{z}$

$$\mathbf{z}_c = F_{sq}(\boldsymbol{U}, c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \boldsymbol{U}(c, i, j)$$

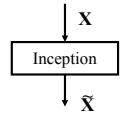
- F_{ex} computes two 1×1 convolutions (= fully connected layer)

- input: $1 \times 1 \times C$ tensor z
- first output $1 \times 1 \times C/r$ tensor
- second output: $1 \times 1 \times C$ tensor s

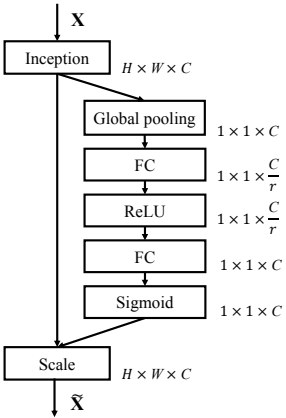
$$\mathbf{s} = F_{ex}(\mathbf{z}, \boldsymbol{W}) = sigmoid(\boldsymbol{W}_2 ReLU(\boldsymbol{W}_1 \mathbf{z}))$$

- F_{scale} scales each channel of tensor $\boldsymbol{U}$ by a scalar

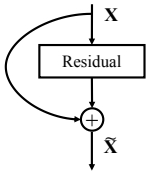
1.22 SENet Architecture



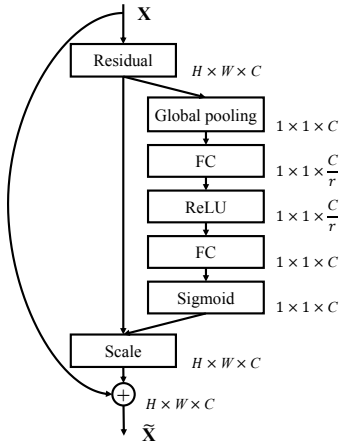
Inception Module



SE-Inception Module

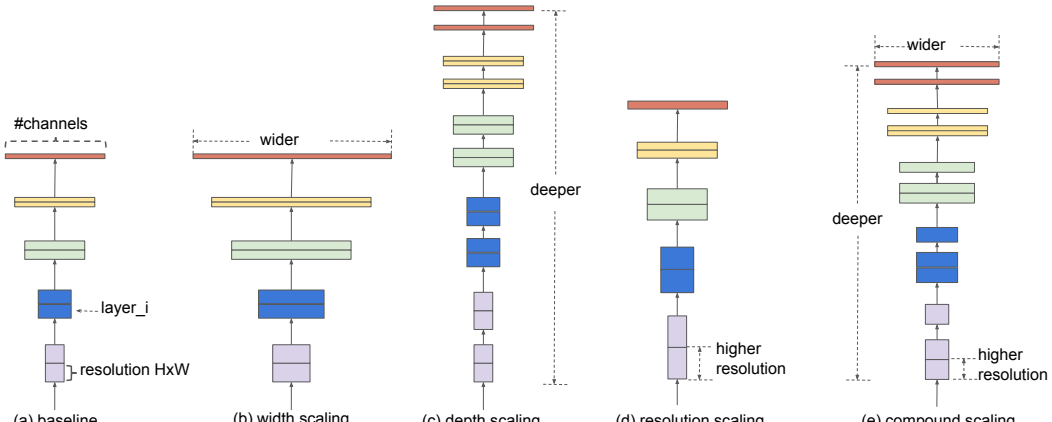


ResNet Module



SE-ResNet Module

1.23 EfficientNet



- Literature: Tan and Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

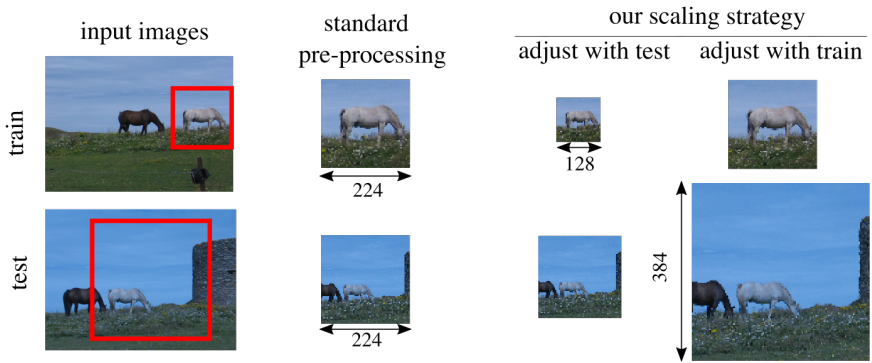
- Idea: develop an efficient smaller network and **scale it up**
- Three methods to scale up a network
  - **Depth**: number of layers
  - **Width**: number of channels
  - **Resolution**: input resolution, how fast to decrease resolution

- A principled method to scale up ConvNets
  - Assume you want to scale up the network by 2^N
  - Determine α, β, γ such that
    - Depth increases by α^N
    - Width increases by β^N
    - Resolution increases by γ^N
- α, β, γ are determined by a small grid search from a baseline model
- Search for baseline architecture **EfficientNet-B0**
- Scale up baseline architecture using α, β, γ (paper: $\alpha = 1.2, \beta = 1.15, \gamma = 1.1$
- Paper proposes a constraint $\alpha \times \beta^2 \times \gamma^2 \approx 2$
  - FLOPs scale linearly with **depth**, but quadratically with **width** and **resolution**
- Scale up from **EfficientNet-B1** to **EfficientNet-B7**

- EfficientNet-B0 baseline network: Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation
  - Uses **mobile inverted bottleneck MBConv** and **squeeze and excite** in blocks

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

1.24 Fixing the Train-Test Resolution Discrepancy

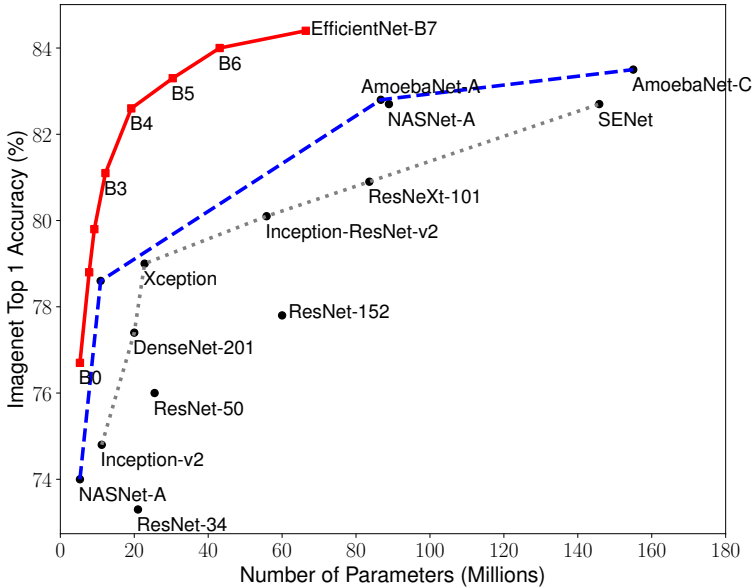


- Literature:
  - Touvron et al., Fixing the train-test resolution discrepancy
  - Touvron et al., Fixing the train-test resolution discrepancy: FixEfficientNet

- Problem: Data augmentation causes a training vs. testing **resolution mismatch**
  - Solution: Resizing
- Problem: resizing causes a change in **activation statistics**
  - Solution: fine-tune last layers using different resolution

1.25 Architecture Comparison

- Different Comparisons
  - Number of parameters vs. classification performance
- Websites that list models:
  - CV leaderboard
  - SOTA Benchmarks
  - Papers with Code


EfficientNet-B7
B6
B5
B4
B3
B0
AmoebaNet-A
AmoebaNet-C
NASNet-A
SENet
ResNeXt-101
Inception-ResNet-v2
Xception
ResNet-152
DenseNet-201
ResNet-50
Inception-v2
NASNet-A
ResNet-34
Imagenet Top 1 Accuracy (%)
Number of Parameters (Millions)
84
82
80
78
76
74
0
20
40
60
80
100
120
140
160
180

1.26 Discussion

- Architecture is **very very important**
- Challenging for research
  - State-of-the-art architectures can be very large
  - Constantly switching to state-of-the-art architecture is difficult
  - When proposing changes to a small architecture (e.g. ResNet-50, VGG-16, ...), you never know if the change is really relevant
- Architecture ideas are relevant for all sub-problems
  - Architecture is typically separately developed for each problem (segmentation, depth estimation, object recognition, pose estimation, ...)
- **Resnet-50 Research**
  - Many ideas are only developed on a smaller network using a smaller dataset
  - You never know if improvements help for bigger networks with bigger datasets