Universität Augsburg University

Deep Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs / ConvNets) - Outline

- Architecture Overview
- ConvNet Layers
  - Convolutional Layer
  - Pooling Layer
  - Normalization Layers
- ConvNet Architectures
  - Layer Patterns
  - Case Studies (AlexNet / VGGNet)
  - ResNet

Architecture Overview (1)

Universität Augsburg University

Source: http://cs231n.github.io/convolutional-networks/

ConvNet architectures make the explicit assumption that the inputs are images (= 3D volumes of neurons)

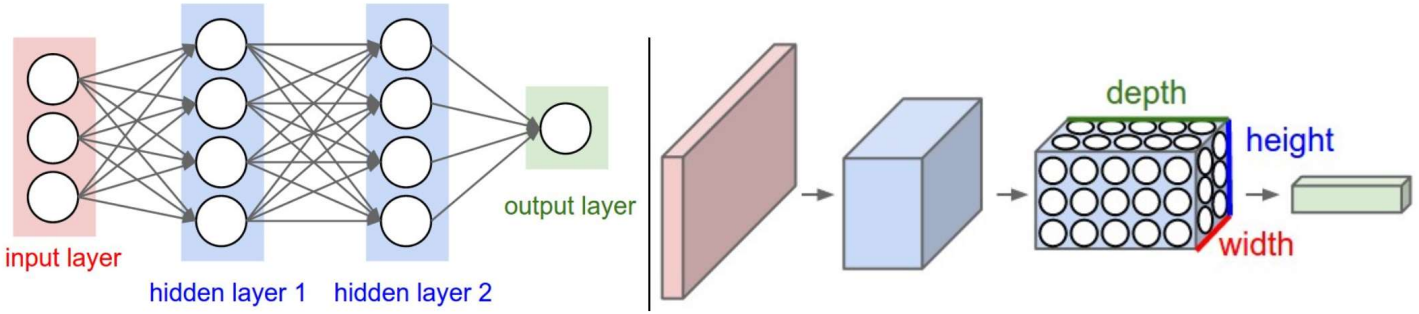
➔allows us to encode certain properties into the architecture.

➔Arrange neurons of layer in 3 dimensions: width, height, depth (= 3D volume of neurons)

➔ make forward function more efficient to implement and vastly reduce the amount of parameters in the network.

Architecture Overview (2)

Source: http://cs231n.github.io/convolutional-networks/



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Assume: Input: 200x200x3 RGB image

- Fully connected NN: => 120 000 * (# of neurons in first hidden layer) weight parameters
- CNN with 3x3x3 convolution: => only 27 weight parameters in total (independent of # of neurons in first hidden layer)

Universität Augsburg University

ConvNet Layers

Convolutional Layer (1)

Universität Augsburg University

Source: http://cs231n.github.io/convolutional-networks/

- Connects each output neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (equivalently this is the filter size)
- The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space (along width and height), but always full along the entire depth of the input volume.

Convolutional Layer (2) - Hyperparameter

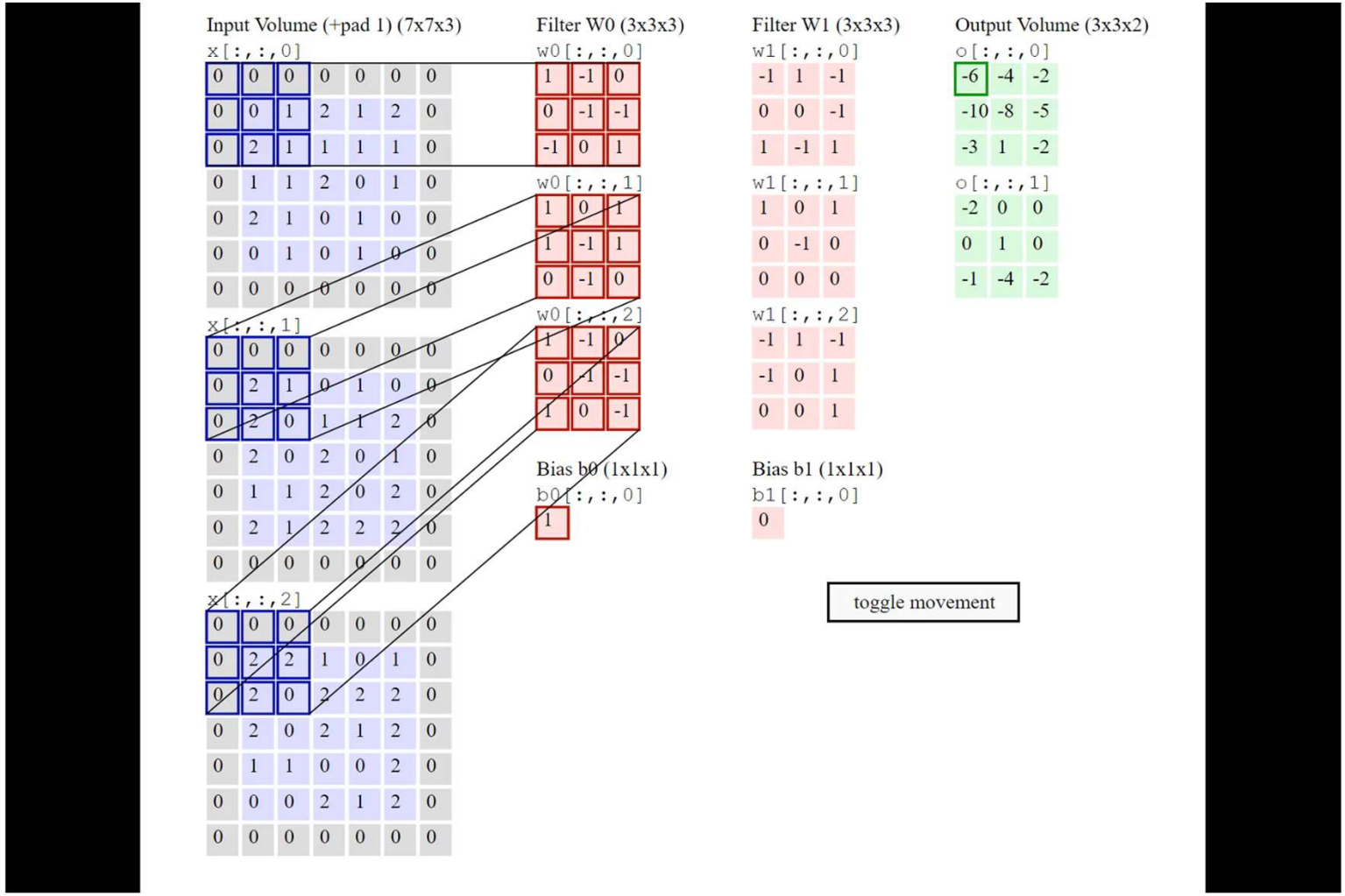
Universität Augsburg University

Source: http://cs231n.github.io/convolutional-networks/

- **Receptive field size (*F*)** of the Conv. Layer neurons
- **Depth of the output volume**: number of filters we would like to use, each learning to look for something different in the input.
- **Stride (*S*)**: $stride == n$ → filters move n pixels at a time. This will produce smaller output volumes spatially for $n > 1$.
- **Amount (*P*) of zero-padding:** Allows to control the spatial size of the output volumes; most commonly used to exactly preserve the spatial size of the input volume so the input and output width and height are the same)

Convolutional Layer (3) - Animation

Source: http://cs231n.github.io/convolutional-networks/



Valid Convolution

Source: Vincent Dumoulin, and Francesco Visin. *A guide to convolution arithmetic for deep learning*. ArXiv 1603.07285, 2016}

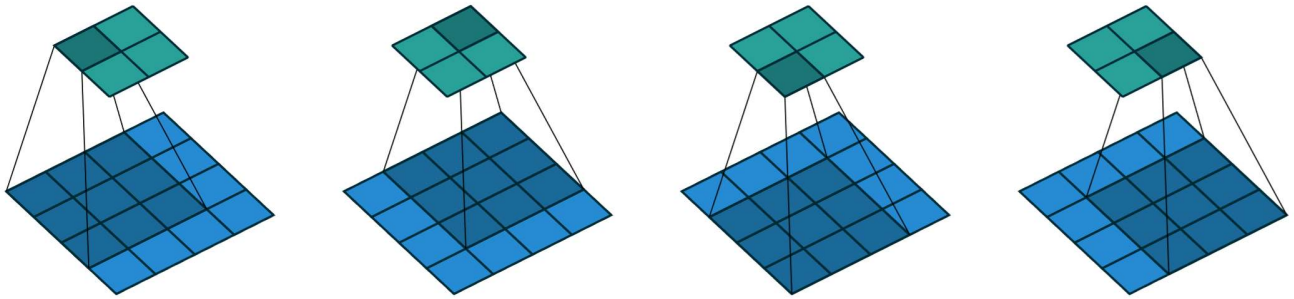


Figure 2.1: (No padding, unit strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

The kernel starts on the leftmost part of the input feature map and slides by steps of one until it touches the right side of the input. The size of the output will be equal to the number of steps made, plus one, accounting for the initial position of the kernel.

For any i and k, and for $s = 1$ and $p = 0$,

$$o = (i - k) + 1$$

Same Convolution

Source: Vincent Dumoulin, and Francesco Visin. *A guide to convolution arithmetic for deep learning*. ArXiv 1603.07285, 2016}

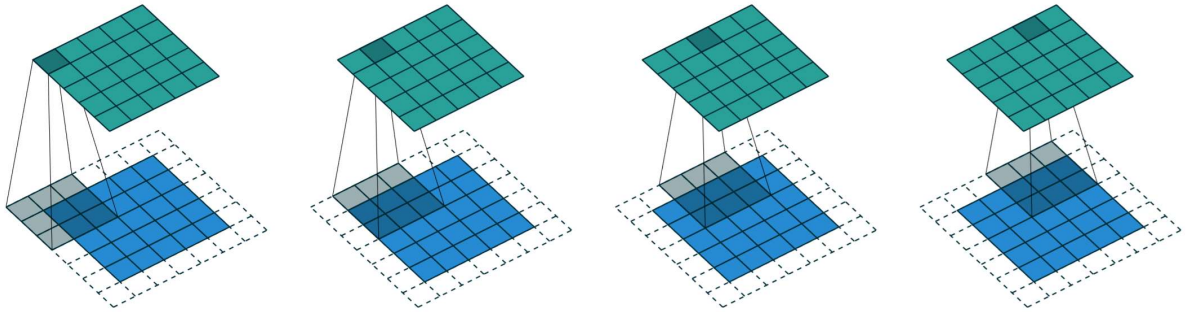


Figure 2.3: (Half padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$).

Having the output size be the same as the input size (i.e., $o = i$) can be a desirable property.

For any i and for k odd ($k = 2n + 1, n \in N$), $s = 1$ and $p = \lfloor k/2 \rfloor = n$,

$$\begin{aligned} o &= i + 2\lfloor k/2 \rfloor - (k-1) \\ &= i + 2n - 2n \\ &= i \end{aligned}$$

Full Convolution

Source: Vincent Dumoulin, and Francesco Visin. *A guide to convolution arithmetic for deep learning*. ArXiv 1603.07285, 2016}

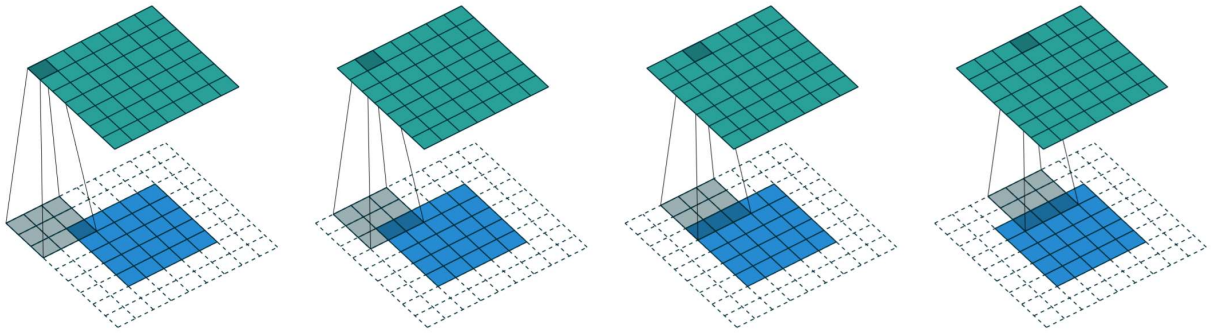


Figure 2.4: (Full padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$).

This is sometimes referred to as full padding, because in this setting every possible partial or complete superimposition of the kernel on the input feature map is taken into account:

For any i and k, and for $p = k - 1$ and $s = 1$,

$$o = i + 2(k-1) - (k-1) = i + (k-1)$$

„Deconvolution“ = Transposed Convolution

Universität Augsburg University

- The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution.

Max Pooling Layer (1)

Universität Augsburg University

Source: http://cs231n.github.io/convolutional-networks/

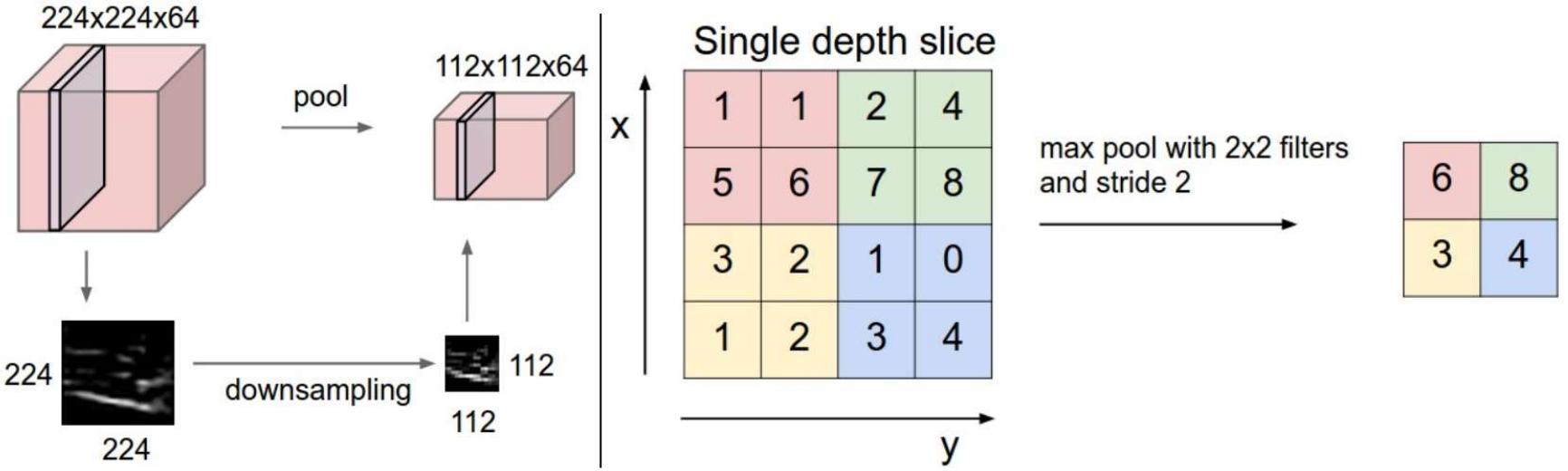
It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down-samples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. The depth dimension remains unchanged.

Hyperparameters

- spatial extent ***F***,
- stride ***S***

Max Pooling Layer (2)

Source: http://cs231n.github.io/convolutional-networks/



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Universität Augsburg University

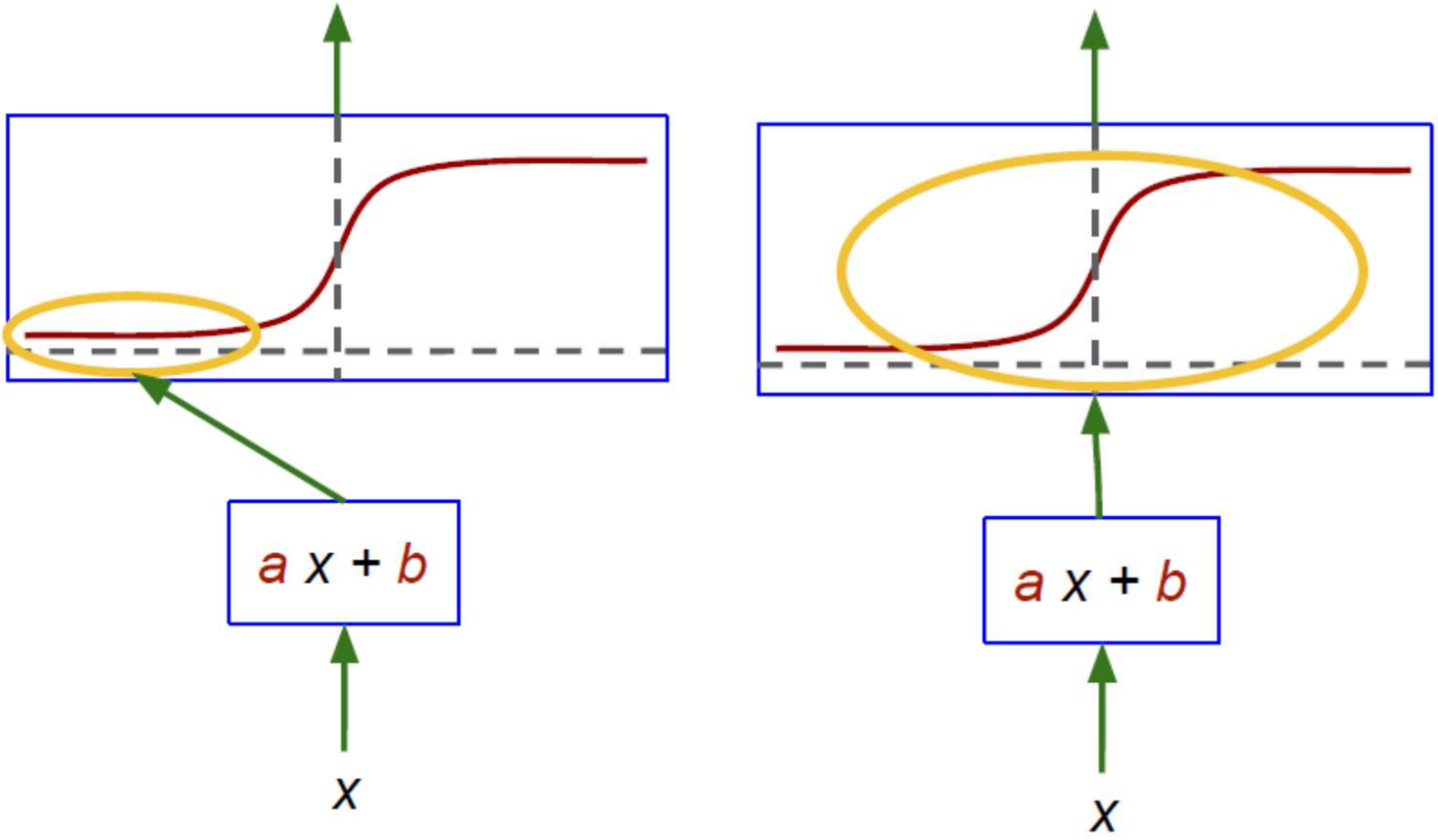
Normalization Layers

References

@article{GN2018,
author = {Yuxin Wu and Kaiming He},
title = {**Group Normalization**},
journal = {CoRR},
volume = {abs/1803.08494},
year = {2018},
url = {http://arxiv.org/abs/1803.08494}}

@article{BN2015,
author = {Sergey Ioffe and Christian Szegedy},
title = {**Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift**},
journal = {CoRR},
volume = {abs/1502.03167},
year = {2015},
url = {http://arxiv.org/abs/1502.03167}}

Batch Normalization (1) - Motivation



Batch Normalization (2) - Goal

Universität Augsburg University

Ref: Sergey Ioffe and Christian Szegedy. **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift**. 2015, arXiv:1502.03167

Goal: Normalize distribution of each input feature in each layer across each mini-batch to $N(0,1)$. Then, find a single scaling and offset value.

Batch Normalization (3) - Learning

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

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

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

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

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

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

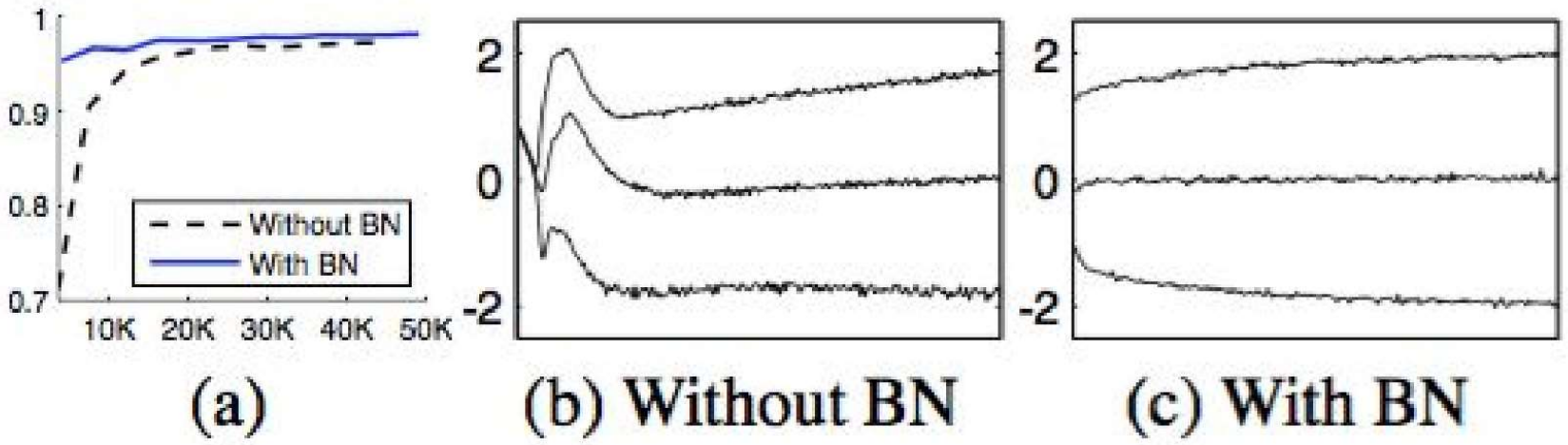
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Replace mini-batch statistics by population statistics:

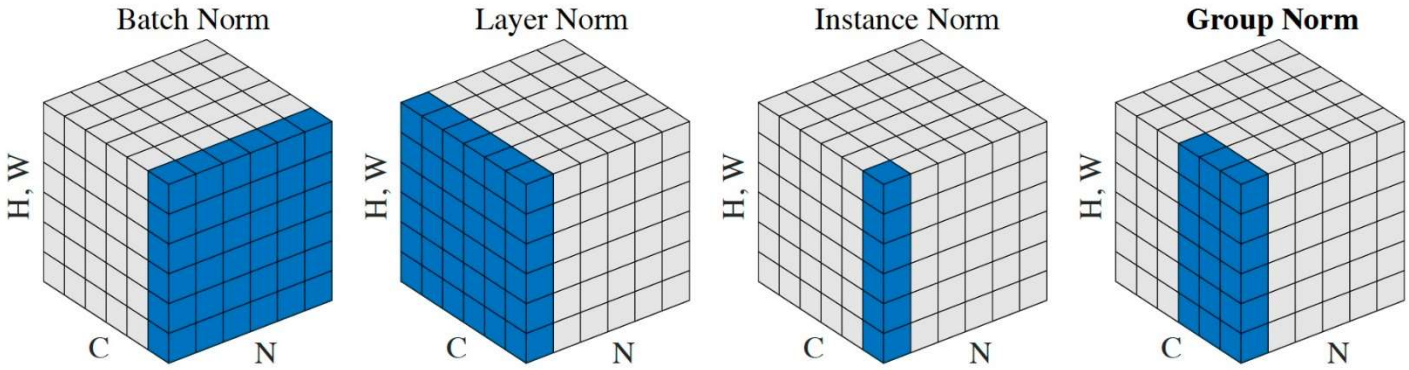
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad \Rightarrow \quad \widehat{x} \leftarrow \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}$$

Batch Normalization (5) - Observations

Much faster convergence during training (e.g., 30x on ImageNet)



GN – Group Normalization



1. Compute mean and standard deviation over value set S (BN, LN, IN, GN) and transform each value x in value set S:

$$\hat{x}_i = \frac{1}{\sigma_S}(x_i - \mu_S)$$

2. Per channel linear transform to compensate for possible loss of representational power

$$y_i = \gamma \hat{x}_i + \beta$$

Shift and scale is learned per channel (pair of values for each $WxHxN$ group)

Python code of GN based on TF

Universität Augsburg University

```
def GroupNorm(x, gamma, beta, G, eps=1e-5):
    # x: input features with shape [N,C,H,W]
    # gamma, beta: scale and offset, with shape [1,C,1,1]
    # G: number of groups for GN

    N, C, H, W = x.shape
    x = tf.reshape(x, [N, G, C // G, H, W])

    mean, var = tf.nn.moments(x, [2, 3, 4], keep_dims=True)
    x = (x - mean) / tf.sqrt(var + eps)

    x = tf.reshape(x, [N, C, H, W])

    return x * gamma + beta
```

ConvNet Architectures

Architecture Overview (3)

Universität Augsburg University

Source: http://cs231n.github.io/convolutional-networks/

We use three main types of layers to build ConvNet architectures:

- Convolutional Layer
- Pooling Layer and
- Fully-Connected Layer (exactly as in regular NN).

We will stack these layers to form a full ConvNet architecture.

Architecture Overview (3) - Example

Universität Augsburg University

Source: http://cs231n.github.io/convolutional-networks/

INPUT - CONV - RELU - POOL - FC

- INPUT [32x32x3]; raw pixel values of the image
- CONV layer such as [32x32x12] if we decided to use 12 filters.
- RELU layer with elementwise activation function of max(0,x); volume unchanged ([32x32x12]).
- POOL layer; performs a downsampling resulting in volume [16x16x12] (max pooling)
- FC (i.e. fully-connected) layer; compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score.

Architecture Overview - Summary

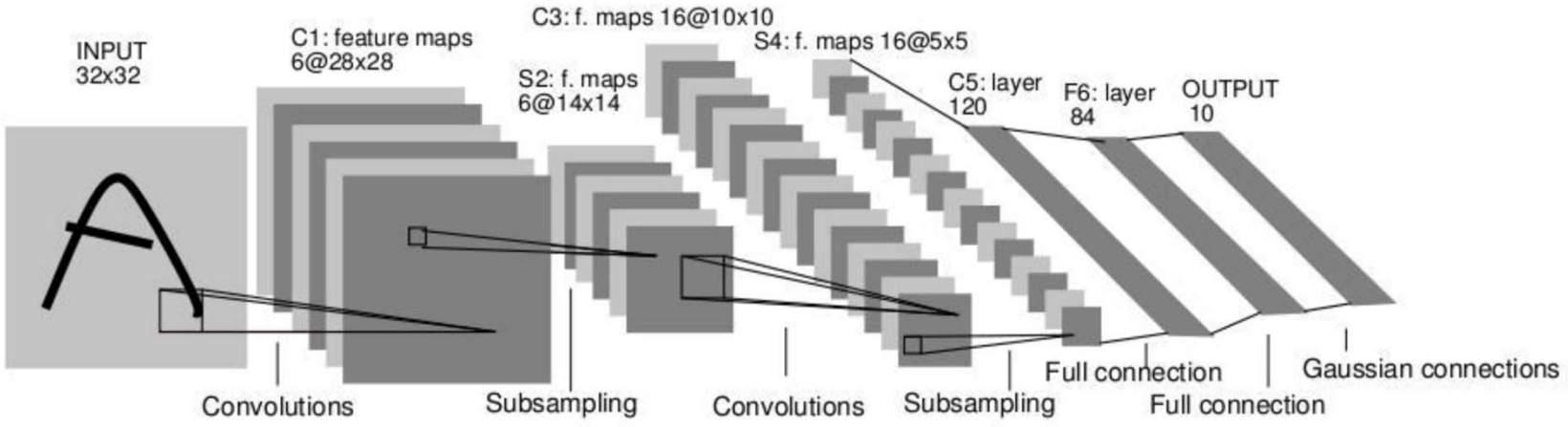
Universität Augsburg University

Source: http://cs231n.github.io/convolutional-networks/

- A ConvNet architecture is in the simplest case a list of layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- 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)
- Each layer may or may not have additional hyper-parameters (e.g. CONV/FC/POOL do, RELU doesn't)

Convolutional Networks
LeCun et al. 1989

- Convolutional layer is a restricted standard layer
- At that time: trained with limited data

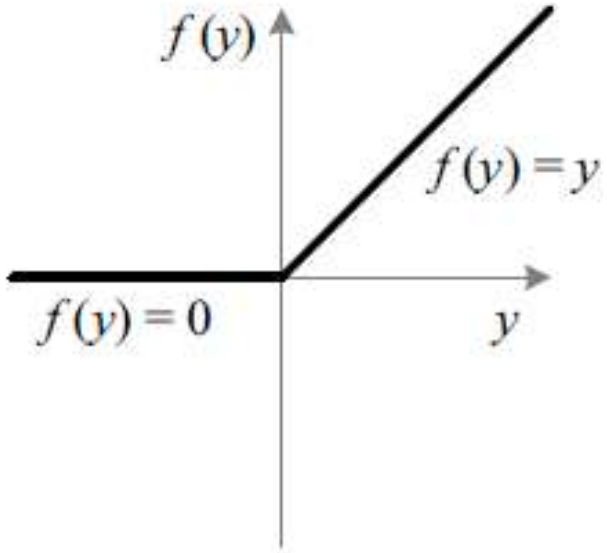


Novelities – Some new tweaks (1)

- Revival in 2012 with some new tweaks
- New thresholding function: ReLU
  - Faster to compute
  - Easy derivative:

$$ReLU(x) = \max(0, x)$$

  - Biologically motivated
  - Observed substantially faster training convergence



Novelities – Some new tweaks (2)

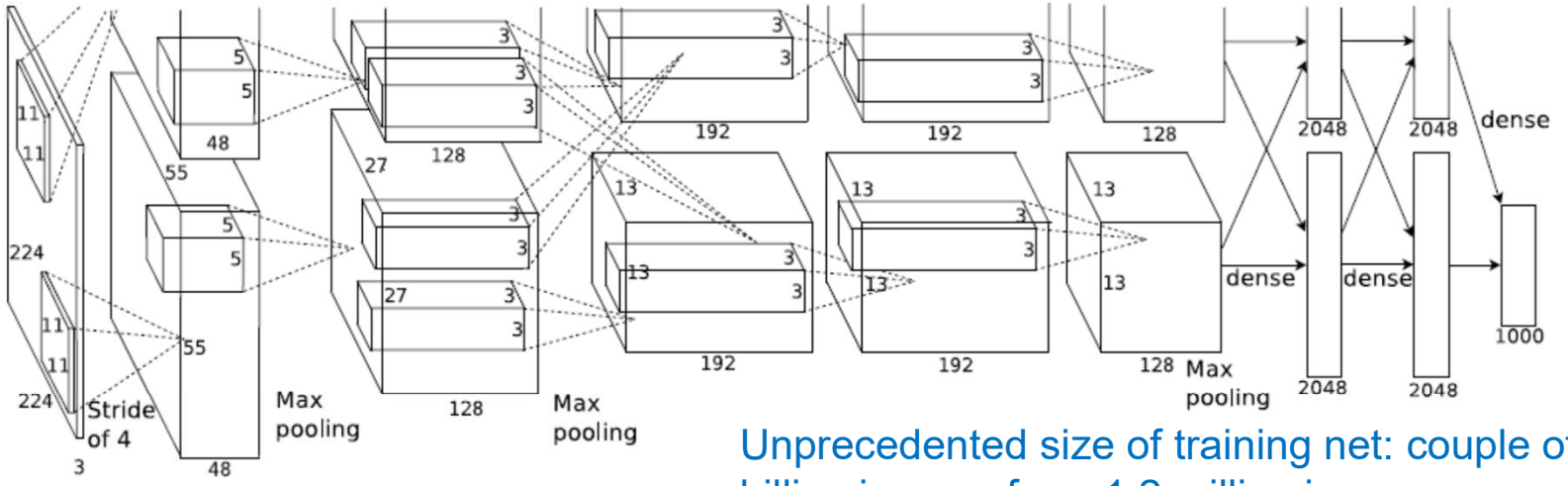
Universität Augsburg University

- Learn with drop-outs in fully-connected layers
  - *Drop-Out Weights* = $x\%$ of the weights are randomly set to zero
  - *Drop-Out Activations* = $x\%$ of the activations are set to zero

  $\Rightarrow$ learns a family of networks
- Max-Pooling layers
  - *Max-Pooling* = *max-convolution* = Going with a sliding window of stride x over a spatial layer and retrieve the maximum value

  => Incorporating (partially) spatial invariance

Revival 2012 – AlexNet (1)



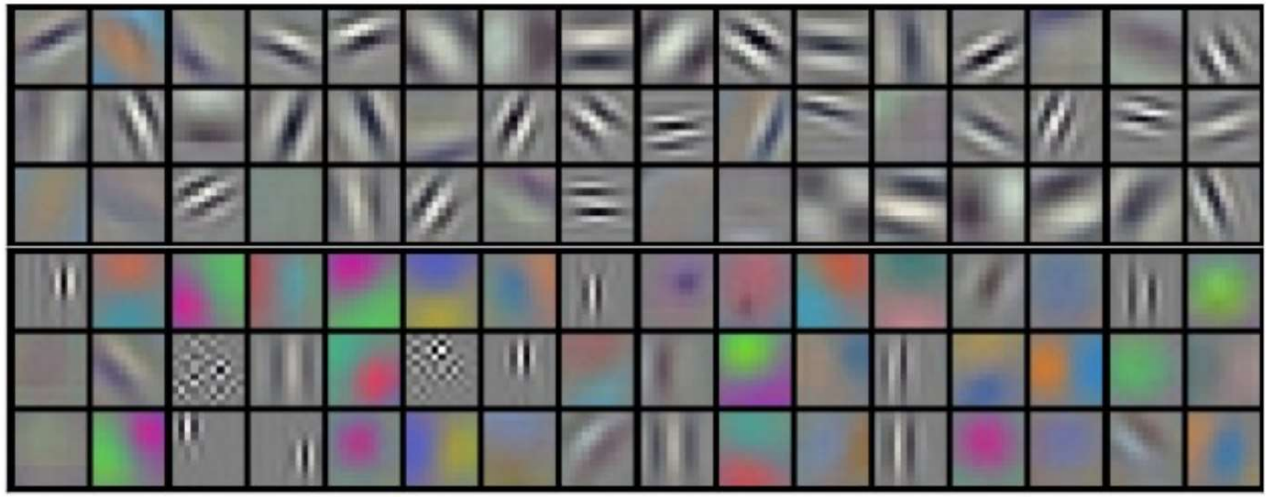
Unprecedented size of training net: couple of billion images from 1.2 million images

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.

Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E.: Imagenet classification with deep convolutional neural networks. In: *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, S. 2012

Revival 2012 – AlexNet (2)

Source: http://cs231n.github.io/convolutional-networks/



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55*55 distinct locations in the Conv layer output volume.

VGG-16 – 2014 (1)

Universität Augsburg University

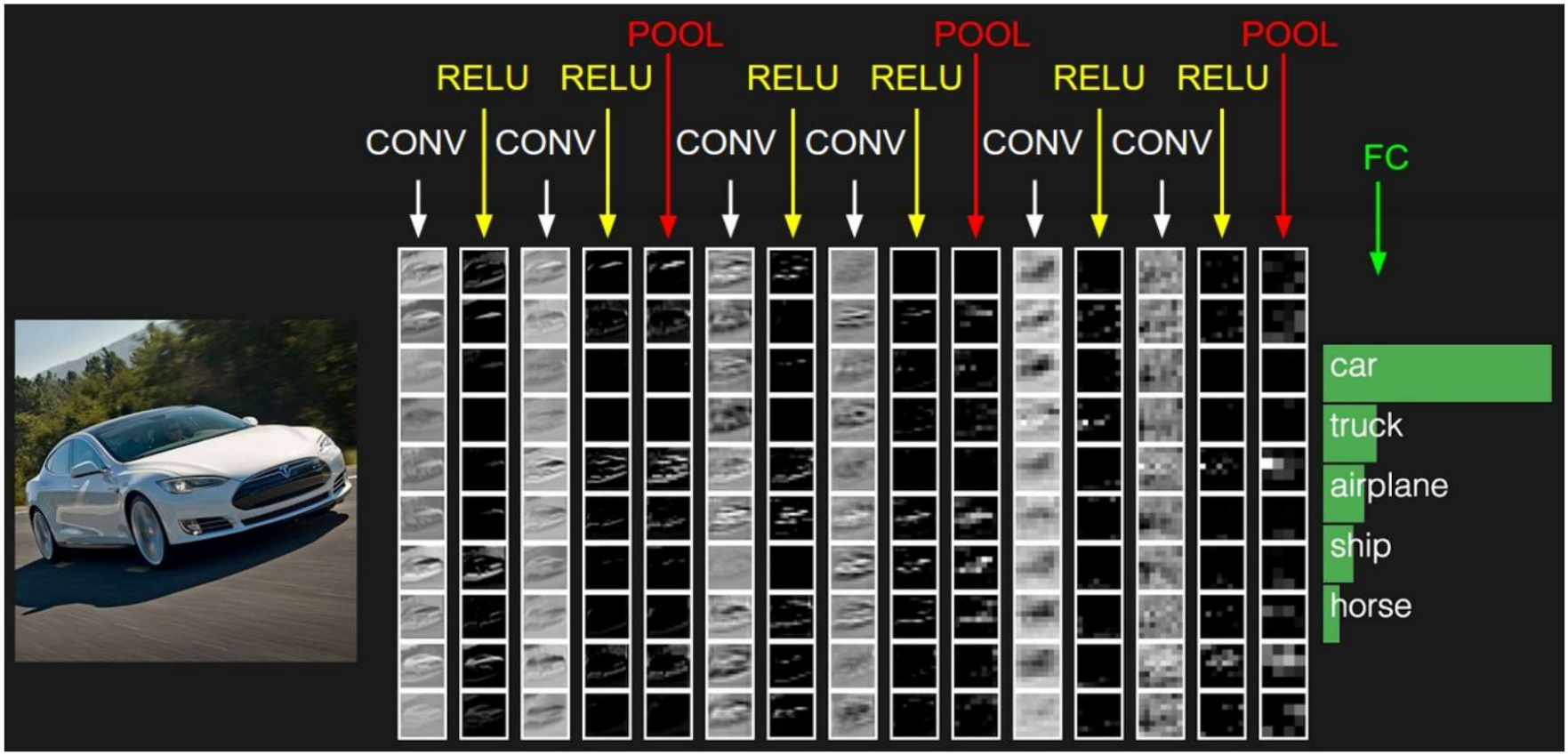
Source: http://cs231n.github.io/convolutional-networks/

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

VGG-16 – 2014 (2)

Source: http://cs231n.github.io/convolutional-networks/

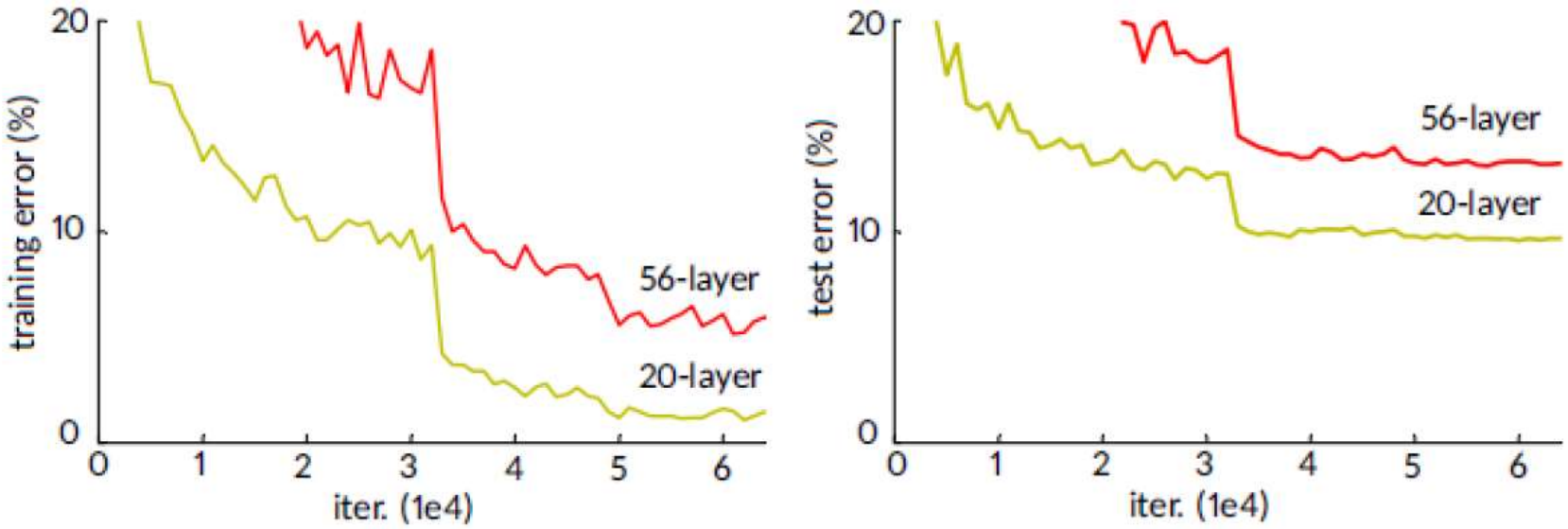


The activations of an example ConvNet architecture. The initial volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one. The full web-based demo is shown in the header of our website. The architecture shown here is a tiny VGG Net, which we will discuss later.

Novelities – Some new tweaks (3)

Source: „Exploiting deep residual networks for human action recognition from skeletal data“ (2018)

Observation



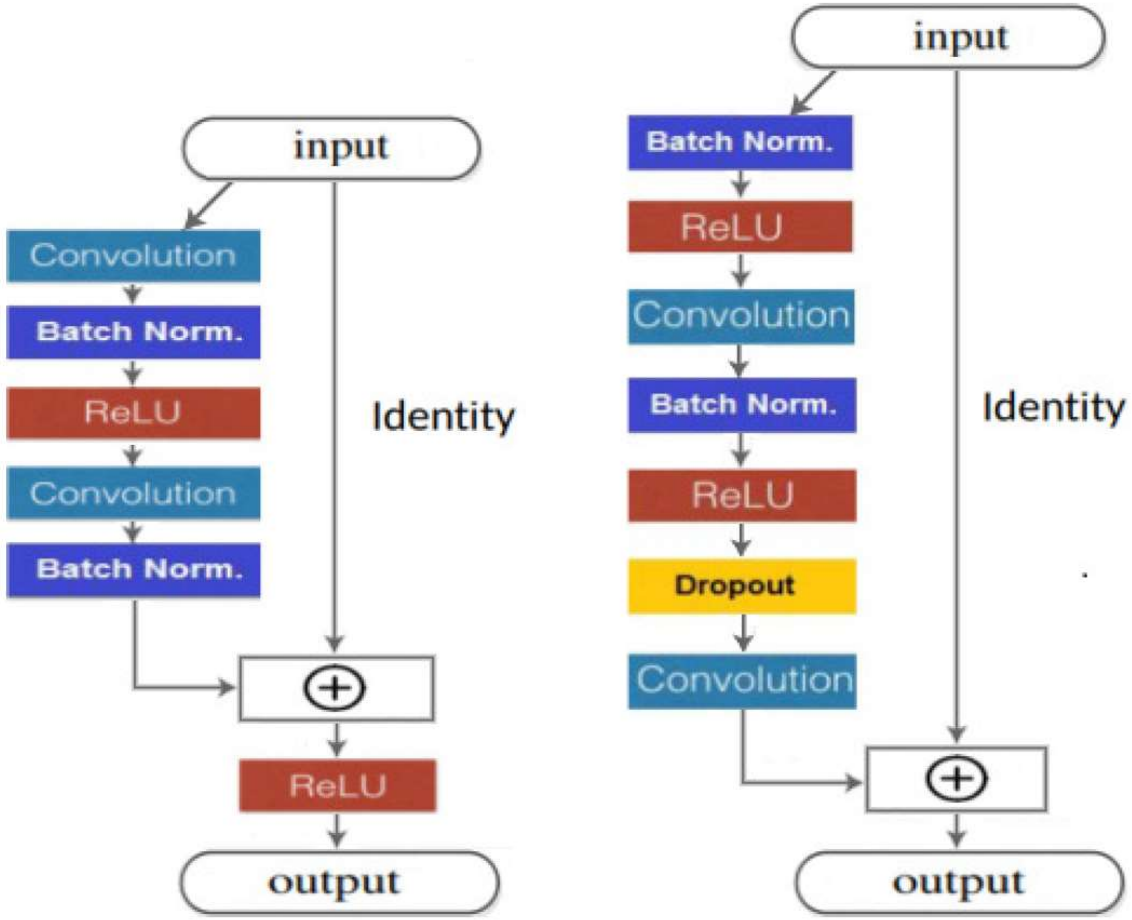
Novelities – Some new tweaks (3)

Source: „Exploiting deep residual networks for human action recognition from skeletal data“ (2018)

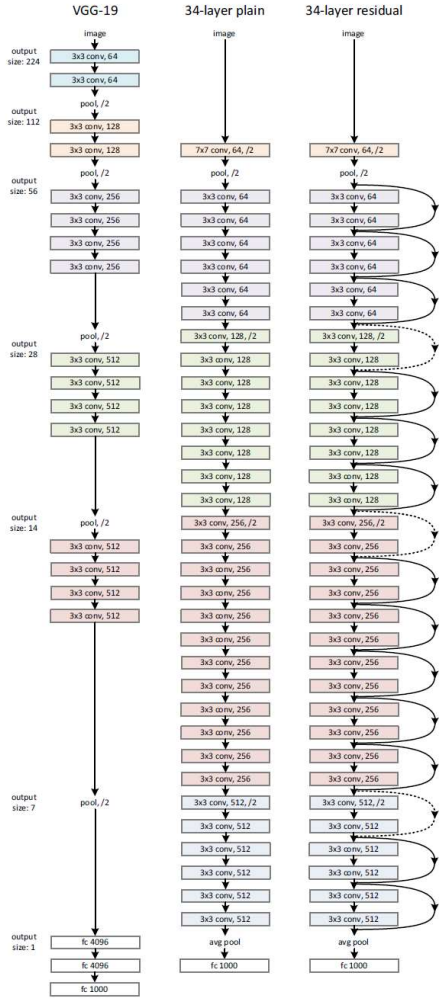
Solution:

ResNet Buildung Unit:

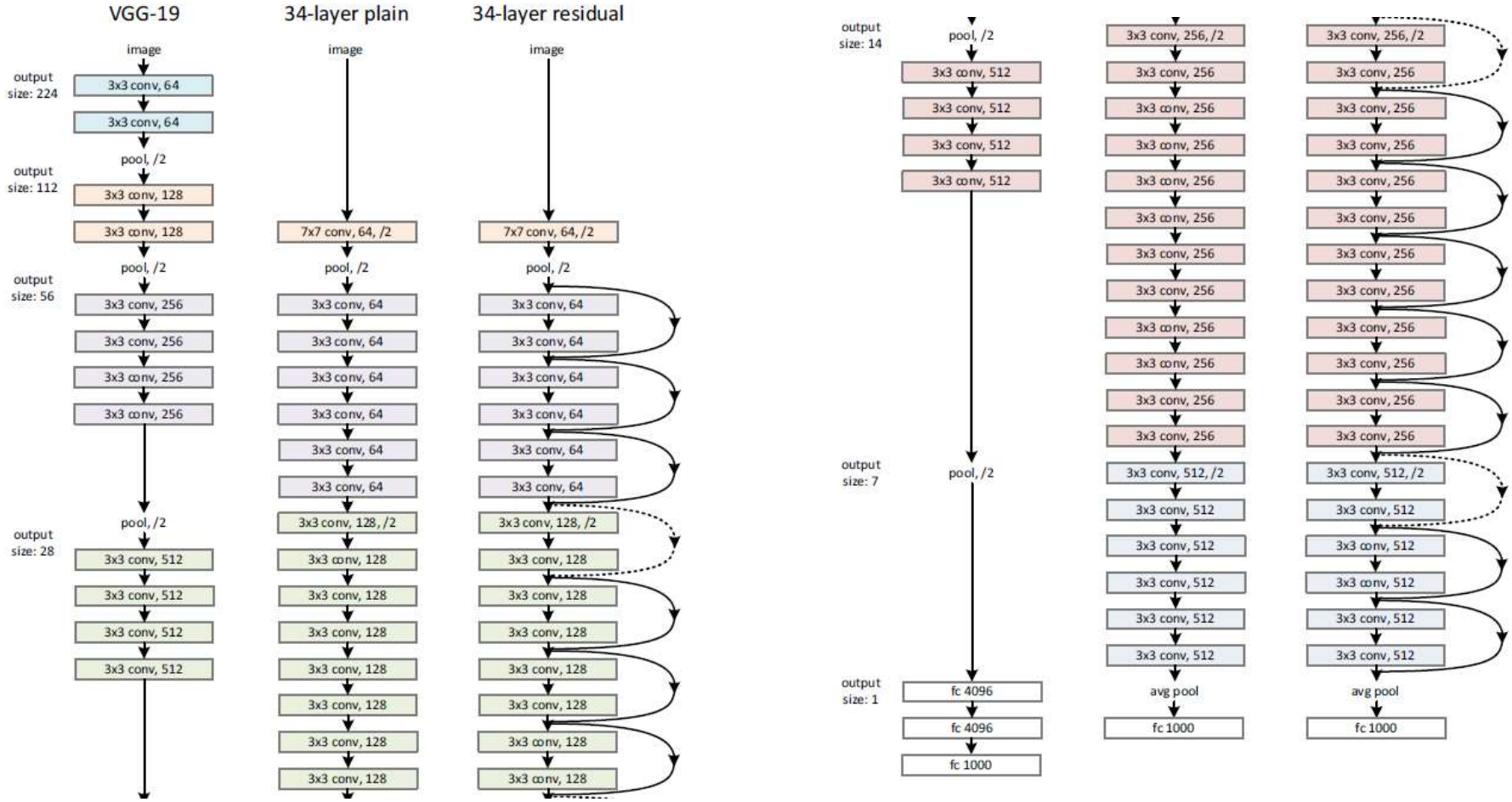
Learn on residual only



State-of-the-Art 2015



State-of-the-Art 2015 – Zoomed In



Convolutional Layer

Universität Augsburg University

- How would you train it?
- How would modify the backpropagation algorithm?