VISUAL COMPUTING
& SECURITY LAB

Deep Learning with Convolutional Neural Networks

Sebastian Hegenbart

November, 2015

Motivation

Did you ever wonder what that deep learning and CNN stuff everyone talks about is?!

Motivation

You might also wonder what this image is about!?

Motivation

You might also wonder what this image is about!?

Worry no more! You will know in approximately 5 hours!

Motivation

Did you ever wonder what that deep learning and CNN stuff everyone talks about is?!

Motivation

Did you ever wonder what that deep learning and CNN stuff everyone talks about is?!

Motivation

Did you ever wonder what that deep learning and CNN stuff everyone talks about is?!

1. Convolutional neural networks (CNN / ConvNets) are a type of **feed-forward artificial neural network** (ANN)

Motivation

Did you ever wonder what that deep learning and CNN stuff everyone talks about is?!

1. Convolutional neural networks (CNN / ConvNets) are a type of **feed-forward artificial neural network** (ANN)
2. ConvNets have been dominating recent challenges in computer vision such as the **Large Scale Visual Recognition Challenge**

Motivation

Did you ever wonder what that deep learning and CNN stuff everyone talks about is?!

1. Convolutional neural networks (CNN / ConvNets) are a type of **feed-forward artificial neural network** (ANN)
2. ConvNets have been dominating recent challenges in computer vision such as the **Large Scale Visual Recognition Challenge**
3. ConvNets have been used with high success in challenging areas such as **classification, localization** and **object detection**

Motivation

Did you ever wonder what that deep learning and CNN stuff everyone talks about is?!

1. Convolutional neural networks (CNN / ConvNets) are a type of **feed-forward artificial neural network** (ANN)
2. ConvNets have been dominating recent challenges in computer vision such as the **Large Scale Visual Recognition Challenge**
3. ConvNets have been used with high success in challenging areas such as **classification, localization** and **object detection**
4. The increasing **availability of high performance GPU hardware** allows to train even more complex architectures

Motivation

Did you ever wonder what that deep learning and CNN stuff everyone talks about is?!

1. Convolutional neural networks (CNN / ConvNets) are a type of **feed-forward artificial neural network** (ANN)
2. ConvNets have been dominating recent challenges in computer vision such as the **Large Scale Visual Recognition Challenge**
3. ConvNets have been used with high success in challenging areas such as **classification, localization** and **object detection**
4. The increasing **availability of high performance GPU hardware** allows to train even more complex architectures
5. ConvNets **sparked new research** interest in neural nets in the late 2000s

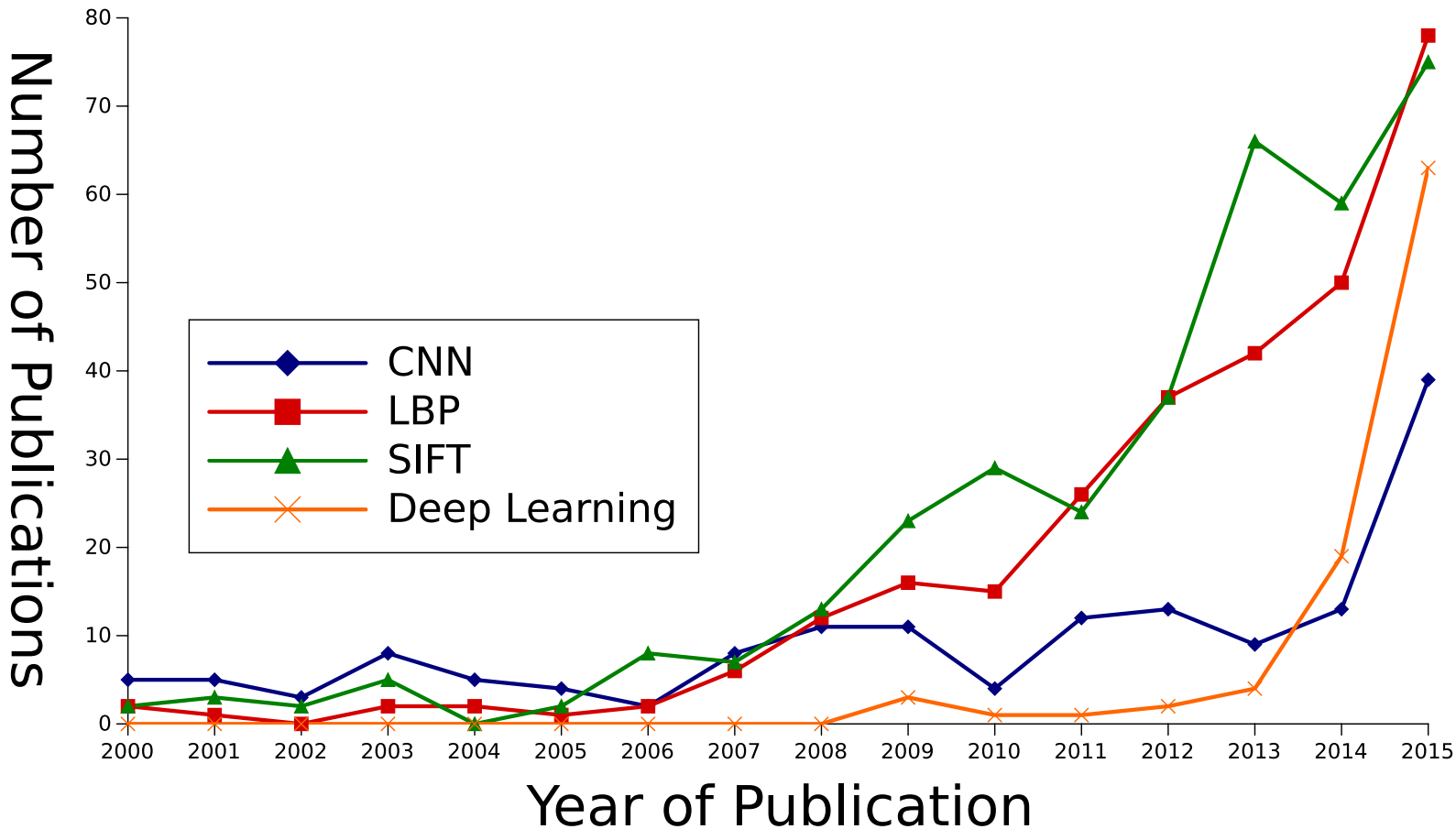
Motivation

Did you ever wonder what that deep learning and CNN stuff everyone talks about is?!

1. Convolutional neural networks (CNN / ConvNets) are a type of **feed-forward artificial neural network** (ANN)
2. ConvNets have been dominating recent challenges in computer vision such as the **Large Scale Visual Recognition Challenge**
3. ConvNets have been used with high success in challenging areas such as **classification, localization** and **object detection**
4. The increasing **availability of high performance GPU hardware** allows to train even more complex architectures
5. ConvNets **sparked new research** interest in neural nets in the late 2000s
6. ConvNets are seeping into **industry** as well!

Motivation cont.

As quick search on **Science Direct** gives a raw estimate of the number of journal publications from 2000 to 2015. Deep learning and CNNs are **gaining** more and more **interest**.



Motivation cont.

1. **Literature** in the field **dates back decades ago** (often using different terminologies)

Motivation cont.

1. **Literature** in the field **dates back decades ago** (often using different terminologies)
2. Various **concepts** were **discovered** and **forgotten** just to be re-discovered later using new names

Motivation cont.

1. **Literature** in the field **dates back decades ago** (often using different terminologies)
2. Various **concepts** were **discovered** and **forgotten** just to be re-discovered later using new names
3. There are some good tutorials on ConvNets but it takes quite some **effort to collect all required information**

Motivation cont.

1. **Literature** in the field **dates back decades ago** (often using different terminologies)
2. Various **concepts** were **discovered** and **forgotten** just to be re-discovered later using new names
3. There are some good tutorials on ConvNets but it takes quite some **effort to collect all required information**
4. It can be **hard** to **connect** various **concepts** and terminologies

Motivation cont.

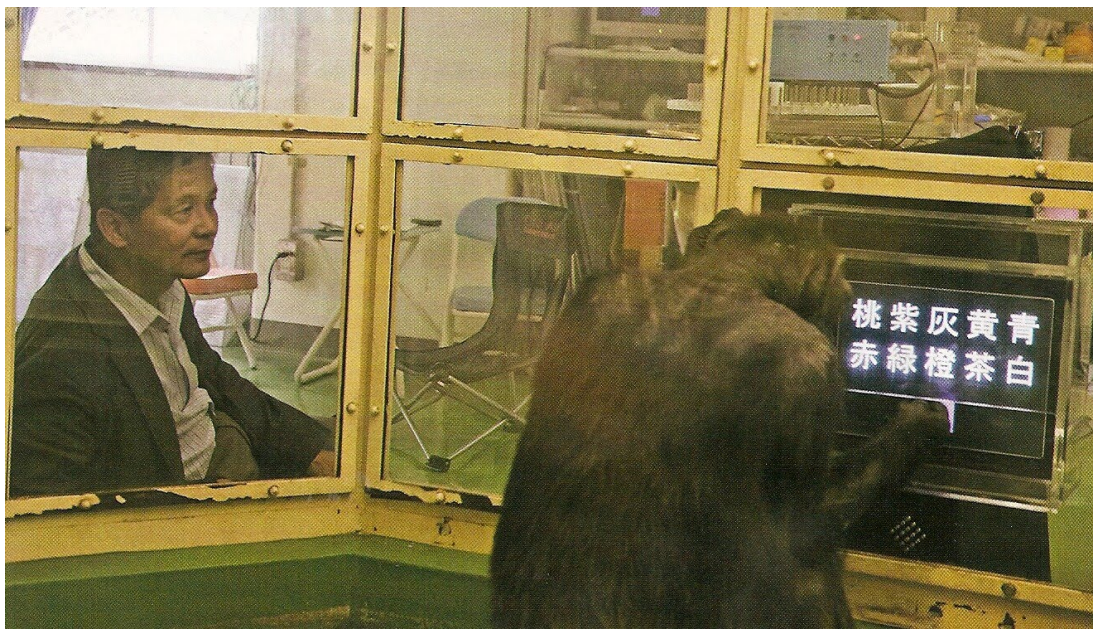
1. **Literature** in the field **dates back decades ago** (often using different terminologies)
2. Various **concepts** were **discovered** and **forgotten** just to be re-discovered later using new names
3. There are some good tutorials on ConvNets but it takes quite some **effort to collect all required information**
4. It can be **hard** to **connect** various **concepts** and terminologies

This talk should give you a head start getting into ConvNets and deep learning, even if you know little or nothing about ANNs.

From the Brain to Convolutional Neural Networks

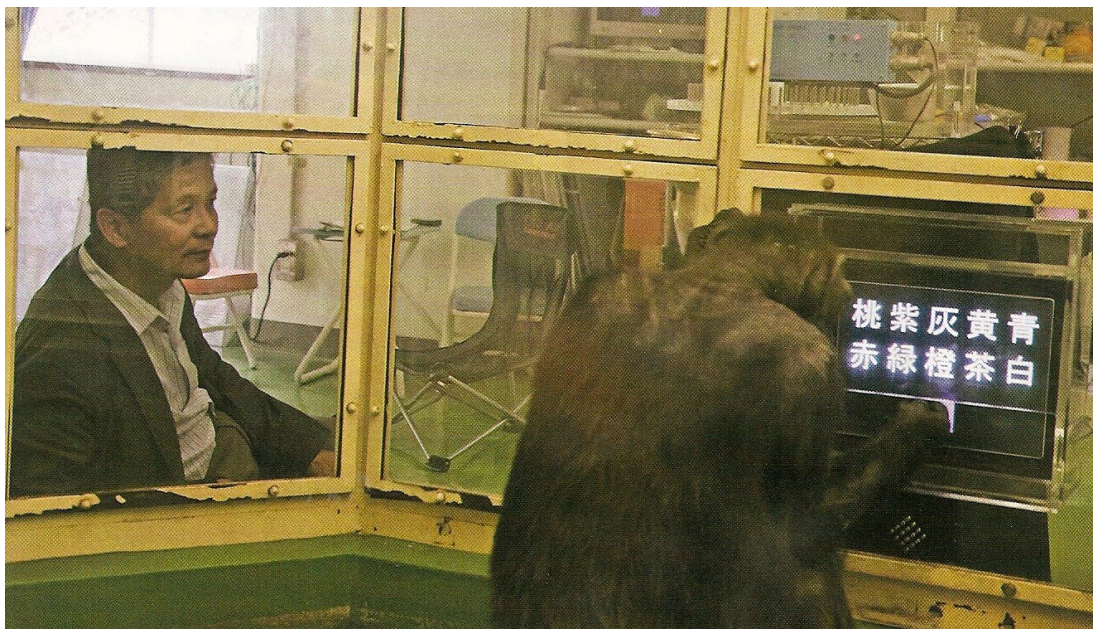
From the Brain to Convolutional Neural Networks

1. The ability to **recognize objects** and **categorize** them is the **most important cognitive activity**



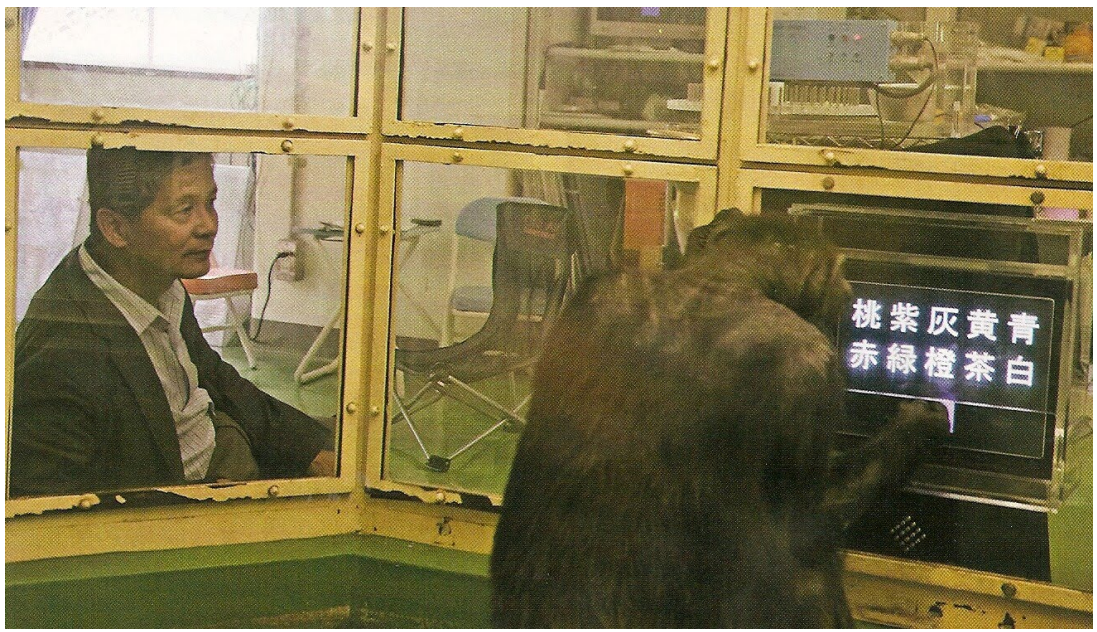
From the Brain to Convolutional Neural Networks

1. The ability to **recognize objects** and **categorize** them is the **most important cognitive activity**
2. Consequently, **animals and humans perform excellent** in these categories as compared to computer vision based systems

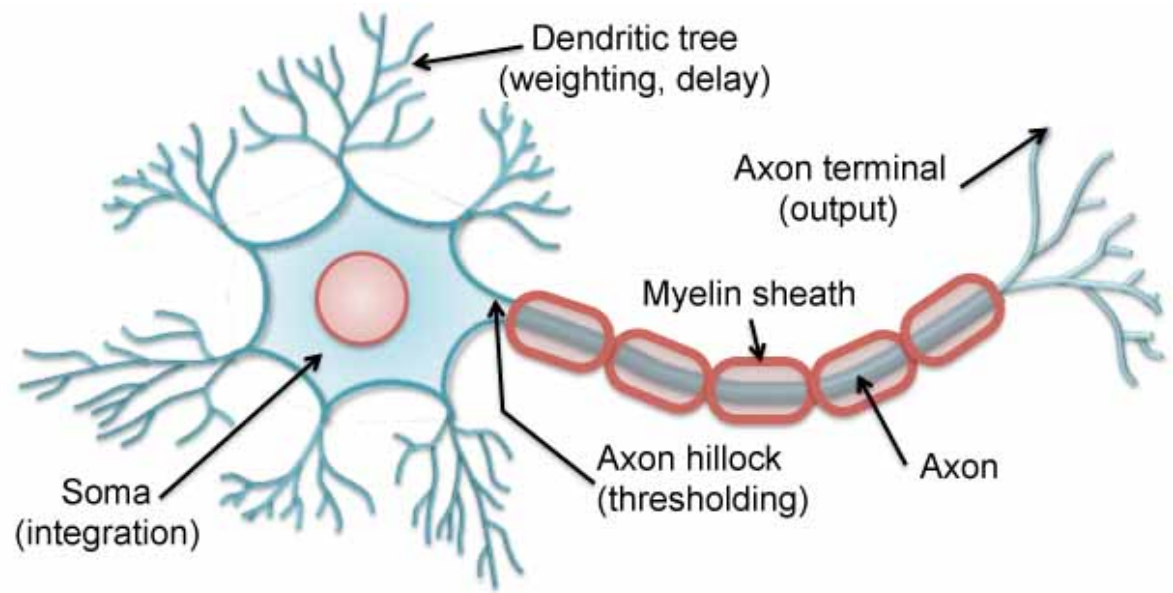


From the Brain to Convolutional Neural Networks

1. The ability to **recognize objects** and **categorize** them is the **most important cognitive activity**
2. Consequently, **animals and humans perform excellent** in these categories as compared to computer vision based systems
3. **Neural networks** are **inspired** by the mammal **brain** and are capable of "programming themselves" or "learn" to solve a problem

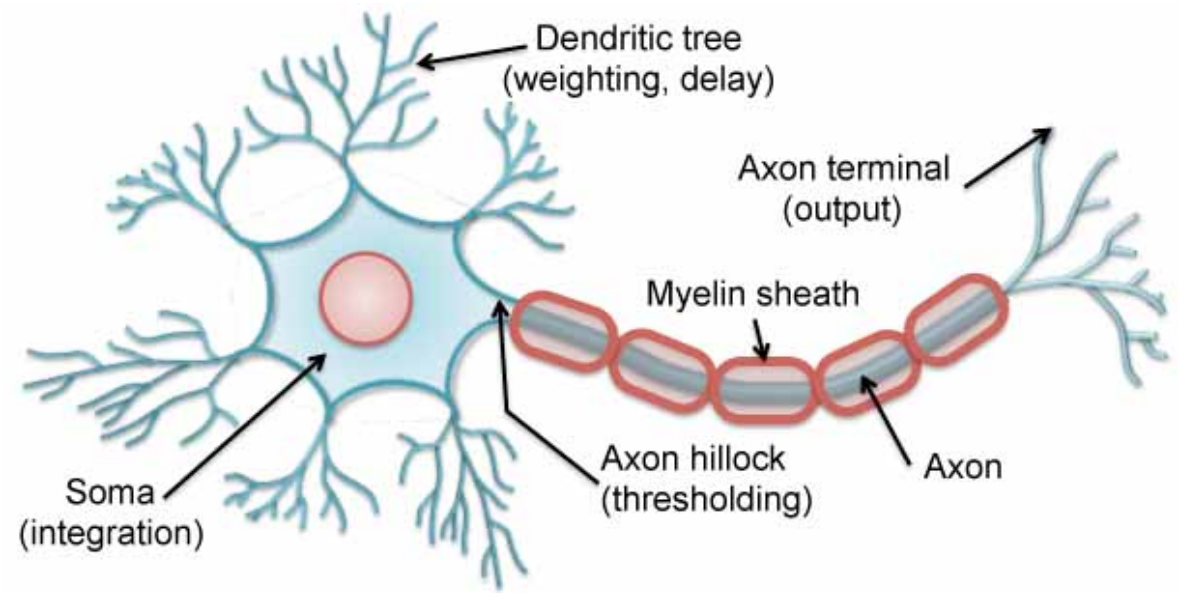


1. The central nervous system of mammals is primarily composed of two classes of cells: **neurons and glial cells**



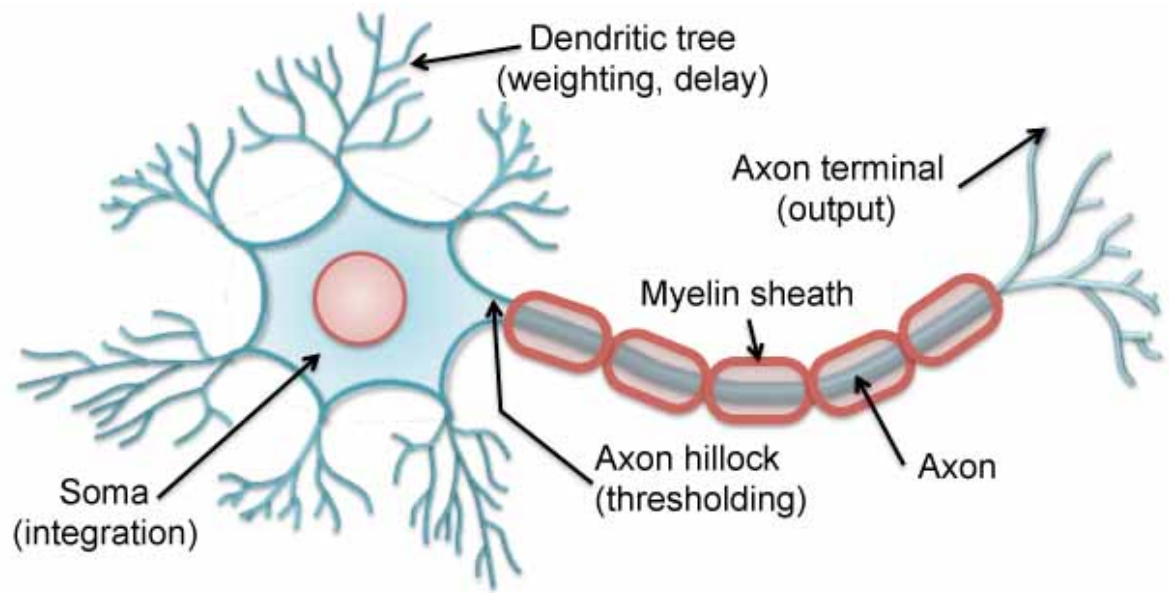
From the Brain to Convolutional Neural Networks

1. The central nervous system of mammals is primarily composed of two classes of cells: **neurons and glial cells**
2. **Glial** cells perform critical functions such as **structural support and metabolic support**

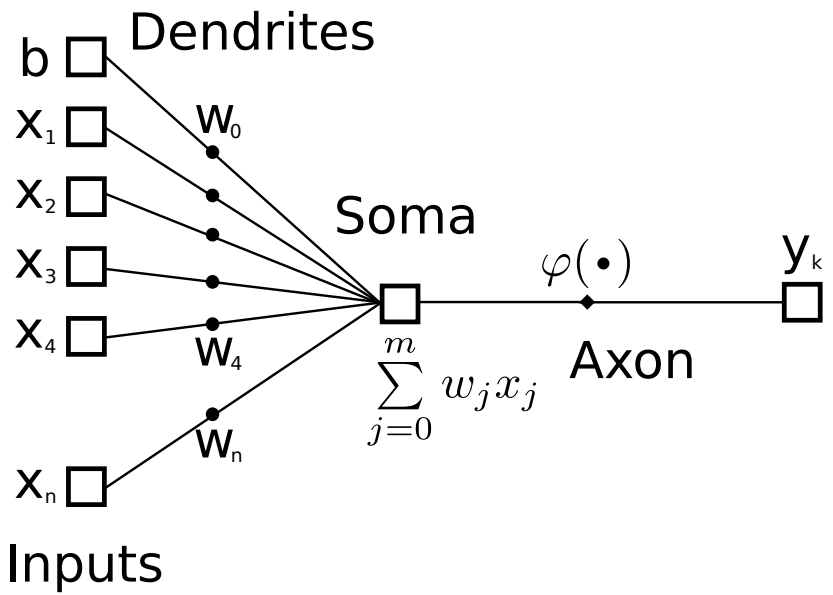


From the Brain to Convolutional Neural Networks

1. The central nervous system of mammals is primarily composed of two classes of cells: **neurons and glial cells**
2. **Glial** cells perform critical functions such as **structural support and metabolic support**
3. With the use of **synapses**, **neurons** form a **network sending signals** to specific target cells



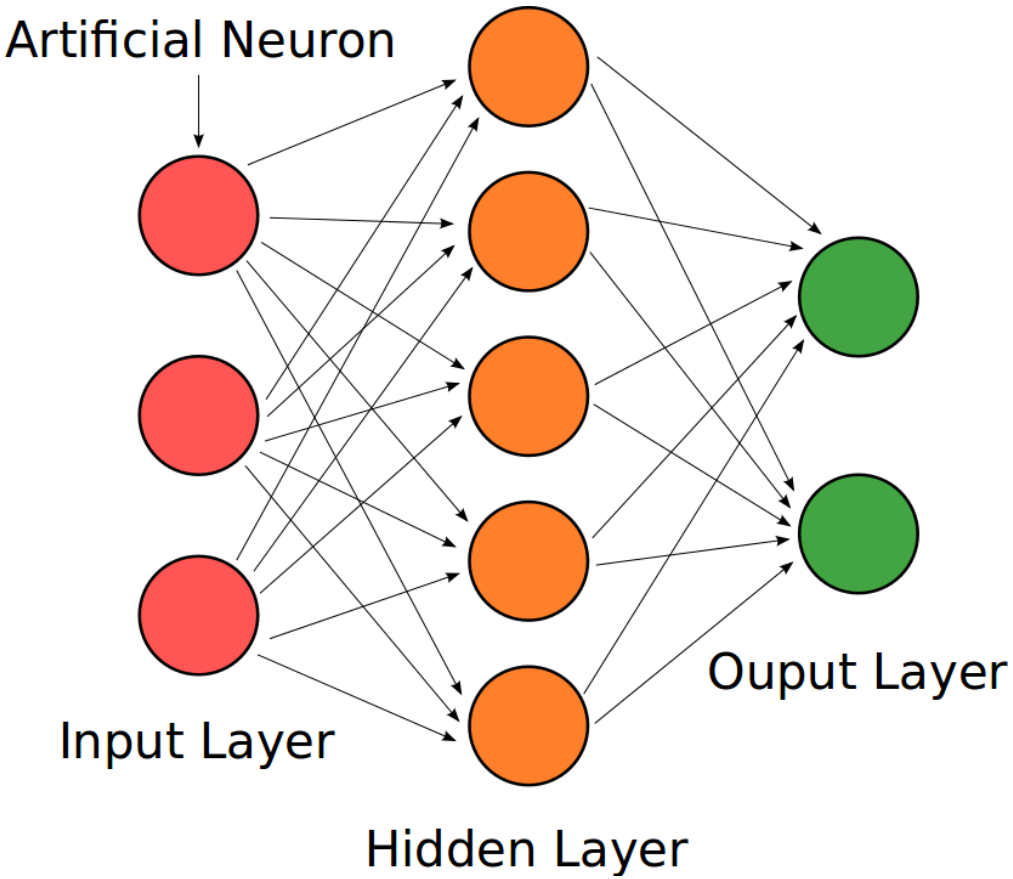
The **neurons modeled** in NN are a **simplification** of the biological counterpart.



Artificial Neuron used in NN

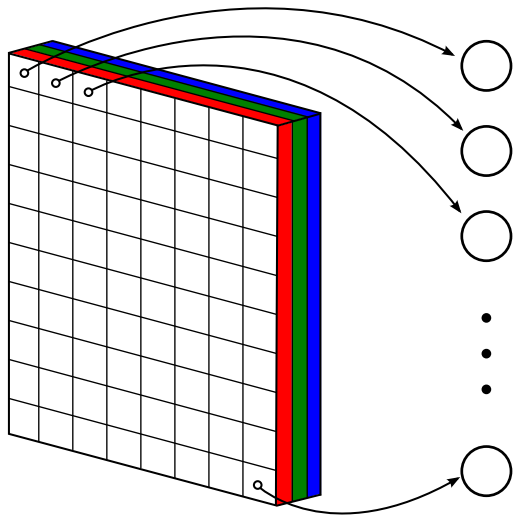
From the Brain to Convolutional Neural Networks

Multiple **fully connected neurons** in multiple layers then **build** a full neural **network**.



From the Brain to Convolutional Neural Networks

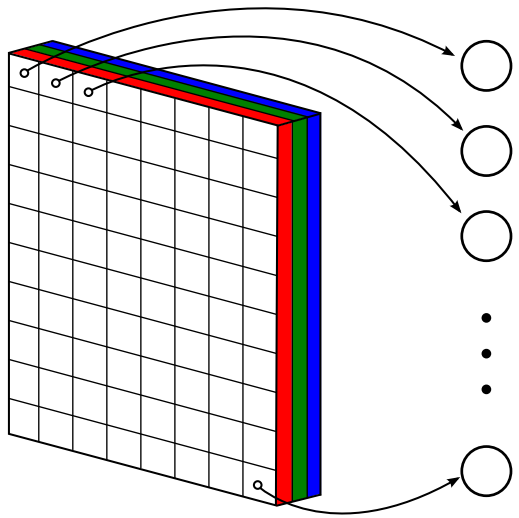
1. Standard **ANNs** are **not well suited** to classify **raw** image **data**



Each input feeds into a separate neuron.

From the Brain to Convolutional Neural Networks

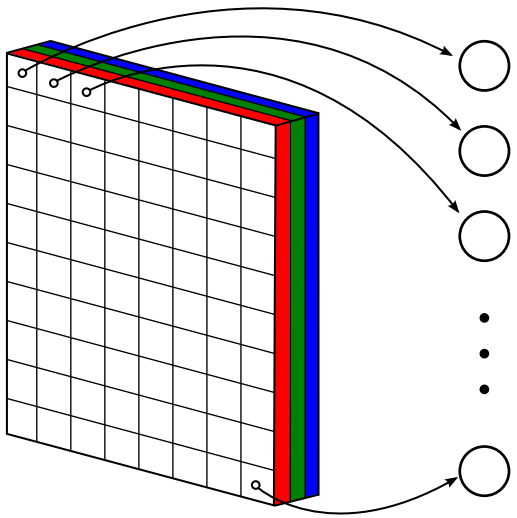
1. Standard **ANNs** are **not well suited** to classify **raw** image **data**
2. Imagine transforming the **pixel** values into a **vector** and put it into an **SVM**



Each input feeds into a separate neuron.

From the Brain to Convolutional Neural Networks

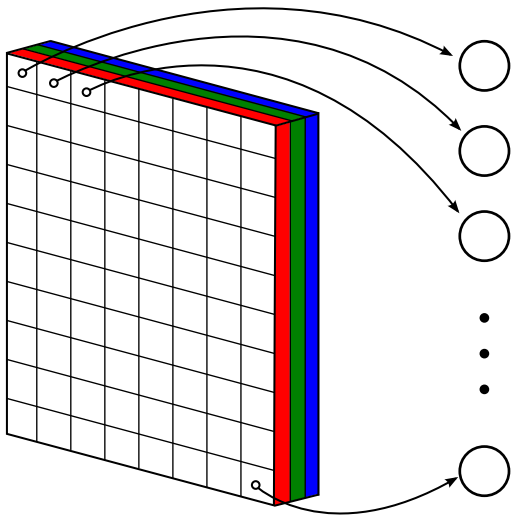
1. Standard **ANNs** are **not well suited** to classify **raw** image **data**
2. Imagine transforming the **pixel** values into a **vector** and put it into an **SVM**
3. A **huge** amount of **training data** is required to train a well generalizing network



Each input feeds into a separate neuron.

From the Brain to Convolutional Neural Networks

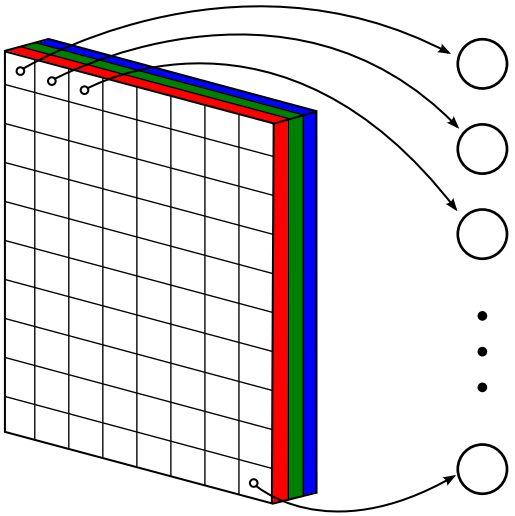
1. Standard **ANNs** are **not well suited** to classify **raw** image **data**
2. Imagine transforming the **pixel** values into a **vector** and put it into an **SVM**
3. A **huge** amount of **training data** is required to train a well generalizing network
4. Encoding **local dependencies** make any type of **image transformation** a **problem**



Each input feeds into a separate neuron.

From the Brain to Convolutional Neural Networks

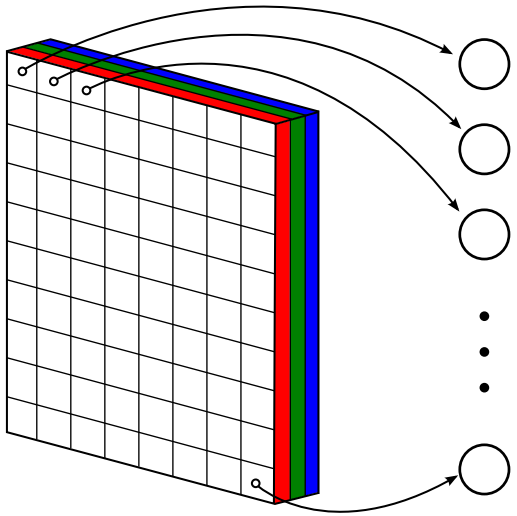
1. Standard **ANNs** are **not well suited** to classify **raw** image **data**
2. Imagine transforming the **pixel** values into a **vector** and put it into an **SVM**
3. A **huge** amount of **training data** is required to train a well generalizing network
4. Encoding **local dependencies** make any type of **image transformation** a **problem**
5. Suitable **image representations** (usually hand-crafted) are **needed**



Each input feeds into a separate neuron.

From the Brain to Convolutional Neural Networks

1. Standard **ANNs** are **not well suited** to classify **raw** image **data**
2. Imagine transforming the **pixel** values into a **vector** and put it into an **SVM**
3. A **huge** amount of **training data** is required to train a well generalizing network
4. Encoding **local dependencies** make any type of **image transformation** a **problem**
5. Suitable **image representations** (usually hand-crafted) are **needed**

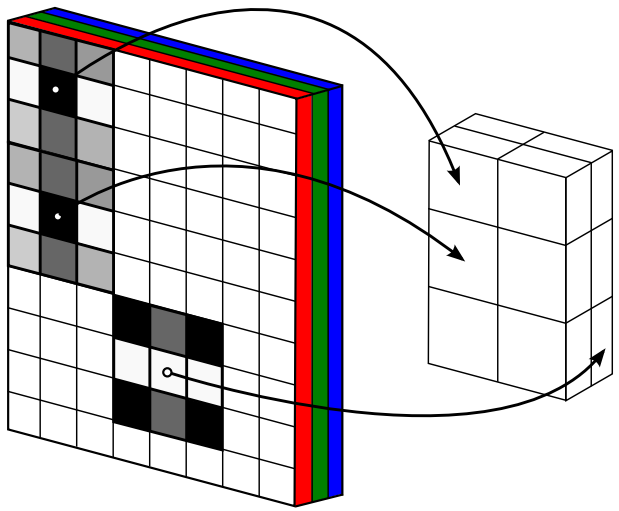


Each input feeds into a separate neuron.

SVMs and other simpler (and faster) methods gradually overtook ANNs in machine learning over the last 15 years

From the Brain to Convolutional Neural Networks

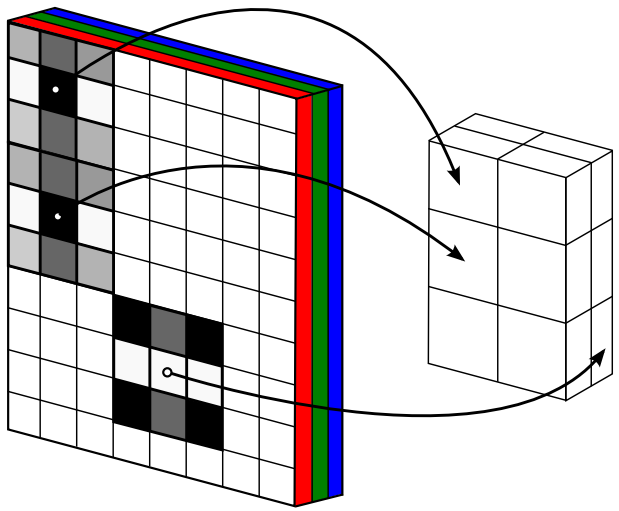
1. ConvNets are **inspired** by the **cat's visual cortex**



Weights are shared between neurons.

From the Brain to Convolutional Neural Networks

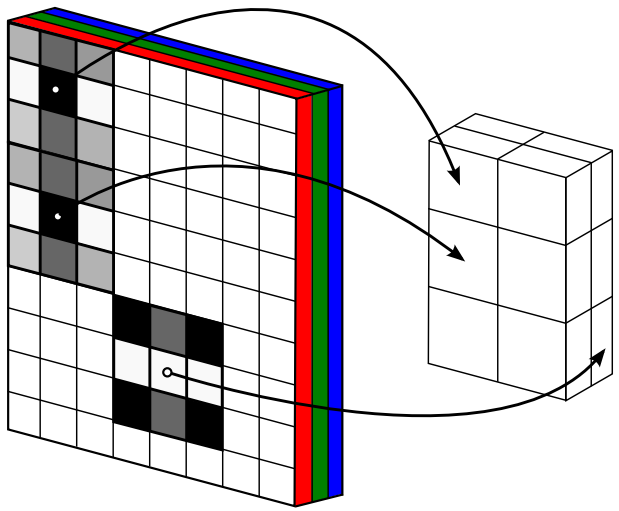
1. ConvNets are **inspired** by the **cat's visual cortex**
2. **Receptive fields** are small sub-regions of the visual field which are **tiled** to cover the entire visual field



Weights are shared between neurons.

From the Brain to Convolutional Neural Networks

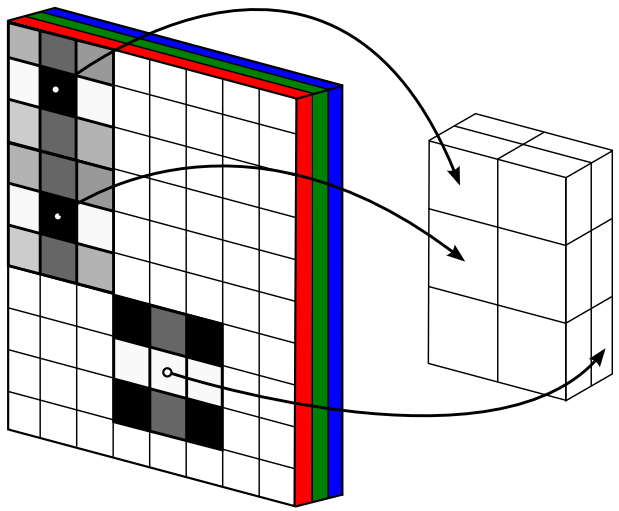
1. ConvNets are **inspired** by the **cat's visual cortex**
2. **Receptive fields** are small sub-regions of the visual field which are **tiled** to cover the entire visual field
3. Multiple **neurons** in a layer **share weights**



Weights are shared between neurons.

From the Brain to Convolutional Neural Networks

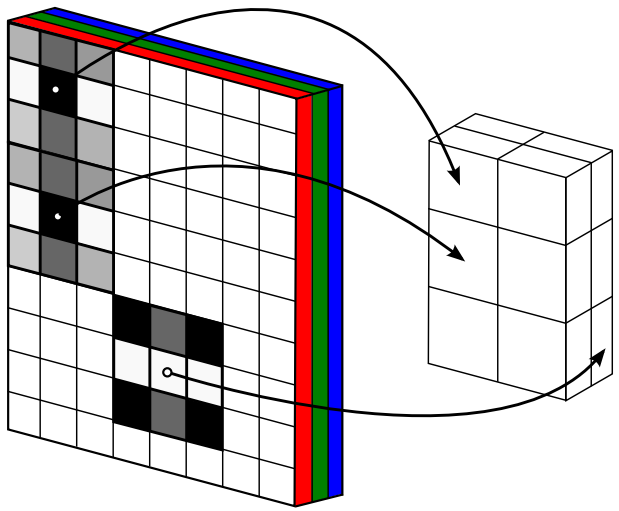
1. ConvNets are **inspired** by the **cat's visual cortex**
2. **Receptive fields** are small sub-regions of the visual field which are **tiled** to cover the entire visual field
3. Multiple **neurons** in a layer **share weights**
4. Features are **detected regardless of position**



Weights are shared between neurons.

From the Brain to Convolutional Neural Networks

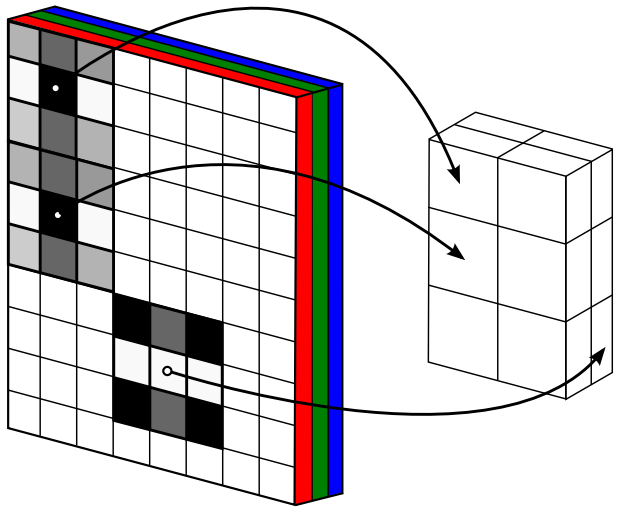
1. ConvNets are **inspired** by the **cat's visual cortex**
2. **Receptive fields** are small sub-regions of the visual field which are **tiled** to cover the entire visual field
3. Multiple **neurons** in a layer **share weights**
4. Features are **detected regardless of position**
5. This **reduces** the **number** of free **parameters** and leads to better generalization



Weights are shared between neurons.

From the Brain to Convolutional Neural Networks

1. ConvNets are **inspired** by the **cat's visual cortex**
2. **Receptive fields** are small sub-regions of the visual field which are **tiled** to cover the entire visual field
3. Multiple **neurons** in a layer **share weights**
4. Features are **detected regardless of position**
5. This **reduces** the **number** of free **parameters** and leads to better generalization

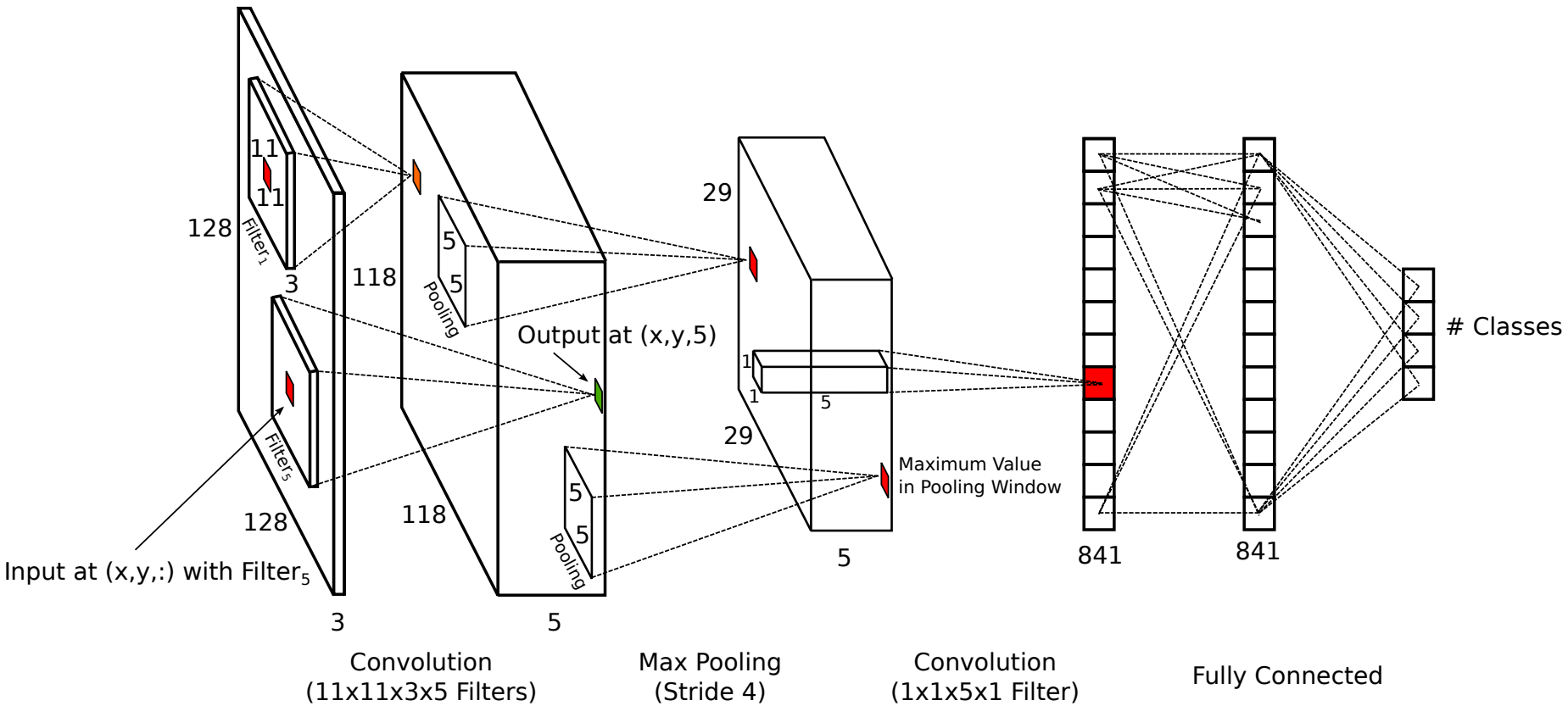


Weights are shared between neurons.

CNNs no more require hand crafted features, they are capable of learning feature detectors!

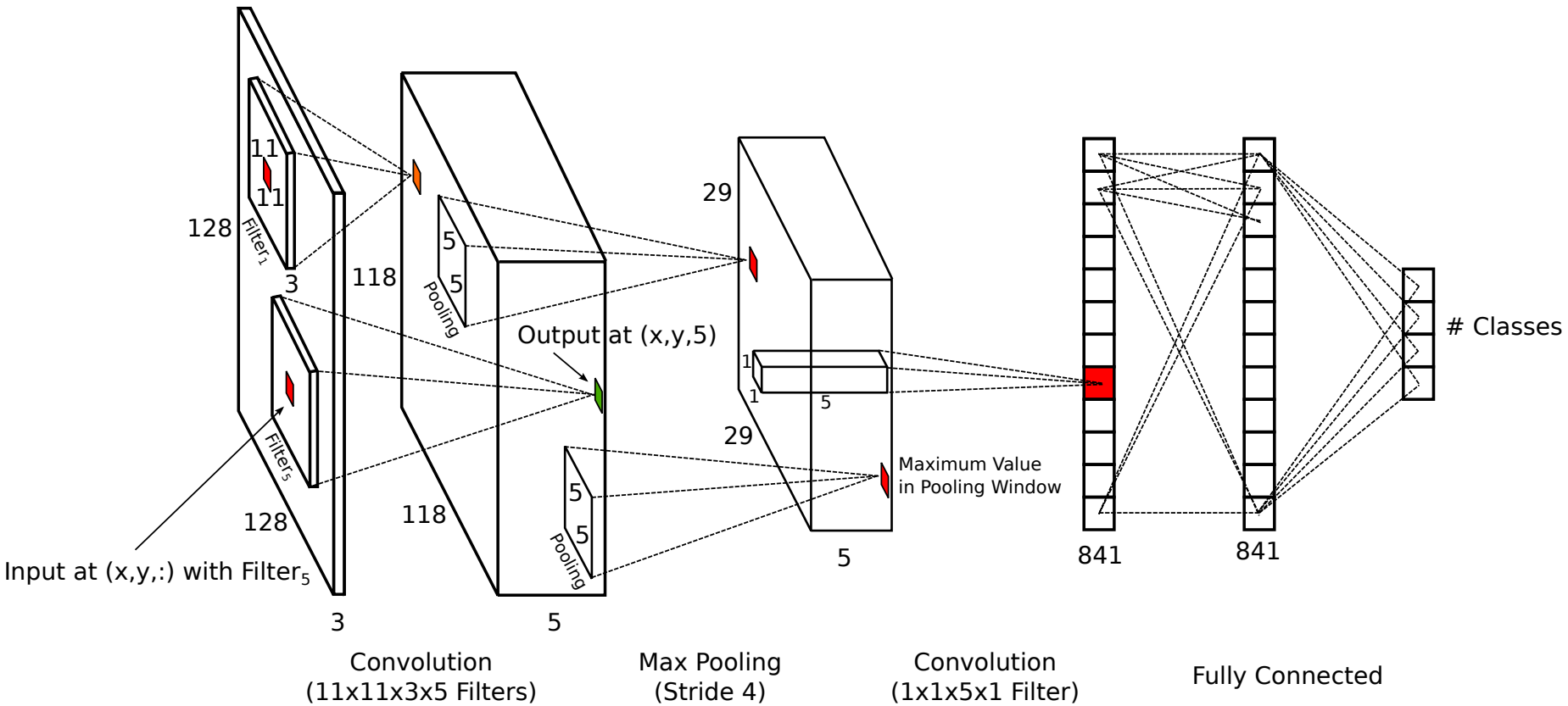
From the Brain to Convolutional Neural Networks

A standard CNN **architecture** is usually described in terms of **layer geometry**. The notion of using **neurons as single units** is split into multiple **layers** which are composed following the same abstraction as classical NNs.



From the Brain to Convolutional Neural Networks

A standard CNN **architecture** is usually described in terms of **layer geometry**. The notion of using **neurons as single units** is split into multiple **layers** which are composed following the same abstraction as classical NNs.



Do not worry, we will come back to this later again!

Note that **certain layers** which are important to the entire net but **do not perform learning** or **change the geometry** are often **left out** in such charts.

Note that **certain layers** which are important to the entire net but **do not perform learning** or **change the geometry** are often **left out** in such charts.

1. **Non-linear layers** are usually left out

Note that **certain layers** which are important to the entire net but **do not perform learning** or **change the geometry** are often **left out** in such charts.

1. **Non-linear layers** are usually left out
2. **Dropout layers** are usually net shown

Note that **certain layers** which are important to the entire net but **do not perform learning** or **change the geometry** are often **left out** in such charts.

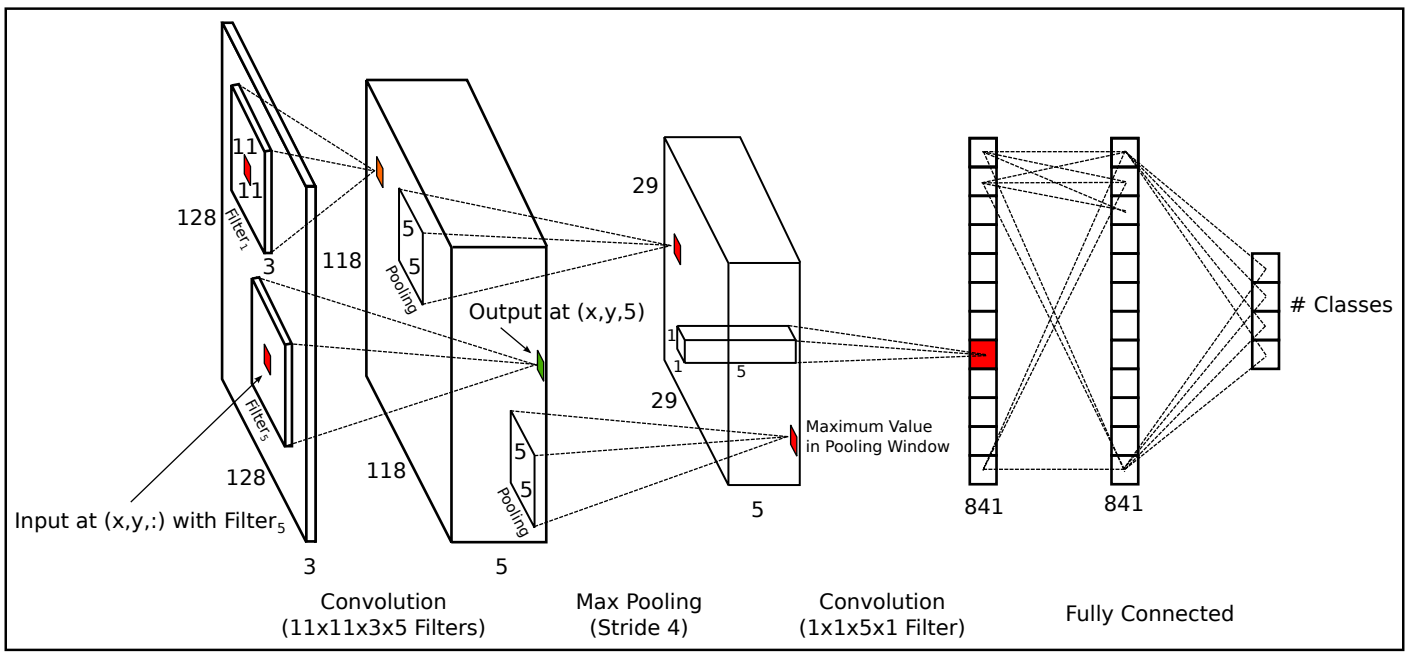
1. **Non-linear layers** are usually left out
2. **Dropout layers** are usually net shown
3. **Softmax and Loss layers** are often simply integrated in the fully connected part

Note that **certain layers** which are important to the entire net but **do not perform learning** or **change the geometry** are often **left out** in such charts.

1. **Non-linear layers** are usually left out
2. **Dropout layers** are usually net shown
3. **Softmax and Loss layers** are often simply integrated in the fully connected part
4. **Normalization** layers are not shown

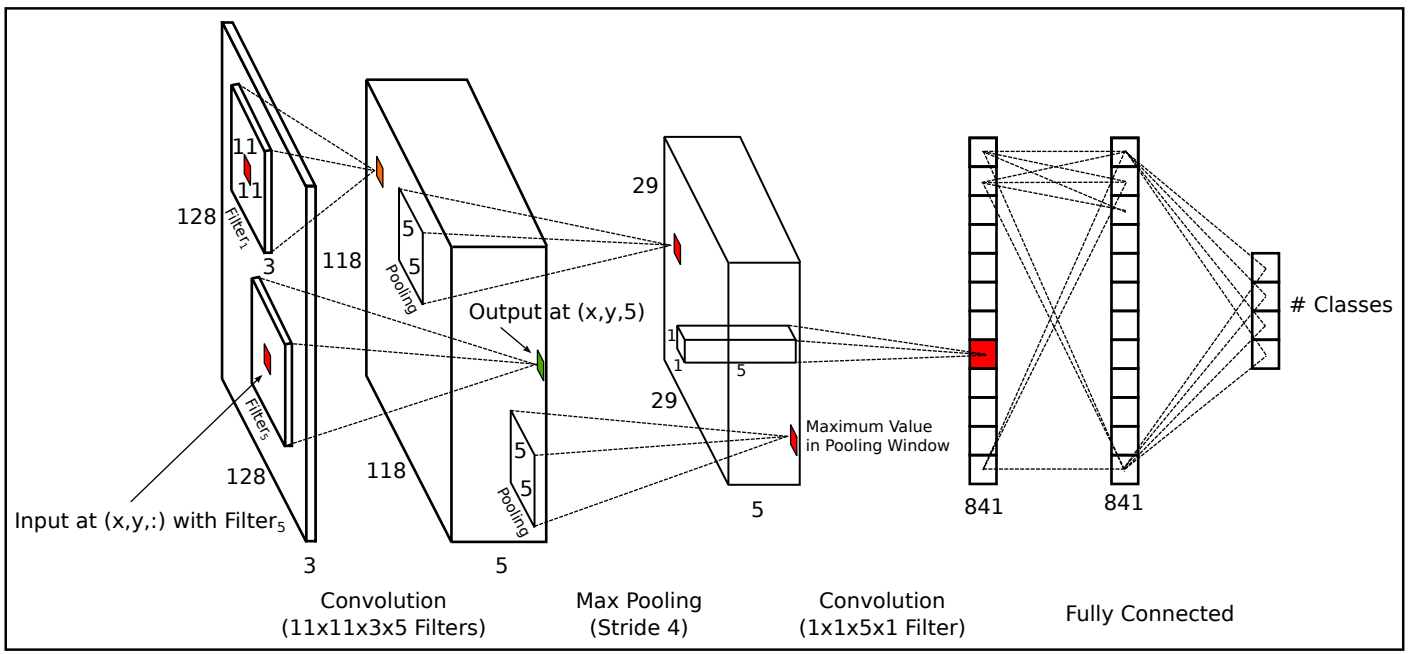
ConvNets are **trained** as classical neural networks are: Using the **back-propagation** algorithm.

Forward Pass: Compute Neuron Activations



ConvNets are **trained** as classical neural networks are: Using the **back-propagation** algorithm.

Backward Pass: Compute Derivative of the Loss w.r.t. each Parameter

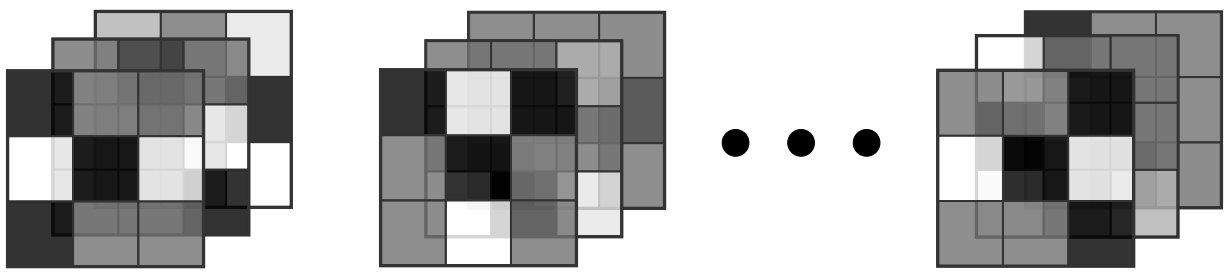


Foward Pass
CNN Building Blocks: Layers

The Convolutional Layer

The Convolutional Layer: Kernel-Maps

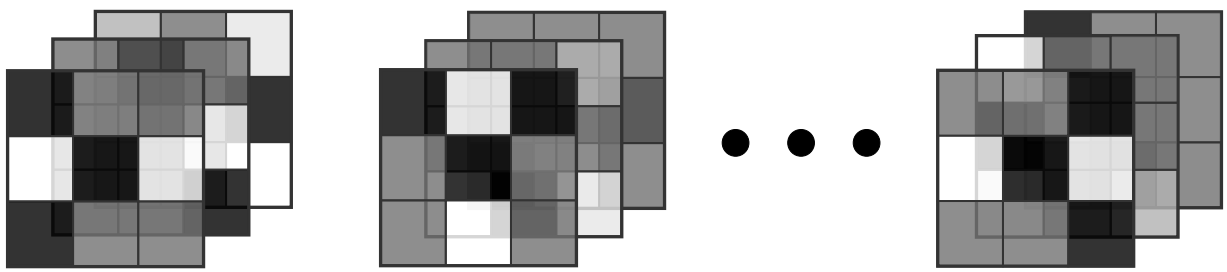
1. **Kernel-maps** are *HxWxDxN* Tensors containing the **shared weights**



3x3x3xN - Kernel Map

The Convolutional Layer: Kernel-Maps

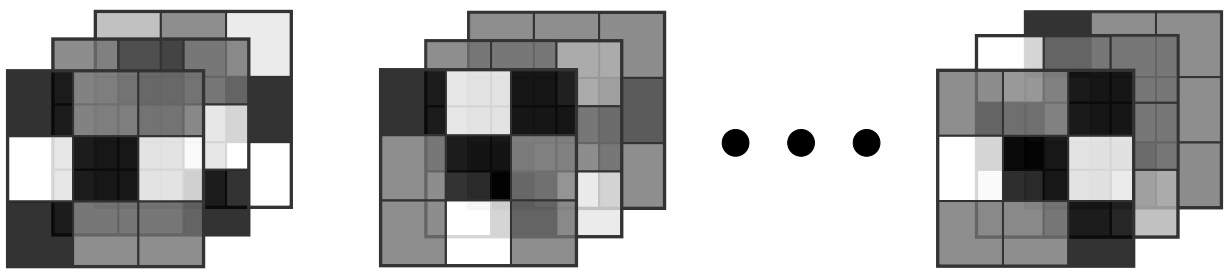
1. **Kernel-maps** are *HxWxDxN* Tensors containing the **shared weights**
2. The kernel-maps are **convolved with** the **input** (feature-map)



3x3x3xN - Kernel Map

The Convolutional Layer: Kernel-Maps

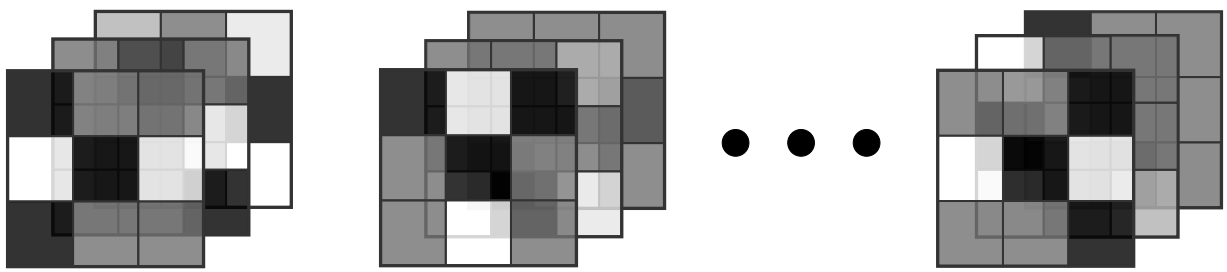
1. **Kernel-maps** are *HxWxDxN* Tensors containing the **shared weights**
2. The kernel-maps are **convolved with** the **input** (feature-map)
3. The **depth dimension** *D* of kernel-map and input must be **equal**



3x3x3xN - Kernel Map

The Convolutional Layer: Kernel-Maps

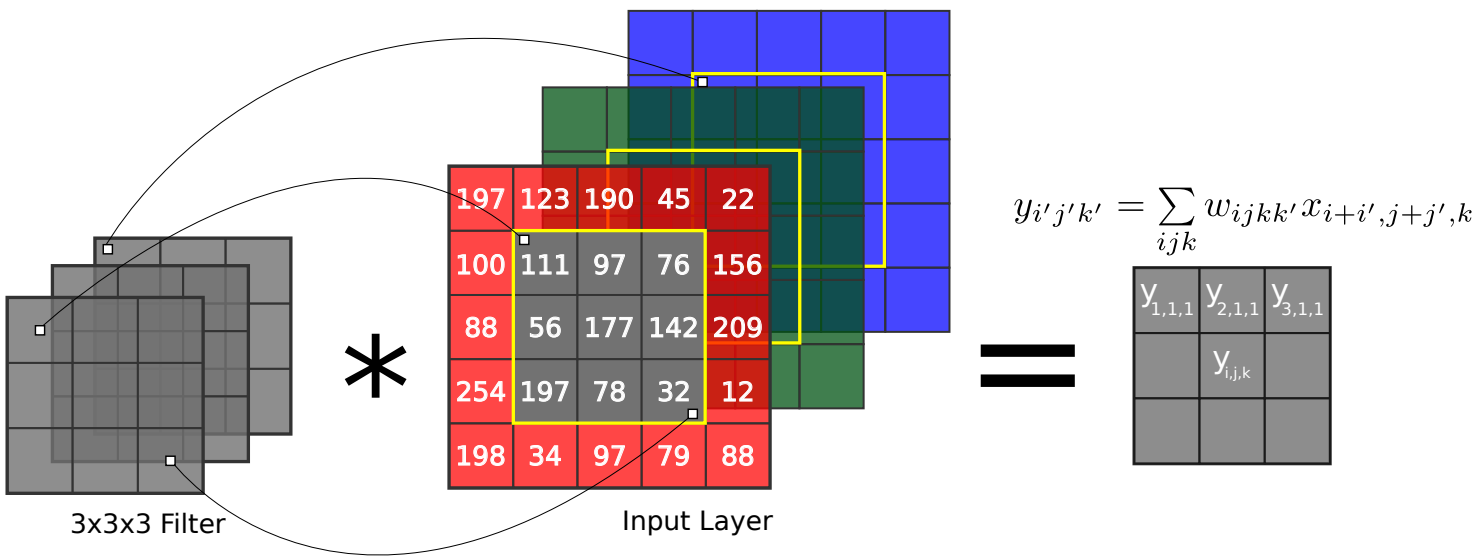
1. **Kernel-maps** are *HxWxDxN* Tensors containing the **shared weights**
2. The kernel-maps are **convolved with** the **input** (feature-map)
3. The **depth dimension** *D* of kernel-map and input must be **equal**
4. **Convolution** of each of the *HxWxD* filters in a kernel-map **gives a 2D-result** which is **concatenated** to a **3D-output**



3x3x3xN - Kernel Map

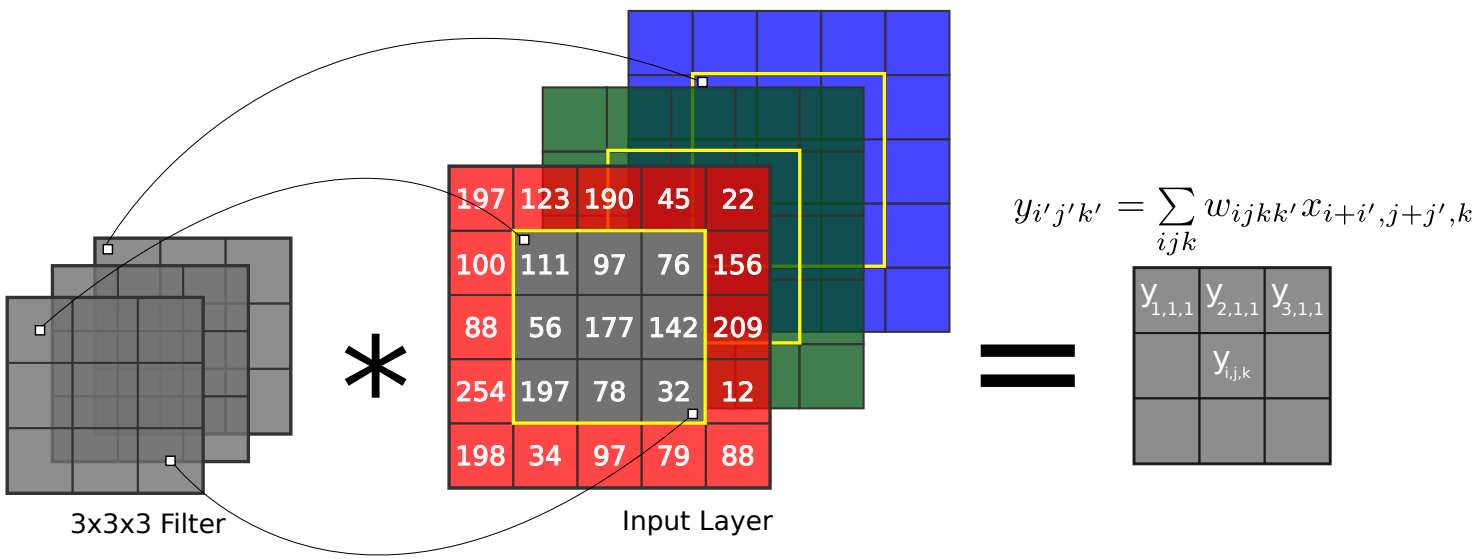
The Convolutional Layer: Forward Pass

1. A *HxWxDxN* **kernel-map** is **convolved** with a *XxYxD* **feature-map** to produce a $X'xY'xN$ output



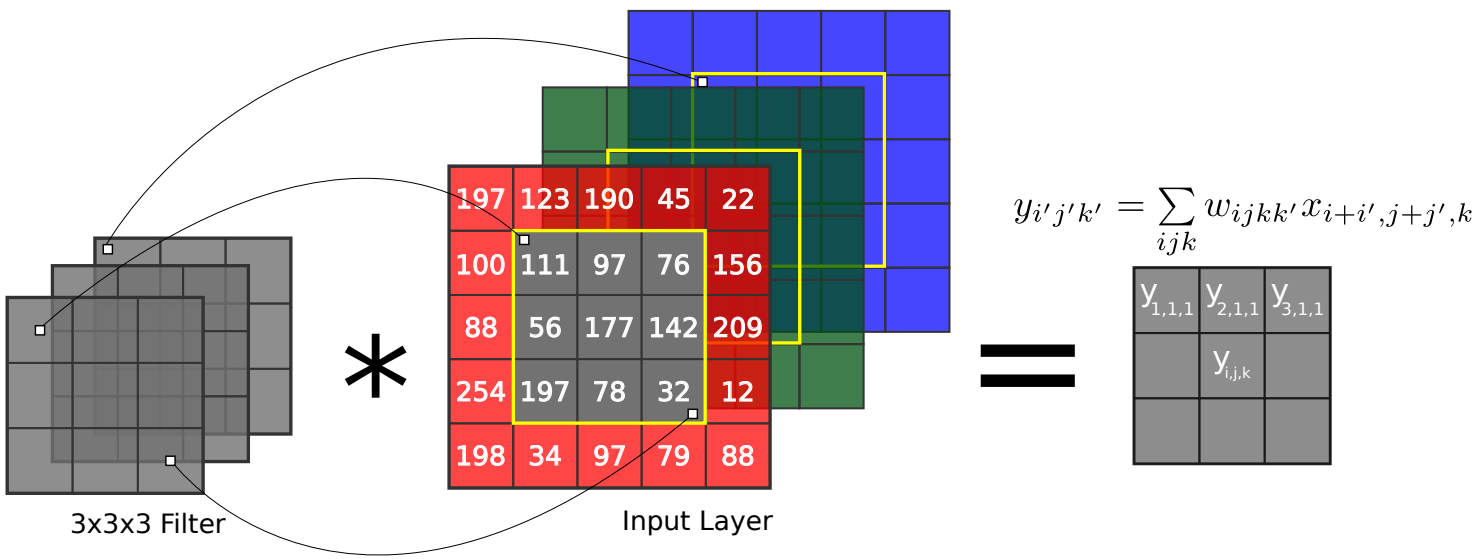
The Convolutional Layer: Forward Pass

1. A *HxWxDxN* **kernel-map** is **convolved** with a *XxYxD* **feature-map** to produce a $X'xY'xN$ output
2. The actual **dimensions** of the **output** depend on **stride** and **padding**



The Convolutional Layer: Forward Pass

1. A *HxWxDxN* **kernel-map** is **convolved** with a *XxYxD* **feature-map** to produce a $X'xY'xN$ output
2. The actual **dimensions** of the **output** depend on **stride** and **padding**
3. Note: The below charts actually performs **cross-correlation** instead of convolution (as implemented in MatConvNet and Caffe) which does not matter as the filters are initialized randomly



The Convolutional Layer: Feature Visualization

Visualizing the **input patterns** that cause certain activations in a ConvNet **helps understanding** the learning performed by the net.

The Convolutional Layer: Feature Visualization

Visualizing the **input patterns** that cause certain activations in a ConvNet **helps understanding** the learning performed by the net.

1. At **lower levels** (near the input to the net) ConvNets will perform **basic edge detection**

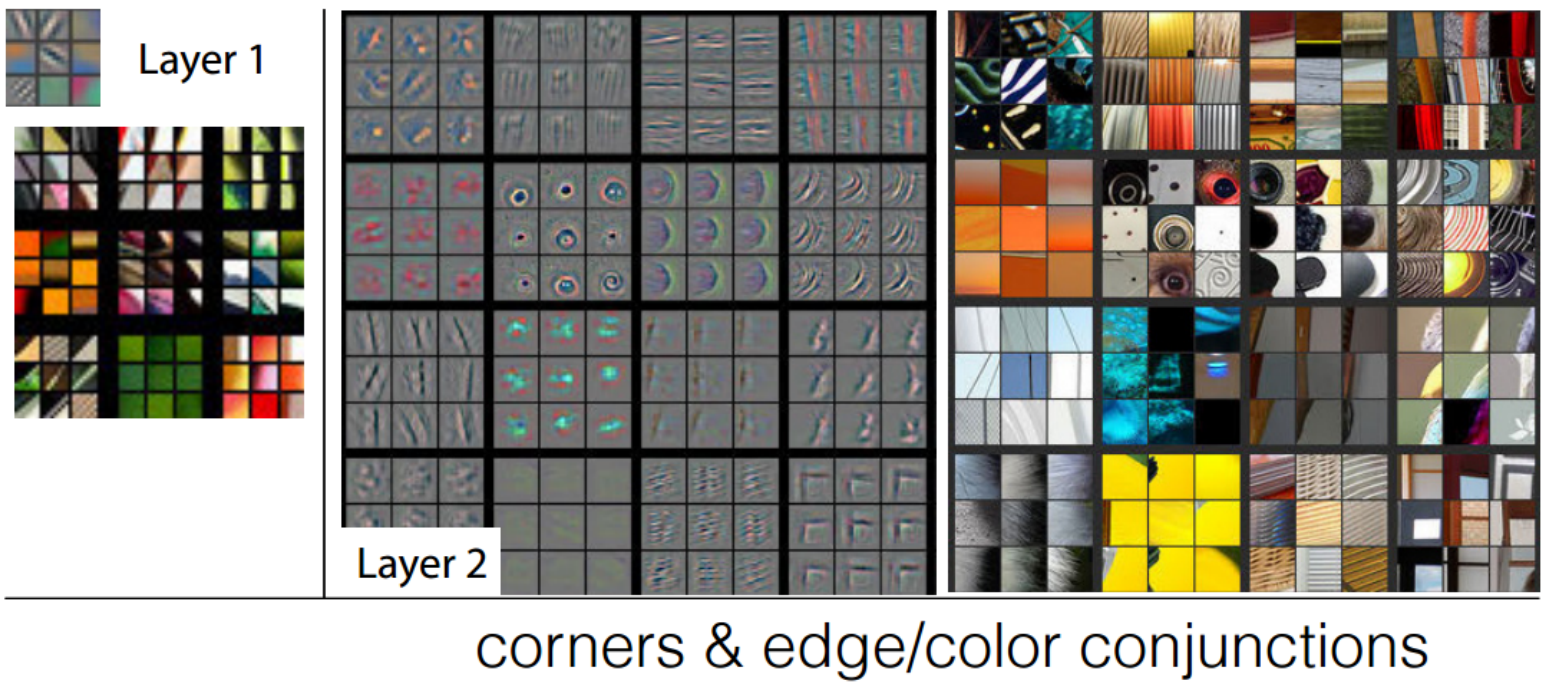


Figure : Zeiler et al.: Input pattern that caused a given activation

The Convolutional Layer: Feature Visualization

Visualizing the **input patterns** that cause certain activations in a ConvNet **helps understanding** the learning performed by the net.

1. At **lower levels** (near the input to the net) ConvNets will perform **basic edge detection**
2. At **higher levels** more and more **abstract** features are generated

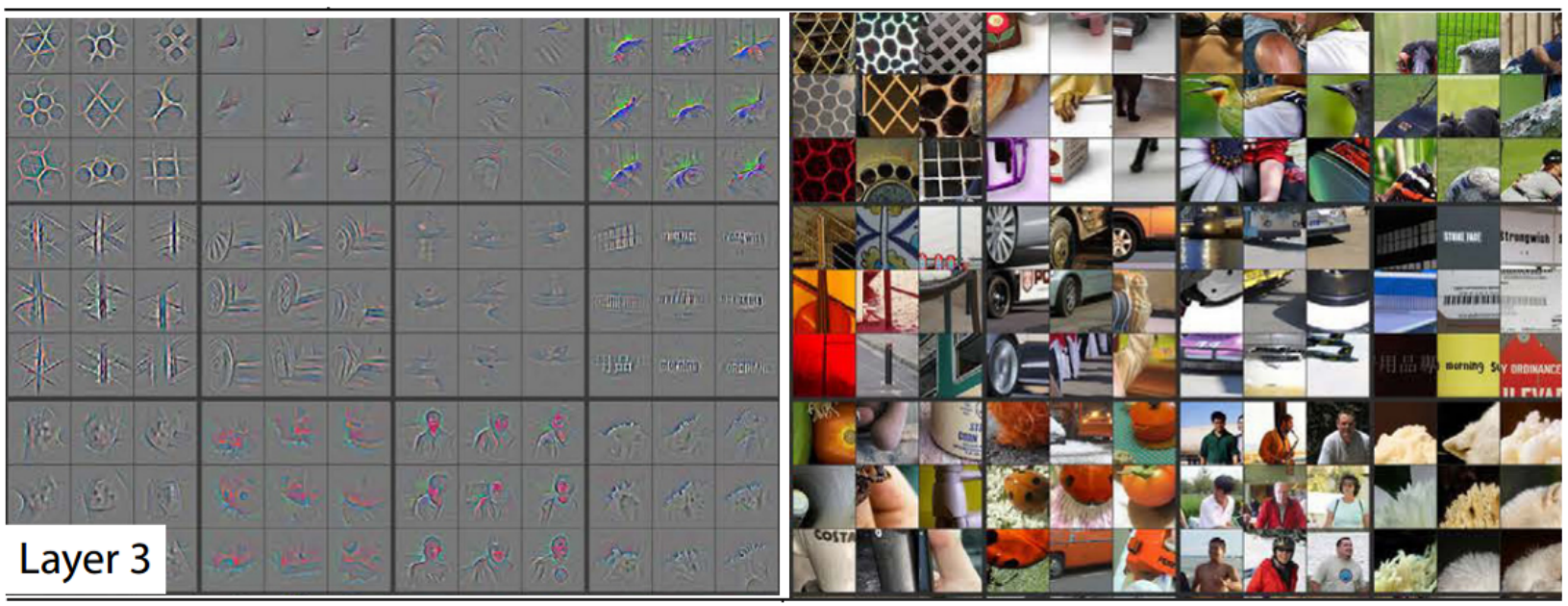


Figure : Zeiler et al.: Input pattern that caused a given activation

The Convolutional Layer: Feature Visualization

Visualizing the **input patterns** that cause certain activations in a ConvNet **helps understanding** the learning performed by the net.

1. At **lower levels** (near the input to the net) ConvNets will perform **basic edge detection**
2. At **higher levels** more and more **abstract** features are generated

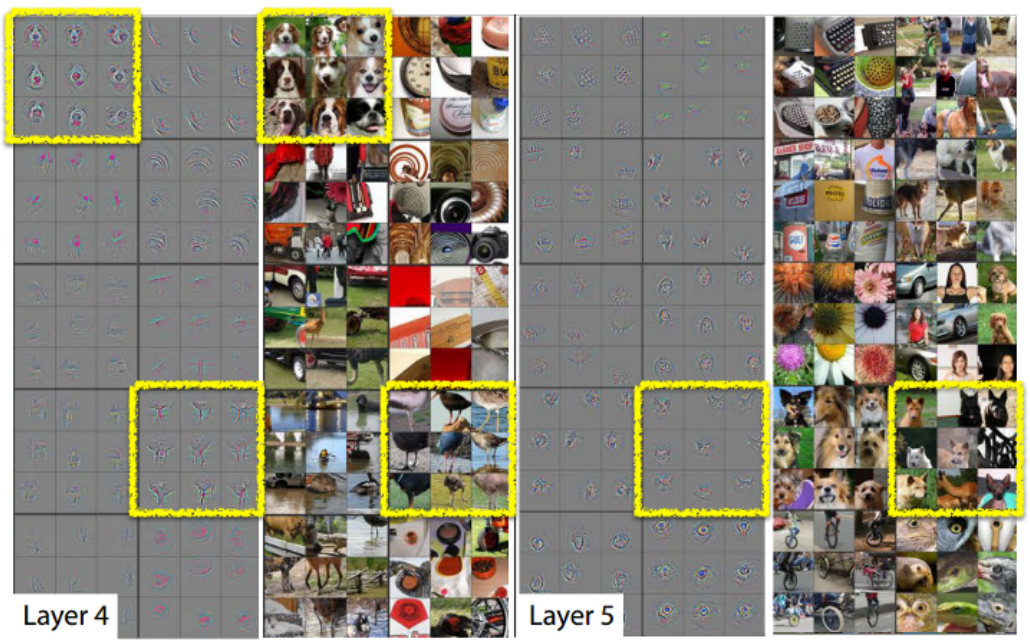


Figure : Zeiler et al.: Input pattern that caused a given activation

The Convolution Layer: Implementation

1. A **naive implementation** of the convolution layer is **very slow**

The Convolution Layer: Implementation

1. A **naive implementation** of the convolution layer is **very slow**
2. **GEMM** is currently the **backbone** of all **ConvNet** implementations

The Convolution Layer: Implementation

1. A **naive implementation** of the convolution layer is **very slow**
2. **GEMM** is currently the **backbone** of all **ConvNet** implementations
3. By using highly optimized **linear algebra packages**, drastic speed improvements can be achieved

The Convolution Layer: Implementation

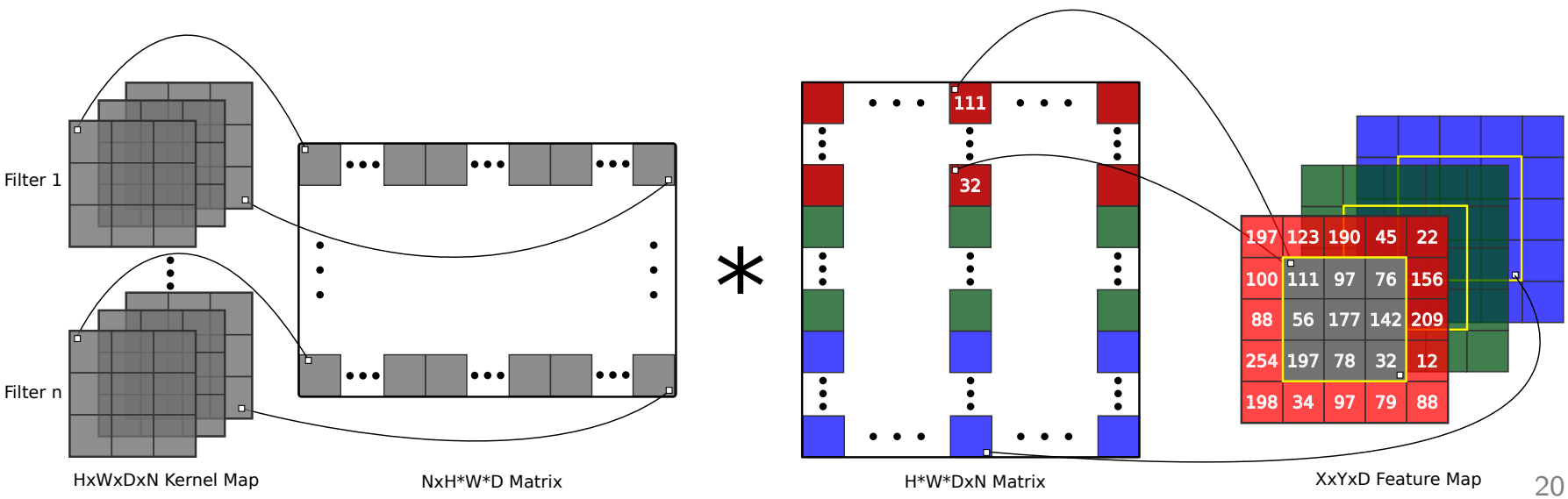
1. A **naive implementation** of the convolution layer is **very slow**
2. **GEMM** is currently the **backbone** of all **ConvNet** implementations
3. By using highly optimized **linear algebra packages**, drastic speed improvements can be achieved
4. Consequently the **forward pass** is reduced to a single **matrix multiplication**

The Convolution Layer: Implementation

1. A **naive implementation** of the convolution layer is **very slow**
2. **GEMM** is currently the **backbone** of all **ConvNet** implementations
3. By using highly optimized **linear algebra packages**, drastic speed improvements can be achieved
4. Consequently the **forward pass** is reduced to a single **matrix multiplication**
5. The **layout of data in memory** can have a significant **impact on performance** (cache optimization)

The Convolution Layer: Implementation

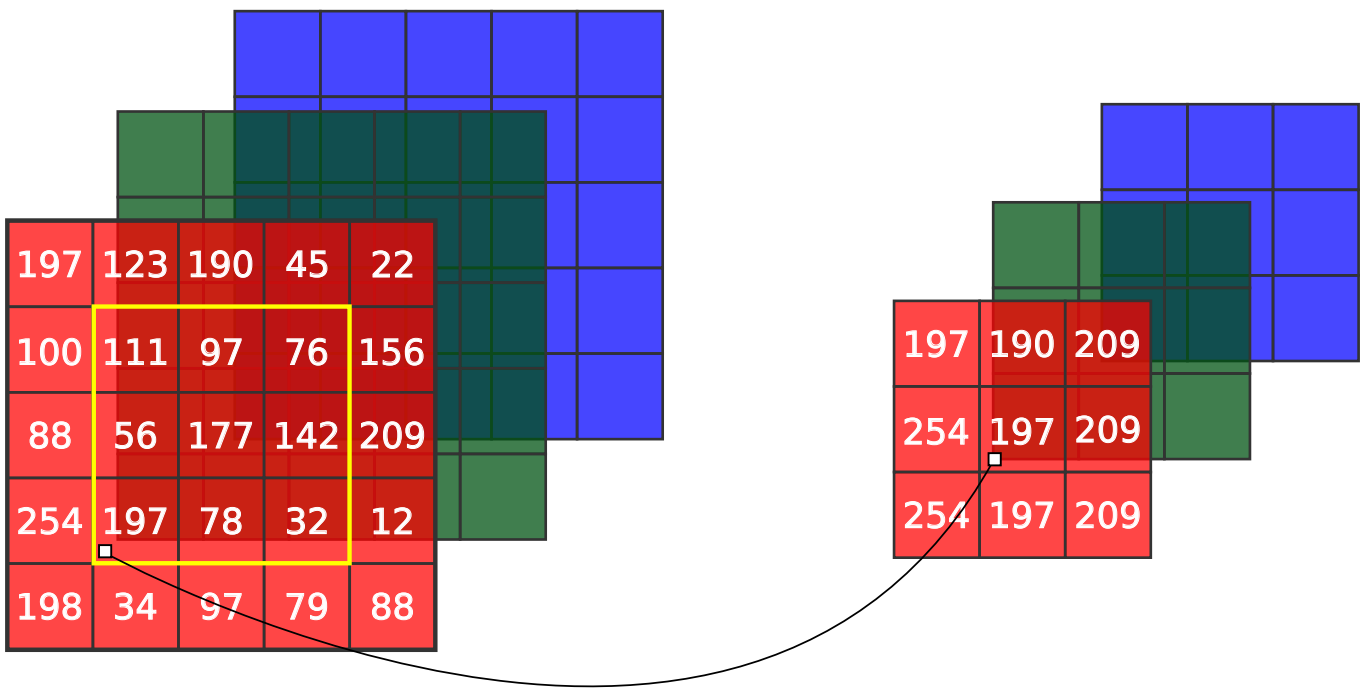
1. A **naive implementation** of the convolution layer is **very slow**
2. **GEMM** is currently the **backbone** of all **ConvNet** implementations
3. By using highly optimized **linear algebra packages**, drastic speed improvements can be achieved
4. Consequently the **forward pass** is reduced to a single **matrix multiplication**
5. The **layout of data in memory** can have a significant **impact on performance** (cache optimization)



The Pooling Layer

The Pooling Layer: Max-pooling

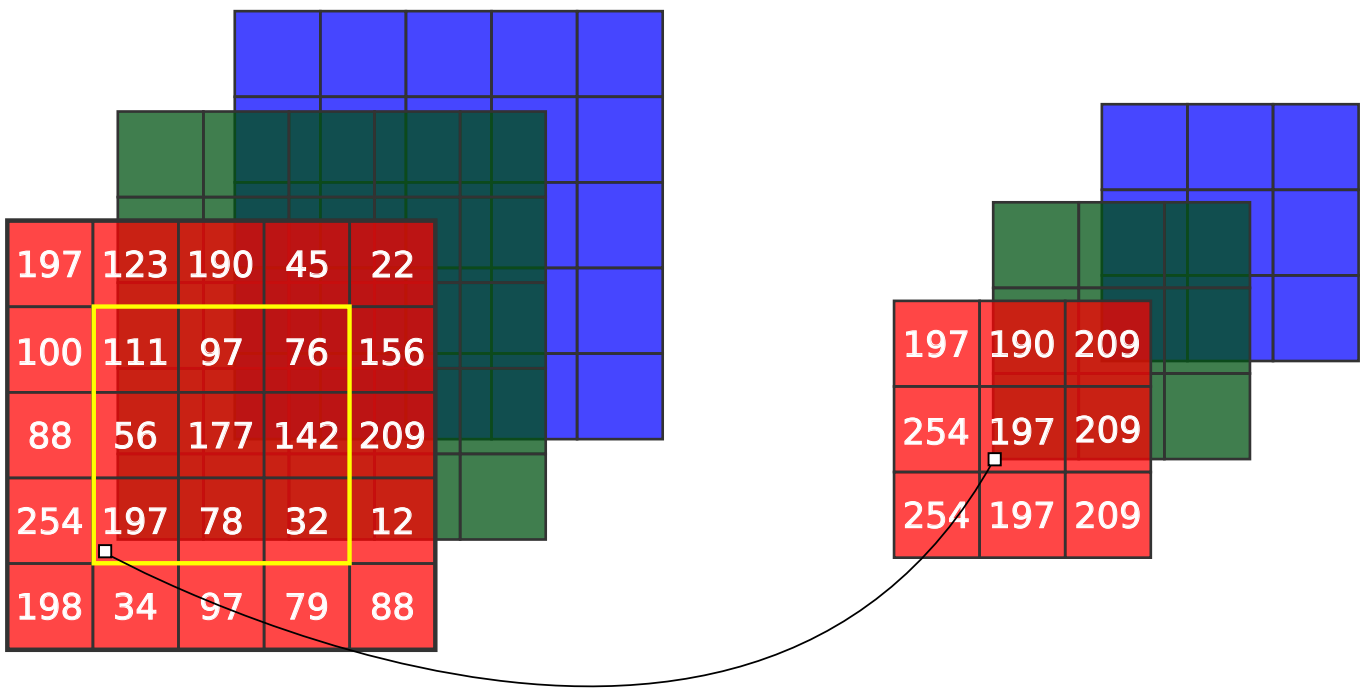
Almost all modern deep architectures rely on **max-pooling** nowadays.



The Pooling Layer: Max-pooling

Almost all modern deep architectures rely on **max-pooling** nowadays.

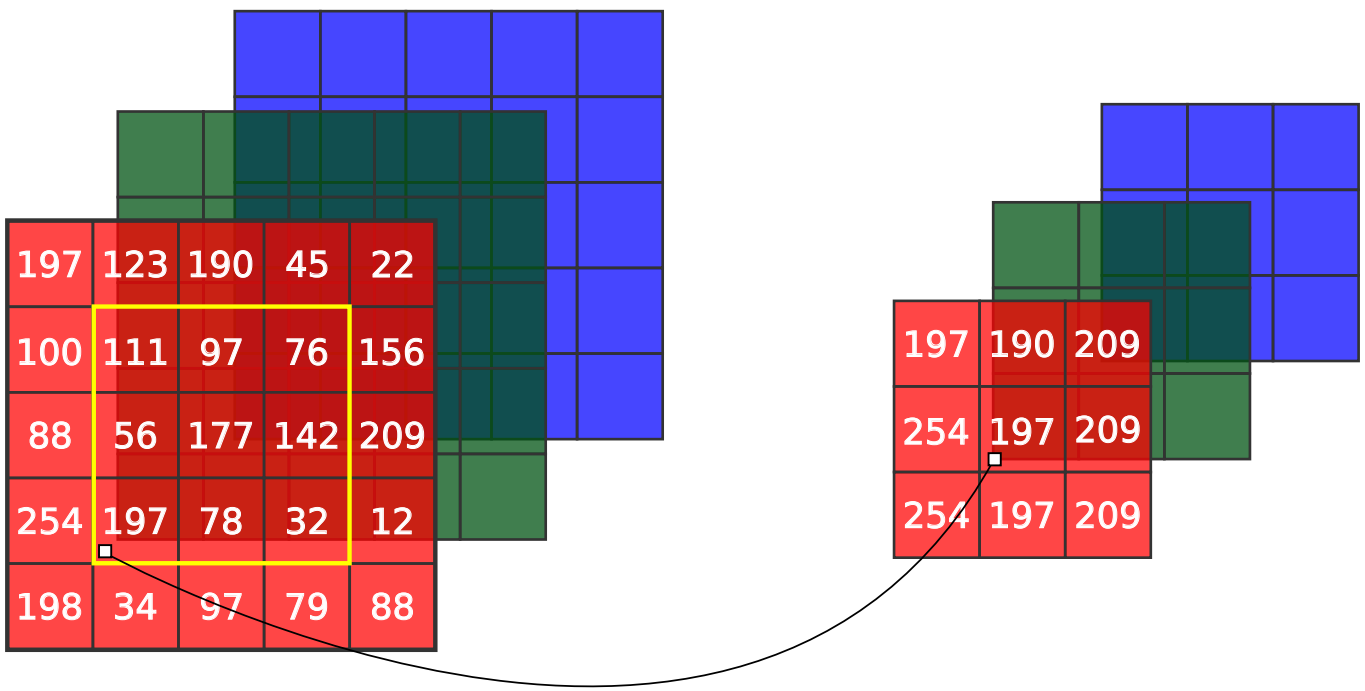
1. Max-pooling is used to improve **translation invariance**



The Pooling Layer: Max-pooling

Almost all modern deep architectures rely on **max-pooling** nowadays.

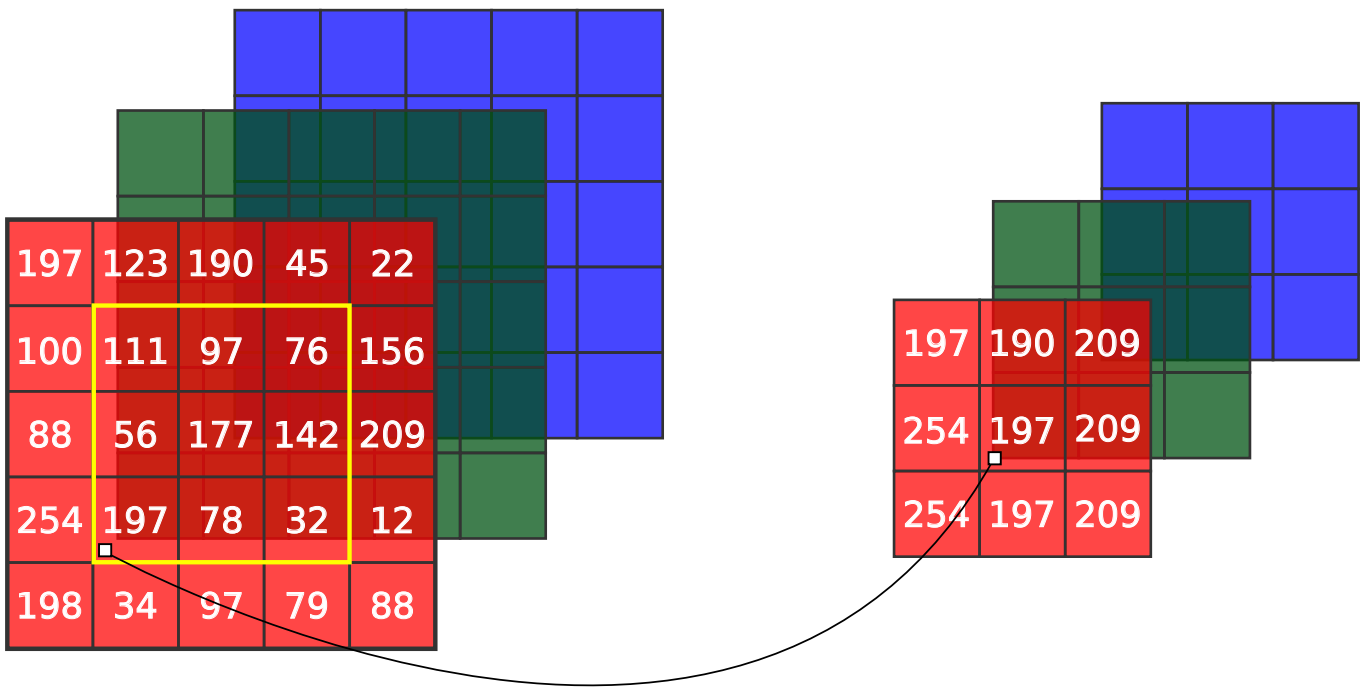
1. Max-pooling is used to improve **translation invariance**
2. Max-pooling **reduces the dimensionality** of the intermediate representation



The Pooling Layer: Max-pooling

Almost all modern deep architectures rely on **max-pooling** nowadays.

1. Max-pooling is used to improve **translation invariance**
2. Max-pooling **reduces the dimensionality** of the intermediate representation
3. Usually done directly **after a convolution layer**



The Non-Linear Layer

The Non-Linear Layer

The non-linear **layer corresponds to computing the activation** function $\varphi(\cdot)$ on the weighted sum of inputs in the neuron.

The Non-Linear Layer

The non-linear **layer corresponds to computing the activation** function $\varphi(\cdot)$ on the weighted sum of inputs in the neuron.

1. A multi-layer network with **linear activation function** can be represented by a **single-layer network**

The Non-Linear Layer

The non-linear **layer corresponds to computing the activation** function $\varphi(\cdot)$ on the weighted sum of inputs in the neuron.

1. A multi-layer network with **linear activation function** can be represented by a **single-layer network**
2. The non-linear layer is used to **increase the nonlinearity** of the net

The Non-Linear Layer

The non-linear **layer corresponds to computing the activation** function $\varphi(\cdot)$ on the weighted sum of inputs in the neuron.

1. A multi-layer network with **linear activation function** can be represented by a **single-layer network**
2. The non-linear layer is used to **increase the nonlinearity** of the net
3. The **output** of the non-linear layer are the **neuron activations**

The Non-Linear Layer

The non-linear **layer corresponds to computing the activation** function $\varphi(\cdot)$ on the weighted sum of inputs in the neuron.

1. A multi-layer network with **linear activation function** can be represented by a **single-layer network**
2. The non-linear layer is used to **increase the nonlinearity** of the net
3. The **output** of the non-linear layer are the **neuron activations**

$\varphi(\cdot)$: Hyperbolic Tangent

$$y = tanh(x)$$

The Non-Linear Layer

The non-linear **layer corresponds to computing the activation** function $\varphi(\cdot)$ on the weighted sum of inputs in the neuron.

1. A multi-layer network with **linear activation function** can be represented by a **single-layer network**
2. The non-linear layer is used to **increase the nonlinearity** of the net
3. The **output** of the non-linear layer are the **neuron activations**

$\varphi(\cdot)$: Sigmoid

$$y = \frac{1}{(1 + e^{-x})}$$

The Non-Linear Layer

The non-linear **layer corresponds to computing the activation** function $\varphi(\cdot)$ on the weighted sum of inputs in the neuron.

1. A multi-layer network with **linear activation function** can be represented by a **single-layer network**
2. The non-linear layer is used to **increase the nonlinearity** of the net
3. The **output** of the non-linear layer are the **neuron activations**

$\varphi(\cdot)$: Rectified Linear Unit (ReLU)

$$y = max(0, x)$$

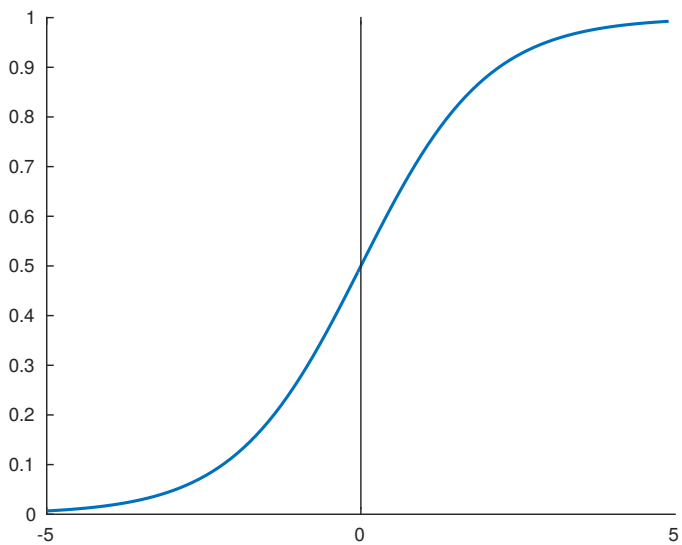
The Non-Linear Layer: Why does everybody use ReLU?

Current deep **architectures** all **rely on ReLU** for computing the non-linearity. This has several **practical reasons** and **implications**:

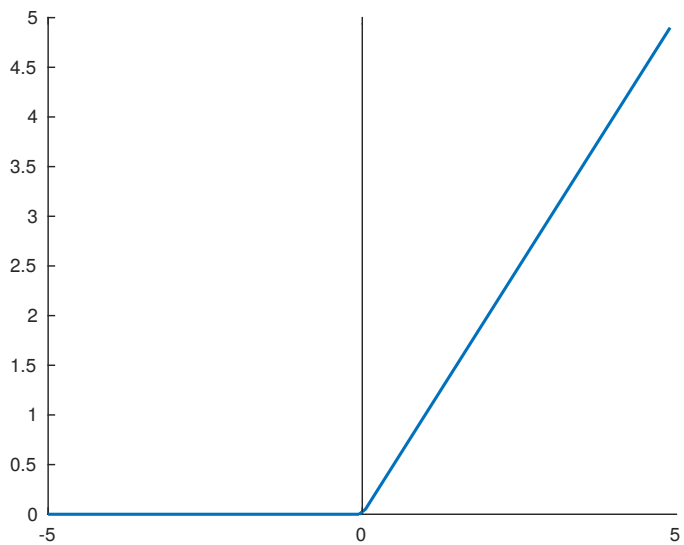
The Non-Linear Layer: Why does everybody use ReLU?

Current deep **architectures** all **rely on ReLU** for computing the non-linearity. This has several **practical reasons** and **implications**:

1. **Vanishing problem**: Sigmoid **saturate** and kill gradients



(a) Sigmoid

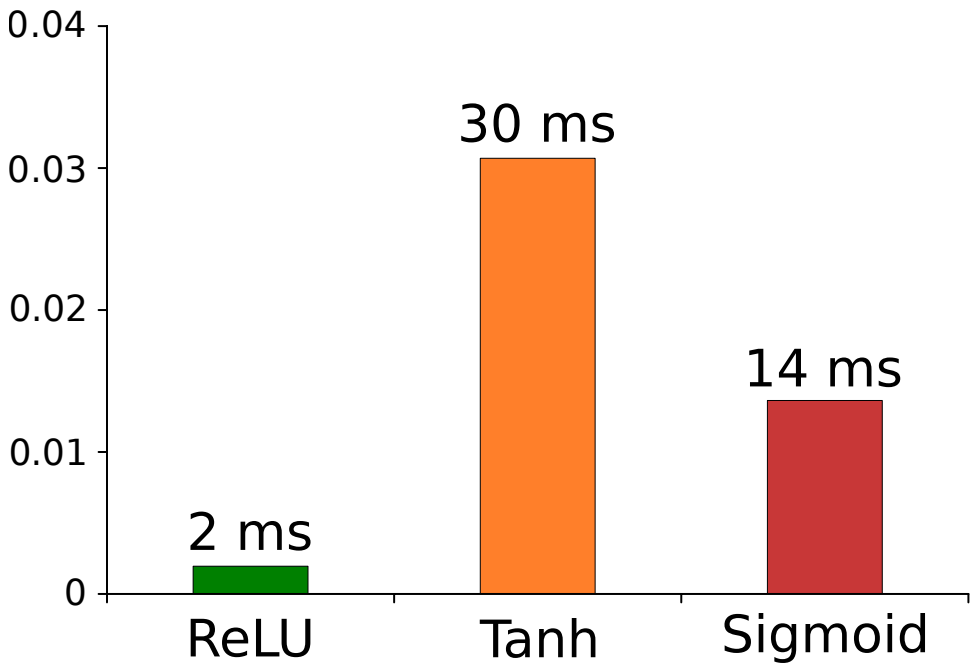


(b) ReLU

The Non-Linear Layer: Why does everybody use ReLU?

Current deep **architectures** all **rely on ReLU** for computing the non-linearity. This has several **practical reasons** and **implications**:

1. **Vanishing problem**: Sigmoid **saturate** and kill gradients
2. **ReLU** can be computed **much faster** than sigmoid or tanh



Computational Time for 1024*1024 neurons.

The Non-Linear Layer: Why does everybody use ReLU?

Current deep **architectures** all **rely on ReLU** for computing the non-linearity. This has several **practical reasons** and **implications**:

1. **Vanishing problem**: Sigmoid **saturate** and kill gradients
2. **ReLU** can be computed **much faster** than sigmoid or tanh
3. Faster **convergence** hence **less iterations** in training are **required**

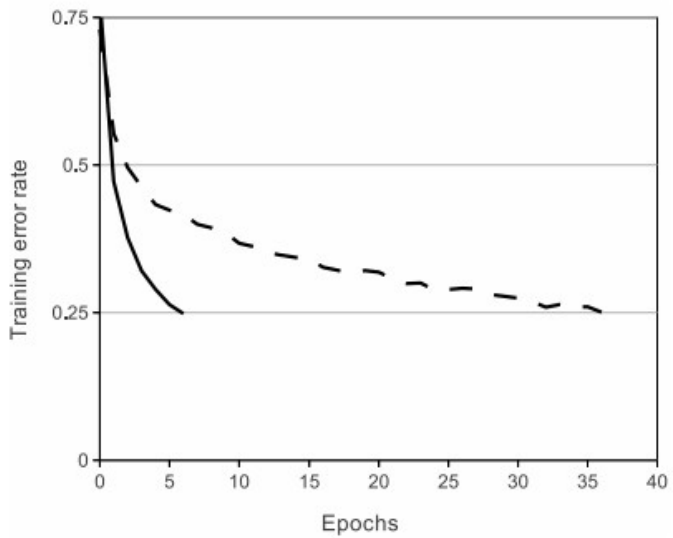


Figure : Krizhevsky et al.: ReLU (solid line) convergence compared to tanh (dashed line) in a 4-layer ConvNet on the CIFAR-10 dataset

The Non-Linear Layer: More about ReLUs

ReLUs are **not the holy grail** however. There are still some other **problems** that should be considered:

1. **Dying ReLU problem**: ReLUs outputs always 0, neurons can not learn

The Non-Linear Layer: More about ReLUs

ReLUs are **not the holy grail** however. There are still some other **problems** that should be considered:

1. **Dying ReLU problem**: ReLUs outputs always 0, neurons can not learn
2. **Leaky ReLUs** can help to recover: $y = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$

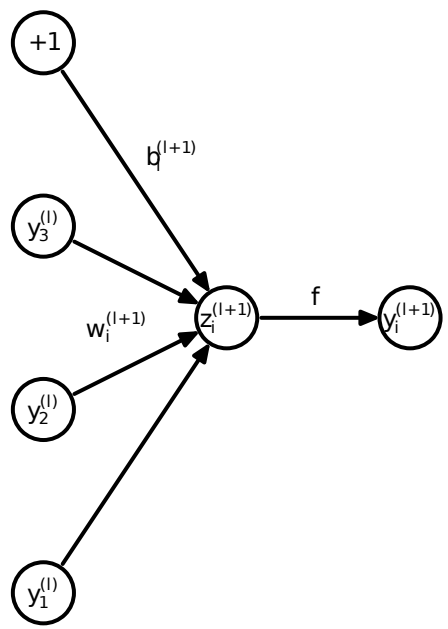
The Dropout Layer

The Dropout Layer

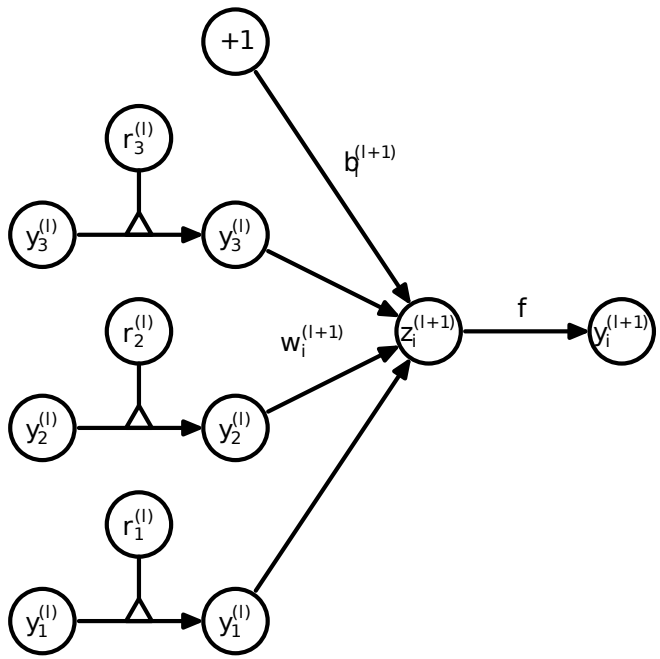
1. Due to the **huge number** of free **parameters** NNs tend to **over-fitting**

The Dropout Layer

1. Due to the **huge number** of free **parameters** NNs tend to **over-fitting**
2. Dropout provides a way of **preventing over-fitting** and combining many different NN-architectures efficiently



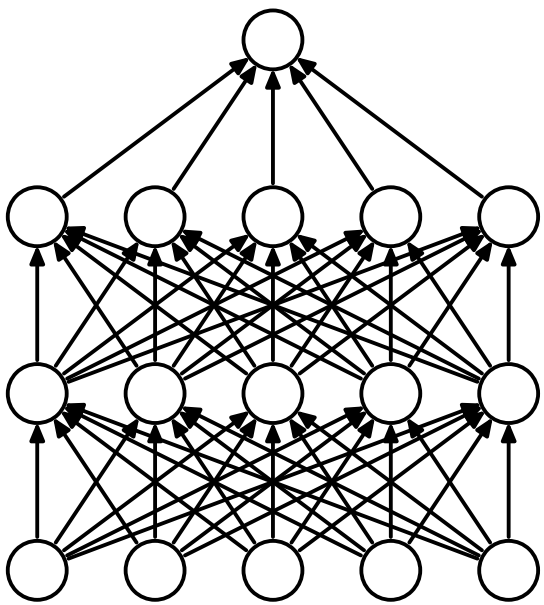
Standard Network



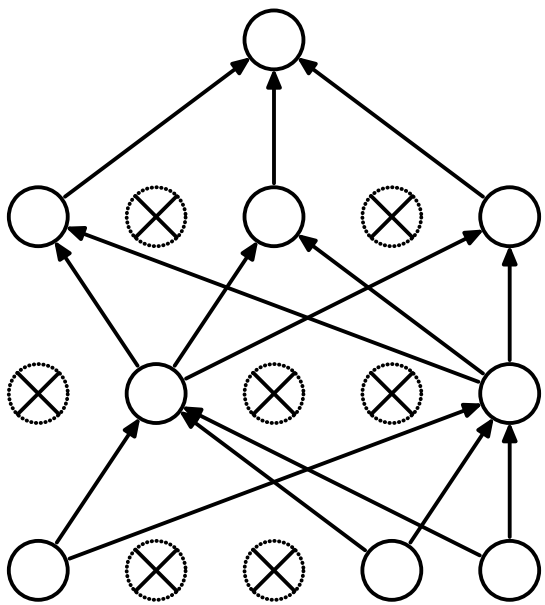
Dropout Network

The Dropout Layer

1. Due to the **huge number** of free **parameters** NNs tend to **over-fitting**
2. Dropout provides a way of **preventing over-fitting** and combining many different NN-architectures efficiently
3. Dropout performs regularization by **switching off neurons randomly**



Standard Neural Net



After applying dropout

The Dropout Layer

1. Due to the **huge number** of free **parameters** NNs tend to **over-fitting**
2. Dropout provides a way of **preventing over-fitting** and combining many different NN-architectures efficiently
3. Dropout performs regularization by **switching off neurons randomly**
4. A NN with n units can be seen as a **collection** of approx. 2^n **thinned neural networks**

The Dropout Layer

1. Due to the **huge number** of free **parameters** NNs tend to **over-fitting**
2. Dropout provides a way of **preventing over-fitting** and combining many different NN-architectures efficiently
3. Dropout performs regularization by **switching off neurons randomly**
4. A NN with n units can be seen as a **collection** of approx. 2^n **thinned neural networks**
5. During **test time** the **weights are scaled down** by multiplication with p

The Dropout Layer

This **approach** has several **benefits**:

1. **Increase sparsity** (few highly activated neurons for any data case)

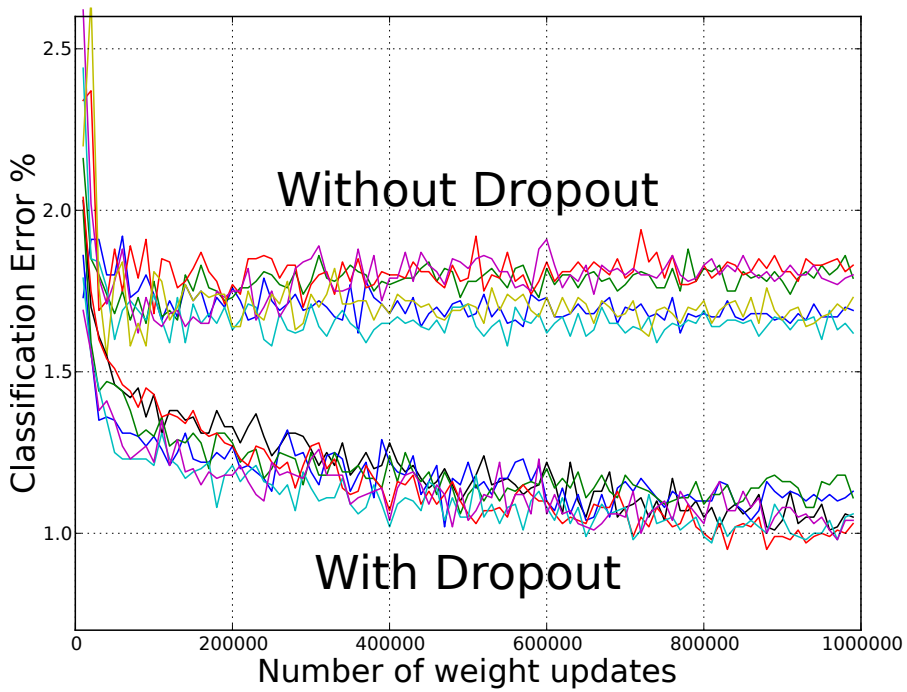


Figure : Srivatsava et al.: Dropout Layer

The Dropout Layer

This **approach** has several **benefits**:

1. **Increase sparsity** (few highly activated neurons for any data case)
2. **Prevents co-adaptation** of neurons

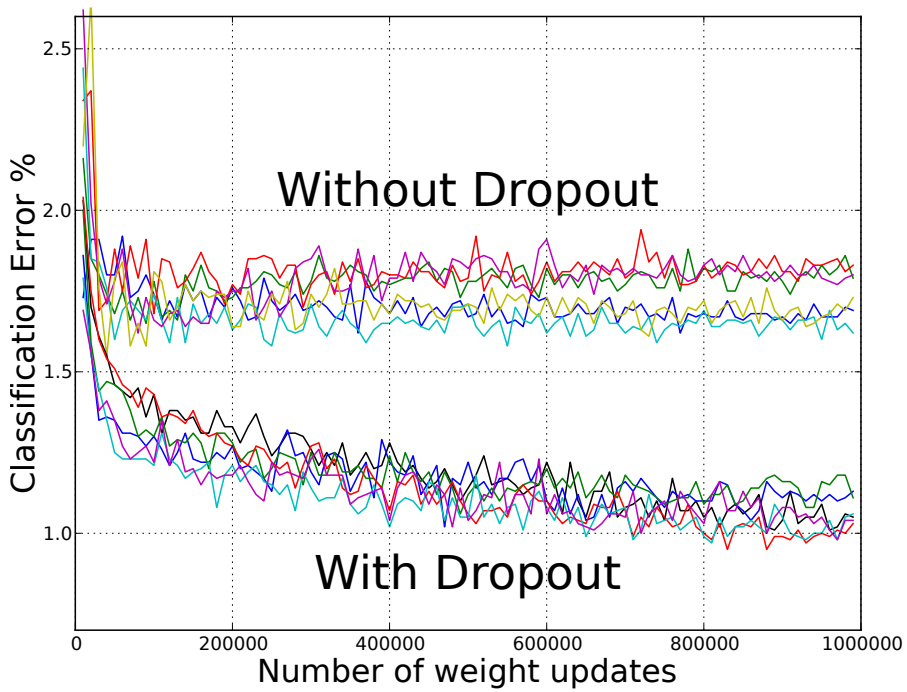


Figure : Srivatsava et al.: Dropout Layer

The Dropout Layer

This **approach** has several **benefits**:

1. **Increase sparsity** (few highly activated neurons for any data case)
2. **Prevents co-adaptation** of neurons
3. However: dropout has a **"sweet spot"**

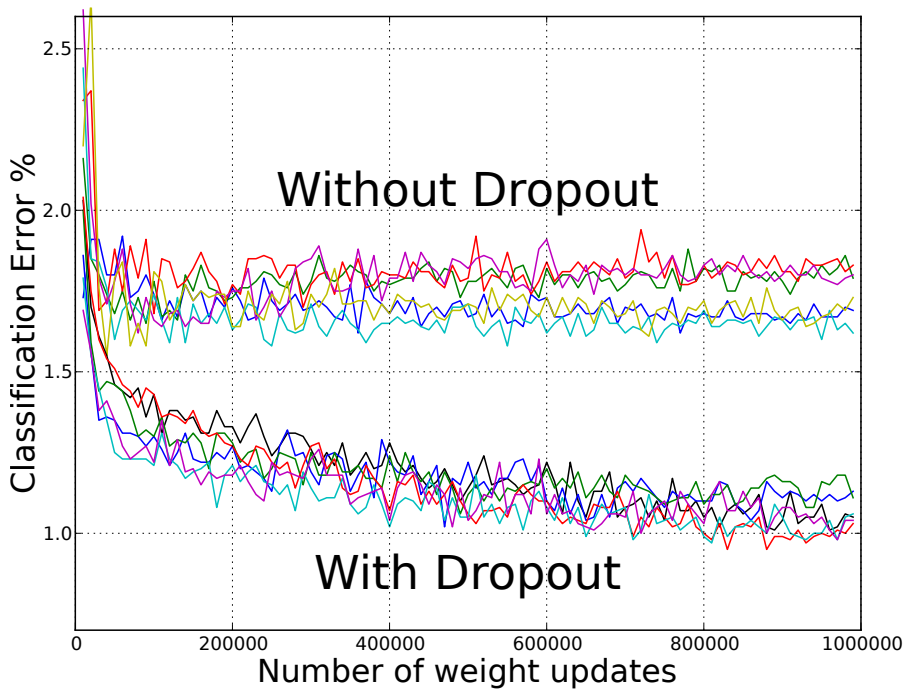


Figure : Srivatsava et al.: Dropout Layer

The Softmax Layer

The Softmax Layer

The softmax function maps a K-dimensional vector of real numbers to a K-dimensional vector between 0 and 1.

The Softmax Function:

$$y_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \qquad \text{for } j = 1, \ldots, K$$

The Softmax Layer

The softmax function maps a K-dimensional vector of real numbers to a K-dimensional vector between 0 and 1.

The Softmax Function:

$$y_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \qquad \text{for } j = 1, \ldots, K$$

1. The softmax layer acts as a **non-linear and normalization layer**

The Softmax Layer

The softmax function maps a K-dimensional vector of real numbers to a K-dimensional vector between 0 and 1.

The Softmax Function:

$$y_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \qquad \text{for } j = 1, \ldots, K$$

1. The softmax layer acts as a **non-linear and normalization layer**
2. **Prepares** output for the **loss layer**

The Softmax Layer

The softmax function maps a K-dimensional vector of real numbers to a K-dimensional vector between 0 and 1.

The Softmax Function:

$$y_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \qquad \text{for } j = 1, \ldots, K$$

1. The softmax layer acts as a **non-linear and normalization layer**
2. **Prepares** output for the **loss layer**
3. Output is usually **fed** into the **loss layer** (logistic, cross-entropy loss)

The Softmax Layer

The softmax function maps a K-dimensional vector of real numbers to a K-dimensional vector between 0 and 1.

The Softmax Function:

$$y_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \qquad \text{for } j = 1, \ldots, K$$

1. The softmax layer acts as a **non-linear and normalization layer**
2. **Prepares** output for the **loss layer**
3. Output is usually **fed** into the **loss layer** (logistic, cross-entropy loss)
4. This basically is a variant of **multinomial logistic regression**

The Softmax Layer

The softmax function maps a K-dimensional vector of real numbers to a K-dimensional vector between 0 and 1.

The Softmax Function:

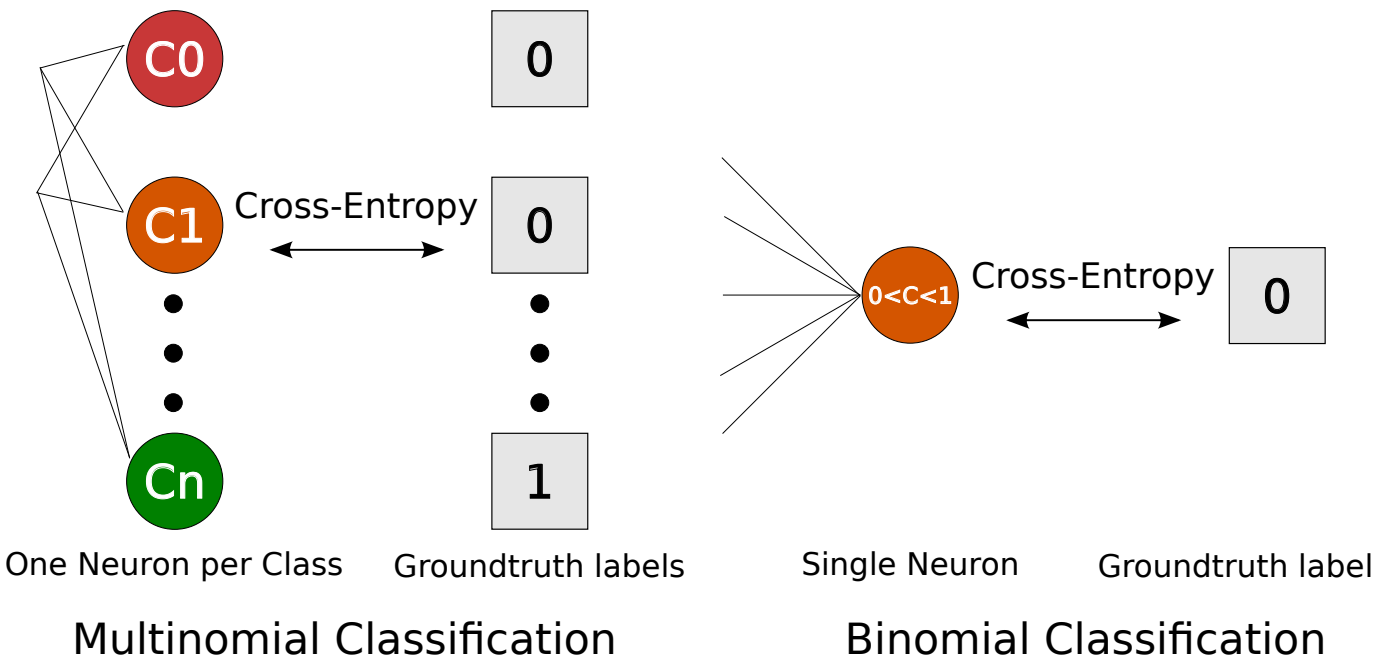
$$y_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \qquad \text{for } j = 1, \ldots, K$$

1. The softmax layer acts as a **non-linear and normalization layer**
2. **Prepares** output for the **loss layer**
3. Output is usually **fed** into the **loss layer** (logistic, cross-entropy loss)
4. This basically is a variant of **multinomial logistic regression**
5. Each output can be interpreted as **posterior probability of a class** given the input

The Loss Layer

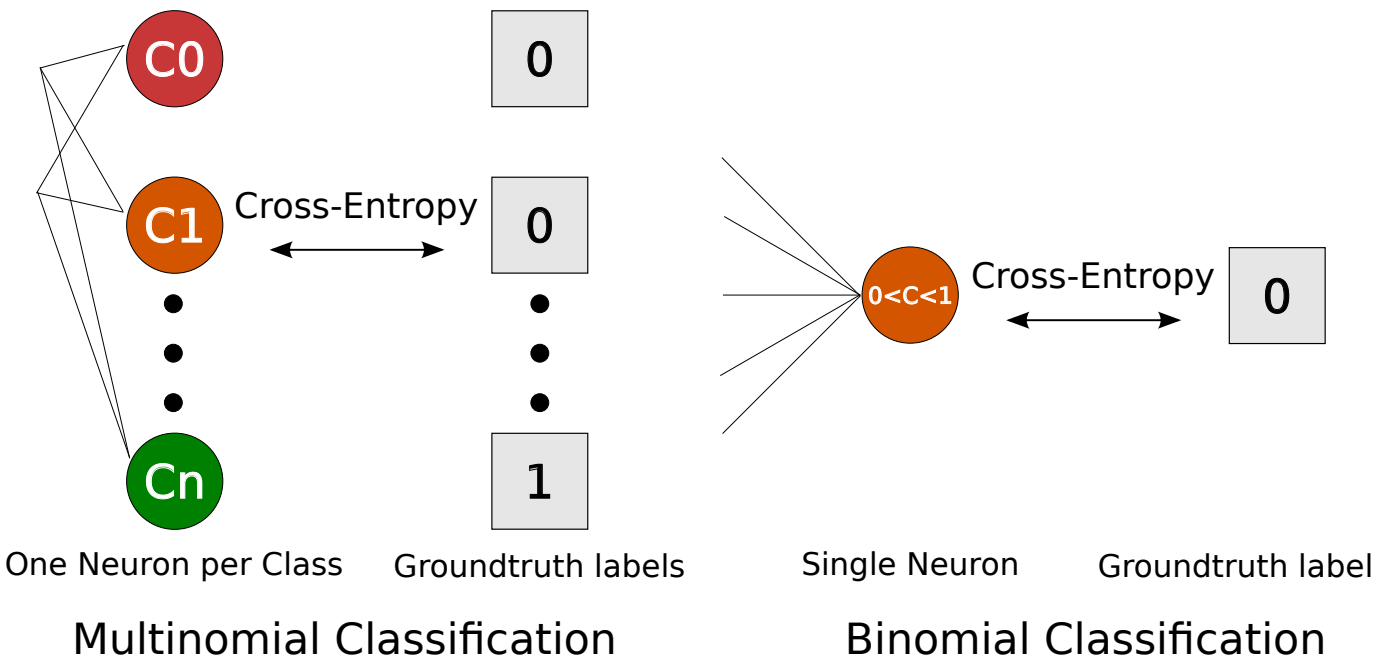
The Loss Layer: Output Encoding

1. A **binomial problem** can be solved using a network with a **single logistic output neuron**



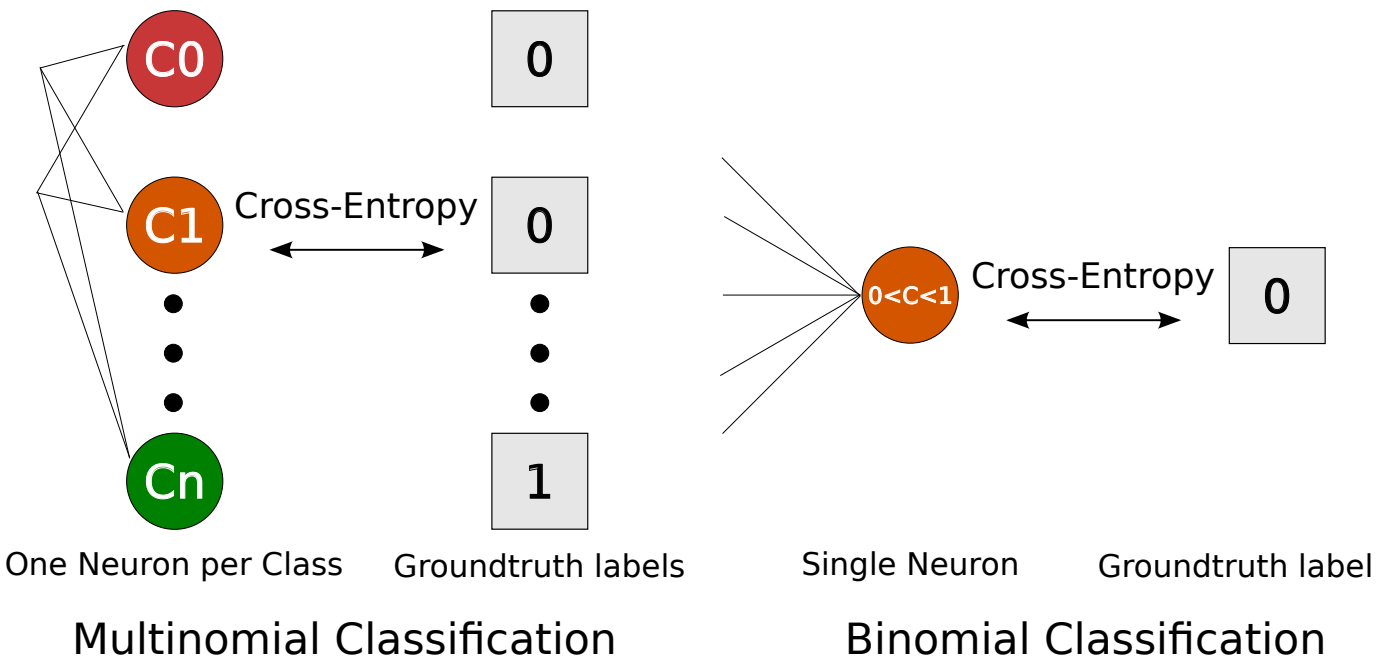
The Loss Layer: Output Encoding

1. A **binomial problem** can be solved using a network with a **single logistic output neuron**
2. **Multinomial problems** are modeled using a **set of neurons based** on the softmax activation function



The Loss Layer: Output Encoding

1. A **binomial problem** can be solved using a network with a **single logistic output neuron**
2. **Multinomial problems** are modeled using a **set of neurons based** on the softmax activation function
3. In both cases the **cross-entropy loss** can be used to compute the **network error**



The Loss Layer: Computing the Loss

The loss layer is where the **groundtruth** (l) is used to compute **how well the network performed** in predicting the class labels (z) of the input for a C-class problem.

The Cross-Entropy Loss:

$$y = -\sum_{c=1}^{C} l_c \, log(z_c)$$

The Loss Layer: Computing the Loss

The loss layer is where the **groundtruth** (l) is used to compute **how well the network performed** in predicting the class labels (z) of the input for a C-class problem.

The Cross-Entropy Loss:

$$y = -\sum_{c=1}^{C} l_c \, log(z_c)$$

1. **Final layer** in a ConvNet

The Loss Layer: Computing the Loss

The loss layer is where the **groundtruth** (l) is used to compute **how well the network performed** in predicting the class labels (z) of the input for a C-class problem.

The Cross-Entropy Loss:

$$y = -\sum_{c=1}^{C} l_c \; log(z_c)$$

1. **Final layer** in a ConvNet
2. Also known as **log-loss**, **logistic-loss** and **cross-entropy loss**

The Loss Layer: Computing the Loss

The loss layer is where the **groundtruth** (l) is used to compute **how well the network performed** in predicting the class labels (z) of the input for a C-class problem.

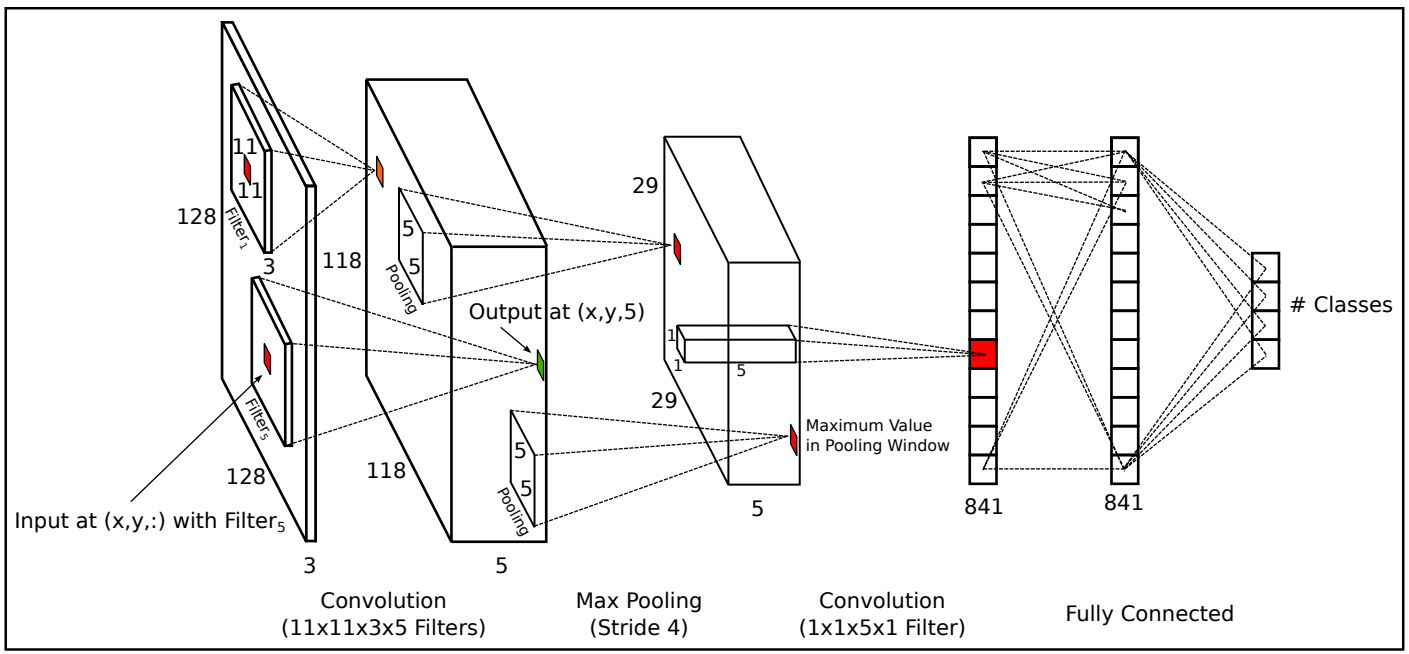
The Cross-Entropy Loss:

$$y = -\sum_{c=1}^{C} l_c \, log(z_c)$$

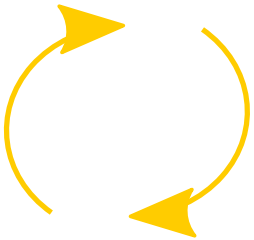
1. **Final layer** in a ConvNet
2. Also known as **log-loss**, **logistic-loss** and **cross-entropy loss**
3. A **large loss is bad**, a **small loss is good**

Backward Pass
CNN Building Blocks: Layers

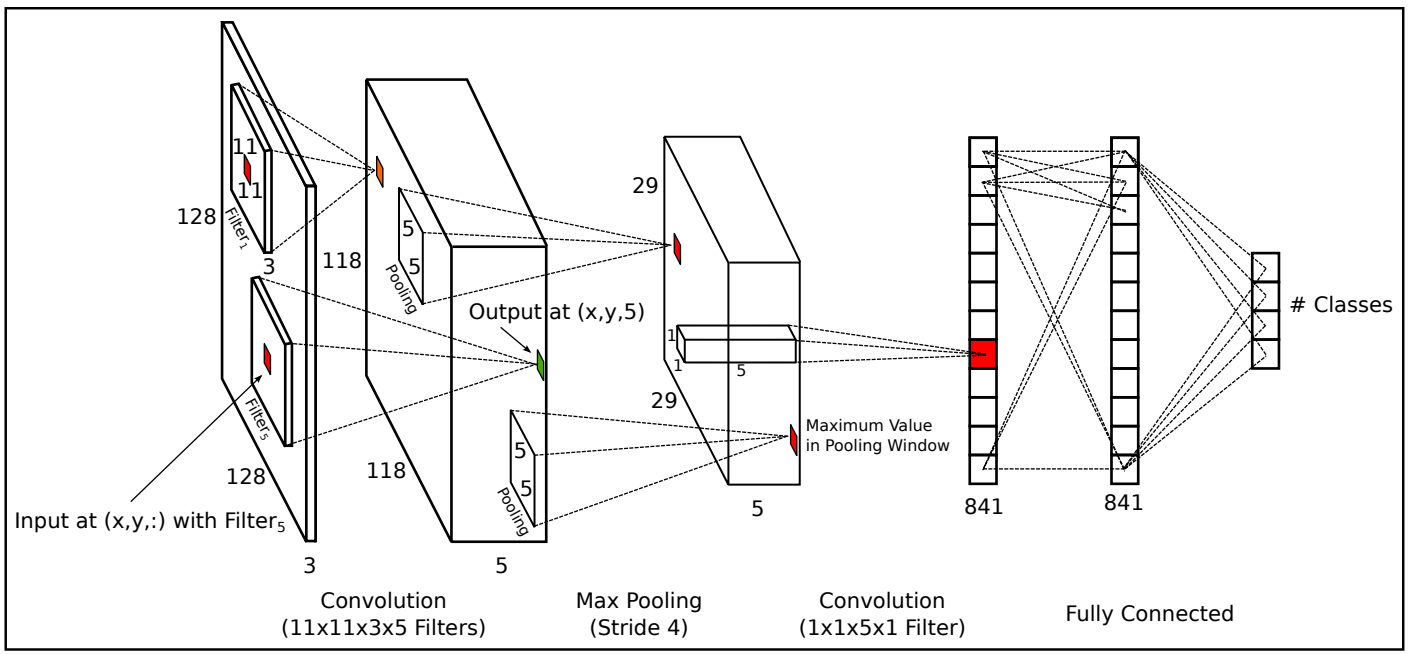
Backward Pass: Compute Derivative of the Loss w.r.t. each Parameter



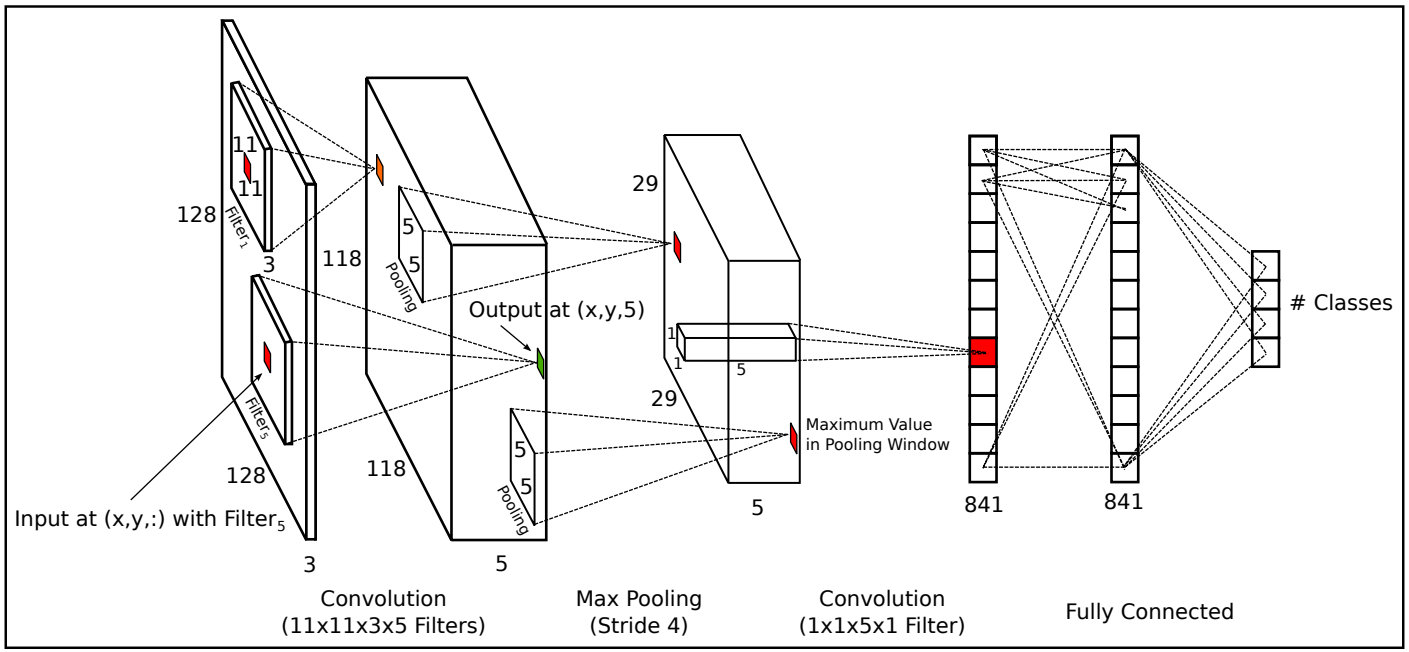
CNN Building Blocks: Layers - Backward Pass



Update Weights to minimize Loss.



Forward Pass: Compute Neuron Activations



Learning: Stochastic Gradient Descent

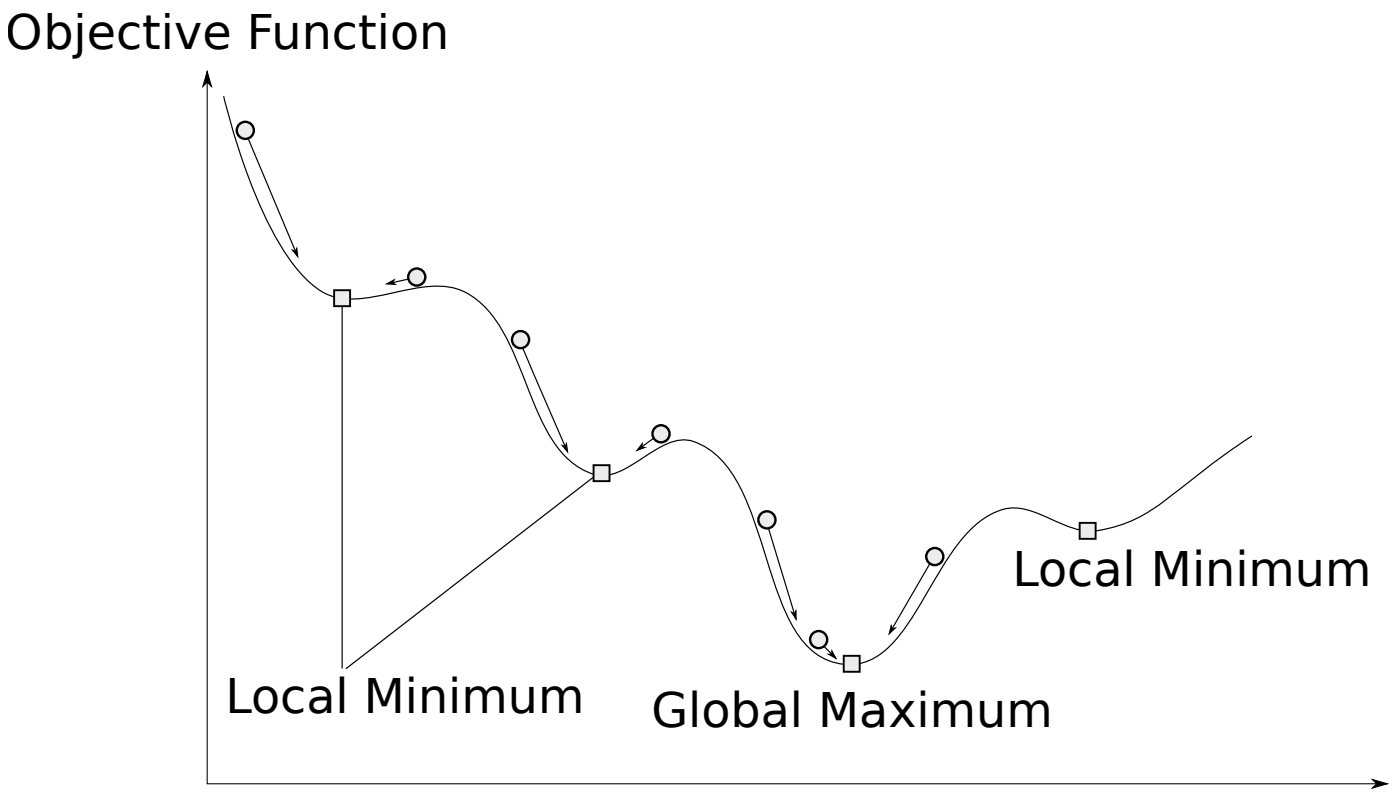
Stochastic Gradient Descent (SGD) is used to Optimize the Loss:

$$w_{i+1} \leftarrow w_i - \eta_i \frac{\partial L}{\partial w_i} + \alpha \Delta w_i - \eta_i \lambda w_i$$

Learning: Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is used to Optimize the Loss:

$$w_{i+1} \leftarrow w_i - \eta_i \frac{\partial L}{\partial w_i} + \alpha \Delta w_i - \eta_i \lambda w_i$$



Learning: Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is used to Optimize the Loss:

$$w_{i+1} \leftarrow w_i - \eta_i \frac{\partial L}{\partial w_i} + \alpha \Delta w_i - \eta_i \lambda w_i$$

1. **Momentum** is used to **"jump"** over small **local minima** (also speeds up convergence)

Learning: Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is used to Optimize the Loss:

$$w_{i+1} \leftarrow w_i - \eta_i \frac{\partial L}{\partial w_i} + \alpha \Delta w_i - \eta_i \lambda w_i$$

1. **Momentum** is used to **"jump"** over small **local minima** (also speeds up convergence)
2. A **weight decay** term is sometimes used for **regularization**

Learning: Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is used to Optimize the Loss:

$$w_{i+1} \leftarrow w_i - \eta_i \frac{\partial L}{\partial w_i} + \alpha \Delta w_i - \eta_i \lambda w_i$$

1. **Momentum** is used to **"jump"** over small **local minima** (also speeds up convergence)
2. A **weight decay** term is sometimes used for **regularization**
3. SGD is based on **batches of randomly drawn inputs** (improves stability of the gradient)

Learning: Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is used to Optimize the Loss:

$$w_{i+1} \leftarrow w_i - \eta_i \frac{\partial L}{\partial w_i} + \alpha \Delta w_i - \eta_i \lambda w_i$$

1. **Momentum** is used to **"jump"** over small **local minima** (also speeds up convergence)
2. A **weight decay** term is sometimes used for **regularization**
3. SGD is based on **batches of randomly drawn inputs** (improves stability of the gradient)
4. **AdaGrad** is an enhanced variation which determines the **learning rate per parameters** based on previous updates

Learning: Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is used to Optimize the Loss:

$$w_{i+1} \leftarrow w_i - \eta_i \frac{\partial L}{\partial w_i} + \alpha \Delta w_i - \eta_i \lambda w_i$$

1. **Momentum** is used to **"jump"** over small **local minima** (also speeds up convergence)
2. A **weight decay** term is sometimes used for **regularization**
3. SGD is based on **batches of randomly drawn inputs** (improves stability of the gradient)
4. **AdaGrad** is an enhanced variation which determines the **learning rate per parameters** based on previous updates

To learn the weights for a problem we have to compute the derivative of the loss function w.r.t. each parameter in the NN.

Backward Pass: Backpropagation

Backpropagation Sketch

A feed-forward network can be seen as a **composition of functions**:

$$f(x) = f_L(\dots f_2(f_1(x; w_1); w_2)\dots); w_L);$$

Using the **chain-rule** this allows to **compute the derivatives layer by layer**.

Backpropagation Sketch

A feed-forward network can be seen as a **composition of functions**:

$$f(x) = f_L(\dots f_2(f_1(x; w_1); w_2)\dots); w_L);$$

Using the **chain-rule** this allows to **compute the derivatives layer by layer**.

Total Derivative

The **total or full derivative** of a function $y = f(t, u_1(t), \dots, u_m(t))$ of **several variables that depend on each other** is computed as:

$$\frac{\partial y}{\partial t} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial u_1}\frac{\partial u_1}{\partial t} + \dots + \frac{\partial f}{\partial u_m}\frac{\partial u_m}{\partial t}$$

Backward Pass: The Loss Layer

Loss Layer

$$\begin{aligned} \frac{\partial y}{\partial z_i} &= \frac{\partial}{\partial zi}\left(-\sum_{c=1}^{C} l_c \; log(z_c)\right) \\ &= -\frac{l_i}{z_i} \end{aligned}$$

Loss Layer

$$\begin{aligned} \frac{\partial y}{\partial z_i} &= \frac{\partial}{\partial zi}\left(-\sum_{c=1}^{C} l_c \; log(z_c)\right) \\ &= -\frac{l_i}{z_i} \end{aligned}$$

1. Most implementations assume that the class probabilities are $\in \{0, 1\}$ and only **compute the gradient for the correct class**

Loss Layer

$$\begin{aligned} \frac{\partial y}{\partial z_i} &= \frac{\partial}{\partial zi}\left(-\sum_{c=1}^{C} l_c \; log(z_c)\right) \\ &= -\frac{l_i}{z_i} \end{aligned}$$

1. Most implementations assume that the class probabilities are $\in \{0, 1\}$ and only **compute the gradient for the correct class**
2. The **net only adapts parameters** (learns) that **contributed to the activation** of the corresponding neuron

Backward Pass: The Softmax Layer

Softmax Layer: Backward Pass

Softmax Layer

$$y_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \qquad \text{for } j = 1, \ldots, K$$

The **total derivative** of output y_i w.r.t. a parameter x_j is:

$$\frac{\partial y_i}{\partial x_j} = \frac{\partial y_i}{\partial x_1} + \frac{\partial y_i}{\partial x_2} + \cdots + \frac{\partial y_i}{\partial x_K} \tag{1}$$

Softmax Layer: Backward Pass

Case $i = j$:

$$\frac{\partial y_i}{\partial x_i} = \frac{e^{x_i}\left(\sum_j e^{x_j}\right) - e^{x_i}e^{x_i}}{\left(\sum_j e^{x_j}\right)^2} = \frac{e^{x_i}\sum_j e^{x_j}}{\left(\sum_j e^{x_j}\right)^2} - \frac{e^{2x_i}}{\left(\sum_j e^{x_j}\right)^2}$$

$$= \underbrace{\frac{e^{x_i}}{\sum_j e^{x_j}}}_{=y_i} - \left(\underbrace{\frac{e^{x_i}}{\sum_j e^{x_j}}}_{=y_i}\underbrace{\frac{e^{x_i}}{\sum_j e^{x_j}}}_{=y_i}\right)$$

$$= y_i\left(1 - y_i\right)$$

Softmax Layer: Backward Pass

Case $i \neq j$:

$$\frac{\partial y_i}{\partial x_j} = \frac{-e^{x_j} e^{x_i}}{\left(\sum_j e^{x_j}\right)^2} = -\underbrace{\frac{e^{x_j}}{\sum_j e^{x_j}}}_{=y_j} \underbrace{\frac{e^{x_i}}{\sum_j e^{x_j}}}_{=y_i} = -y_j \, y_i$$

The derivatives of the softmax layer are then **multiplied** with the **derivatives of the previous layer**. Note how the output of the forward pass can be re-used.

Backward Pass: The Dropout Layer

Dropout Layer: Backward Pass

The dropout operation $y = x \circ d$ is a **point-wise multiplication** of n independent random variables D following a Bernoulli distribution with the activations produced by n neurons x.

Dropout Layer

$$\frac{\partial y}{\partial x_i} = \begin{cases} 1, & \text{if } d_n = 1 \\ 0, & \text{otherwise} \end{cases}$$

Backward Pass: The Non-Linear Layer

Non-Linear Layer: Backward Pass (ReLU)

Note that the **ReLU function** $y = max(0, x)$ is not **differentiable in 0**. But it has **subdifferential** $\partial f(0) \in [0, 1]$. Therefore any value in $[0, 1]$ can be used as **subderivative** in SGD (which is then actually sub-gradient stochastic descent).

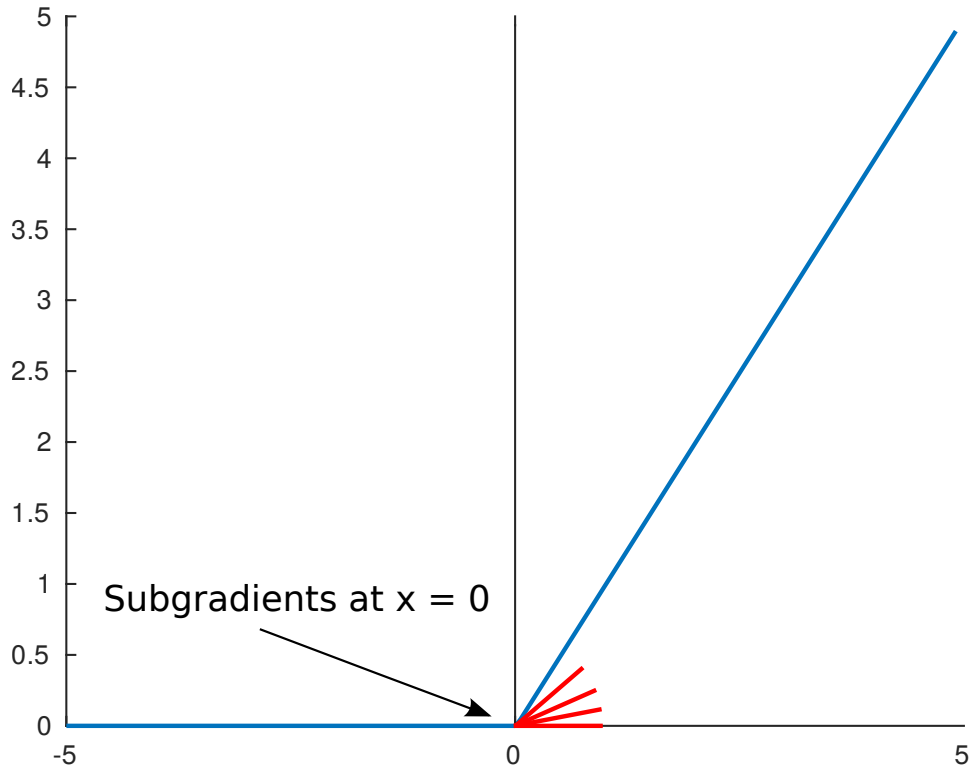


Figure : Subgradients of the ReLU function at x = 0

Non-Linear Layer: Backward Pass (ReLU)

Note that the **ReLU function** $y = max(0, x)$ is not **differentiable in 0**. But it has **subdifferential** $\partial f(0) \in [0, 1]$. Therefore any value in $[0, 1]$ can be used as **subderivative** in SGD (which is then actually sub-gradient stochastic descent).

ReLU Derivative

$$\frac{\partial y}{\partial x_i} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

Non-Linear Layer: Backward Pass (ReLU)

Note that the **ReLU function** $y = max(0, x)$ is not **differentiable in 0**. But it has **subdifferential** $\partial f(0) \in [0, 1]$. Therefore any value in $[0, 1]$ can be used as **subderivative** in SGD (which is then actually sub-gradient stochastic descent).

ReLU Derivative

$$\frac{\partial y}{\partial x_i} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

1. Most implementations choose the **subgradient** at $x = 0$ to be 0

Non-Linear Layer: Backward Pass (ReLU)

Note that the **ReLU function** $y = max(0, x)$ is not **differentiable in 0**. But it has **subdifferential** $\partial f(0) \in [0, 1]$. Therefore any value in $[0, 1]$ can be used as **subderivative** in SGD (which is then actually sub-gradient stochastic descent).

ReLU Derivative

$$\frac{\partial y}{\partial x_i} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

1. Most implementations choose the **subgradient** at $x = 0$ to be 0
2. Units with **zero activation do not** actively **contribute** to the decision in **upper layers**

Non-Linear Layer: Backward Pass (ReLU)

Note that the **ReLU function** $y = max(0, x)$ is not **differentiable in 0**. But it has **subdifferential** $\partial f(0) \in [0, 1]$. Therefore any value in $[0, 1]$ can be used as **subderivative** in SGD (which is then actually sub-gradient stochastic descent).

ReLU Derivative

$$\frac{\partial y}{\partial x_i} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

1. Most implementations choose the **subgradient** at $x = 0$ to be 0
2. Units with **zero activation do not** actively **contribute** to the decision in **upper layers**
3. **Non activated units** do **not contribute** to weight updates in **lower layers**

Backward Pass: The Pooling Layer

Pooling Layer: Backward Pass

The derivative of the max-pooling layer $y = max\{x_1, \ldots, x_n\}$ is again computed using **sub-gradients**.

Max-Pooling

$$\frac{\partial y}{\partial x_i} = \begin{cases} 1, & \text{if } x_i = max\{x_1, \ldots, x_n\} \\ 0, & \text{otherwise} \end{cases}$$

Pooling Layer: Backward Pass

The derivative of the max-pooling layer $y = max\{x_1, \ldots, x_n\}$ is again computed using **sub-gradients**.

Max-Pooling

$$\frac{\partial y}{\partial x_i} = \begin{cases} 1, & \text{if } x_i = max\{x_1, \ldots, x_n\} \\ 0, & \text{otherwise} \end{cases}$$

1. Only the **"winning" unit contributes** to the decision in upper layers

Pooling Layer: Backward Pass

The derivative of the max-pooling layer $y = max\{x_1, \ldots, x_n\}$ is again computed using **sub-gradients**.

Max-Pooling

$$\frac{\partial y}{\partial x_i} = \begin{cases} 1, & \text{if } x_i = max\{x_1, \ldots, x_n\} \\ 0, & \text{otherwise} \end{cases}$$

1. Only the **"winning" unit contributes** to the decision in upper layers
2. Other **units do not contribute** to weight-updates in lower layers

Backward Pass: The Convolution Layer

Convolution Layer: Backward Pass

The **convolution layer** is the only layer where **learning happens**. There are **no parameter updates for the other layers**. Consequently, the backward pass in the convolution layer requires a bit more work:

Convolution Layer: Backward Pass

The **convolution layer** is the only layer where **learning happens**. There are **no parameter updates for the other layers**. Consequently, the backward pass in the convolution layer requires a bit more work:

1. Compute the **derivatives w.r.t. the inputs**

Convolution Layer: Backward Pass

The **convolution layer** is the only layer where **learning happens**. There are **no parameter updates for the other layers**. Consequently, the backward pass in the convolution layer requires a bit more work:

1. Compute the **derivatives w.r.t. the inputs**
   - Required to compute **derivatives of lower layers**

Convolution Layer: Backward Pass

The **convolution layer** is the only layer where **learning happens**. There are **no parameter updates for the other layers**. Consequently, the backward pass in the convolution layer requires a bit more work:

1. Compute the **derivatives w.r.t. the inputs**
   - Required to compute **derivatives of lower layers**
2. Compute the **derivatives w.r.t. the weights** (kernel maps)

Convolution Layer: Backward Pass

The **convolution layer** is the only layer where **learning happens**. There are **no parameter updates for the other layers**. Consequently, the backward pass in the convolution layer requires a bit more work:

1. Compute the **derivatives w.r.t. the inputs**
   - Required to compute **derivatives of lower layers**
2. Compute the **derivatives w.r.t. the weights** (kernel maps)
   - Required to **learn filters**

Convolution Layer: Backward Pass

The **convolution layer** is the only layer where **learning happens**. There are **no parameter updates for the other layers**. Consequently, the backward pass in the convolution layer requires a bit more work:

1. Compute the **derivatives w.r.t. the inputs**
   - Required to compute **derivatives of lower layers**
2. Compute the **derivatives w.r.t. the weights** (kernel maps)
   - Required to **learn filters**

And it should be fast!

Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

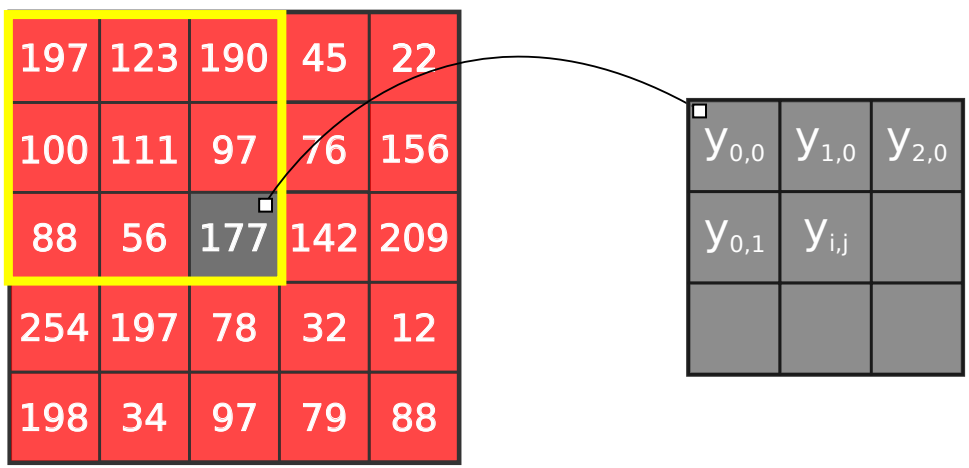
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



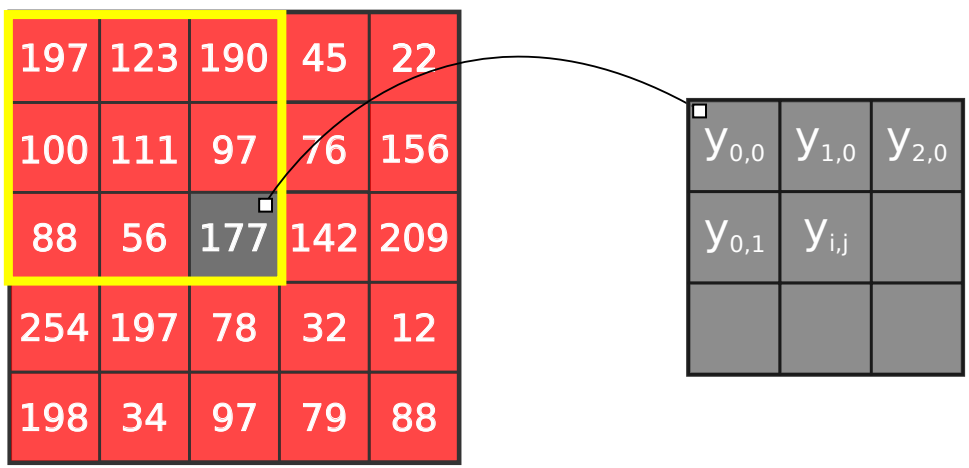
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



$w_{2,2} \frac{\partial z}{\partial y_{0,0}}$

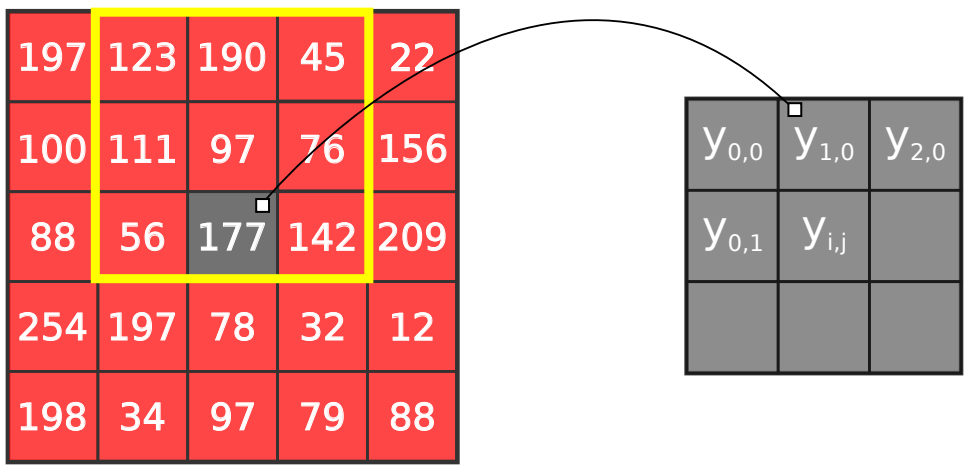
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



$w_{2,2} \frac{\partial z}{\partial y_{0,0}} + w_{1,2} \frac{\partial z}{\partial y_{1,0}}$

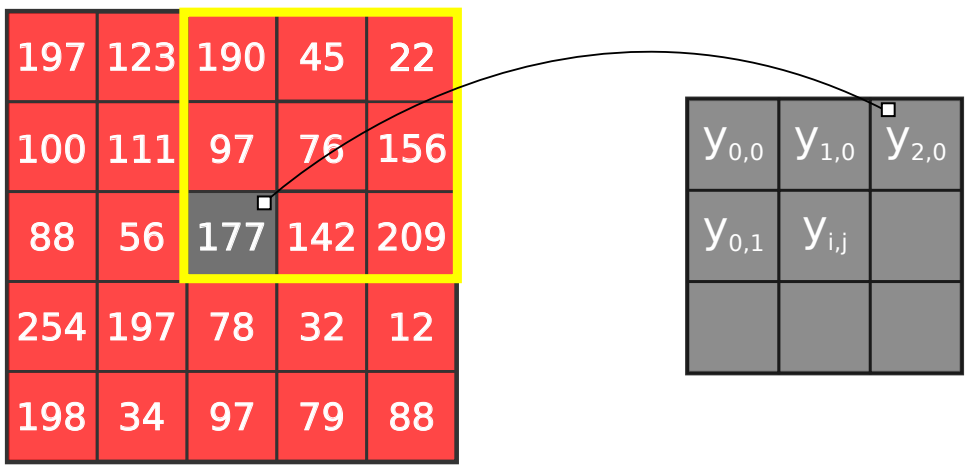
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



$w_{2,2} \frac{\partial z}{\partial y_{0,0}} + w_{1,2} \frac{\partial z}{\partial y_{1,0}} + w_{0,2} \frac{\partial z}{\partial y_{2,0}}$

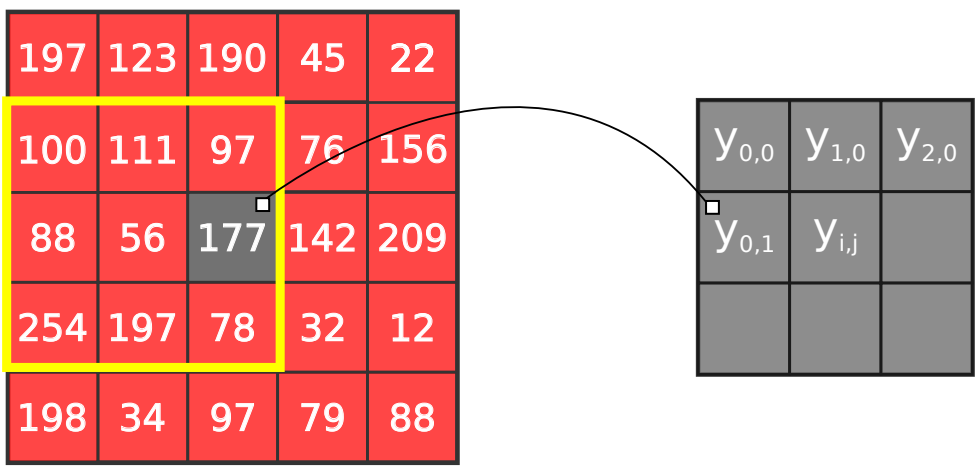
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



$w_{2,2} \frac{\partial z}{\partial y_{0,0}} + w_{1,2} \frac{\partial z}{\partial y_{1,0}} + w_{0,2} \frac{\partial z}{\partial y_{2,0}} + w_{2,1} \frac{\partial z}{\partial y_{0,1}}$

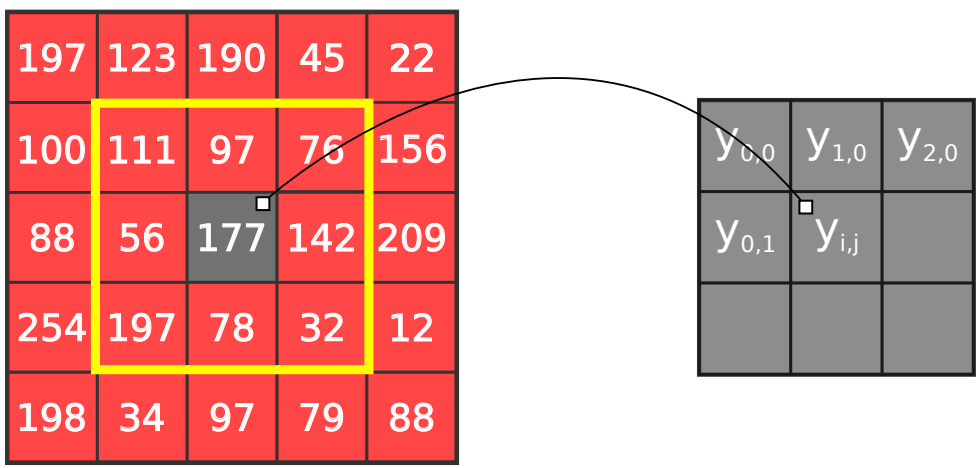
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



$$w_{2,2}\frac{\partial z}{\partial y_{0,0}} + w_{1,2}\frac{\partial z}{\partial y_{1,0}} + w_{0,2}\frac{\partial z}{\partial y_{2,0}} + w_{2,1}\frac{\partial z}{\partial y_{0,1}} + w_{1,1}\frac{\partial z}{\partial y_{1,1}}$$

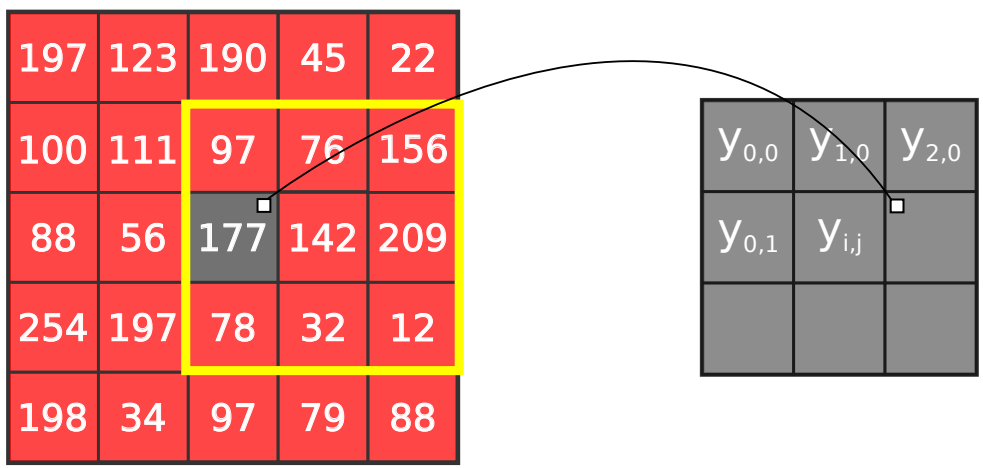
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



$$w_{2,2}\frac{\partial z}{\partial y_{0,0}} + w_{1,2}\frac{\partial z}{\partial y_{1,0}} + w_{0,2}\frac{\partial z}{\partial y_{2,0}} + w_{2,1}\frac{\partial z}{\partial y_{0,1}} + w_{1,1}\frac{\partial z}{\partial y_{1,1}} + w_{0,1}\frac{\partial z}{\partial y_{2,1}}$$

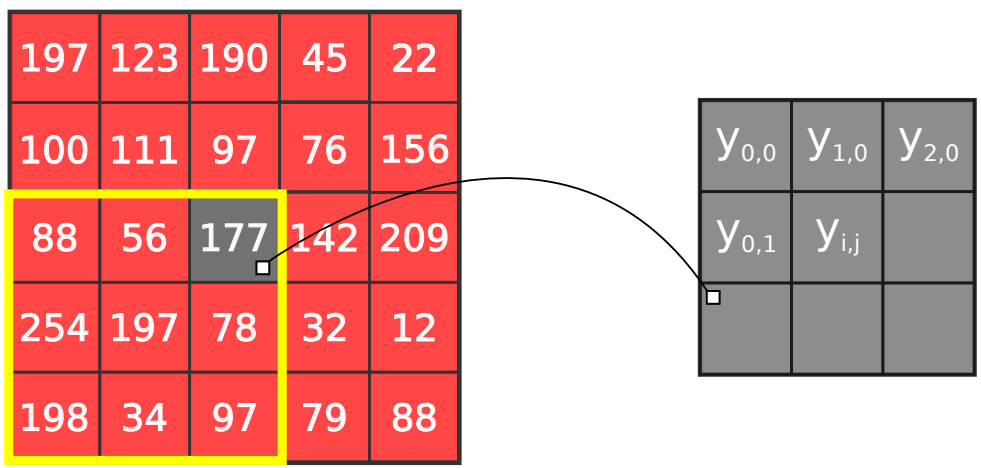
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



$$w_{2,2}\frac{\partial z}{\partial y_{0,0}} + w_{1,2}\frac{\partial z}{\partial y_{1,0}} + w_{0,2}\frac{\partial z}{\partial y_{2,0}} + w_{2,1}\frac{\partial z}{\partial y_{0,1}} + w_{1,1}\frac{\partial z}{\partial y_{1,1}} + w_{0,1}\frac{\partial z}{\partial y_{2,1}} + w_{2,0}\frac{\partial z}{\partial y_{0,2}}$$

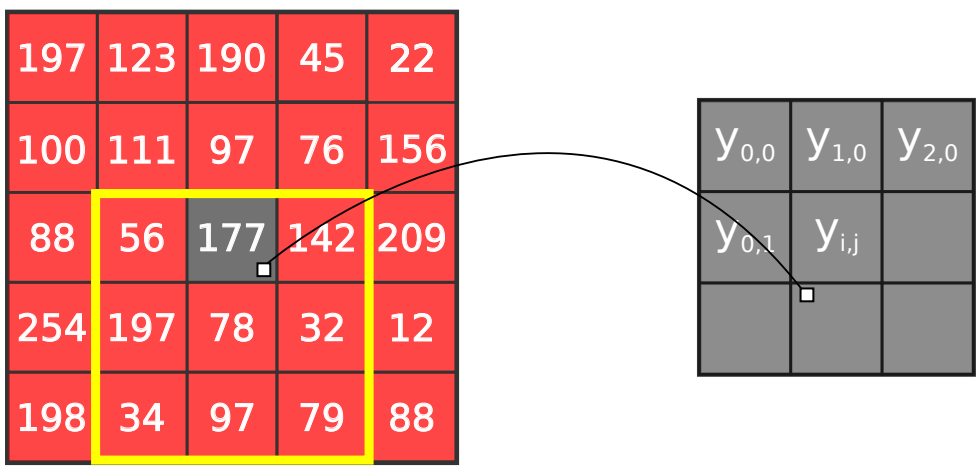
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



$$w_{2,2}\frac{\partial z}{\partial y_{0,0}} + w_{1,2}\frac{\partial z}{\partial y_{1,0}} + w_{0,2}\frac{\partial z}{\partial y_{2,0}} + w_{2,1}\frac{\partial z}{\partial y_{0,1}} + w_{1,1}\frac{\partial z}{\partial y_{1,1}} + w_{0,1}\frac{\partial z}{\partial y_{2,1}} + w_{2,0}\frac{\partial z}{\partial y_{0,2}} + w_{1,0}\frac{\partial z}{\partial y_{1,2}}$$

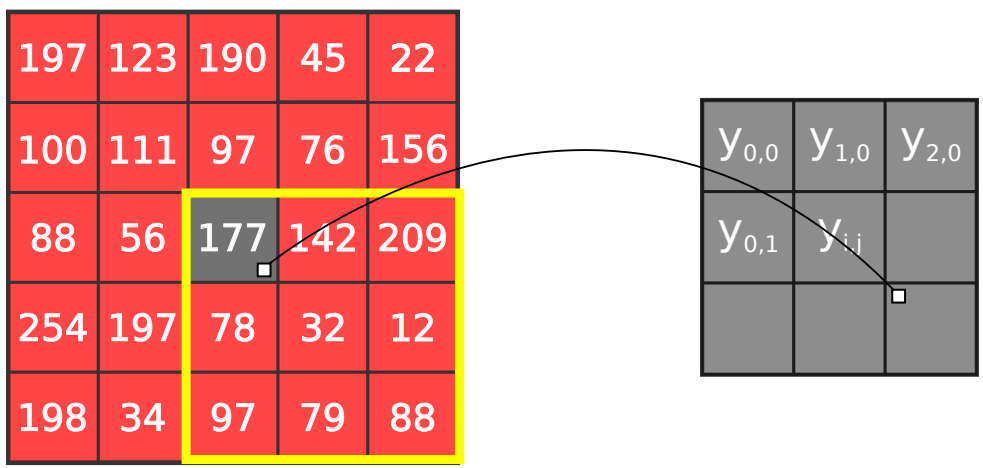
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



$$w_{2,2}\frac{\partial z}{\partial y_{0,0}} + w_{1,2}\frac{\partial z}{\partial y_{1,0}} + w_{0,2}\frac{\partial z}{\partial y_{2,0}} + w_{2,1}\frac{\partial z}{\partial y_{0,1}} + w_{1,1}\frac{\partial z}{\partial y_{1,1}} + w_{0,1}\frac{\partial z}{\partial y_{2,1}} + w_{2,0}\frac{\partial z}{\partial y_{0,2}} + w_{1,0}\frac{\partial z}{\partial y_{1,2}} + w_{0,0}\frac{\partial z}{\partial y_{2,2}}$$

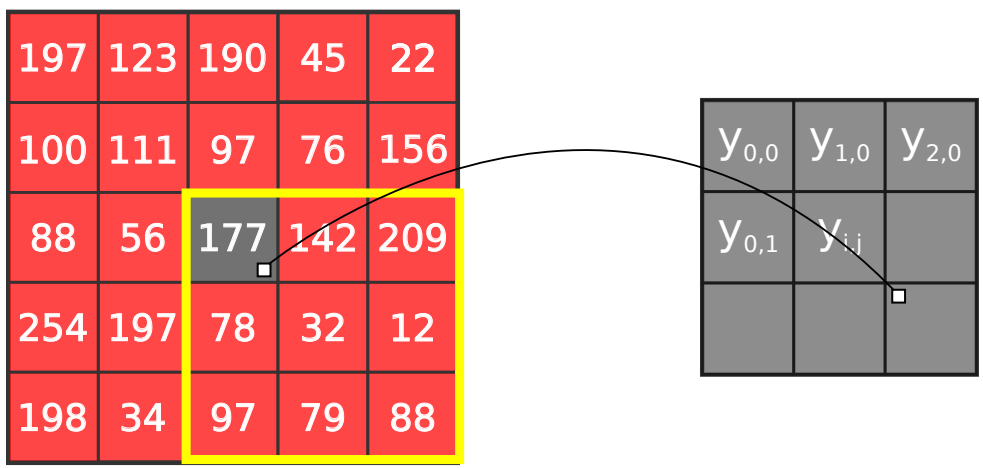
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

Again: A single input **contributes to multiple outputs**!



$w_{2,2} \frac{\partial z}{\partial y_{0,0}} + w_{1,2} \frac{\partial z}{\partial y_{1,0}} + w_{0,2} \frac{\partial z}{\partial y_{2,0}} + w_{2,1} \frac{\partial z}{\partial y_{0,1}} + w_{1,1} \frac{\partial z}{\partial y_{1,1}} + w_{0,1} \frac{\partial z}{\partial y_{2,1}} + w_{2,0} \frac{\partial z}{\partial y_{0,2}} + w_{1,0} \frac{\partial z}{\partial y_{1,2}} + w_{0,0} \frac{\partial z}{\partial y_{2,2}}$
$= \sum_i \sum_j w_{ij} \frac{\partial z}{\partial y_{n-i,m-j}}$.

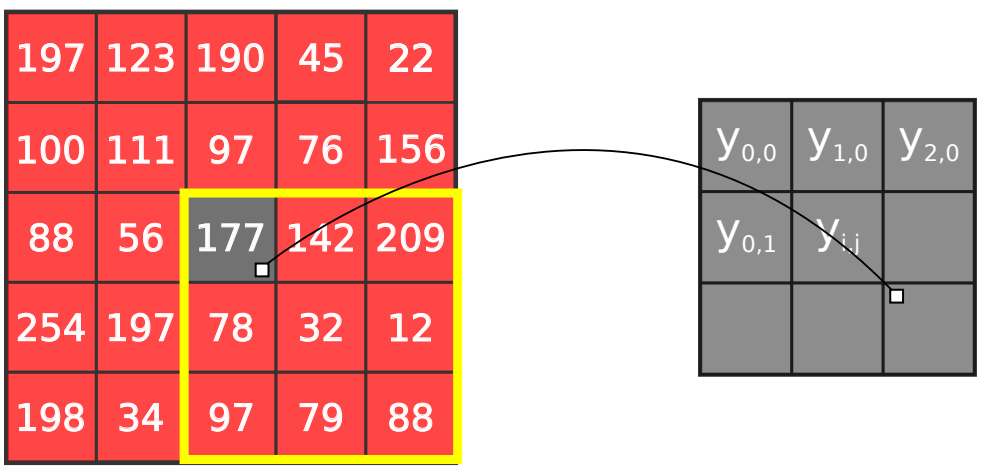
Convolution Layer: Backward Pass

Cross-Correlation Derivative

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

The **derivative** w.r.t. an **input parameter** x_{lmn} is $\frac{\partial y_{i'j'k'}}{\partial x_{lmn}} = w_{lmnk'}$.

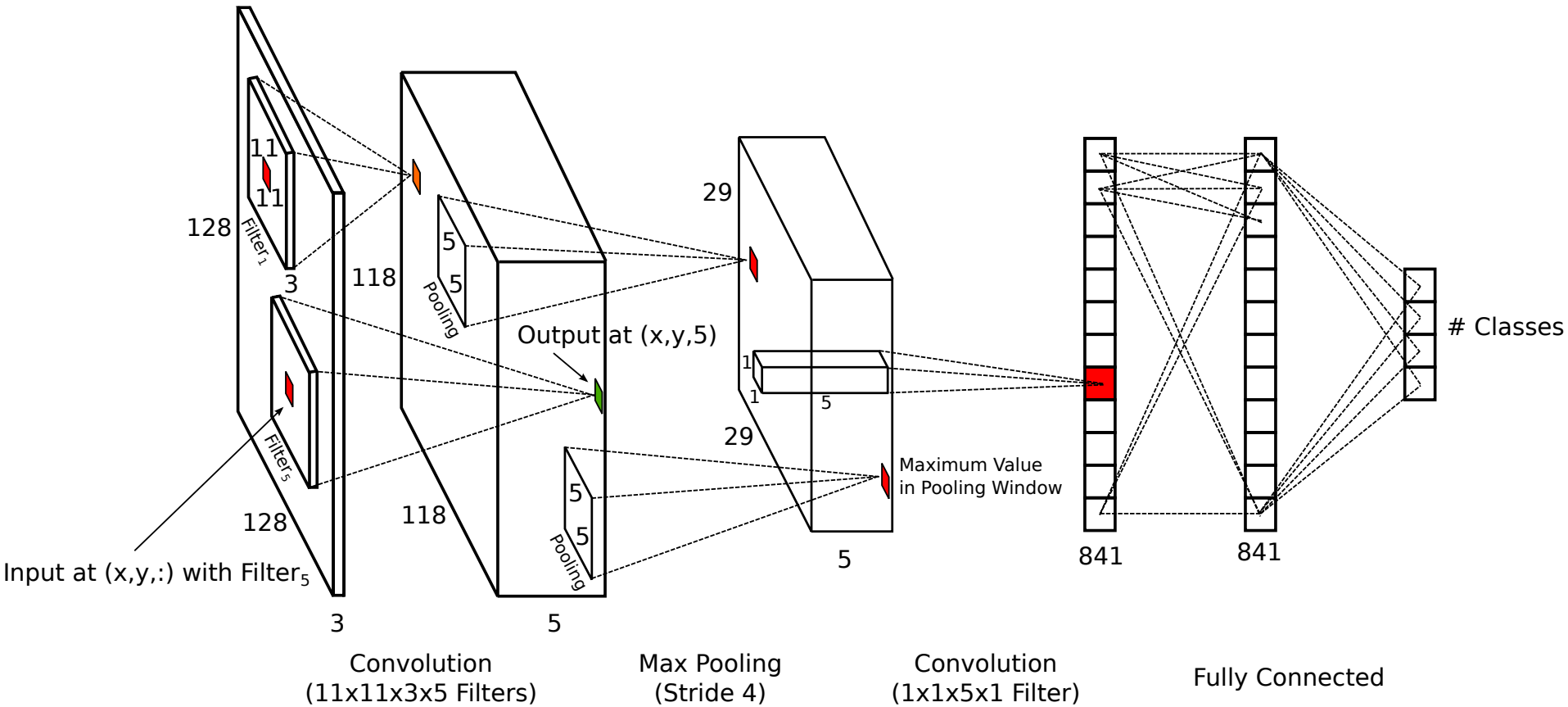
Again: A single input **contributes to multiple outputs**!



It's a convolution!

Back to the Beginning

Back to the Beginning



ConvNets in the Wild: Again Performance!

Performance in the Wild

The winning architecture of the LSVRC-2010 challenge (**AlexNet**) comes with **60 million parameters and 500 000 neurons**. It is trained on **1.3 million high-resolution images** in a **1000 class** problem.

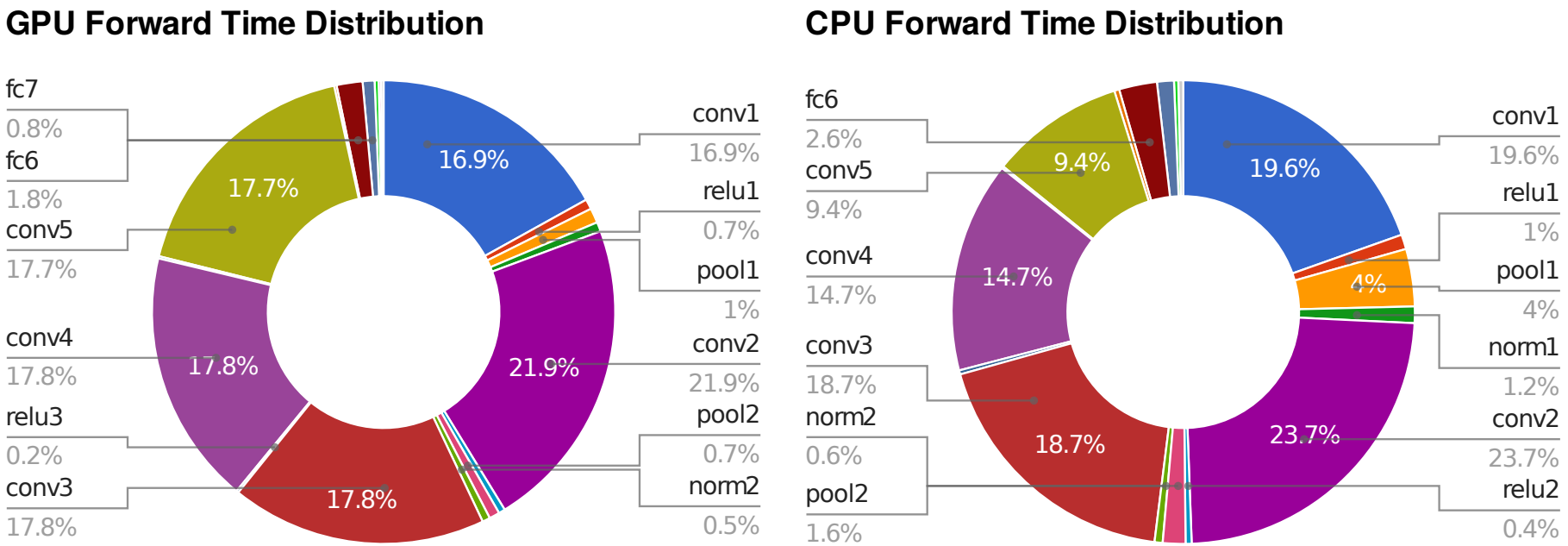


Figure : Jia: Time distribution of CNN layers in the AlexNet architecture. 95% GPU and 89% CPU of time is spent in convolution layers.

Performance in the Wild

Processing performance is usually measured in floating-point operations per second (FLOPS). All mathematical **operations** in ConvNets are **memory bound** hence memory bandwidth is the **most important factor**.

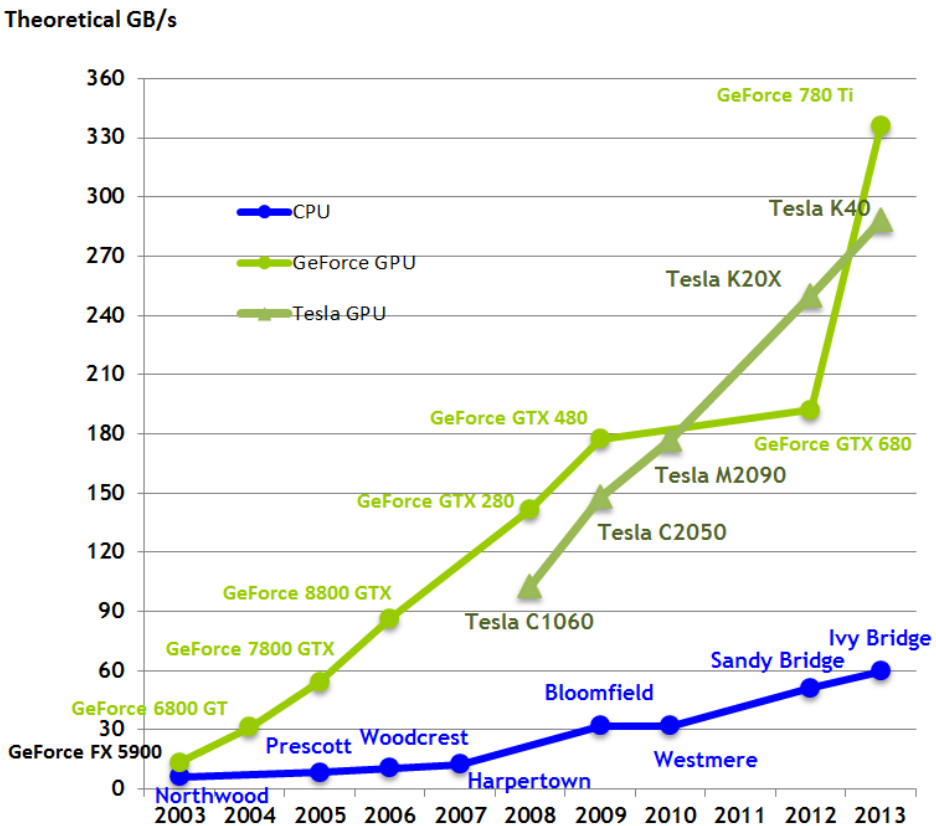


Figure : Dettmers: Memory bandwidth in GB/s for CPUs and GPUs over time

Network Refinement or Learning Transfer

Help: I can not train this net on my **crappy hardware!**

Network Refinement or Learning Transfer

Help: I can not train this net on my **crappy hardware!**

1. A **coming trend** will be **refining pre-trained** nets for specific problems

Network Refinement or Learning Transfer

Help: I can not train this net on my **crappy hardware!**

1. A **coming trend** will be **refining pre-trained** nets for specific problems
2. Humans **generalize** pretty well to **exotic tasks**

Network Refinement or Learning Transfer

Help: I can not train this net on my **crappy hardware!**

1. A **coming trend** will be **refining pre-trained** nets for specific problems
2. Humans **generalize** pretty well to **exotic tasks**
3. The idea is pretty simple: Use a **trained net** (you need the architecture plus all trained parameters) and just **re-iterate SGD** for certain layers

Network Refinement or Learning Transfer

Help: I can not train this net on my **crappy hardware!**

1. A **coming trend** will be **refining pre-trained** nets for specific problems
2. Humans **generalize** pretty well to **exotic tasks**
3. The idea is pretty simple: Use a **trained net** (you need the architecture plus all trained parameters) and just **re-iterate SGD** for certain layers
4. Adapt the **learning rates**: Low for lower layers, high for higher layers

Network Refinement or Learning Transfer

Help: I can not train this net on my **crappy hardware!**

1. A **coming trend** will be **refining pre-trained** nets for specific problems
2. Humans **generalize** pretty well to **exotic tasks**
3. The idea is pretty simple: Use a **trained net** (you need the architecture plus all trained parameters) and just **re-iterate SGD** for certain layers
4. Adapt the **learning rates**: Low for lower layers, high for higher layers
5. This potentially **reduces the required time** for training and **amount of data** significantly

ConvNets in the Wild: Software

ConvNet Software

Help: I do not want to implement this on my **crappy hardware!**

ConvNet Software

Help: I do not want to implement this on my **crappy hardware!**

1. **Few** people go through **writing their on ConvNet** implementation (except for educational purposes)

ConvNet Software

Help: I do not want to implement this on my **crappy hardware!**

1. **Few** people go through **writing their on ConvNet** implementation (except for educational purposes)
2. There is a **lot of good** (and free) **software** available

ConvNet Software

Help: I do not want to implement this on my **crappy hardware!**

1. **Few** people go through **writing their on ConvNet** implementation (except for educational purposes)
2. There is a **lot of good** (and free) **software** available
3. The current hype leads to **many people** using ConvNets **without knowing much** about inner workings

ConvNet Software

Help: I do not want to implement this on my **crappy hardware!**

1. **Few** people go through **writing their on ConvNet** implementation (except for educational purposes)
2. There is a **lot of good** (and free) **software** available
3. The current hype leads to **many people** using ConvNets **without knowing much** about inner workings
4. There are **trained nets available** to use and refine

ConvNet Software

Help: What software should I run on my **crappy hardware!**

ConvNet Software

Help: What software should I run on my **crappy hardware!**

1. Some good implementations are: **Caffe, MatConvNet** and **Theano**

ConvNet Software

Help: What software should I run on my **crappy hardware!**

1. Some good implementations are: **Caffe, MatConvNet** and **Theano**
2. **Caffe** comes pretty much as **black box** (there is a python and MATLAB interface but APIs are changing a lot)

ConvNet Software

Help: What software should I run on my **crappy hardware!**

1. Some good implementations are: **Caffe, MatConvNet** and **Theano**
2. **Caffe** comes pretty much as **black box** (there is a python and MATLAB interface but APIs are changing a lot)
3. **MatConvNet** gives you a **lot of freedom** by requires more work and a solid knowledge of ConvNets

ConvNet Software

Help: What software should I run on my **crappy hardware!**

1. Some good implementations are: **Caffe, MatConvNet** and **Theano**
2. **Caffe** comes pretty much as **black box** (there is a python and MATLAB interface but APIs are changing a lot)
3. **MatConvNet** gives you a **lot of freedom** by requires more work and a solid knowledge of ConvNets
4. **Theano** is capable of doing **symbolic differentiation**, you don't have to write code for back-propagation

ConvNet Software

Help: What software should I run on my **crappy hardware!**

1. Some good implementations are: **Caffe, MatConvNet** and **Theano**
2. **Caffe** comes pretty much as **black box** (there is a python and MATLAB interface but APIs are changing a lot)
3. **MatConvNet** gives you a **lot of freedom** by requires more work and a solid knowledge of ConvNets
4. **Theano** is capable of doing **symbolic differentiation**, you don't have to write code for back-propagation
5. **BLAS** packages **fine-tuned** for your machine can improve the **speed** drastically

ConvNets in the Wild: MatConvNet

MatConvNet

MatConvNet is a free **CNN MATLAB toolbox** written by Andrea Vedaldi and Karel Lenc (VLFeat).

MatConvNet

MatConvNet is a free **CNN MATLAB toolbox** written by Andrea Vedaldi and Karel Lenc (VLFeat).

1. **Efficient** CPU and GPU implementations based on C/C++

MatConvNet

MatConvNet is a free **CNN MATLAB toolbox** written by Andrea Vedaldi and Karel Lenc (VLFeat).

1. **Efficient** CPU and GPU implementations based on C/C++
2. **Pre-trained models**

MatConvNet

MatConvNet is a free **CNN MATLAB toolbox** written by Andrea Vedaldi and Karel Lenc (VLFeat).

1. **Efficient** CPU and GPU implementations based on C/C++
2. **Pre-trained models**
3. **Examples**

MatConvNet

MatConvNet is a free **CNN MATLAB toolbox** written by Andrea Vedaldi and Karel Lenc (VLFeat).

1. **Efficient** CPU and GPU implementations based on C/C++
2. **Pre-trained models**
3. **Examples**
4. **Well documented**

MatConvNet

MatConvNet is a free **CNN MATLAB toolbox** written by Andrea Vedaldi and Karel Lenc (VLFeat).

1. **Efficient** CPU and GPU implementations based on C/C++
2. **Pre-trained models**
3. **Examples**
4. **Well documented**
5. Can be used from MATLAB for **fast prototyping**

MatConvNet: Feature-maps and Kernel-maps

MatConvNet makes heavy used of MATLAB's **array data type**.

Feature-maps in MatConvNet

Feature-maps come as **4-dimensional** arrays (Height x Width x Depth x #Feature-maps).

MatConvNet: Feature-maps and Kernel-maps

MatConvNet makes heavy used of MATLAB's **array data type**.

Feature-maps in MatConvNet

Feature-maps come as **4-dimensional** arrays (Height x Width x Depth x #Feature-maps).

Kernel-maps in MatConvNet

A **Kernel-map** (or filter) is defined as a **4-dimensional** (Height x Width x Depth x #Filters) array. Filters for a convolution layer can simply be initialized as:

```
w = 0.1 .* randn(5, 5, 3, 256, 'single');
```

MatConvNet: Feature-maps and Kernel-maps

MatConvNet makes heavy used of MATLAB's **array data type**.

Feature-maps in MatConvNet

Feature-maps come as **4-dimensional** arrays (Height x Width x Depth x #Feature-maps).

Kernel-maps in MatConvNet

A **Kernel-map** (or filter) is defined as a **4-dimensional** (Height x Width x Depth x #Filters) array. Filters for a convolution layer can simply be initialized as:

```
w = 0.1 .* randn(5, 5, 3, 256, 'single');
```

Note that the **depth dimension** of kernel-maps and feature-maps have to be **equal** (except for filter-groups which I do not cover here).

MatConvNet: Layers

Each **layer** is implemented as a **separate function** in MatConvNet. The function prototypes are very similar and easy to learn:

MatConvNet: Layers

Each **layer** is implemented as a **separate function** in MatConvNet. The function prototypes are very similar and easy to learn:

1. Each function is used to compute the **forward and backward pass**

MatConvNet: Layers

Each **layer** is implemented as a **separate function** in MatConvNet. The function prototypes are very similar and easy to learn:

1. Each function is used to compute the **forward and backward pass**
2. Each function **requires the gradient of the previous layer** to compute the backward pass

MatConvNet: Layers

Each **layer** is implemented as a **separate function** in MatConvNet. The function prototypes are very similar and easy to learn:

1. Each function is used to compute the **forward and backward pass**
2. Each function **requires the gradient of the previous layer** to compute the backward pass

Convolution Layer in MatConvNet

For a given kernel-map w and an input batch x, the **forward pass** of the **convolution layer** can be computed as:

```
y = vl_nnconv(x, w, []);
```

To compute the **backward pass** we have to **supply the gradient** of the previous layer (or ones if there was no previous layer):

```
[DZDX, DZDW] = vl_nnconv(x, w, [], ones(size(y), 'single'));
```

MatConvNet: Layers

Other **layer-types** and their corresponding **functions** in MatConvNet are:

MatConvNet: Layers

Other **layer-types** and their corresponding **functions** in MatConvNet are:

- **Pooling**: `vl_nnpool`

MatConvNet: Layers

Other **layer-types** and their corresponding **functions** in MatConvNet are:

- **Pooling**: `vl_nnpool`
- **Non-Linear**: `vl_nnrelu`, `vl_nnsigmoid`

MatConvNet: Layers

Other **layer-types** and their corresponding **functions** in MatConvNet are:

- **Pooling**: `vl_nnpool`
- **Non-Linear**: `vl_nnrelu`, `vl_nnsigmoid`
- **Dropout**: `vl_nndropout`

MatConvNet: Layers

Other **layer-types** and their corresponding **functions** in MatConvNet are:

- **Pooling**: `vl_nnpool`
- **Non-Linear**: `vl_nnrelu`, `vl_nnsigmoid`
- **Dropout**: `vl_nndropout`
- **Softmax**: `vl_nnsoftmax`

MatConvNet: Layers

Other **layer-types** and their corresponding **functions** in MatConvNet are:

- **Pooling**: `vl_nnpool`
- **Non-Linear**: `vl_nnrelu`, `vl_nnsigmoid`
- **Dropout**: `vl_nndropout`
- **Softmax**: `vl_nnsoftmax`
- **Loss**: `vl_nnloss`, `vl_nnsoftmaxloss`, `vl_nnpdist`

MatConvNet: Layers

Other **layer-types** and their corresponding **functions** in MatConvNet are:

- **Pooling**: `vl_nnpool`
- **Non-Linear**: `vl_nnrelu`, `vl_nnsigmoid`
- **Dropout**: `vl_nndropout`
- **Softmax**: `vl_nnsoftmax`
- **Loss**: `vl_nnloss`, `vl_nnsoftmaxloss`, `vl_nnpdist`
- **Normalization**: `vl_nnoffset`, `vl_nnspnorm`, `vl_nnnormalize`

MatConvNet: Layers

Other **layer-types** and their corresponding **functions** in MatConvNet are:

- **Pooling**: `vl_nnpool`
- **Non-Linear**: `vl_nnrelu`, `vl_nnsigmoid`
- **Dropout**: `vl_nndropout`
- **Softmax**: `vl_nnsoftmax`
- **Loss**: `vl_nnloss`, `vl_nnsoftmaxloss`, `vl_nnpdist`
- **Normalization**: `vl_nnoffset`, `vl_nnspnorm`, `vl_nnnormalize`

For a full API reference visit:
`http://www.vlfeat.org/matconvnet/functions`

MatConvNet: SimpleCNN Wrapper

There is a **wrapper** that let's us **define** a CNN **architecture** and does the **forward and backward passes**: `vl_simplenn`.

Defining a Convolution Layer in MatConvNet

A layer is defined as a **structure array** with an entry for each layer:

```
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{w, b}}, ...
                           'stride', 1, 'pad', 0, ...
                           'learningRate', [1 2], ...
                           'weightDecay', [0.0001 0]);
```

MatConvNet: SimpleCNN Wrapper

There is a **wrapper** that let's us **define** a CNN **architecture** and does the **forward and backward passes**: `vl_simplenn`.

Defining a Pooling Layer in MatConvNet

```
net.layers{end+1} = struct('type', 'pool', ...
                           'method', 'max', ...
                           'pool', [3 3], ...
                           'stride', 2, ...
                           'pad', 0) ;
```

MatConvNet: SimpleCNN Wrapper

There is a **wrapper** that let's us **define** a CNN **architecture** and does the **forward and backward passes**: `vl_simplenn`.

Defining a Pooling Layer in MatConvNet

```
net.layers{end+1} = struct('type', 'pool', ...
                           'method', 'max', ...
                           'pool', [3 3], ...
                           'stride', 2, ...
                           'pad', 0) ;
```

Each layer requires specific parameters. There are many examples to learn (or steal) from in MatConvNet.

MatConvNet: SimpleCNN Wrapper

For a defined ConvNet `vl_simplenn` can be called to **compute an iteration of forward and backward passes**.

Evaluating a ConvNet

```
res = vl_simplenn(x, net)
```

1. `res` is a **structure array** with one element per network layer plus one representing the input to the net
2. `res(i+1).x` is the **output of layer** `i`
3. `res(i+1).dzdx` is the **derivative of the network output** w.r.t. `res(i+1).x`

MatConvNet: Things you have to do yourself

MatConvNet gives you a **lot of control** but this also means you have to **write quite some code** yourself:

MatConvNet: Things you have to do yourself

MatConvNet gives you a **lot of control** but this also means you have to **write quite some code** yourself:

1. **Data processing** code has to be written

MatConvNet: Things you have to do yourself

MatConvNet gives you a **lot of control** but this also means you have to **write quite some code** yourself:

1. **Data processing** code has to be written
2. **SGD/Optimization** code has to be written

MatConvNet: Things you have to do yourself

MatConvNet gives you a **lot of control** but this also means you have to **write quite some code** yourself:

1. **Data processing** code has to be written
2. **SGD/Optimization** code has to be written
3. **Cross-validation/Evaluation** code has to be written

MatConvNet: Things you have to do yourself

MatConvNet gives you a **lot of control** but this also means you have to **write quite some code** yourself:

1. **Data processing** code has to be written
2. **SGD/Optimization** code has to be written
3. **Cross-validation/Evaluation** code has to be written

But you know what is actually done!

ConvNets in the Wild: Rules of Thumb

ConvNets in the Wild: Rules of Thumb

When **designing** a deep network some **rules of thumb** can help:

ConvNets in the Wild: Rules of Thumb

When **designing** a deep network some **rules of thumb** can help:

1. Use **few large filters** in lower layers, **many small filters** in upper layers

ConvNets in the Wild: Rules of Thumb

When **designing** a deep network some **rules of thumb** can help:

1. Use **few large filters** in lower layers, **many small filters** in upper layers
2. Use **large strides** in lower layers, **small strides** in upper layers

ConvNets in the Wild: Rules of Thumb

When **designing** a deep network some **rules of thumb** can help:

1. Use **few large filters** in lower layers, **many small filters** in upper layers
2. Use **large strides** in lower layers, **small strides** in upper layers
3. Use the **general building block**: **INPUT - [CONV, RELU, POOL]* DROPOUT - FC - RELU - DROPOUT - FC - SOFTMAX - LOSS**

ConvNets in the Wild: Rules of Thumb

When **designing** a deep network some **rules of thumb** can help:

1. Use **few large filters** in lower layers, **many small filters** in upper layers
2. Use **large strides** in lower layers, **small strides** in upper layers
3. Use the **general building block**: **INPUT - [CONV, RELU, POOL]* DROPOUT - FC - RELU - DROPOUT - FC - SOFTMAX - LOSS**
4. Use **data augmentation**

ConvNets in the Wild: Rules of Thumb

When **designing** a deep network some **rules of thumb** can help:

1. Use **few large filters** in lower layers, **many small filters** in upper layers
2. Use **large strides** in lower layers, **small strides** in upper layers
3. Use the **general building block**: **INPUT - [CONV, RELU, POOL]* DROPOUT - FC - RELU - DROPOUT - FC - SOFTMAX - LOSS**
4. Use **data augmentation**
5. Use **oversampling**

ConvNets in the Wild: Rules of Thumb

When **designing** a deep network some **rules of thumb** can help:

1. Use **few large filters** in lower layers, **many small filters** in upper layers
2. Use **large strides** in lower layers, **small strides** in upper layers
3. Use the **general building block**: **INPUT - [CONV, RELU, POOL]* DROPOUT - FC - RELU - DROPOUT - FC - SOFTMAX - LOSS**
4. Use **data augmentation**
5. Use **oversampling**
6. Use **data normalization** (mean subtraction)

ConvNets in the Wild: Rules of Thumb

When **designing** a deep network some **rules of thumb** can help:

1. Use **few large filters** in lower layers, **many small filters** in upper layers
2. Use **large strides** in lower layers, **small strides** in upper layers
3. Use the **general building block**: **INPUT - [CONV, RELU, POOL]* DROPOUT - FC - RELU - DROPOUT - FC - SOFTMAX - LOSS**
4. Use **data augmentation**
5. Use **oversampling**
6. Use **data normalization** (mean subtraction)
7. Use **batch-sizes appropriate** for your hardware and problem

ConvNets in the Wild: Visualizing and Analyzing Deep Architectures

Fooling ConvNets

Deep architectures have demonstrated **human-competitive results**. But it is **relatively easy** to create high confidence images that **fool a ConvNet**.

Fooling ConvNets

Deep architectures have demonstrated **human-competitive results**. But it is **relatively easy** to create high confidence images that **fool a ConvNet**.

1. One popular benchmark is the **recognition of digits** (e.g., MNIST).

Figure : The MNIST dataset.

Fooling ConvNets

Deep architectures have demonstrated **human-competitive results**. But it is **relatively easy** to create high confidence images that **fool a ConvNet**.

1. One popular benchmark is the **recognition of digits** (e.g., MNIST).
2. Starting randomly, **inputs are optimized** to **maximize certain activations**

Fooling ConvNets

Deep architectures have demonstrated **human-competitive results**. But it is **relatively easy** to create high confidence images that **fool a ConvNet**.

1. One popular benchmark is the **recognition of digits** (e.g., MNIST).
2. Starting randomly, **inputs are optimized** to **maximize certain activations**
3. This is either done **evolutionary** or with **gradient ascent**

Fooling ConvNets

Deep architectures have demonstrated **human-competitive results**. But it is **relatively easy** to create high confidence images that **fool a ConvNet**.

1. One popular benchmark is the **recognition of digits** (e.g., MNIST).
2. Starting randomly, **inputs are optimized** to **maximize certain activations**
3. This is either done **evolutionary** or with **gradient ascent**
4. Consequently **inputs** can be **constructed** that will **fool a ConvNet**

Fooling ConvNets

Created images that **fool a ConvNet** but would **never be interpreted** as a certain class by a human observer:

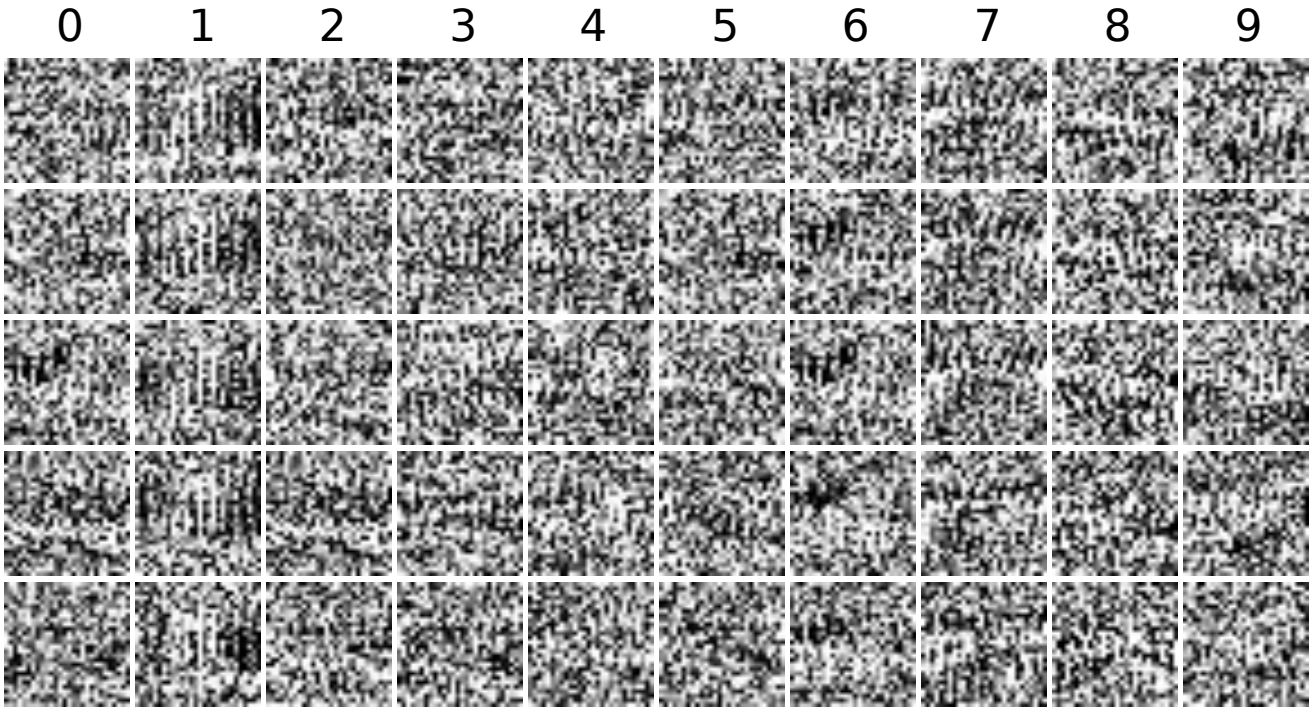


Figure : Nguyen et al.: Evolutionary optimized images that MNIST CNNs believe with 99.99% confidence are digits 0-9

Fooling ConvNets

Created images that **fool a ConvNet** but would **never be interpreted** as a certain class by a human observer:

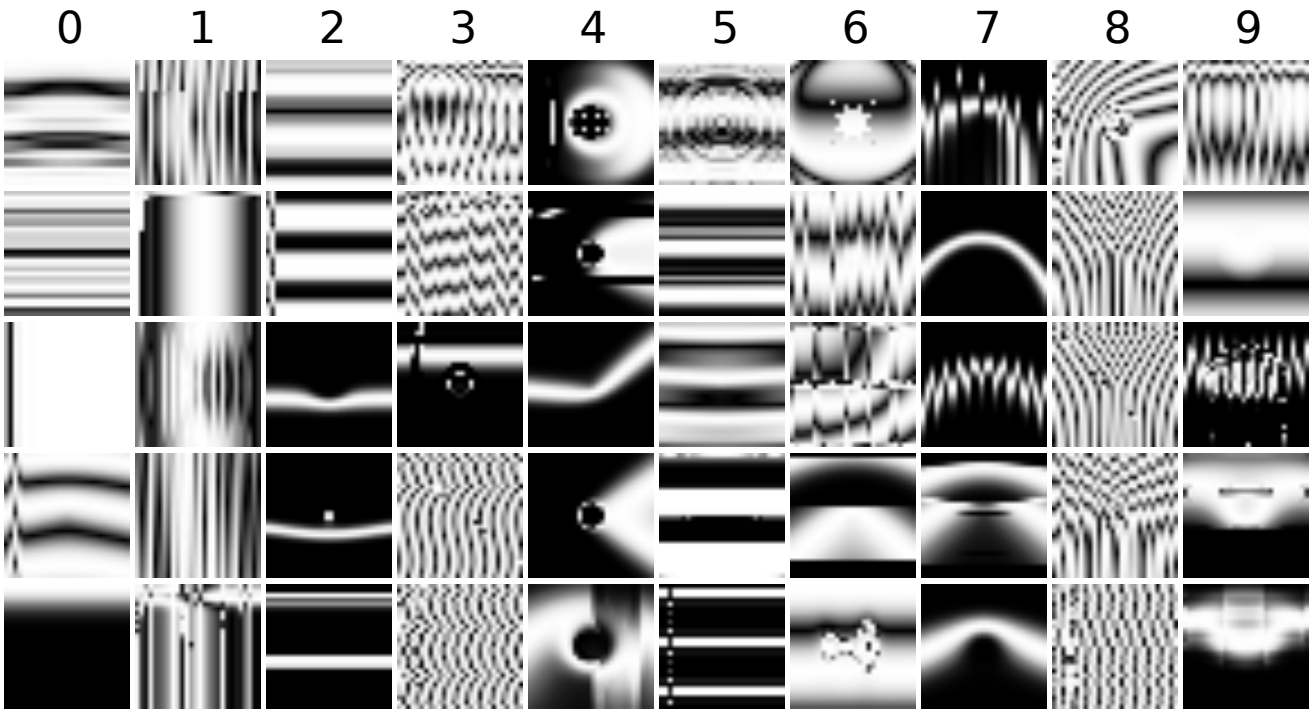


Figure : Nguyen et al.: Evolutionary optimized images that MNIST CNNs believe with 99.99% confidence are digits 0-9

Fooling ConvNets

Created images that **fool a ConvNet** but would **never be interpreted** as a certain class by a human observer:

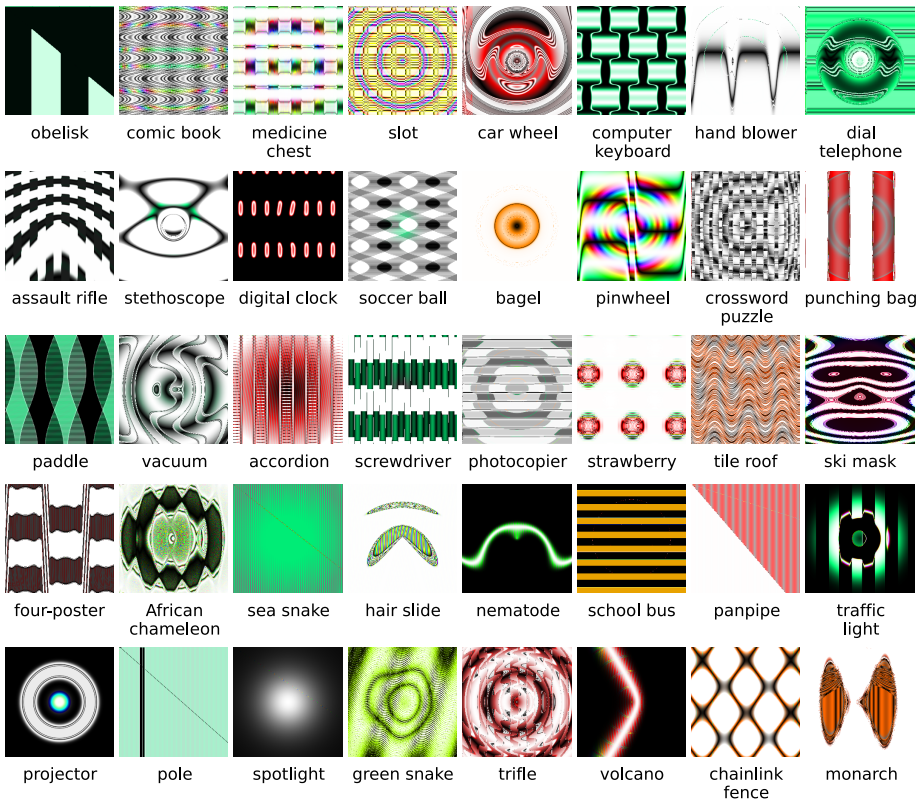


Figure : Nguyen et al.: Evolutionary optimized images for ImageNet CNNs with mean confidence score of 99.12%

Fooling ConvNets

Created images that **fool a ConvNet** but would **never be interpreted** as a certain class by a human observer:

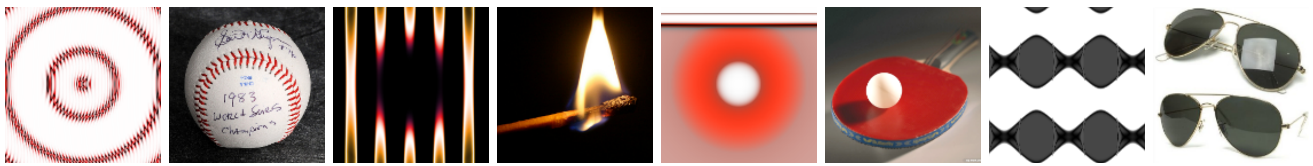


Figure : Nguyen et al.: Evolved images resembling target class.

ConvNets in the Wild: Deep Dreaming

Deep Dreaming

Another way of **visualizing what a ConvNets sees** is called **Inceptionism** and was invented by Google Research.

Deep Dreaming

Another way of **visualizing what a ConvNets sees** is called **Inceptionism** and was invented by Google Research.

1. **Gradient** of the posterior probability for a **specific class** is computed

Deep Dreaming

Another way of **visualizing what a ConvNets sees** is called **Inceptionism** and was invented by Google Research.

1. **Gradient** of the posterior probability for a **specific class** is computed
2. **Gradient ascent** with L^2 regularization **maximizing the activation** of a certain unit

Deep Dreaming

Another way of **visualizing what a ConvNets sees** is called **Inceptionism** and was invented by Google Research.

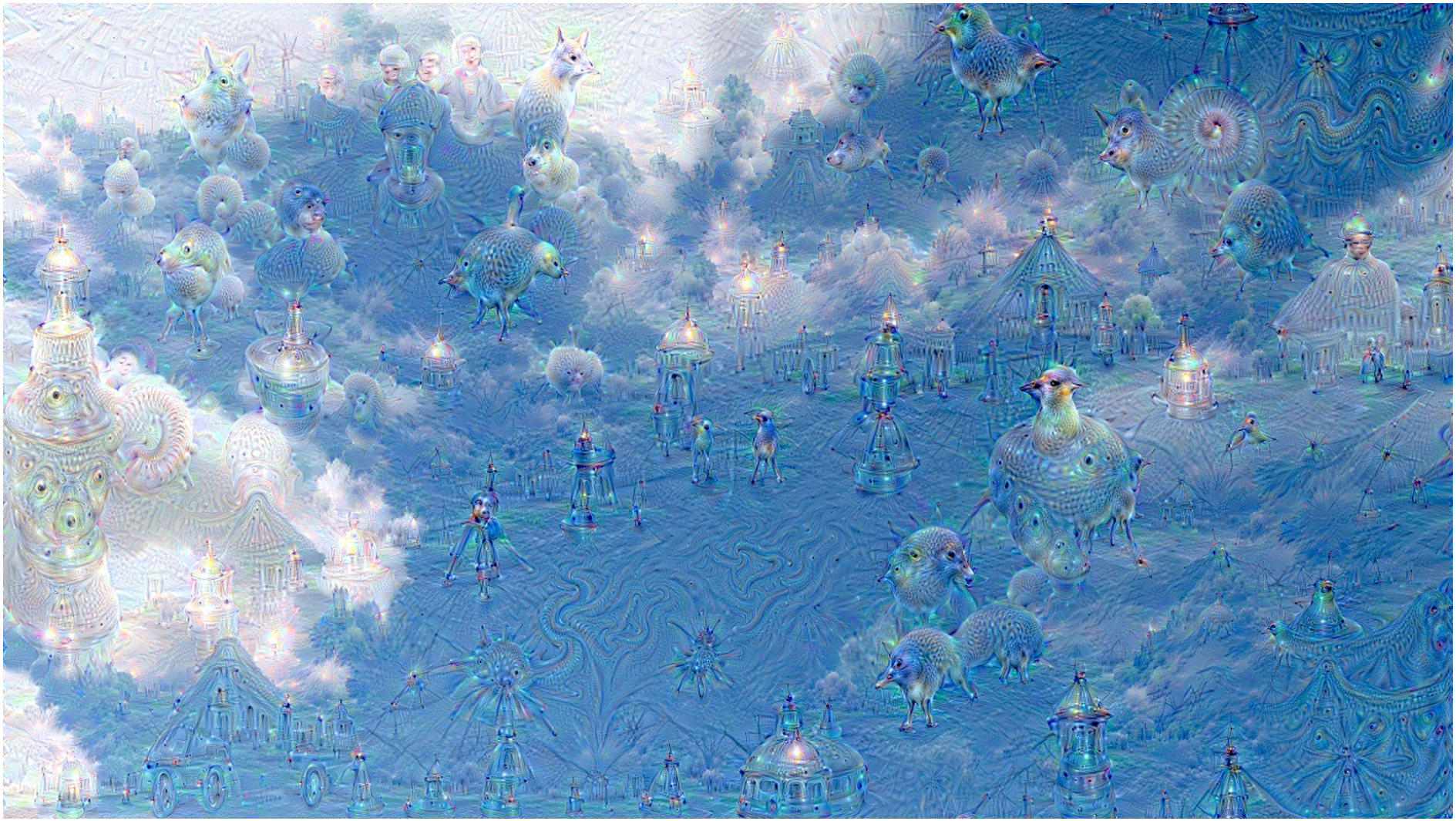
1. **Gradient** of the posterior probability for a **specific class** is computed
2. **Gradient ascent** with L^2 regularization **maximizing the activation** of a certain unit
3. Instead of weights, the **activations** in a layer are **optimized**

Deep Dreaming

Another way of **visualizing what a ConvNets sees** is called **Inceptionism** and was invented by Google Research.

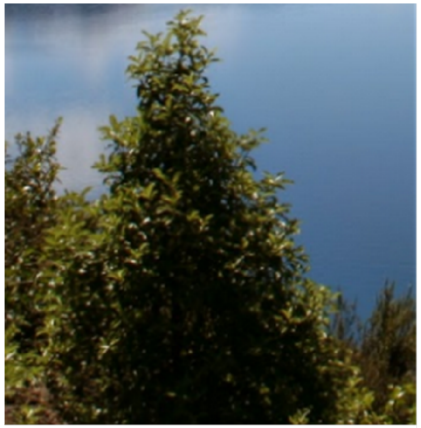
1. **Gradient** of the posterior probability for a **specific class** is computed
2. **Gradient ascent** with L^2 regularization **maximizing the activation** of a certain unit
3. Instead of weights, the **activations** in a layer are **optimized**
4. This is **iterated multiple times feeding the maximized output into the net** and leads to very interesting results

Deep Dreaming



Deep Dreaming

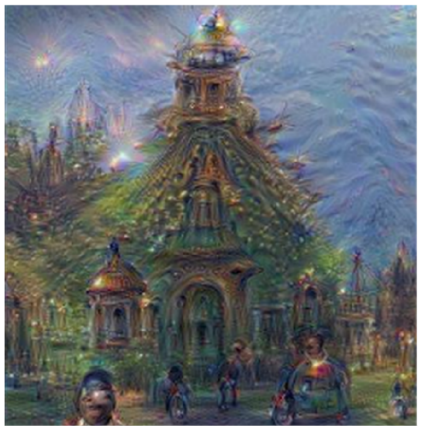
Horizon



Trees

Leaves

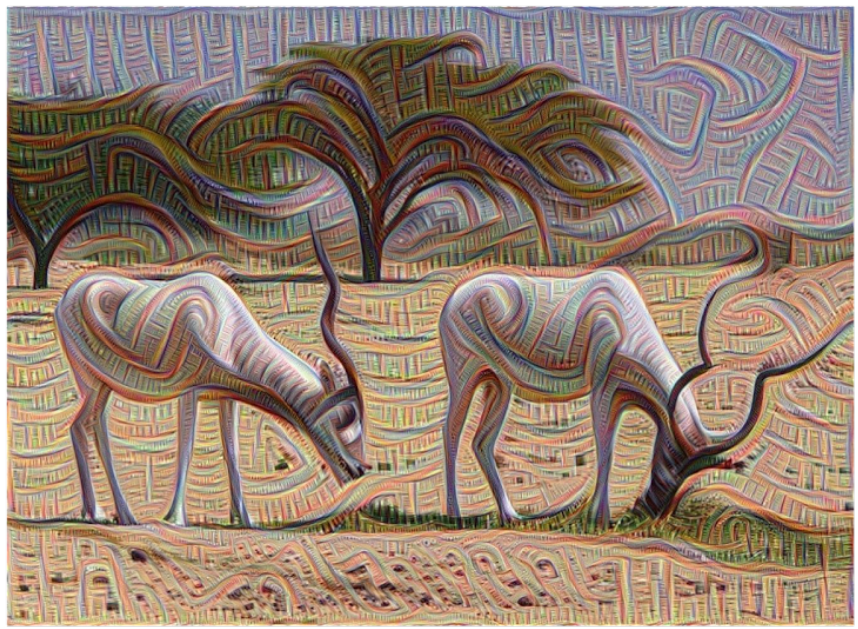
Towers & Pagodas



Buildings

Birds & Insects

Deep Dreaming



Deep Dreaming

Deep Dreaming

I am (finally) done!