Convolutional Neural Networks

Hwann-Tzong Chen

Naitonal Tsing Hua University

7 November 2017

Convolutional Neural Networks (CNNs/ConvNets)

References

- Stanford CS231n
- Chapter 8 of deeplearningbook.org

Assumptions

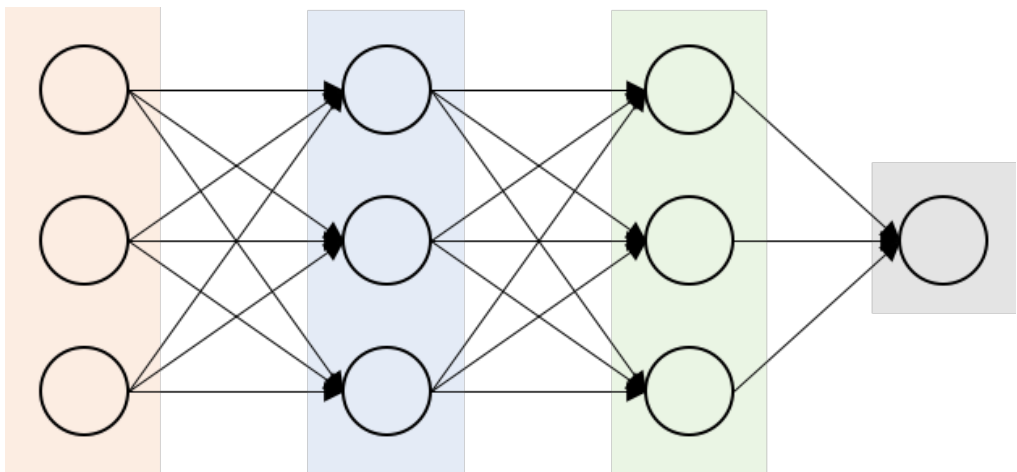
- Inputs are images
- Encoding spatial structures
- Making the forward function more efficient to implement (convolution on GPU)
- Reducing the amount of parameters

Very popular in computer vision, used in almost all state-of-the-art methods for image classification, object detection, semantic segmentation, human pose estimation, super-resolution, ...

Architecture Overview

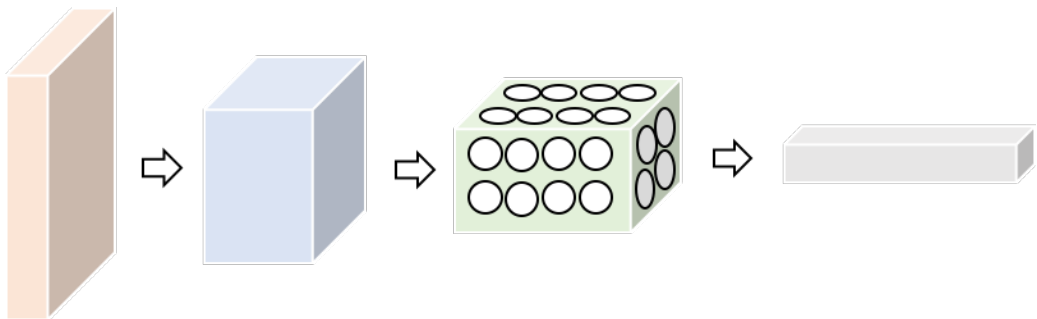
Regular (fully connected) neural nets do not scale well to full images. Consider an input image of $200 \times 200 \times 3$ pixels. A neuron at the second layer connecting to all pixels in the input image would have $120,000$ weights.

Full connectivity $\Rightarrow$ redundancy.



Containing too many parameters might lead to overfitting.

ConvNet Architecture



Quote from CS231n:
A ConvNet is made up of layers. Every layer has a simple API: It transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.

Layers used to build ConvNets

Convolutional layer (CONV), pooling layer (POOL), fully-connected layer (FC)

Example architecture: a simple ConvNet for CIFAR-10

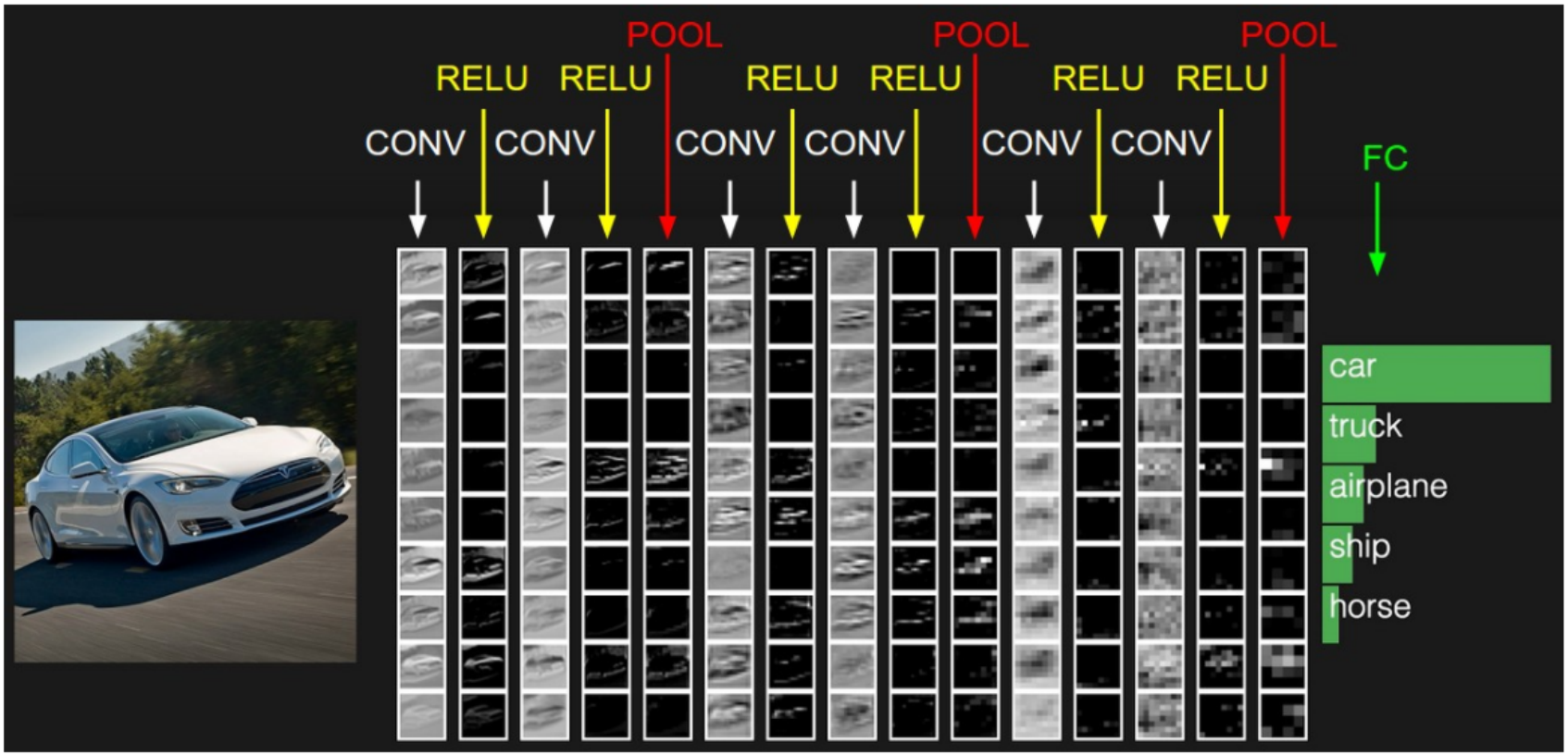
[INPUT - CONV - RELU - POOL - FC]

- INPUT [32x32x3]: raw pixel values of the RGB image.
- CONV [32x32x12]: computes a dot product between the weights and a small region in the input volume. Done by convolution with 12 filters.
- RELU [32x32x12]: applied elementwise, the size of the volume unchanged.
- POOL [16x16x12]: downsampling along the spatial dimensions (width, height).
- FC [1x1x10]: connected to all the units in the previous volume.

Summary (CS231n)

- A ConvNet architecture is a list of layers that transform the image volume into an output volume (e.g. holding the class scores)
- Different types of layers (e.g. CONV/FC/RELU/POOL). Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't). Parameters (weights and biases) are trained with gradient descent.
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't)

Web-based Demo at http://cs231n.stanford.edu/

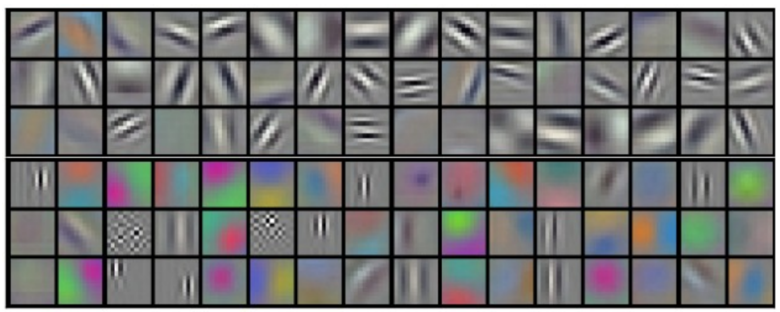


Convolutional Layers

Main ideas

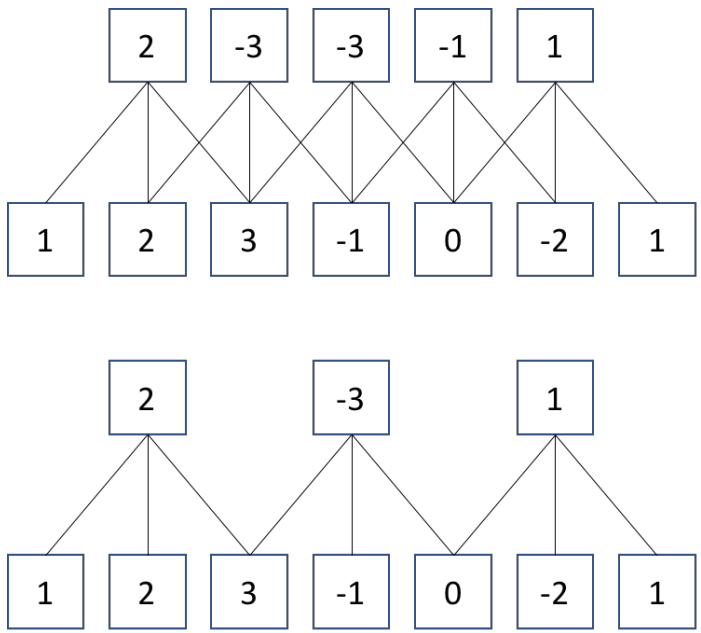
- Filters: detecting oriented edges, corners, ... (what to be learned)
- Local connectivity: receptive field
- Spatial arrangement: depth, stride, zero-padding
- Parameter sharing

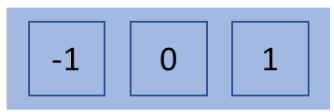
Note: The connections are local in space (along width and height), but always full along the entire depth (all channels) of the input volume. (Don't confuse with 3D convolution.)
Figure from CS231n



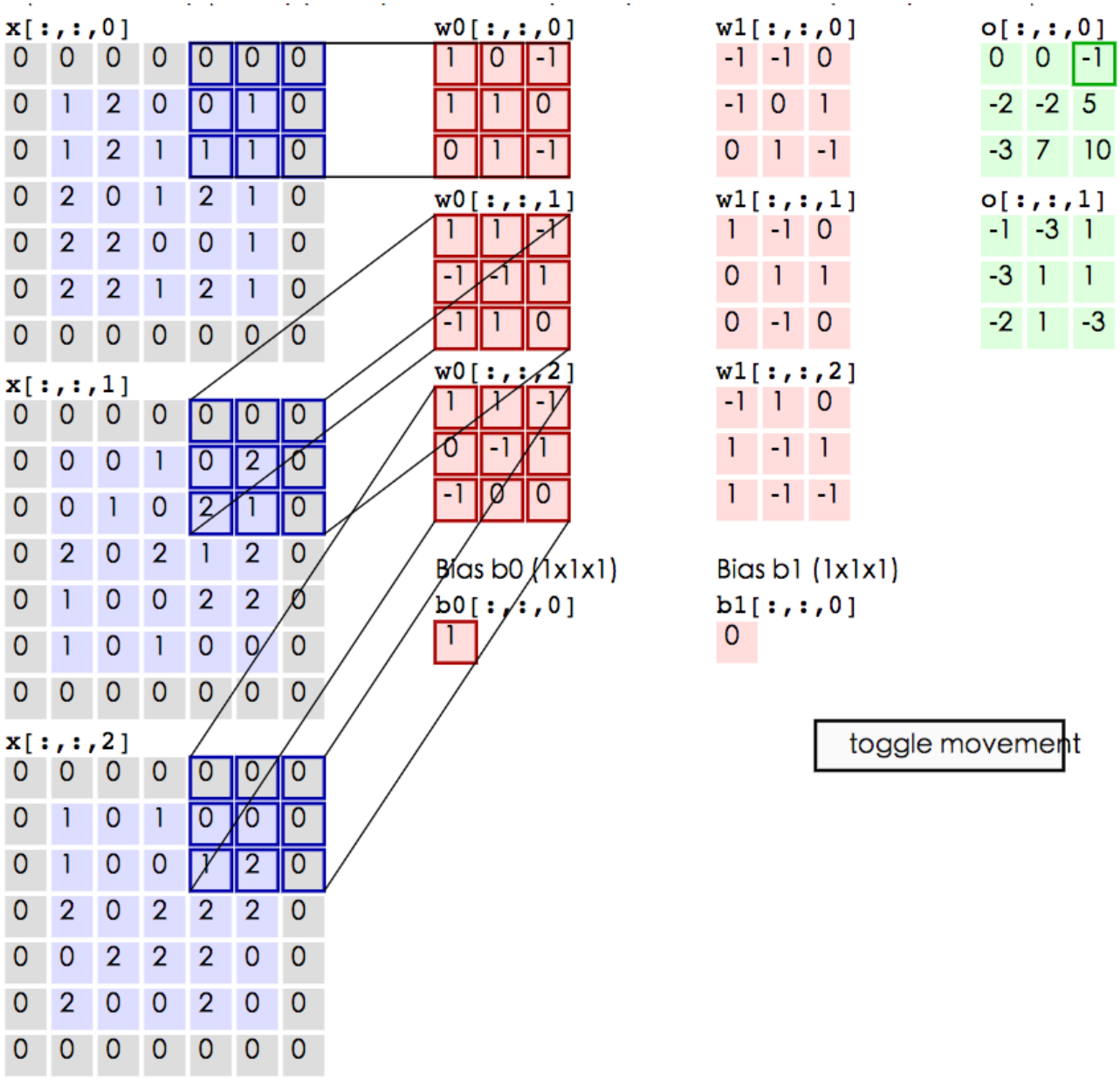
Kernel Size, Stride, Zero-padding

W: input size
F: receptive field size
S: stride
P: zero-padding
Output size $= (W - F + 2P)/S + 1$

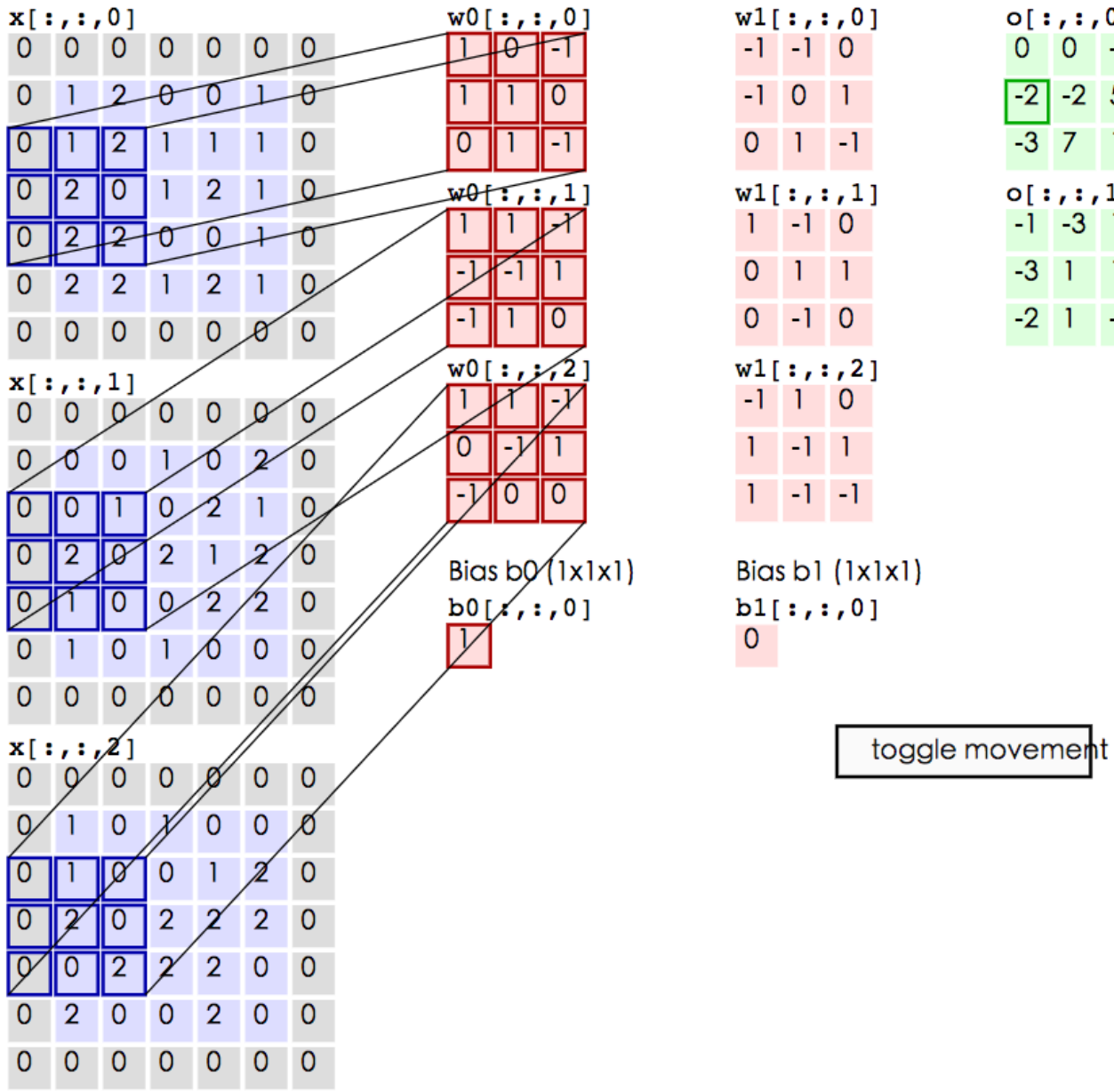




2D Convolution over the Entire Depth



2D Convolution over the Entire Depth



Real-world Example

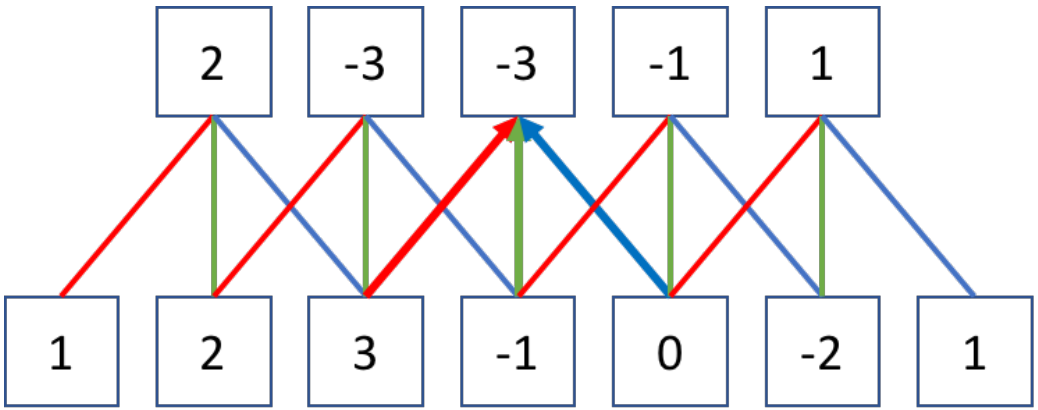
AlexNet for 2012 ImageNet Challenge

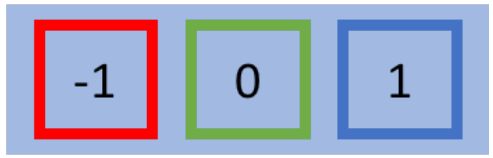
- Input image size: 224×224 pixels
- The first convolutional layer: kernel size $F = 11$, stride $S = 4$, zero-padding $P = 0$, depth $K = 96 \Rightarrow (227 - 11)/4 + 1 = 55$
- The size of the first CONV layer output volume: [55x55x96] $= 290{,}400$
- # parameters in a filter (with RGB channels): [11x11x3] $=$ 363 (plus 1 bias)

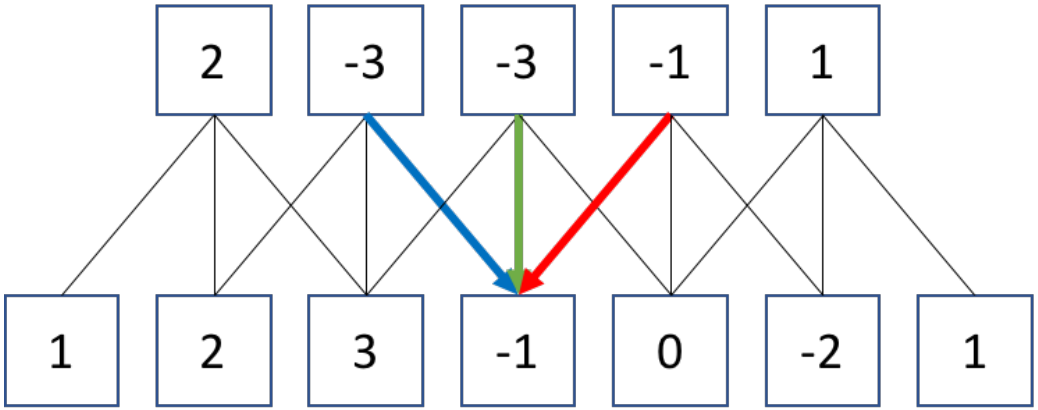
Without parameter sharing, the number of parameters should be $290{,}400 \times 364 = 105{,}705{,}600$.
With parameter sharing (same filter at different locations), the number of parameters is $96 \times 11 \times 11 \times 3 = 34{,}848$ weights plus 96 biases.

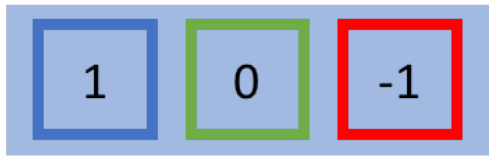
Backpropagation for Convolution

Using spatially-flipped filters:





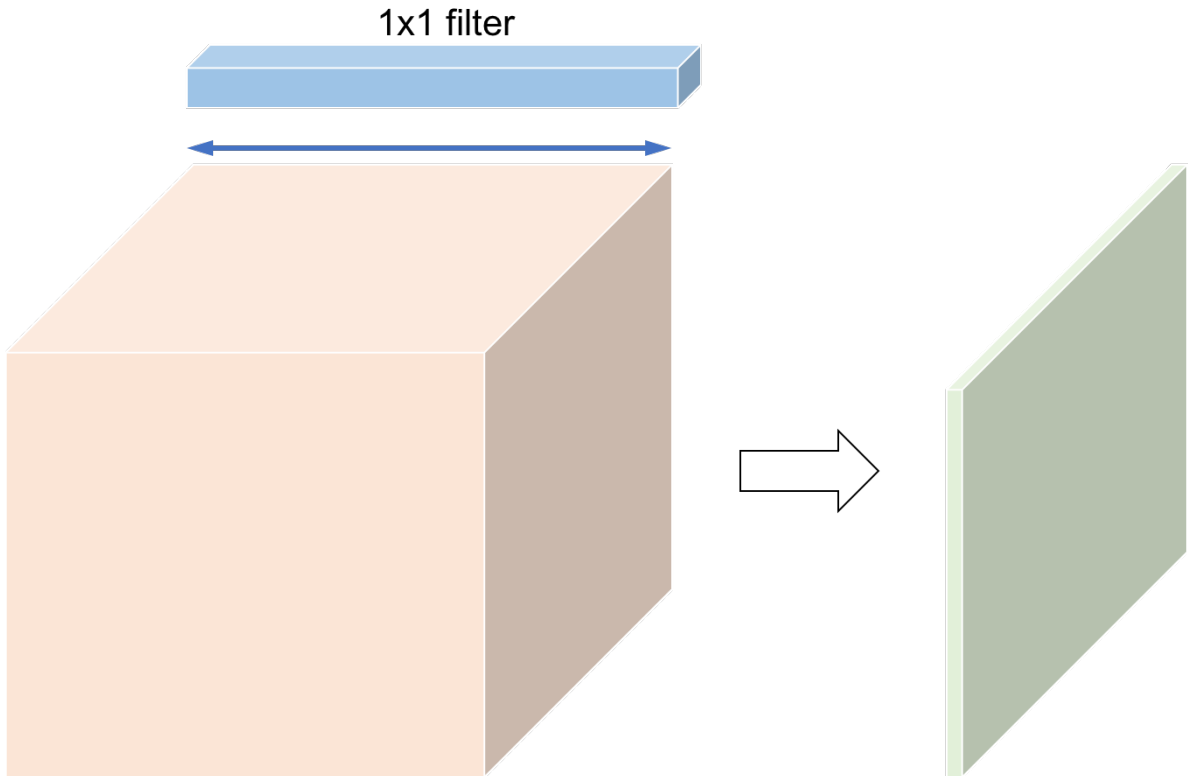




1×1 Convolutions

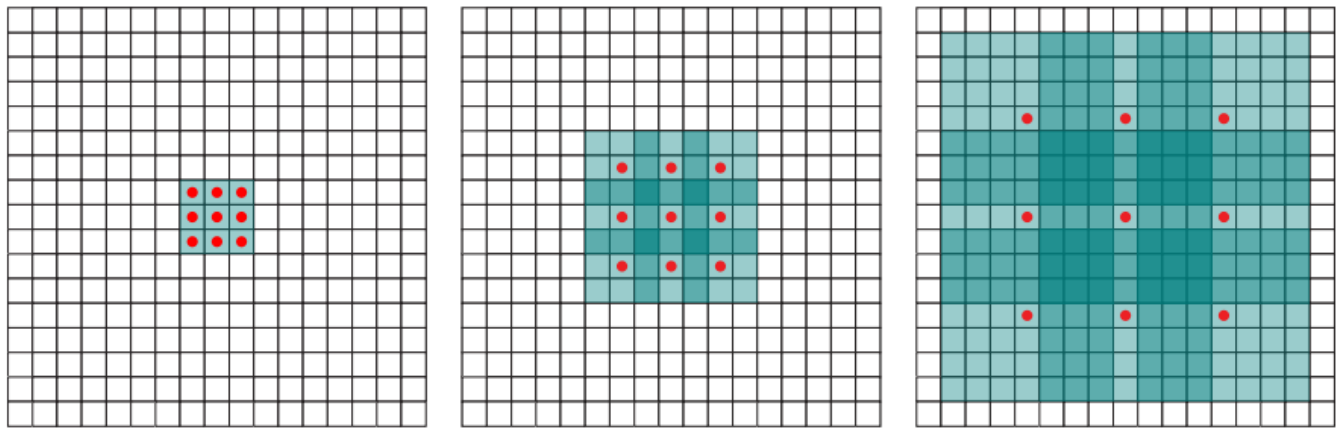
An convenient way to reduce the depth of an output volume (to reduce the number of feature maps).

E.g., from 96 feature maps to 32 feature maps using 32 filters of 1×1 convolution



Dilated Convolutions

Yu & Koltun, "Multi-scale Context Aggregation by Dilated Convolutions", ICLR 2016.
Filters with holes:

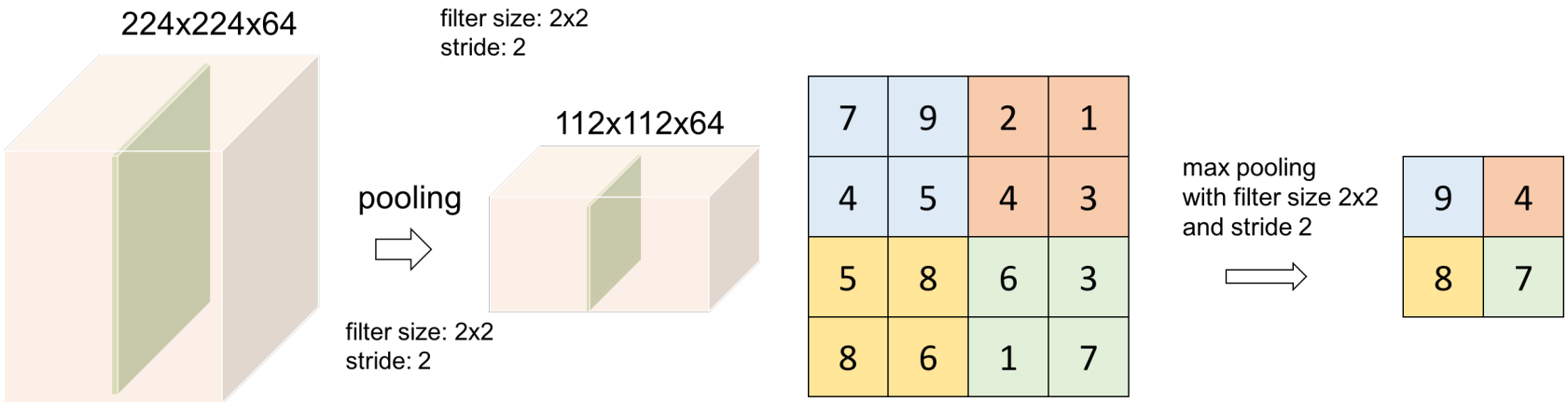


Expanding the receptive field without losing the resolution

Useful for semantic segmentation, increasing segmentation accuracy

Pooling Layer

Max pooling: $F = 3, S = 2$ or $F = 2, S = 2$



- Backpropagation (for max-pooling): keeping track of the index
- Getting rid of pooling: all convolutions and no pooling, common in variational auto-encoders (VAEs) and generative adversarial networks (GANs)

Fully-connected Layer

You already know: as in regular feed-forward NNs.

Converting FC layers to CONV layers

- "*For any CONV layer there is an FC layer that implements the same forward function. The weight matrix would be a large matrix that is mostly zero except for at certain blocks (due to local connectivity) where the weights in many of the blocks are equal (due to parameter sharing).*" — CS231n
- Conversely, any FC layer can be converted to a CONV layer by setting the filter size to be exactly the size of the input volume.

FC to CONV Conversion

Example of AlexNet

Five pooling layers of downsampling $224/2/2/2/2/2 = 7$ result in a volume of size [7x7x512].
Followed by two FC layers of size 4096 and the last FC layer with 1000 neurons that computes the 1000 class scores.

- Replace the first FC layer of a volume [7x7x512] with a CONV layer that uses filter size $F = 7$, giving output volume [1x1x4096].
- Replace the second FC layer with a CONV layer that uses filter size $F = 1$, giving output volume [1x1x4096].
- Replace the last FC layer with $F = 1$, giving final output [1x1x1000].

FC to CONV Conversion

Advantages

Allows "sliding" the original ConvNet very efficiently across many spatial positions in a larger image, in a single forward pass.

- 224x224 image $\rightarrow$ a volume of size [7x7x512] — a reduction by 32,
- forwarding an image of size 384x384 would give the equivalent volume in size [12x12x512], since $384/32 = 12$.
- following through with the next 3 FC-convected CONV layers would now give the final volume of size [6x6x1000], since $(12 - 7)/1 + 1 = 6$.

"*This trick is often used in practice to get better performance, where for example, it is common to resize an image to make it bigger, use a converted ConvNet to evaluate the class scores at many spatial positions and then average the class scores.*" — CS231n

FC to CONV Conversion

"Evaluating the original ConvNet (with FC layers) independently across 224x224 crops of the 384x384 image in strides of 32 pixels gives an identical result to forwarding the converted ConvNet one time." — CS231n

$224 + (6 - 1) \times 32 = 384$

Question: How to efficiently apply the original ConvNet over the image at a stride smaller than 32 pixels, say 16 pixels?
Ans: Applying twice: first over the original image and second over the image but with the image shifted spatially by 16 pixels along both width and height.

ConvNet Architectures: Layer Patterns

INPUT $\rightarrow$ [[CONV $\rightarrow$ RELU]*N $\rightarrow$ POOL?]*M $\rightarrow$ [FC $\rightarrow$ RELU]*K $\rightarrow$ FC

Prefer a stack of small filter CONV to one large receptive field CONV layer:
Three 3x3 CONV layers vs. one 7x7 CONV layer?

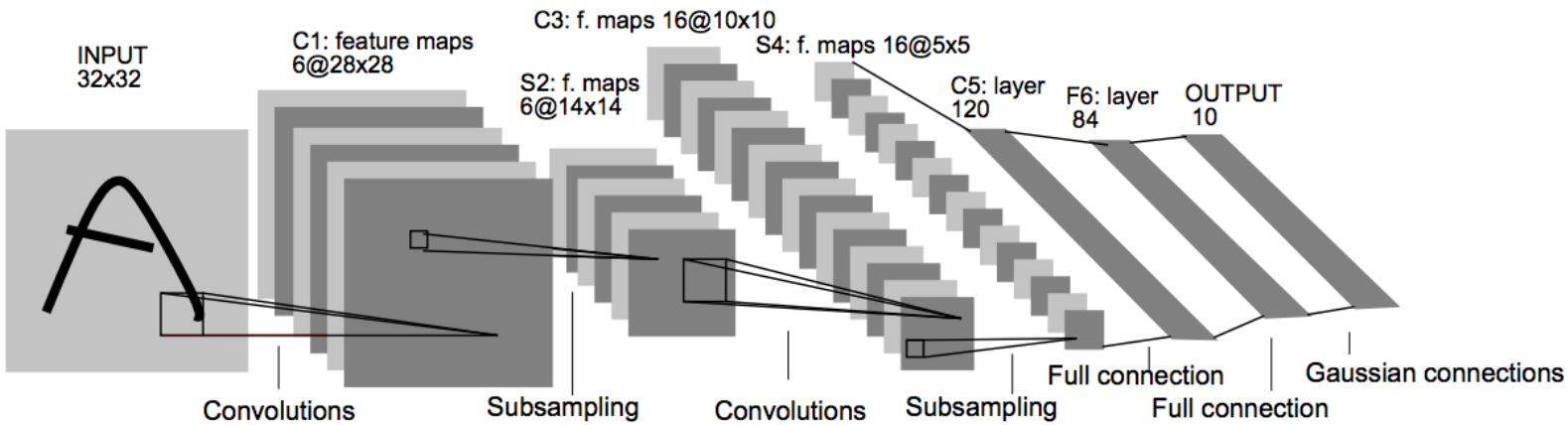
In practice: use whatever works best on ImageNet.
Download a pretrained model and finetune it on your data.

ConvNet Architectures: Layer Sizing Patterns

- Input layer size: 32, 64, 96, 224, 384, 512
- CONV: using small filters, using stride $S = 1$, preserving the input volume size with zero-padding ($F = 3, P = 1$, $F = 5, P = 2$)
  Smaller strides work better in practice.
- POOL: max-pooling with $F = 2, S = 2$

Case Study: LeNet

Developed by Yann LeCun in 1990's
MNIST

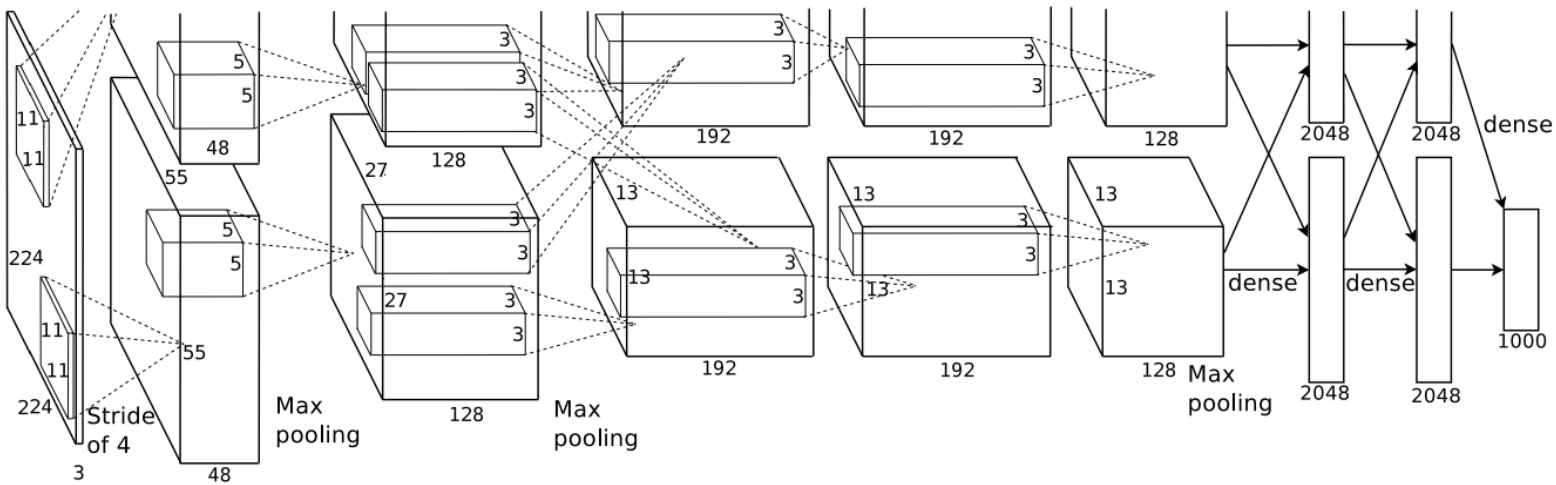


Case Study: AlexNet

Developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton in 2012

ImageNet ILSVRC Challenge (top 5 error of 16% compared to runner-up with 26% error)

60M parameters



Case Study: VGGNet

Developed by Karen Simonyan and Andrew Zisserman

2014 ImageNet ILSVRC Challenge runner-up.

VGG-16 contains 16 CONV/FC layers with 3x3 convolutions and 2x2 pooling from the beginning to the end. Their pretrained model is very popular.

Downside: more expensive to evaluate and uses a lot more memory and parameters (140M). Most of these parameters are in the first fully connected layer.

It was found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

Case Study: VGGNet

```
INPUT: [224x224x3]        memory:  224*224*3=150K   weights: 0
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory:  112*112*64=800K   weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory:  7*7*512=25K  weights: 0
FC: [1x1x4096]  memory:  4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory:  4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory:  1000 weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

Case Study: GoogLeNet

Developed by Szegedy et al. from Google

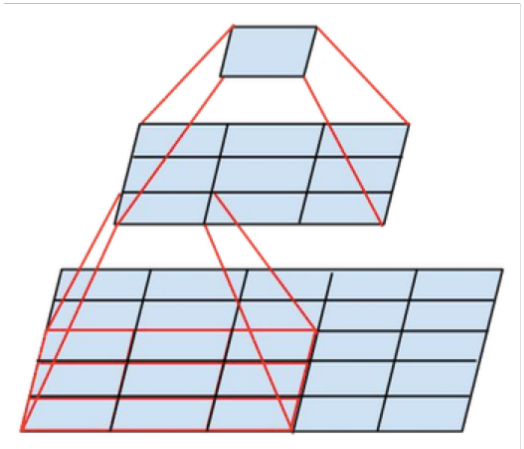
The winner of ILSVRC 2014

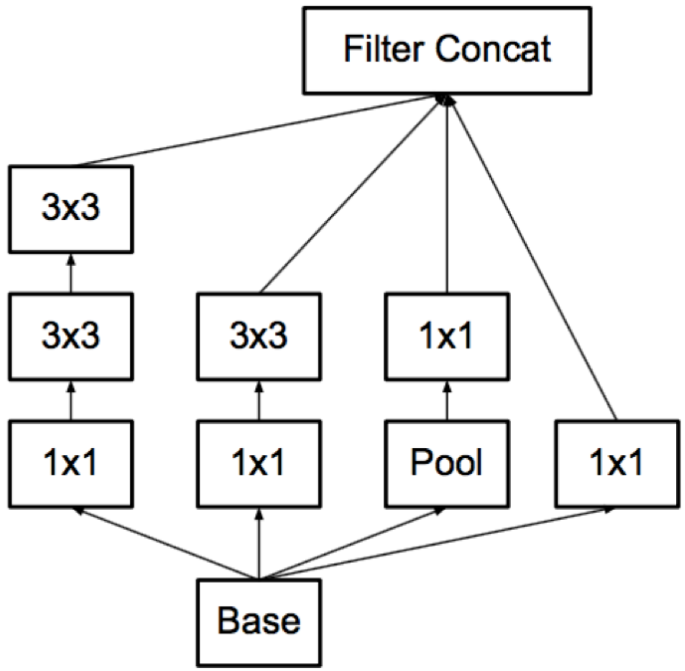
Inception Module that dramatically reduced the number of parameters in the network (4M).

Uses average pooling instead of FC layers at the top of the ConvNet, eliminating a large amount of parameters that do not seem to matter much.

Most recently, inception-v4.

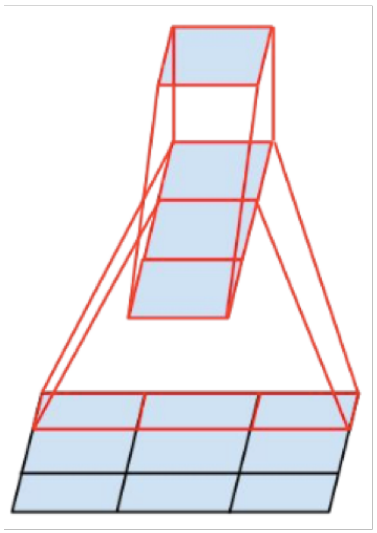
Inception Modules

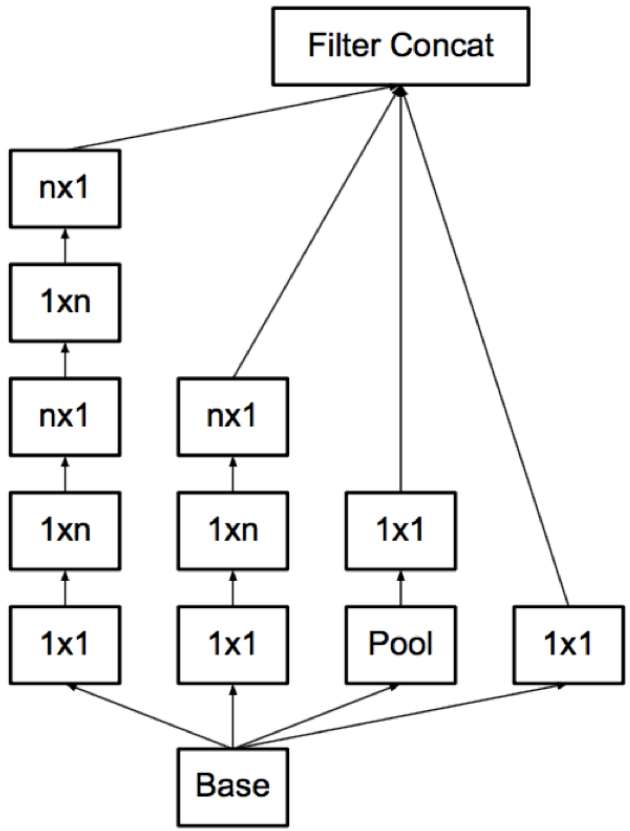




Inception Modules

Factorization





Case Study: Residual Networks (ResNets)

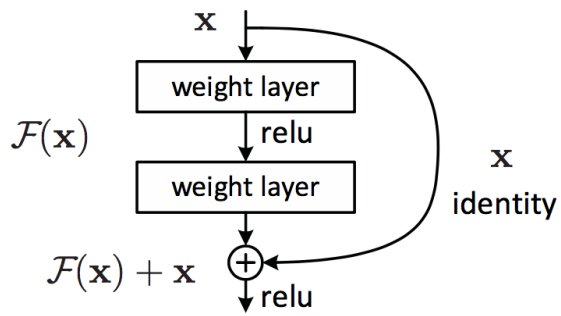
Developed by Kaiming He et al.

The winner ILSVRC 2015 (3.57% error)

Using shortcut connections and batch normalization

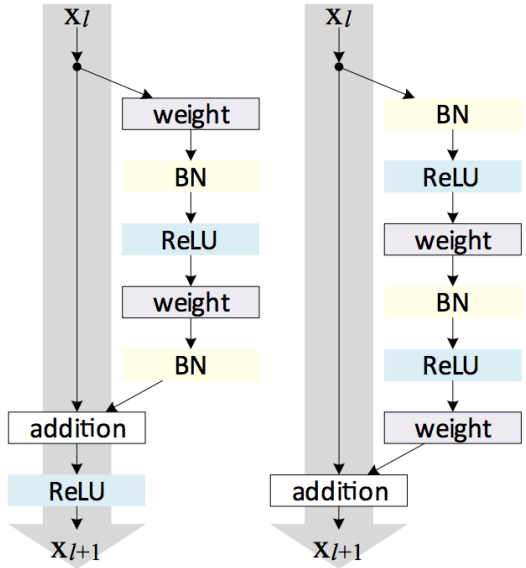
Omitting fully-connected layers at the end of the network

Very deep—152 layers



Inception-v4 (with residual) 3.08% top-5 error

ResNet Pre-activation



(a) original (b) proposed

200-layer ResNet for ImageNet 1001-layer ResNet for CIFAR-10/100 (10.2M parameters)

On CIFAR, ResNet-1001 takes about 27 hours to train on 2 GPUs; on ImageNet, ResNet- 200 takes about 3 weeks to train on 8 GPUs

Training ImageNet Using ResNet in One Hour

Accurate, Large Minibatch SGD:
Training ImageNet in 1 Hour

Priya Goyal Piotr Dollár Ross Girshick Pieter Noordhuis
Lukasz Wesolowski Aapo Kyrola Andrew Tulloch Yangqing Jia Kaiming He

Facebook

Abstract

Deep learning thrives with large neural networks and large datasets. However, larger networks and larger datasets result in longer training times that impede research and development progress. Distributed synchronous SGD offers a potential solution to this problem by dividing SGD minibatches over a pool of parallel workers. Yet to make this scheme efficient, the per-worker workload must be large, which implies nontrivial growth in the SGD minibatch size. In this paper, we empirically show that on the ImageNet dataset large minibatches cause optimization difficulties, but when these are addressed the trained networks exhibit good generalization. Specifically, we show no loss of accuracy when training with large minibatch sizes up to 8192 images. To achieve this result, we adopt a linear scaling rule for adjusting learning rates as a function of minibatch size and develop a new warmup scheme that overcomes optimization challenges early in training. With these simple techniques, our Caffe2-based system trains ResNet-50 with a minibatch size of 8192 on 256 GPUs in one hour, while matching small minibatch accuracy. Using commodity hardware, our implementation achieves ~90% scaling efficiency when moving from 8 to 256 GPUs. This system enables us to train visual recognition models on internet-scale data with high efficiency.

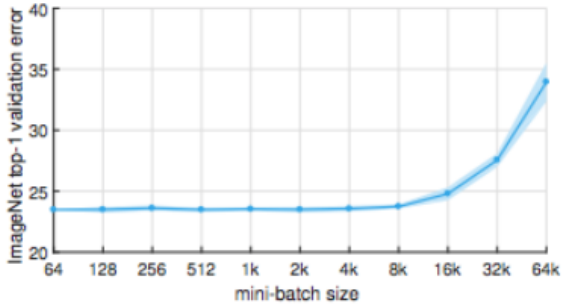


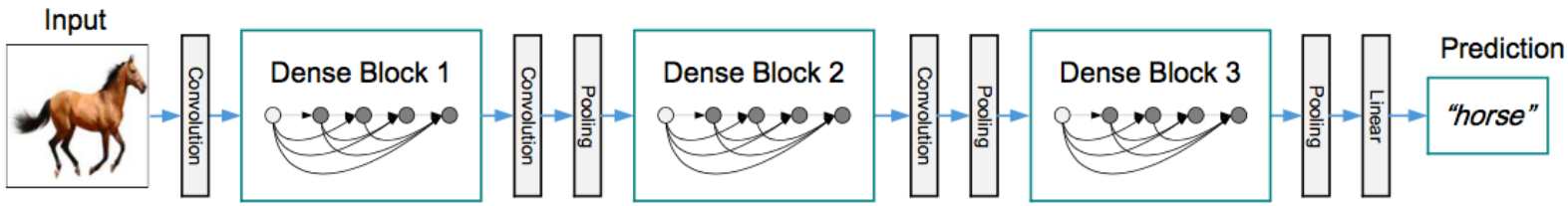
Figure 1. **ImageNet top-1 validation error *vs.* minibatch size.** Error range of plus/minus *two* standard deviations is shown. We present a simple and general technique for scaling distributed synchronous SGD to minibatches of up to 8k images *while maintaining the top-1 error of small minibatch training*. For all minibatch sizes we set the learning rate as a *linear* function of the minibatch size and apply a simple warmup phase for the first few epochs of training. All other hyper-parameters are kept fixed. Using this simple approach, accuracy of our models is invariant to minibatch size (up to an 8k minibatch size). Our techniques enable a linear reduction in training time with ~90% efficiency as we scale to large minibatch sizes, allowing us to train an accurate 8k minibatch ResNet-50 model in 1 hour on 256 GPUs.

mentation [8, 10, 27]. Moreover, this pattern generalizes: larger datasets and network architectures consistently yield improved accuracy across all tasks that benefit from pre-

ResNet-50 with a minibatch size of 8192 on 256 GPUs in one hour

Case Study: DenseNets

Developed by Huang et al.



The idea of DenseNets is based on the observation that if each layer is directly con- nected to every other layer in a feed-forward fashion then the network will be more accurate and easier to train.

Transition layers: 1x1 convolution followed by 2x2 average pooling

Dense Block

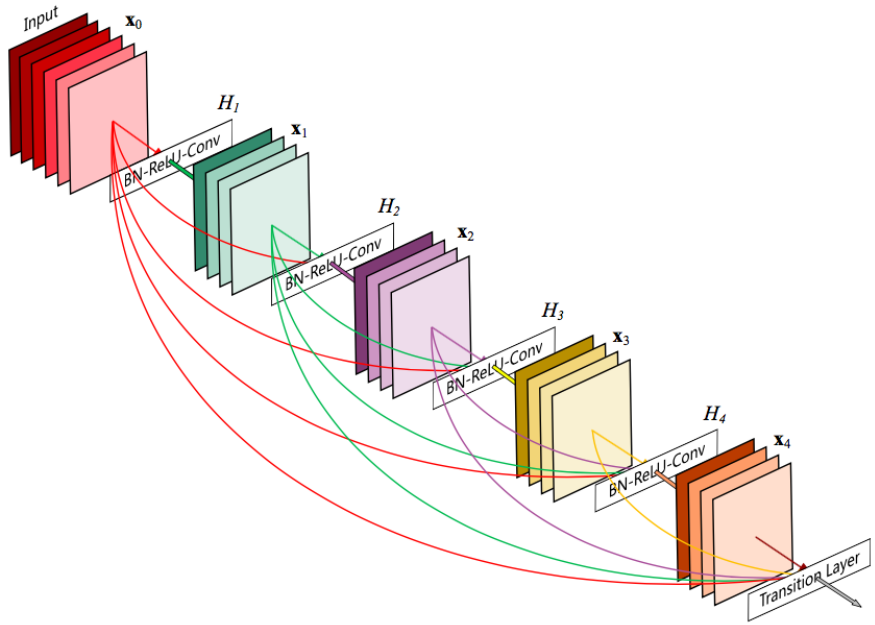


Figure 1: $L = 5, k = 4$: $L(L+1)/2 = 15$ links

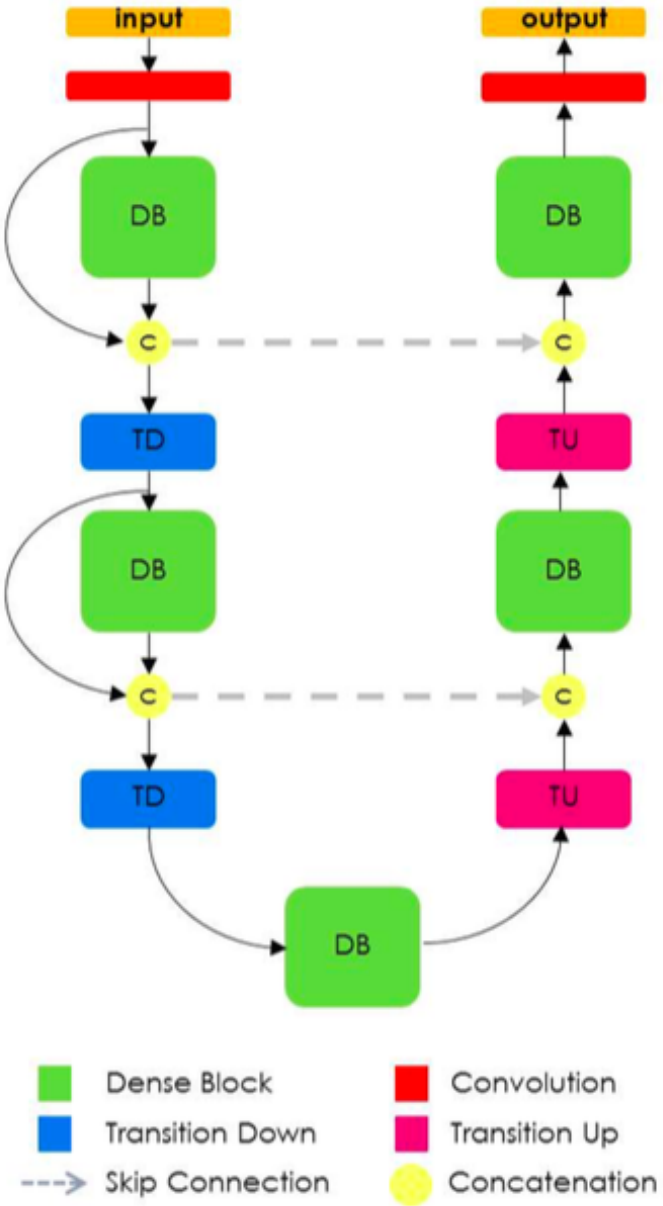
Receiving the feature maps of all preceding layers
The ℓ layer has $k \times (\ell - 1) + k_0$ input feature maps and produces k outputs.
Bottleneck layers: BN-ReLU-Conv(1×1)-BN-ReLU-Conv(3×3)

Case Study: The One Hundred Layers Tiramisu

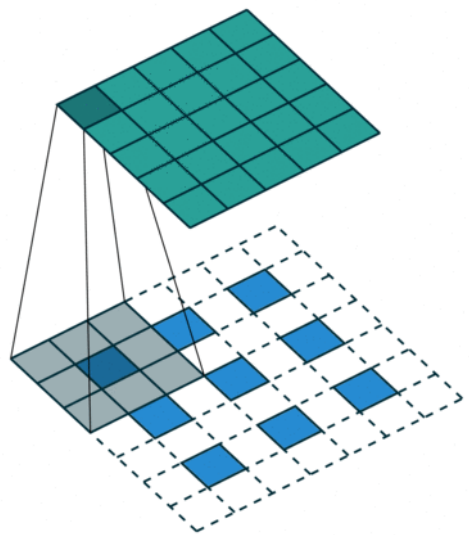
Developed by Jégou et al.

Figure 2: Fully convolutional DenseNets for semantic segmentation. From left to right: input image, ground truth, prediction result.

Tiramisu: Transition Down/Transition Up



Transposed Convolutions (Fractional Strides for Upsampling)

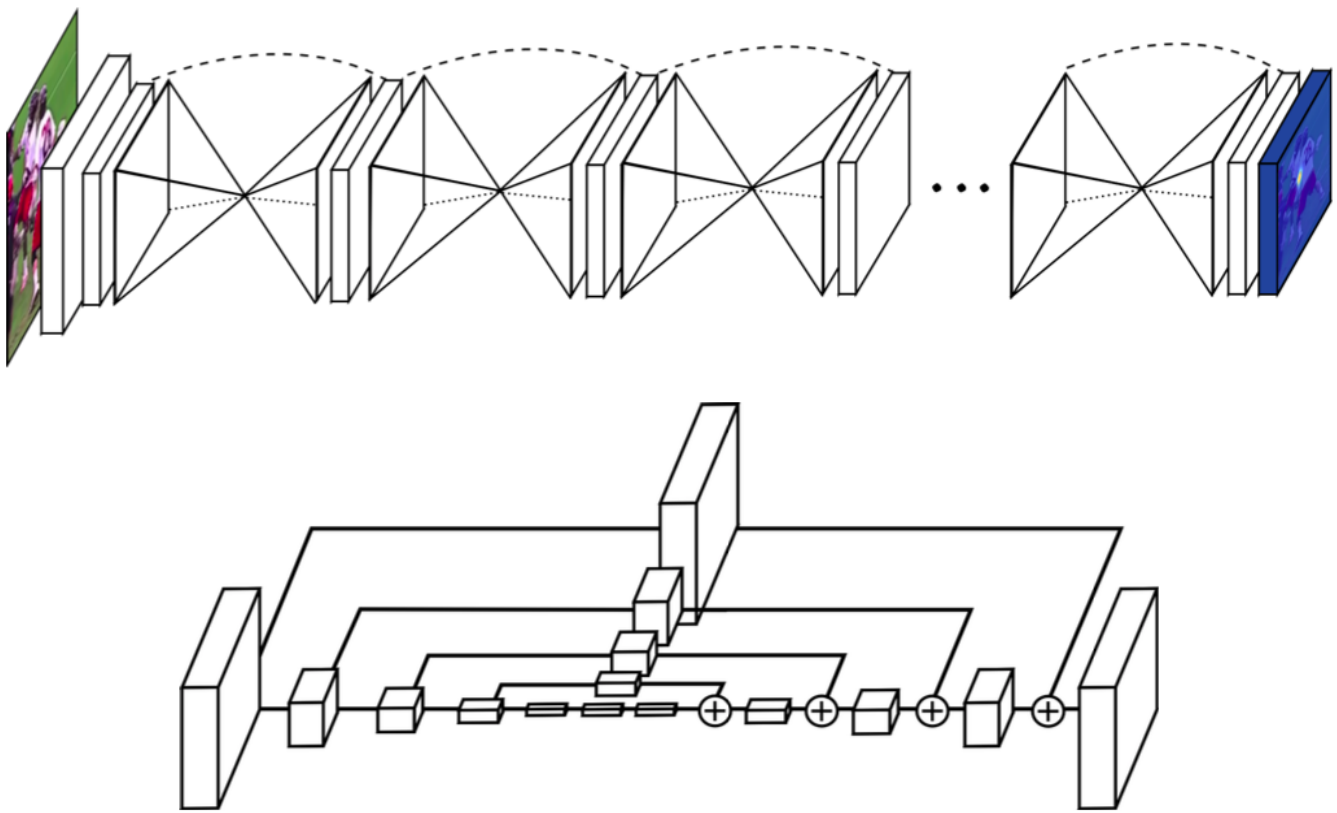


`https://github.com/vdumoulin/conv_arithmetic`

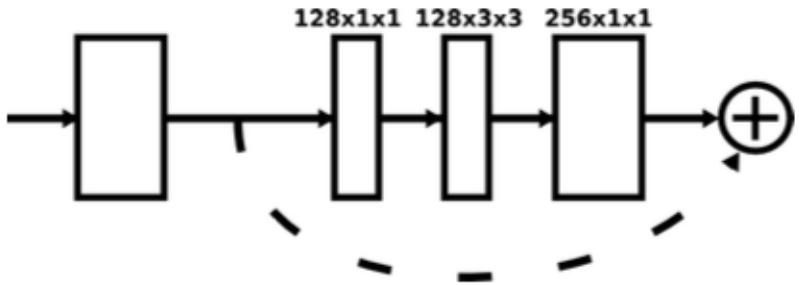
Case Study: Stacked Hourglass Networks

Developed by Newell et al.

State-of-the-art human pose estimator



Stacked Hourglass Networks



3x3 convolutions
Max-pooling for downsampling
Nearest neighbor upsampling
Residual modules
Two rounds of 1x1 convolutions at the end

Results of Human Pose Estimation

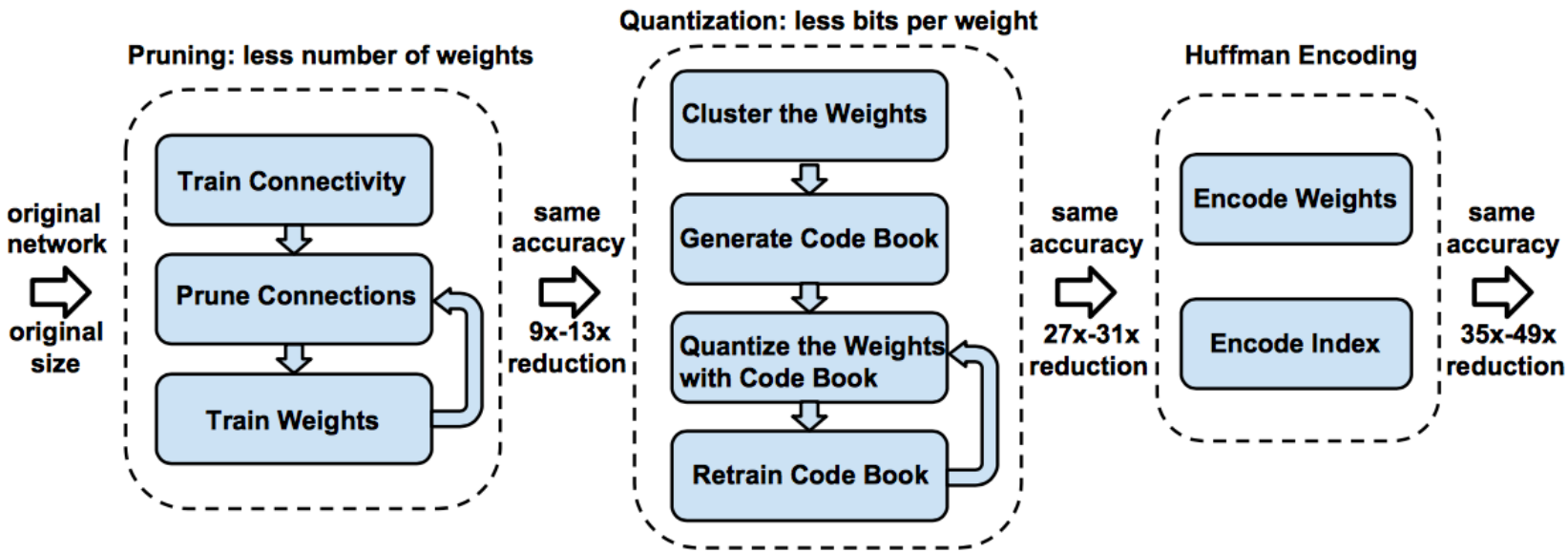
Case Study: Object Detection

R-CNN, Fast R-CNN, Faster R-CNN, YOLO, YOLO9000, SSD, FPN, RetinaNet, Mask R-CNN, ...

(Switch to Google slides)

Case Study: Deep Compression

Han et al., ICLR 2016



Other recent methods, e.g., XNOR-Net (Rastegari et al.), MobileNet, Network Slimming