NPFL114, Lecture 05

Convolutional Networks II

Milan Straka

Convolution Layer

The K is usually called a *kernel* or a *filter*, and we generally apply several of them at the same time.

Consider an input image with C channels. The convolution layer with F filters of width W, height H and stride S produces an output with F channels kernels of total size $W \times H \times C \times F$ and is computed as

$$(\boldsymbol{I} * \boldsymbol{K})_{i,j,k} = \sum_{m,n,o} \boldsymbol{I}_{i \cdot S + m, j \cdot S + n, o} \boldsymbol{K}_{m,n,o,k}.$$

Convolution Layer

The K is usually called a *kernel* or a *filter*, and we generally apply several of them at the same time.

Consider an input image with C channels. The convolution layer with F filters of width W, height H and stride S produces an output with F channels kernels of total size $W \times H \times C \times F$ and is computed as

$$(\boldsymbol{I} * \boldsymbol{K})_{i,j,k} = \sum_{m,n,o} \boldsymbol{I}_{i \cdot S + m, j \cdot S + n, o} \boldsymbol{K}_{m,n,o,k}.$$

There are multiple padding schemes, most common are:

- `valid`: we only use valid pixels, which causes the result to me smaller
- `same`: we pad original image with zero pixels so that the result is exactly the size of the input

Convolution Layer

There are two prevalent image formats:

- `NHWC` or `channels_last`: The dimensions of the 4-dimensional image tensor are batch, height, width, and channels.

  The original TensorFlow format, faster on CPU.

Convolution Layer

There are two prevalent image formats:

- `NHWC` or `channels_last`: The dimensions of the 4-dimensional image tensor are batch, height, width, and channels.

  The original TensorFlow format, faster on CPU.

- `NCHW` or `channels_first`: The dimensions of the 4-dimensional image tensor are batch, channel, height, and width.

  Usual GPU format (used by CUDA and nearly all frameworks); on TensorFlow, not all CPU kernels are available with this layout.

Pooling

Pooling is an operation similar to convolution, but we perform a fixed operation instead of multiplying by a kernel.

- Max pooling: minor translation invariance
- Average pooling

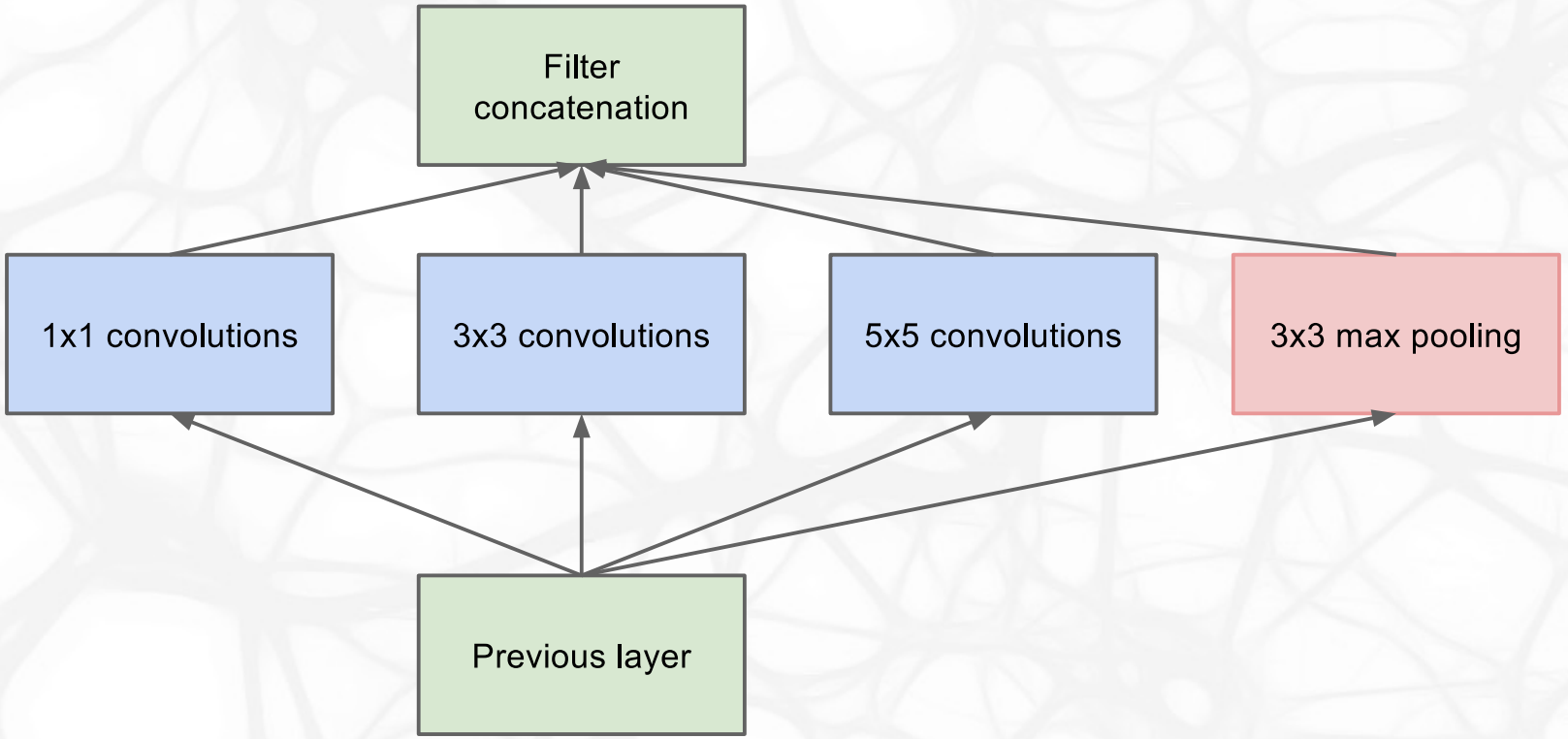
VGG – 2014 (6.8% error)

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					

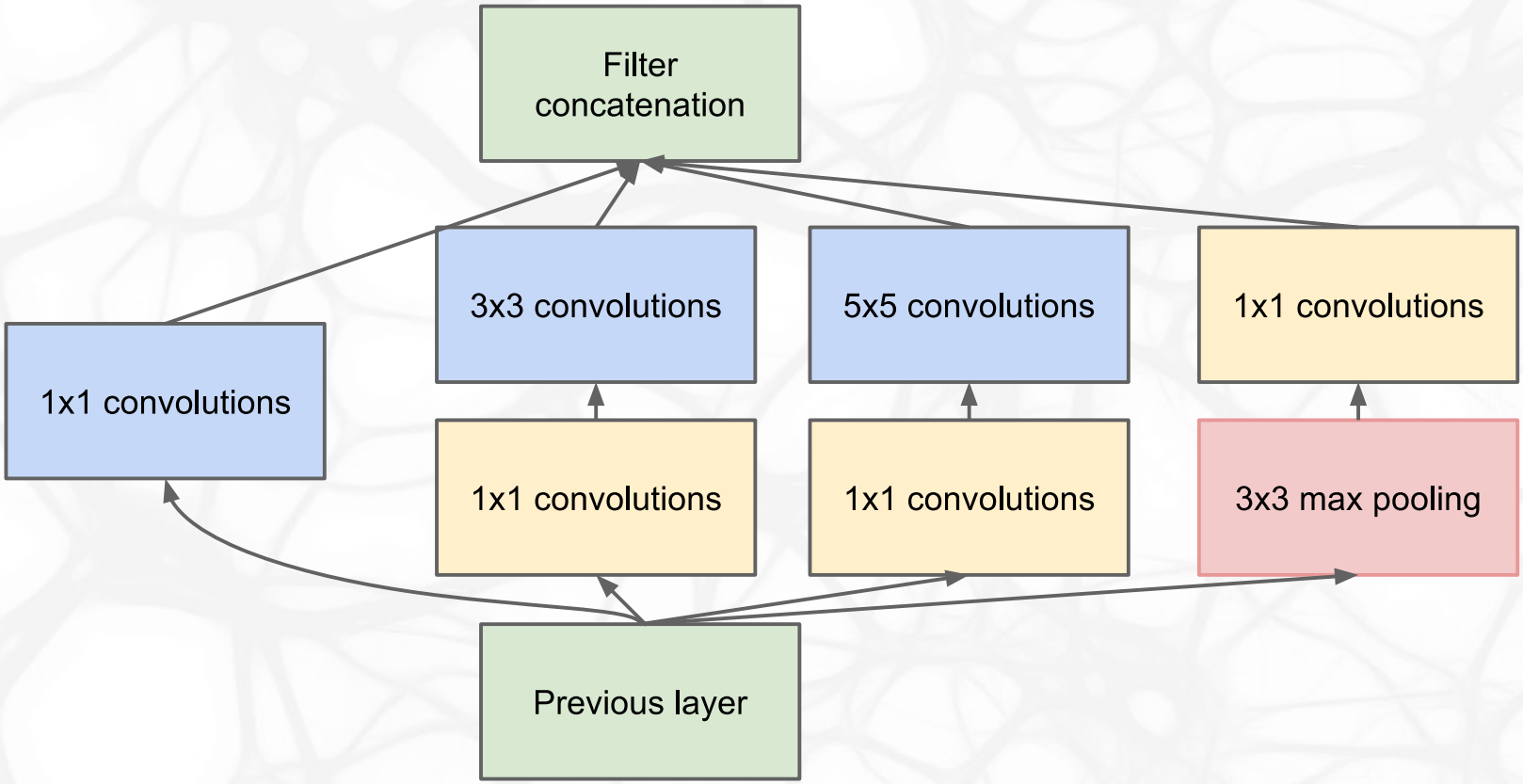
Table 2: **Number of parameters** (in millions).

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

Inception (GoogLeNet) – 2014 (6.7% error)



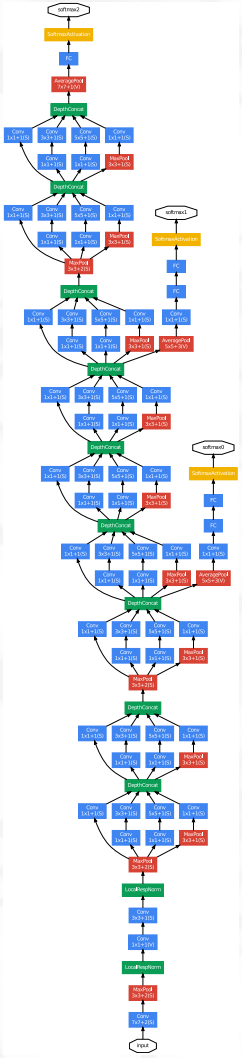
Inception (GoogLeNet) – 2014 (6.7% error)



Inception (GoogLeNet) – 2014 (6.7% error)

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Inception (GoogLeNet) – 2014 (6.7% error)



Also note the two auxiliary classifiers (they have weight 0.3).

Batch Normalization

Internal covariate shift refers to the change in the distributions of hidden node activations due to the updates of network parameters during training.

Let $\boldsymbol{x} = (x_1, \ldots, x_d)$ be d-dimensional input. We would like to normalize each dimension as

$$\hat{x}_i = \frac{x_i - \mathbb{E}[x_i]}{\sqrt{\mathrm{Var}[x_i]}}.$$

Furthermore, it may be advantageous to learn suitable scale γ_i and shift β_i to produce normalized value

$$y_i = \gamma_i \hat{x}_i + \beta_i.$$

Batch Normalization

Consider a mini-batch of m examples $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$.

Batch normalizing transform of the mini-batch is the following transformation.

Inputs: Mini-batch $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}), \varepsilon \in \mathbb{R}$
Outputs: Normalized batch $(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)})$

- $\boldsymbol{\mu} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}^{(i)}$
- $\boldsymbol{\sigma}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{x}^{(i)} - \mu)^2$
- $\hat{\boldsymbol{x}}^{(i)} \leftarrow (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}) / \sqrt{\sigma^2 + \varepsilon}$
- $\boldsymbol{y}^{(i)} \leftarrow \boldsymbol{\gamma} \hat{\boldsymbol{x}}^{(i)} + \boldsymbol{\beta}$

Batch Normalization

Consider a mini-batch of m examples $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$.

Batch normalizing transform of the mini-batch is the following transformation.

Inputs: Mini-batch $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}), \varepsilon \in \mathbb{R}$
Outputs: Normalized batch $(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)})$

- $\boldsymbol{\mu} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}^{(i)}$
- $\boldsymbol{\sigma}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{x}^{(i)} - \mu)^2$
- $\hat{\boldsymbol{x}}^{(i)} \leftarrow (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}) / \sqrt{\sigma^2 + \varepsilon}$
- $\boldsymbol{y}^{(i)} \leftarrow \boldsymbol{\gamma} \hat{\boldsymbol{x}}^{(i)} + \boldsymbol{\beta}$

Batch normalization is commonly added just before a nonlinearity. Therefore, we replace $f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ by $f(BN(\boldsymbol{W}\boldsymbol{x}))$.

Batch Normalization

Consider a mini-batch of m examples $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$.

Batch normalizing transform of the mini-batch is the following transformation.

Inputs: Mini-batch $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}), \varepsilon \in \mathbb{R}$
Outputs: Normalized batch $(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)})$

- $\boldsymbol{\mu} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}^{(i)}$
- $\boldsymbol{\sigma}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{x}^{(i)} - \mu)^2$
- $\hat{\boldsymbol{x}}^{(i)} \leftarrow (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}) / \sqrt{\sigma^2 + \varepsilon}$
- $\boldsymbol{y}^{(i)} \leftarrow \boldsymbol{\gamma} \hat{\boldsymbol{x}}^{(i)} + \boldsymbol{\beta}$

Batch normalization is commonly added just before a nonlinearity. Therefore, we replace $f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ by $f(BN(\boldsymbol{W}\boldsymbol{x}))$.

During inference, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are fixed. They are either precomputed after training on the whole training data, or an exponential moving average is updated during training.

Inception with BatchNorm (4.8% error)

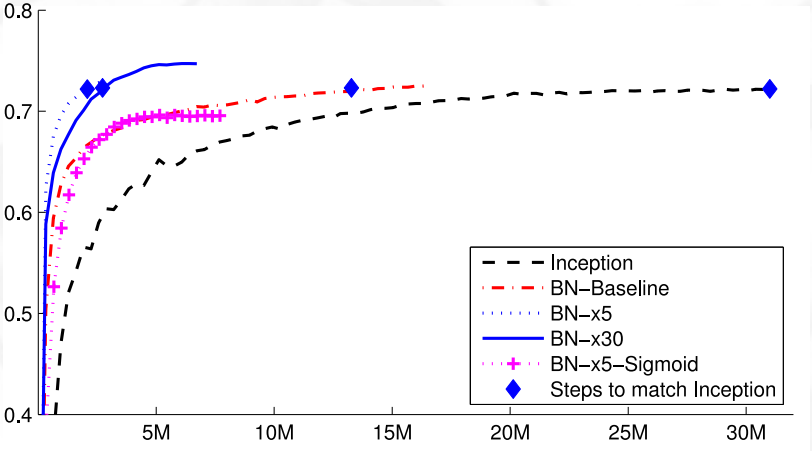
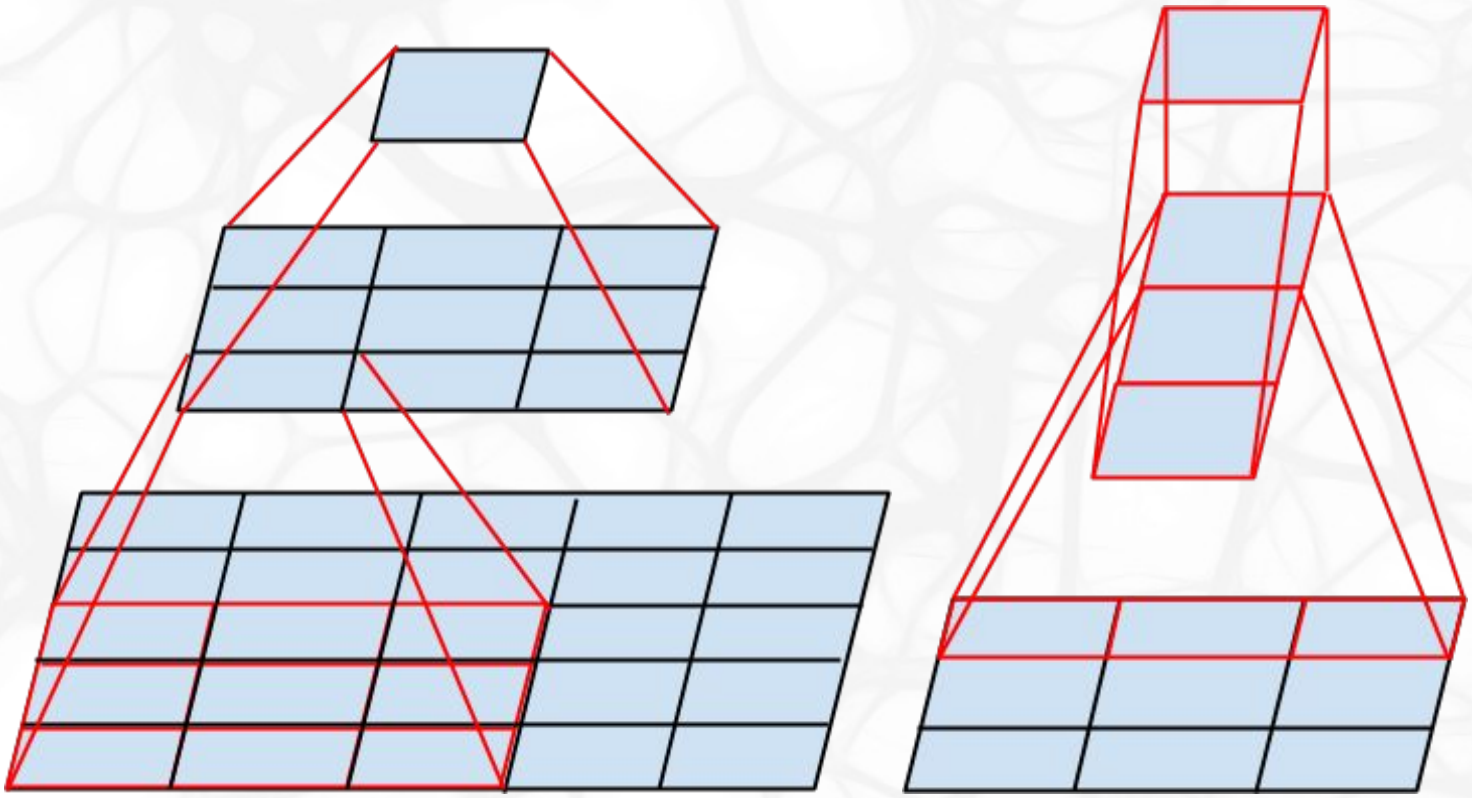


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

F gure 3: *For Inception and the batch-normalized variants, the number of training steps required to reach the maximum accuracy of Inception (72.2%), and the maximum accuracy achieved by the network.*

Inception v2 and v3 – 2015 (3.6% error)



Inception v2 and v3 – 2015 (3.6% error)

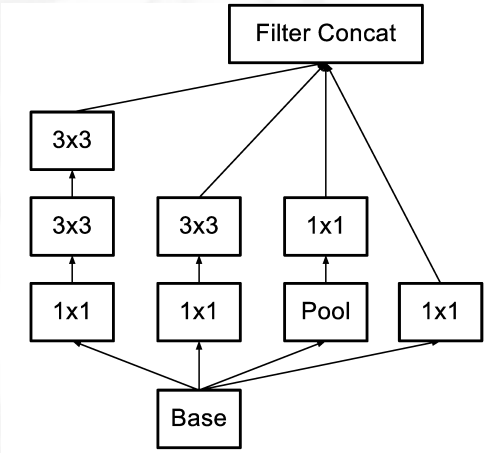


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

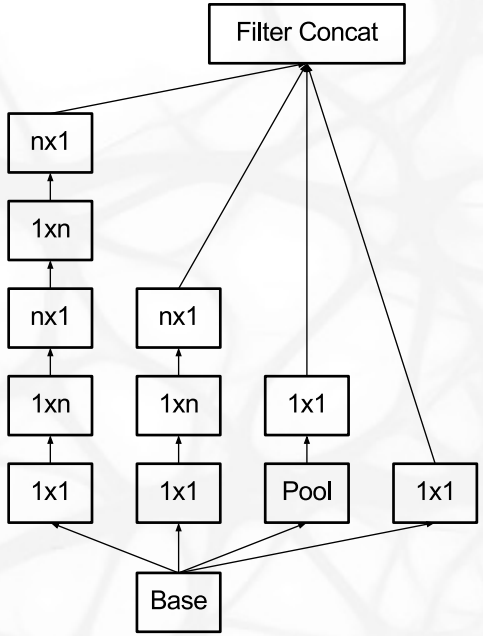


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

.

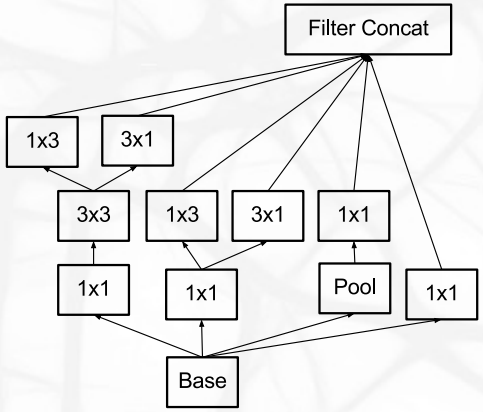


Figure 7. Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8×8) grids to promote high dimensional representations, as suggested by principle 2 of Section 2. We are using this solution only on the coarsest grid, since that is the place where producing high dimensional sparse representation is the most critical as the ratio of local processing (by 1×1 convolutions) is increased compared to the spatial aggregation.

Inception v2 and v3 – 2015 (3.6% error)

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	$3\times3/1$	$35\times35\times192$
$3\times$Inception	As in figure 5	$35\times35\times288$
$5\times$Inception	As in figure 6	$17\times17\times768$
$2\times$Inception	As in figure 7	$8\times8\times1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

Inception v2 and v3 – 2015 (3.6% error)

Network	Top-1 Error	Top-5 Error	Cost Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	**1.5**
BN-Inception [7]	25.2%	7.8	2.0
Inception-v2	23.4%	-	3.8
Inception-v2 RMSProp	23.1%	6.3	3.8
Inception-v2 Label Smoothing	22.8%	6.1	3.8
Inception-v2 Factorized 7×7	21.6%	5.8	4.8
Inception-v2 BN-auxiliary	**21.2%**	**5.6%**	4.8

ResNet – 2015 (3.6% error)

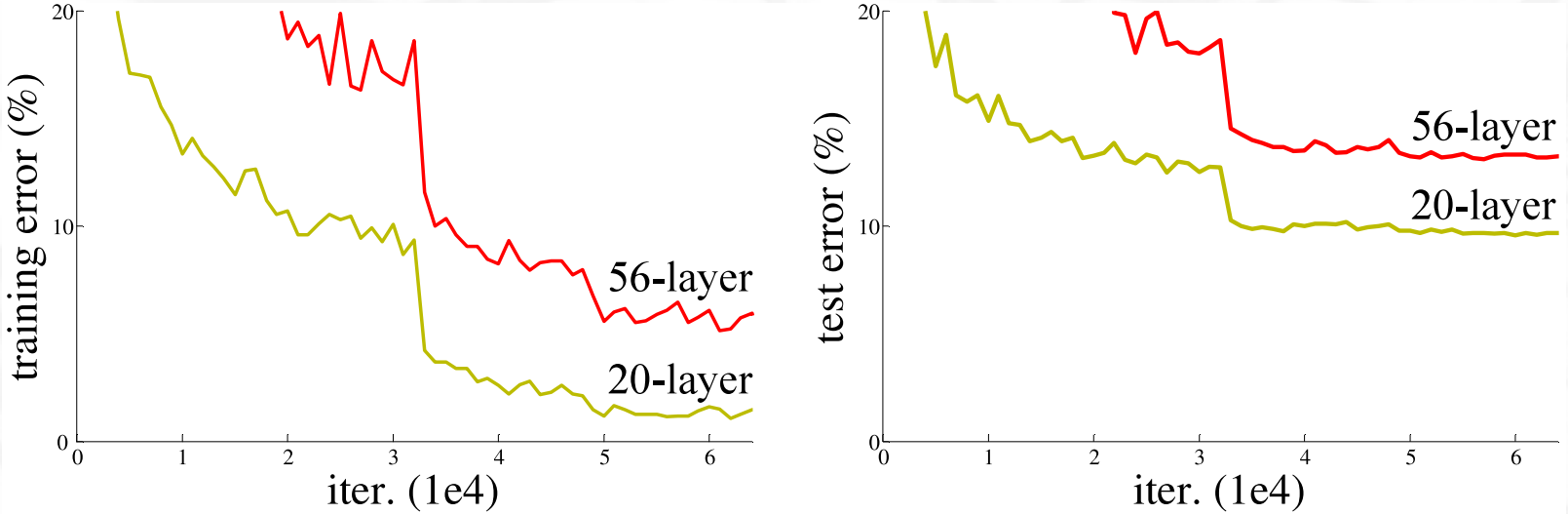


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

ResNet – 2015 (3.6% error)

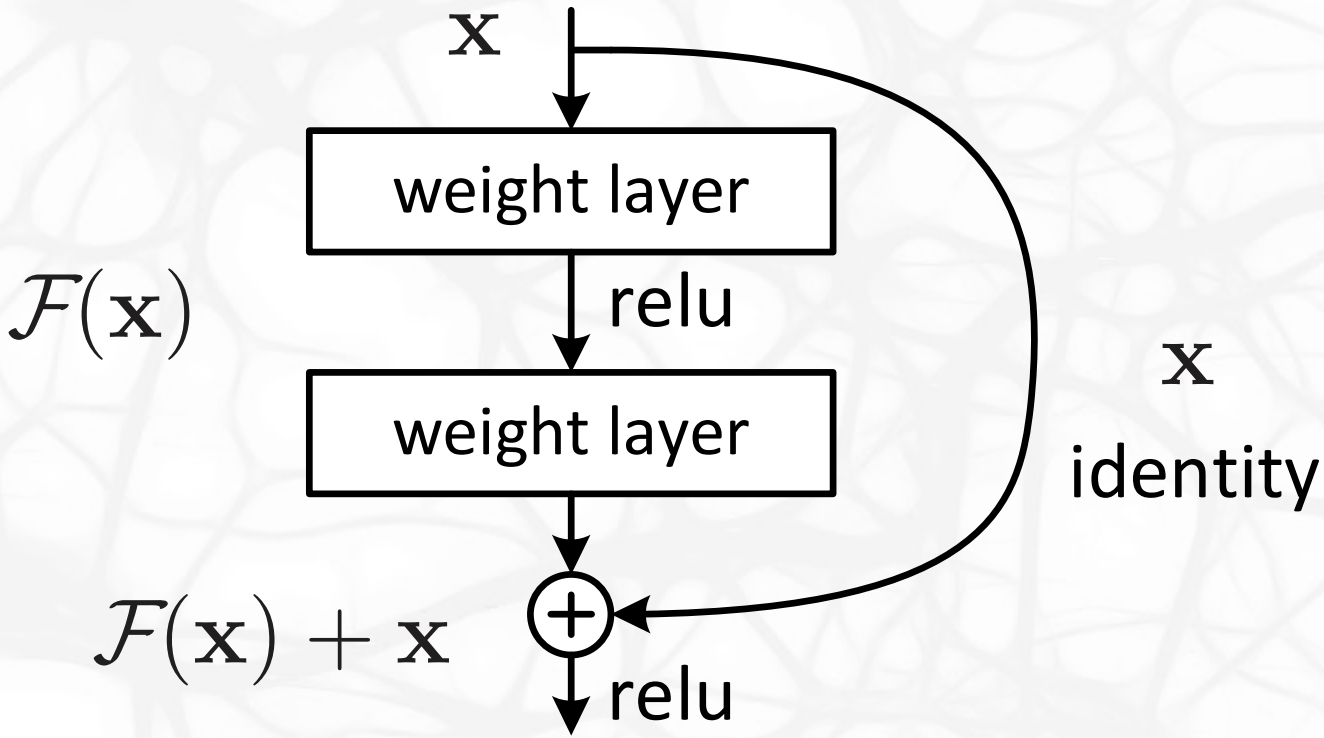


Figure 2. Residual learning: a building block.

ResNet – 2015 (3.6% error)

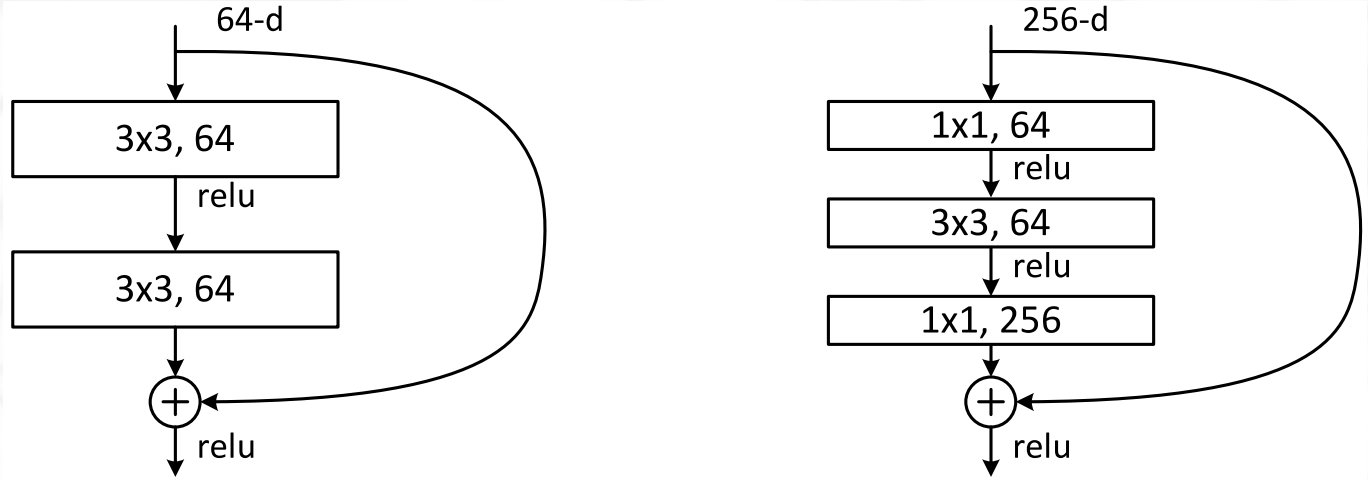
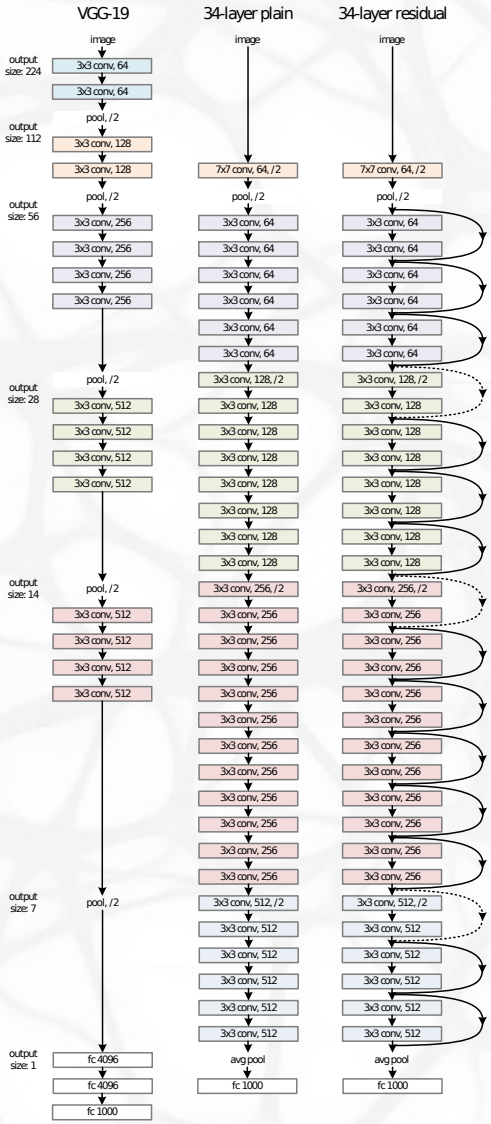


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

ResNet – 2015 (3.6% error)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet – 2015 (3.6% error)



ResNet – 2015 (3.6% error)

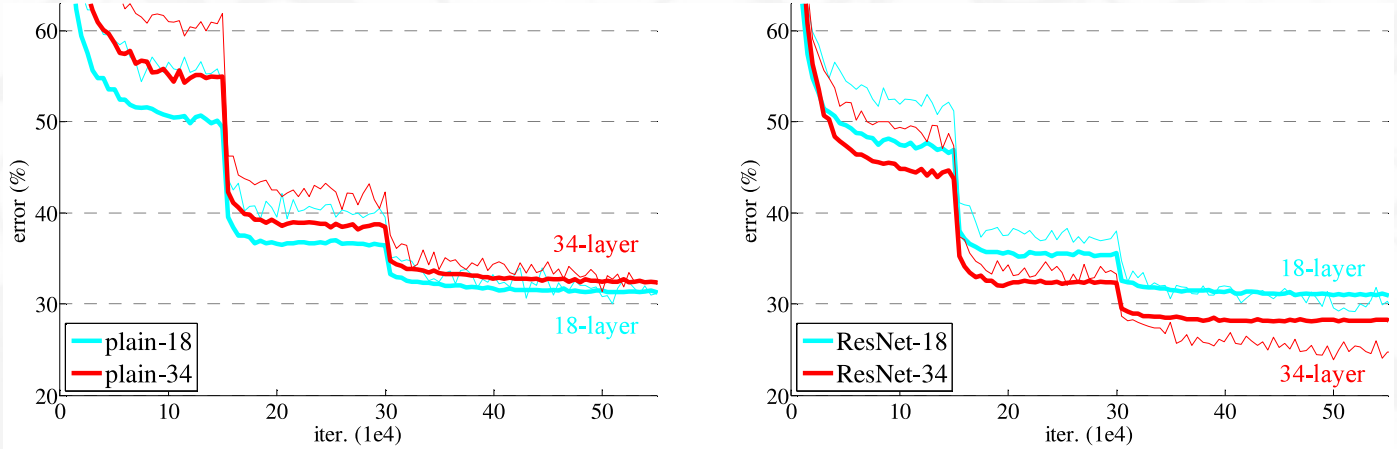
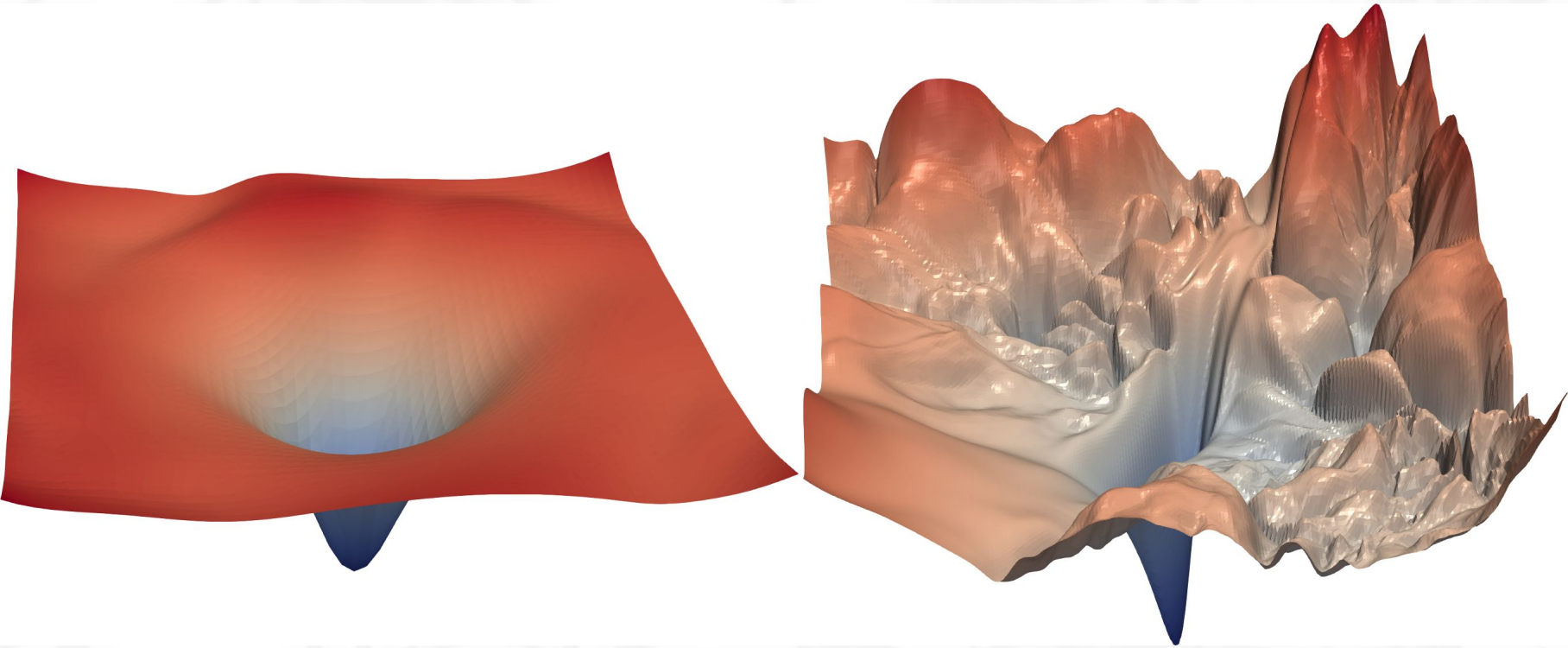
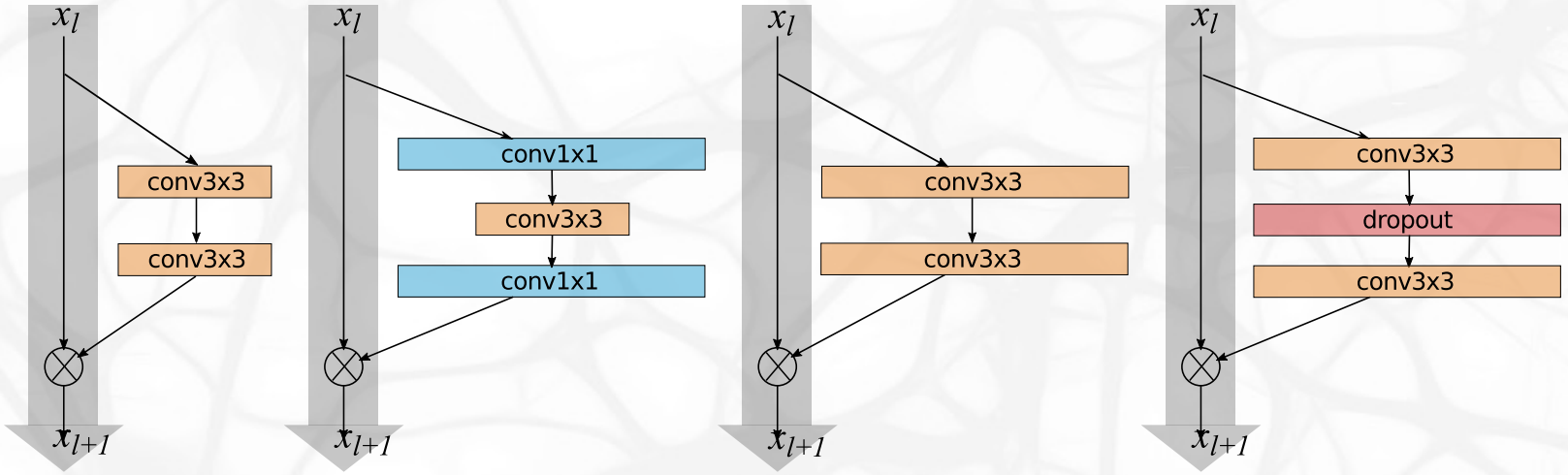


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 laye s. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

ResNet – 2015 (3.6% error)



WideNet – 2016



(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

Figure 1: Various residual blocks used in the paper. Batch normalization and ReLU precede each convolution (omitted for clarity)

ResNeXt – 2016

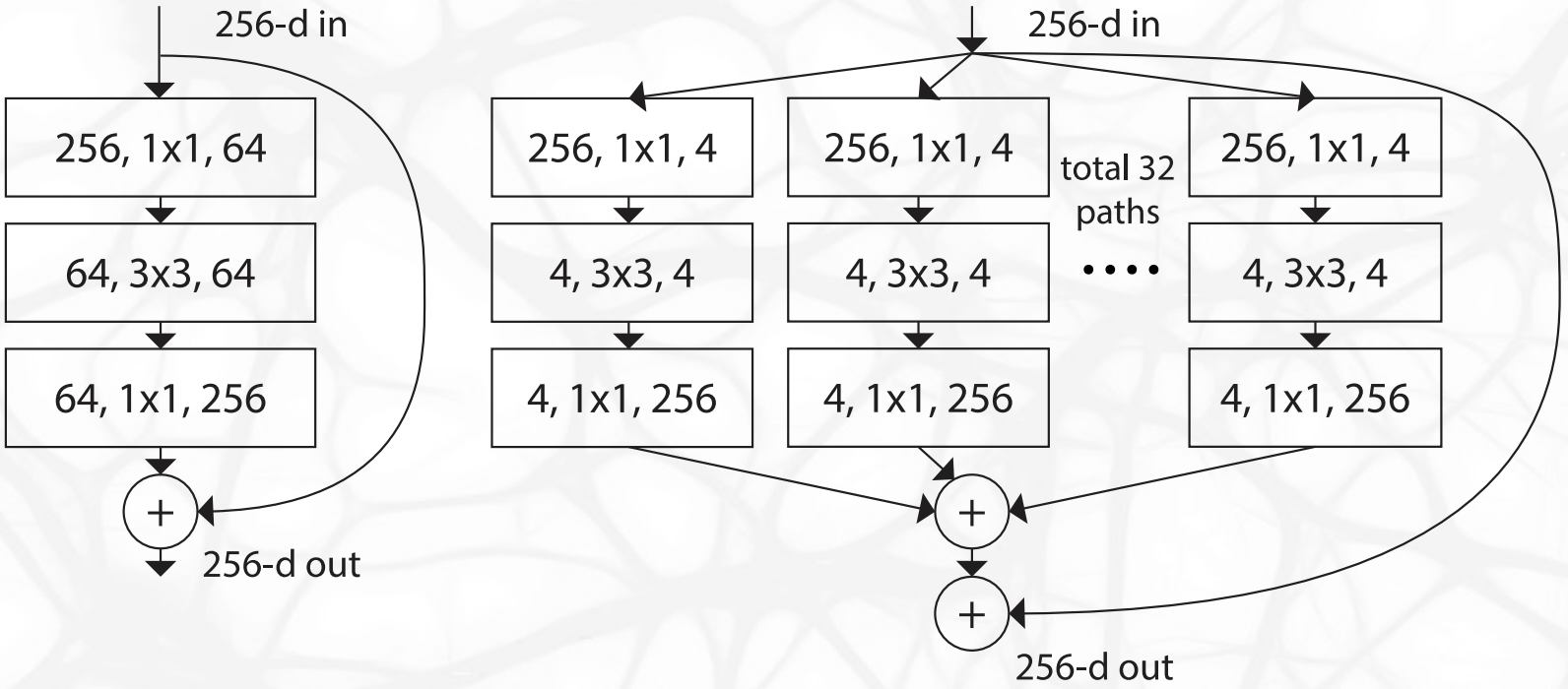
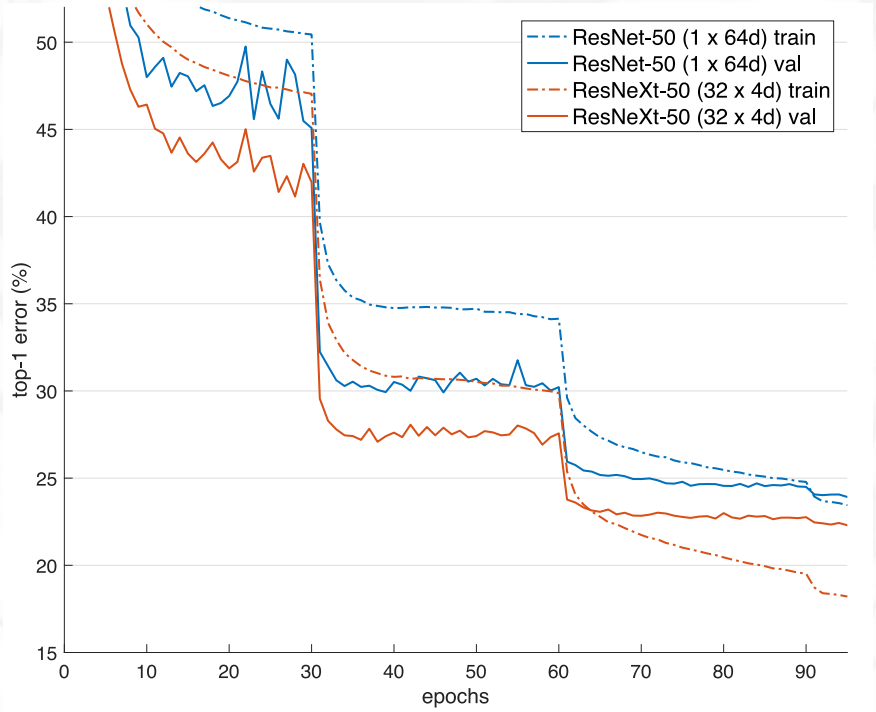


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

ResNeXt – 2016

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix}\times6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 5\ \ 2 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

ResNeXt – 2016



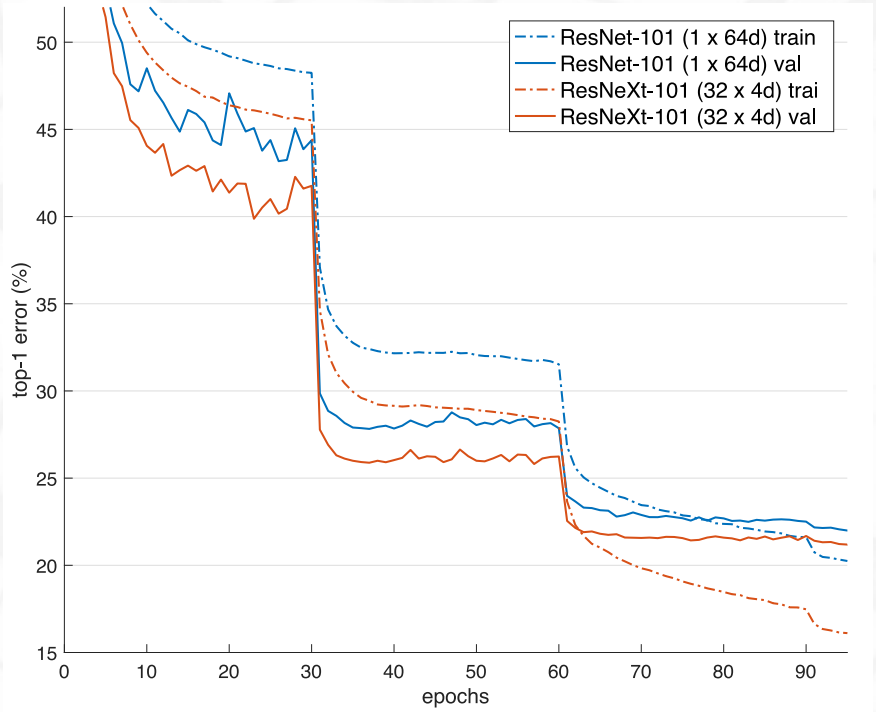
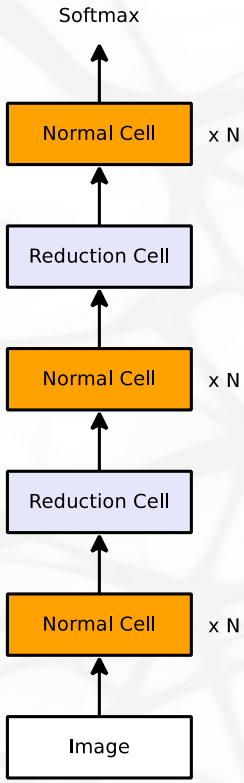
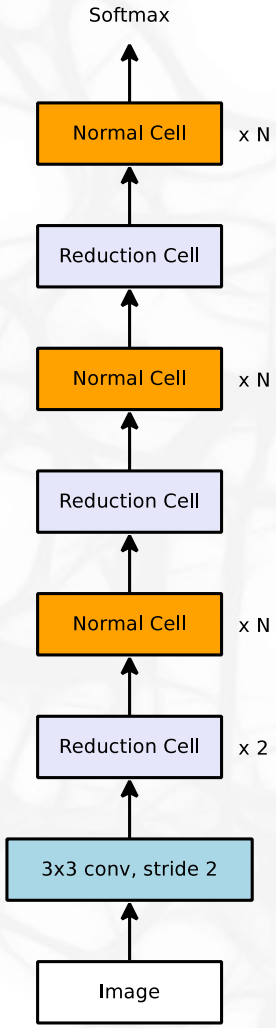


Figure 5. Training curves on ImageNet-1K. (**Left**): ResNet/ResNeXt-50 with preserved complexity (∼4.1 billion FLOPs, ∼25 million parameters); (**Right**): ResNet/ResNeXt-101 with preserved complexity (∼7.8 billion FLOPs, ∼44 million parameters).

NasNet – 2017



CIFAR10
Architecture



ImageNet
Architecture

NasNet – 2017

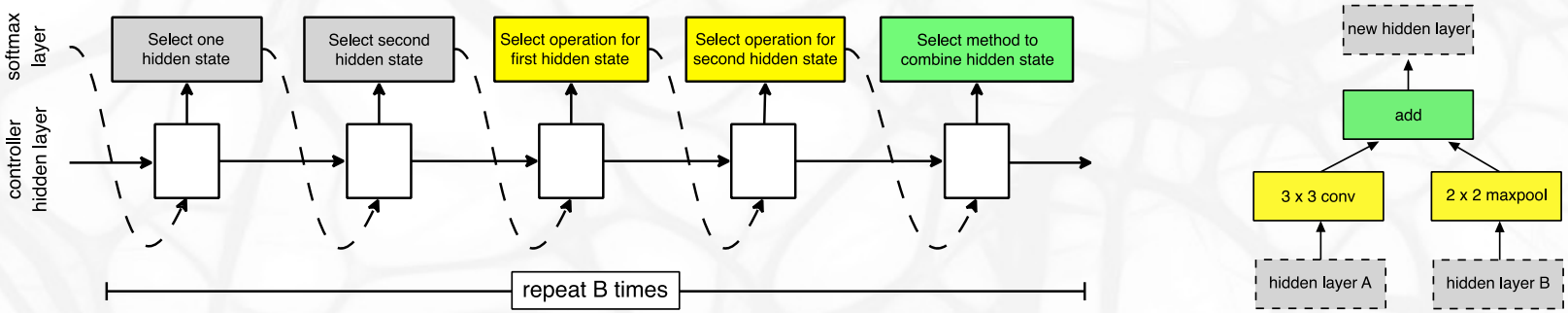
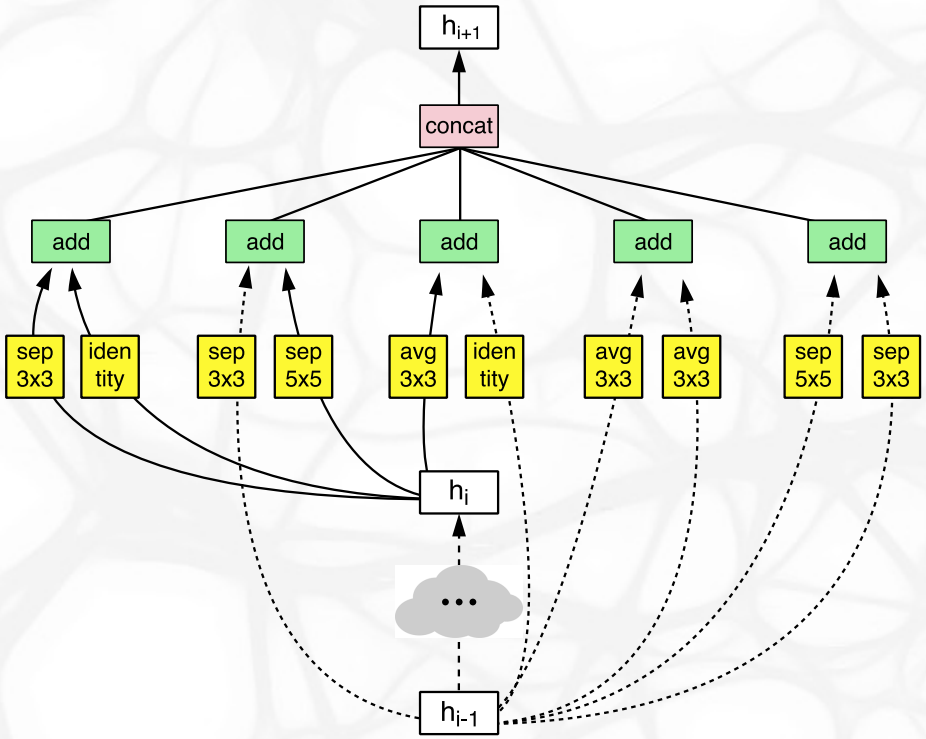
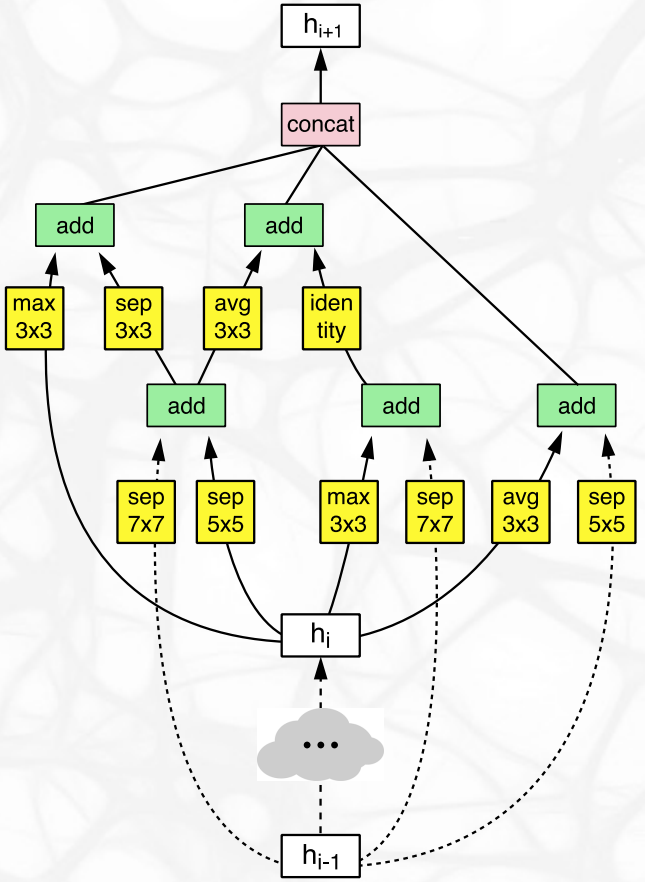


Figure 3. Controller model architecture for recursively constructing one block of a convolutional cell. Each bl ck requires selecting 5 discrete parameters, each of which corresponds to the output of a softmax layer. Example constructed block shown on right. A convolutional cell contains B blocks, hence the controller contains $5B$ softmax layers for predicting the architecture of a convolutional cell. In our experiments, the number of blocks B is 5.

NasNet – 2017



Normal Cell

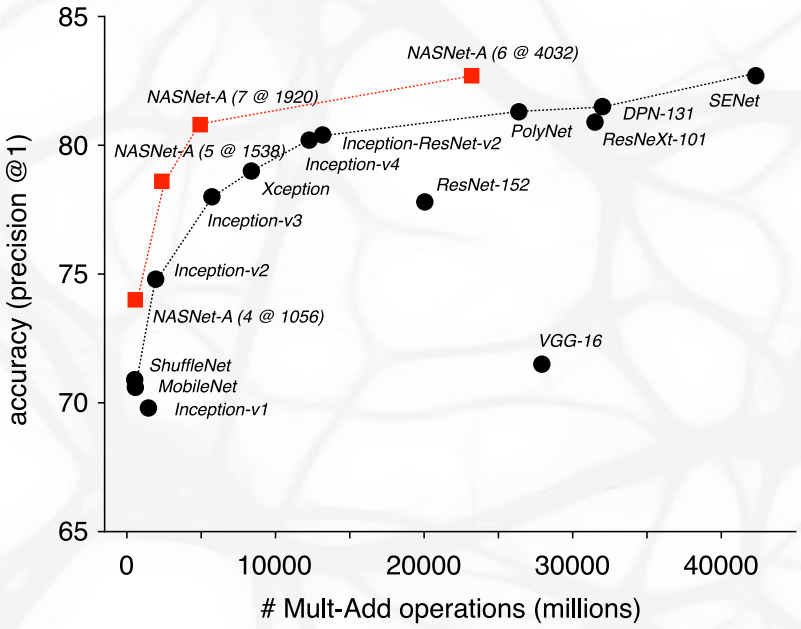


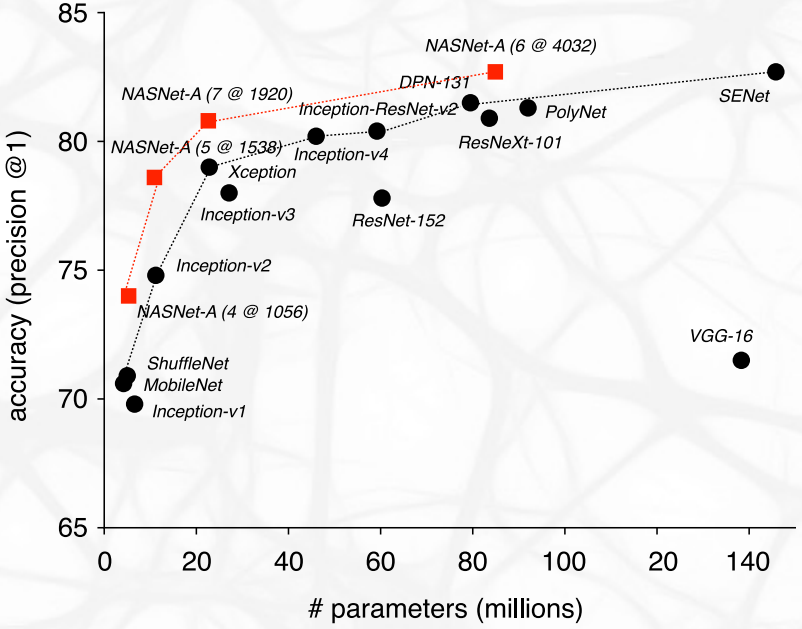
Reduction Cell

NasNet – 2017

Model	image size	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V2 [29]	224×224	11.2 M	1.94 B	74.8	92.2
NASNet-A (5 @ 1538)	**299×299**	**10.9 M**	**2.35 B**	**78.6**	**94.2**
Inception V3 [59]	299×299	23.8 M	5.72 B	78.0	93.9
Xception [9]	299×299	22.8 M	8.38 B	79.0	94.5
Inception ResNet V2 [57]	299×299	55.8 M	13.2 B	80.4	95.3
NASNet-A (7 @ 1920)	**299×299**	**22.6 M**	**4.93 B**	**80.8**	**95.3**
ResNeXt-101 (64 x 4d) [67]	320×320	83.6 M	31.5 B	80.9	95.6
PolyNet [68]	331×331	92 M	34.7 B	81.3	95.8
DPN-131 [8]	320×320	79.5 M	32.0 B	81.5	95.8
SENet [25]	**320×320**	**145.8 M**	**42.3 B**	**82.7**	**96.2**
NASNet-A (6 @ 4032)	**331×331**	**88.9 M**	**23.8 B**	**82.7**	**96.2**

NasNet – 2017


85
80
75
70
65
accuracy (precision @1)
0
10000
20000
30000
40000
Mult-Add operations (millions)
NASNet-A (6 @ 4032)
NASNet-A (7 @ 1920)
NASNet-A (5 @ 1538)
NASNet-A (4 @ 1056)
SENet
DPN-131
PolyNet
ResNeXt-101
Inception-ResNet-v2
Inception-v4
Xception
Inception-v3
Inception-v2
ResNet-152
VGG-16
ShuffleNet
MobileNet
Inception-v1


85
80
75
70
65
accuracy (precision @1)
0
20
40
60
80
100
20
140
parameters (millions)
NASNet-A (6 @ 4032)
NASNet-A (7 @ 1920)
NASNet-A (5 @ 1538)
NASNet-A (4 @ 1056)
DPN-131
Inception-ResNet-v2
PolyNet
SENet
ResNeXt-101
Inception-v4
Xception
Inception-v3
ResNet-152
Inception-v2
VGG-16
ShuffleNet
MobileNet
Inception-v1

Beyond Image Classification

Beyond Image Classification

- Transfer learning

Beyond Image Classification

- Transfer learning
- Convolutions can be applied to other dimensions than 2:

Beyond Image Classification

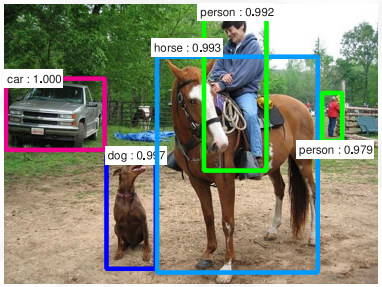
- Transfer learning
- Convolutions can be applied to other dimensions than 2:
  - 1D: speech, text (e.g., text classification)

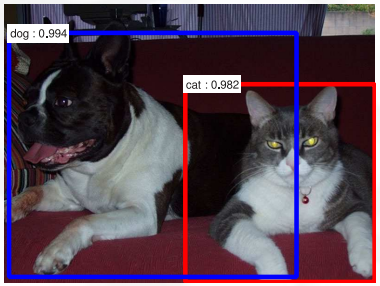
Beyond Image Classification

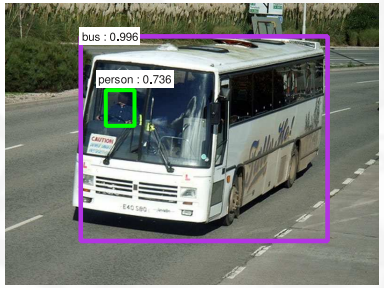
- Transfer learning
- Convolutions can be applied to other dimensions than 2:
  - 1D: speech, text (e.g., text classification)
  - 3D: the dimensions can be either spacial (e.g., 3D object recognition) or temporal-spacial (e.g., videos)

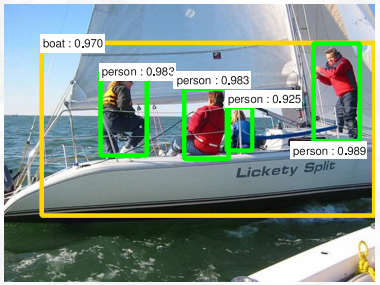
Beyond Image Classification

- Object detection (including location)



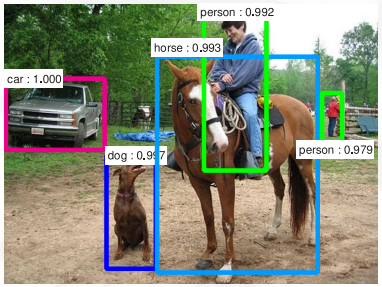


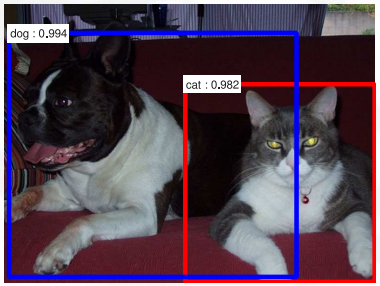


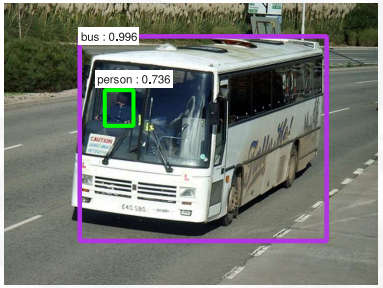


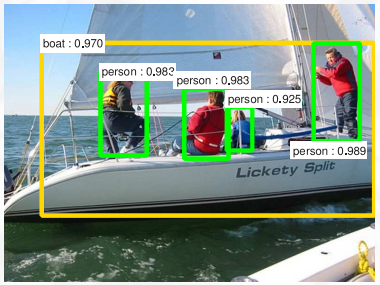
Beyond Image Classification

- Object detection (including location)

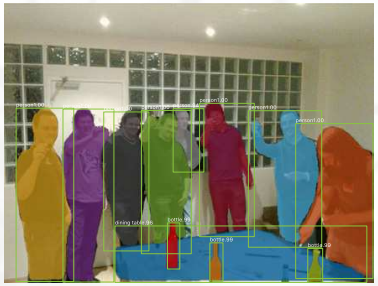
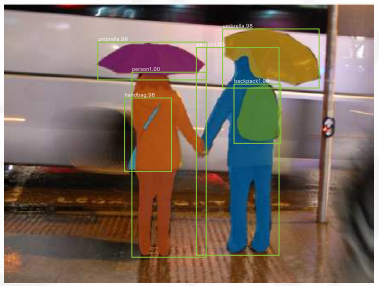
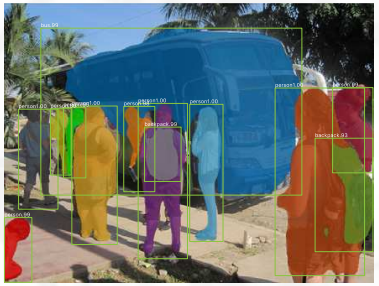
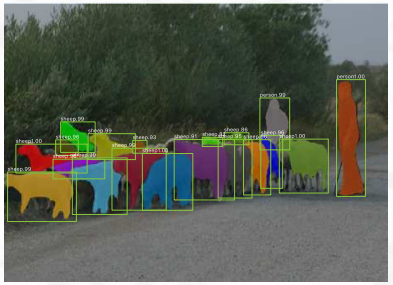






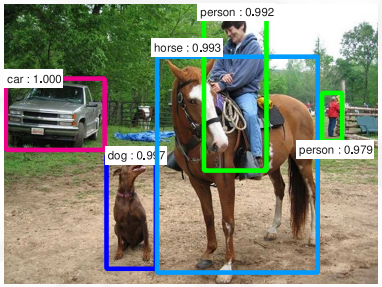


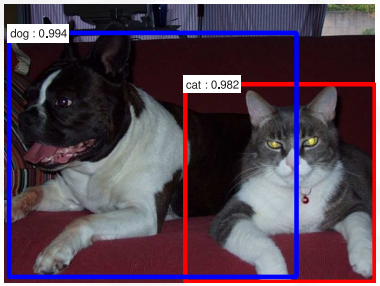
- Image segmentation

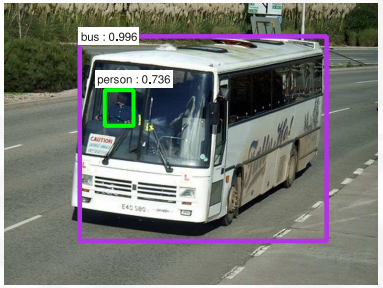





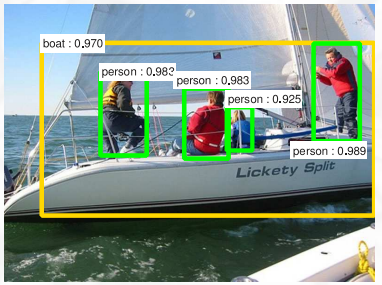
Beyond Image Classification

- Object detection (including location)

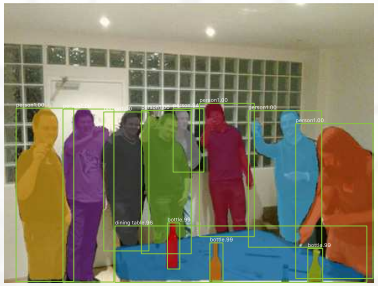
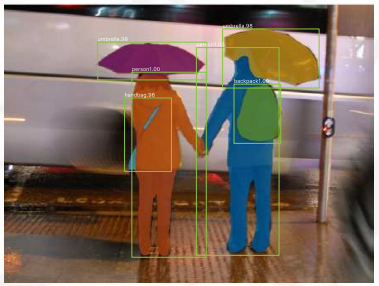
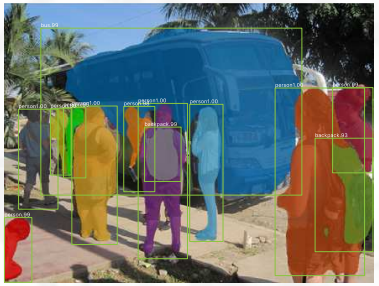
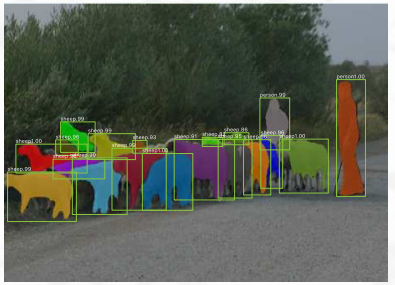




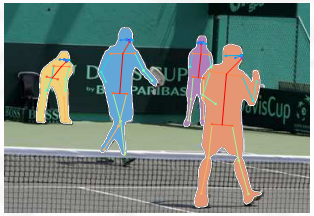
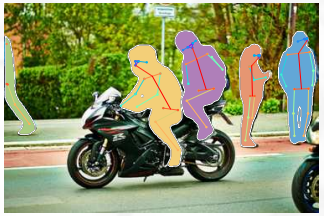




- Image segmentation

- Human pose estimation


Fast R-CNN

- Start with a network pre-trained on ImageNet (VGG-16 is used in the original paper).

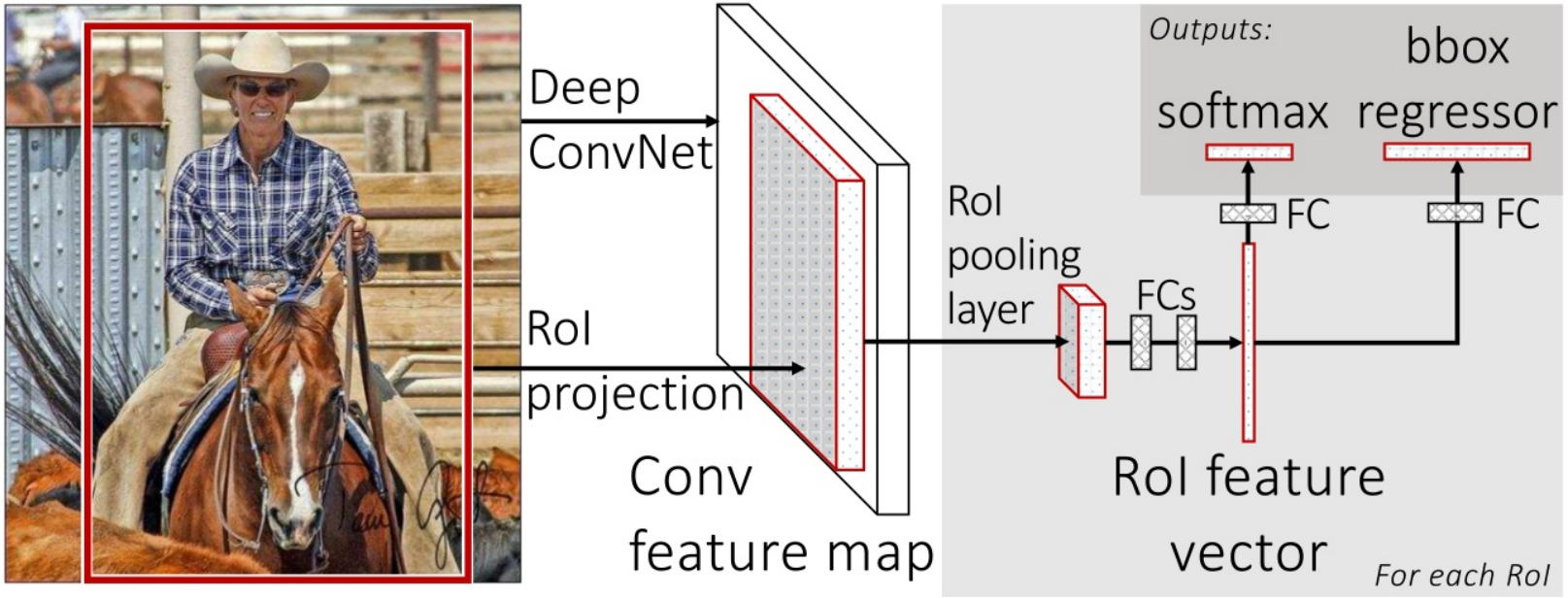
Fast R-CNN

- Start with a network pre-trained on ImageNet (VGG-16 is used in the original paper).

RoI Pooling

- Crucial for fast performance.
- The last max-pool layer ($14 \times 14 \rightarrow 7 \times 7$ in VGG) is replaced by a RoI pooling layer, producing output of the same size. For each output sub-window we max-pool the corresponding values in the output layer.
- Two sibling layers are added, one predicting $K + 1$ categories and the other one predicting 4 bounding box parameters for each of K categories.

Fast R-CNN


Deep
ConvNet
RoI
projection
Conv
feature map
RoI
pooling
layer
FCs
FC
FC
Outputs:
softmax
bbox
regressor
RoI feature
vector
For each RoI

Fast R-CNN

The bounding box is parametrized as follows. Let x_r, y_r, w_r, h_r be center coordinates and width and height of the RoI, and let x, y, w, h be parameters of the bounding box. We represent them as follows:

$$t_x = (x - x_r)/w_r, \quad t_y = (y - y_r)/h_r$$
$$t_w = \log(w/w_r), \quad t_h = \log(h/h_r)$$

Fast R-CNN

The bounding box is parametrized as follows. Let x_r, y_r, w_r, h_r be center coordinates and width and height of the RoI, and let x, y, w, h be parameters of the bounding box. We represent them as follows:

$$t_x = (x - x_r)/w_r, \quad t_y = (y - y_r)/h_r$$
$$t_w = \log(w/w_r), \quad t_h = \log(h/h_r)$$

Usually a smooth_{L_1} loss is employed for bounding box parameters

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

Fast R-CNN

The bounding box is parametrized as follows. Let x_r, y_r, w_r, h_r be center coordinates and width and height of the RoI, and let x, y, w, h be parameters of the bounding box. We represent them as follows:

$$t_x = (x - x_r)/w_r, \quad t_y = (y - y_r)/h_r$$
$$t_w = \log(w/w_r), \quad t_h = \log(h/h_r)$$

Usually a smooth_{L_1} loss is employed for bounding box parameters

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

The complete loss is then

$$L(\hat{c}, \hat{t}, c, t) = L_{\text{cls}}(\hat{c}, c) + \lambda[c \geq 1] \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(\hat{t}_i - t_i).$$

Intersection over union

For two bounding boxes (or two masks) the *intersection over union* (*IoU*) is a ration of the intersection of the boxes (or masks) and the union of the boxes (or masks).

Intersection over union

For two bounding boxes (or two masks) the *intersection over union* (*IoU*) is a ration of the intersection of the boxes (or masks) and the union of the boxes (or masks).

Choosing RoIs for training

During training, we use 2 images with 64 RoIs each. The RoIs are selected so that 25% have intersection over union (IoU) overlap with ground-truth boxes at least 0.5; the others are chosen to have the IoU in range $[0.1, 0.5)$.

Intersection over union

For two bounding boxes (or two masks) the *intersection over union* (*IoU*) is a ration of the intersection of the boxes (or masks) and the union of the boxes (or masks).

Choosing RoIs for training

During training, we use 2 images with 64 RoIs each. The RoIs are selected so that 25% have intersection over union (IoU) overlap with ground-truth boxes at least 0.5; the others are chosen to have the IoU in range $[0.1, 0.5)$.

Choosing RoIs during inference

Single object can be found in multiple RoIs. To choose the most salient one, we perform *non-maximum suppression* -- we ignore RoIs which have an overlap with a higher scoring larger than a given threshold (usually, 0.3 is used).

Object Detection Evaluation

Average Precision

Evaluation is performed using *Average Precision* (*AP*).

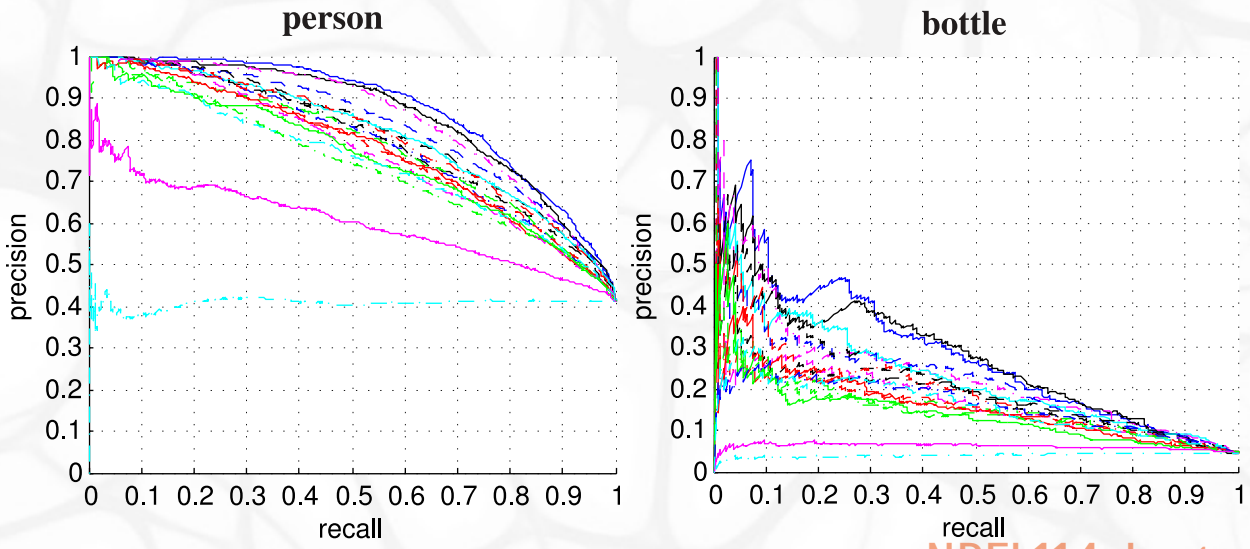
We assume all bounding boxes (or masks) produced by a system have confidence values which can be used to rank them. Then, for a single class, we take the boxes (or masks) in the order of the ranks and generate precision/recall curve, considering a bounding box correct if it has IoU at least 0.5 with any ground-truth box. We define *AP* as an average of precisions for recall levels $0, 0.1, 0.2, \ldots, 1$.

Object Detection Evaluation

Average Precision

Evaluation is performed using *Average Precision* (*AP*).

We assume all bounding boxes (or masks) produced by a system have confidence values which can be used to rank them. Then, for a single class, we take the boxes (or masks) in the order of the ranks and generate precision/recall curve, considering a bounding box correct if it has IoU at least 0.5 with any ground-truth box. We define *AP* as an average of precisions for recall levels $0, 0.1, 0.2, \ldots, 1$.



Faster R-CNN

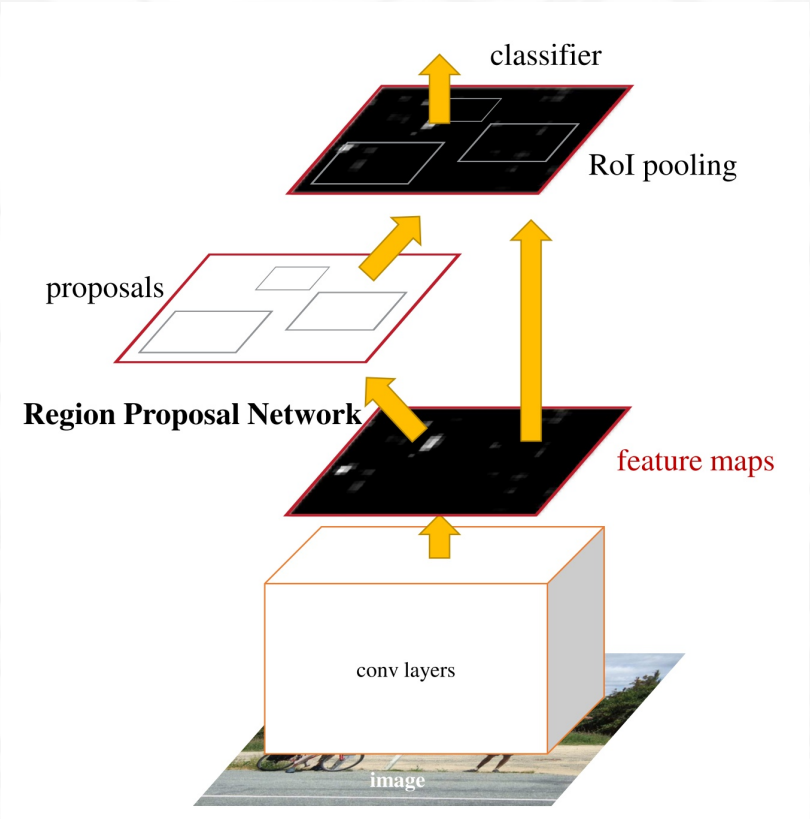
For Fast R-CNN, the most time consuming part is generating the RoIs.

Therefore, Faster R-CNN jointly generates *regions of interest* using a *region proposal network* and performs object detection.

Faster R-CNN

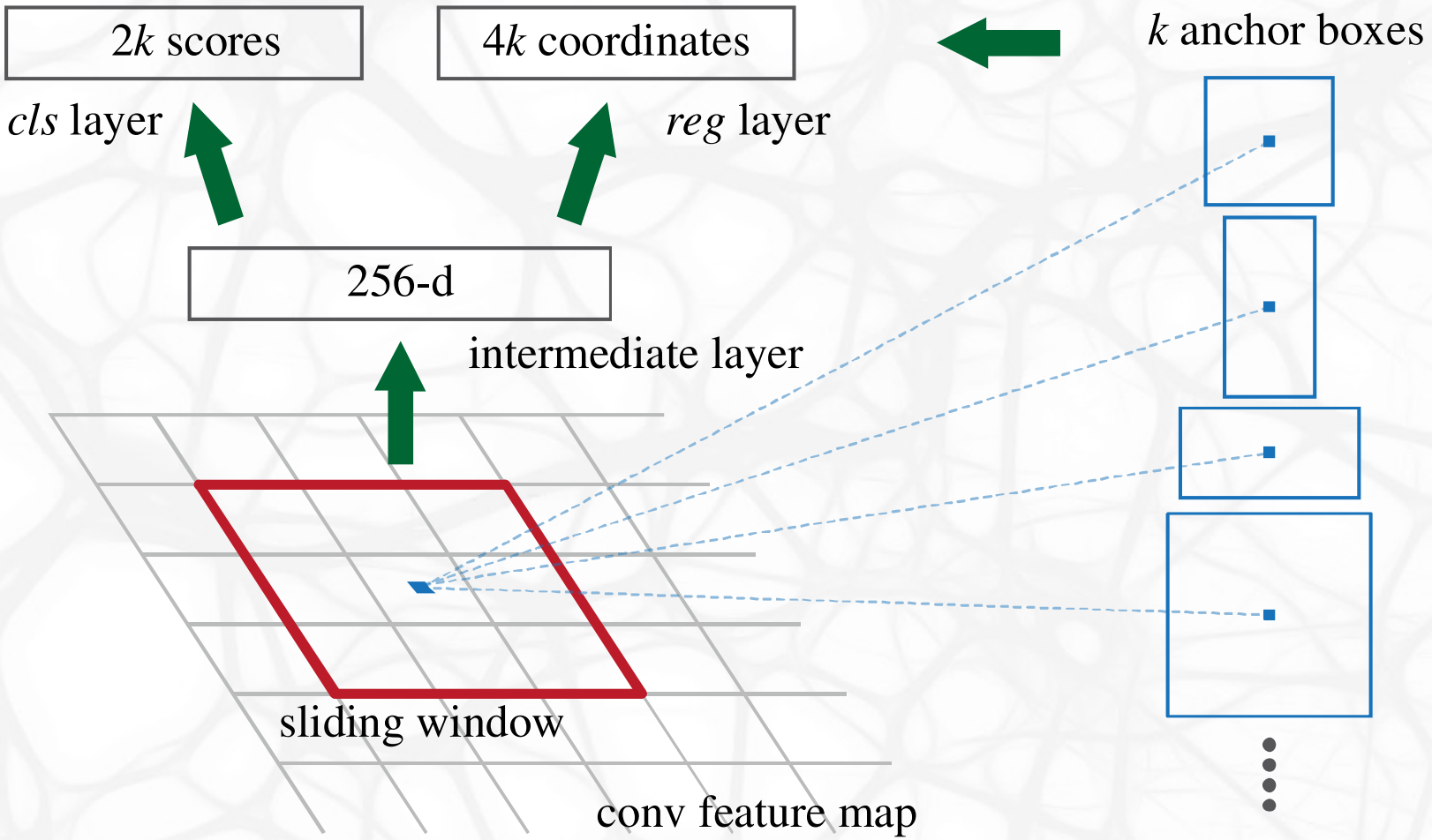
For Fast R-CNN, the most time consuming part is generating the RoIs.

Therefore, Faster R-CNN jointly generates *regions of interest* using a *region proposal network* and performs object detection.



Faster R-CNN

The region proposals are generated using a 3×3 sliding window, with 3 different scales (128^2, 256^2 and 512^2) and 3 aspect ratios ($1:1, 1:2, 2:1$).



Faster R-CNN

Table 3: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: this number was reported in [2]; using the repository provided by this paper, this result is higher (68.1).

method	# proposals	data	mAP (%)
SS	2000	07	66.9†
SS	2000	07+12	70.0
RPN+VGG, unshared	300	07	68.5
RPN+VGG, shared	300	07	69.9
RPN+VGG, shared	300	07+12	**73.2**
RPN+VGG, shared	300	COCO+07+12	**78.8**

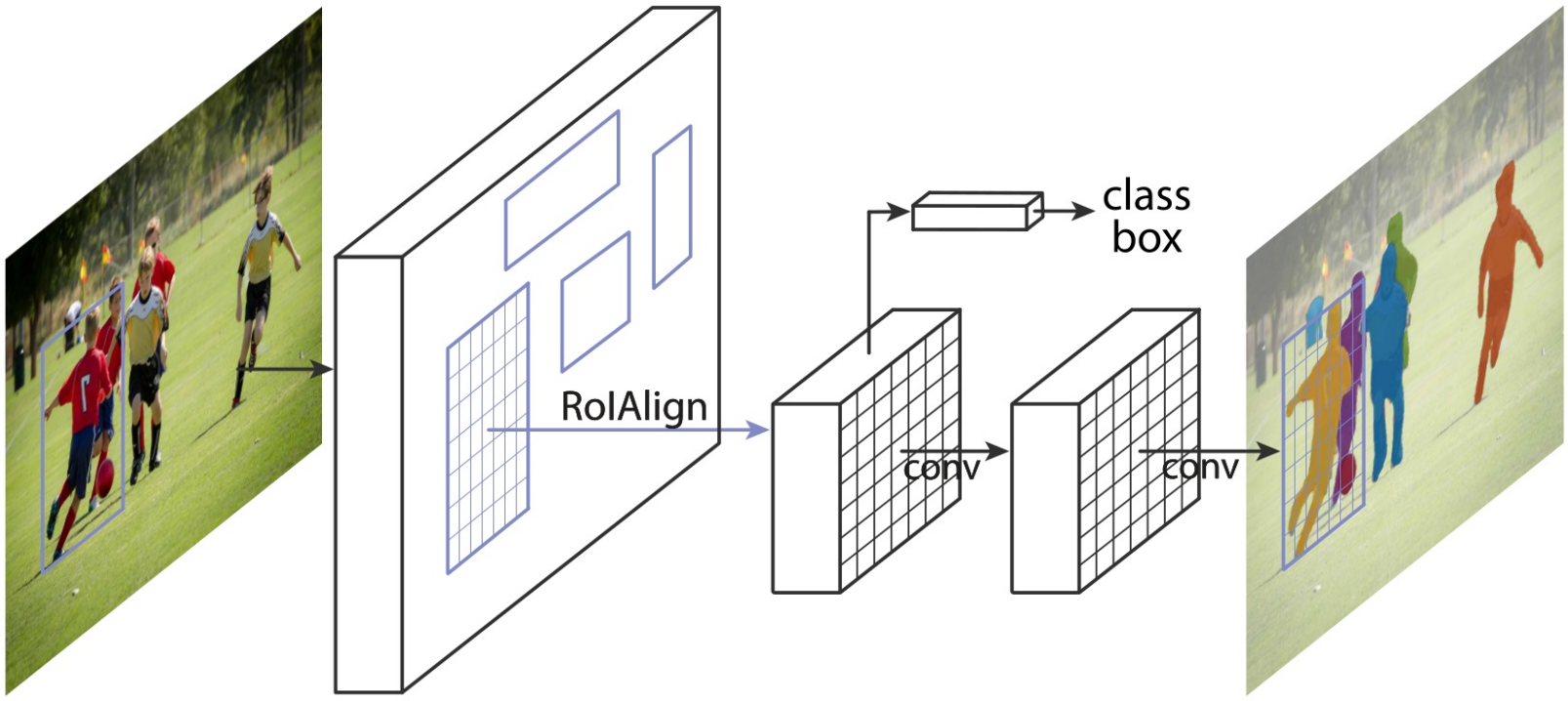
Table 4: Detection results on **PASCAL VOC 2012 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07++12": union set of VOC 2007 trainval+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html. ‡: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html. §: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**
RPN+VGG, shared§	300	COCO+07++12	**75.9**

Mask R-CNN

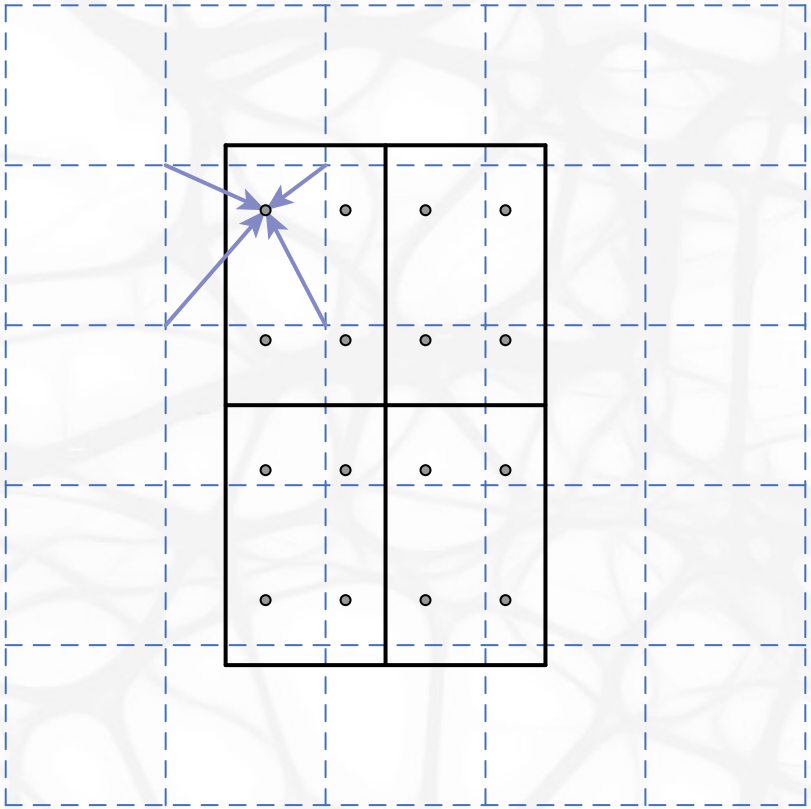
"Straightforward" extension of Faster R-CNN able to produce image segmentation (i.e., masks for every object).

Mask R-CNN



RoIAlign

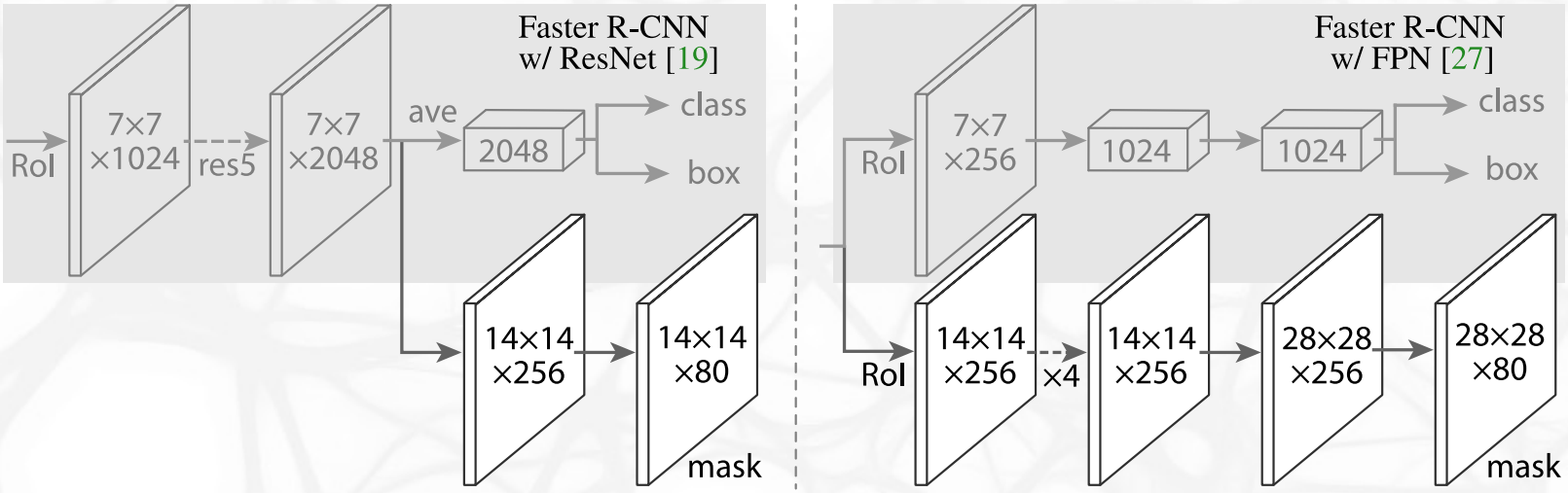
More precise alignment if required for the RoI in order to predict the masks. Therefore, instead of max-pooling used in the RoI pooling, RoIAlign with bilinear interpolation is used instead.



Mask R-CNN

Masks are predicted in a third branch of the object detector.

- Usually higher resolution is needed (14×14 instead of 7×7).
- The masks are predicted for each class separately.
- The masks are predicted using convolutions instead of fully connected layers.



Mask R-CNN

net-depth-features	AP	AP_{50}	AP_{75}
ResNet-50-C4	30.3	51.2	31.5
ResNet-101-C4	32.7	54.2	34.3
ResNet-50-FPN	33.6	55.2	35.3
ResNet-101-FPN	35.4	57.3	37.5
ResNeXt-101-FPN	**36.7**	**59.5**	**38.9**

(a) **Backbone Architecture**: Better backbones bring expected gains: deeper networks do better, FPN outperforms C4 features, and ResNeXt improves on ResNet.

	AP	AP_{50}	AP_{75}
softmax	24.8	44.1	25.1
sigmoid	**30.3**	**51.2**	**31.5**
	+5.5	*+7.1*	*+6.4*

(b) **Multinomial *vs.* Independent Masks** (ResNet-50-C4): *Decoupling* via per-class binary masks (sigmoid) gives large gains over multinomial masks (softmax).

	align?	bilinear?	agg.	AP	AP_{50}	AP_{75}
RoIPool [12]			max	26.9	48.8	26.4
RoIWarp [10]		✓	max	27.2	49.2	27.1
		✓	ave	27.1	48.9	27.1
RoIAlign	✓	✓	max	**30.2**	**51.0**	**31.8**
	✓	✓	ave	**30.3**	**51.2**	**31.5**

(c) **RoIAlign** (ResNet-50-C4): Mask results with various RoI layers. Our RoIAlign layer improves AP by ~3 points and AP_{75} by ~5 points. Using proper alignment is the only factor that contributes to the large gap between RoI layers.

	AP	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}
RoIPool	23.6	46.5	21.6	28.2	52.7	26.9
RoIAlign	**30.9**	**51.8**	**32.1**	**34.0**	**55.3**	**36.4**
	+7.3	*+ 5.3*	*+10.5*	*+5.8*	*+2.6*	*+9.5*

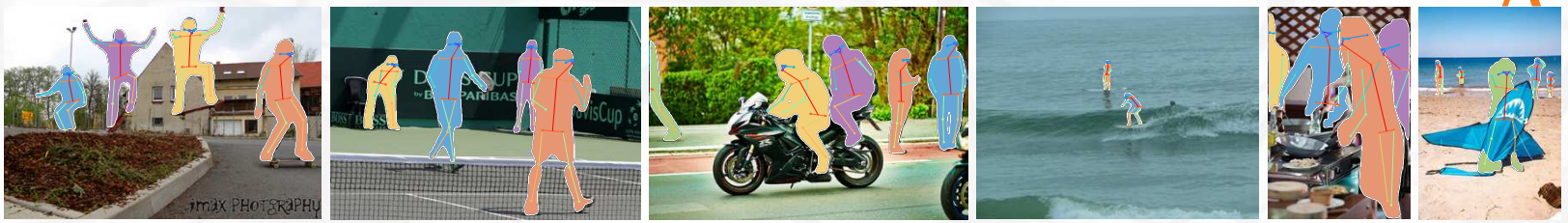
(d) **RoIAlign** (ResNet-50-**C5**, *stride 32*): Mask-level and box-level AP using *large-stride* features. Misalignments are more severe than with stride-16 features (Table 2c), resulting in big accuracy gaps.

	mask branch	AP	AP_{50}	AP_{75}
MLP	fc: 1024→1024→80·28^2	31.5	53.7	32.8
MLP	fc: 1024→1024→1024→80·28^2	31.5	54.0	32.6
FCN	conv: 256→256→256→256→256→80	**33.6**	**55.2**	**35.3**

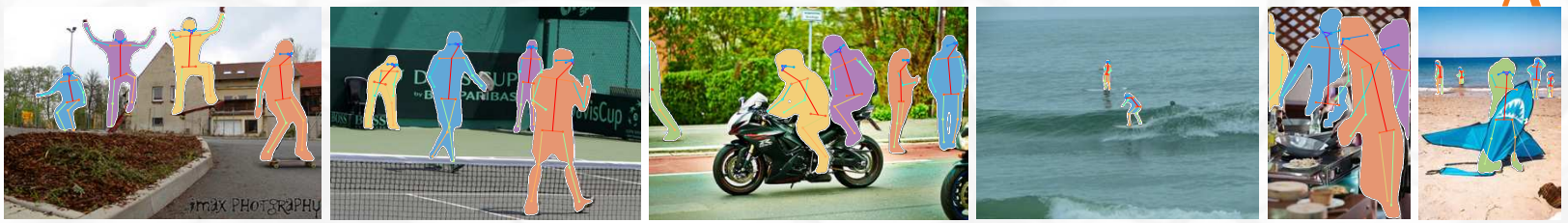
(e) **Mask Branch** (ResNet-50-FPN): Fully convolutional networks (FCN) *vs.* multi-layer perceptrons (MLP, fully-connected) for mask prediction. FCNs improve results as they take advantage of explicitly encoding spatial layout.

Table 2. **Ablations**. We train on `trainval35k`, test on `minival`, and report *mask* AP unless otherwise noted.

Mask R-CNN – Human Pose Estimation

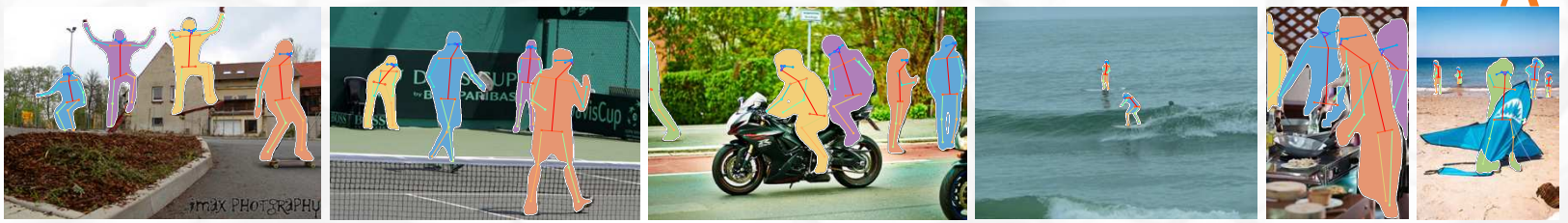


Mask R-CNN – Human Pose Estimation



- Testing applicability of Mask R-CNN architecture.
- Keypoints (e.g., left shoulder, right elbow, …) are detected as independent one-hot masks of size 56×56 with $\mathrm{softmax}$ output function.

Mask R-CNN – Human Pose Estimation



- Testing applicability of Mask R-CNN architecture.
- Keypoints (e.g., left shoulder, right elbow, …) are detected as independent one-hot masks of size 56×56 with $\mathrm{softmax}$ output function.

	AP^{kp}	AP^{kp}_{50}	AP^{kp}_{75}	AP^{kp}_{M}	AP^{kp}_{L}
CMU-Pose+++ [6]	61.8	84.9	67.5	57.1	68.2
G-RMI [32]†	62.4	84.0	68.5	**59.1**	68.1
Mask R-CNN, keypoint-only	62.7	87.0	68.4	57.4	71.1
Mask R-CNN, keypoint & mask	**63.1**	**87.3**	**68.7**	57.8	**71.4**