Convolutional Neural Networks

Milan Straka

March 11, 2025

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research, Development and Education

Going Deeper

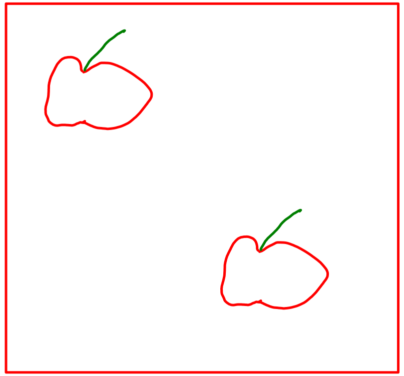
Convolution Operation

Convolution Operation

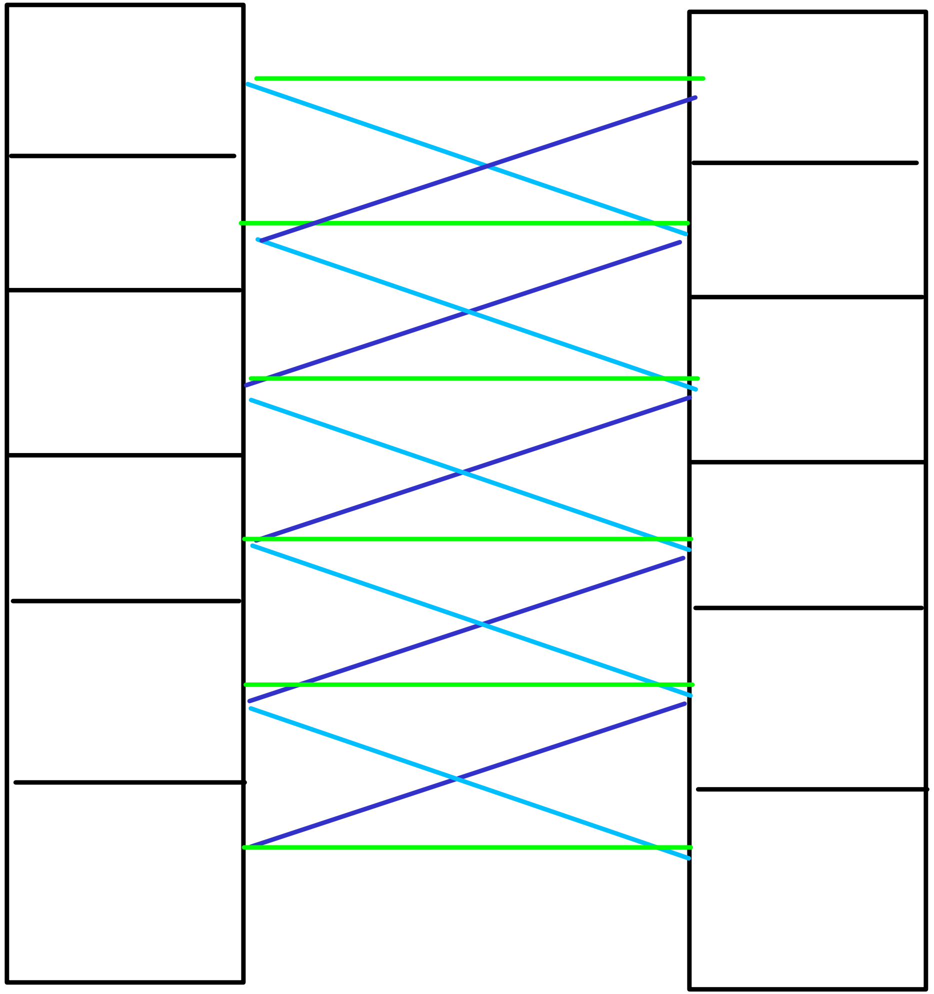
Consider data with some structure (temporal data, speech, images, …).

Unlike densely connected layers, we might want:

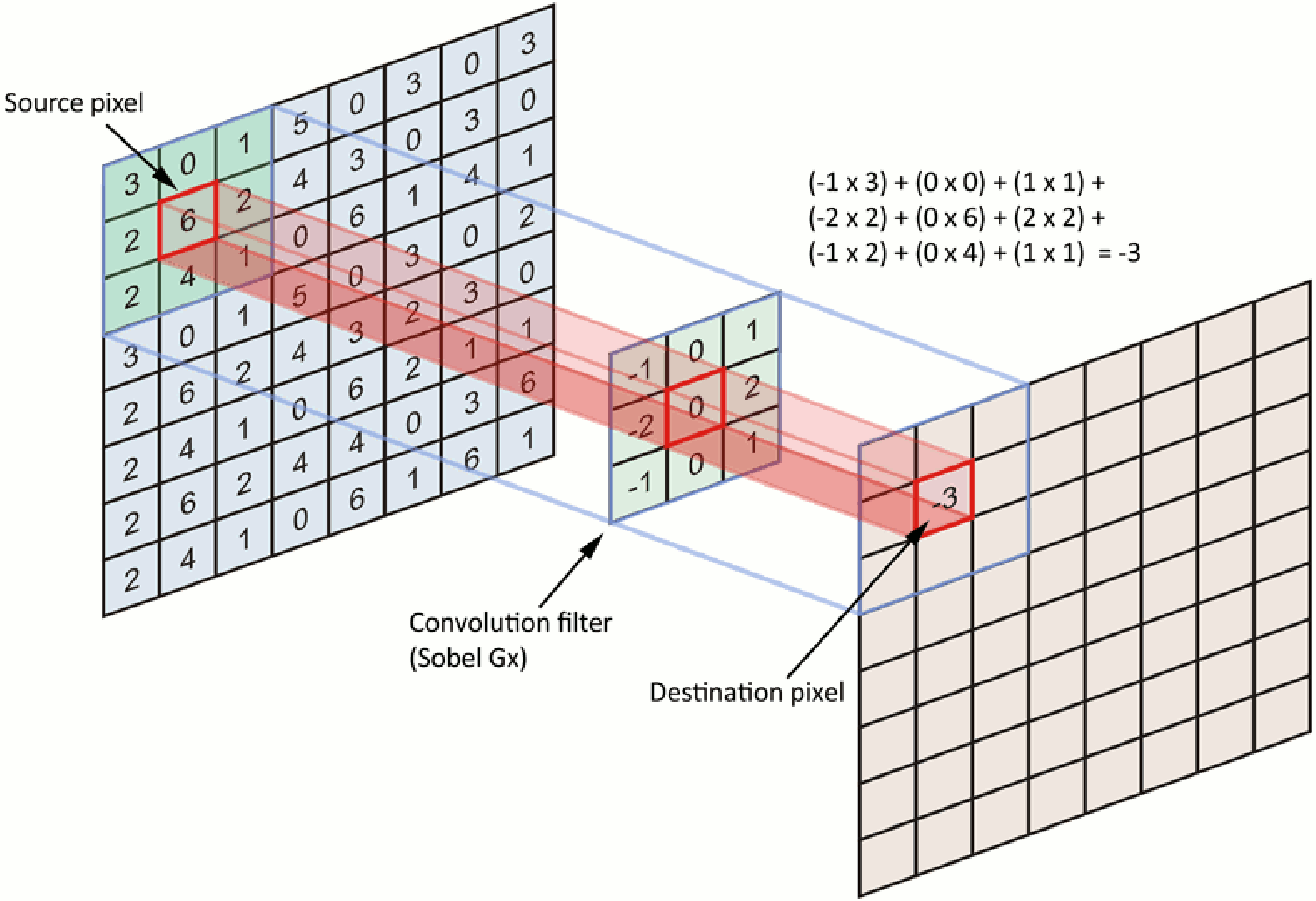
- local interactions only;
- shift invariance (equal response everywhere).



1D Convolution



2D Convolution



https://i.stack.imgur.com/YDusp.png

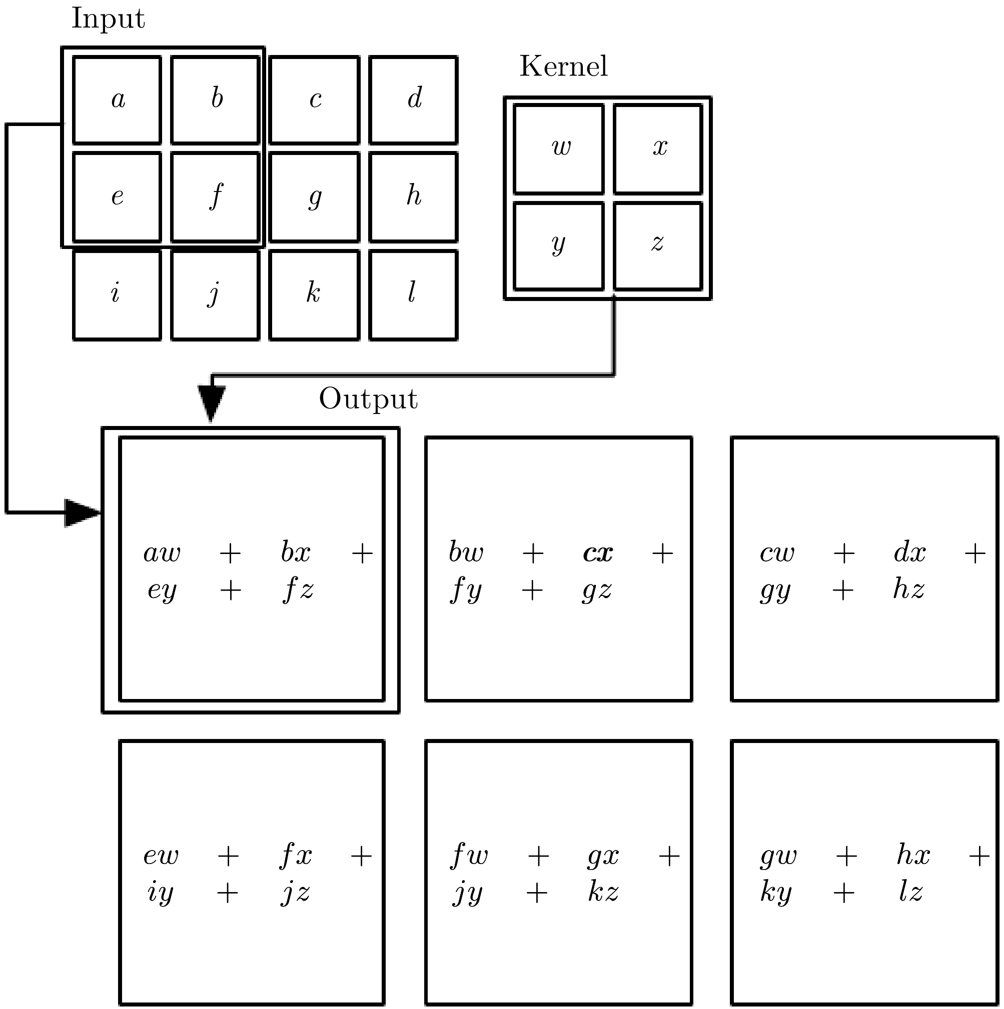
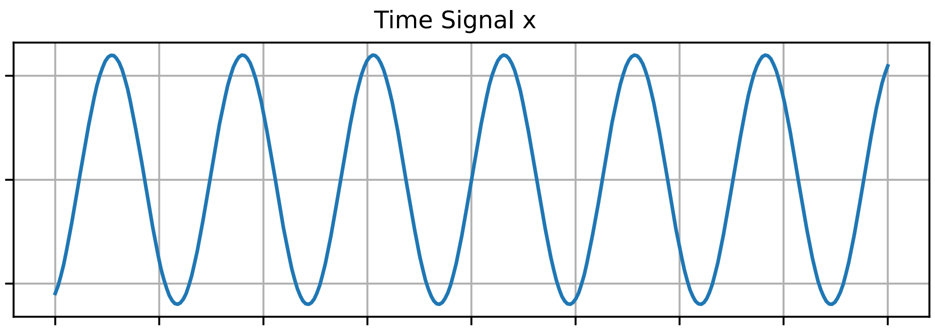


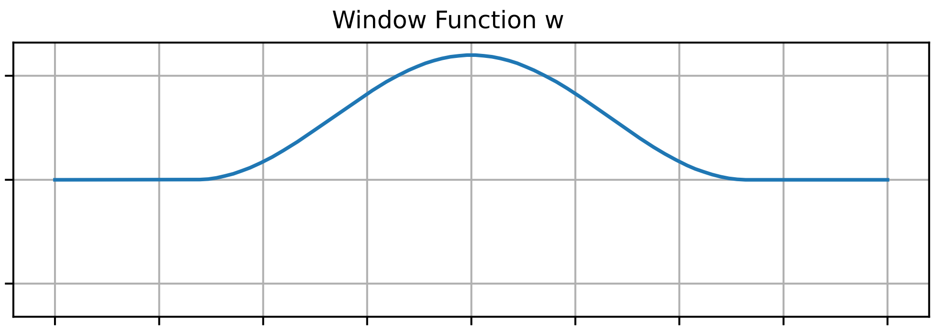
Figure 9.1 of "Deep Learning" book, https://www.deeplearningbook.org

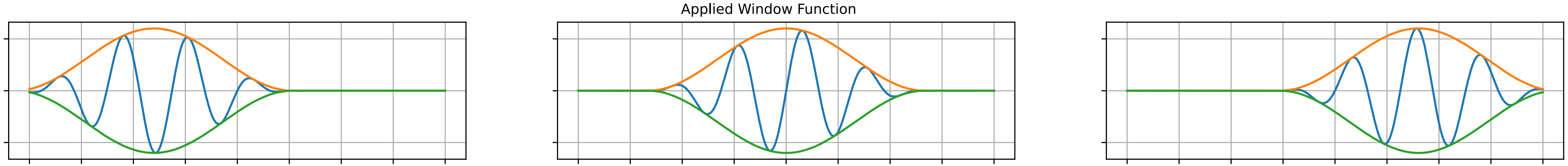
Convolution Operation

For functions x and w, *convolution* $w * x$ is defined as

$$(w * x)(t) = \int x(t-a)w(a)\,\mathrm{d}a.$$



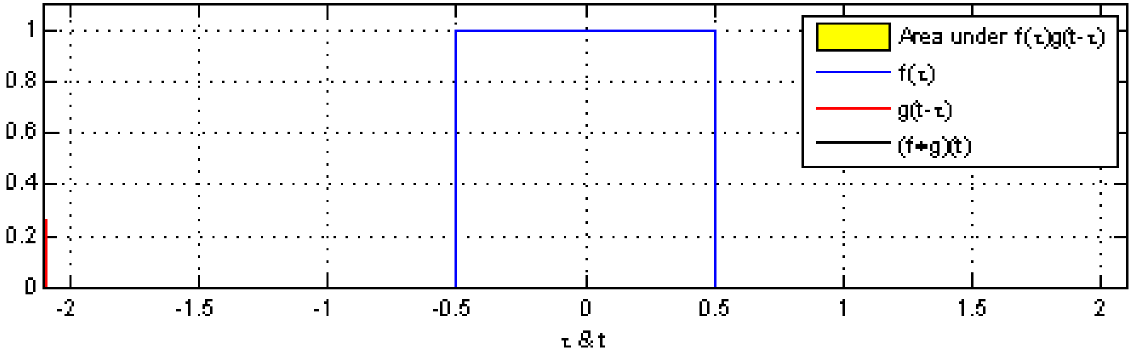




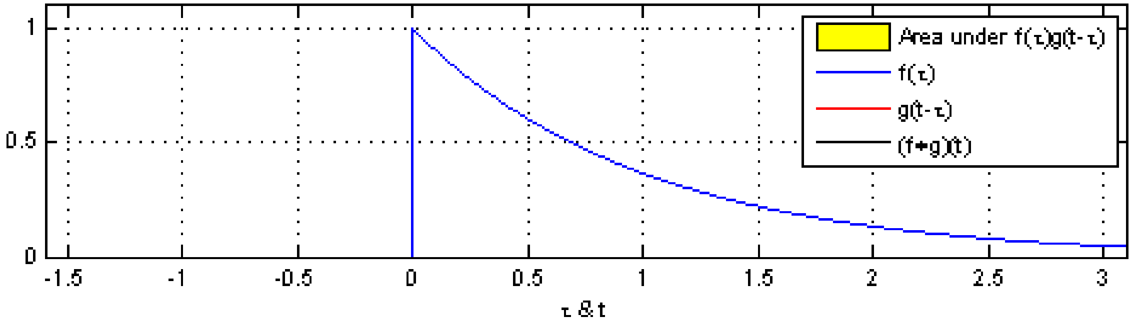
Convolution Operation

For functions x and w, *convolution* $w * x$ is defined as

$$(w * x)(t) = \int x(t - a) w(a) \, \mathrm{d}a.$$



https://commons.wikimedia.org/wiki/File:Convolution_of_box_signal_with_itself2.gif



https://commons.wikimedia.org/wiki/File:Convolution_of_spiky_function_with_box2.gif

Convolution Operation

For functions x and w, *convolution* $w * x$ is defined as

$$(w * x)(t) = \int x(t-a)w(a)\,\mathrm{d}a.$$

For vectors, we have

$$(\boldsymbol{w} * \boldsymbol{x})_t = \sum\nolimits_i x_{t-i} w_i.$$

Convolution operation can be generalized to two dimensions by

$$(\boldsymbol{K} * \boldsymbol{I})_{i,j} = \sum\nolimits_{m,n} \boldsymbol{I}_{i-m,j-n} \boldsymbol{K}_{m,n}.$$

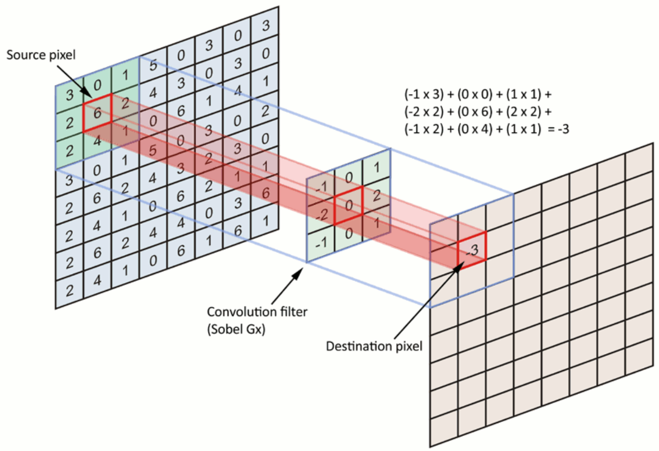
Closely related is *cross-correlation*, where K is flipped:

$$(\boldsymbol{K} \star \boldsymbol{I})_{i,j} = \sum\nolimits_{m,n} \boldsymbol{I}_{i+m,j+n} \boldsymbol{K}_{m,n}.$$

The K is usually called a **kernel** or a **filter**.

Note that usually we have a whole vector of values for a single pixel, the so-called **channels**. These single pixel channel values have no longer any spatial structure, so the kernel contains a different set of weights for every input dimension, obtaining

$$(\mathsf{K} \star \mathsf{I})_{i,j} = \sum_{m,n,c} \mathsf{I}_{i+m,j+n,c} \mathsf{K}_{m,n,c}.$$



https://i.stack.imgur.com/YDusp.png

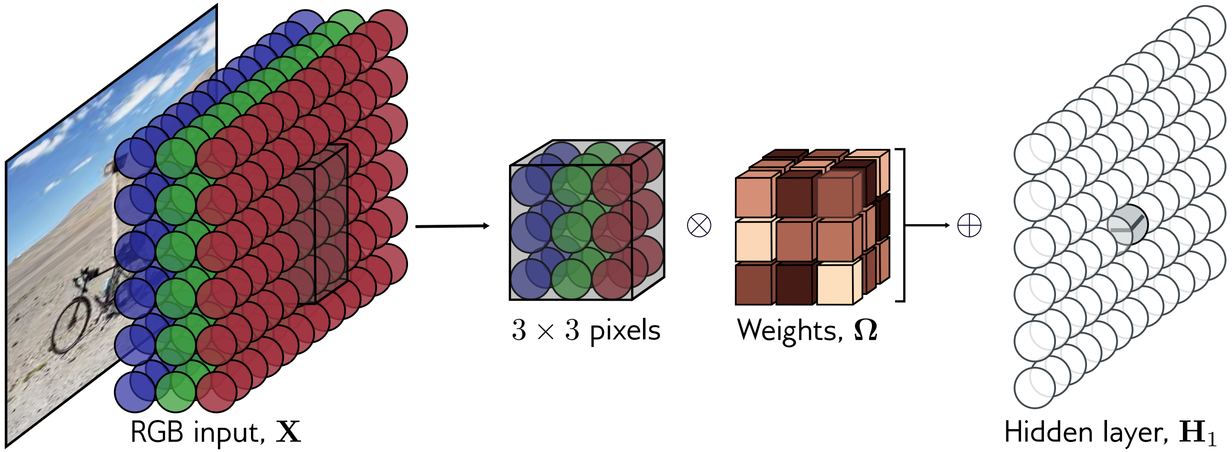
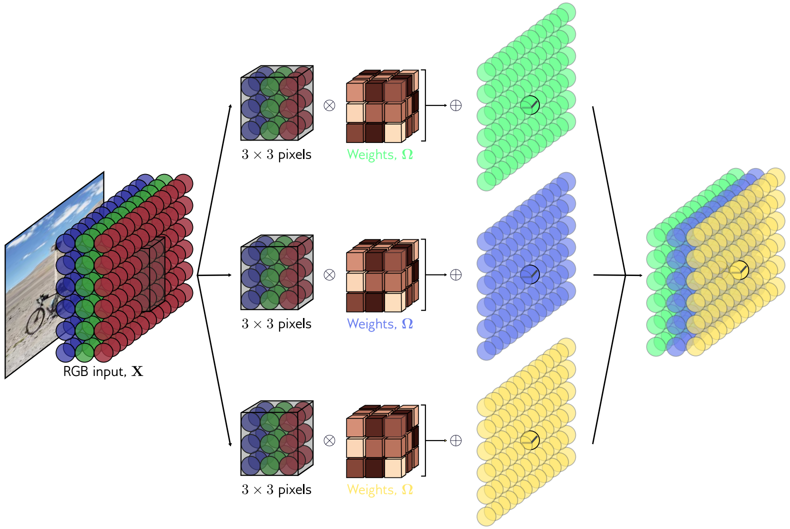


Figure 10.10 of "Understanding Deep Learning", https://udlbook.github.io/udlbook/

Furthermore, we usually want to be able to specify the output dimensionality similarly to for example a fully connected layer – the number of **output channels** for every pixel. Each output channel is then the output of an independent convolution operation, so we can consider K to be a four-dimensional tensor and the convolution if computed as

$$(\mathsf{K} \star \mathsf{I})_{i,j,o} = \sum_{m,n,c} \mathsf{I}_{i+m,j+n,c} \mathsf{K}_{m,n,c,o}.$$



Modification of Figure 10.10 of "Understanding Deep Learning", https://udlbook.github.io/udlbook/

Convolutional Neural Networks

Convolution Layer

To define a complete convolution layer, we need to specify:

- the width W and height H of the kernel;
- the number of output channels F;
- the **stride** denoting that every output pixel is computed for every **stride**-th input pixel (e.g., the output is half the size if stride is 2).

Considering an input image with C channels, the convolution layer is then parametrized by a kernel K of total size $W \times H \times C \times F$ and is computed as

$$(\mathsf{K} \star \mathsf{I})_{i,j,o} = \sum_{m,n,c} \mathsf{I}_{i \cdot S+m, j \cdot S+n, c} \mathsf{K}_{m,n,c,o}.$$

Note that while only local interactions are performed in the image spatial dimensions (width and height), we combine input channels in a fully connected manner.

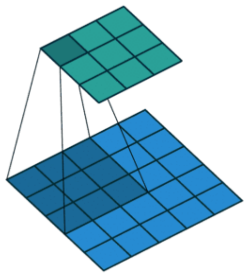
Convolution Layer

There are multiple padding schemes, most common are:

- `valid`: Only use valid pixels, which causes the result to be smaller than the input.
- `same`: Pad original image with zero pixels so that the result is exactly the size of the input.

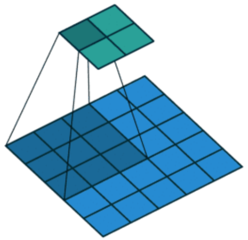
Illustration of the padding schemes and different strides for a 3×3 kernel:

- **valid** padding, stride=1:



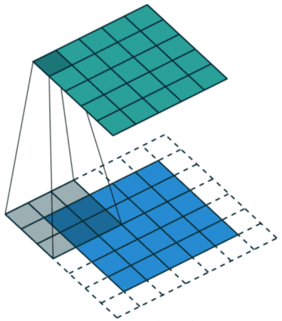
https://github.com/vdumoulin/conv_arithmetic

stride=2:



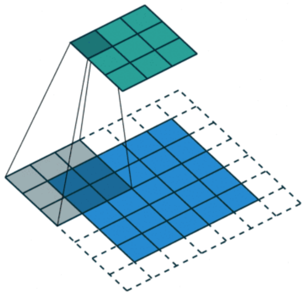
https://github.com/vdumoulin/conv_arithmetic

- **same** padding, stride=1:



https://github.com/vdumoulin/conv_arithmetic

stride=2:



https://github.com/vdumoulin/conv_arithmetic

Convolution Layer Representation

There are two common image formats:

- `channels_last`: The dimensions of a 3-dimensional image tensor are height, width, and channels; also called `HWC` for a single image and `NHWC` for a batch of images.
  - The format used by PIL, scipy, TensorFlow, JAX, Keras, …, faster on CPU.
- `channels_first`: The dimensions of a 3-dimensional image tensor are channel, height, and width; also called `CHW` for a single image and `NCHW` for a batch of images.
  - Used by PyTorch, originally faster on GPUs; `channels_last` is faster on new GPUs.

In PyTorch, the image shape is always `channels_first`, so `[N, C, H, W]` for a batch; however, you can choose a `memory_format` using `tensor.to(memory_format=...)`, where

- `memory_format=torch.contiguous_format` is dense non-overlapping `NCHW`, the default;
- `memory_format=torch.channels_last` is dense non-overlapping `NHWC`;
- `memory_format=torch.channels_last_3d` is dense non-overlapping `NDHWC`.

In TensorFlow, image shape is `channels_last` with runtime choosing the faster format.

In PyTorch, my recommendation is to always use **torch.channels_last** memory format.

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

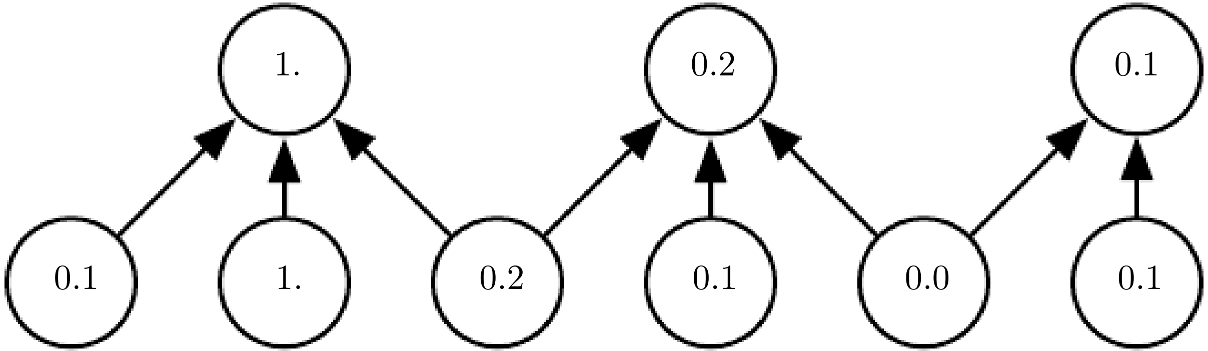
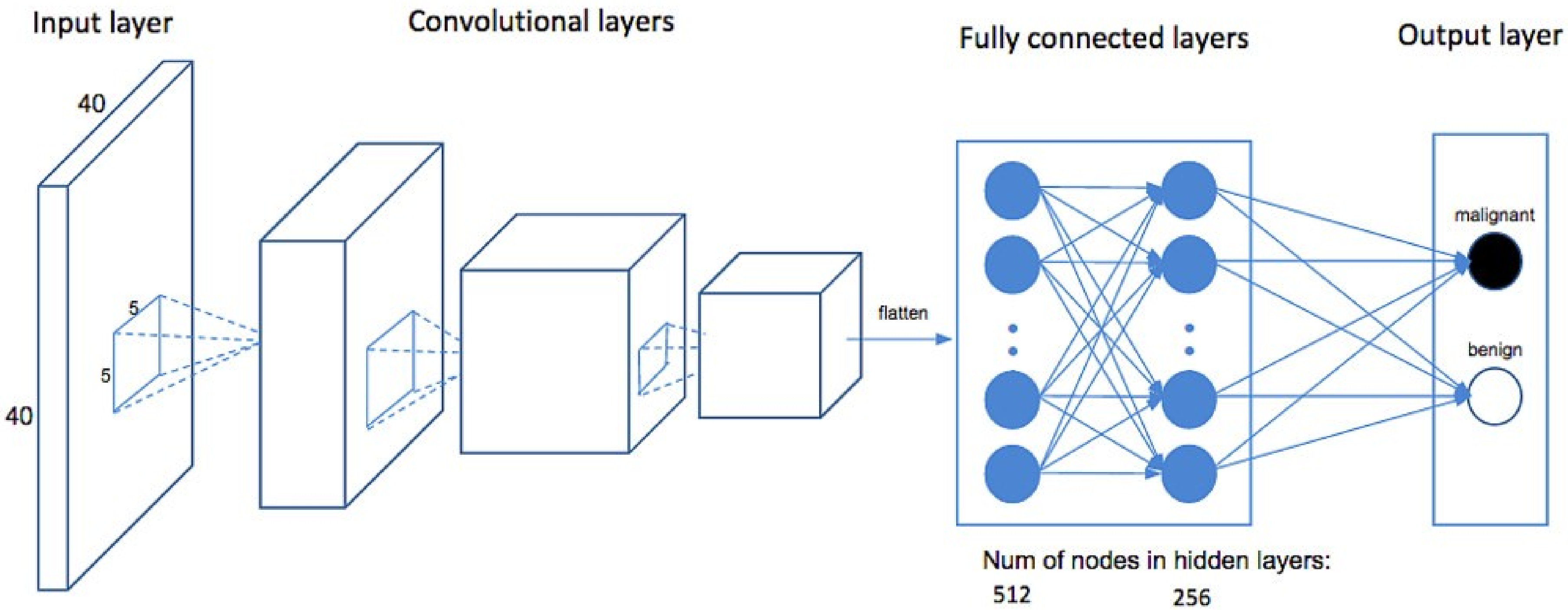


Figure 9.10 of "Deep Learning" book, https://www.deeplearningbook.org

High-level CNN Architecture

We repeatedly use the following block:

1. Convolution operation
2. Non-linear activation (usually ReLU)
3. Pooling



*https://cdn-images-1.medium.com/max/1200/0*QyXSpqpm1wc_Dt6V.*

AlexNet

AlexNet – 2012 (16.4% ILSVRC top-5 error)

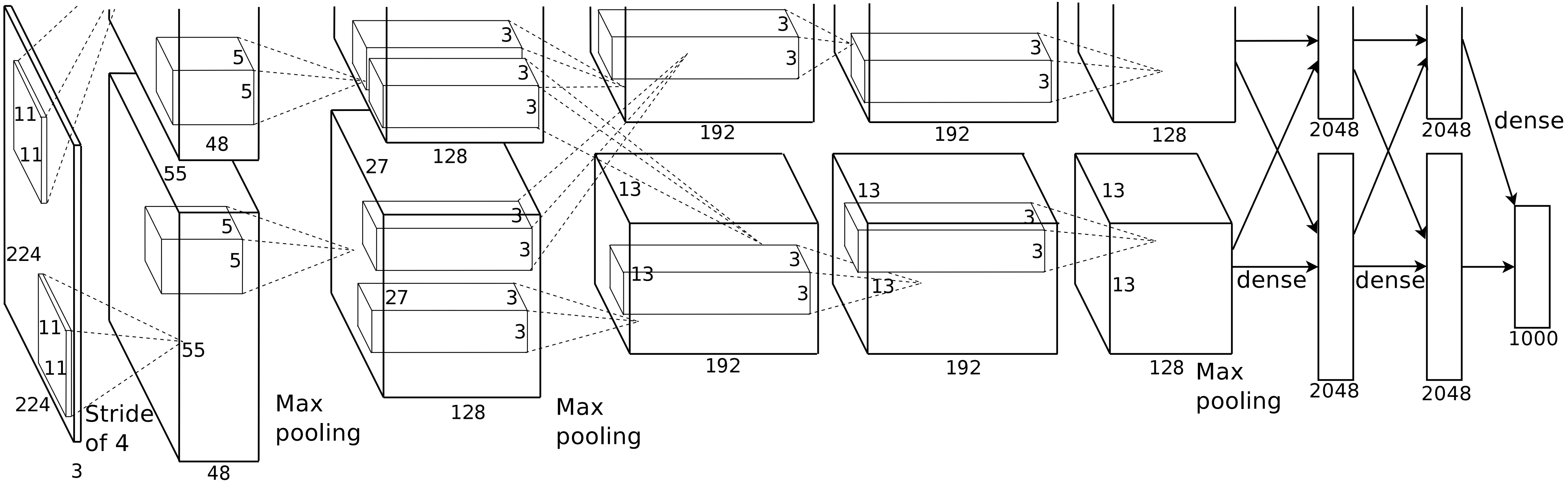


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Figure 2 of "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky et al.

AlexNet – 2012 (16.4% ILSVRC top-5 error)

Training details:

- 61M parameters, 2 GPUs for 5-6 days
- SGD with batch size 128, momentum 0.9, L^2 regularization strength (weight decay) 0.0005
  - $\boldsymbol{v} \leftarrow 0.9 \cdot \boldsymbol{v} - \alpha \cdot \frac{\partial L}{\partial \boldsymbol{\theta}} - 0.0005 \cdot \alpha \cdot \boldsymbol{\theta}$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
- initial learning rate 0.01, manually divided by 10 when validation error rate stopped improving
- ReLU nonlinearities
- dropout with rate 0.5 on the fully-connected layers (except for the output layer)
- data augmentation using translations and horizontal reflections (choosing random 224×224 patches from 256×256 images)
  - during inference, 10 patches are used (four corner patches and a center patch, as well as their reflections)

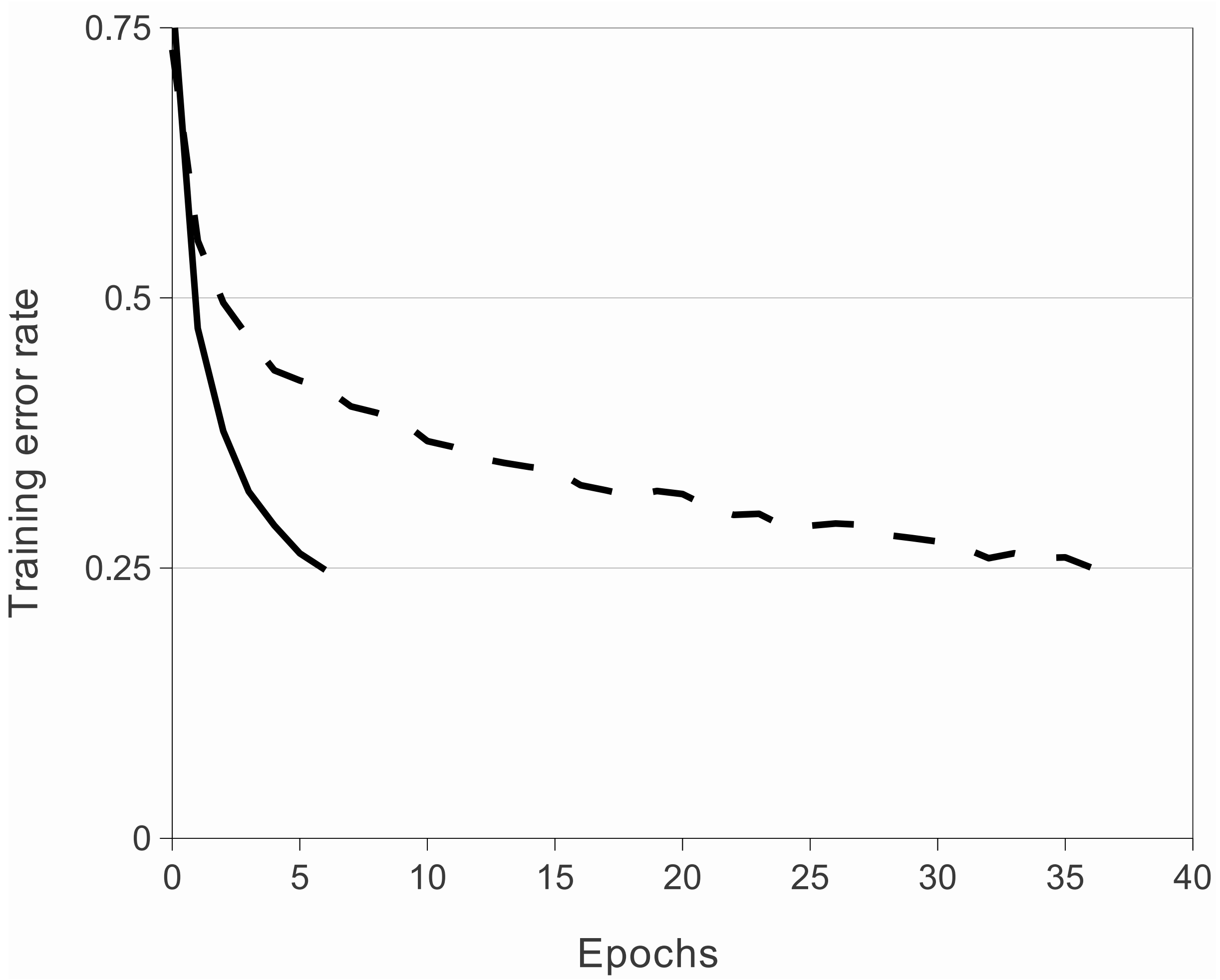


Figure 1 of "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky et al.

AlexNet built on already existing CNN architectures, mostly on LeNet, which achieved 0.8% test error on MNIST.

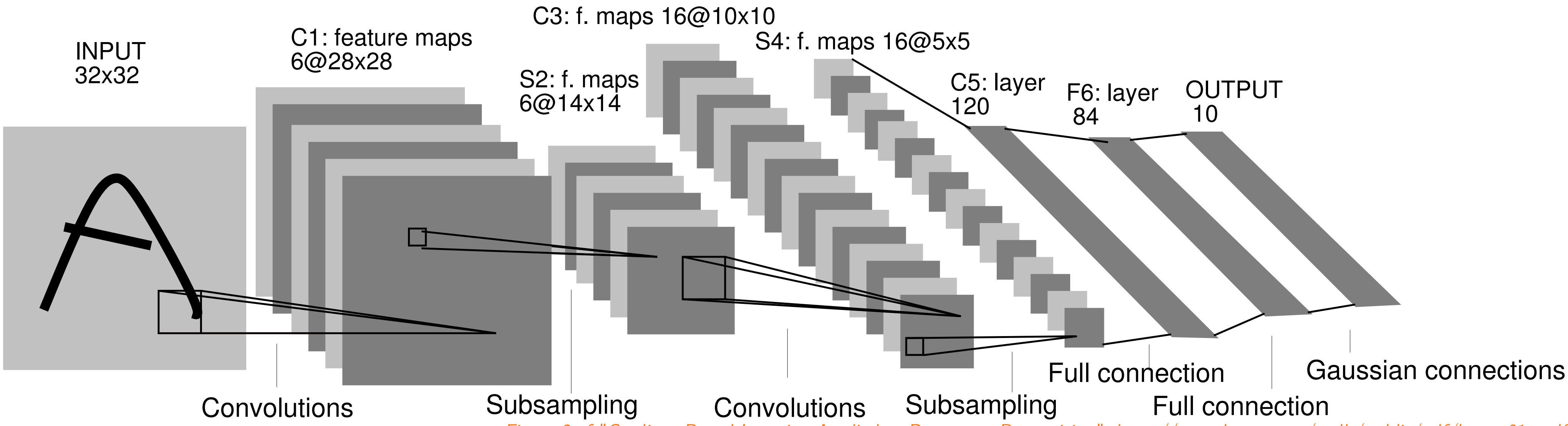


Figure 2 of "Gradient-Based Learning Applied to Document Recognition", http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

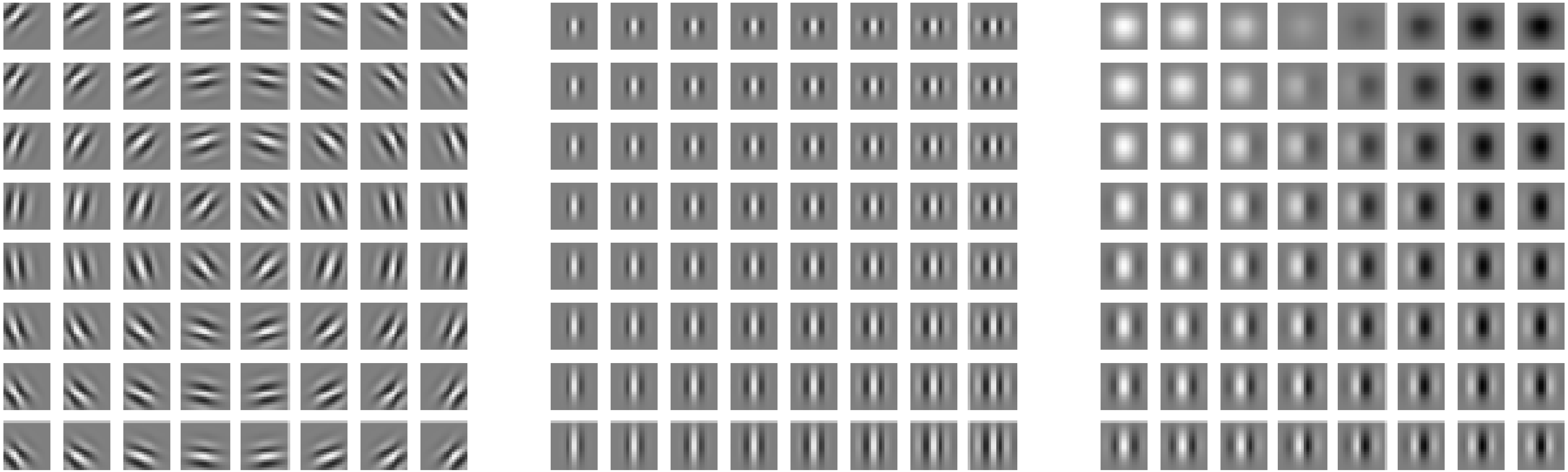


Figure 9.18 of "Deep Learning" book, https://www.deeplearningbook.org

The primary visual cortex recognizes Gabor functions.

Similarities in Primary Visual Cortex (V1) and CNNs

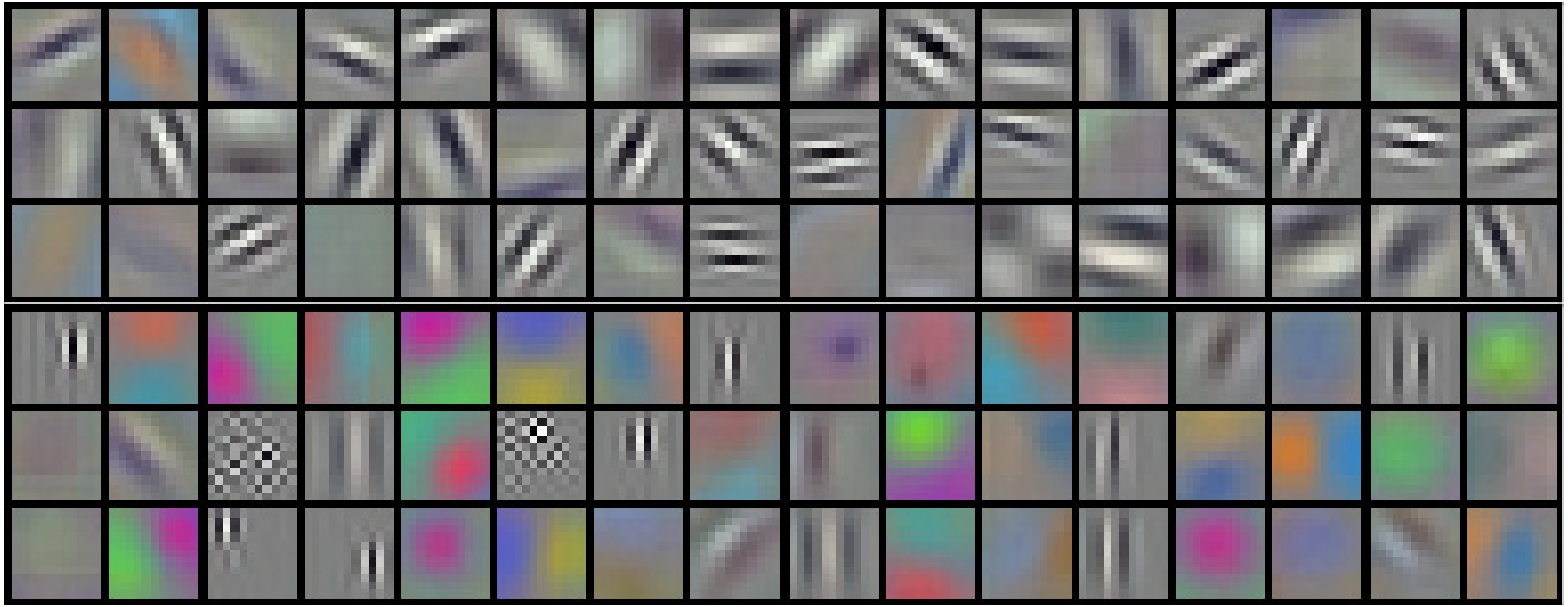


Figure 3 of "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky et al.

The 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer of AlexNet on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2.

CNNs as Regularizers: Deep Prior

CNNs as Regularizers: Deep Prior

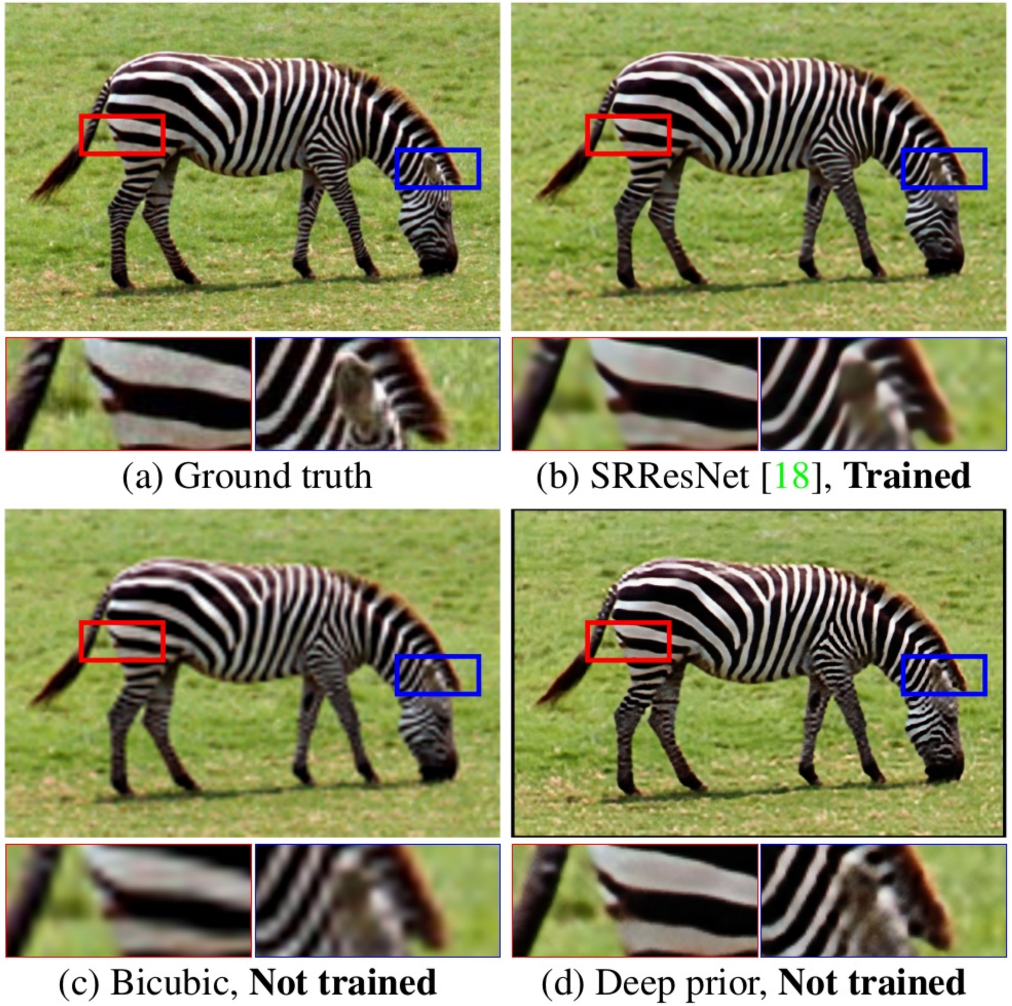
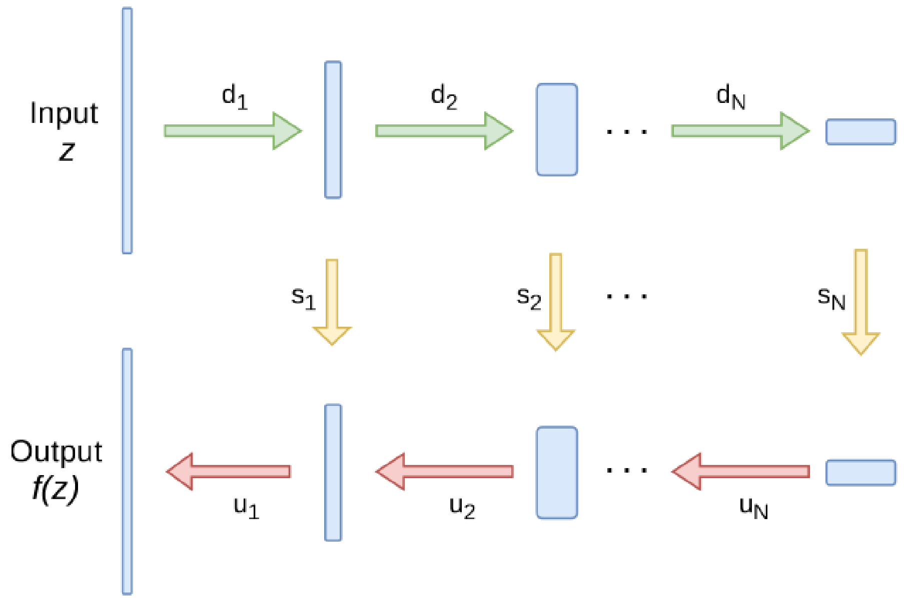


Figure 1 of "Deep Image Prior", https://arxiv.org/abs/1711.10925



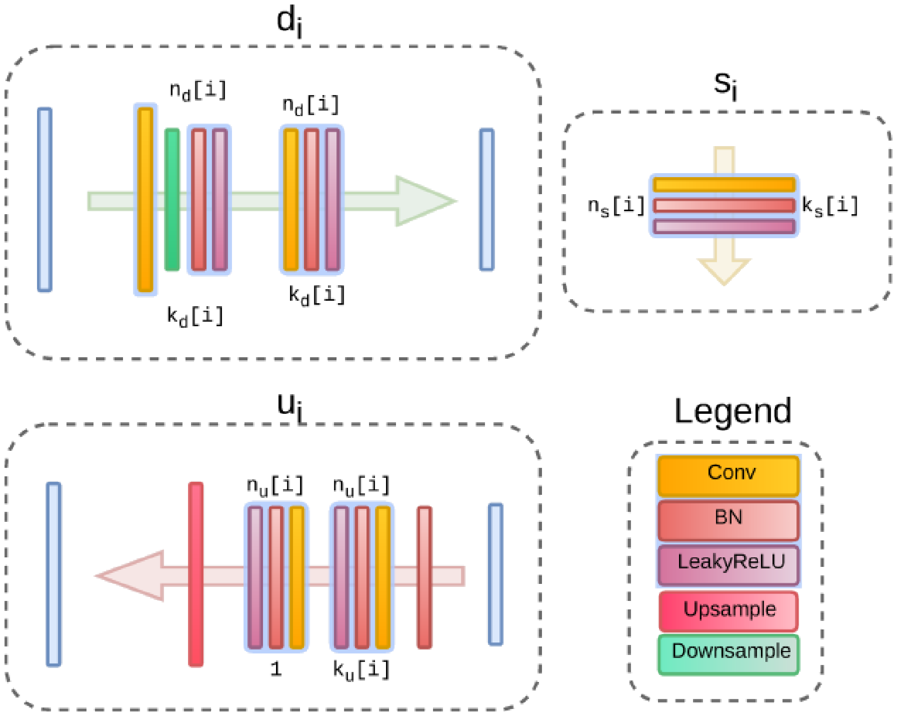


Figure 1 of "Deep Image Prior" supplementary materials, https://arxiv.org/abs/1711.10925

Random noise from $U[0, \frac{1}{10}]$ used on input; in large inpainting, meshgrid is used instead and the skip-connections are not used.

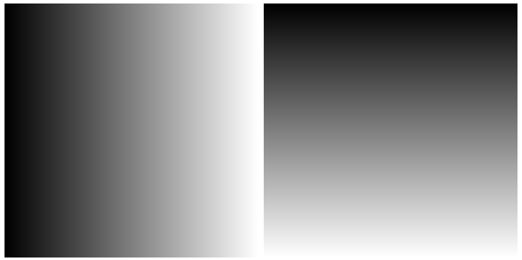


Figure 2 of "Deep Image Prior" supplementary materials, https://arxiv.org/abs/1711.10925

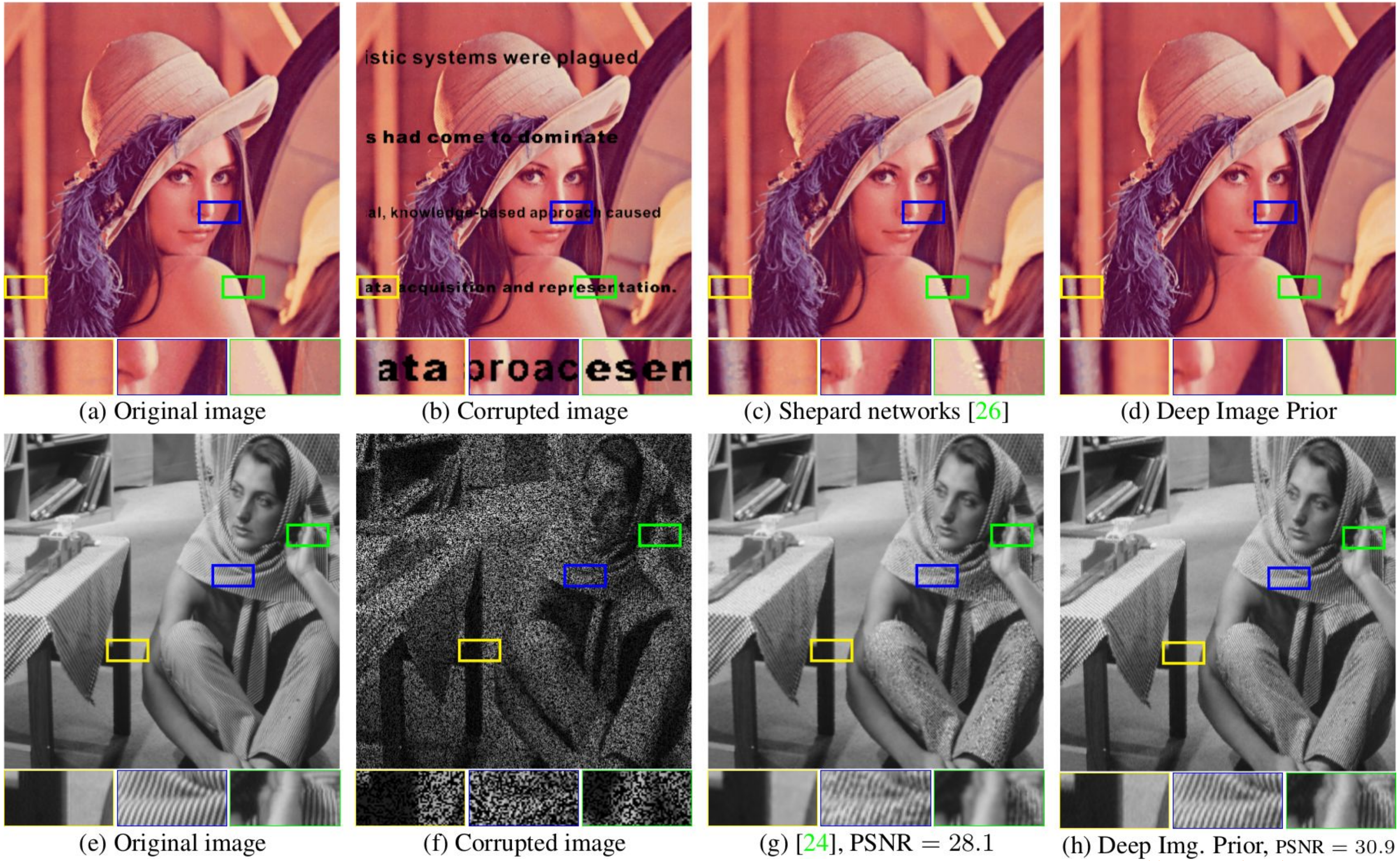


Figure 7 of "Deep Image Prior", https://arxiv.org/abs/1711.10925v2

Figure 5: **Inpainting diversity.** Left: original image (black pixels indicate holes). The remaining four images show results obtained using deep prior corresponding to different input vector z.

Figure 5 of "Deep Image Prior" supplementary materials, https://arxiv.org/abs/1711.10925

Deep Prior paper website with supplementary material

VGG

VGG – 2014 (6.8% ILSVRC top-5 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					

Figure 1 of "Very Deep Convolutional Networks For Large-Scale Image Recognition", https://arxiv.org/abs/1409.1556

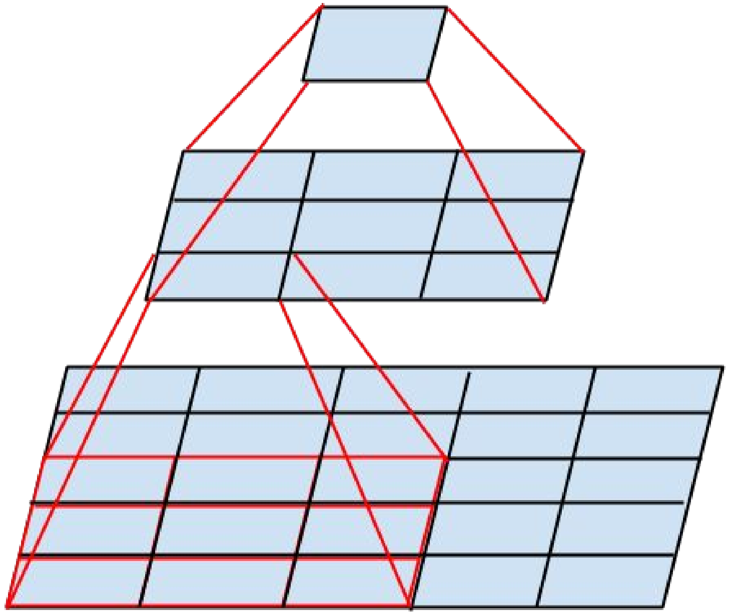


Figure 1 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

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

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

Figure 2 of "Very Deep Convolutional Networks For Large-Scale Image Recognition", https://arxiv.org/abs/1409.1556

VGG – 2014 (6.8% ILSVRC top-5 error)

Training detail similar to AlexNet:

- SGD with batch size ~~128~~ 256, momentum 0.9, weight decay 0.0005
- initial learning rate 0.01, manually divided by 10 when validation error rate stopped improving
- ReLU nonlinearities
- dropout with rate 0.5 on the fully-connected layers (except for the output layer)
- data augmentation using translations and horizontal reflections (choosing random 224×224 patches from 256×256 images)
  - additionally, multi-scale training and evaluation is performed. During training, each image is resized so that its smaller size is S, sampled uniformly from $[256, 512]$
  - during test time, the image is rescaled three times so that the smaller size is $256, 384, 512$, respectively, and the results on the three images were averaged
  - inference is performed on images of possible larger sizes – therefore, obtaining possibly larger than 7×7 resolution before the FC layer; the remaining layers are then evaluated on all 7×7 patches and the results are averaged

VGG – 2014 (6.8% ILSVRC top-5 error)

Table 3: **ConvNet performance at a single test scale.**

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**

Table 3 of "Very Deep Convolutional Networks For Large-Scale Image Recognition", https://arxiv.org/abs/1409.1556

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

Table 4 of "Very Deep Convolutional Networks For Large-Scale Image Recognition", https://arxiv.org/abs/1409.1556

VGG – 2014 (6.8% ILSVRC top-5 error)

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

Figure 2 of "Very Deep Convolutional Networks For Large-Scale Image Recognition", https://arxiv.org/abs/1409.1556

Inception

Inception (GoogLeNet) – 2014 (6.7% ILSVRC top-5 error)

Inception block:

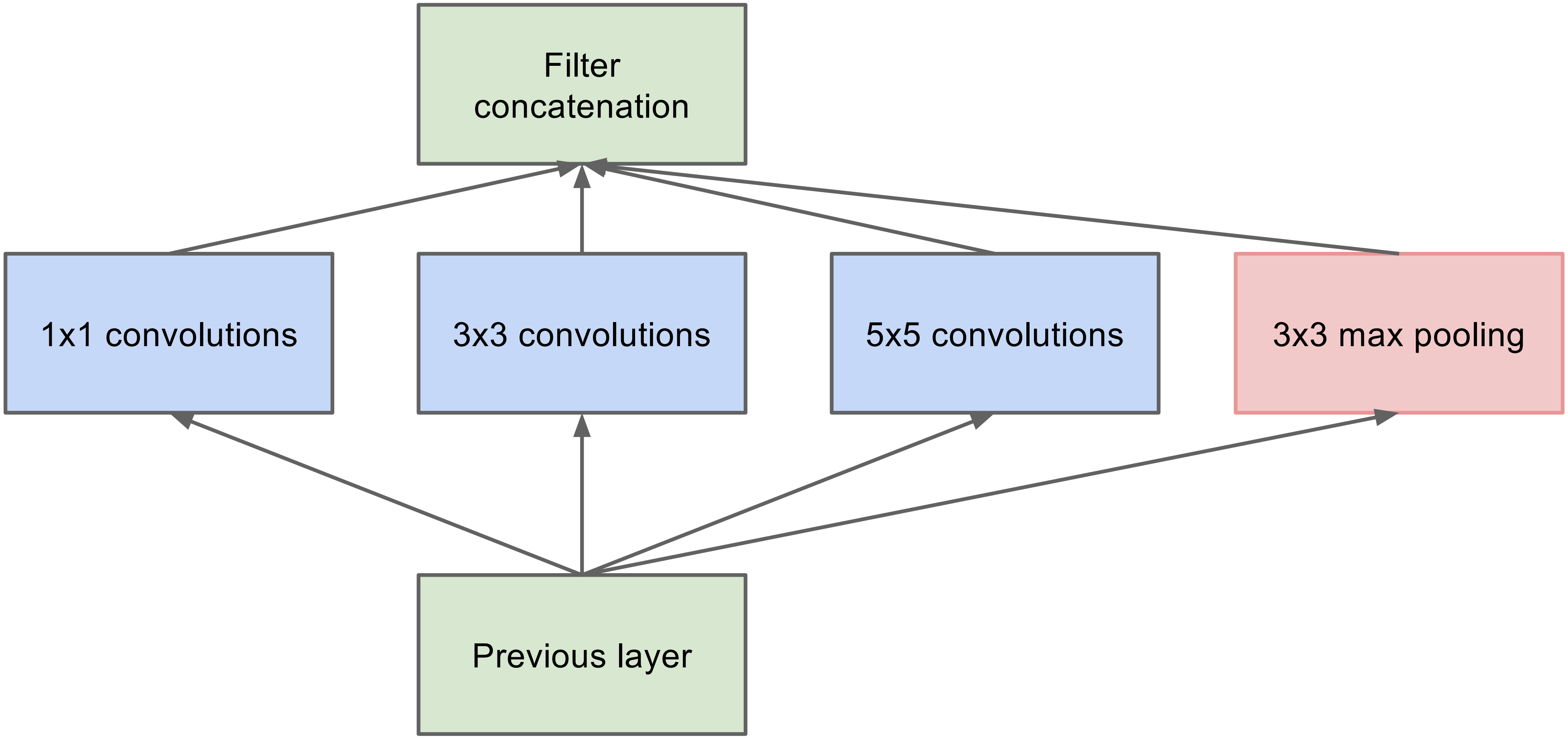


Figure 2 of "Going Deeper with Convolutions", https://arxiv.org/abs/1409.4842

Inception (GoogLeNet) – 2014 (6.7% ILSVRC top-5 error)

Inception block with dimensionality reduction:

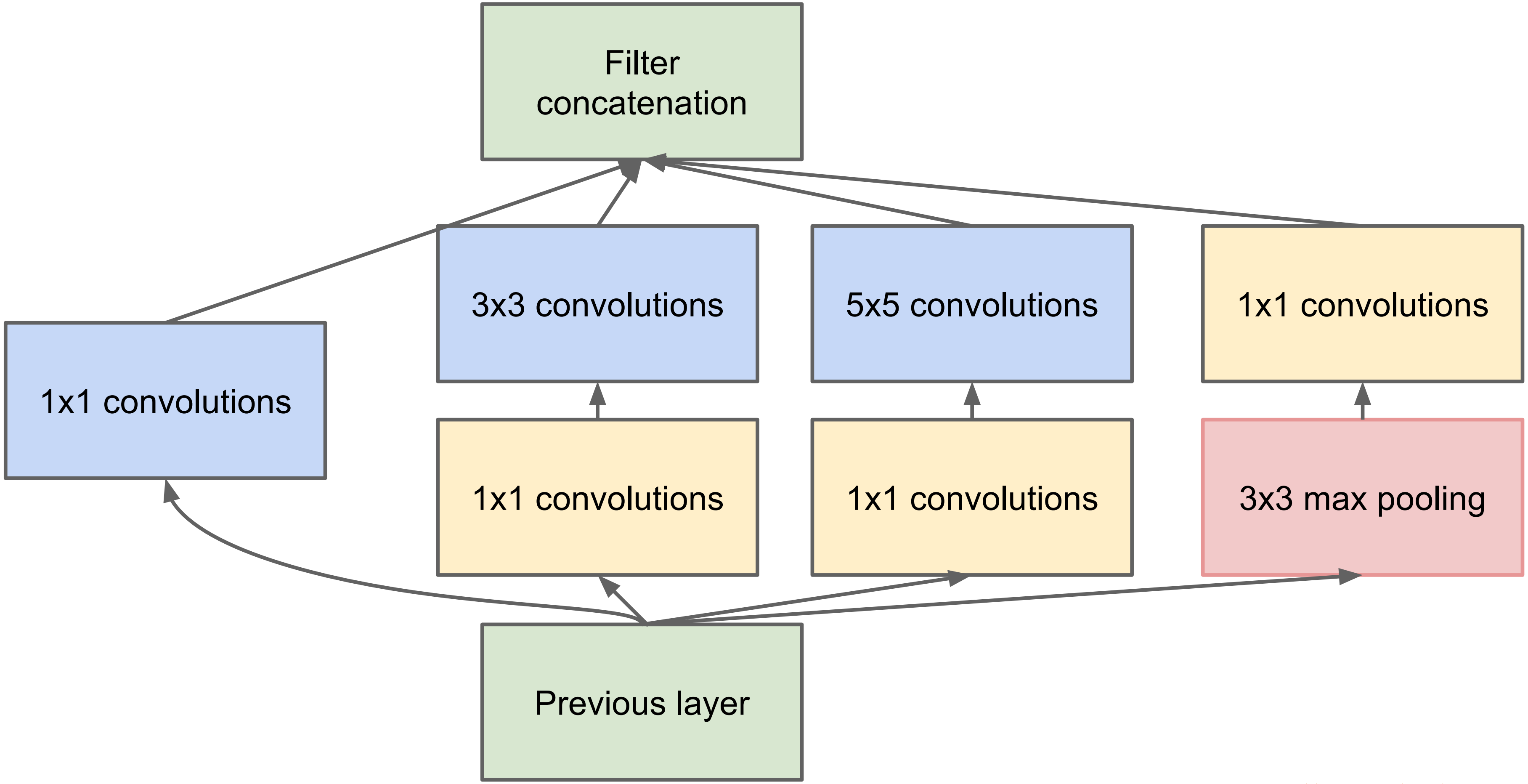


Figure 2 of "Going Deeper with Convolutions", https://arxiv.org/abs/1409.4842

Inception (GoogLeNet) – 2014 (6.7% ILSVRC top-5 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	159K	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								

Table 1 of "Going Deeper with Convolutions", https://arxiv.org/abs/1409.4842

Inception (GoogLeNet) – 2014 (6.7% ILSVRC top-5 error)

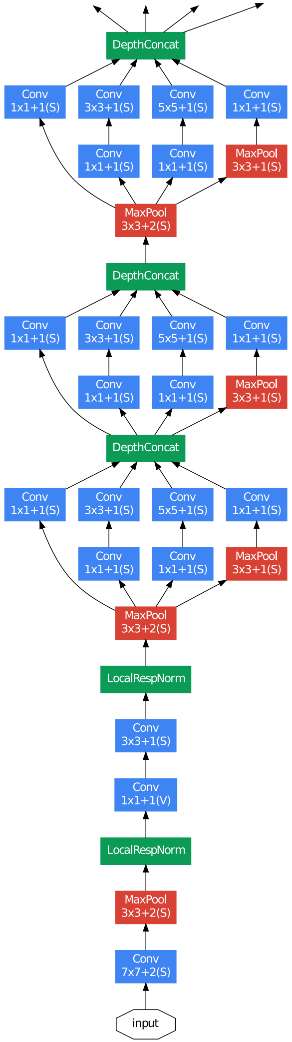


Figure 3 of "Going Deeper with Convolutions", https://arxiv.org/abs/1409.4842

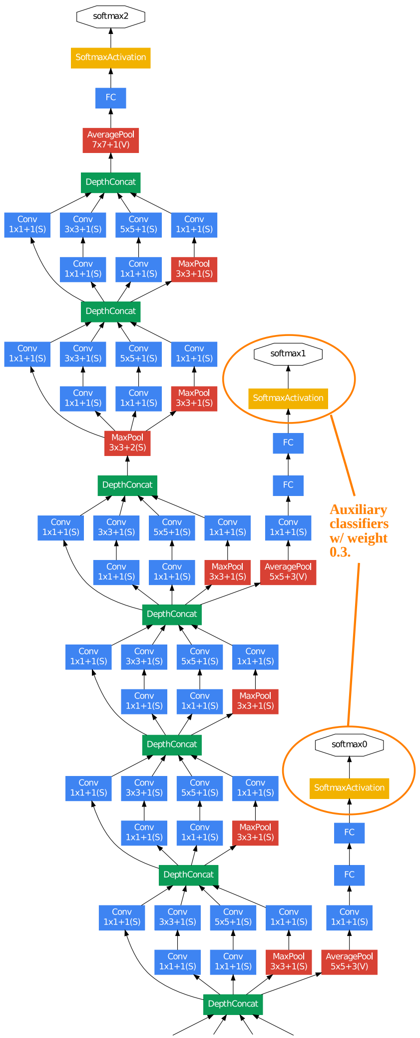


Figure 3 of "Going Deeper with Convolutions", https://arxiv.org/abs/1409.4842

Inception (GoogLeNet) – 2014 (6.7% ILSVRC top-5 error)

Training details:

- SGD with momentum 0.9
- fixed learning rate schedule of decreasing the learning rate by 4% each 8 epochs
- during test time, the image was rescaled four times so that the smaller size was $256, 288, 320, 352$, respectively.

  For each image, the left, center and right square was considered, and from each square six crops of size 224×224 were extracted (4 corners, middle crop and the whole scaled-down square) together with their horizontal flips, arriving at $4 \cdot 3 \cdot 6 \cdot 2 = 144$ crops per image
- 7 independently trained models were ensembled

Inception (GoogLeNet) – 2014 (6.7% ILSVRC top-5 error)

Number of models	Number of Crops	Cost	Top-5 error	compared to base
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

Table 3 of "Going Deeper with Convolutions", https://arxiv.org/abs/1409.4842

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

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

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 of a mini-batch of m examples $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$ is the following:

Inputs: Mini-batch $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$, $\varepsilon \in \mathbb{R}$ with default value 0.001
Parameters: $\boldsymbol{\beta}$ initialized to $\mathbf{0}$, $\boldsymbol{\gamma}$ initialized to $\mathbf{1}$; both trained by the optimizer
Outputs: Normalized batch $(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)})$

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

Batch normalization is added just before a nonlinearity f, and it is useless to add bias before it (because it will cancel out). Therefore, we replace $f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ by

$$f(BN(\boldsymbol{W}\boldsymbol{x})).$$

Batch Normalization during Inference

During inference, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are fixed (so that prediction does not depend on other examples in a batch).

They could be precomputed after training on the whole training data, but in practice we estimate $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}^2$ during training using an exponential moving average.

Additional Inputs: momentum $\tau \in \mathbb{R}$ with default value of 0.99

Additional Parameters: $\hat{\boldsymbol{\mu}}$ initialized to $\mathbf{0}$, $\hat{\boldsymbol{\sigma}}^2$ initialized to $\mathbf{1}$; both updated manually

During training, also perform:

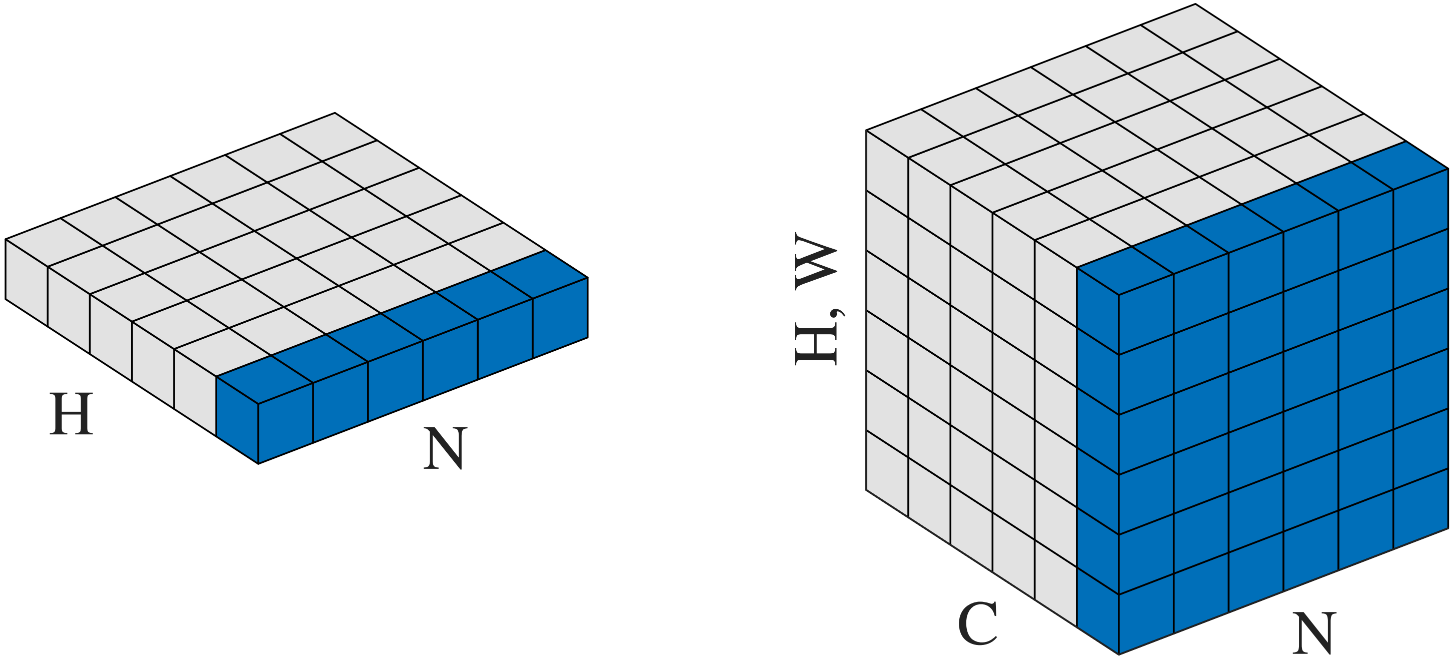
- $\hat{\boldsymbol{\mu}} \leftarrow \tau \hat{\boldsymbol{\mu}} + (1 - \tau)\boldsymbol{\mu}$
- $\hat{\boldsymbol{\sigma}}^2 \leftarrow \tau \hat{\boldsymbol{\sigma}}^2 + (1 - \tau)\boldsymbol{\sigma}^2$

Batch normalization is then during inference computed as:

- $\hat{\boldsymbol{x}}^{(i)} \leftarrow (\boldsymbol{x}^{(i)} - \hat{\boldsymbol{\mu}}) / \sqrt{\hat{\boldsymbol{\sigma}}^2 + \varepsilon}$
- $\boldsymbol{y}^{(i)} \leftarrow \boldsymbol{\gamma} \odot \hat{\boldsymbol{x}}^{(i)} + \boldsymbol{\beta}$

When a batch normalization is used on a fully connected layer, each neuron is normalized individually across the minibatch.

However, for convolutional networks we would like the normalization to honour their properties, most notably the shift invariance. We therefore normalize each channel across not only the minibatch, but also across all corresponding spatial/temporal locations.



Adapted from Figure 2 of "Group Normalization", https://arxiv.org/abs/1803.08494

Inception with BatchNorm (4.8% ILSVRC top-5 error)

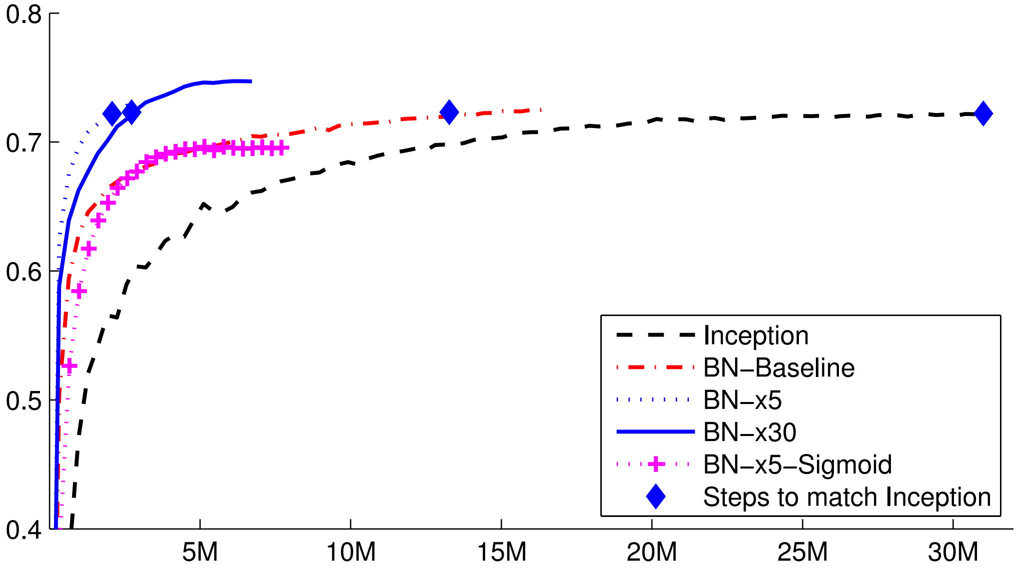


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

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%

Figure 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.*

Figures 2 and 3 of "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", https://arxiv.org/abs/1502.03167

The BN-x5 and BN-x30 use 5/30 times larger initial learning rate, faster learning rate decay, no dropout, weight decay smaller by a factor of 5, and several more minor changes.

ILSVRC Image Recognition Top-5 Error Rates

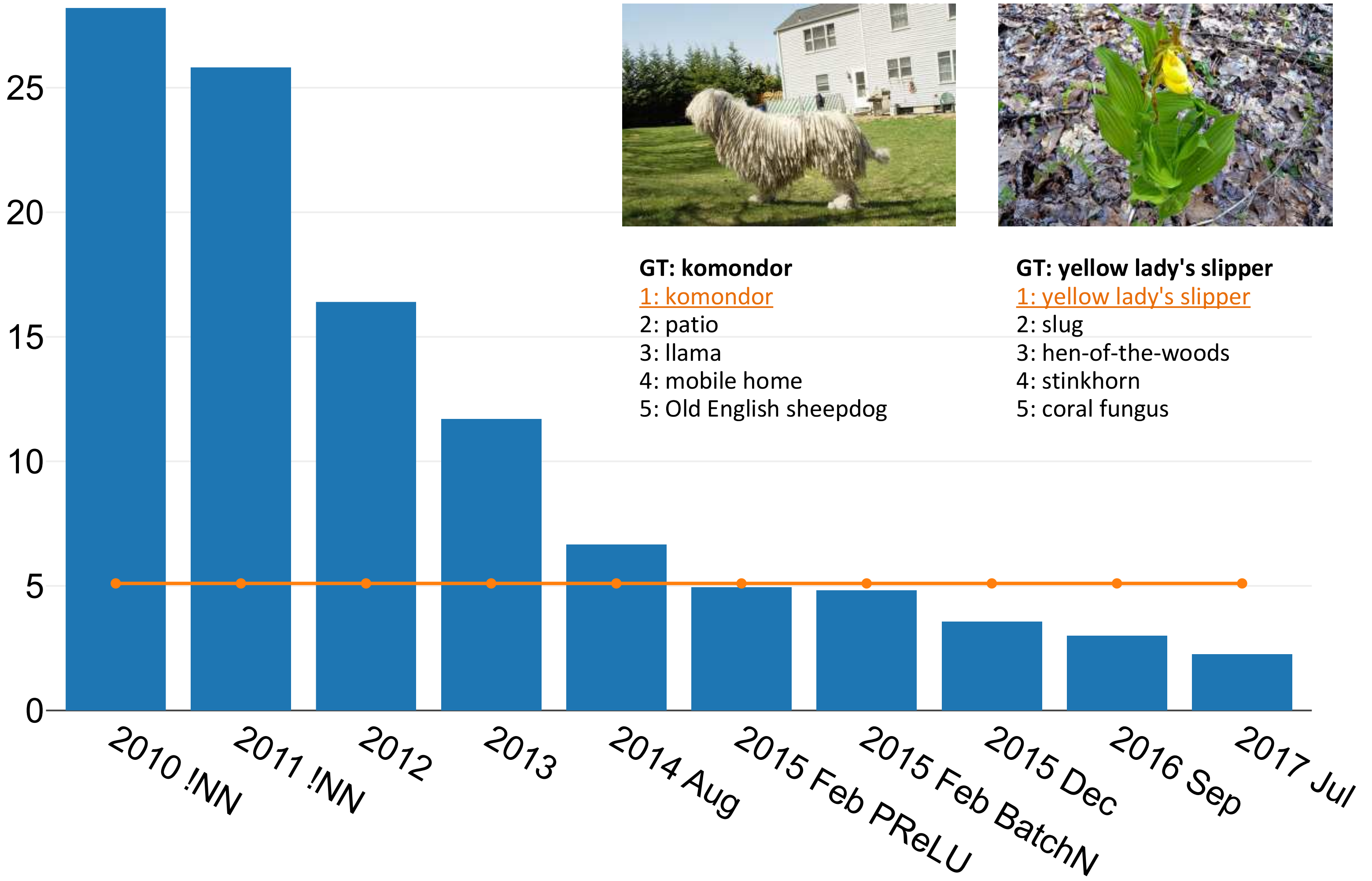


Figure 4 of "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", https://arxiv.org/abs/1502.01852

Inception v2 and v3 – 2015 (3.6% ILSVRC top-5 error)

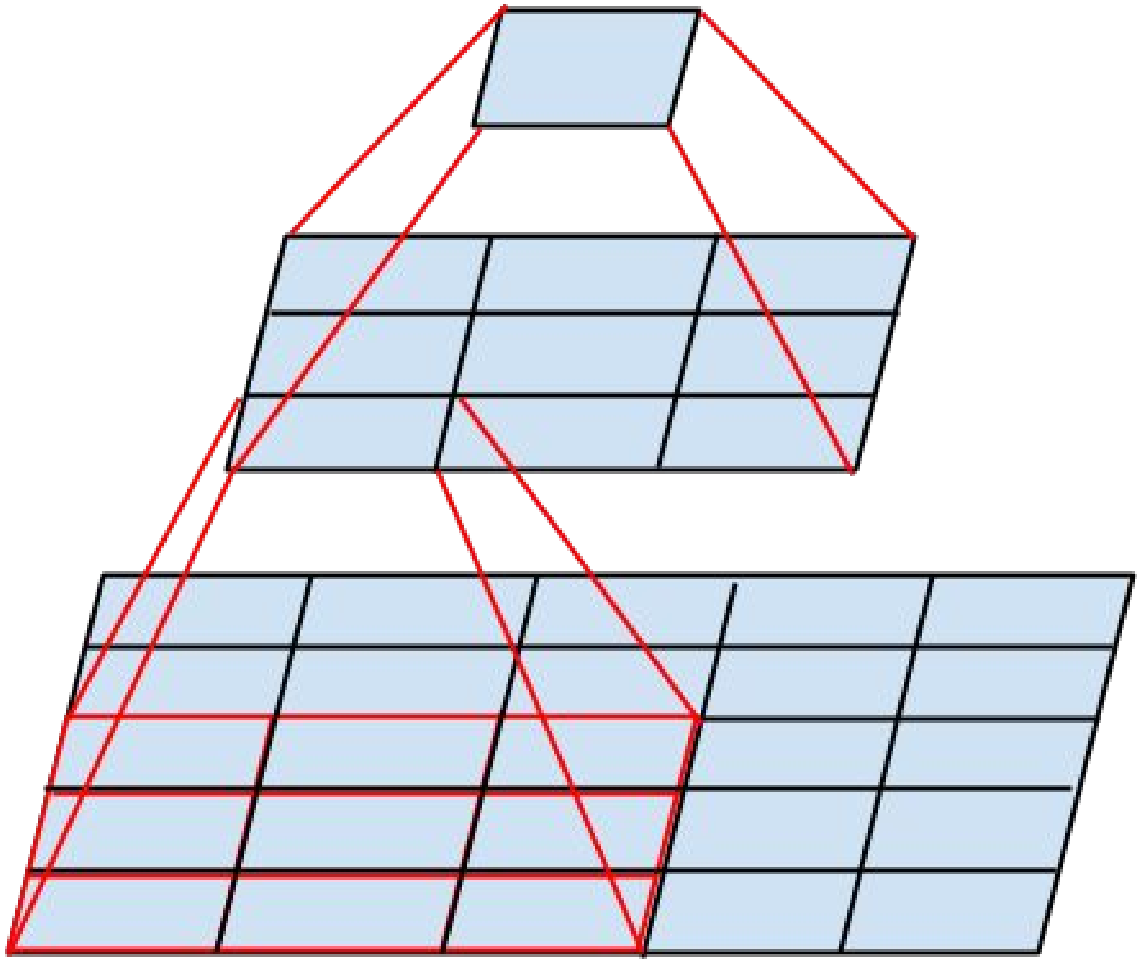


Figure 1 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

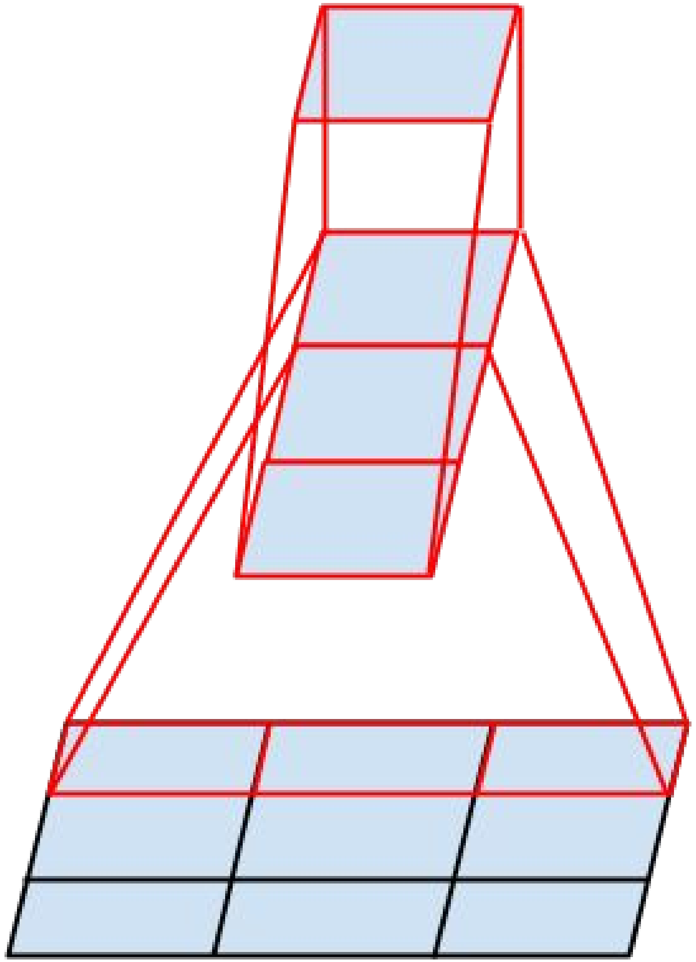


Figure 3 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

Inception v2 and v3 – 2015 (3.6% ILSVRC top-5 error)

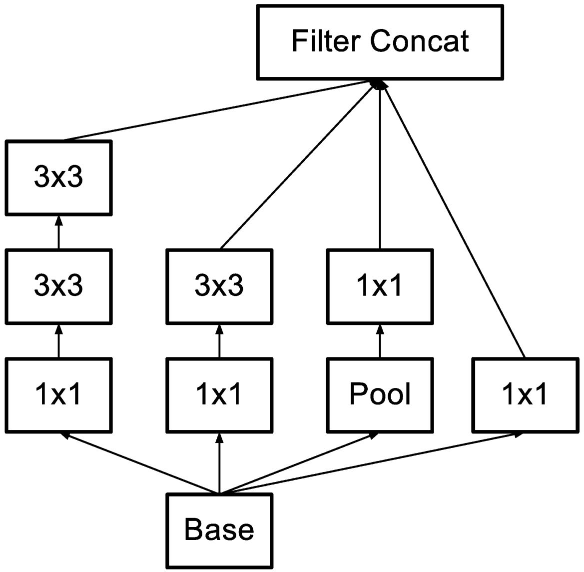


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

Figure 5 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

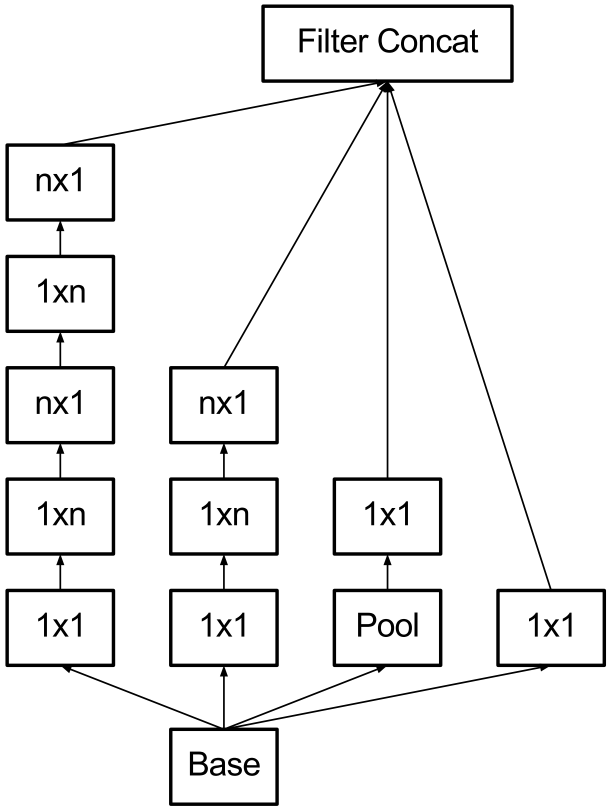


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

Figure 6 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

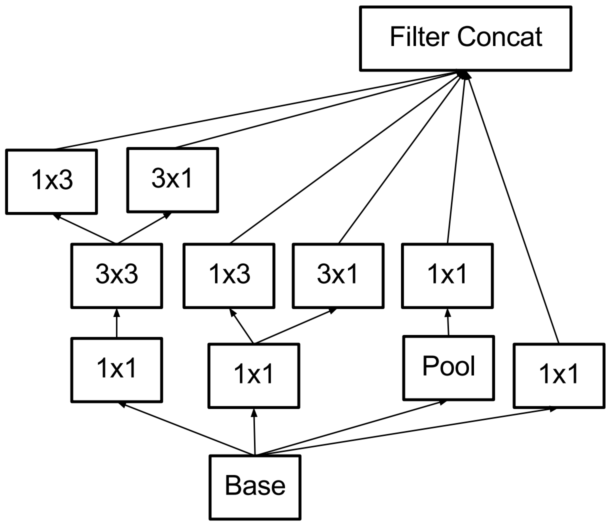


Figure 7. Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8×8) grids to promote high dimensional representations, as suggested by principle 2 of Section 2. We are using this solution only on the coarsest grid, since that is the place where producing 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.

Figure 7 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

Training details:

- RMSProp with momentum of $\beta = 0.9$ and $\varepsilon = 1.0$
  - no weight decay
- batch size of 32 for 100 epochs
- initial learning rate of 0.045, decayed by 6% every two epochs
- gradient clipping with threshold 2.0 was used to stabilize the training
- label smoothing was first used in this paper, with $\alpha = 0.1$
- input image size enlarged to 299×299

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	As in figure 5	$35\times35\times288$
5×Inception	As in figure 6	$17\times17\times768$
2×Inception	As in figure 7	$8\times8\times1280$
pool	8×8	$8\times8\times2048$
linear	logits	$1\times1\times2048$
softmax	classifier	$1\times1\times1000$

Table 1 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

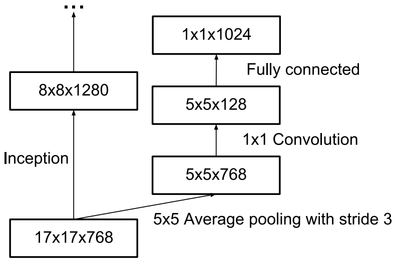


Figure 8. Auxiliary classifier on top of the last 17×17 layer. Batch normalization[7] of the layers in the side head results in a 0.4% absolute gain in top-1 accuracy. The lower axis shows the number of itertions performed, each with batch size 32.

Figure 8 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

Inception v2 and v3 – 2015 (3.6% ILSVRC top-5 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

Table 3 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

Network	**Crops Evaluated**	**Top-5 Error**	**Top-1 Error**
GoogLeNet [20]	10	-	9.15%
GoogLeNet [20]	144	-	7.89%
VGG [18]	-	24.4%	6.8%
BN-Inception [7]	144	22%	5.82%
PReLU [6]	10	24.27%	7.38%
PReLU [6]	-	21.59%	5.71%
Inception-v3	12	19.47%	4.48%
Inception-v3	144	**18.77%**	**4.2%**

Table 4 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

Network	**Models Evaluated**	**Crops Evaluated**	**Top-1 Error**	**Top-5 Error**
VGGNet [18]	2	-	23.7%	6.8%
GoogLeNet [20]	7	144	-	6.67%
PReLU [6]	-	-	-	4.94%
BN-Inception [7]	6	144	20.1%	4.9%
Inception-v3	4	144	**17.2%**	**3.58%***

Table 5 of "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567

Summary

- Convolutions can provide
  - local interactions in spatial/temporal dimensions
  - shift invariance
  - *much* less parameters than a fully connected layer
- Usually repeated 3×3 convolutions are enough, no need for larger filter sizes.
- When pooling is performed, double the number of channels (i.e., the first convolution following the pooling layer will have twice as many output channels).
- If your network is deep enough (the last hidden neurons have a large receptive fields), final fully connected layers are not needed, and global average pooling is enough.
- Batch normalization is a great regularization method for CNNs, allowing removal/decrease of dropout and L^2 regularization.
- Small weight decay (i.e., L^2 regularization) of usually 1e-4 is still useful for regularizing convolutional kernels.