More on Convolutional Neural Networks

Romuald A. Janik

Data science/machine learning journal club at WFAIS UJ

Outline

Convolutional Neural Networks (CNN)

Going deep

CNN example: MNIST

Image classification datasets

Recent CNN architectures

- Inception/Xception
- Residual networks
- Densely connected convolutional networks

Interlude — fooling deep CNN's

CNN for segmentation

- Fully Convolutional Networks — U-Net

Conclusions

What are neural networks?

- Each neuron with inputs x_k from previous layer has the output y_i given by

$$y_i = f(W_{ik} x_k + b_i)$$

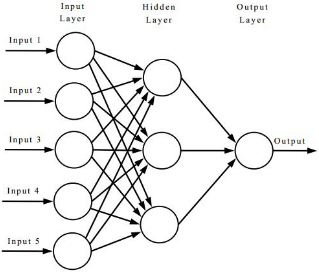
- The coefficients W_{ik} (*weights*) and b_i (*bias*) are determined during training
- $f(.)$ is a **nonlinear** 'activation function'

$$f(x) = \tanh x \longrightarrow \underbrace{f(x) = max(0, x)}_{ReLU}$$

- All neurons of a layer are connected to all neurons of the previous one
- This is good for general situations – but has lots of parameters (especially for deep networks)!
- Suboptimal for images

Neural networks

What are neural networks?



- Each neuron with inputs x_k from previous layer has the output y_i given by

$$y_i = f(W_{ik}x_k + b_i)$$

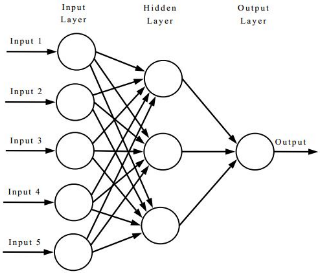
- The coefficients W_{ik} (*weights*) and b_i (*bias*) are determined during training
- $f(.)$ is a **nonlinear** 'activation function'

$$f(x) = \tanh x \longrightarrow \underbrace{f(x) = max(0, x)}_{ReLU}$$

- All neurons of a layer are connected to all neurons of the previous one
- This is good for general situations – but has lots of parameters (especially for deep networks)!
- Suboptimal for images

Neural networks

What are neural networks?



- Each neuron with inputs x_k from previous layer has the output y_i given by

$$y_i = f(W_{ik}x_k + b_i)$$

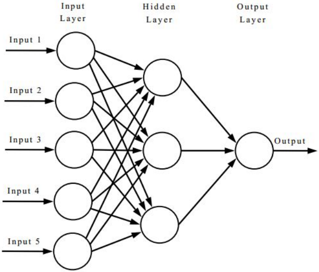
- The coefficients W_{ik} (*weights*) and b_i (*bias*) are determined during training
- $f(.)$ is a **nonlinear** 'activation function'

$$f(x) = \tanh x \longrightarrow \underbrace{f(x) = max(0, x)}_{ReLU}$$

- All neurons of a layer are connected to all neurons of the previous one
- This is good for general situations – but has lots of parameters (especially for deep networks)!
- Suboptimal for images

Neural networks

What are neural networks?



- Each neuron with inputs x_k from previous layer has the output y_i given by

$$y_i = f(W_{ik} x_k + b_i)$$

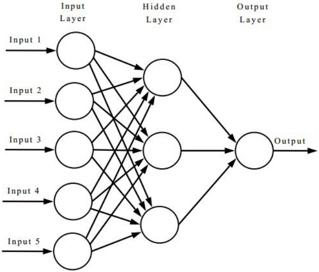
- The coefficients W_{ik} (*weights*) and b_i (*bias*) are determined during training
- $f(.)$ is a **nonlinear** 'activation function'

$$f(x) = \tanh x \longrightarrow \underbrace{f(x) = max(0, x)}_{ReLU}$$

- All neurons of a layer are connected to all neurons of the previous one
- This is good for general situations – but has lots of parameters (especially for deep networks)!
- Suboptimal for images

Neural networks

What are neural networks?



- Each neuron with inputs x_k from previous layer has the output y_i given by

$$y_i = f(W_{ik}x_k + b_i)$$

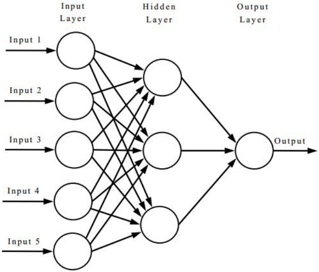
- The coefficients W_{ik} (*weights*) and b_i (*bias*) are determined during training
- $f(.)$ is a **nonlinear** 'activation function'

$$f(x) = \tanh x \longrightarrow \underbrace{f(x) = max(0, x)}_{ReLU}$$

- All neurons of a layer are connected to all neurons of the previous one
- This is good for general situations – but has lots of parameters (especially for deep networks)!
- Suboptimal for images

Neural networks

What are neural networks?



- Each neuron with inputs x_k from previous layer has the output y_i given by

$$y_i = f(W_{ik} x_k + b_i)$$

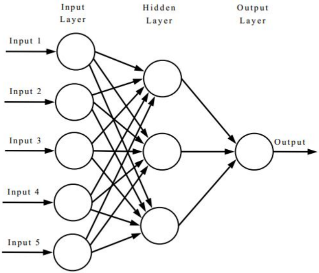
- The coefficients W_{ik} (*weights*) and b_i (*bias*) are determined during training
- $f(.)$ is a **nonlinear** ‘activation function’

$$f(x) = \tanh x \longrightarrow \underbrace{f(x) = max(0, x)}_{ReLU}$$

- All neurons of a layer are connected to all neurons of the previous one
- This is good for general situations – but has lots of parameters (especially for deep networks)!
- Suboptimal for images

Neural networks

What are neural networks?



- Each neuron with inputs x_k from previous layer has the output y_i given by

$$y_i = f(W_{ik} x_k + b_i)$$

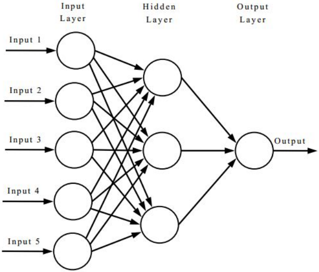
- The coefficients W_{ik} (*weights*) and b_i (*bias*) are determined during training
- $f(.)$ is a **nonlinear** 'activation function'

$$f(x) = \tanh x \longrightarrow \underbrace{f(x) = max(0, x)}_{ReLU}$$

- All neurons of a layer are connected to all neurons of the previous one
- This is good for general situations – but has lots of parameters (especially for deep networks)!
- Suboptimal for images

Neural networks

What are neural networks?



- Each neuron with inputs x_k from previous layer has the output y_i given by

$$y_i = f(W_{ik} x_k + b_i)$$

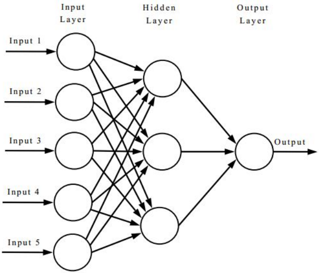
- The coefficients W_{ik} (*weights*) and b_i (*bias*) are determined during training
- $f(.)$ is a **nonlinear** 'activation function'

$$f(x) = \tanh x \longrightarrow \underbrace{f(x) = max(0, x)}_{ReLU}$$

- All neurons of a layer are connected to all neurons of the previous one
- This is good for general situations – but has lots of parameters (especially for deep networks)!
- Suboptimal for images

Neural networks

What are neural networks?



- Each neuron with inputs x_k from previous layer has the output y_i given by

$$y_i = f(W_{ik}x_k + b_i)$$

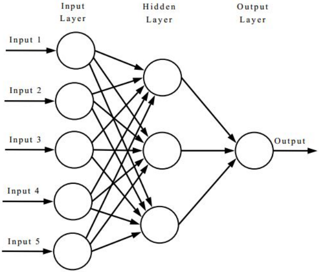
- The coefficients W_{ik} (*weights*) and b_i (*bias*) are determined during training
- $f(.)$ is a **nonlinear** 'activation function'

$$f(x) = \tanh x \longrightarrow \underbrace{f(x) = max(0, x)}_{ReLU}$$

- All neurons of a layer are connected to all neurons of the previous one
- This is good for general situations – but has lots of parameters (especially for deep networks)!
- Suboptimal for images

Neural networks

What are neural networks?



- Each neuron with inputs x_k from previous layer has the output y_i given by

$$y_i = f(W_{ik} x_k + b_i)$$

- The coefficients W_{ik} (*weights*) and b_i (*bias*) are determined during training
- $f(.)$ is a **nonlinear** 'activation function'

$$f(x) = \tanh x \longrightarrow \underbrace{f(x) = max(0, x)}_{ReLU}$$

- All neurons of a layer are connected to all neurons of the previous one
- This is good for general situations – but has lots of parameters (especially for deep networks)!
- Suboptimal for images

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights

from `http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution`

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights

from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

Main ideas:

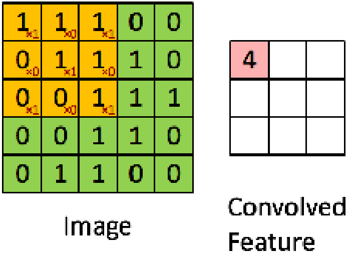
1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights

from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights



$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

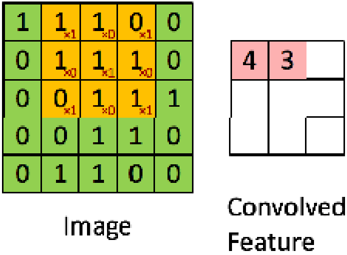
Convolved Feature

from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights

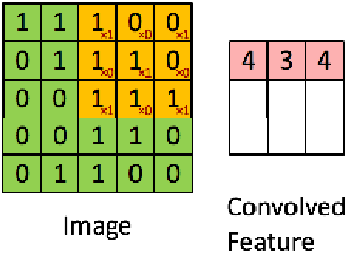


from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights



1	1	$1_{\times 1}$	$0_{\times 0}$	$0_{\times 1}$
0	1	$1_{\times 0}$	$1_{\times 1}$	$0_{\times 0}$
0	0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$
0	0	1	1	0
0	1	1	0	0

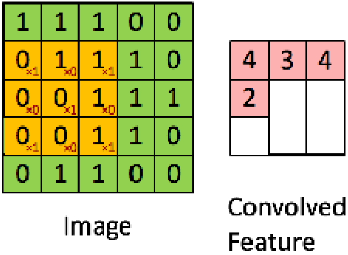
4	3	4

from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights



1	1	1	0	0
$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	1	0
$0_{\times 0}$	$0_{\times 1}$	$1_{\times 0}$	1	1
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	0
0	1	1	0	0

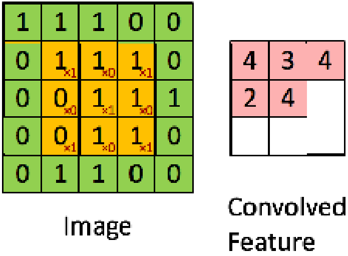
4	3	4
2		

from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights



1	1	1	0	0
0	$1_{\times1}$	$1_{\times0}$	$1_{\times1}$	0
0	$0_{\times0}$	$1_{\times1}$	$1_{\times0}$	1
0	$0_{\times1}$	$1_{\times0}$	$1_{\times1}$	0
0	1	1	0	0

Image

4	3	4
2	4	

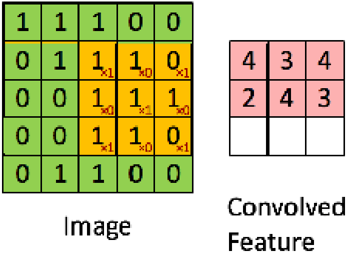
Convolved Feature

from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights



1	1	1	0	0
0	1	$1_{\times1}$	$1_{\times0}$	$0_{\times1}$
0	0	$1_{\times0}$	$1_{\times1}$	$1_{\times0}$
0	0	$1_{\times1}$	$1_{\times0}$	$0_{\times1}$
0	1	1	0	0

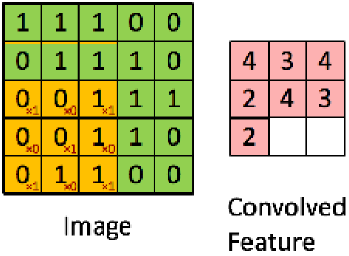
4	3	4
2	4	3

from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights



1	1	1	0	0
0	1	1	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
$0_{\times 0}$	$0_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0

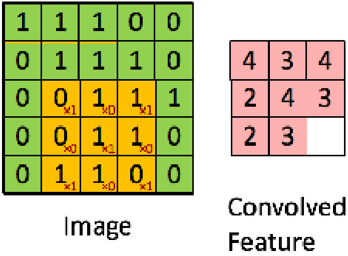
4	3	4
2	4	3
2		

from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say 3*x*3 neurons in the previous layer
3. For all these connections take the same weights



1	1	1	0	0
0	1	1	1	0
0	$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	1
0	$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	0
0	$1_{\times 1}$	$1_{\times 0}$	$0_{\times 1}$	0

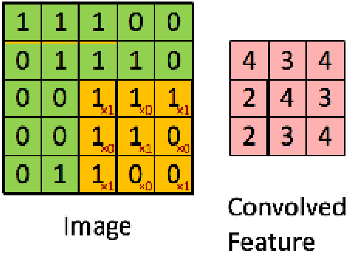
4	3	4
2	4	3
2	3	

from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

Main ideas:

1. Preserve the 2D structure of the input layer
2. Connect neurons in the next layer only to a block of say $3x3$ neurons in the previous layer
3. For all these connections take the same weights



1	1	1	0	0
0	1	1	1	0
0	0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$0_{\times 0}$
0	1	$1_{\times 1}$	$0_{\times 0}$	$0_{\times 1}$

Image

4	3	4
2	4	3
2	3	4

Convolved Feature

from http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks (CNN)

- Motivation from visual system:
  1. First layers detect simple features – like edges at various angles (convolution weights are also called filters)
  2. This information is composed into higher level features in deeper layers of the network
- We will have multiple filters at each hidden layer

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

Convolutional Neural Networks (CNN)

- Motivation from visual system:
  1. First layers detect simple features – like edges at various angles (convolution weights are also called filters)
  2. This information is composed into higher level features in deeper layers of the network
- We will have multiple filters at each hidden layer

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

Convolutional Neural Networks (CNN)

- Motivation from visual system:
  1. First layers detect simple features – like edges at various angles (convolution weights are also called filters)
  2. This information is composed into higher level features in deeper layers of the network
- We will have multiple filters at each hidden layer

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

Convolutional Neural Networks (CNN)

- Motivation from visual system:
  1. First layers detect simple features – like edges at various angles (convolution weights are also called filters)
  2. This information is composed into higher level features in deeper layers of the network
- We will have multiple filters at each hidden layer

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

Convolutional Neural Networks (CNN)

- Motivation from visual system:
  1. First layers detect simple features – like edges at various angles (convolution weights are also called filters)
  2. This information is composed into higher level features in deeper layers of the network
- We will have multiple filters at each hidden layer

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

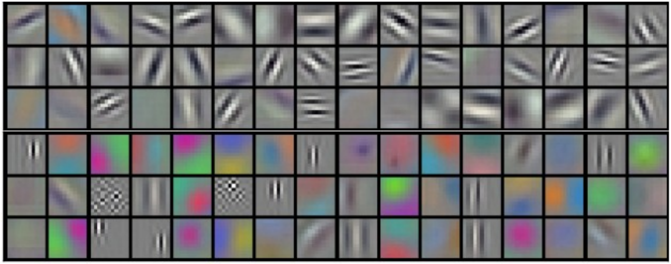
Convolutional Neural Networks (CNN)

- Motivation from visual system:
  1. First layers detect simple features – like edges at various angles (convolution weights are also called filters)
  2. This information is composed into higher level features in deeper layers of the network
- We will have multiple filters at each hidden layer

from `http://cs231n.github.io/convolutional-networks/`

Convolutional Neural Networks (CNN)

- Motivation from visual system:
  1. First layers detect simple features – like edges at various angles (convolution weights are also called filters)
  2. This information is composed into higher level features in deeper layers of the network
- We will have multiple filters at each hidden layer



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

Convolutional Neural Networks (CNN)

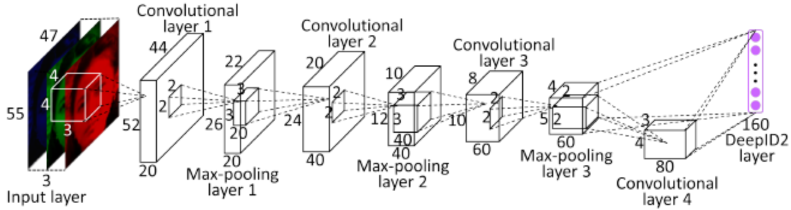
Sample structure of a CNN:

Implement translational invariance

⟶ pooling layers

Convolutional Neural Networks (CNN)

Sample structure of a CNN:

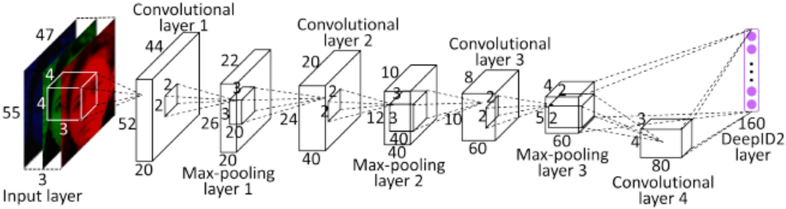


Implement translational invariance

⟶ pooling layers

Convolutional Neural Networks (CNN)

Sample structure of a CNN:



Implement translational invariance

⟶ **pooling layers**

Convolutional Neural Networks (CNN)

Max-pooling layer

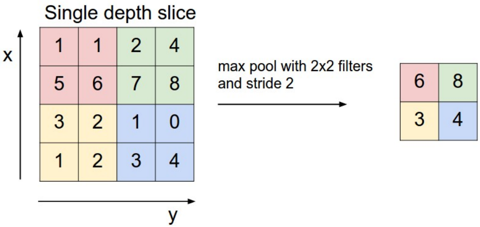
from http://cs231n.github.io/convolutional-networks/

Less popular: Average-pooling

Pooling layers realize approximate translational invariance

Convolutional Neural Networks (CNN)

Max-pooling layer



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

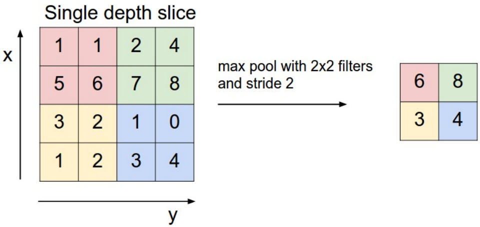
from http://cs231n.github.io/convolutional-networks/

Less popular: Average-pooling

Pooling layers realize approximate translational invariance

Convolutional Neural Networks (CNN)

Max-pooling layer



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

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

Less popular: Average-pooling

Pooling layers realize approximate translational invariance

Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:

  For a CNN:

  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..

Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:

  For a CNN:

  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..

Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:
  For a CNN:
  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..

Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:

  For a CNN:

  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..

Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:
  For a CNN:
  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..

Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:

  For a CNN:

  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..

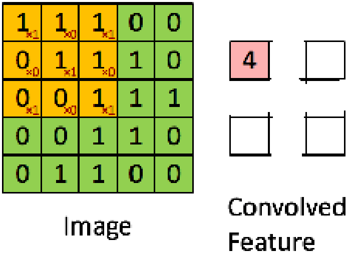
Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:

  For a CNN:

  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..



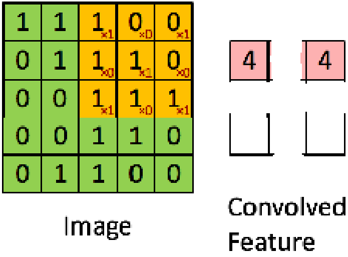
Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:

  For a CNN:

  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..



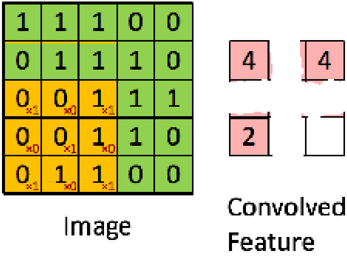
Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:

  For a CNN:

  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..



1	1	1	0	0
0	1	1	1	0
$0_{\times1}$	$0_{\times0}$	$1_{\times1}$	1	1
$0_{\times0}$	$0_{\times1}$	$1_{\times0}$	1	0
$0_{\times1}$	$1_{\times0}$	$1_{\times1}$	0	0

4	4
2	

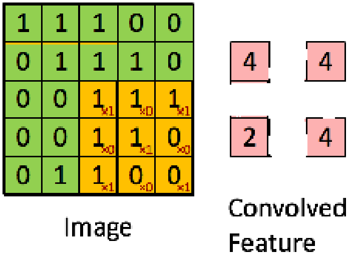
Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:

  For a CNN:

  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..



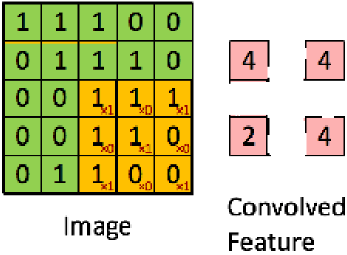
Convolutional Neural Networks (CNN)

In a machine learning model we can distinguish:

- **Parameters** – learned during training (e.g. weights of the connections between neurons)
- **Hyperparameters** – fixed *a-priori* – define the structure of the network:

  For a CNN:

  1. number and type of layers
  2. for each convolutional layer: filter size, number of channels/filters, stride (stride=2 in the figure below)
  3. Additional layers: dropout/batch normalization/flatten/fully connected..


1 1 1 0 0
0 1 1 1 0
0 0 $1_{\times 1}$ $1_{\times 0}$ $1_{\times 1}$
0 0 $1_{\times 0}$ $1_{\times 1}$ $0_{\times 0}$
0 1 $1_{\times 1}$ $0_{\times 0}$ $0_{\times 1}$
Image
4 4
2 4
Convolved Feature

Going deep

Key advances allowing for training deep networks:

1. **ReLU activation:** (Rectified Linear Unit) i.e. nonlinearity for a neuron:

$$y_i = f(W_{ik} x_k + b_i) \qquad \underbrace{f(x) = max(0, x)}_{ReLU}$$

2. **Dropout:** during training we turn off say 50% randomly chosen neurons from the previous layer

   https://arxiv.org/abs/1207.0580

   turns out to be absolutely crucial for training deep networks with lots of parameters and prevent overfitting..

3. **Batch normalization:** A layer which normalizes inputs for each training batch

   https://arxiv.org/abs/1502.03167

   increases training speed/also acts as a regularizer

4. **GPU computations** – computations on (NVIDIA) graphics cards...

Going deep

Key advances allowing for training deep networks:

1. **ReLU activation:** (Rectified Linear Unit) i.e. nonlinearity for a neuron:

$$y_i = f(W_{ik} x_k + b_i) \qquad \underbrace{f(x) = max(0, x)}_{ReLU}$$

2. **Dropout:** during training we turn off say 50% randomly chosen neurons from the previous layer

   https://arxiv.org/abs/1207.0580

   turns out to be absolutely crucial for training deep networks with lots of parameters and prevent overfitting..

3. **Batch normalization:** A layer which normalizes inputs for each training batch

   https://arxiv.org/abs/1502.03167

   increases training speed/also acts as a regularizer

4. **GPU computations** – computations on (NVIDIA) graphics cards...

Going deep

Key advances allowing for training deep networks:

1. **ReLU activation:** (Rectified Linear Unit) i.e. nonlinearity for a neuron:

$$y_i = f(W_{ik}x_k + b_i) \qquad \underbrace{f(x) = max(0, x)}_{ReLU}$$

2. **Dropout:** during training we turn off say 50% randomly chosen neurons from the previous layer

   https://arxiv.org/abs/1207.0580

   turns out to be absolutely crucial for training deep networks with lots of parameters and prevent overfitting..

3. **Batch normalization:** A layer which normalizes inputs for each training batch

   https://arxiv.org/abs/1502.03167

   increases training speed/also acts as a regularizer

4. **GPU computations** – computations on (NVIDIA) graphics cards...

Going deep

Key advances allowing for training deep networks:

1. **ReLU activation:** (Rectified Linear Unit) i.e. nonlinearity for a neuron:

$$y_i = f(W_{ik} x_k + b_i) \qquad \underbrace{f(x) = max(0, x)}_{ReLU}$$

2. **Dropout:** during training we turn off say 50% randomly chosen neurons from the previous layer

   https://arxiv.org/abs/1207.0580

   turns out to be absolutely crucial for training deep networks with lots of parameters and prevent overfitting..

3. **Batch normalization:** A layer which normalizes inputs for each training batch

   https://arxiv.org/abs/1502.03167

   increases training speed/also acts as a regularizer

4. **GPU computations** – computations on (NVIDIA) graphics cards...

Going deep

Key advances allowing for training deep networks:

1. **ReLU activation:** (Rectified Linear Unit) i.e. nonlinearity for a neuron:

$$y_i = f(W_{ik}x_k + b_i) \qquad \underbrace{f(x) = max(0, x)}_{ReLU}$$

2. **Dropout:** during training we turn off say 50% randomly chosen neurons from the previous layer

   https://arxiv.org/abs/1207.0580

   turns out to be absolutely crucial for training deep networks with lots of parameters and prevent overfitting..

3. **Batch normalization:** A layer which normalizes inputs for each training batch

   https://arxiv.org/abs/1502.03167

   increases training speed/also acts as a regularizer

4. **GPU computations** – computations on (NVIDIA) graphics cards...

Going deep

Key advances allowing for training deep networks:

1. **ReLU activation:** (Rectified Linear Unit) i.e. nonlinearity for a neuron:

$$y_i = f(W_{ik}x_k + b_i) \qquad \underbrace{f(x) = max(0, x)}_{ReLU}$$

2. **Dropout:** during training we turn off say 50% randomly chosen neurons from the previous layer

   https://arxiv.org/abs/1207.0580

   turns out to be absolutely crucial for training deep networks with lots of parameters and prevent overfitting..

3. **Batch normalization:** A layer which normalizes inputs for each training batch

   https://arxiv.org/abs/1502.03167

   increases training speed/also acts as a regularizer

4. **GPU computations** – computations on (NVIDIA) graphics cards...

Going deep

Key advances allowing for training deep networks:

1. **ReLU activation:** (Rectified Linear Unit) i.e. nonlinearity for a neuron:

$$y_i = f(W_{ik} x_k + b_i) \qquad \underbrace{f(x) = max(0, x)}_{ReLU}$$

2. **Dropout:** during training we turn off say 50% randomly chosen neurons from the previous layer

   `https://arxiv.org/abs/1207.0580`

   turns out to be absolutely crucial for training deep networks with lots of parameters and prevent overfitting..

3. **Batch normalization:** A layer which normalizes inputs for each training batch

   `https://arxiv.org/abs/1502.03167`

   increases training speed/also acts as a regularizer

4. **GPU computations** – computations on (NVIDIA) graphics cards...

CNN example: MNIST

Example from the first journal club using keras:

```
model = Sequential()
model.add(Convolution2D(32, 3, 3,
                        border_mode=’valid’,
                        input_shape=(1, 28, 28)))
model.add(Activation(’relu’))
model.add(Convolution2D(32, 3, 3))
model.add(Activation(’relu’))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation(’relu’))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation(’softmax’))
```

---

reaches 99.0% accuracy (MNIST subset)
CPU computation time: **14min**
GPU computation time: **42s**

CNN example: MNIST

Example from the first journal club using `keras`:

```
model = Sequential()
model.add(Convolution2D(32, 3, 3,
                        border_mode=’valid’,
                        input_shape=(1, 28, 28)))
model.add(Activation(’relu’))
model.add(Convolution2D(32, 3, 3))
model.add(Activation(’relu’))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation(’relu’))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation(’softmax’))
```

---

reaches 99.0% accuracy (MNIST subset)

CPU computation time: **14min**

GPU computation time: **42s**

CNN example: MNIST

Example from the first journal club using `keras`:

```
model = Sequential()
model.add(Convolution2D(32, 3, 3,
                        border_mode=’valid’,
                        input_shape=(1, 28, 28)))
model.add(Activation(’relu’))
model.add(Convolution2D(32, 3, 3))
model.add(Activation(’relu’))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation(’relu’))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation(’softmax’))
```

---

reaches 99.0% accuracy (MNIST subset)
CPU computation time: **14min**
GPU computation time: **42s**

CNN example: MNIST

Example from the first journal club using keras:

```
model = Sequential()
model.add(Convolution2D(32, 3, 3,
                        border_mode=’valid’,
                        input_shape=(1, 28, 28)))
model.add(Activation(’relu’))
model.add(Convolution2D(32, 3, 3))
model.add(Activation(’relu’))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation(’relu’))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation(’softmax’))
```

---

reaches 99.0% accuracy (MNIST subset)
CPU computation time: **14min**
GPU computation time: **42s**

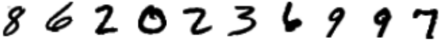
Image classification datasets – benchmarks for new CNN architectures

- **MNIST**: The *Hello World* of Machine Learning...
  60000+10000 *28x28* hand written digits

- **CIFAR-10** and **CIFAR-100**
  60000 32x32 colour images in 10 classes (resp. 100 classes)

Image classification datasets – benchmarks for new CNN architectures

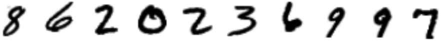
- **MNIST**: The *Hello World* of Machine Learning...
  60000+10000 28*x*28 hand written digits



- **CIFAR-10** and **CIFAR-100**
  60000 32x32 colour images in 10 classes (resp. 100 classes)

Image classification datasets – benchmarks for new CNN architectures

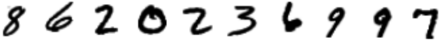
- **MNIST**: The *Hello World* of Machine Learning...
  60000+10000 28*x*28 hand written digits



- **CIFAR-10** and **CIFAR-100**
  60000 32x32 colour images in 10 classes (resp. 100 classes)

Image classification datasets – benchmarks for new CNN architectures

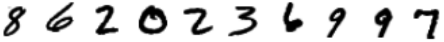
- **MNIST**: The *Hello World* of Machine Learning...
  60000+10000 28*x*28 hand written digits



- **CIFAR-10** and **CIFAR-100**
  60000 32x32 colour images in 10 classes (resp. 100 classes)

Image classification datasets – benchmarks for new CNN architectures

- **MNIST**: The *Hello World* of Machine Learning...
  60000+10000 28x28 hand written digits



- **CIFAR-10** and **CIFAR-100**
  60000 32x32 colour images in 10 classes (resp. 100 classes)

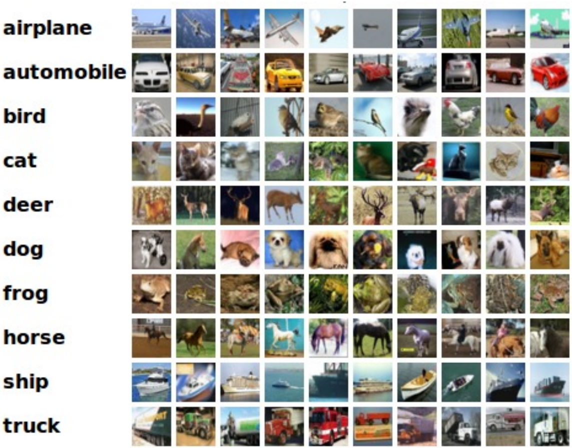


Image classification datasets – benchmarks for new CNN architectures

- **ImageNet**: state of the art...
  1.2 million images (256x256+) in 1000 classes
  t-SNE visualization http://stanford.io/2iEFn9L

Image classification datasets – benchmarks for new CNN architectures

- **ImageNet**: state of the art...

  1.2 million images (256x256+) in 1000 classes

  t-SNE visualization http://stanford.io/2iEFn9L

Image classification datasets – benchmarks for new CNN architectures

- **ImageNet**: state of the art...
  1.2 million images (256x256+) in 1000 classes
  t-SNE visualization http://stanford.io/2iEFn9L

Image classification datasets – benchmarks for new CNN architectures

- **ImageNet**: state of the art...
  1.2 million images (256x256+) in 1000 classes
  t-SNE visualization http://stanford.io/2iEFn9L

Data augmentation

- In order to train deep neural networks one has to have lots of data
- This is necessary to
  1. avoid overfitting
  2. tell the network what are the inessential features (which are intuitive for us)
- For this reason one employs **data augmentation**
  — construct additional training examples by flipping/rotating/ choosing different crops etc.

Data augmentation

- In order to train deep neural networks one has to have lots of data
- This is necessary to
  1. avoid overfitting
  2. tell the network what are the inessential features (which are intuitive for us)
- For this reason one employs **data augmentation** — construct additional training examples by flipping/rotating/ choosing different crops etc.

Data augmentation

- In order to train deep neural networks one has to have lots of data
- This is necessary to
  1. avoid overfitting
  2. tell the network what are the inessential features (which are intuitive for us)
- For this reason one employs **data augmentation**
  — construct additional training examples by flipping/rotating/ choosing different crops etc.

Data augmentation

- In order to train deep neural networks one has to have lots of data
- This is necessary to
  1. avoid overfitting
  2. tell the network what are the inessential features (which are intuitive for us)
- For this reason one employs **data augmentation** — construct additional training examples by flipping/rotating/ choosing different crops etc.

Data augmentation

- In order to train deep neural networks one has to have lots of data
- This is necessary to
  1. avoid overfitting
  2. tell the network what are the inessential features (which are intuitive for us)
- For this reason one employs **data augmentation**

  — construct additional training examples by flipping/rotating/ choosing different crops etc.

Data augmentation

- In order to train deep neural networks one has to have lots of data
- This is necessary to
  1. avoid overfitting
  2. tell the network what are the inessential features (which are intuitive for us)
- For this reason one employs **data augmentation**
  — construct additional training examples by flipping/rotating/ choosing different crops etc.

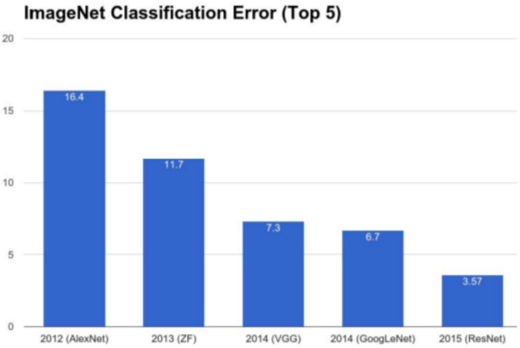
Image classification datasets – benchmarks for new CNN architectures

Some key results on ImageNet:

- AlexNet result in 2012 was a breakthrough — 2^{nd} result: 26.2%
- The progress was **not** only due to brute force!
  AlexNet $\sim 60M$ parameters, VGG $\sim 180M$ but GoogLeNet only $5M$
  residual network also has a new architectural ingredient
- Human perfomance: 5.1% error (see `http://bit.ly/2f45u9i`)
- Top-1 errors are much larger ($17\% - 20\%+$)
- 2016: GoogLeNet Inception V4 (3.08%)

Image classification datasets – benchmarks for new CNN architectures

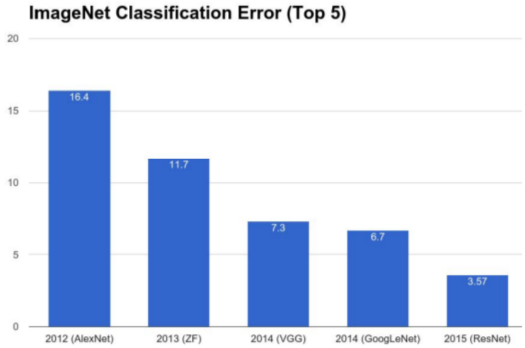
Some key results on ImageNet:



- AlexNet result in 2012 was a breakthrough — 2^{nd} result: 26.2%
- The progress was **not** only due to brute force!
  AlexNet $\sim 60M$ parameters, VGG $\sim 180M$ but GoogLeNet only $5M$
  residual network also has a new architectural ingredient
- Human perfomance: 5.1% error (see `http://bit.ly/2f45u9i`)
- Top-1 errors are much larger ($17\% - 20\%+$)
- 2016: GoogLeNet Inception V4 (3.08%)

Image classification datasets – benchmarks for new CNN architectures

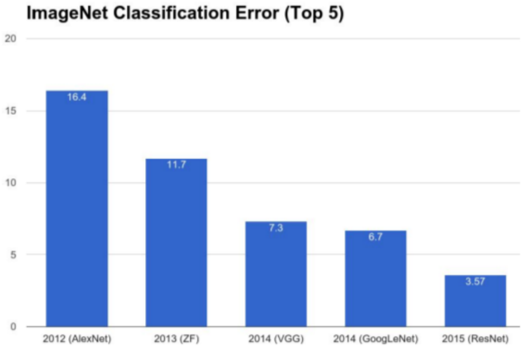
Some key results on ImageNet:



- AlexNet result in 2012 was a breakthrough — 2^{nd} result: 26.2%
- The progress was **not** only due to brute force!
  AlexNet $\sim 60M$ parameters, VGG $\sim 180M$ but GoogLeNet only $5M$
  residual network also has a new architectural ingredient
- Human perfomance: 5.1% error (see `http://bit.ly/2f45u9i`)
- Top-1 errors are much larger ($17\% - 20\%+$)
- 2016: GoogLeNet Inception V4 (3.08%)

Image classification datasets – benchmarks for new CNN architectures

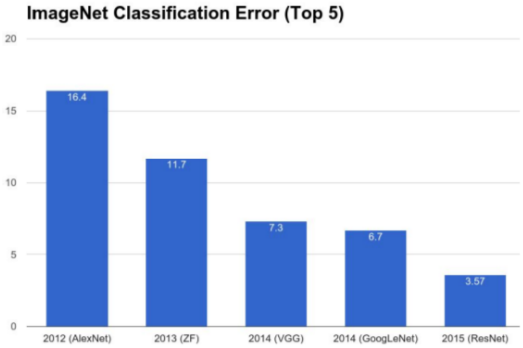
Some key results on ImageNet:



- AlexNet result in 2012 was a breakthrough — 2^{nd} result: 26.2%
- The progress was **not** only due to brute force!
  AlexNet $\sim 60M$ parameters, VGG $\sim 180M$ but GoogLeNet only $5M$
  residual network also has a new architectural ingredient
- Human perfomance: 5.1% error (see `http://bit.ly/2f45u9i`)
- Top-1 errors are much larger ($17\% - 20\%+$)
- 2016: GoogLeNet Inception V4 (3.08%)

Image classification datasets – benchmarks for new CNN architectures

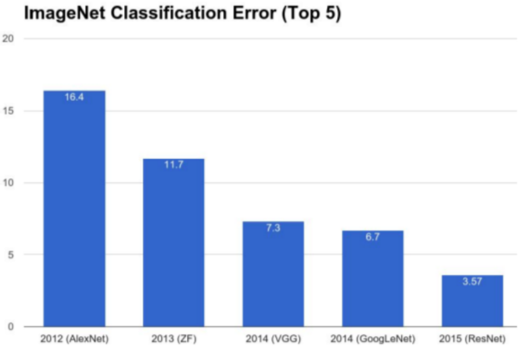
Some key results on ImageNet:



- AlexNet result in 2012 was a breakthrough — 2^{nd} result: 26.2%
- The progress was **not** only due to brute force!
  AlexNet $\sim 60M$ parameters, VGG $\sim 180M$ but GoogLeNet only $5M$
  residual network also has a new architectural ingredient
- Human perfomance: 5.1% error (see `http://bit.ly/2f45u9i`)
- Top-1 errors are much larger ($17\% - 20\%+$)
- 2016: GoogLeNet Inception V4 (3.08%)

Image classification datasets – benchmarks for new CNN architectures

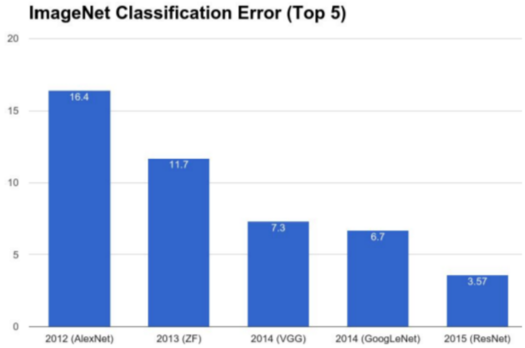
Some key results on ImageNet:



- AlexNet result in 2012 was a breakthrough — 2^{nd} result: 26.2%
- The progress was **not** only due to brute force!
  AlexNet $\sim 60M$ parameters, VGG $\sim 180M$ but GoogLeNet only $5M$
  residual network also has a new architectural ingredient
- Human perfomance: 5.1% error (see `http://bit.ly/2f45u9i`)
- Top-1 errors are much larger (17% – 20%+)
- 2016: GoogLeNet Inception V4 (3.08%)

Image classification datasets – benchmarks for new CNN architectures

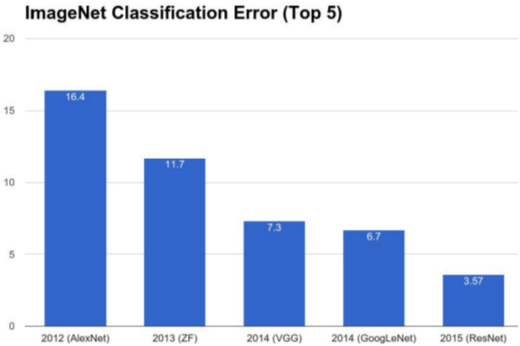
Some key results on ImageNet:



- AlexNet result in 2012 was a breakthrough — 2^{nd} result: 26.2%
- The progress was **not** only due to brute force!
  AlexNet $\sim 60M$ parameters, VGG $\sim 180M$ but GoogLeNet only $5M$
  residual network also has a new architectural ingredient
- Human perfomance: 5.1% error (see `http://bit.ly/2f45u9i`)
- Top-1 errors are much larger ($17\% - 20\%+$)
- 2016: GoogLeNet Inception V4 (3.08%)

Image classification datasets – benchmarks for new CNN architectures

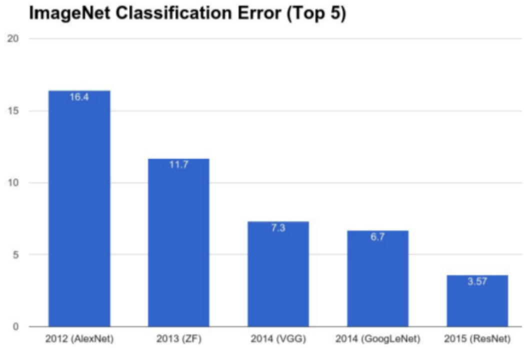
Some key results on ImageNet:



- AlexNet result in 2012 was a breakthrough — 2^{nd} result: 26.2%
- The progress was **not** only due to brute force!
  AlexNet $\sim 60M$ parameters, VGG $\sim 180M$ but GoogLeNet only $5M$
  residual network also has a new architectural ingredient
- Human perfomance: 5.1% error (see `http://bit.ly/2f45u9i`)
- Top-1 errors are much larger ($17\% - 20\%+$)
- 2016: GoogLeNet Inception V4 (3.08%)

Image classification datasets – benchmarks for new CNN architectures

Some key results on ImageNet:



- AlexNet result in 2012 was a breakthrough — 2^{nd} result: 26.2%
- The progress was **not** only due to brute force!
  AlexNet $\sim 60M$ parameters, VGG $\sim 180M$ but GoogLeNet only $5M$
  residual network also has a new architectural ingredient
- Human perfomance: 5.1% error (see `http://bit.ly/2f45u9i`)
- Top-1 errors are much larger ($17\% - 20\%+$)
- 2016: GoogLeNet Inception V4 (3.08%)

Recent CNN architectures

GoogLeNet – Inception

Google

https://arxiv.org/abs/1409.4842

- Classical CNN's are a sequence of convolution and pooling layers
- Two ideas:
  1. Allow filters of different sizes and concatenate
  2. Each convolution takes as input $n_{channels} \times N_x \times N_y$.
     First perform pointwise convolution over channels (1×1 filter) and then spatial convolution

- This significantly reduces the number of parameters..
- GoogLeNet stacks multiple Inception modules: 22 layer deep

GoogLeNet – Inception

Google

https://arxiv.org/abs/1409.4842

- Classical CNN's are a sequence of convolution and pooling layers
- Two ideas:
  1. Allow filters of different sizes and concatenate
  2. Each convolution takes as input $n_{channels} \times N_x \times N_y$.
     First perform pointwise convolution over channels (1×1 filter) and then spatial convolution
- This significantly reduces the number of parameters..
- GoogLeNet stacks multiple Inception modules: 22 layer deep

https://arxiv.org/abs/1409.4842

- Classical CNN's are a sequence of convolution and pooling layers
- Two ideas:
  1. Allow filters of different sizes and concatenate
  2. Each convolution takes as input $n_{channels} \times N_x \times N_y$.
     First perform pointwise convolution over channels (1×1 filter) and then spatial convolution

- This significantly reduces the number of parameters..
- GoogLeNet stacks multiple Inception modules: 22 layer deep

GoogLeNet – Inception

Google

https://arxiv.org/abs/1409.4842

- Classical CNN's are a sequence of convolution and pooling layers
- Two ideas:
  1. Allow filters of different sizes and concatenate
  2. Each convolution takes as input $n_{channels} \times N_x \times N_y$.
     First perform pointwise convolution over channels (1×1 filter) and then spatial convolution

- This significantly reduces the number of parameters..
- GoogLeNet stacks multiple Inception modules: 22 layer deep

GoogLeNet – Inception

https://arxiv.org/abs/1409.4842

- Classical CNN's are a sequence of convolution and pooling layers
- Two ideas:
  1. Allow filters of different sizes and concatenate
  2. Each convolution takes as input $n_{channels} \times N_x \times N_y$.
     First perform pointwise convolution over channels (1×1 filter) and then spatial convolution

- This significantly reduces the number of parameters..
- GoogLeNet stacks multiple Inception modules: 22 layer deep

GoogLeNet – Inception

https://arxiv.org/abs/1409.4842

- Classical CNN's are a sequence of convolution and pooling layers
- Two ideas:
  1. Allow filters of different sizes and concatenate
  2. Each convolution takes as input $n_{channels} \times N_x \times N_y$.
     First perform pointwise convolution over channels (1×1 filter) and then spatial convolution

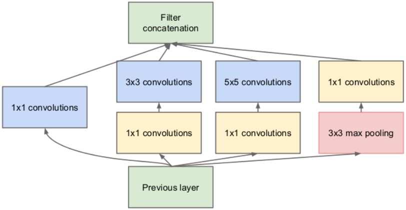
- This significantly reduces the number of parameters..
- GoogLeNet stacks multiple Inception modules: 22 layer deep

GoogLeNet – Inception

Google

https://arxiv.org/abs/1409.4842

- Classical CNN's are a sequence of convolution and pooling layers
- Two ideas:
  1. Allow filters of different sizes and concatenate
  2. Each convolution takes as input $n_{channels} \times N_x \times N_y$.
     First perform pointwise convolution over channels (1×1 filter) and then spatial convolution

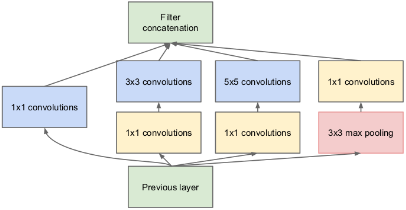


- This significantly reduces the number of parameters..
- GoogLeNet stacks multiple Inception modules: 22 layer deep

GoogLeNet – Inception

https://arxiv.org/abs/1409.4842

- Classical CNN's are a sequence of convolution and pooling layers
- Two ideas:
  1. Allow filters of different sizes and concatenate
  2. Each convolution takes as input $n_{channels} \times N_x \times N_y$.
     First perform pointwise convolution over channels (1×1 filter) and then spatial convolution



- This significantly reduces the number of parameters..
- GoogLeNet stacks multiple Inception modules: 22 layer deep

Xception

Google

F. Chollet https://arxiv.org/abs/1610.02357

- One can take factorization to an extreme...
- Factor all convolutions into a pointwise and spatial convolution
- This implemented in TensorFlow/Keras as first performing spatial convolution for each channel separately and then projecting pointwise by a 1×1 convolution
- Works better than Inception V3 with same number of parameters
- Use also an internal Google dataset: JFT: 350 million high resolution images annotated with 17000 classes (mutli-label)
- Models trained on **60** NVIDIA GPU's

F. Chollet https://arxiv.org/abs/1610.02357

- One can take factorization to an extreme...
- Factor all convolutions into a pointwise and spatial convolution
- This implemented in TensorFlow/Keras as first performing spatial convolution for each channel separately and then projecting pointwise by a 1×1 convolution
- Works better than Inception V3 with same number of parameters
- Use also an internal Google dataset: JFT: 350 million high resolution images annotated with 17000 classes (mutli-label)
- Models trained on **60** NVIDIA GPU's

Xception

Google

F. Chollet https://arxiv.org/abs/1610.02357

- One can take factorization to an extreme...
- Factor all convolutions into a pointwise and spatial convolution
- This implemented in TensorFlow/Keras as first performing spatial convolution for each channel separately and then projecting pointwise by a 1×1 convolution
- Works better than Inception V3 with same number of parameters
- Use also an internal Google dataset: JFT: 350 million high resolution images annotated with 17000 classes (mutli-label)
- Models trained on **60** NVIDIA GPU's

F. Chollet https://arxiv.org/abs/1610.02357

- One can take factorization to an extreme...
- Factor all convolutions into a pointwise and spatial convolution
- This implemented in TensorFlow/Keras as first performing spatial convolution for each channel separately and then projecting pointwise by a 1×1 convolution
- Works better than Inception V3 with same number of parameters
- Use also an internal Google dataset: JFT: 350 million high resolution images annotated with 17000 classes (mutli-label)
- Models trained on **60** NVIDIA GPU's

Xception

Google

F. Chollet https://arxiv.org/abs/1610.02357

- One can take factorization to an extreme...
- Factor all convolutions into a pointwise and spatial convolution
- This implemented in TensorFlow/Keras as first performing spatial convolution for each channel separately and then projecting pointwise by a 1×1 convolution
- Works better than Inception V3 with same number of parameters
- Use also an internal Google dataset: JFT: 350 million high resolution images annotated with 17000 classes (mutli-label)
- Models trained on **60** NVIDIA GPU's

Xception Google

F. Chollet https://arxiv.org/abs/1610.02357

- One can take factorization to an extreme...
- Factor all convolutions into a pointwise and spatial convolution
- This implemented in TensorFlow/Keras as first performing spatial convolution for each channel separately and then projecting pointwise by a 1×1 convolution
- Works better than Inception V3 with same number of parameters
- Use also an internal Google dataset: JFT: 350 million high resolution images annotated with 17000 classes (mutli-label)
- Models trained on **60** NVIDIA GPU's

Xception

Google

F. Chollet https://arxiv.org/abs/1610.02357

- One can take factorization to an extreme...
- Factor all convolutions into a pointwise and spatial convolution
- This implemented in TensorFlow/Keras as first performing spatial convolution for each channel separately and then projecting pointwise by a 1×1 convolution
- Works better than Inception V3 with same number of parameters
- Use also an internal Google dataset: JFT: 350 million high resolution images annotated with 17000 classes (mutli-label)
- Models trained on **60** NVIDIA GPU's

Residual networks Microsoft

https://arxiv.org/abs/1512.03385
https://arxiv.org/abs/1603.05027

- In what way are *very* deep networks difficult to train?
  Not due to overfitting! – have difficulty learning the training set

- But in principle one could find a trivial 56-layer network which is as good as a 20 layer one — just take a 20 layer one an add 36 layers which realize an identity mapping..
- **Conclusion:** in conventional CNN's it is difficult to make deep layers learn the identity mapping!

Residual networks

Microsoft

https://arxiv.org/abs/1512.03385

https://arxiv.org/abs/1603.05027

- In what way are *very* deep networks difficult to train?
  Not due to overfitting! – have difficulty learning the training set

- But in principle one could find a trivial 56-layer network which is as good as a 20 layer one — just take a 20 layer one an add 36 layers which realize an identity mapping..
- **Conclusion:** in conventional CNN's it is difficult to make deep layers learn the identity mapping!

https://arxiv.org/abs/1512.03385
https://arxiv.org/abs/1603.05027

- In what way are *very* deep networks difficult to train?
  Not due to overfitting! – have difficulty learning the training set
- But in principle one could find a trivial 56-layer network which is as good as a 20 layer one — just take a 20 layer one an add 36 layers which realize an identity mapping..
- **Conclusion:** in conventional CNN's it is difficult to make deep layers learn the identity mapping!

Residual networks

Microsoft

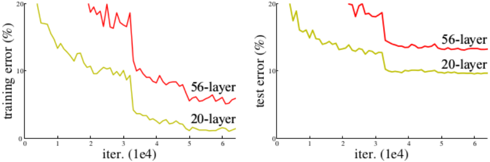
https://arxiv.org/abs/1512.03385
https://arxiv.org/abs/1603.05027

- In what way are *very* deep networks difficult to train?
  Not due to overfitting! – have difficulty learning the training set

- But in principle one could find a trivial 56-layer network which is as good as a 20 layer one — just take a 20 layer one an add 36 layers which realize an identity mapping..
- **Conclusion:** in conventional CNN's it is difficult to make deep layers learn the identity mapping!

https://arxiv.org/abs/1512.03385
https://arxiv.org/abs/1603.05027

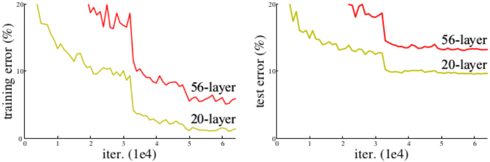
- In what way are *very* deep networks difficult to train?
  Not due to overfitting! – have difficulty learning the training set



- But in principle one could find a trivial 56-layer network which is as good as a 20 layer one — just take a 20 layer one an add 36 layers which realize an identity mapping..
- **Conclusion:** in conventional CNN's it is difficult to make deep layers learn the identity mapping!

https://arxiv.org/abs/1512.03385
https://arxiv.org/abs/1603.05027

- In what way are *very* deep networks difficult to train?
  Not due to overfitting! – have difficulty learning the training set

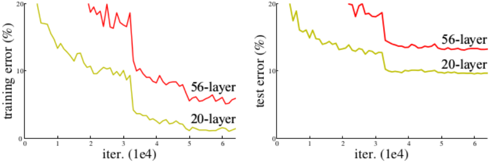


- But in principle one could find a trivial 56-layer network which is as good as a 20 layer one — just take a 20 layer one an add 36 layers which realize an identity mapping..
- **Conclusion:** in conventional CNN's it is difficult to make deep layers learn the identity mapping!

Residual networks

https://arxiv.org/abs/1512.03385
https://arxiv.org/abs/1603.05027

- In what way are *very* deep networks difficult to train?
  Not due to overfitting! – have difficulty learning the training set

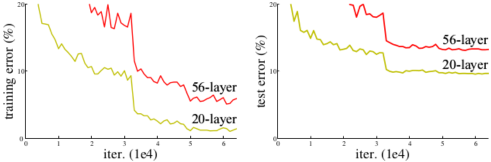


- But in principle one could find a trivial 56-layer network which is as good as a 20 layer one — just take a 20 layer one an add 36 layers which realize an identity mapping..
- **Conclusion:** in conventional CNN's it is difficult to make deep layers learn the identity mapping!

Residual networks

https://arxiv.org/abs/1512.03385
https://arxiv.org/abs/1603.05027

- In what way are *very* deep networks difficult to train?
  Not due to overfitting! – have difficulty learning the training set



- But in principle one could find a trivial 56-layer network which is as good as a 20 layer one — just take a 20 layer one an add 36 layers which realize an identity mapping..
- **Conclusion:** in conventional CNN's it is difficult to make deep layers learn the identity mapping!

Residual networks Microsoft

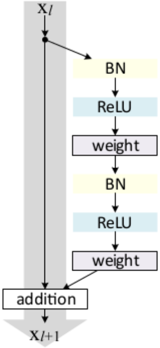
Proposal: Learn deviations from identity rather than the full mapping...

- Stacking such residual units gives CNN's with many layers: 200 on ImageNet and even 1001 on CIFAR!
- Various variations of this idea have been tested in the second paper (this is the latest version)
- Residual networks have become the state-of-the-art
- The residual network ideas are being incorporated into other architectures (like Inception, Fully Convolutional Networks (FCN) for segmentation etc.)

Proposal: Learn deviations from identity rather than the full mapping...

- Stacking such residual units gives CNN's with many layers: 200 on ImageNet and even 1001 on CIFAR!
- Various variations of this idea have been tested in the second paper (this is the latest version)
- Residual networks have become the state-of-the-art
- The residual network ideas are being incorporated into other architectures (like Inception, Fully Convolutional Networks (FCN) for segmentation etc.)

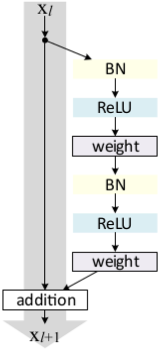
Proposal: Learn deviations from identity rather than the full mapping...



- Stacking such residual units gives CNN's with many layers: 200 on ImageNet and even 1001 on CIFAR!
- Various variations of this idea have been tested in the second paper (this is the latest version)
- Residual networks have become the state-of-the-art
- The residual network ideas are being incorporated into other architectures (like Inception, Fully Convolutional Networks (FCN) for segmentation etc.)

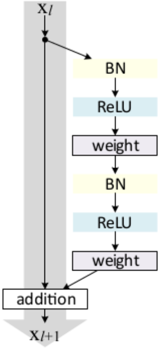
Residual networks

Proposal: Learn deviations from identity rather than the full mapping...



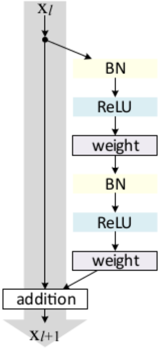
- Stacking such residual units gives CNN's with many layers: 200 on ImageNet and even 1001 on CIFAR!
- Various variations of this idea have been tested in the second paper (this is the latest version)
- Residual networks have become the state-of-the-art
- The residual network ideas are being incorporated into other architectures (like Inception, Fully Convolutional Networks (FCN) for segmentation etc.)

Proposal: Learn deviations from identity rather than the full mapping...



- Stacking such residual units gives CNN's with many layers: 200 on ImageNet and even 1001 on CIFAR!
- Various variations of this idea have been tested in the second paper (this is the latest version)
- Residual networks have become the state-of-the-art
- The residual network ideas are being incorporated into other architectures (like Inception, Fully Convolutional Networks (FCN) for segmentation etc.)

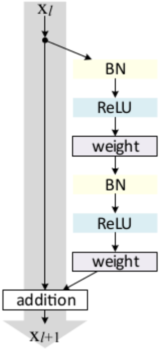
Proposal: Learn deviations from identity rather than the full mapping...



- Stacking such residual units gives CNN's with many layers: 200 on ImageNet and even 1001 on CIFAR!
- Various variations of this idea have been tested in the second paper (this is the latest version)
- Residual networks have become the state-of-the-art
- The residual network ideas are being incorporated into other architectures (like Inception, Fully Convolutional Networks (FCN) for segmentation etc.)

Residual networks

Proposal: Learn deviations from identity rather than the full mapping...



- Stacking such residual units gives CNN's with many layers: 200 on ImageNet and even 1001 on CIFAR!
- Various variations of this idea have been tested in the second paper (this is the latest version)
- Residual networks have become the state-of-the-art
- The residual network ideas are being incorporated into other architectures (like Inception, Fully Convolutional Networks (FCN) for segmentation etc.)

Densely connected convolutional networks

Facebook+Cornell+Tsinghua U.

https://arxiv.org/abs/1608.06993

- Residual networks (and earlier more complicated *Highway networks* + other ideas: *stochastic depth*, *FractalNets*) provide multiple paths for data to flow through the networks
- Densely connected convolutional networks (DensNet) exploit this idea

- The outputs of the layers in each block are concatenated (and not summed)
- In the transition layers the authors reduce the number of features while in the convolution layers introduce 1×1 convolutions (bottleneck layers)

Densely connected convolutional networks

Facebook+Cornell+Tsinghua U.

https://arxiv.org/abs/1608.06993

- Residual networks (and earlier more complicated *Highway networks* + other ideas: *stochastic depth*, *FractalNets*) provide multiple paths for data to flow through the networks
- Densely connected convolutional networks (DensNet) exploit this idea
- The outputs of the layers in each block are concatenated (and not summed)
- In the transition layers the authors reduce the number of features while in the convolution layers introduce 1×1 convolutions (bottleneck layers)

Densely connected convolutional networks

Facebook+Cornell+Tsinghua U.

https://arxiv.org/abs/1608.06993

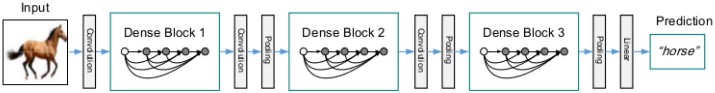
- Residual networks (and earlier more complicated *Highway networks* + other ideas: *stochastic depth*, *FractalNets*) provide multiple paths for data to flow through the networks
- Densely connected convolutional networks (DensNet) exploit this idea

- The outputs of the layers in each block are concatenated (and not summed)
- In the transition layers the authors reduce the number of features while in the convolution layers introduce 1×1 convolutions (bottleneck layers)

Densely connected convolutional networks

https://arxiv.org/abs/1608.06993

- Residual networks (and earlier more complicated *Highway networks* + other ideas: *stochastic depth*, *FractalNets*) provide multiple paths for data to flow through the networks
- Densely connected convolutional networks (DensNet) exploit this idea

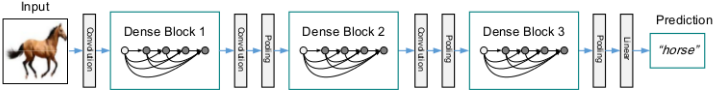


- The outputs of the layers in each block are concatenated (and not summed)
- In the transition layers the authors reduce the number of features while in the convolution layers introduce 1×1 convolutions (bottleneck layers)

Densely connected convolutional networks

https://arxiv.org/abs/1608.06993

- Residual networks (and earlier more complicated *Highway networks* + other ideas: *stochastic depth*, *FractalNets*) provide multiple paths for data to flow through the networks
- Densely connected convolutional networks (DensNet) exploit this idea



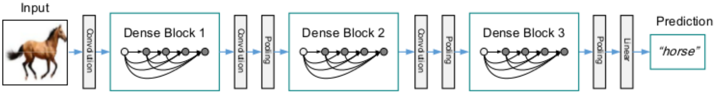
- The outputs of the layers in each block are concatenated (and not summed)
- In the transition layers the authors reduce the number of features while in the convolution layers introduce 1×1 convolutions (bottleneck layers)

Densely connected convolutional networks

Facebook+Cornell+Tsinghua U.

https://arxiv.org/abs/1608.06993

- Residual networks (and earlier more complicated *Highway networks* + other ideas: *stochastic depth*, *FractalNets*) provide multiple paths for data to flow through the networks
- Densely connected convolutional networks (DensNet) exploit this idea



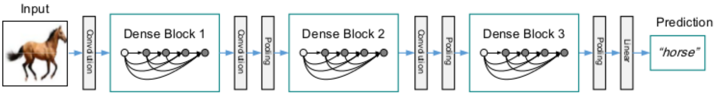
- The outputs of the layers in each block are concatenated (and not summed)
- In the transition layers the authors reduce the number of features while in the convolution layers introduce 1×1 convolutions (bottleneck layers)

Densely connected convolutional networks

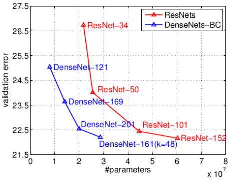
Facebook+Cornell+Tsinghua U.

https://arxiv.org/abs/1608.06993

- Residual networks (and earlier more complicated *Highway networks* + other ideas: *stochastic depth*, *FractalNets*) provide multiple paths for data to flow through the networks
- Densely connected convolutional networks (DensNet) exploit this idea



- The outputs of the layers in each block are concatenated (and not summed)
- In the transition layers the authors reduce the number of features while in the convolution layers introduce 1×1 convolutions (bottleneck layers)



Top-1 error on ImageNet

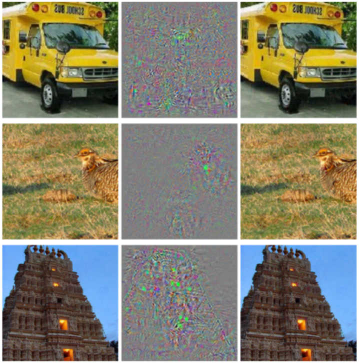
Interlude

Intriguing properties of neural networks

https://arxiv.org/abs/1312.6199

- On the left: images which correctly classified by AlexNet (deep CNN trained on ImageNet)
- Center: perturbations magnified $10\times$
- On the right: images classified by the same net as **ostrich!!!**

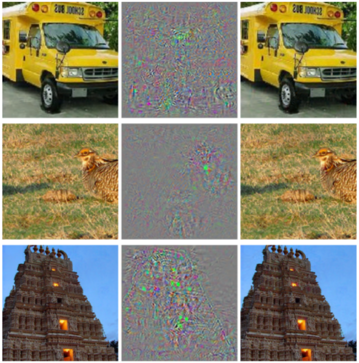
Intriguing properties of neural networks



https://arxiv.org/abs/1312.6199

- On the left: images which correctly classified by AlexNet (deep CNN trained on ImageNet)
- Center: perturbations magnified $10\times$
- On the right: images classified by the same net as **ostrich!!!**

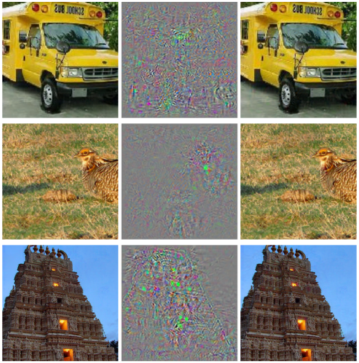
Intriguing properties of neural networks



https://arxiv.org/abs/1312.6199

- On the left: images which correctly classified by AlexNet (deep CNN trained on ImageNet)
- Center: perturbations magnified 10×
- On the right: images classified by the same net as **ostrich!!!**

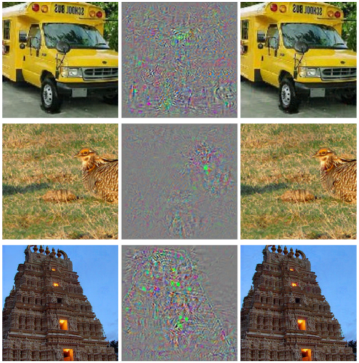
Intriguing properties of neural networks



https://arxiv.org/abs/1312.6199

- On the left: images which correctly classified by AlexNet (deep CNN trained on ImageNet)
- Center: perturbations magnified 10×
- On the right: images classified by the same net as **ostrich!!!**

Intriguing properties of neural networks



https://arxiv.org/abs/1312.6199

- On the left: images which correctly classified by AlexNet (deep CNN trained on ImageNet)
- Center: perturbations magnified 10×
- On the right: images classified by the same net as **ostrich!!!**

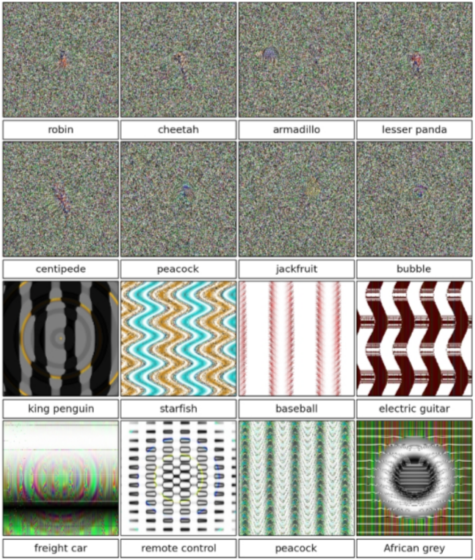
Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images

https://arxiv.org/abs/1412.1897

- Images which are classified by AlexNet (deep CNN trained on ImageNet) with more than 99.6% confidence
- These examples were constructed by a kind of evolutionary procedure...

Our results shed light on interesting differences between human vision and current DNNs, and raise questions about the generality of DNN computer vision.

Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images

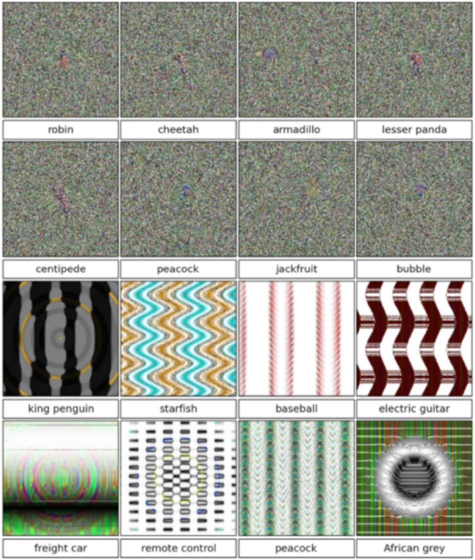


https://arxiv.org/abs/1412.1897

- Images which are classified by AlexNet (deep CNN trained on ImageNet) with more than 99.6% confidence
- These examples were constructed by a kind of evolutionary procedure...

Our results shed light on interesting differences between human vision and current DNNs, and raise questions about the generality of DNN computer vision.

Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images

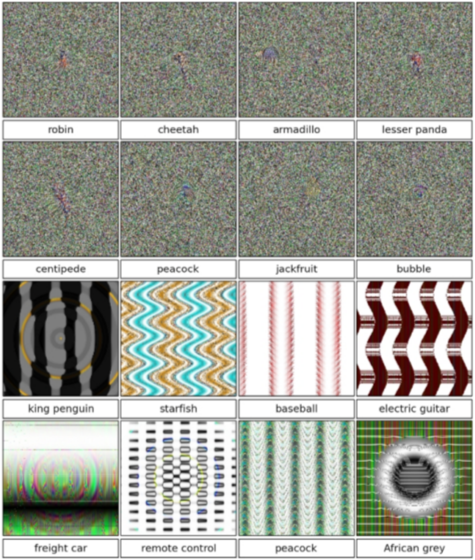


https://arxiv.org/abs/1412.1897

- Images which are classified by AlexNet (deep CNN trained on ImageNet) with more than 99.6% confidence
- These examples were constructed by a kind of evolutionary procedure...

Our results shed light on interesting differences between human vision and current DNNs, and raise questions about the generality of DNN computer vision.

Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images



https://arxiv.org/abs/1412.1897

- Images which are classified by AlexNet (deep CNN trained on ImageNet) with more than 99.6% confidence
- These examples were constructed by a kind of evolutionary procedure...

Our results shed light on interesting differences between human vision and current DNNs, and raise questions about the generality of DNN computer vision.

Goodfellow, Shlens, Szegedy Explaining and Harnessing Adversarial Examples https://arxiv.org/abs/1412.6572

- Adversarial examples are due to *linear* nature (and high dimensionality) of the NN
- Not restricted to deep CNN's...

Goodfellow, Shlens, Szegedy Explaining and Harnessing Adversarial Examples
https://arxiv.org/abs/1412.6572

- Adversarial examples are due to *linear* nature (and high dimensionality) of the NN
- Not restricted to deep CNN's...

Goodfellow, Shlens, Szegedy Explaining and Harnessing Adversarial Examples
https://arxiv.org/abs/1412.6572

- Adversarial examples are due to *linear* nature (and high dimensionality) of the NN
- Not restricted to deep CNN's...

Goodfellow, Shlens, Szegedy Explaining and Harnessing Adversarial Examples
https://arxiv.org/abs/1412.6572

- Adversarial examples are due to *linear* nature (and high dimensionality) of the NN
- Not restricted to deep CNN's...

Image segmentation

Image segmentation task:

Not only identify the class(es) of the objects in the image but locate them pixelwise

Very good set of lectures @Stanford '16 (slides and notes online!):
CS321n: Convolutional Neural Networks for Visual Recognition
http://cs231n.stanford.edu/syllabus.html

Image segmentation

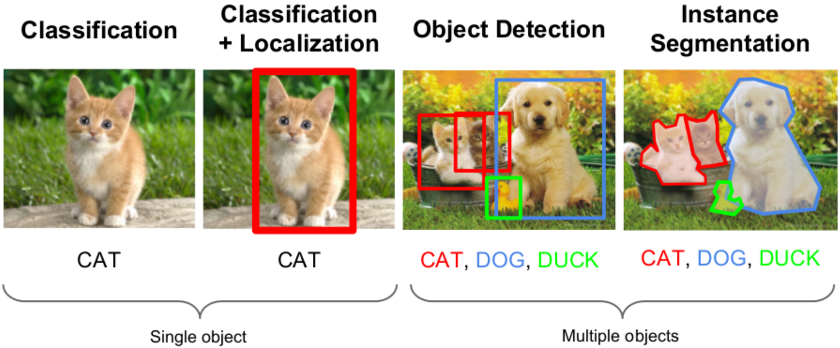


Image segmentation task:

Not only identify the class(es) of the objects in the image but locate them pixelwise

Very good set of lectures @Stanford '16 (slides and notes online!):
CS321n: Convolutional Neural Networks for Visual Recognition

http://cs231n.stanford.edu/syllabus.html

Image segmentation

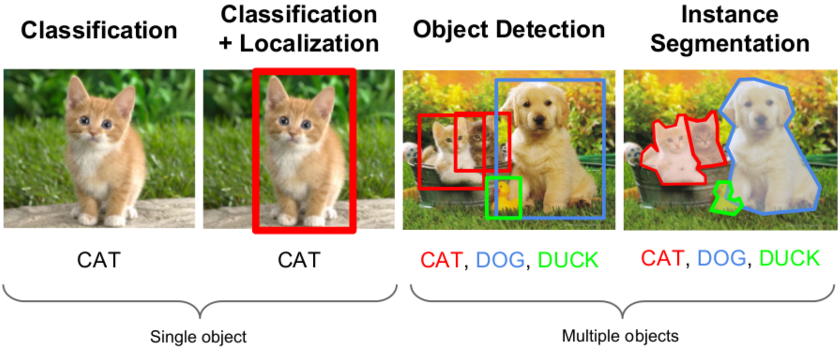


Image segmentation task:

Not only identify the class(es) of the objects in the image but locate them pixelwise

Very good set of lectures @Stanford '16 (slides and notes online!):
CS321n: Convolutional Neural Networks for Visual Recognition
http://cs231n.stanford.edu/syllabus.html

Image segmentation

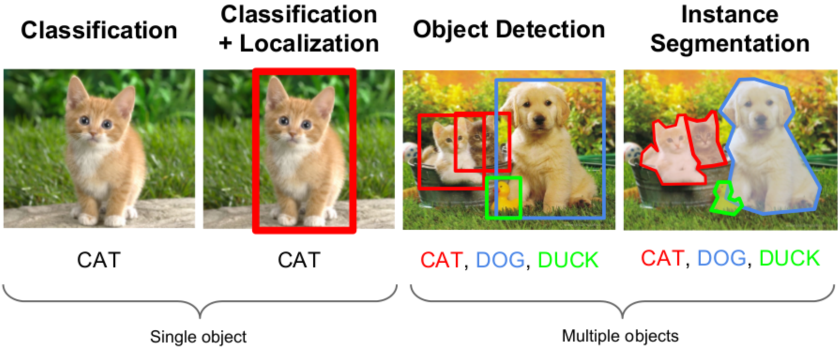


Image segmentation task:

Not only identify the class(es) of the objects in the image but locate them pixelwise

Very good set of lectures @Stanford '16 (slides and notes online!):
CS321n: Convolutional Neural Networks for Visual Recognition
http://cs231n.stanford.edu/syllabus.html

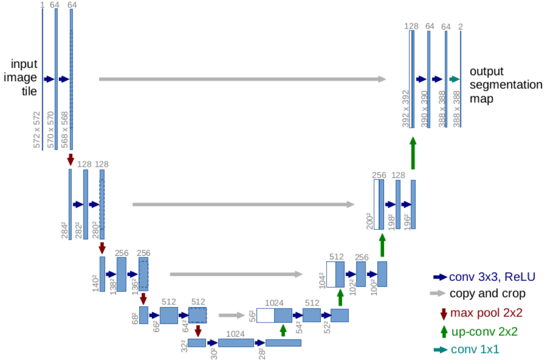
Fully Convolutional Networks — U-Net

https://arxiv.org/abs/1505.04597

- Network used for biomedical image segmentation... (but generic segmentation architecture)
- Lots of papers, see e.g. https://arxiv.org/abs/1611.09326 Y. Bengio et.al. *Fully convolutional DenseNets for semantic segmentation*

Fully Convolutional Networks — U-Net

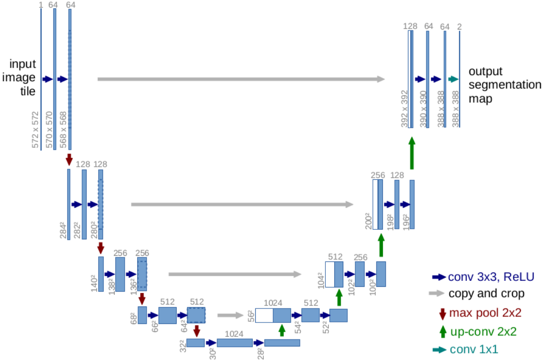
https://arxiv.org/abs/1505.04597



- Network used for biomedical image segmentation... (but generic segmentation architecture)
- Lots of papers, see e.g. https://arxiv.org/abs/1611.09326 Y. Bengio et.al. *Fully convolutional DenseNets for semantic segmentation*

Fully Convolutional Networks — U-Net

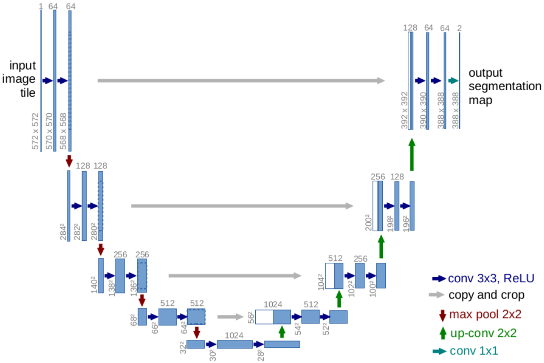
https://arxiv.org/abs/1505.04597



- Network used for biomedical image segmentation... (but generic segmentation architecture)
- Lots of papers, see e.g. https://arxiv.org/abs/1611.09326 Y. Bengio et.al. *Fully convolutional DenseNets for semantic segmentation*

Fully Convolutional Networks — U-Net

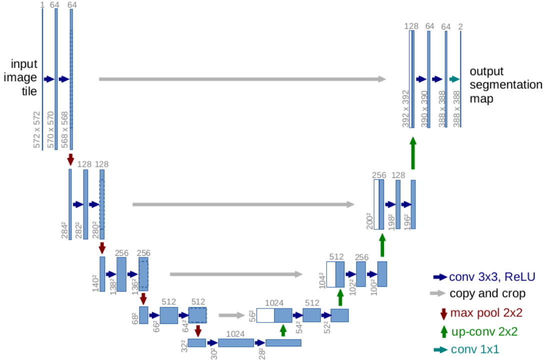
https://arxiv.org/abs/1505.04597



- Network used for biomedical image segmentation... (but generic segmentation architecture)
- Lots of papers, see e.g. https://arxiv.org/abs/1611.09326 Y. Bengio et.al. *Fully convolutional DenseNets for semantic segmentation*

Fully Convolutional Networks — U-Net

https://arxiv.org/abs/1505.04597



- Network used for biomedical image segmentation... (but generic segmentation architecture)
- Lots of papers, see e.g. https://arxiv.org/abs/1611.09326 Y. Bengio et.al. *Fully convolutional DenseNets for semantic segmentation*

Conclusions

- Convolutional networks are used primarily for images but can be also used for time-series data (1D convolutions) and video (3D convolution)
- Apart from classification, CNNs are also used e.g. for image segmentation
- GPU's are currently absolutely crucial for computation...
- Convolutional networks are still being developed and improved...
  ... not only brute force but also novel architectures
- It seems that having multiple possibilities of traversing the network is very beneficial

Conclusions

- Convolutional networks are used primarily for images but can be also used for time-series data (1D convolutions) and video (3D convolution)
- Apart from classification, CNNs are also used e.g. for image segmentation
- GPU's are currently absolutely crucial for computation...
- Convolutional networks are still being developed and improved...
  ... not only brute force but also novel architectures
- It seems that having multiple possibilities of traversing the network is very beneficial

Conclusions

- Convolutional networks are used primarily for images but can be also used for time-series data (1D convolutions) and video (3D convolution)
- Apart from classification, CNNs are also used e.g. for image segmentation
- GPU's are currently absolutely crucial for computation...
- Convolutional networks are still being developed and improved...
  ... not only brute force but also novel architectures
- It seems that having multiple possibilities of traversing the network is very beneficial

Conclusions

- Convolutional networks are used primarily for images but can be also used for time-series data (1D convolutions) and video (3D convolution)
- Apart from classification, CNNs are also used e.g. for image segmentation
- GPU's are currently absolutely crucial for computation...
- Convolutional networks are still being developed and improved...
  ... not only brute force but also novel architectures
- It seems that having multiple possibilities of traversing the network is very beneficial

Conclusions

- Convolutional networks are used primarily for images but can be also used for time-series data (1D convolutions) and video (3D convolution)
- Apart from classification, CNNs are also used e.g. for image segmentation
- GPU's are currently absolutely crucial for computation...
- Convolutional networks are still being developed and improved...
  ... not only brute force but also novel architectures
- It seems that having multiple possibilities of traversing the network is very beneficial

Conclusions

- Convolutional networks are used primarily for images but can be also used for time-series data (1D convolutions) and video (3D convolution)
- Apart from classification, CNNs are also used e.g. for image segmentation
- GPU's are currently absolutely crucial for computation...
- Convolutional networks are still being developed and improved... ... not only brute force but also novel architectures
- It seems that having multiple possibilities of traversing the network is very beneficial

Conclusions

- Convolutional networks are used primarily for images but can be also used for time-series data (1D convolutions) and video (3D convolution)
- Apart from classification, CNNs are also used e.g. for image segmentation
- GPU's are currently absolutely crucial for computation...
- Convolutional networks are still being developed and improved...
  ... not only brute force but also novel architectures
- It seems that having multiple possibilities of traversing the network is very beneficial