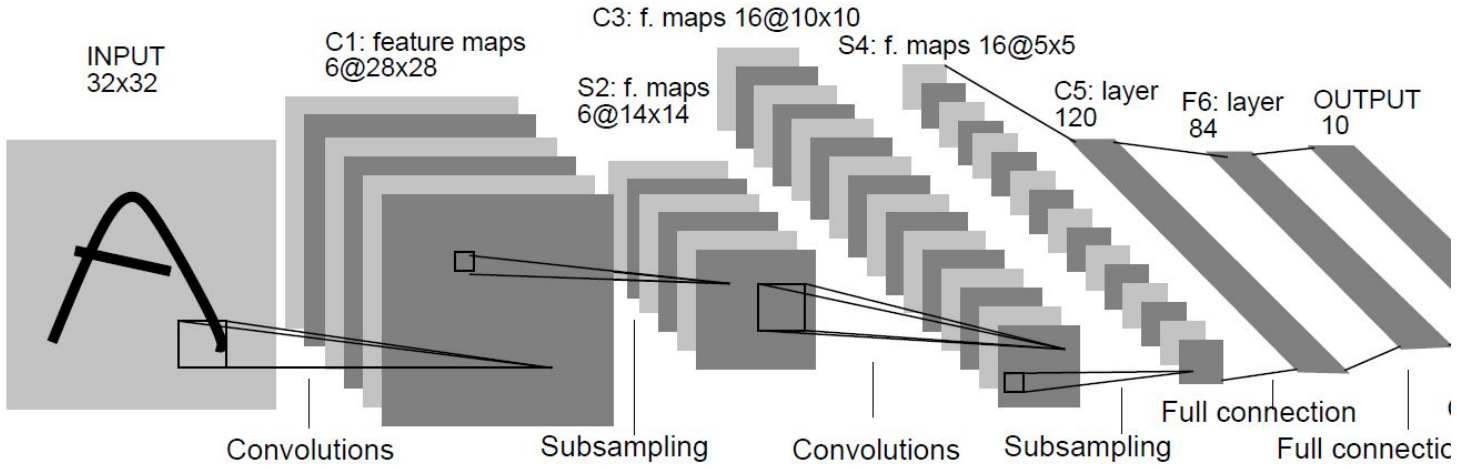
Evolution of Convolutional Neural Networks

Michael Klachko

Strukov's Research Group

UCSB

Lenet-5 (1998)



MNIST: handwritten digits

- 70,000 28x28 pixel images
- Gray scale
- 10 classes

CIFAR-10: simple objects

- 60,000 32x32 pixel images
- RGB
- 10 classes

1989 (Lecun) A convnet is used for an image classification task (zip codes)

- First time backprop is used to automatically learn visual features
- Two convolutional layers, two fully connected layer (16x16 input, 12 FMs each layer, 5x5 filters)
- Stride=2 is used to reduce image dimensions
- Scaled Tanh activation function
- Uniform random weight initialization

1998 (Lecun) LeNet-5 convnet achieves state of the art result on MNIST

- Two convolutional layers, three fully connected layers (32x32 input, 6 and 12 FMs, 5x5 filters)
- Average pooling to reduce image dimensions
- Sparse connectivity between feature maps

LeCun et al, Gradient-Based Learning Applied to Document Recognition

ImageNet Dataset (2010)

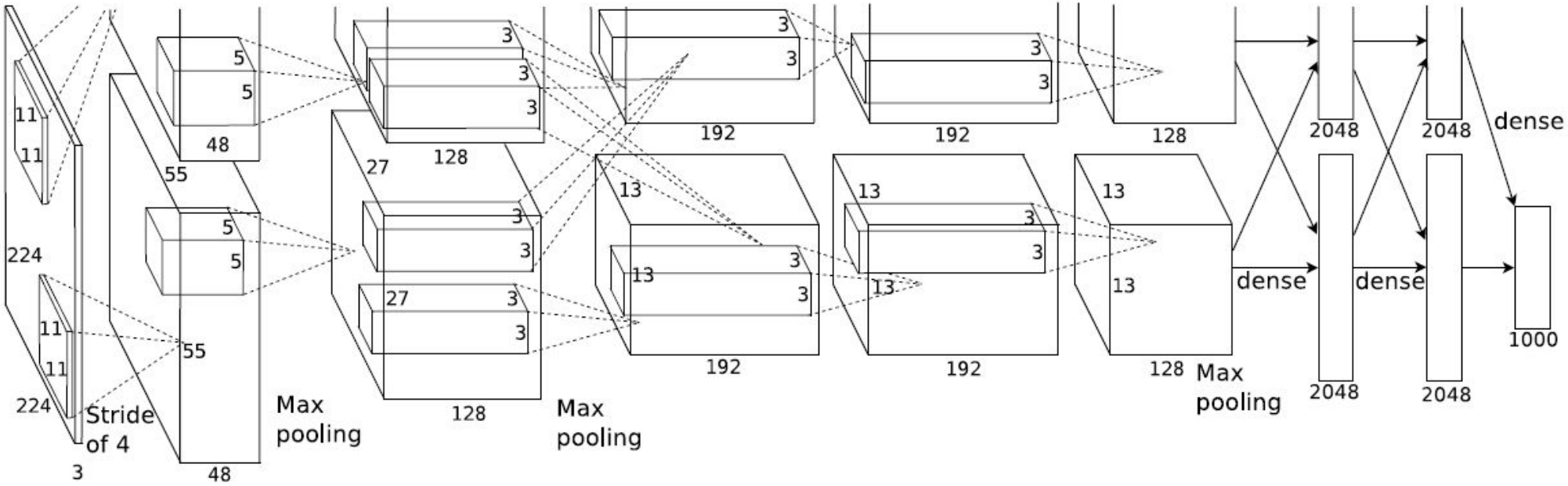
- 10M hand labelled images
- Variable resolution (between 512 and 256 pixels)
- 22k categories (based on WordNet synsets)
- ILSVRC: 1k categories, 1M training images
- 100k images for testing, 50k validation set
- State of the art results: 97%/85% (Top-5/Top-1)
- Human: 95% (Top-5, one week training)
- Typically, for training, input images are resized input to 256 pixels (shorter side), and multiple random crops of 224x224 are used together with their horizontal reflections
- For testing, multiple 224x224 crops are evaluated (anywhere from single to dense cropping)
- Multiscale training/evaluation has been tried as well

(a) Siberian husky

(b) Eskimo dog

Russakovsky et al, ImageNet Large Scale Visual Recognition Challenge

AlexNet (2012)



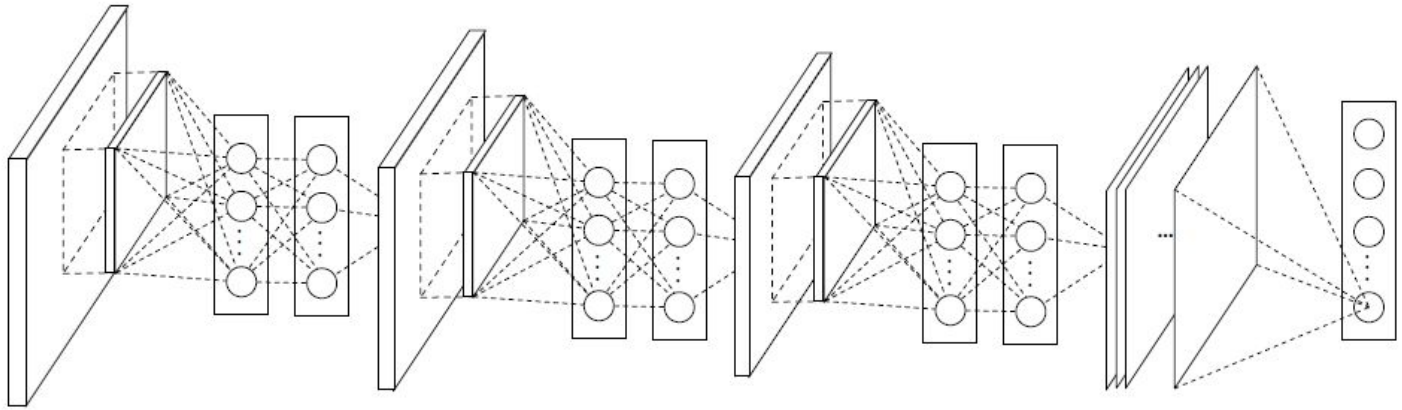
- ReLU
- Dropout
- Overlapping Max Pooling
- No pre-training

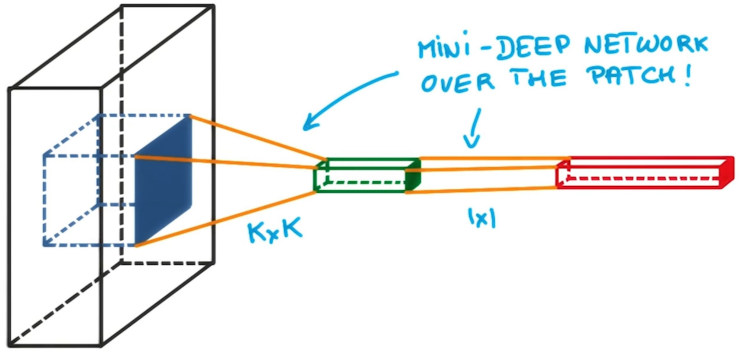
- 8 layers, 60M parameters
- 90% of weights is in FC layers
- 90% of computation is in convolutional layers

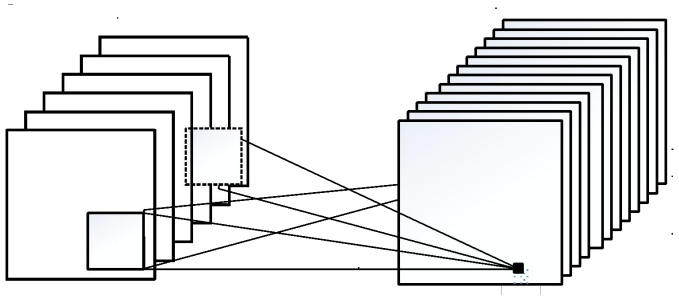
Krizhevsky et al, ImageNet Classification with Deep Convolutional Neural Networks

Network in Network (2014)

- Insert MLP between conv layers:
  - Extra non-linearity (ReLU)
  - Better combination of feature maps
  - Can be thought of as 1x1 convolution layer
- Global Average Pooling:
  - Last conv layer has as many feature maps as classes
  - Average activations in each feature map to produce final outputs
  - Easy to interpret visually
  - Less overfitting







Lin et al, Network In Network

VGG (2014)

- Increase depth and width
- Use only 3x3 filters
- 16 layers and lots of parameters (150M)
- Hard to train

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

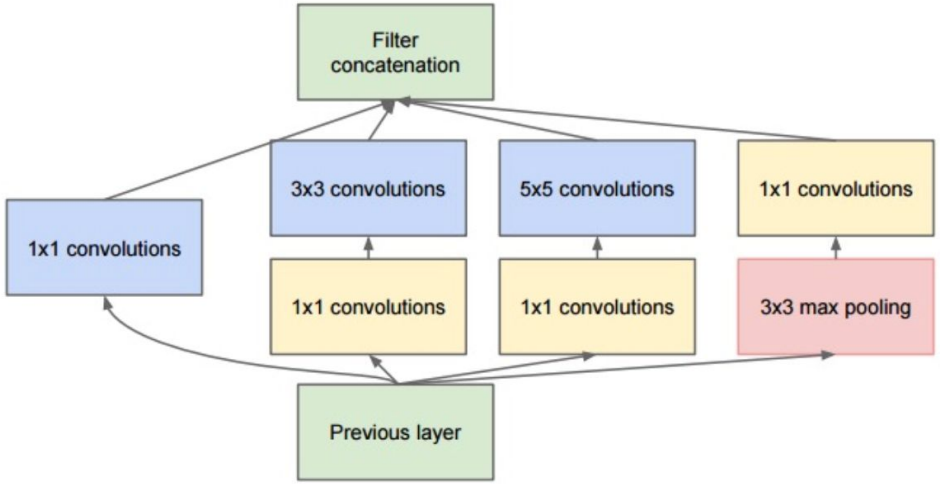
Simonyan et al, Very Deep Convolutional Networks for Large-Scale Image Recognition

GoogLeNet (Inception v1, 2014)

- How to reduce amount of computation?
  - Move from fully connected to sparse connectivity between layers
  - Bottleneck Layers: 256x256 x 3x3 = 589,000s MAC ops
    - 256×64 × 1×1 = 16,000s
      64×64 × 3×3 = 36,000s
      64×256 × 1×1 = 16,000s

      600k □ 70k MACs
- 22 layers, 5M weights, better accuracy than VGG with 150M weights
- Auxiliary classifiers to help propagate gradients

"Inception" module:



Szegedy et al, Going Deeper with Convolutions

Batch Normalization (Inception v2)

Problem: “Internal Covariate Shift”

- Updating weights changes distribution of outputs at each layer: when we change first layer weights, inputs distribution to the second layer changes, and now its weights have to compensate for that, in addition to their own update.
- Training would be more efficient if, for each layer, inputs distribution does not change from one minibatch to the next, and from training data to test data
- Changes to parameters cause many of input vector components to grow outside of efficient learning region (saturation for sigmoids, or negative region for ReLU), and slow down learning

Solution:

- Normalize each input component independently, so that it has mean 0 and variance 1 (using the same dimension across all training images)
- Simple normalization might change what the layer can represent. Therefore, we must insure it can be adjusted (and even reverted) as needed during training: use two learned parameters to perform a linear transformation after normalization
- Use minibatch instead of the entire training set
- for inference (testing): use entire training set mean and variance, or compute moving averages during training

Batch-normalized GoogLeNet:

- Less sensitive to weight initialization
- Can use large learning rate
- Better regularization: a training example representation depends on other examples in its minibatch: this jitters its place in the representation space of a layer (and reduces need for dropout and L2)
- Reaches the same accuracy as Googlenet 14 times faster!

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

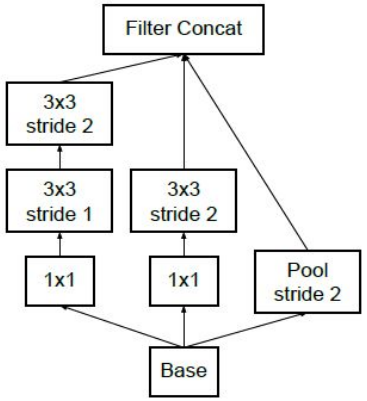
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

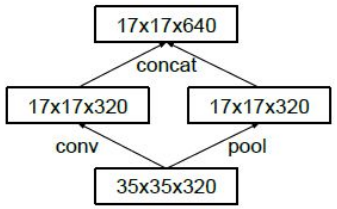
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Ioffe et al, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

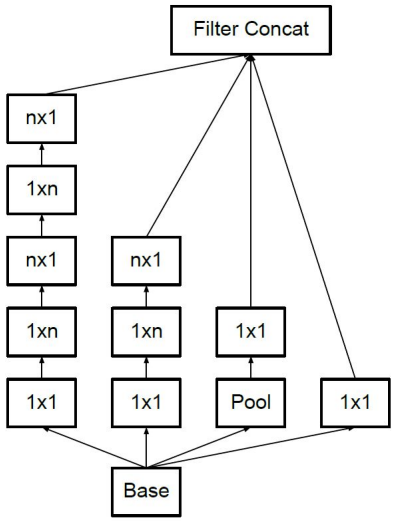
Inception v3 (2015)

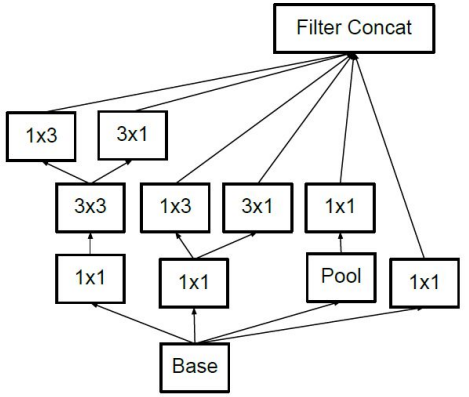
- Efficient ways to scale up GoogLeNet
  - Gradually reduce dimensionality, but increase number of feature maps towards the output layer
  - Balance width and depth
  - 42 layers, 25M params
- Label Smoothing: prevent the largest output to be much larger than other outputs. Replace the correct label with a random one with probability 0.1
  - Too confident prediction lead to poor generalization
  - Large difference between largest and second largest result in poor adaptability
- Reduce dimensionality by using stride 2 convolutions instead of max pooling between layers:

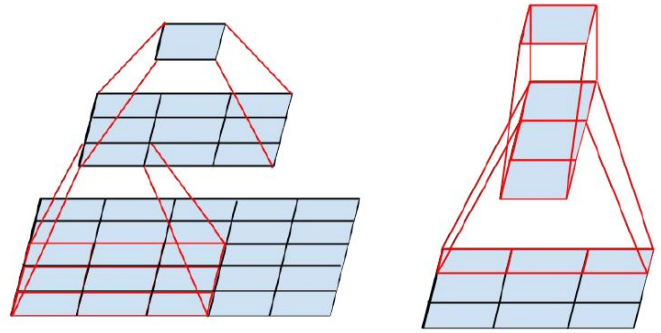




- Smaller convolutions: replace 5x5 filters with two level 3x3 convolutions
- Both number of weights and amount of computation is reduced by 28% (9+9)/25
- No loss of expressiveness, in fact better accuracy (possibly due to extra non-linearity)
- Asymmetrical convolutions: replace nxn convolutions with two level nx1 and 1xn convolutions (33% reduction for n=3)
  Good results achieved for n=7 applied to medium size feature maps (12x12 to 20x20)







type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Szegedy et al. Rethinking the Inception Architecture for Computer Vision

ResNet (2015)

- Add more layers, but allow bypassing them:
  - The network can learn whether to bypass or not
- Simple, uniform architecture, no extra parameters or computation Top-5: 3.57%
- Skip 2 layers, or 3 layers (1x1, 3x3, 1x1 blocks) for deeper networks
- Degradation problem for plain deep networks

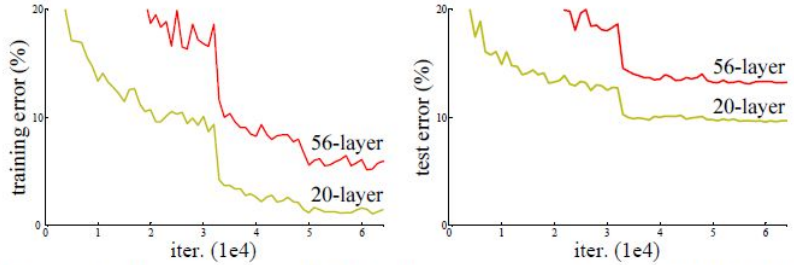
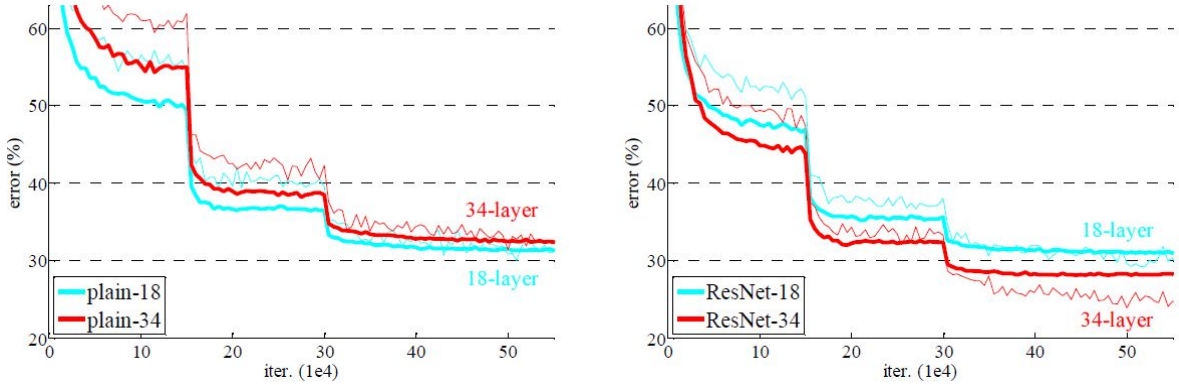


Figure 1. Training error (left) and test error (right) on CIFAR-10

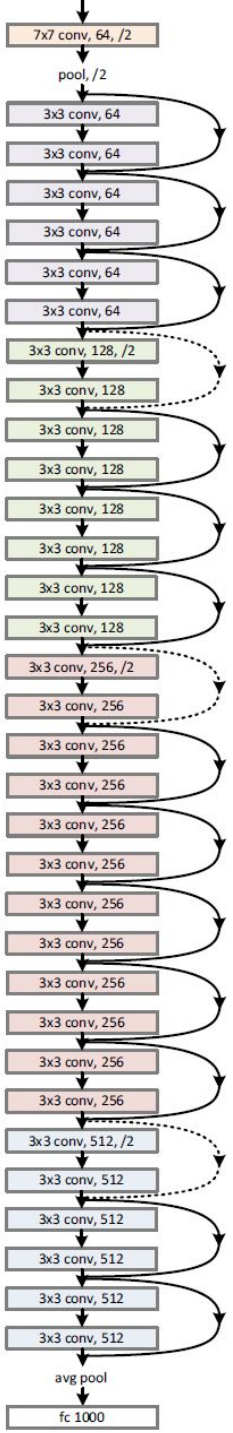


If the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counterpart.

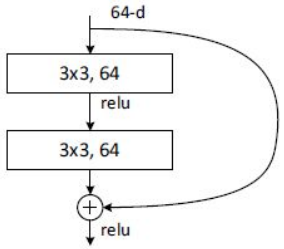
The degradation problem suggests that the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers.

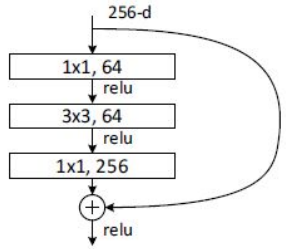
With the residual learning, if identity mappings are optimal, the solvers may simply drive the weights of the multiple nonlinear layers toward zero to approach identity mappings.

It's not entirely clear why plain (non-resnets) deep networks have difficulties, but it's not overfitting (training error also degrades), and not vanishing/exploding gradients (networks are trained with batch normalization, and gradients are healthy).



The operation F + x is performed by a shortcut connection and element-wise addition (e.g. 64 original feature maps are added to the new 64 feature maps to produce 64 output feature maps.





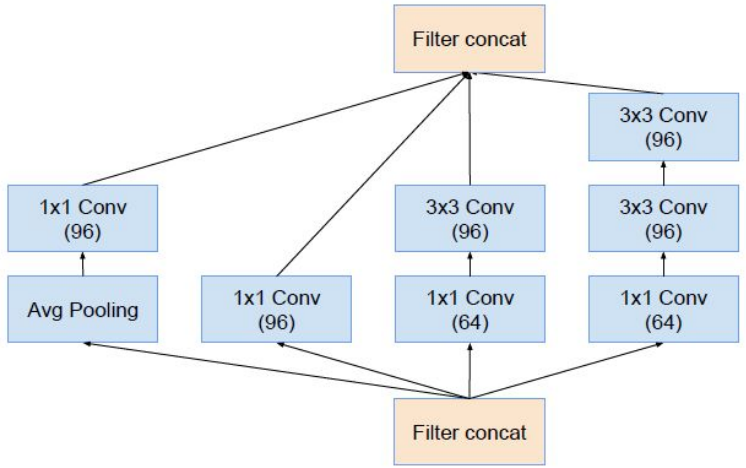
When changing dimensions or number of feature maps:

(A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameters
(B) 1x1 convolutions are used to match dimensions (this adds parameters)
For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.
B performs slightly better than A

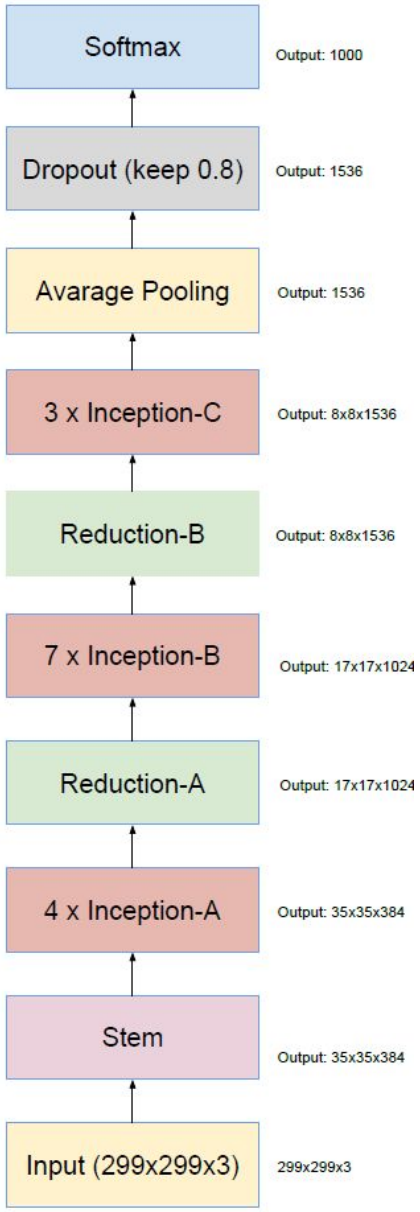
He et al. Deep Residual Learning for Image Recognition

Inception v4 (2015)

- Demonstrated no degradation problem reported in ResNet paper, while training very deep networks
- Wider and deeper Inception v3
- Inception-ResNet: Inception module with a shortcut connection (speeds up learning)
- Stabilized training by scaling down residual activations (0.1) before adding with a shortcut
- Inception v4 + 3 Inception-ResNets ensemble:
  Top-1: 16.5% Top-5: 3.1%



Network	Crops	Top-1 Error	Top-5 Error
ResNet-151 [5]	10	21.4%	5.7%
Inception-v3 [15]	12	19.8%	4.6%
Inception-ResNet-v1	12	19.8%	4.6%
Inception-v4	12	18.7%	4.2%
Inception-ResNet-v2	12	18.7%	4.1%



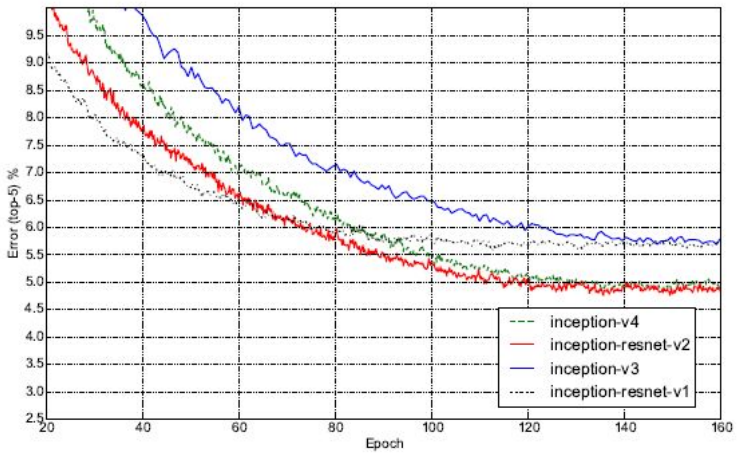
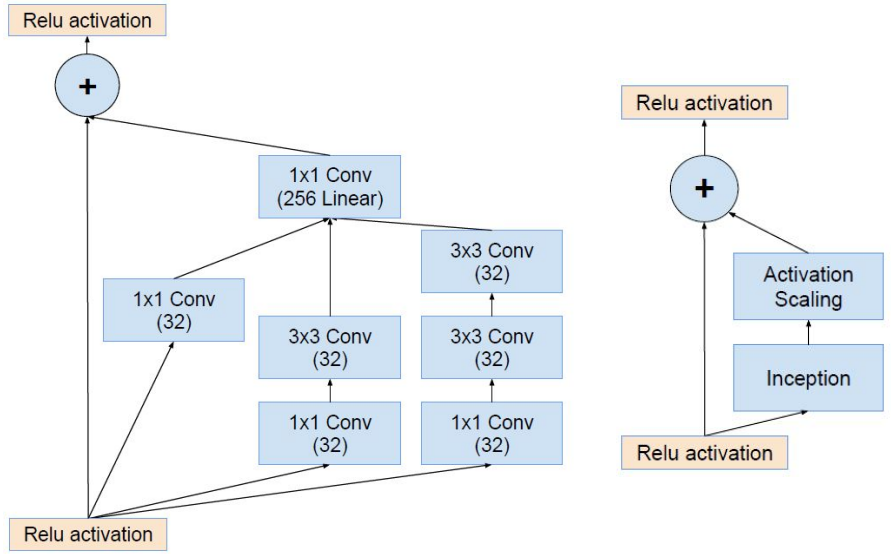


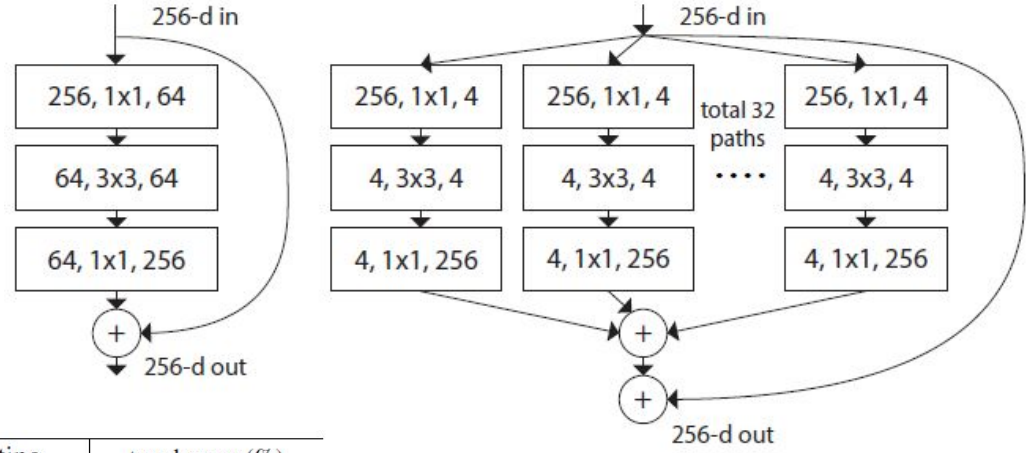
Figure 25. Top-5 error evolution of all four models (single model, single crop). Showing the improvement due to larger model size. Although the residual version converges faster, the final accuracy seems to mainly depend on the model size.



Szegedy et al. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning

ResNeXt (2016)

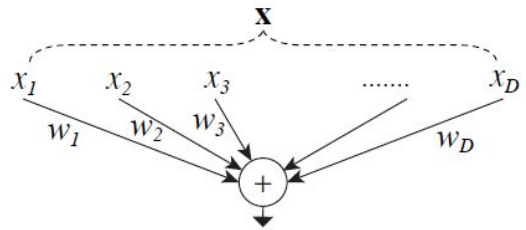
- Split-Transform-Merge principle from Inception
- Grouped Convolutions (from AlexNet)
- New model parameter: Cardinality
- Simpler design than Inception
  - Same topology along multiple paths
- Better accuracy at the same cost



	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [14]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

“Network-in-Neuron”

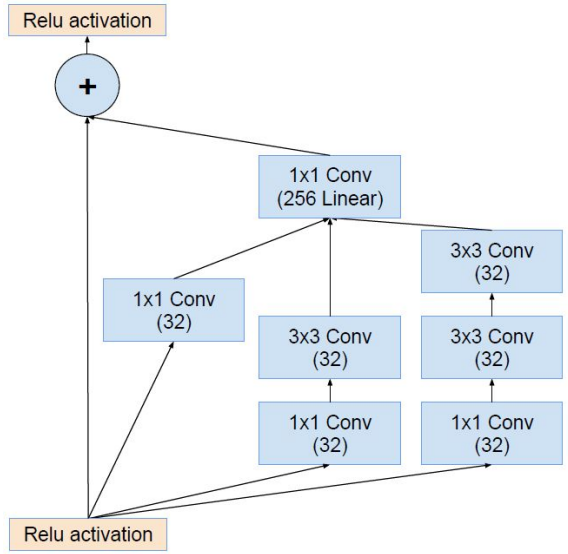


re 2. A simple neuron that performs inner product

$$\mathcal{F}(\mathbf{x}) = \sum_{i=1}^{C} \mathcal{T}_i(\mathbf{x})$$

$\mathcal{T}_i(\mathbf{x})$ can be an arbitrary function

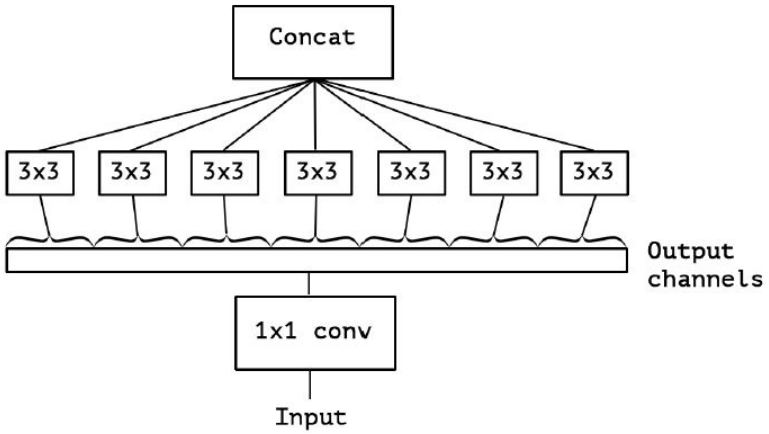
Inception-ResNet module

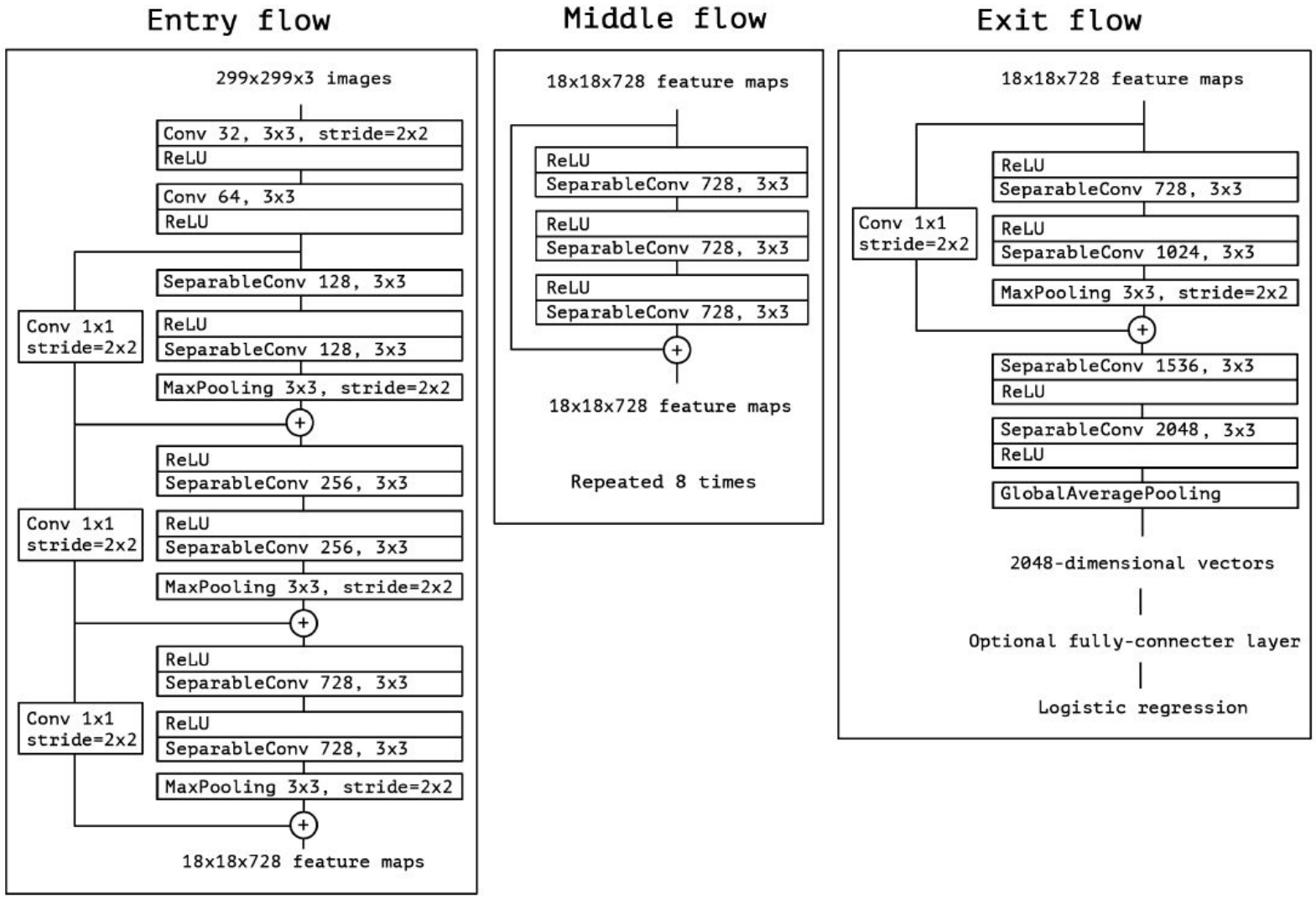


Xie et al. Aggregated Residual Transformations for Deep Neural Networks

Xception (2016)

- Same idea as ResNeXt, taken to the eXtreme
- Separable Convolutions: decouple channel correlations and spatial correlations: “it’s preferable not to map them jointly”
- Do not use ReLU between 1x1 and 3x3 mappings (helps for Inception though)
- Faster training and better accuracy than Inception v3 even without optimizations



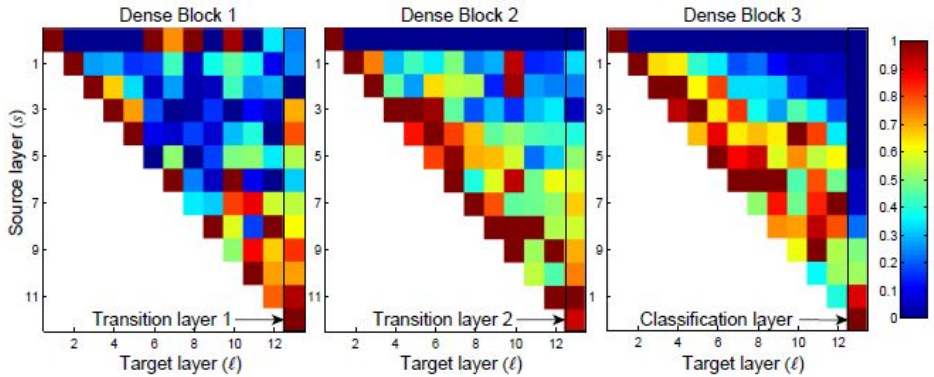


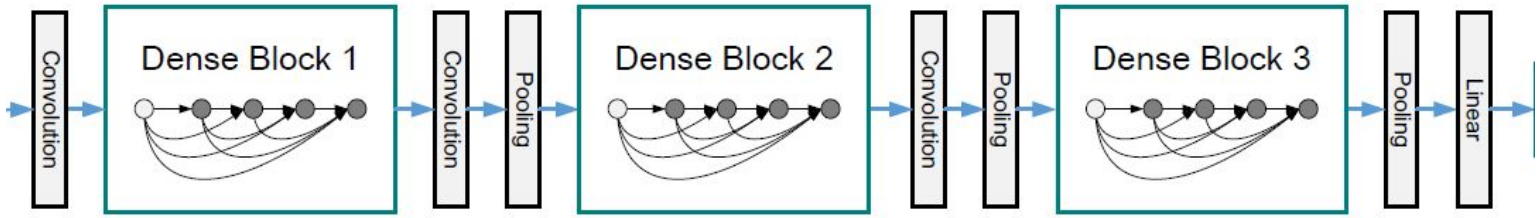
Chollet, Deep Learning with Separable Convolutions

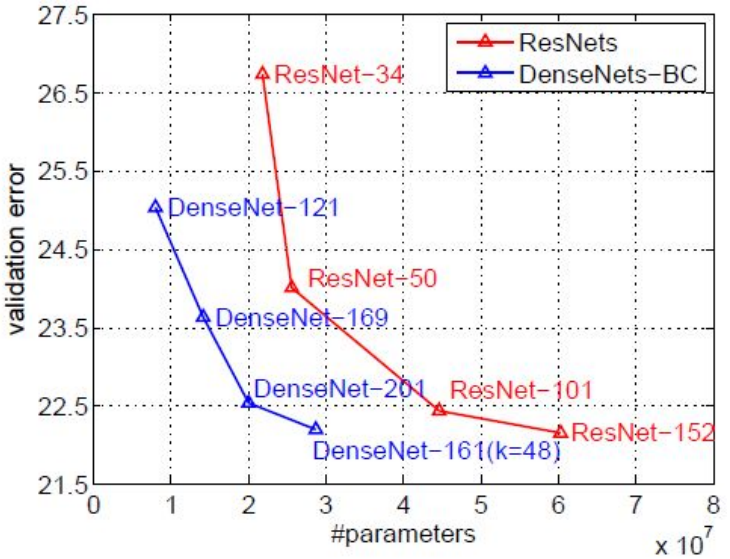
DenseNet (2016)

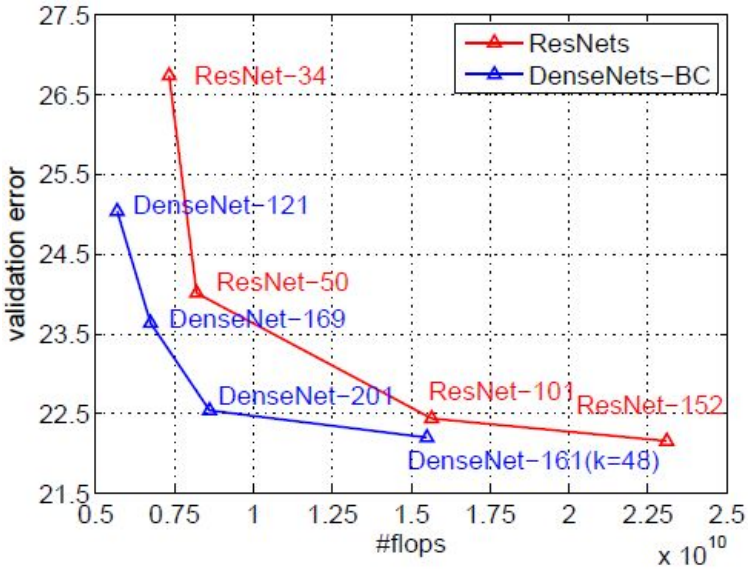
- Feature maps of each layer serve as input to all consecutive layers
- Feature maps are concatenated (not summed as in ResNets)
- Feature reuse allows very narrow layers, thus fewer parameters, and no need to relearn redundant feature maps
- Each layer has short path for gradients from the loss function, and the original input signal
- Inside and outside of Dense Blocks 1x1 layers are used to reduce number of FMs
- A single classifier on top of the network provides direct supervision to all layers through at most 2 or 3 transition layers

DenseNet-169($k = 32$)	DenseNet-201($k = 32$)
7×7 conv, stride 2	
3×3 max pool, stride 2	
$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
1×1 conv	
2×2 average pool, stride 2	
$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
1×1 conv	
2×2 average pool, stride 2	
$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
1×1 conv	
2×2 average pool, stride 2	
$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$
7×7 global average pool	
1000D fully-connected, softmax	





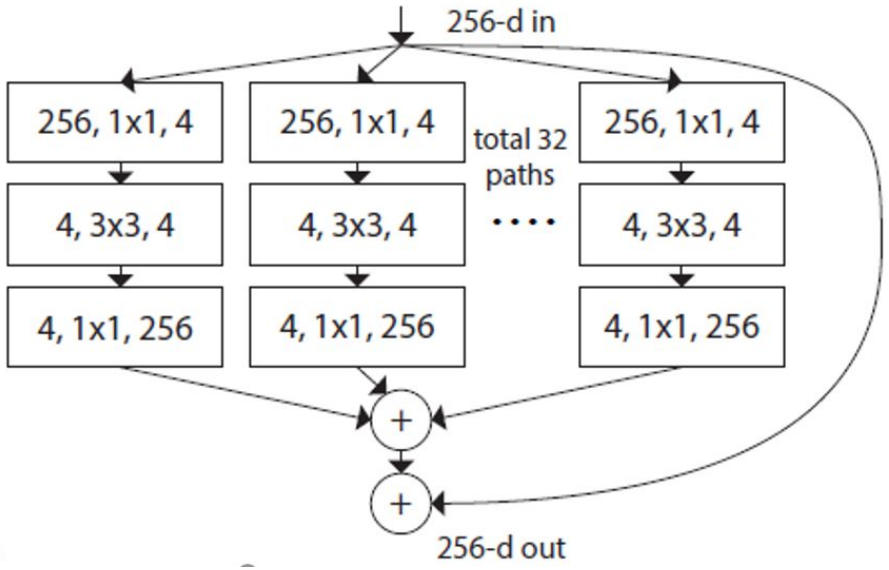


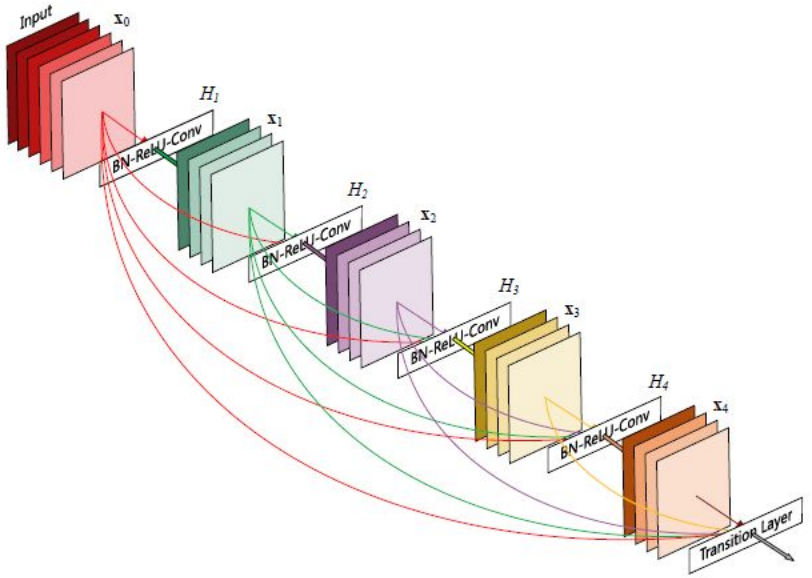


Huang et al, Densely Connected Convolutional Networks

What's next: Dense ResNeXt?

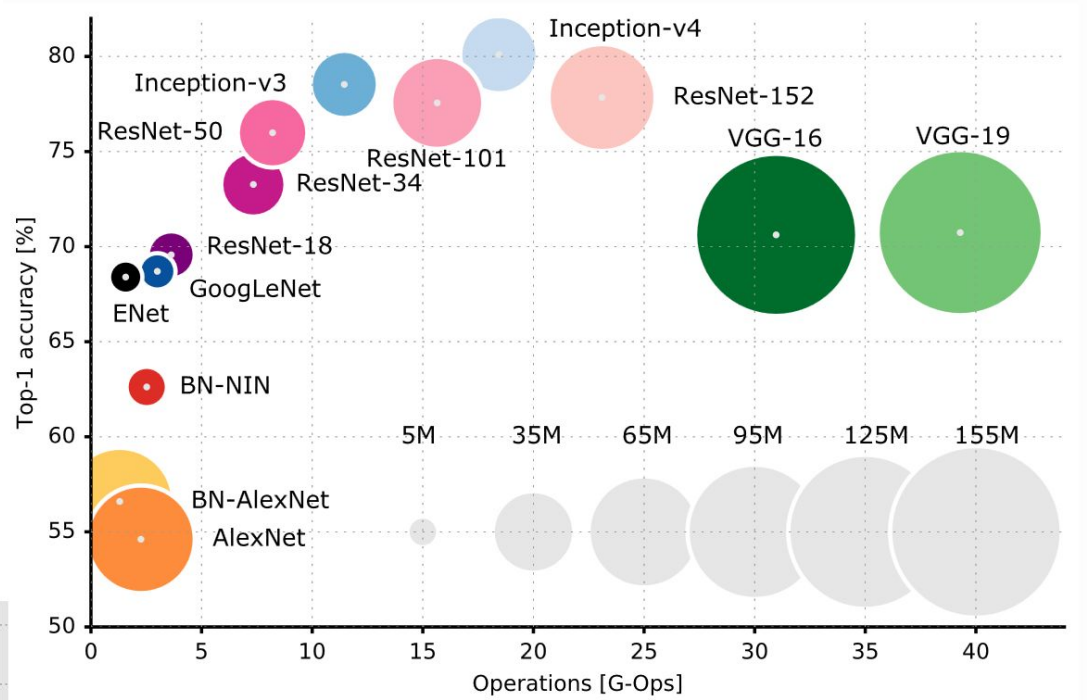
- Combine grouped convolutions idea from ResNeXt and full connectivity of DenseNet
- Replace 1x1-3x3 modules in Dense Blocks with 1x1-3x3-1x1 grouped convolution modules
- Concatenate output feature maps with feature maps from previous layers
  - Interleave or side-by-side? (does not matter for Xception stype network)
- Try longer parallel paths?
  - Instead of "split-transform-merge" do "split-transform-transform-transform-merge"
  - Extreme variant is multiple narrow parallel networks scanning the same input, and sharing the output layer
- Multiscale feature matching: correlate feature maps of different dimensions

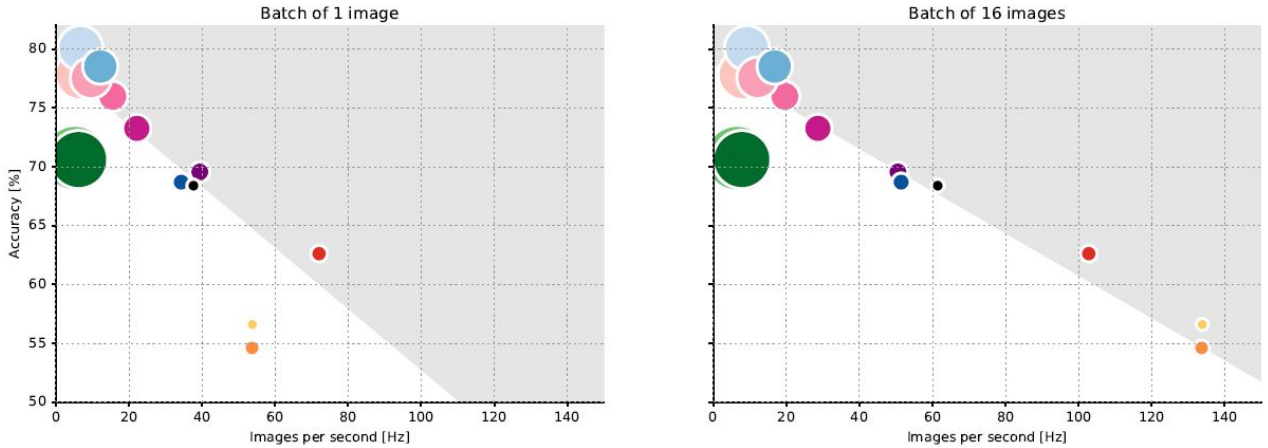




Efficiency

- Various models tested on the same hardware (Nvidia TX1 board)
- Accuracy vs Speed is approximately linear
- Accuracy vs Number of parameters is not clear
- Accuracy vs Weight Precision is not clear
- Number of weights, weight precision, and number of operations can be balanced to provide optimal efficiency for target accuracy





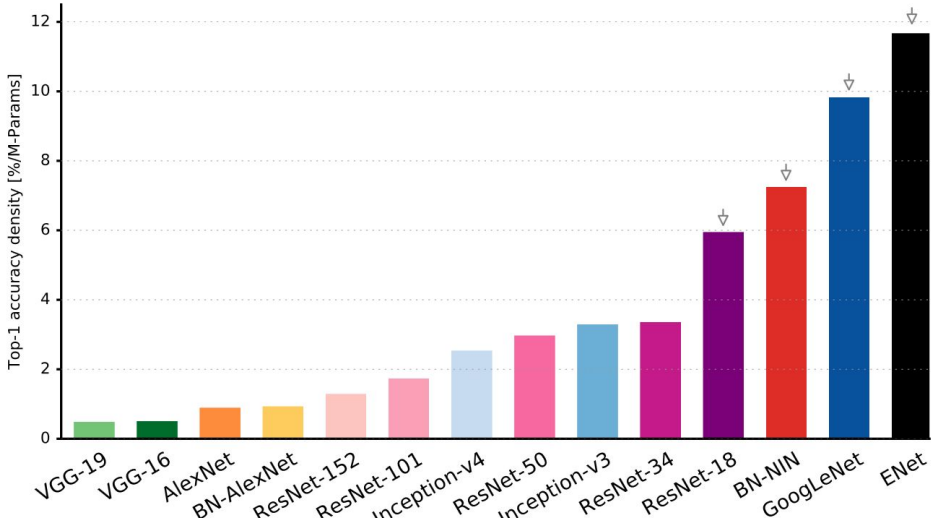


Figure 9: **Accuracy *vs.* inferences per second, size $\propto$ operations.** Non trivial linear upper bound is shown in these scatter plots, illustrating the relationship between prediction accuracy and throughput of all examined architectures. These are the first charts in which the area of the blobs is proportional to the amount of operations, instead of the parameters count. We can notice that larger blobs are concentrated on the left side of the charts, in correspondence of low throughput, *i.e.* longer inference times. Most of the architectures lay on the linear interface between the grey and white areas. If a network falls in the shaded area, it means it achieves exceptional accuracy or inference speed. The white area indicates a suboptimal region. *E.g.* both AlexNet architectures

Canziani & Culurciello, An Analysis of Deep Neural Network Models for Practical Applications