



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Activation Functions and Convolutional Neural Networks

A. Maier, K. Breininger, L. Mill, N. Ravikumar, T. Würfel

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 2, 2018



Outline

Activation Functions

Convolutional Neural Networks

Convolutional Layers

Pooling Layers



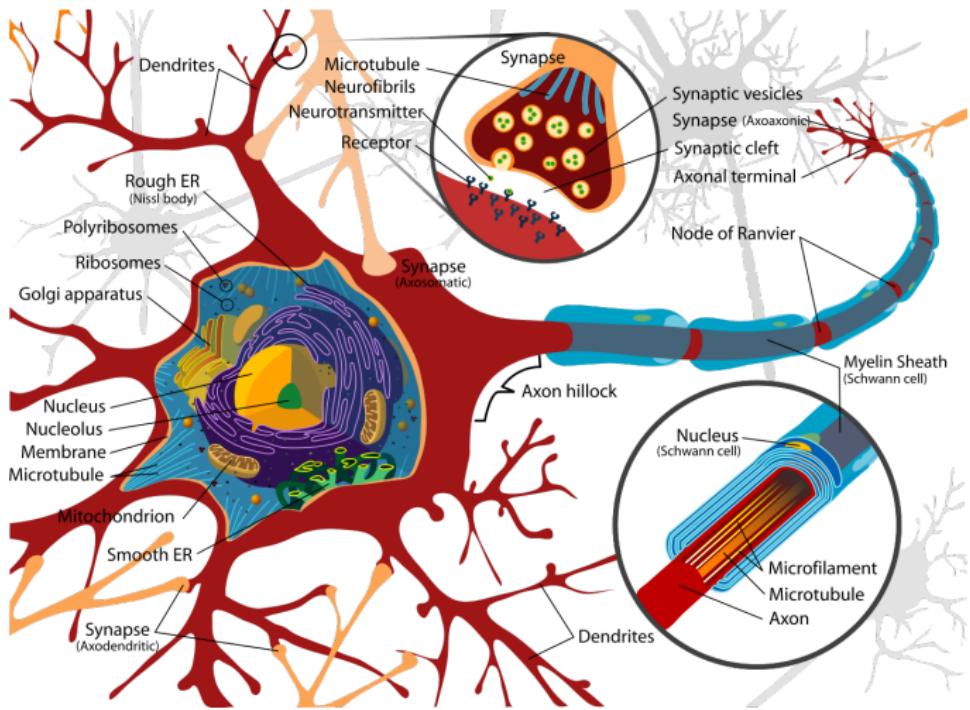
FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

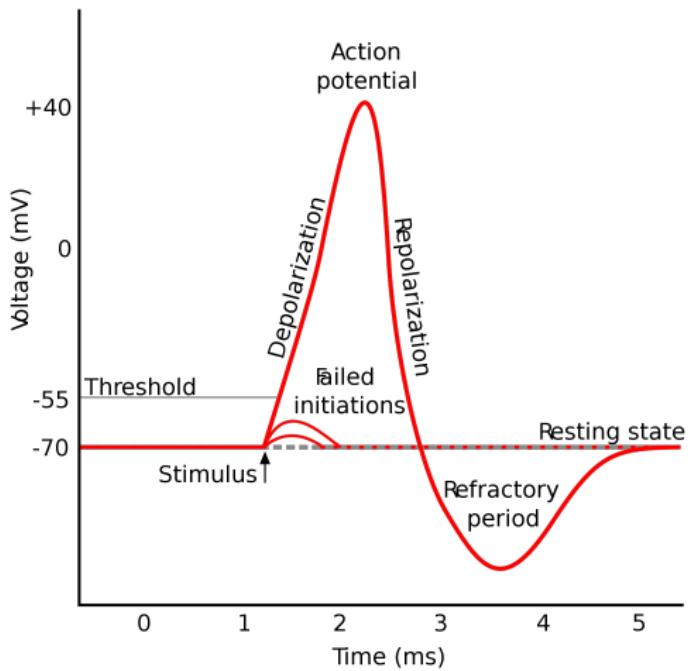
Activation Functions



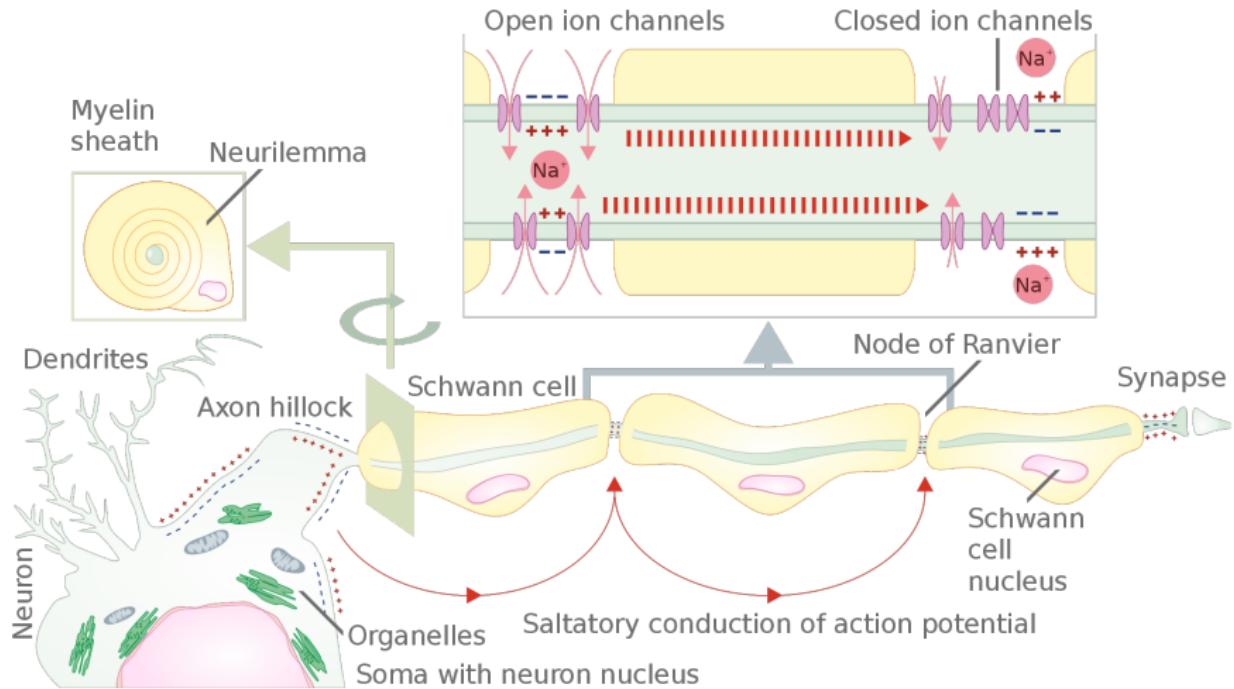
Biological Activation



Biological Activation



Biological Activation



Summary - Activations in Biological Neurons

- The knowledge lies in the **connections**
- Both **inhibitory** and **excitatory** connections exist
- The synapses anatomically enforce **feed forward** processing
- However, the connections can be in any direction
- The **sum** of activations is crucial
- Activations are electric spikes
 - With **specified intensity**
 - Which also encode **information over time**

Activations in Artificial Neural Networks so far

- Non-linear activation functions enable **universal function approximation**:
→ Highly important for a powerful network

Activations in Artificial Neural Networks so far

- Non-linear activation functions enable **universal function approximation**:
→ Highly important for a powerful network
- Compared to biology:
 - ✓ Heaviside step (sign) function can model **all or nothing response**
 - ✗ Artificial activations have **no time component** (model by activation strength?)

Activations in Artificial Neural Networks so far

- Non-linear activation functions enable **universal function approximation**:
→ Highly important for a powerful network
- Compared to biology:
 - ✓ Heaviside step (sign) function can model **all or nothing response**
 - ✗ Artificial activations have **no time component** (model by activation strength?)
- Sign function is mathematically undesirable:

$$f'(x) = \begin{cases} \infty & \text{for } x = 0 \\ 0 & \text{else} \end{cases}$$

↳ back-propagation

Activations in Artificial Neural Networks so far

- Non-linear activation functions enable **universal function approximation**:
 → Highly important for a powerful network
- Compared to biology:
 - ✓ Heaviside step (sign) function can model **all or nothing response**
 - ✗ Artificial activations have **no time component** (model by activation strength?)
- Sign function is mathematically undesirable:

$$f'(x) = \begin{cases} \infty & \text{for } x = 0 \\ 0 & \text{else} \end{cases}$$

↳ back-propagation

- So far: Sigmoid-function $f(x) = \frac{1}{1+\exp(-x)}$

Activations in Artificial Neural Networks so far

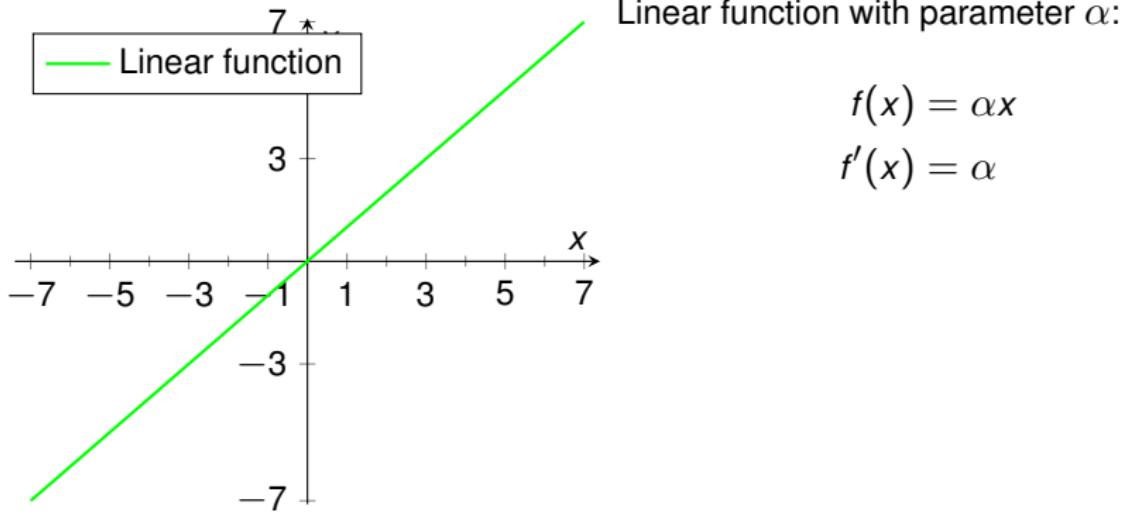
- Non-linear activation functions enable **universal function approximation**:
→ Highly important for a powerful network
- Compared to biology:
 - ✓ Heaviside step (sign) function can model **all or nothing response**
 - ✗ Artificial activations have **no time component** (model by activation strength?)
- Sign function is mathematically undesirable:

$$f'(x) = \begin{cases} \infty & \text{for } x = 0 \\ 0 & \text{else} \end{cases}$$

↳ back-propagation

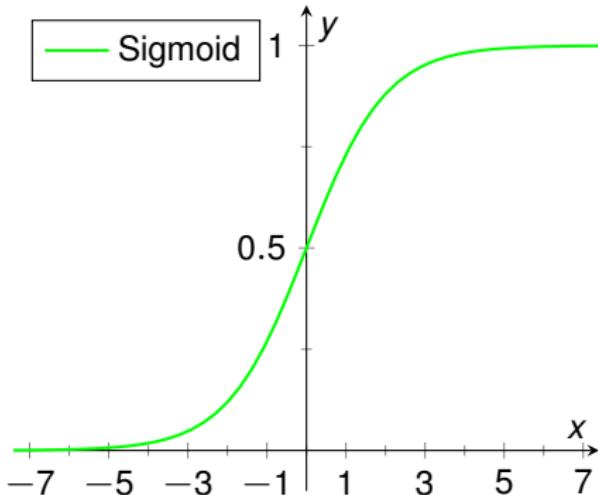
- So far: Sigmoid-function $f(x) = \frac{1}{1+\exp(-x)}$
- Can we do better?

Linear Activation Function



- + Simple
- Does not introduce non-linearity
- + Therefore, it renders the optimization problem convex
- Listed here mainly for completeness

Sigmoid Activation Function



Sigmoid (logistic function)

$$f(x) = \frac{1}{1 + \exp(-x)}$$

$$f'(x) = f(x)(1 - f(x))$$

- Close to biological model, but differentiable
- + Probabilistic output
- Saturates for $x \ll 0$ and $x \gg 0$
- Not zero-centered

Zero-Centering

- Sigmoid: $f : \mathbb{R} \mapsto]0, 1[$

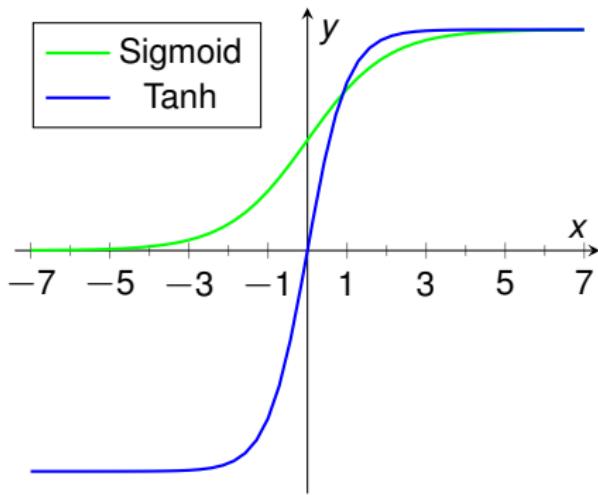
Zero-Centering

- Sigmoid: $f : \mathbb{R} \mapsto]0, 1[$
- Output of activation always +
 - ∇_w will either be all + or all -

Zero-Centering

- Sigmoid: $f : \mathbb{R} \mapsto]0, 1[$
- Output of activation always +
 - ∇_w will either be all + or all -
- A mean $\mu = 0$ of the input distribution will always be shifted to $\mu > 0$
 - **co-variate shift** of successive layers
 - layers **constantly** have to **adapt** to the shifting distribution
- Batch learning reduces the variance σ of the updates

Tanh Activation Function



Tanh

$$f(x) = \tanh(x)$$

$$f'(x) = 1 - f(x)^2$$

- + Zero-centered (LeCun '91)
- Shifted version of sigmoid σ : $\tanh(x) = 2\sigma(2x) - 1$
 - Still saturates
 - Still causes vanishing gradients

Vanishing and Exploding Gradients

- Essence of learning: How does x affect y ?
- Sigmoid/tanh map **large regions** of X to a **small range** in Y

Vanishing and Exploding Gradients

- Essence of learning: How does x affect y ?
- Sigmoid/tanh map **large regions** of X to a **small range** in Y
- **Large changes** in $x \mapsto$ minimal changes in y
 - gradient vanishes

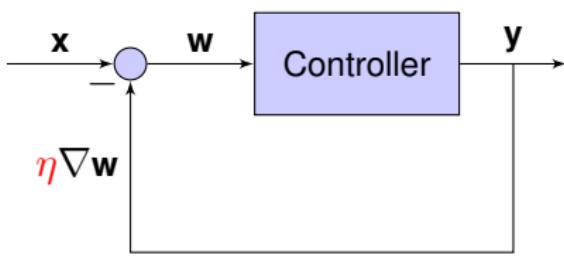
Vanishing and Exploding Gradients

- Essence of learning: How does x affect y ?
- Sigmoid/tanh map **large regions** of X to a **small range** in Y
- **Large changes** in $x \mapsto$ minimal changes in y
 - gradient vanishes
- Problem is amplified by back-propagation: Multiplication of small gradients

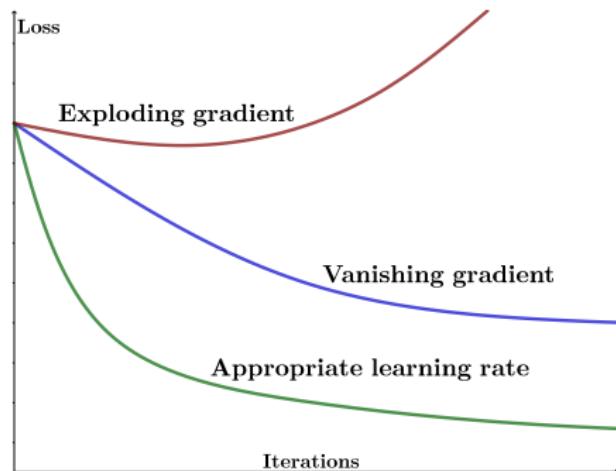
Vanishing and Exploding Gradients

- Essence of learning: How does x affect y ?
- Sigmoid/tanh map **large regions** of X to a **small range** in Y
- **Large changes** in $x \mapsto$ minimal changes in y
 - gradient vanishes
- Problem is amplified by back-propagation: Multiplication of small gradients
- Related problem: Exploding gradients

Recap: Feedback Loop - Vanishing and Exploding Gradients

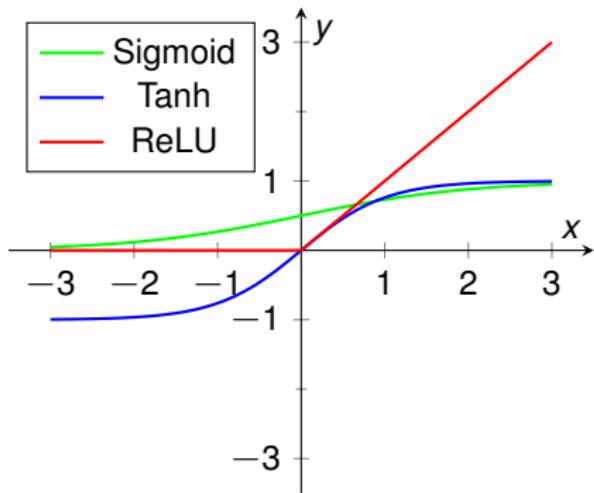


Analogy to control theory



- If η is too high \mapsto **positive feedback** \mapsto loss grows **without bounds**
- If η is too small \mapsto **negative feedback** \mapsto **gradient vanishes**
- Choice of η is **critical** for learning

Rectified Linear Units (ReLU)

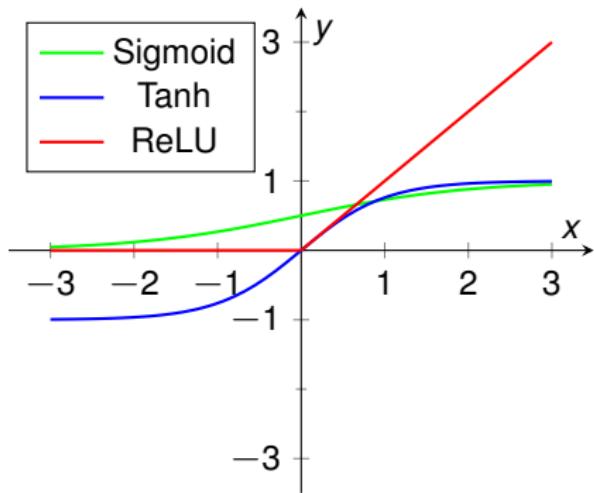


Rectified Linear Unit (ReLU)

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

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

Rectified Linear Units (ReLU)



Rectified Linear Unit (ReLU)

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

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

- + Good generalization due to piece-wise linearity
- + Speed up during learning ($\approx 6x$ (Krizhevsky '12))
- + No vanishing gradient problem
- Not zero-centered

More on ReLUs

- ReLUs were a **big step forward!**
- ReLUs enable training of **deep** supervised neural networks **without unsupervised pre-training**
- First derivative is 1 if the unit is active, second derivative is 0 almost anywhere
 - No second-order effects

Dying ReLUs



- Weight/biases trained to yield negative values for **any x**

Dying ReLUs



- Weight/biases trained to yield negative values for **any x**
- ReLU now only performs: $\mathbf{x} \mapsto 0$
- ReLU does not contribute to dividing the feature space

Dying ReLUs



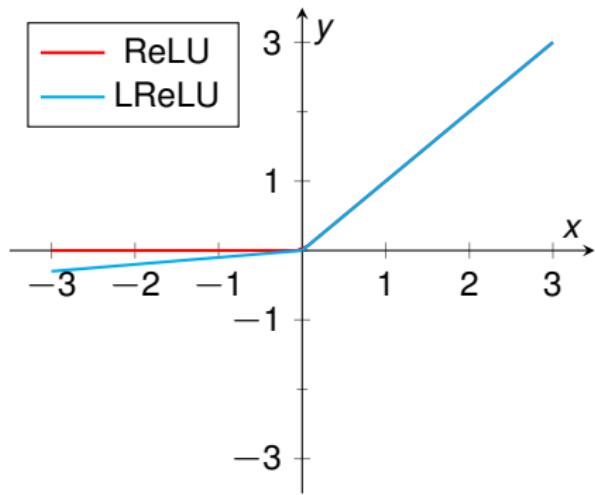
- Weight/biases trained to yield negative values for **any x**
- ReLU now only performs: $\mathbf{x} \mapsto 0$
- ReLU does not contribute to dividing the feature space
- No more updates because $f'(x) \mapsto 0$
 - Our precious ReLU is “dead”

Dying ReLUs



- Weight/biases trained to yield negative values for **any x**
- ReLU now only performs: $\mathbf{x} \mapsto 0$
- ReLU does not contribute to dividing the feature space
- No more updates because $f'(x) \mapsto 0$
 - Our precious ReLU is “dead”
- Often related to a (too) high learning rate

Activation Function



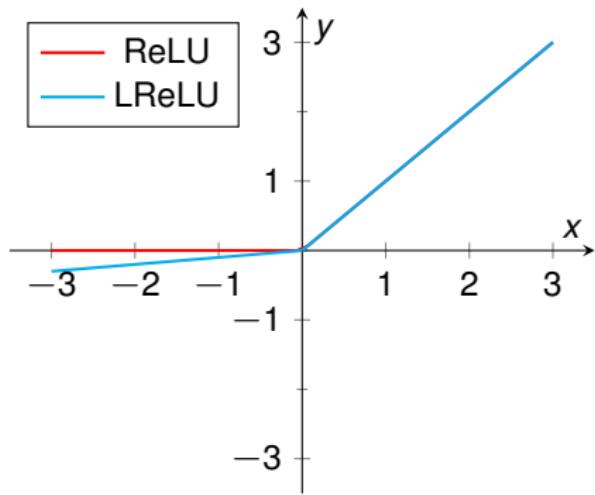
Leaky ReLU / Parametric ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{else} \end{cases}$$

- + Fixes dying ReLU problem
- Leaky ReLU: $\alpha = 0.01$ [5]

Activation Function



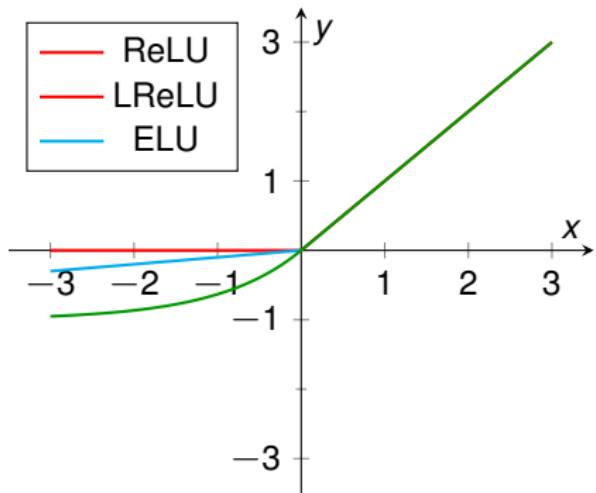
Leaky ReLU / Parametric ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{else} \end{cases}$$

- + Fixes dying ReLU problem
- Leaky ReLU: $\alpha = 0.01$ [5]
- Parametric ReLU (PReLU): learn α [2]

Exponential Linear Units (ELU)

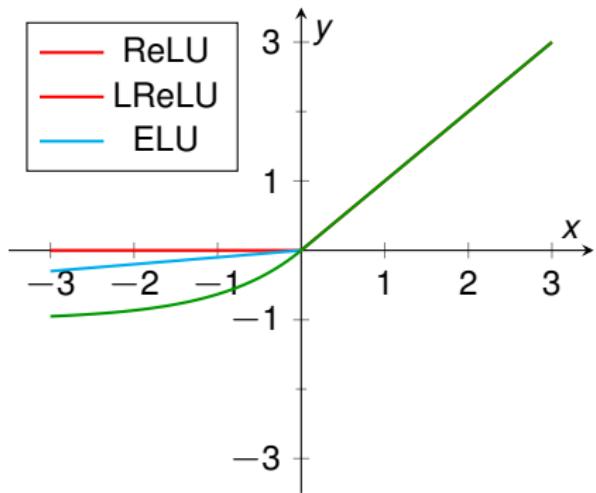


Exponential Linear Unit (ELU)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha \exp(x) & \text{else} \end{cases}$$

Exponential Linear Units (ELU)



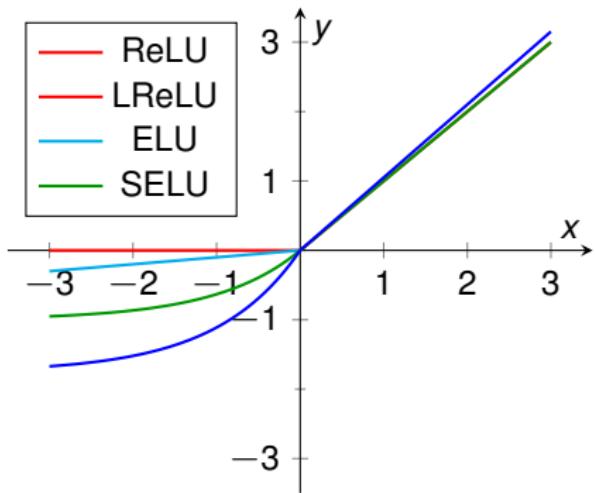
Exponential Linear Unit (ELU)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha \exp(x) & \text{else} \end{cases}$$

- + Vanishing gradient is not back
- + Can keep zero mean activations
- Gradient is in general not continuous!

Scaled ELU (SELU)



Scaled Exponential Linear Unit

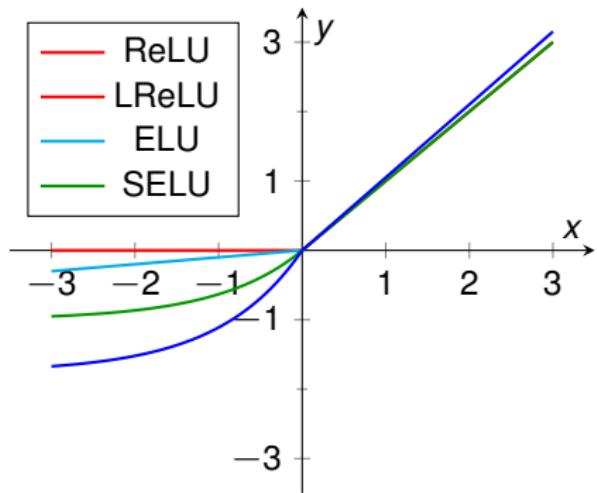
$$f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

$$f'(x) = \lambda \begin{cases} 1 & \text{if } x > 0 \\ \alpha f(x) & \text{else} \end{cases}$$

$$\lambda_{01} = 1.0507$$

$$\alpha_{01} = 1.6733$$

Scaled ELU (SELU)



Scaled Exponential Linear Unit

$$f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

$$f'(x) = \lambda \begin{cases} 1 & \text{if } x > 0 \\ \alpha f(x) & \text{else} \end{cases}$$

$$\lambda_{01} = 1.0507$$

$$\alpha_{01} = 1.6733$$

- Brand new variant [3]
- Idea: Self-normalizing - $\mu = 0, \sigma = 1 \mapsto \lambda_{01}, \alpha_{01}$
- Alternative to Batch Normalization? (see next lecture)

Other Activation Functions

Other Activation Functions

- Maxout: Learns the activation function [1]

Other Activation Functions

- Maxout: Learns the activation function [1]
- Radial basis functions

Other Activation Functions

- Maxout: Learns the activation function [1]
- Radial basis functions
- Softplus $f(x) = \ln(1 + e^x)$: less efficient than ReLU

Other Activation Functions

- Maxout: Learns the activation function [1]
- Radial basis functions
- Softplus $f(x) = \ln(1 + e^x)$: less efficient than ReLU

This is getting ridiculous - what should we use?

Finding the Optimal Activation Function

Finding the Optimal Activation Function

- Reinforcement learning/search problem

Finding the Optimal Activation Function

- Reinforcement learning/search problem
 - Unfortunately a single “step” means **training a network** from scratch

Finding the Optimal Activation Function

- Reinforcement learning/search problem
 - Unfortunately a single “step” means **training a network** from scratch
 - If you have a **cloud/supercomputer** you can do something like this
- “Searching for Activation Functions” [6] published by Google on Oct 16, 2017

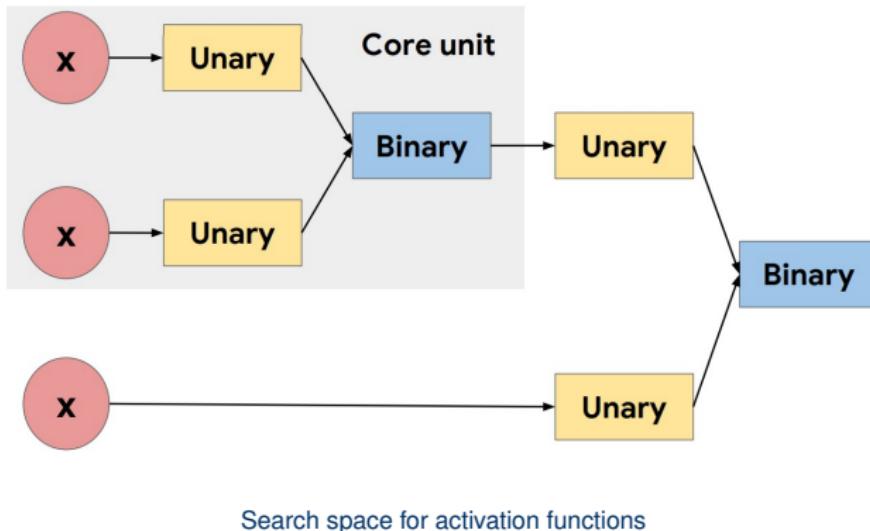
Finding the Optimal Activation Function

- Reinforcement learning/search problem
 - Unfortunately a single “step” means **training a network** from scratch
 - If you have a **cloud/supercomputer** you can do something like this
- “Searching for Activation Functions” [6] published by Google on Oct 16, 2017

Strategy

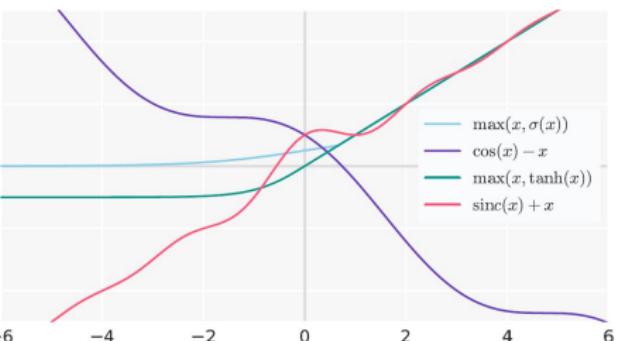
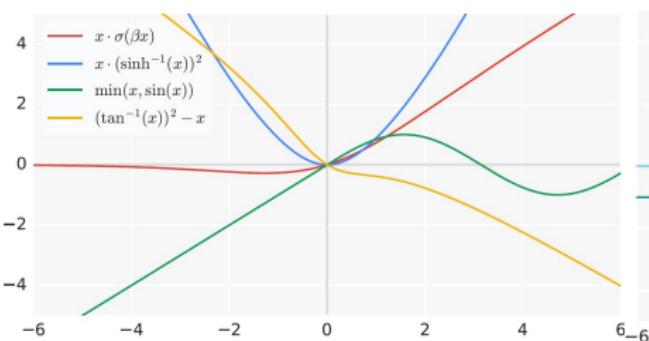
1. Define a search-space
2. Perform the search using a RNN with reinforcement learning
3. Use the best result

Searching for Activation Functions [6]



Source: <https://arxiv.org/pdf/1710.05941.pdf>

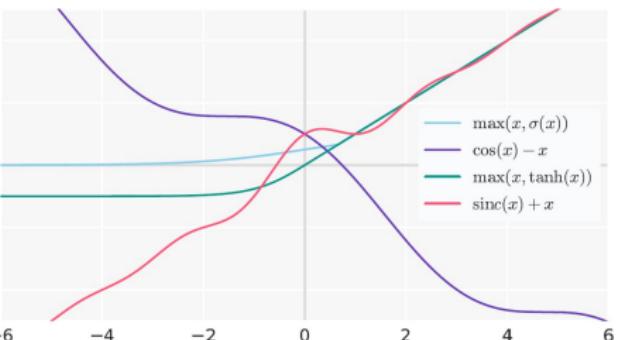
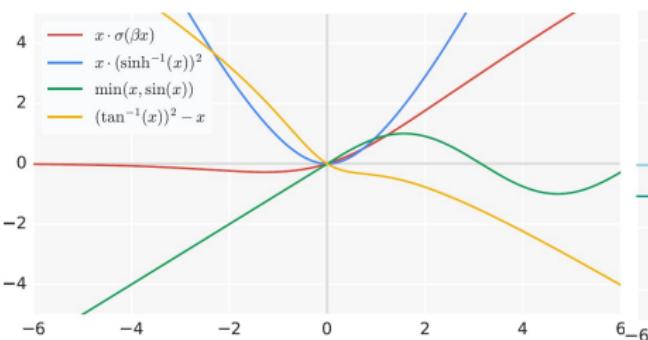
Searching for Activation Functions [6]



- We don't have a **cloud**, but we can use those

Source: <https://arxiv.org/pdf/1710.05941.pdf>

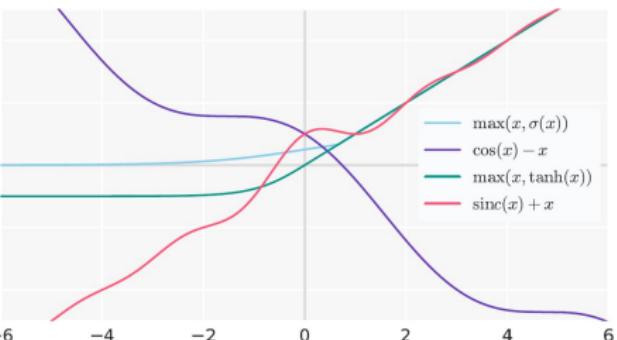
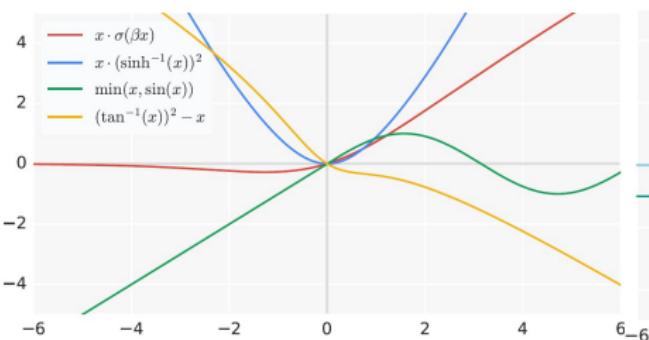
Searching for Activation Functions [6]



- We don't have a **cloud**, but we can use those
- **Complicated** activation functions **didn't perform well**

Source: <https://arxiv.org/pdf/1710.05941.pdf>

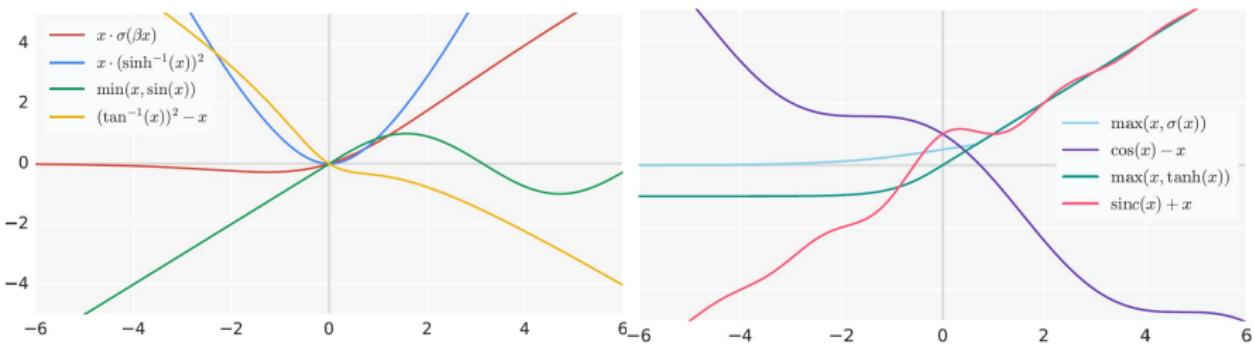
Searching for Activation Functions [6]



- We don't have a **cloud**, but we can use those
- **Complicated** activation functions **didn't perform well**
- They now call $x \cdot \sigma(\beta x)$ the “Swish” function

Source: <https://arxiv.org/pdf/1710.05941.pdf>

Searching for Activation Functions [6]



- We don't have a **cloud**, but we can use those
- **Complicated** activation functions **didn't perform well**
- They now call $x \cdot \sigma(\beta x)$ the “Swish” function
- Has been proposed before as “Sigmoid-weighted Linear Unit” [7]

Source: <https://arxiv.org/pdf/1710.05941.pdf>

Let's look into their results in detail

Disclaimer

Never show tables in your slides. Try hard to find a better representation.

Source: <https://arxiv.org/pdf/1710.05941.pdf>

Let's look into their results in detail

Model	Top-1 Acc. (%)		
LReLU	79.5	79.5	79.6
PReLU	79.7	79.8	80.1
Softplus	80.1	80.2	80.4
ELU	75.8	79.9	80.0
SELU	79.0	79.2	79.2
GELU	79.6	79.6	79.9
ReLU	79.5	79.6	79.8
Swish-1	80.2	80.3	80.4
Swish	80.2	80.2	80.3

Inception-Resnet-V2 architecture trained on ImageNet.

Source: <https://arxiv.org/pdf/1710.05941.pdf>

Let's look into their results in detail

Model	Top-1 Acc. (%)		
LReLU	79.5	79.5	79.6
PReLU	79.7	79.8	80.1
Softplus	80.1	80.2	80.4
ELU	75.8	79.9	80.0
SELU	79.0	79.2	79.2
GELU	79.6	79.6	79.9
ReLU	79.5	79.6	79.8
Swish-1	80.2	80.3	80.4
Swish	80.2	80.2	80.3

- Are any of these differences significant?

Inception-Resnet-V2 architecture trained on ImageNet.

Source: <https://arxiv.org/pdf/1710.05941.pdf>

Let's look into their results in detail

Model	Top-1 Acc. (%)		
LReLU	79.5	79.5	79.6
PReLU	79.7	79.8	80.1
Softplus	80.1	80.2	80.4
ELU	75.8	79.9	80.0
SELU	79.0	79.2	79.2
GELU	79.6	79.6	79.9
ReLU	79.5	79.6	79.8
Swish-1	80.2	80.3	80.4
Swish	80.2	80.2	80.3

- Are any of these differences significant?
→ Use t-test

Inception-Resnet-V2 architecture trained on ImageNet.

Source: <https://arxiv.org/pdf/1710.05941.pdf>

Let's look into their results in detail

Model	Top-1 Acc. (%)		
LReLU	79.5	79.5	79.6
PReLU	79.7	79.8	80.1
Softplus	80.1	80.2	80.4
ELU	75.8	79.9	80.0
SELU	79.0	79.2	79.2
GELU	79.6	79.6	79.9
ReLU	79.5	79.6	79.8
Swish-1	80.2	80.3	80.4
Swish	80.2	80.2	80.3

- Are any of these differences significant?
→ Use t-test
- We'd need a difference from -1.043 to 1.776 to be 95% sure that it's better!

Inception-Resnet-V2 architecture trained on ImageNet.

Source: <https://arxiv.org/pdf/1710.05941.pdf>

Let's look into their results in detail

Model	Top-1 Acc. (%)		
LReLU	79.5	79.5	79.6
PReLU	79.7	79.8	80.1
Softplus	80.1	80.2	80.4
ELU	75.8	79.9	80.0
SELU	79.0	79.2	79.2
GELU	79.6	79.6	79.9
ReLU	79.5	79.6	79.8
Swish-1	80.2	80.3	80.4
Swish	80.2	80.2	80.3

- Are any of these differences significant?
→ Use t-test
- We'd need a difference from -1.043 to 1.776 to be 95% sure that it's better!
- So no - none of this is significant
→ ReLU vs Swish is 0.51%

Inception-Resnet-V2 architecture trained on ImageNet.

Source: <https://arxiv.org/pdf/1710.05941.pdf>

Let's look into their results in detail

Model	Top-1 Acc. (%)		
LReLU	79.5	79.5	79.6
PReLU	79.7	79.8	80.1
Softplus	80.1	80.2	80.4
ELU	75.8	79.9	80.0
SELU	79.0	79.2	79.2
GELU	79.6	79.6	79.9
ReLU	79.5	79.6	79.8
Swish-1	80.2	80.3	80.4
Swish	80.2	80.2	80.3

- Are any of these differences significant?
→ Use t-test
- We'd need a difference from -1.043 to 1.776 to be 95% sure that it's better!
- So no - none of this is significant
→ ReLU vs Swish is 0.51%
- ImageNet has become a bad benchmark

Inception-Resnet-V2 architecture trained on ImageNet.

Source: <https://arxiv.org/pdf/1710.05941.pdf>

Summary Activation Functions

- Go-to solution: **ReLU**

Summary Activation Functions

- Go-to solution: **ReLU**
- SELUs have **attractive** theoretical properties.

Summary Activation Functions

- Go-to solution: **ReLU**
- SELUs have **attractive** theoretical properties.
- ... but try ReLU first.

Summary Activation Functions

- Go-to solution: **ReLU**
- SELUs have **attractive** theoretical properties.
- ... but try ReLU first.
- Best activation function is a **difficult, expensive optimization problem**.

Summary Activation Functions

- Go-to solution: **ReLU**
- SELUs have **attractive** theoretical properties.
- ... but try ReLU first.
- Best activation function is a **difficult, expensive optimization problem**.

- Still a very active area of research!

Summary Activation Functions

- Go-to solution: **ReLU**
- SELUs have **attractive** theoretical properties.
- ... but try ReLU first.
- Best activation function is a **difficult, expensive optimization problem**.

- Still a very active area of research!

What we know about good activation functions:

- They have almost linear areas to prevent **vanishing gradients**.
- They have **saturating areas** to provide **non-linearity**.
- They should be **monotonic**.



FAU

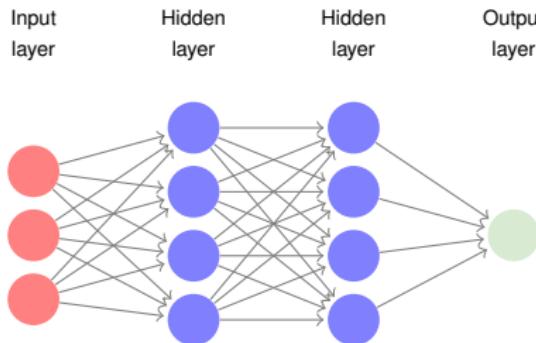
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Convolutional Neural Networks



Motivation

- So far: Fully connected layers - each input is connected to each node
- Very powerful: Can represent any kind of (linear) relationship between inputs



Motivation

- So far: Fully connected layers - each input is connected to each node
- Very powerful: Can represent any kind of (linear) relationship between inputs
- Large part of machine learning: images/videos/sounds

Motivation

- So far: Fully connected layers - each input is connected to each node
 - Very powerful: Can represent any kind of (linear) relationship between inputs
 - Large part of machine learning: images/videos/sounds
 - Assume we have:
 - An image with size 512×512 pixels
 - One hidden layer with 8 neurons
- $(512^2 + 1) \cdot 8 > 2$ million trainable weights!

Motivation (cont.)

So the size is a problem. Is there something else?

Source: <https://news.nationalgeographic.com>

Motivation (cont.)

So the size is a problem. Is there something else?

- Example: Classify between cat and dog



Source: <https://news.nationalgeographic.com>

Motivation (cont.)

So the size is a problem. Is there something else?

- Example: Classify between cat and dog



Source: <https://news.nationalgeographic.com>

Motivation (cont.)

So the size is a problem. Is there something else?

- Example: Classify between cat and dog
- Pixels are a **bad** features!
 - Highly correlated
 - Scale dependent
 - Intensity variations
 - ...



Source: <https://news.nationalgeographic.com>

Motivation (cont.)

So the size is a problem. Is there something else?

- Example: Classify between cat and dog
- Pixels are a **bad** features!
 - Highly correlated
 - Scale dependent
 - Intensity variations
 - ...
- Pixels are a bad representation from a machine learning point of view



Source: <https://news.nationalgeographic.com>

Motivation (cont.)

Can we find a better representation?

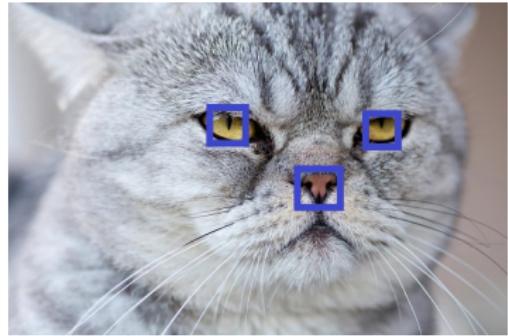


Source: <https://news.nationalgeographic.com>

Motivation (cont.)

Can we find a better representation?

- We have a certain degree of the locality in an image

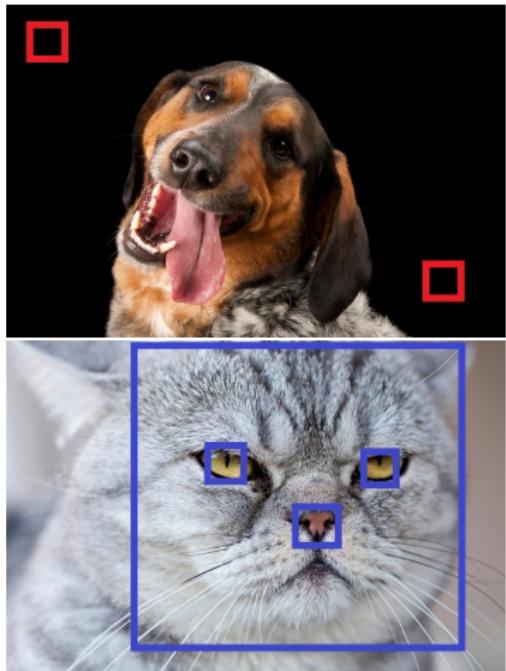


Source: <https://news.nationalgeographic.com>

Motivation (cont.)

Can we find a better representation?

- We have a certain degree of the locality in an image
- Hierarchy of features:
 - edges + corners \mapsto eyes
 - eyes + nose + ears \mapsto face
 - face + body + legs \mapsto animal

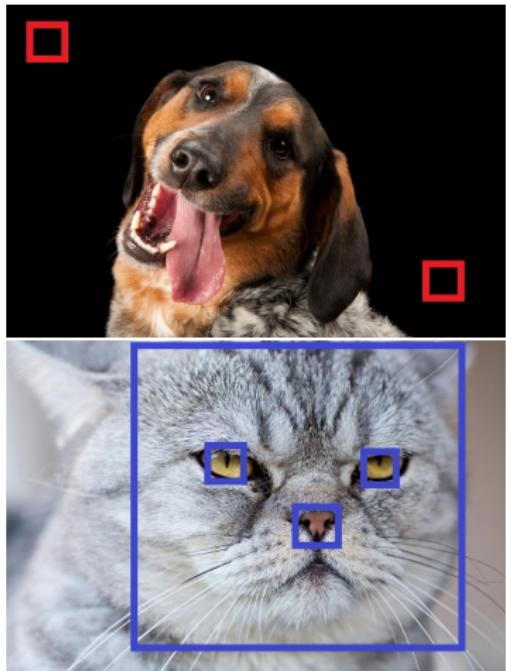


Source: <https://news.nationalgeographic.com>

Motivation (cont.)

Can we find a better representation?

- We have a certain degree of the locality in an image
- Hierarchy of features:
 - edges + corners \mapsto eyes
 - eyes + nose + ears \mapsto face
 - face + body + legs \mapsto animal
- Composition matters!

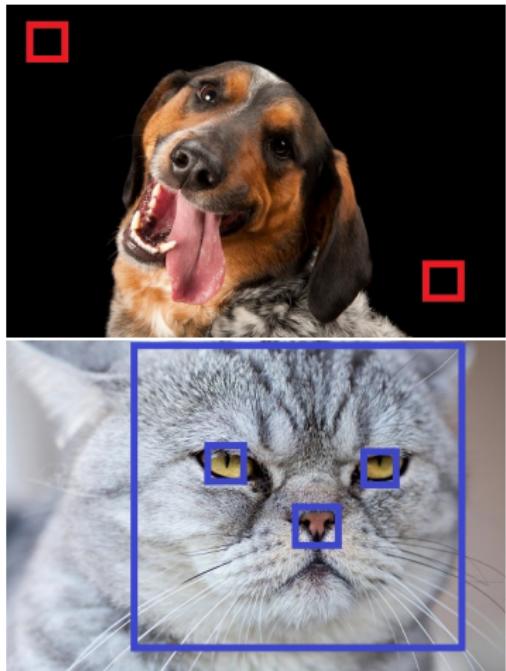


Source: <https://news.nationalgeographic.com>

Motivation (cont.)

Can we find a better representation?

- We have a certain degree of the locality in an image
- Hierarchy of features:
 - edges + corners \mapsto eyes
 - eyes + nose + ears \mapsto face
 - face + body + legs \mapsto animal
- Composition matters!

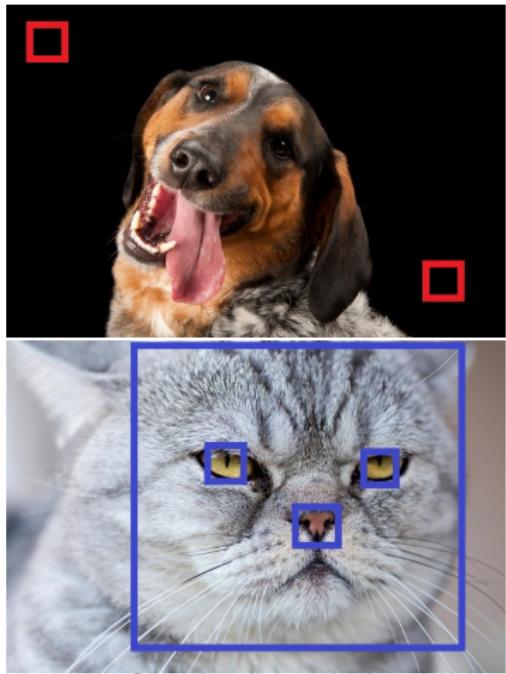


Source: <https://news.nationalgeographic.com>

Motivation (cont.)

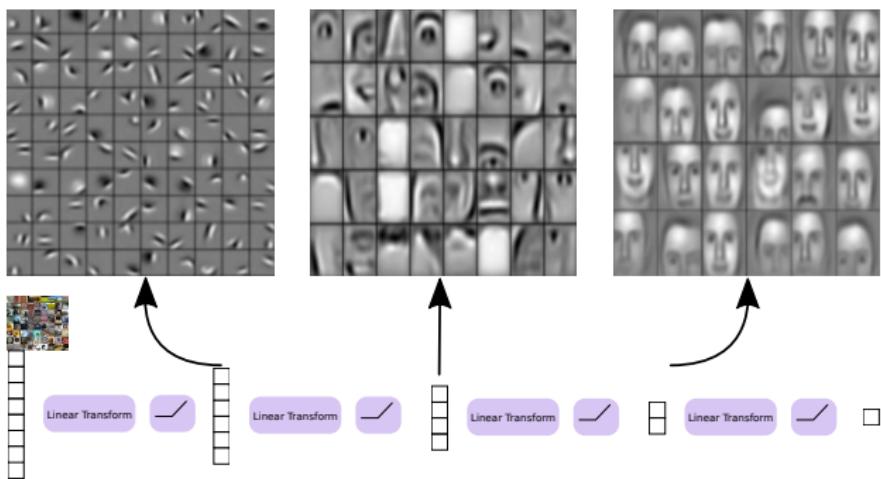
Can we find a better representation?

- We have a certain degree of the locality in an image
- Hierarchy of features:
 - edges + corners \mapsto eyes
 - eyes + nose + ears \mapsto face
 - face + body + legs \mapsto animal
- Composition matters!
- Learn better representation, then classify!



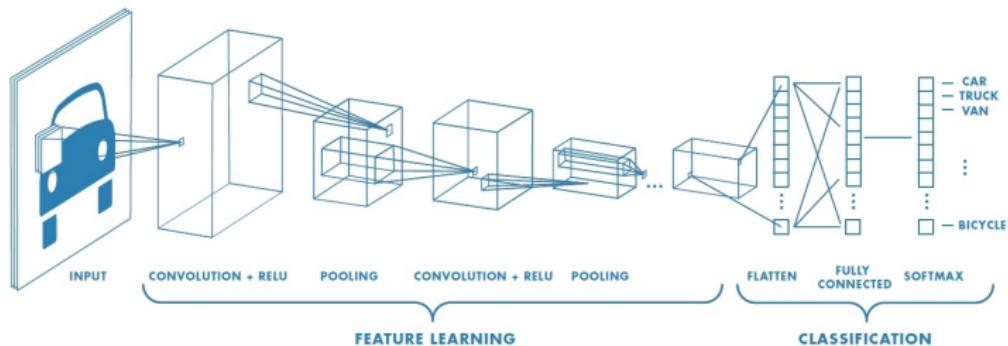
Source: <https://news.nationalgeographic.com>

Convolutional Neural Networks



- Local connectivity → filters
- Use same filters over the whole image → translational equivariance
- Hierarchy of filters working on different scales
- + learning = **Convolutional Neural Networks**

Convolutional Neural Networks - Architecture



Four essential building blocks:

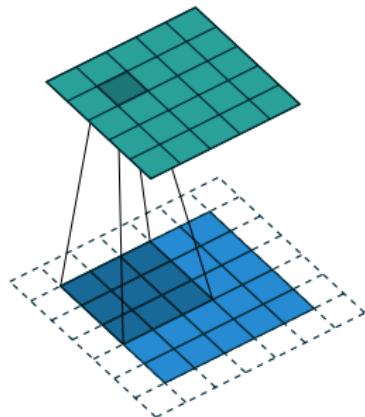
- Convolutional layer: Feature extraction
- Activation function: Nonlinearity
- Pooling layer: Compress information, save parameters
- Last layer: Fully-connected for classification

Source: <https://de.mathworks.com/discovery/convolutional-neural-network.html>

Convolutional Layers

Convolutional Layer - Local Connectivity

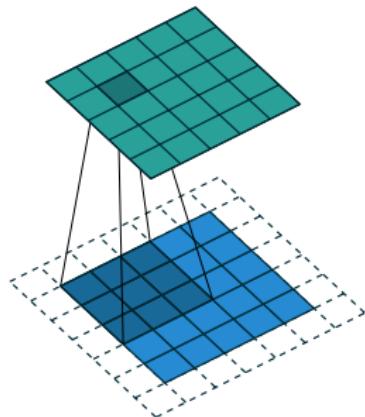
- Exploit spatial structure by only connecting pixels in a neighborhood
- Can be expressed as fully connected layer:
Except for local connections, each entry in \mathbf{W} is 0



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

Convolutional Layer - Local Connectivity

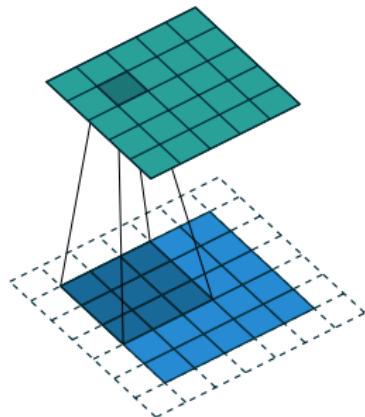
- Exploit spatial structure by only connecting pixels in a neighborhood
- Can be expressed as fully connected layer:
Except for local connections, each entry in \mathbf{W} is 0
- Effective weights: Filter of size 3×3 , 5×5 , 7×7 , ...



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

Convolutional Layer - Local Connectivity

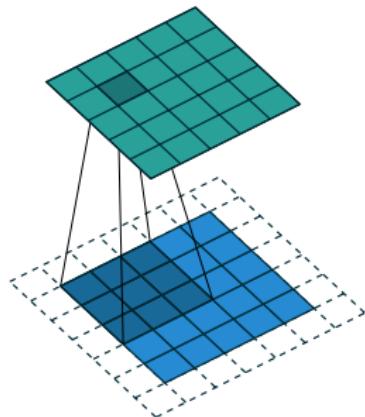
- Exploit spatial structure by only connecting pixels in a neighborhood
- Can be expressed as fully connected layer:
Except for local connections, each entry in \mathbf{W} is 0
- Effective weights: Filter of size 3×3 , 5×5 , 7×7 , ...
- Features that are important at one location are likely important anywhere in the image
 - Use the same weights all over: **tied weights** (also **shared weights**).



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

Convolutional Layer - Local Connectivity

- Exploit spatial structure by only connecting pixels in a neighborhood
- Can be expressed as fully connected layer:
Except for local connections, each entry in \mathbf{W} is 0
- Effective weights: Filter of size 3×3 , 5×5 , 7×7 , ...
- Features that are important at one location are likely important anywhere in the image
 - Use the same weights all over: **tied weights** (also **shared weights**).
 - **Convolution with trainable filters**



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

Recap: Convolution

Convolution

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

Recap: Convolution

- Convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau$$

Recap: Convolution

- Convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau$$

- Cross-correlation:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x + \tau)d\tau$$

Recap: Convolution

- Convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau$$

- Cross-correlation:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x + \tau)d\tau$$

- Cross-correlation is convolution with a flipped kernel g – and vice versa!

Recap: Convolution

- Convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau$$

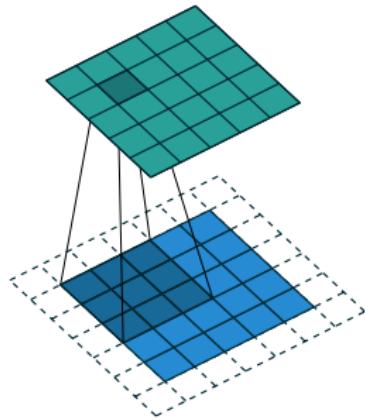
- Cross-correlation:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x + \tau)d\tau$$

- Cross-correlation is convolution with a flipped kernel g – and vice versa!
- Implementation: Cross-correlation is frequently used in the forward pass - the weights are initialized randomly anyway

Padding

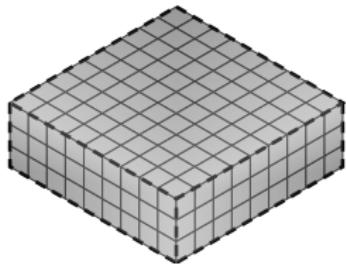
- Convolution reduces image size by $2 \cdot \lfloor \frac{n}{2} \rfloor$ pixels (n : kernel size).
- Necessary to pay attention to the borders:
- ‘Same’ padding (usually zero padding):
 - Input and output have the same size
- ‘Valid’/no padding:
 - The output is smaller than the input



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

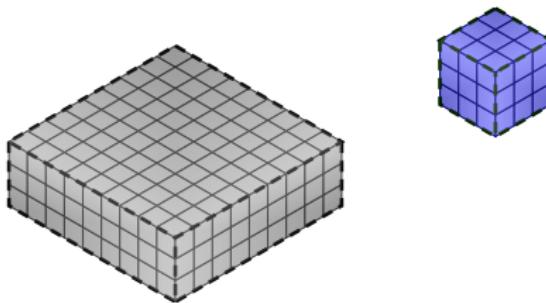
Forward Pass: Multi-channel convolution

- Input of size $X \times Y \times S$, where S is the number of input channels



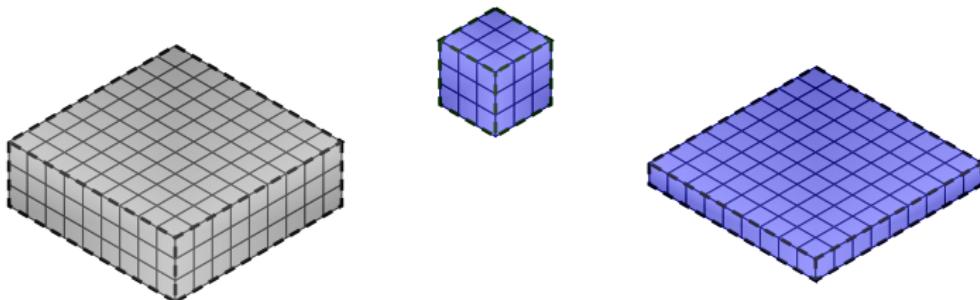
Forward Pass: Multi-channel convolution

- Input of size $X \times Y \times S$, where S is the number of input channels
- H Filters with size $M \times N \times S \mapsto$ **fully connected** across channels



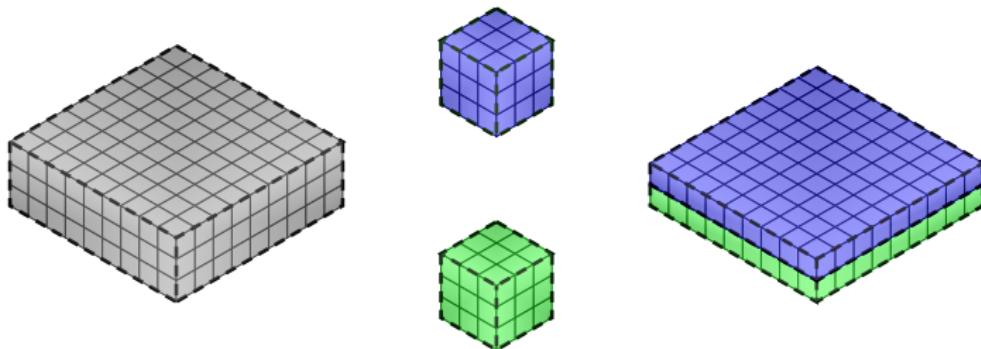
Forward Pass: Multi-channel convolution

- Input of size $X \times Y \times S$, where S is the number of input channels
- H Filters with size $M \times N \times S \mapsto$ **fully connected** across channels
- Output dimensions: $X \times Y \times H$ (with ‘same’ padding)



Forward Pass: Multi-channel convolution

- Input of size $X \times Y \times S$, where S is the number of input channels
- H Filters with size $M \times N \times S \mapsto$ **fully connected** across channels
- Output dimensions: $X \times Y \times H$ (with ‘same’ padding)



Backward pass: Multi-channel convolution

- Convolution can be expressed as matrix multiplication with matrix \mathbf{W} : using a **Toeplitz matrix**
- We can use the **same formulas** as for the fully connected layer!

$$\mathbf{E}_{l-1} = \mathbf{W}^T \mathbf{E}_l$$

$$\nabla \mathbf{W} = \mathbf{E}_l \mathbf{X}^T$$

where

- \mathbf{X} is the input and
- $\mathbf{E}_l/\mathbf{E}_{l-1}$ is the error in layer $l/l - 1$

Backward pass: Multi-channel convolution

- Convolution can be expressed as matrix multiplication with matrix \mathbf{W} : using a **Toeplitz matrix**
- We can use the **same formulas** as for the fully connected layer!

$$\mathbf{E}_{l-1} = \mathbf{W}^T \mathbf{E}_l$$

$$\nabla \mathbf{W} = \mathbf{E}_l \mathbf{X}^T$$

where

- \mathbf{X} is the input and
- $\mathbf{E}_l/\mathbf{E}_{l-1}$ is the error in layer $l/l - 1$
- These can be expressed as convolutions \mapsto exercise

Convolutional Layer - What have we gained?

- Stack multiple filters to get a trainable filter bank.
- Layer with 8 filters (nodes) with 5×5 neighborhood
 - $5^2 \cdot 8 = \underline{200}$ weights

Convolutional Layer - What have we gained?

- Stack multiple filters to get a trainable filter bank.
- Layer with 8 filters (nodes) with 5×5 neighborhood
 - $5^2 \cdot 8 = \underline{200}$ weights
- Convolution: **Independent** of image size!
- Much more training data for one weight!

Strided Convolutions

- Instead of multiplying the filter at each pixel position, we can **skip** some **positions**
- **Stride** s describes the offset



Source: https://en.wikipedia.org/wiki/Monty_Python%27s_Flying_Circus, Dinsdale

Strided Convolutions

- Instead of multiplying the filter at each pixel position, we can **skip** some **positions**
- **Stride** s describes the offset
- **Reduces** the **size** of the output by a factor of s



Source: https://en.wikipedia.org/wiki/Monty_Python%27s_Flying_Circus_episode,_Dinsdale

Strided Convolutions

- Instead of multiplying the filter at each pixel position, we can **skip** some **positions**
- **Stride** s describes the offset
- **Reduces** the **size** of the output by a factor of s
- **Mathematically:** Convolution + subsampling



Source: https://en.wikipedia.org/wiki/Monty_Python%27s_Flying_Circus_episode,_Dinsdale

Strided Convolutions

Strided Convolution with stride $s = 2$

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

Dilated/Atrous Convolutions

- Additional variant of convolution in neural networks
- Dilate convolution kernel: Skip certain pixels
- Goal: Wider receptive field with less parameters/weights

Dilated Convolution

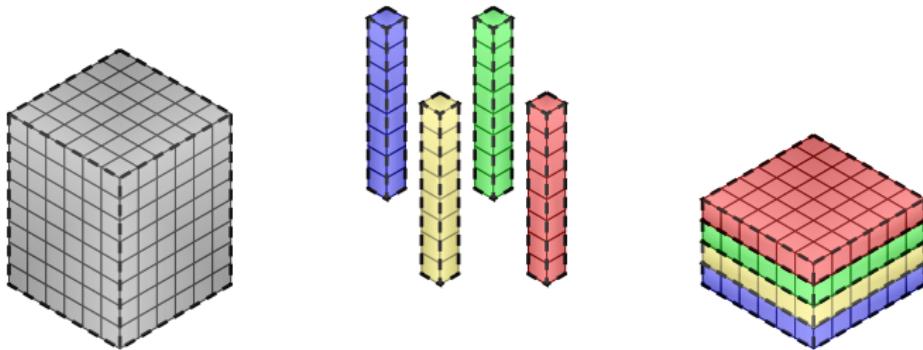
Source: https://github.com/vdumoulin/conv_arithmetic

1 × 1 Convolution Concept

- So far: H filters with neighborhood $3 \times 3, 5 \times 5, \dots$ and ‘depth’ S
- Filters are fully connected in ‘depth’ direction

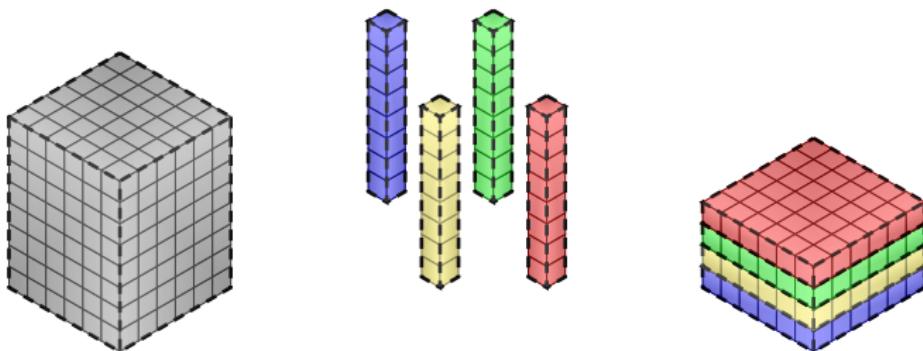
1×1 Convolution Concept

- So far: H filters with neighborhood $3 \times 3, 5 \times 5, \dots$ and ‘depth’ S
- Filters are fully connected in ‘depth’ direction
- We can decrease the neighborhood to 1×1
- And **just** use the fully connected property in the depth dimension



1 × 1 Convolution Concept

- So far: H filters with neighborhood $3 \times 3, 5 \times 5, \dots$ and ‘depth’ S
- Filters are fully connected in ‘depth’ direction
- We can decrease the neighborhood to 1×1
- And **just** use the fully connected property in the depth dimension



- Dimensionality reduction/expansion from S channels to H channels
- If we flatten the input, 1 × 1 convolutions are fully connected layer!

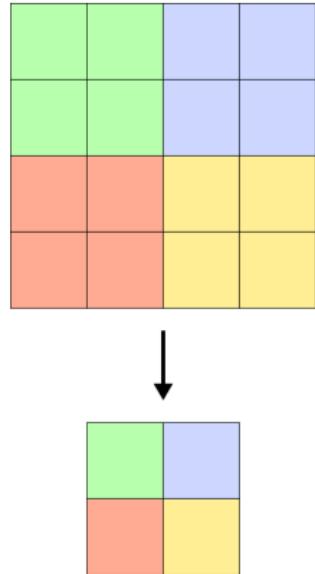
1×1 Convolution Concept (cont.)

- First described in “Network in Network” by Lin et al. [4]
- 1×1 convolutions simply calculate **inner products** at each position
- Simple and efficient method to **decrease** the **size** of a network
- Learns dimensionality reduction, e.g., can reduce redundancy in your feature maps

Pooling Layers

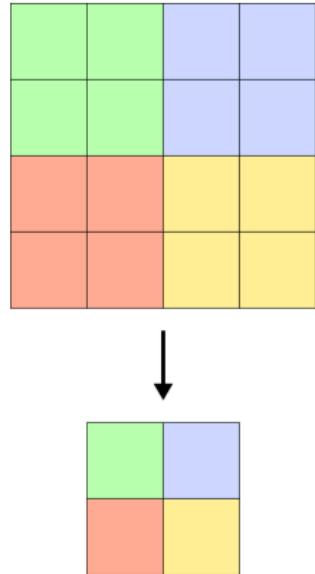
Idea behind Pooling Layers

- Fuses information of input across spatial locations
- Decreases number of parameters
- Reduces computational costs and overfitting



Idea behind Pooling Layers

- Fuses information of input across spatial locations
- Decreases number of parameters
- Reduces computational costs and overfitting
- Assumptions:
 - Features are hierarchically structured
 - Creates “summaries” of regions
 - Provides translational invariance
 - Exact location of a feature is not important



Max Pooling – Forward Pass

- Propagate maximum value in a neighborhood to next layer
- Typical choices: 2×2 or 3×3 neighborhood
- “Stride” of pooling usually equals the neighborhood size
- Maximum propagation adds additional non-linearity

Max pooling concept. Note that usually a stride > 1 is used for pooling.

Max Pooling – Backward Pass



Max Pooling – Backward Pass



- Only one value contributes to error
- Error is propagated only along the path of the maximum value

Average Pooling

- Propagate average of the neighborhood
- Does not consistently perform better than max pooling
- Backward pass: Error is shared to equal parts

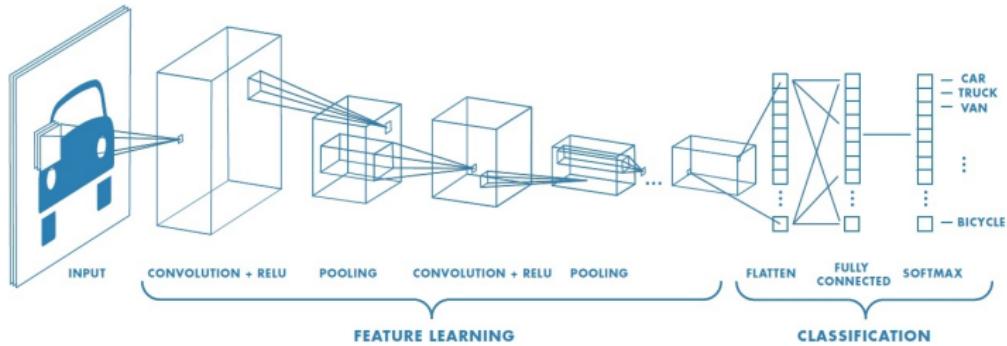
Additional Pooling Strategies

- Fractional max pooling
- L_p pooling
- Stochastic pooling
- Spacial pyramid pooling
- Generalized pooling
- ...

Alternative: Strided Convolution

- Historically, max pooling was the most frequently used pooling strategies due to additional non-linearity
- More recently, convolution with stride $s > 1$ has become more common
 - Allows for trainable downsampling strategy

Recap: Convolutional Neural Networks - Architecture

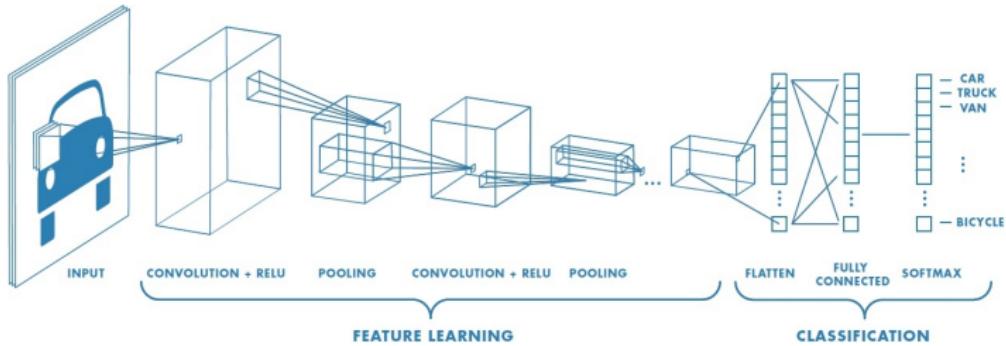


Four essential building blocks:

- Convolutional layer: Feature extraction
- Activation function: Nonlinearity
- Pooling layer: Compress information, save parameters
- Last layer: Fully-connected for classification

Source: <https://de.mathworks.com/discovery/convolutional-neural-network.html>

Recap: Convolutional Neural Networks - Architecture

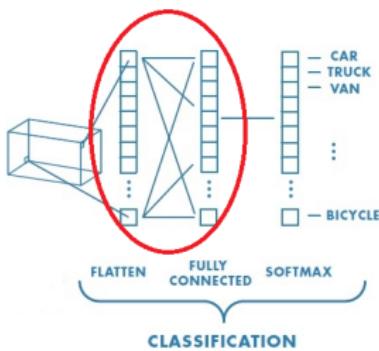


Four Three essential building blocks:

- Convolutional layer: Feature extraction
- Activation function: Nonlinearity
- Pooling layer: Compress information, save parameters
- Last layer: Fully connected for classification **We can replace this layer!**

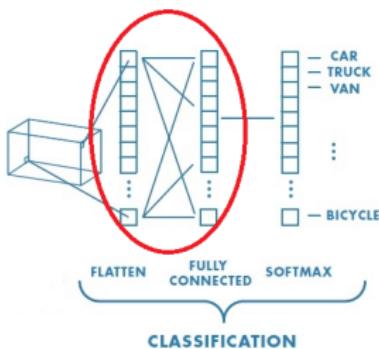
Source: <https://de.mathworks.com/discovery/convolutional-neural-network.html>

Replacing the Fully Connected Layer



- Conv and pooling layers generate better representation → better features
- Fully connected layers for classification

Replacing the Fully Connected Layer



- Conv and pooling layers generate better representation → better features
- Fully connected layers for classification
- **Alternatively and equivalently:** Use flatten & 1×1 convolution or $N \times N$ convolution
- Enables **arbitrary input sizes** in combination with global average pooling!

Inception model

- Szegedy et al. (2014): Going Deeper With Convolutions [8]
 - Very influential publication: > 5000 citations (Nov. '17)
 - Won ImageNet Large-Scale Visual Recognition Challenge 2014
 - GoogLeNet as one incarnation
 - Inspired by Network in Network [4]

Inception model

- Szegedy et al. (2014): Going Deeper With Convolutions [8]
 - Very influential publication: > 5000 citations (Nov. '17)
 - Won ImageNet Large-Scale Visual Recognition Challenge 2014
 - GoogLeNet as one incarnation
 - Inspired by Network in Network [4]
- Self-stated motto:

Inception model

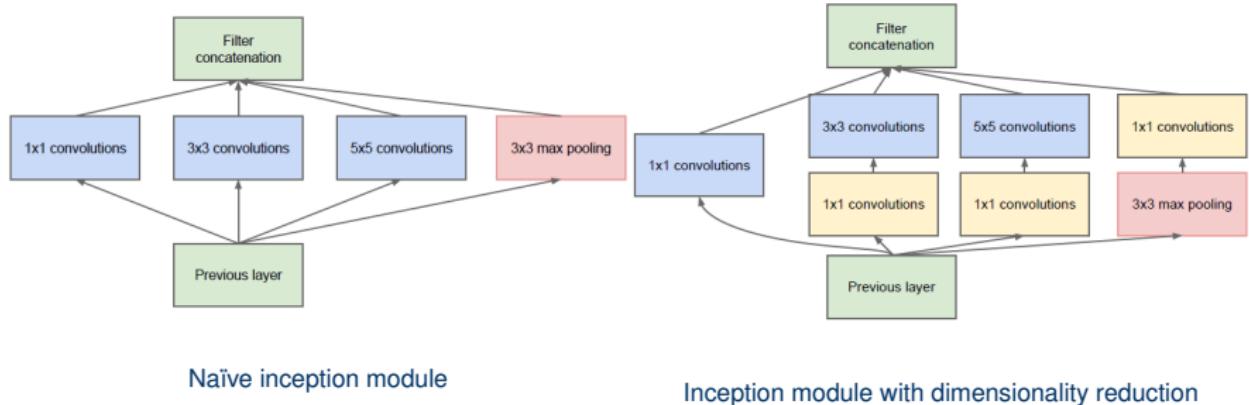
- Szegedy et al. (2014): Going Deeper With Convolutions [8]
 - Very influential publication: > 5000 citations (Nov. '17)
 - Won ImageNet Large-Scale Visual Recognition Challenge 2014
 - GoogLeNet as one incarnation
 - Inspired by Network in Network [4]
- Self-stated motto:



Source: <http://knowyourmeme.com/photos/531557-we-need-to-go-deeper>

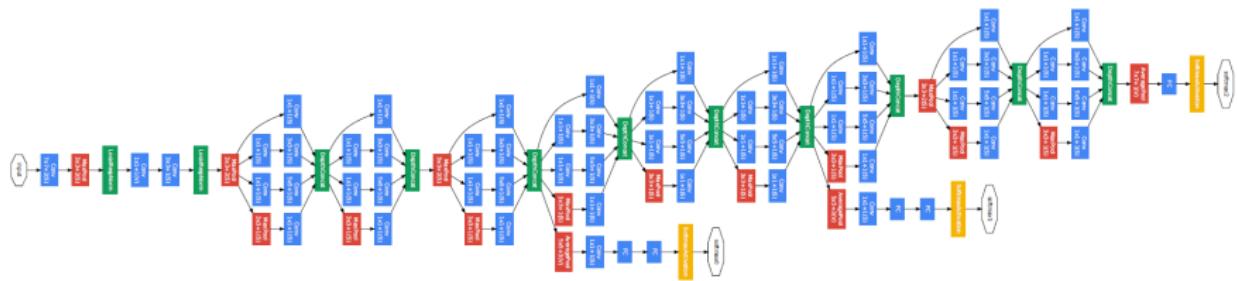
Inception model

- **Idea:** Why use only one type of filter in one layer?
Why not combine different neighborhoods/pooling/etc.?
- Construction of ‘inception modules’ that are stacked to form a large network.



Source: [8]

Inception model – GoogLeNet



GoogLeNet architecture with inception modules.

Source: [8]

**NEXT TIME
ON DEEP LEARNING**

Coming up

- How to prevent networks from just memorizing the training data?
- Is there a way to force features to be independent?
- How can we make sure our network also recognizes cats in different poses?
- Can we fix the covariate shift problem?

Comprehensive Questions

- Name five activation functions.
- Discuss those 5 activation functions.
- What is the zero-centering problem?
- Why does ReLU as activation function perform much better than sigmoid/tanh in a large number of tasks?
- Why are convolutional networks well suited for image and audio processing?
- Write down a mathematical description of strided convolution.
- What is the connection between 1×1 convolutions and fully connected layers?
- How would you implement a classifier which operates on image patches?
- What is a pooling layer?
- Why do we use pooling layers?
- On what data would CNNs probably perform bad?

Further Reading

- [Link](#) - [3] for a paper about Self Normalizing Networks
- [Link](#) - [4] for a creative Network in Network paper
- [Link](#) - [6] for details on learned activation functions
- [Link](#) - [8] if everything so far was not deep enough for you

Questions?



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] I. J. Goodfellow, D. Warde-Farley, M. Mirza, et al. "Maxout Networks". In: [ArXiv e-prints](#) (Feb. 2013). arXiv: 1302.4389 [stat.ML].
- [2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: [CoRR](#) abs/1502.01852 (2015). arXiv: 1502.01852.
- [3] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, et al. "Self-Normalizing Neural Networks". In: [Advances in Neural Information Processing Systems \(NIPS\)](#). Vol. abs/1706.02515. 2017. arXiv: 1706.02515.
- [4] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: [CoRR](#) abs/1312.4400 (2013). arXiv: 1312.4400.

References II

- [5] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: [Proc. ICML](#). Vol. 30. 1. 2013.
- [6] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions". In: [CoRR](#) abs/1710.05941 (2017). arXiv: 1710.05941.
- [7] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning". In: [arXiv preprint arXiv:1702.03118](#) (2017).
- [8] Christian Szegedy, Wei Liu, Yangqing Jia, et al. "Going Deeper with Convolutions". In: [CoRR](#) abs/1409.4842 (2014). arXiv: 1409.4842.