Deep Monte Carlo

Marc Kachelrieß

German Cancer Research Center (DKFZ)

Heidelberg, Germany

www.dkfz.de/ct

DEUTSCHES KREBSFORSCHUNGSZENTRUM IN DER HELMHOLTZ-GEMEINSCHAFT

Content

- Coarse CNN overview
- Deep scatter estimation

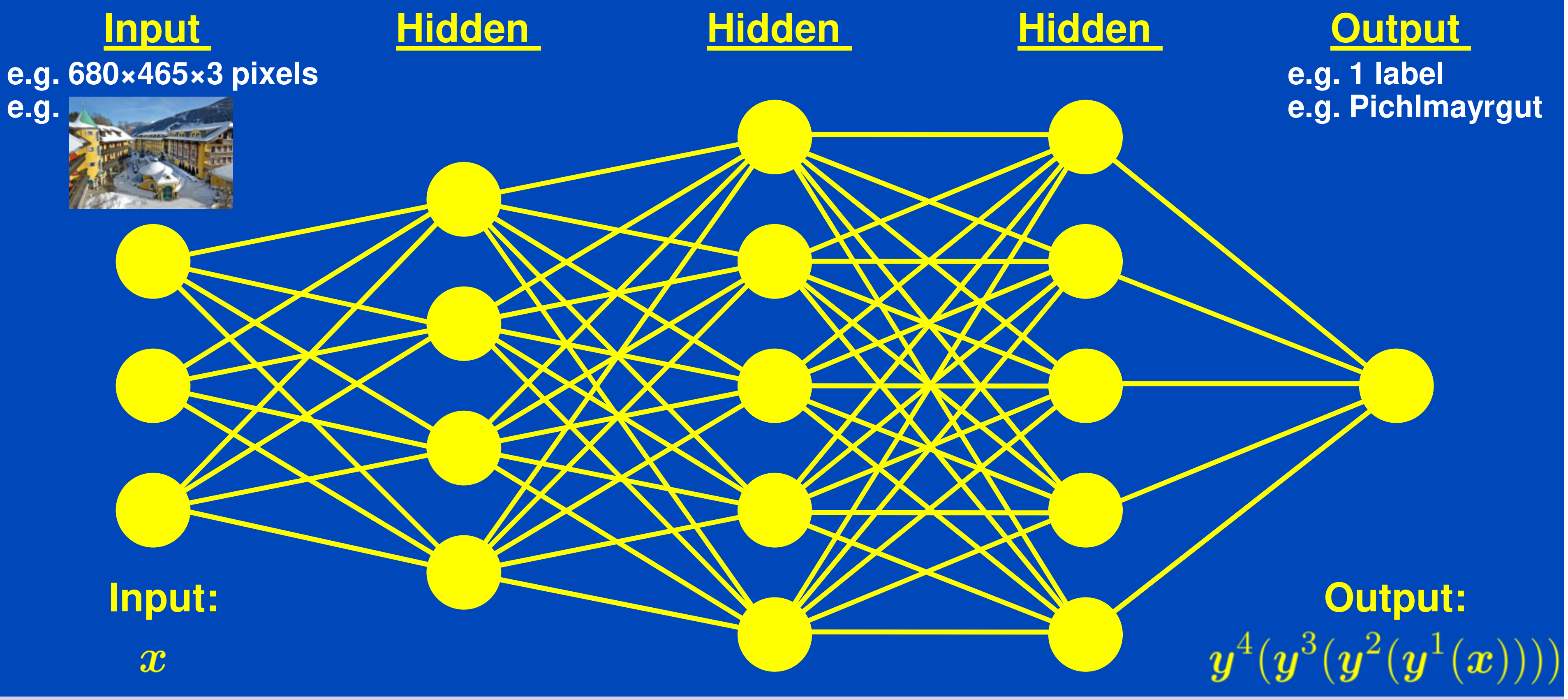
Coarse CNN Overview

Nomenclature

- Iteration = Epoch
- Batch = Subset (randomly changing for each epoch)
- Loss function = Cost function
- Learning rate = η

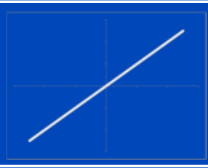
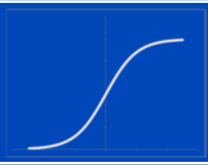
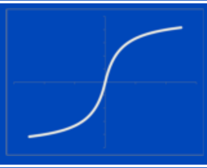
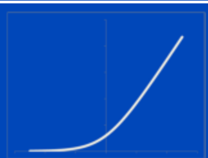
Fully Connected Network

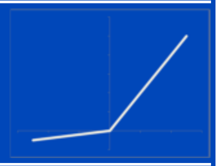
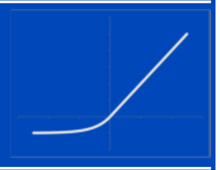
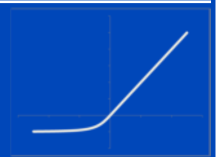
- Each layer fully connects to previous layer
- Difficult to train (many parameters)
- Spatial relations not necessarily preserved



$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}) \quad \text{with} \quad \boldsymbol{f}(\boldsymbol{x}) = \big(f(x_1), f(x_2), \ldots\big)$$

Activation Functions

Function	Equation	Plot
Identity	$f(x) = x$	
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	
Hard sigmoid	$f(x) = \begin{cases} 0 & \text{for } x < -\alpha \\ \frac{\alpha+x}{2\alpha} & \text{for } -\alpha \leq x < \alpha \\ 1 & \text{for } x \geq \alpha \end{cases}$	
Tanh	$f(x) = \frac{2}{1+e^{-2x}} - 1$	
Softsign	$f(x) = \frac{x}{1+\lvert x \rvert}$	
Softplus	$f(x) = \log(1 + \exp x)$	

Function	Equation	Plot
ReLU	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	
Leaky ReLU	$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	
ELU	$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	
Inverse square root LU	$f(x) = \begin{cases} \frac{x}{\sqrt{1+\alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	
...	...	...

Loss Function

- The neural networks parameters (weights) *w* are chosen by minimizing a loss function (cost function)

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} L(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{w})$$

  with x_n being the training data input and y_n being the training data output and *N* being the number of training samples.

- An example for the loss function is

$$L(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{w}) = \big(\boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{y}_n)\big)^2$$

Gradient Descent

- Walk along the direction of the negative gradient
- Steepest descent
- Learning rate η

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \eta\, \nabla_{\boldsymbol{w}} L(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{w})$$

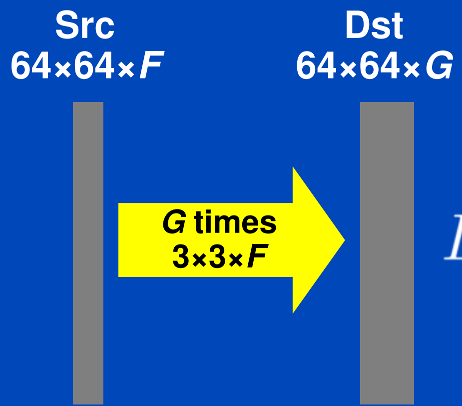
- Easy to understand, but not optimal
- Methods in use
  - Batch gradient descent
  - Sochastic gradient descent
  - Mini-batch gradient descent
  - Conjugate gradient descent
  - Quasi Newton methods
  - Momentum methods
  - …

Convolutional Layers

- Convolution in spatial domain
- Full connectivity in depth
- Filter size = receptive field
- Learns filter kernels
- Less parameters than fully connected net
- Respects properties of many imaging systems

Convolution

- Input layer *S*
  - vector of size *I* with *F* features: $I \times F$
  - image of size *I* by *J* with *F* features: $I \times J \times F$
  - volume of size *I* by *J* by *K* with *F* features: $I \times J \times K \times F$
  - …
- Convolution kernel *K*
  - G kernels of size $(2A+1) \times (2B+1) \times F$ with (e.g. zero) padding
- Output layer *D*
  - same spatial dimensions as input layer
  - *G* features (depth *G*)



$$D_{i,j,g} = \sum_f S_{i,j,f} * K^g_{i,j,f} = \sum_{a,b,f} S_{i-a,j-b,f} K^g_{a,b,f}$$

Attention: No convolution in depth direction!

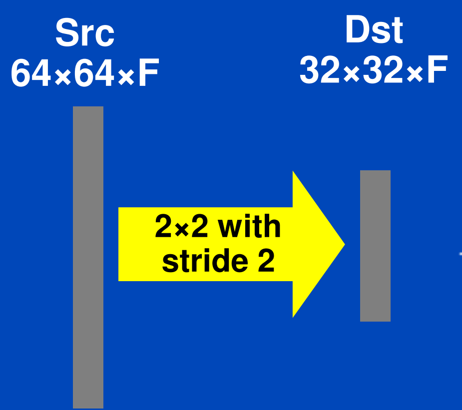
Pooling

- Input layer *S*
  - image of size *I* by *J* with *F* features: $I \times J \times F$
  - …
- Pooling kernel
  - pooling function, e.g. max, mean, stochastic, …
  - size and strides
- Output layer *D*
  - reduced spatial size
  - same depth

1	1	1	3	2	3	1	2
2	3	0	3	1	9	6	9
1	8	0	4	0	8	9	9
1	1	2	3	9	2	3	1
0	5	1	3	2	1	1	3
1	1	1	1	0	0	1	1
2	5	0	7	1	9	7	9
2	0	0	8	2	4	0	1

2×2 stride 2×2 max pool

3	3	9	9
8	4	9	9
5	3	2	3
5	8	9	9



$$D_{i,j,f} = \max_{b,d} S_{ai+b,cj+d,f}$$

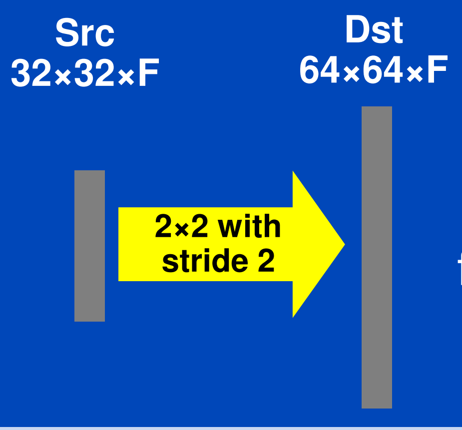
Unpooling

- **Input layer *S***
  - image of size *I* by *J* with *F* features: $I \times J \times F$
  - …
- **Unpooling kernel**
  - pooling function, e.g. max, mean, stochastic, …
  - size and strides
- **Output layer *D***
  - increased spatial size
  - same depth

0	0	0	3	0	0	0	0
0	3	0	3	0	9	0	9
0	8	0	4	0	0	9	9
0	0	0	0	9	0	0	0
0	5	0	3	2	0	0	3
0	0	0	0	0	0	0	0
0	5	0	0	0	9	0	9
0	0	0	8	0	0	0	0

← 2×2 stride 2×2 max unpool

3	3	9	9
8	4	9	9
5	3	2	3
5	8	9	9



Max values at max positions that were originally found during pooling. Zeroes at non-max positions.

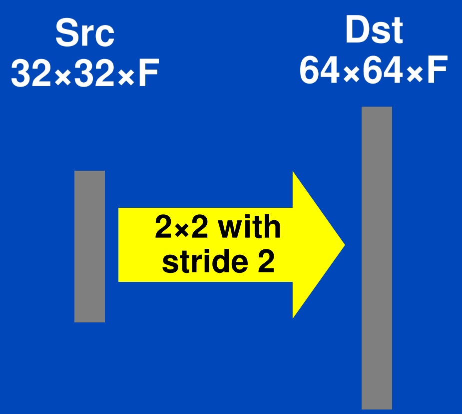
Unpooling
Upsampling

- Input layer *S*
  - image of size *I* by *J* with *F* features: $I \times J \times F$
  - …
- Unpooling kernel
  - pooling function, e.g. max, mean, stochastic, …
  - size and strides
- Output layer *D*
  - increased spatial size
  - same depth

3	3	3	3	9	9	9	9
3	3	3	3	9	9	9	9
8	8	4	4	9	9	9	9
8	8	4	4	9	9	9	9
5	5	3	3	2	2	3	3
5	5	3	3	2	2	3	3
5	5	8	8	9	9	9	9
5	5	8	8	9	9	9	9

← 2×2 stride 2×2 max unpool

3	3	9	9
8	4	9	9
5	3	2	3
5	8	9	9



Max values at all positions.

dkfz.

Dilated Convolutions

- Convolution

$$D_{i,j,g} = \sum_f S_{i,j,f} * K^g_{i,j,f} = \sum_{a,b,f} S_{i-a,j-b,f} K^g_{a,b,f}$$

- 8-dilated convolution

$$D_{i,j,g} = \sum_f S_{i,j,f} *_8 K^g_{i,j,f} = \sum_{a,b,f} S_{i-8a,j-8b,f} K^g_{a,b,f}$$

- Dilation helps to increase the receptive field of the kernel without increasing the number of unknowns in the kernel.
- Similar effect as pooling followed by convolution.

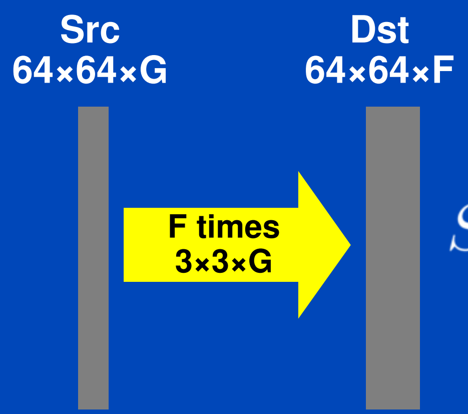
Deconvolution

- Transpose of the convolution
- Deconvolution layer is a very unfortunate name and should rather be called a transposed convolutional layer.
- Uses the weights of the adjunct convolution

$$D_{i,j,g} = \sum_{f} S_{i,j,f} * K^{g}_{i,j,f} = \sum_{a,b,f} S_{i-a,j-b,f} K^{g}_{a,b,f}$$

Convolution

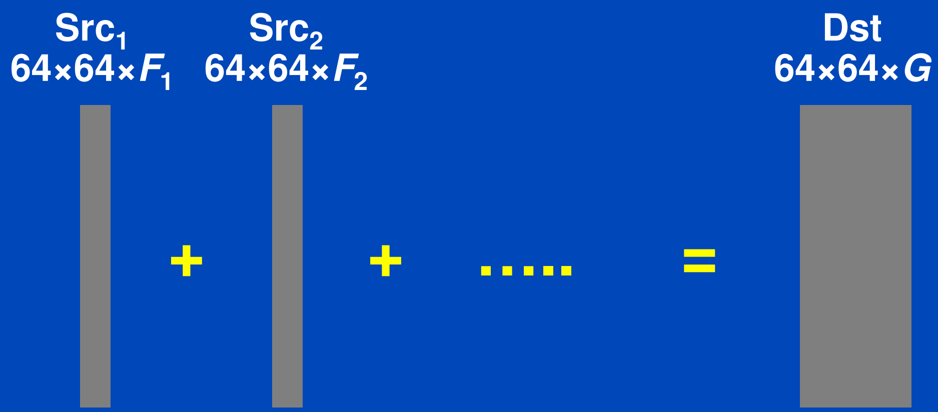
Deconvolution



$$S_{i,j,f} = \sum_{a,b,g} D_{i+a,j+b,g} K^{g}_{a,b,f}$$

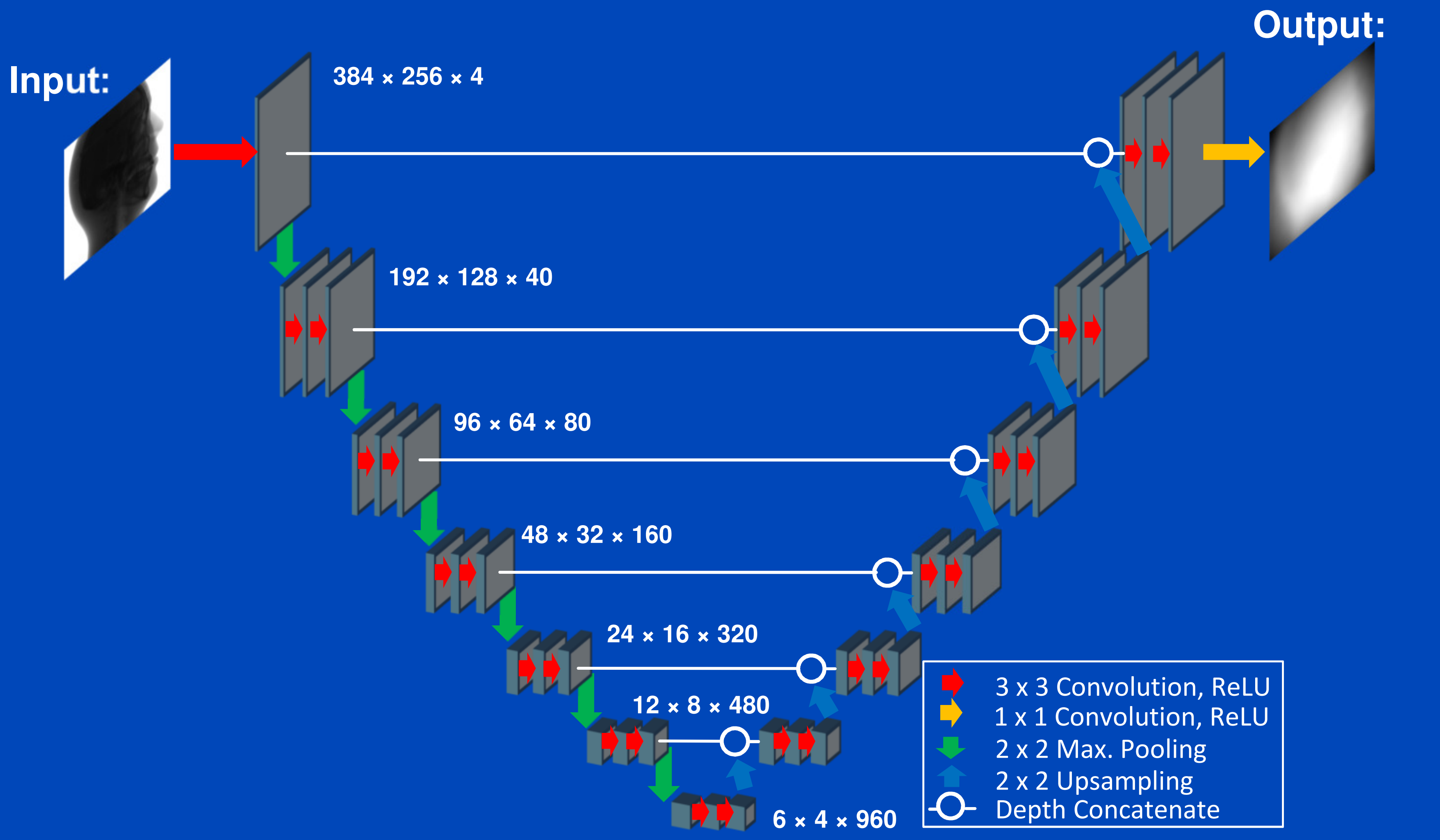
Depth Concatenation

- *N* input layers S_n
  - vector of size *I* with F_n features: $I \times F_n$
  - image of size *I* by *J* with F_n features: $I \times J \times F_n$
  - volume of size *I* by *J* by *K* with F_n features: $I \times J \times K \times F_n$
  - …
- Output layer *D*
  - same spatial dimensions as input layer
  - $G = F_1 + F_2 + \ldots + F_N$ features



$$G = \sum_n F_n$$

U-Net


Input:
Output:
384 × 256 × 4
192 × 128 × 40
96 × 64 × 80
48 × 32 × 160
24 × 16 × 320
12 × 8 × 480
6 × 4 × 960
3 x 3 Convolution, ReLU
1 x 1 Convolution, ReLU
2 x 2 Max. Pooling
2 x 2 Upsampling
Depth Concatenate

dkfz.

Toy Example

Nested 1D functions $f_n(c_n, x)$ with unknown coefficients c_n

loss function
$$L(c_3, c_2, c_1, x) = \left(f_3(c_3, f_2(c_2, f_1(c_1, x))) - y\right)^2$$

intermediate values

1 $$L_3 = \frac{dL}{df_3}$$

3 $$L_2 = \frac{dL}{df_2} = \frac{dL}{df_3}\frac{df_3}{df_2} = L_3 \frac{df_3}{df_2}$$

$$\dots$$

$$L_n = \frac{dL}{df_n} = L_{n+1}\frac{df_{n+1}}{df_n}$$

Backpropagation

desired gradients

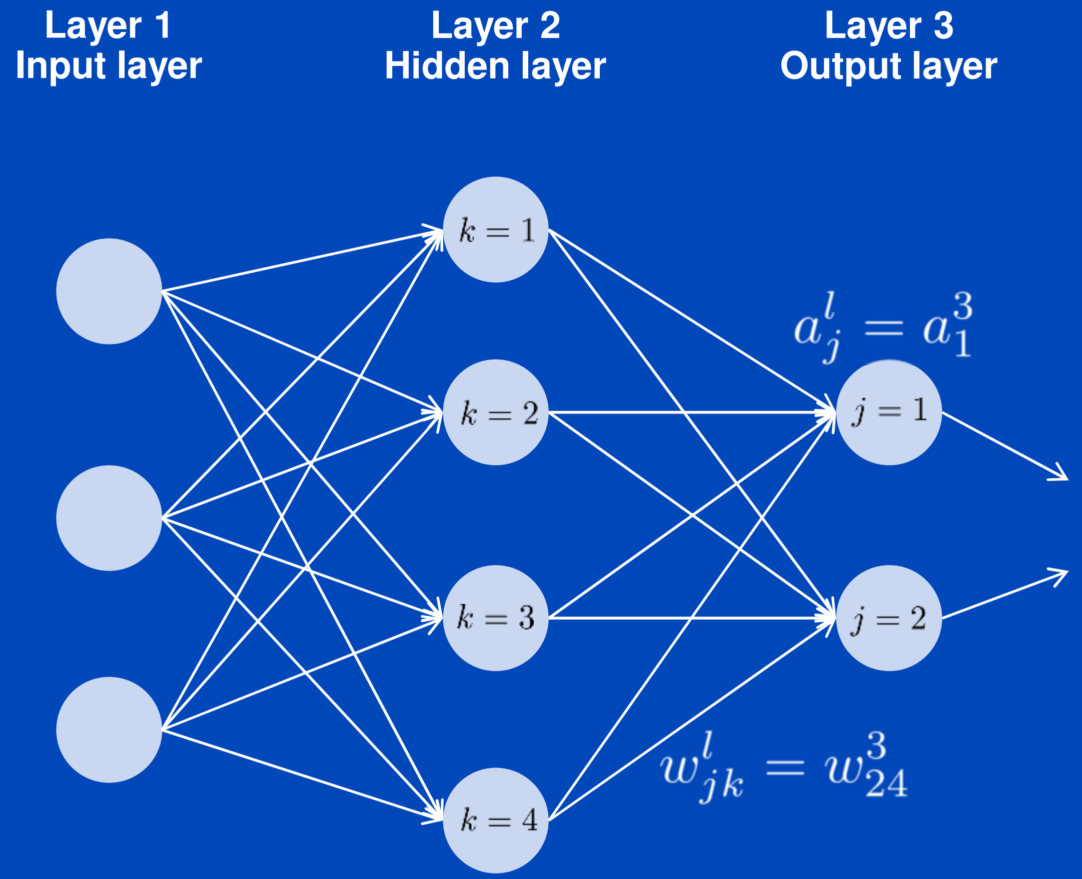
2 $$\frac{dL}{dc_3} = \frac{dL}{df_3}\frac{df_3}{dc_3} = L_3\frac{df_3}{dc_3}$$

4 $$\frac{dL}{dc_2} = \frac{dL}{df_3}\frac{df_3}{df_2}\frac{df_2}{dc_2} = L_2\frac{df_2}{dc_2}$$

$$\dots$$

$$\frac{dL}{dc_n} = L_n\frac{df_n}{dc_n}$$

Neural Network – General Structure



l = layer index

j = neuron index in l^{th} layer

k = neuron index in $(l-1)^{\text{th}}$ layer

σ = activation function

w_{jk}^l = weight from k^{th} neuron in layer $(l-1)$ to j^{th} neuron in layer l

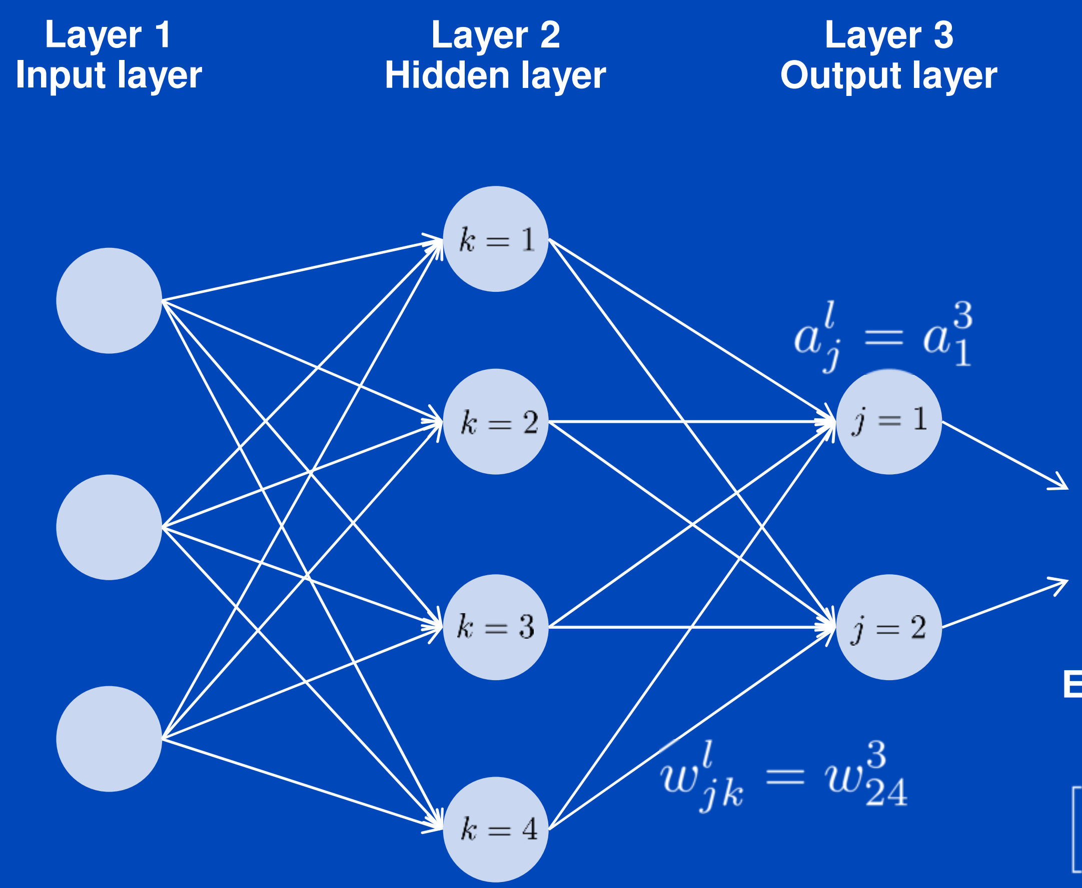
b_j^l = bias of the j^{th} neuron in the l^{th} layer

a_j^l = activation of the j^{th} neuron in the l^{th} layer

$$= \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

Neural Network – General Structure

Matrix notation



a^l =activation vector of layer l

w^l =weight matrix from layer $(l-1)$ to l

b^l =bias vector of layer l

$$a^l = \sigma\left(w^l a^{l-1} + b^l\right)$$

Example: activation of the 3rd layer

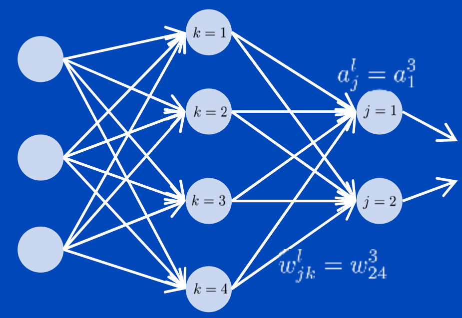
$$\begin{bmatrix} a_1^3 \\ a_2^3 \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{11}^3 & w_{12}^3 & w_{13}^3 & w_{14}^3 \\ w_{21}^3 & w_{22}^3 & w_{23}^3 & w_{24}^3 \end{bmatrix} \cdot \begin{bmatrix} a_1^2 \\ a_2^2 \\ a_3^2 \\ a_4^2 \end{bmatrix} + \begin{bmatrix} b_1^2 \\ b_2^2 \end{bmatrix} \right)$$

Optimization of Weights and Biases

- **The weights and biases can be optimized using a gradient descent approach:**

$$w_{jk}^{l}{}' = w_{jk}^{l} - \eta \frac{\partial C}{\partial w_{jk}^{l}}$$

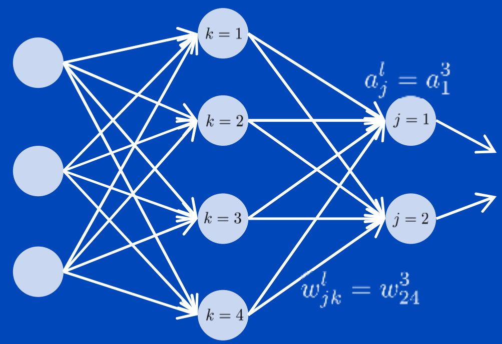
$$b_{j}^{l}{}' = b_{j}^{l} - \eta \frac{\partial C}{\partial b_{j}^{l}}$$



Optimization of Weights and Biases

Backpropagation

- **Backpropagation is an efficient way to calculate the gradient of the weights and biases.**
- **Let us define the error δ_j^l of neuron j in layer l:**



$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}; \qquad z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

- **The error of neurons in the last layer is given as:**

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

Proof: $$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} \overset{a_j^L=\sigma(z_j^L)}{=} \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

- **Vector notation:** $\delta^L = \nabla_a C \odot \sigma'(z^L); \qquad \odot = \text{Hadamard product}$

Optimization of Weights and Biases

Backpropagation

- Given the error δ^{l+1} in layer l+1, the error of the j^{th} neuron of the l^{th} layer is calculated as follows:

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$$

Proof:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

$$z_j^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} \overset{a_j^L = \sigma(z_j^L)}{=} \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

$$\rightarrow \frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

$$\rightarrow \delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$$

- Vector notation: $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$

Optimization of Weights and Biases

Backpropagation

- **The partial derivative of the cost function with respect to the weights is given by:**

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

Proof:

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_k} \frac{\partial z^l_k}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

- **The partial derivative of the cost function with respect to the bias is given by:**

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

Proof:

$$\frac{\partial C}{\partial b^l_j} = \frac{\partial C}{\partial z^l_k} \frac{\partial z^l_k}{\partial b^l_j} = \delta^l_j$$

The Backpropagation Algorithm

1. **Input x_n: Set the corresponding activation a^1 for the input layer.**
2. **Feedforward: For each layer *l* = *2*, …, *L* compute:**

$$z^l = w^l a^{l-1} + b^l \text{ and } a^l = \sigma(z^l)$$

3. **Output error: Compute the error vector of layer *L*:**

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

4. **Backpropagate error: For each *l* = *L*-1, *L*-2, …, 2:**

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

5. **Output: The gradient of the cost function is given by:**

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \qquad \frac{\partial C}{\partial b^l_j} = \delta^l_j$$

dkfz.

Gradient Descent

$M = 1$: Stochastic gradient descent
$1 < M < N$: Mini-batch gradient descent
$M = N$: Batch gradient descent

For each epoch:

Shuffle data

For each (mini-)batch *B* of size *M*:

For each sample x_n of the (mini-)batch:

i. **Set input activation:** $a^{n,1} = x_n$

ii. **Feedforward: For each layer l = 2, 3, …, L compute:**

$$z^{n,l} = w^l a^{n,l-1} + b^l \text{ and } a^{n,l} = \sigma(z^{n,l})$$

iii. **Output error $\delta^{x,l}$: Compute the vector**

$$\delta^{n,L} = \nabla_a C_n \odot \sigma'(z^{n,L})$$

iv. **Backpropagate the error: For each l = L-1, L-2, …, 2 compute:**

$$\delta^{n,l} = ((w^{l+1})^T \delta^{n,l+1}) \odot \sigma'(z^{n,l})$$

Update weights and biases:

$$w^{l'} = w^l - \frac{\eta}{M} \sum_{n \in B} \delta^{n,l} (a^{n,l-1})^T \qquad b^{l'} = b^l - \frac{\eta}{M} \sum_{n \in B} \delta^{n,l}$$

dkfz.

Batch Normalization

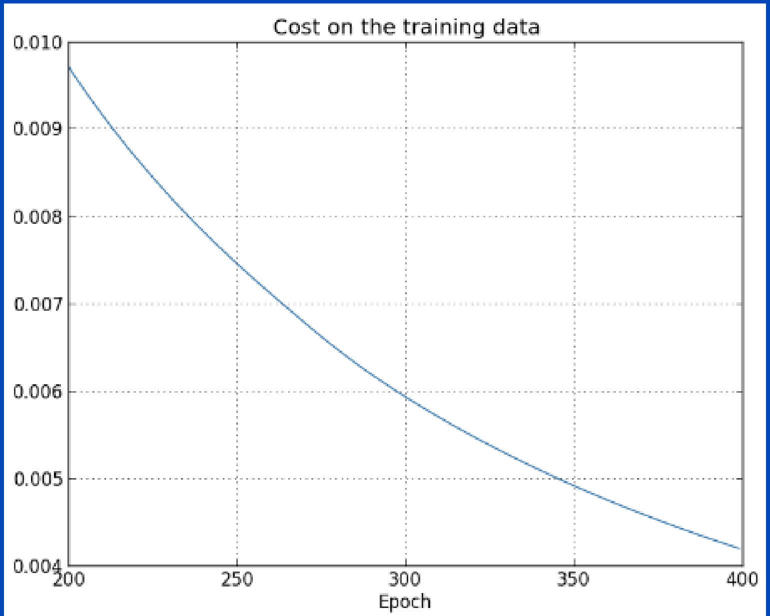
Batch normalization

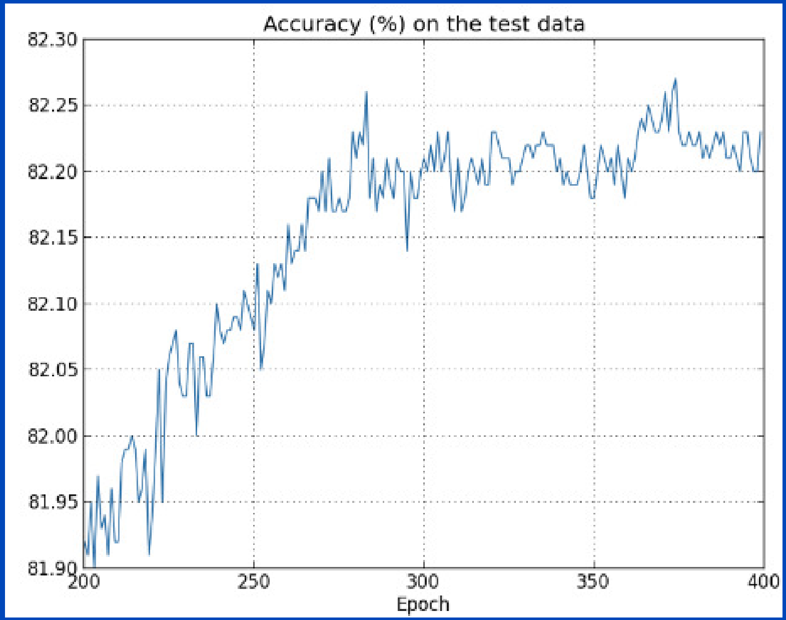
- normalizes each activation to have zero expectation and unit variance within the mini batch
- introduces trainable scale and offset for each activation (or for each feature map) to, potentially, denormalize again
- is part of the model architecture
- reduces the need for dropout
- reduces internal covariate shift and thus accelerates training
- fixes the means and variances of layer inputs
- improves gradient flow through the network
- allows for higher learning rates without the risk of divergence
- prevents the net from getting trapped in saturated modes
- makes it possible to use saturating nonlinearities

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015.

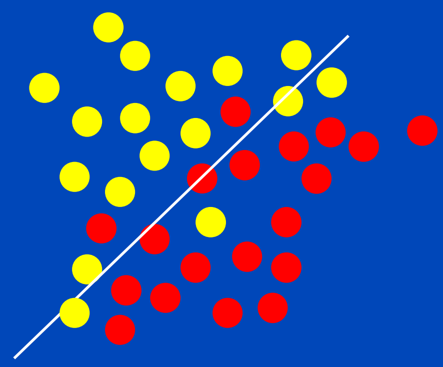
Overfitting

- Overfitting means, that the progress on training data no longer generalizes to test data.
- Overfitting can be prevented by using larger training sets or by applying regularization techniques.

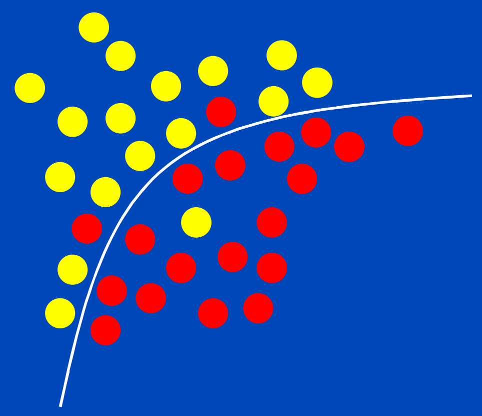




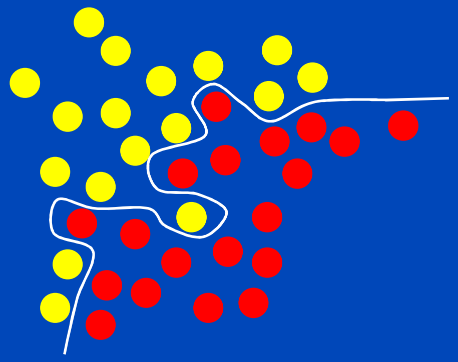
Fitting



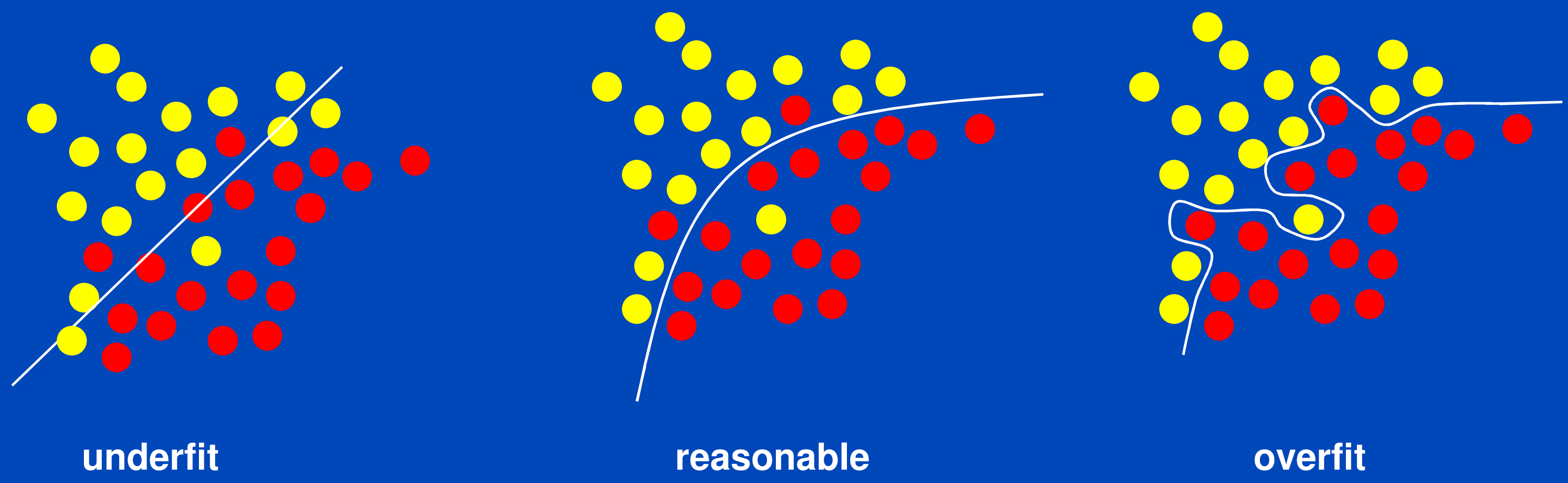
Fitting



Fitting

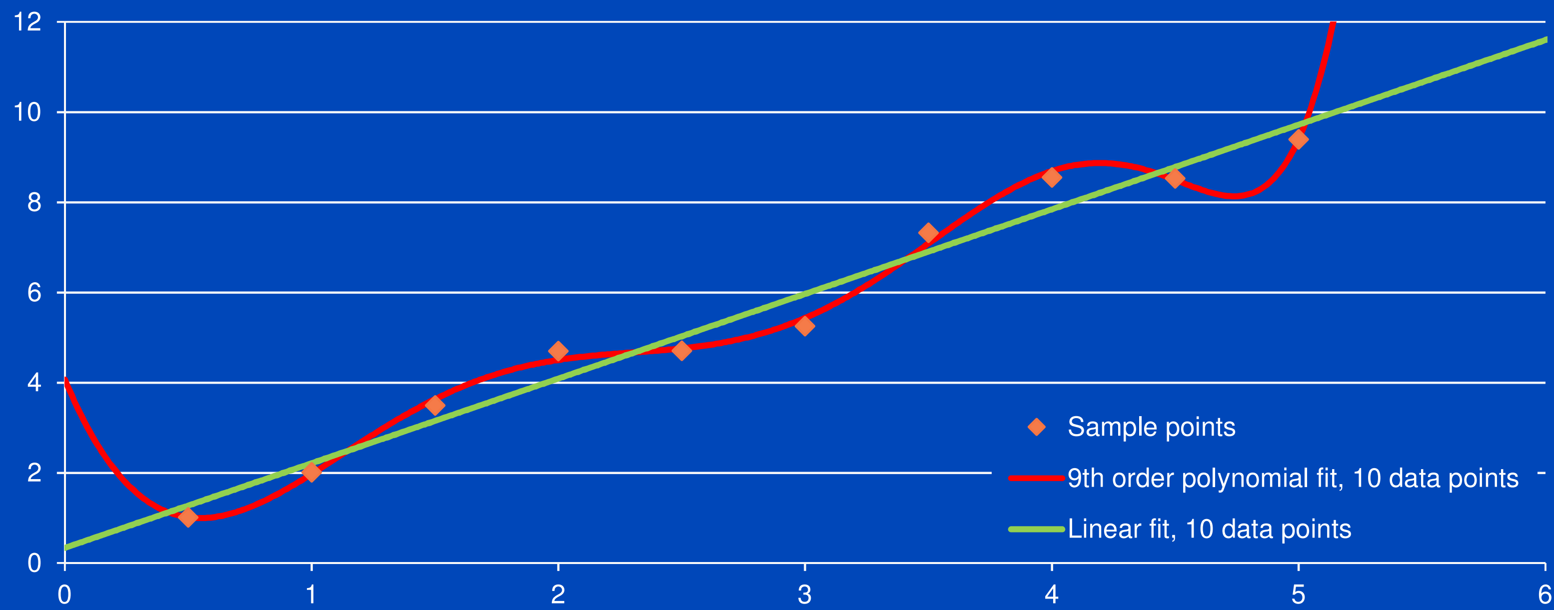


Fitting



Overfitting

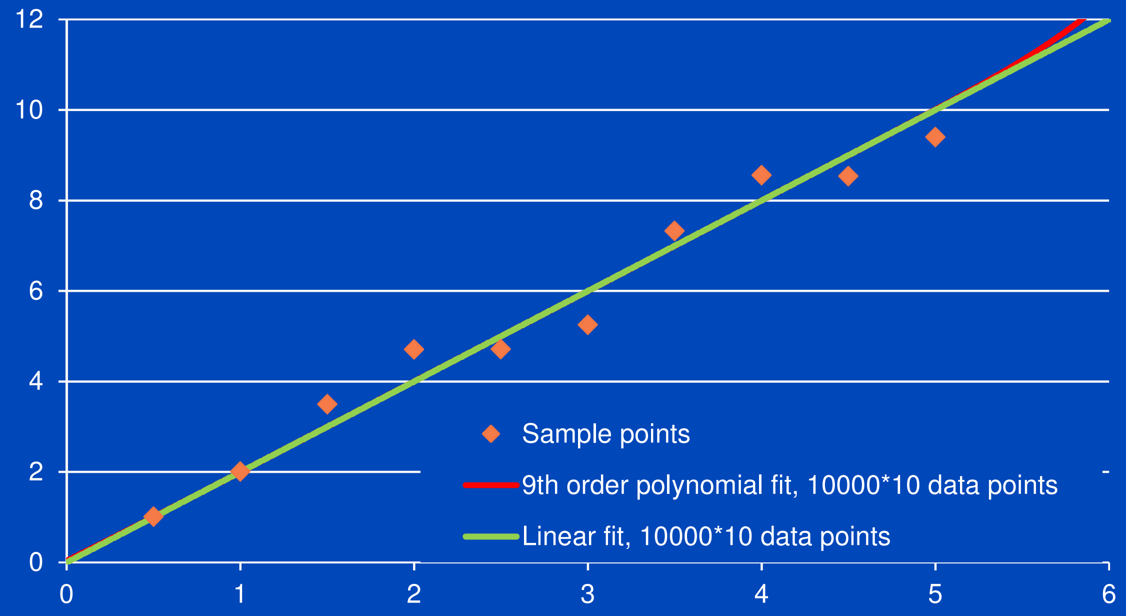
- Assume our training data results from sampling the function f(x) = 2x at a given number of points.
- Since the sampling might include some random noise, the samples slightly deviate from the function f(x) = 2x.
- A 9th order polynomial perfectly fits the training data, but fails to appropriately predict test data such as x = 0.25 for instance.



Regularization

Increase of training data

- The increase of the amount of training data makes the network more robust against single deviations.
- The training data can also be increased artificially.
- Similar results can be observed if the polynomial is fitted to 100000 samples.

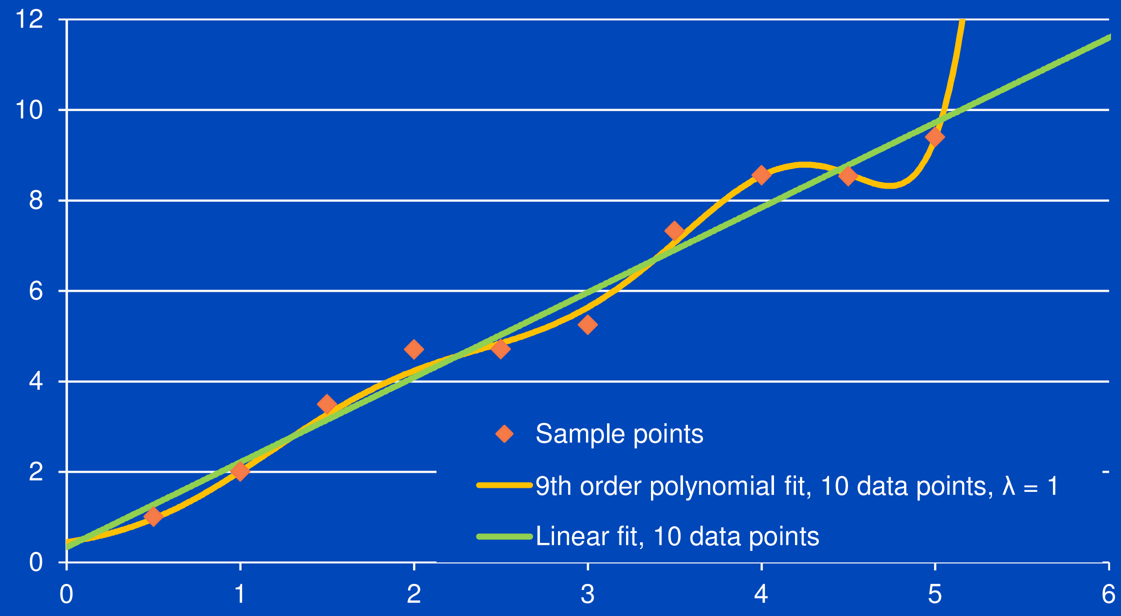


Coefficients	Linear	9th order
c_0	-0.00295	0.03343
c_1	2.000325	1.904762
c_2		0.079125
c_3		-0.02262
c_4		0.000435
c_5		-4.96E-05
c_6		0.000339
c_7		-4.25E-05
c_8		-9.19E-06
c_9		1.43E-06

Regularization

Penalizing large weights

- Modification of the cost function to penalize large weight (i.e. quadratic penalty): $C = C_0 + \lambda \sum_w w^2$
- If a certain weight is large, the output strongly depends on the input of that weight.

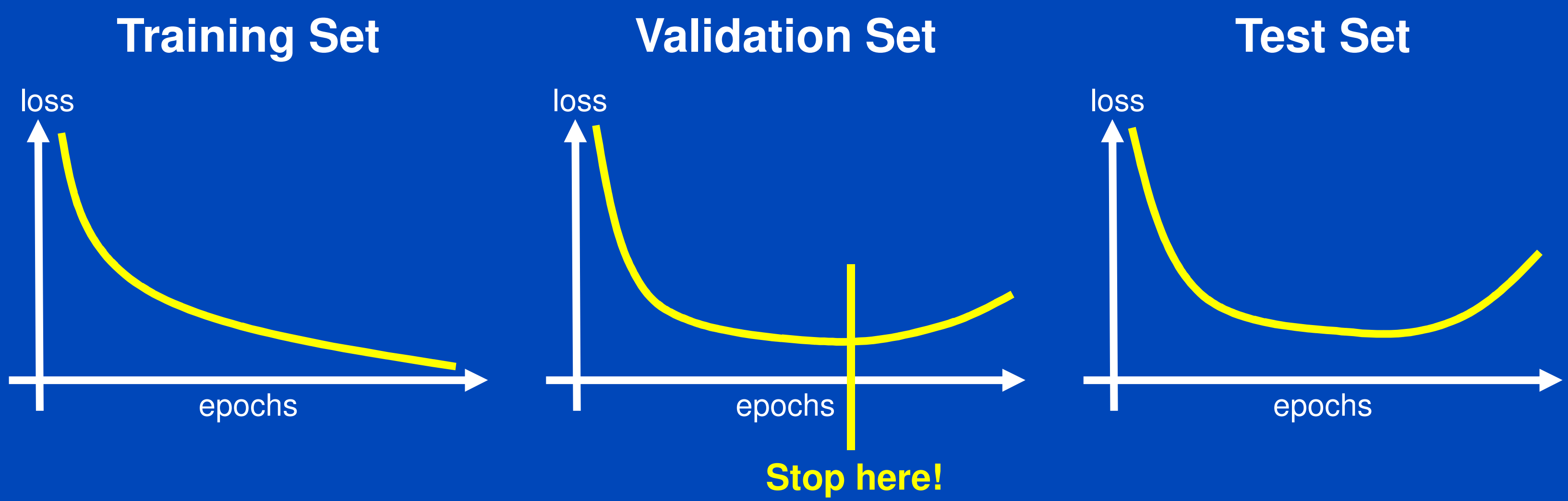


Coefficients	Linear	9th order
c_0	-0.00295	0.447558
c_1	2.000325	0.575279
c_2		0.665781
c_3		0.562606
c_4		0.049884
c_5		-0.45894
c_6		0.186099
c_7		-0.01496
c_8		-0.00342
c_9		0.000471

Avoid Overfitting

- Choose adequate network architecture
- Preprocess data
  - Normalize data (mean, var, …)
  - Add prior knowledge (e.g. exp(-x))
- Data augmentation
  - Random transformations (mirror, affine, deformable, …)
  - Gray value distribution
  - Change spatial resolution
  - Add noise
  - …
- Penalize loss function
  - Enforce small weights
  - Enforce sparse weights
  - …

Learning Curve



- Training and validation set are part of the training
- Do not use test set for training
- Early stopping (at minimum validation loss)
- Training : Validation : Test ≈ 70 : 20 : 10

Weight Initialization

- Weights in neural networks should be initialized such that the neurons are not saturated (since saturation often decreases the learning rate).
- Assume we have a fully connected network with 1000 input neurons.
- Let us further assume that half of the input equals 1 and the other half equals 0.
- If the weights and the bias are initialized with Gaussian random numbers with zero mean and a standard deviation of 1, the weighted sum $z = \sum w_j x_j + b$ to the first hidden neuron is zero mean Gaussian with standard deviation $\sigma = \sqrt{501} \approx 22.4$.
- Thus, it is very likely that $z \gg 1$ or $z \ll 1$. Consequently, it is very likely that the neuron is saturated.
- Therefore, if we have n_{in} inputs, an initialization with Gaussian random numbers with zero mean and a standard deviation of $1/\sqrt{n_{in}}$ would be a better choice.

dkfz.

Libraries

DL Libraries

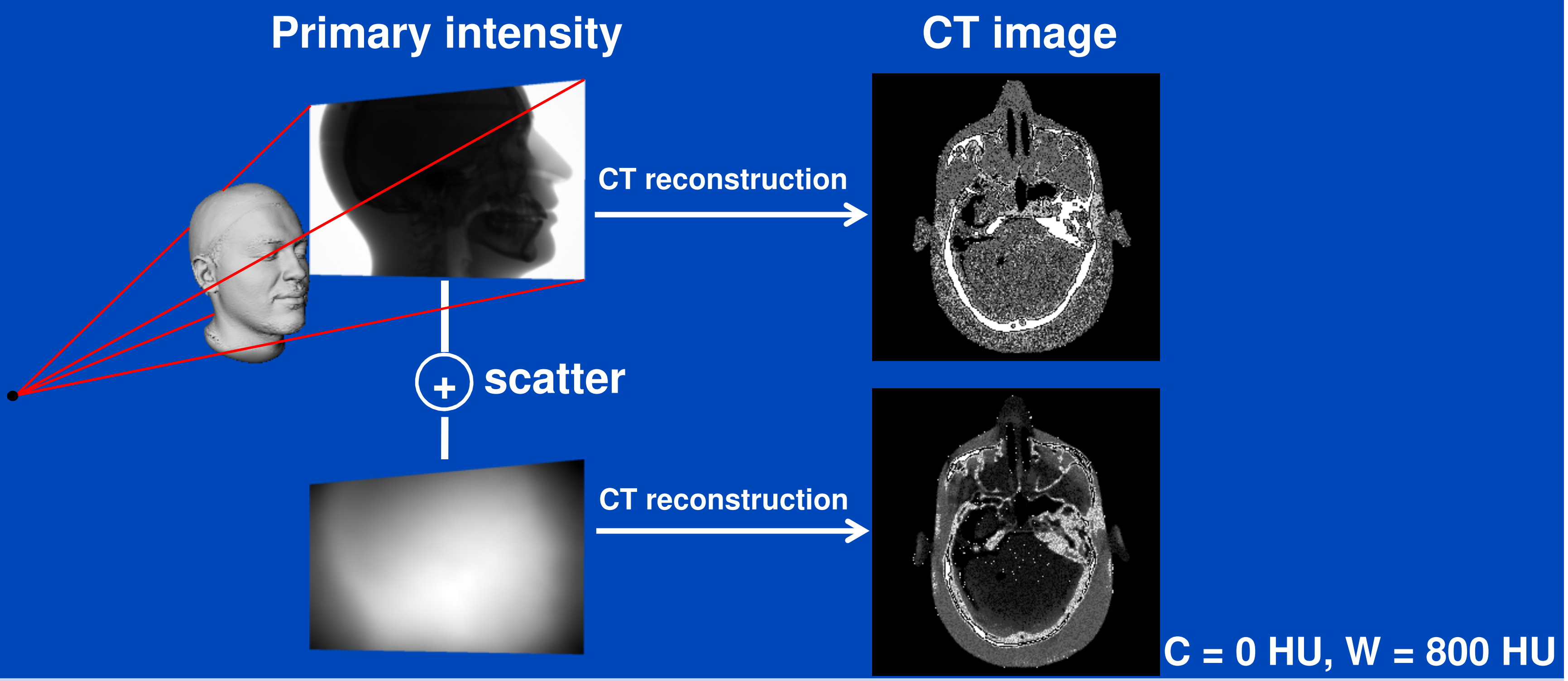
Descending order based on GitHub stars

Framework	(Main) Author(s)	(Main) Language(s)
Tensorflow	Google	Python
Caffe	BVLC	C++
Keras	F. Chollet	Python
CNTK	Microsoft	C++
MXNet adapted	Amazon	C++
Torch	Collobert, Kavukcuoglu, Farabet (also: Facebook)	Lua
Convnetjs	A. Karpathy	JavaScript
Theano	Université de Montréal	Python
Deeplearning4j	startup Skymind	Java
Paddle	Baidu	C++
DSSTNE	Amazon	C++
Neon	Nervana Systems	Python, Sass
Chainer		Python
h2o		Java
Brainstorm	IDSIA	Python
Matconvnet	A. Vedaldi	Matlab

Deep Scatter Estimation

Motivation

- X-ray scatter is a major cause of image quality degradation in CT and CBCT.
- Appropriate scatter correction is crucial to maintain the diagnostic value of the CT examination.



dkfz.

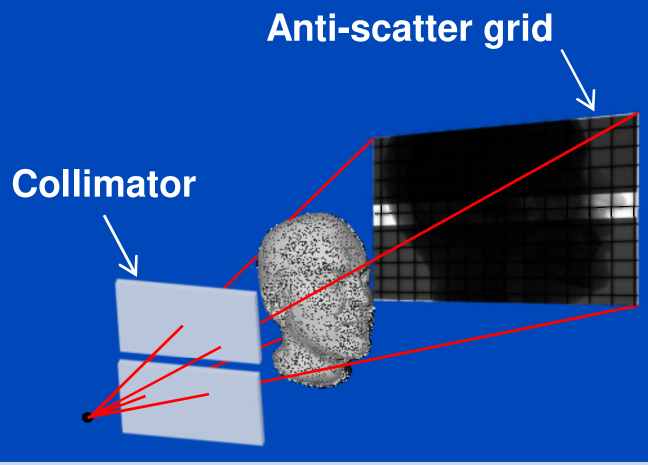
Scatter Correction

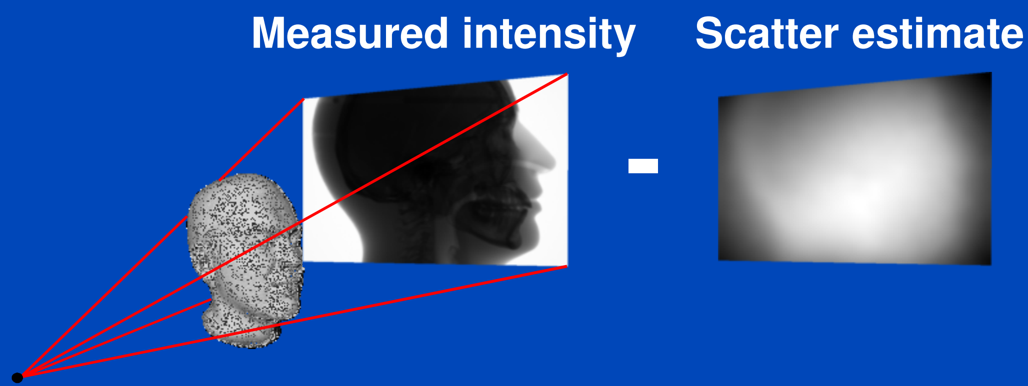
Scatter suppression

- Anti-scatter grids
- Collimators
- …

Scatter estimation

- Monte Carlo simulation
- Kernel-based approaches
- Boltzmann transport
- Primary modulation
- Beam blockers
- …

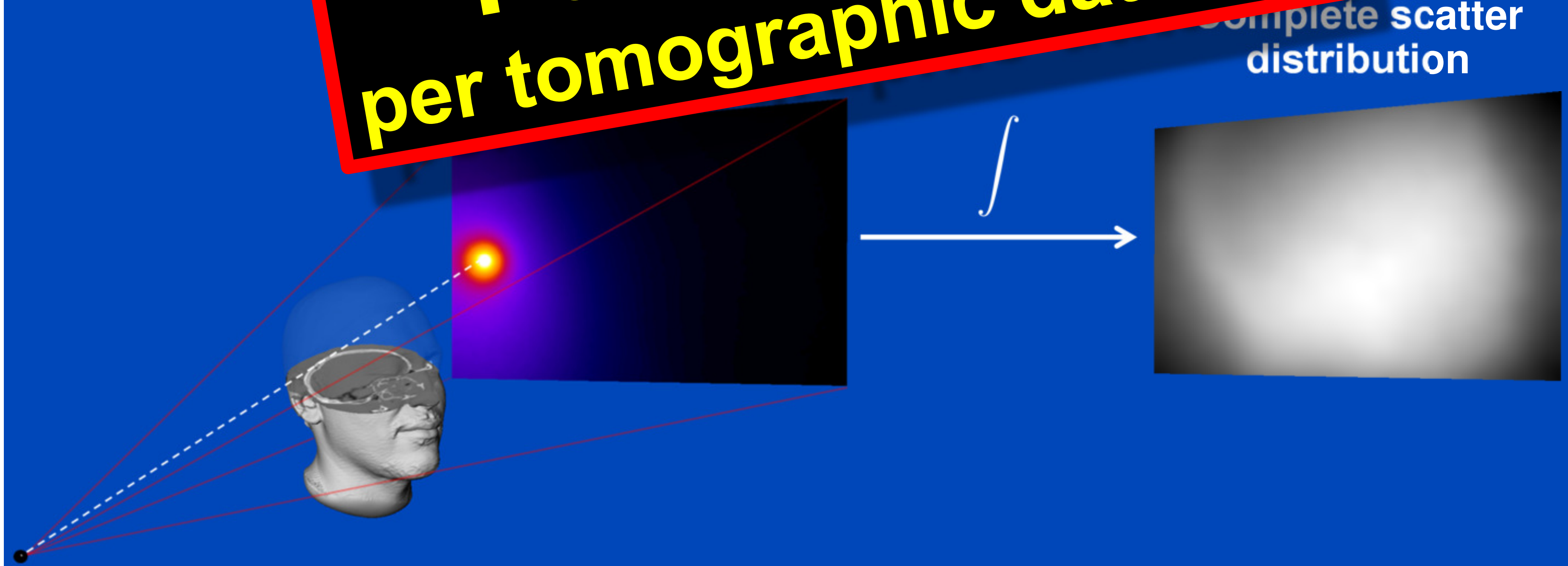




Monte Carlo Scatter Estimation

- Simulation of photon trajectories according to physical interaction probabilities.
- Simulating a large numb[redacted]ries well approximat[redacted]

1 to 10 hours
per tomographic data set

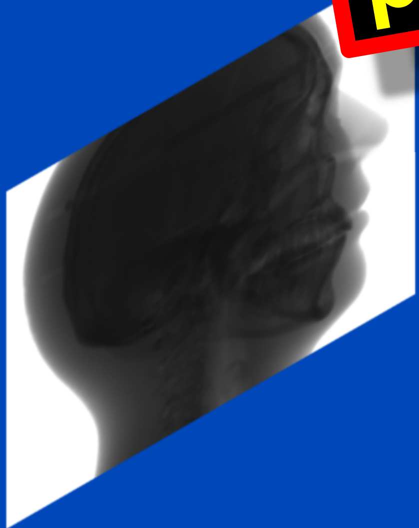


Deep Scatter Estimation (DSE)

Train a deep convolutional neural network (CNN) to estimate scatter using a function ... d projection data as in...

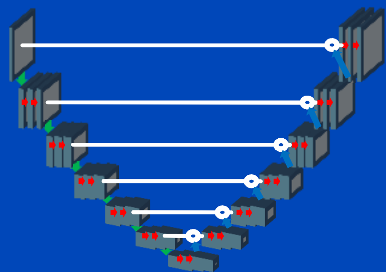
0.1 to 1 minute per tomographic data set

Input: $T(p)$

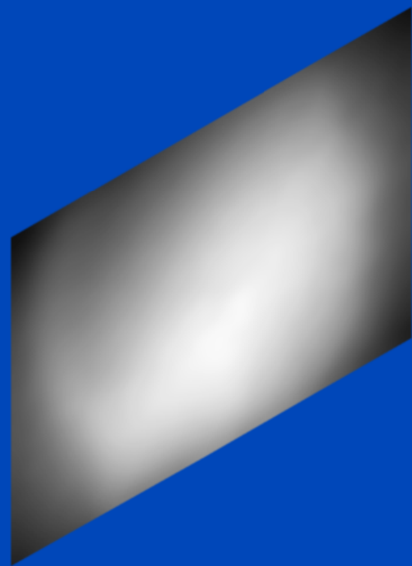


~~Monte Carlo~~

Convolutional neural network

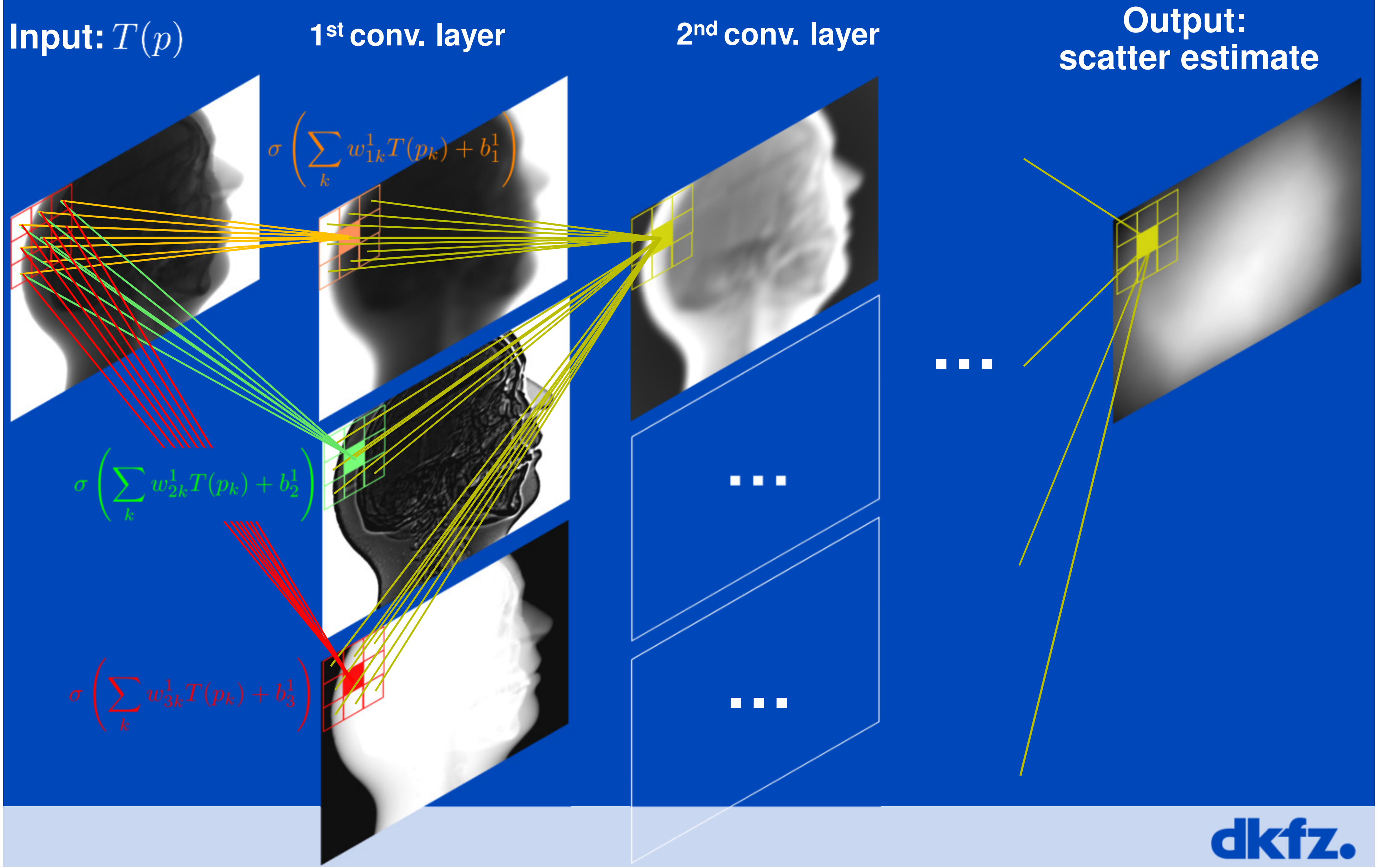


Scatter estimate



Convolutional Neural Networks

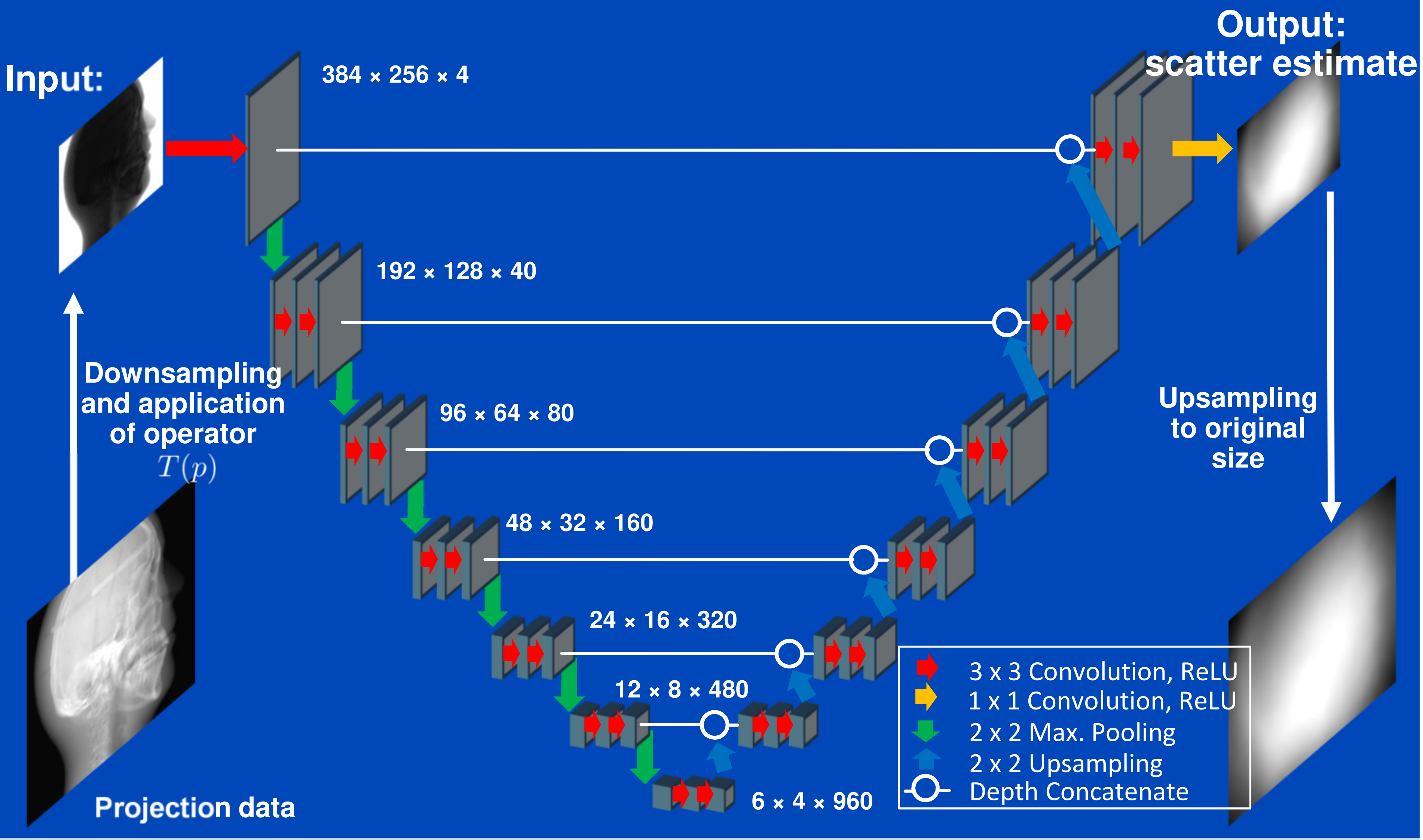
Basic principle



dkfz.

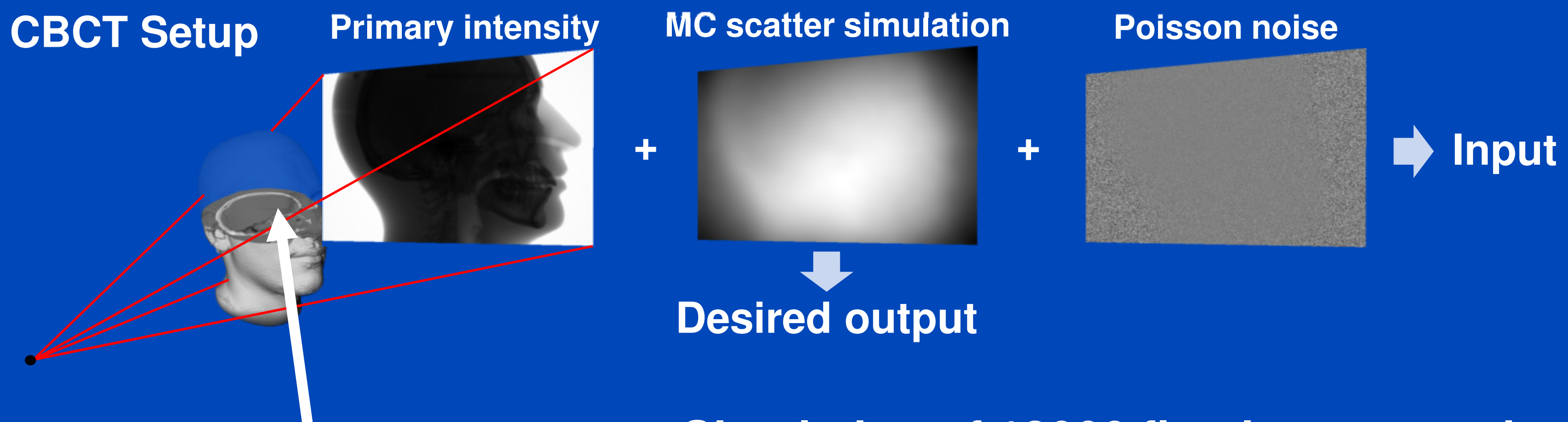
Deep Scatter Estimation

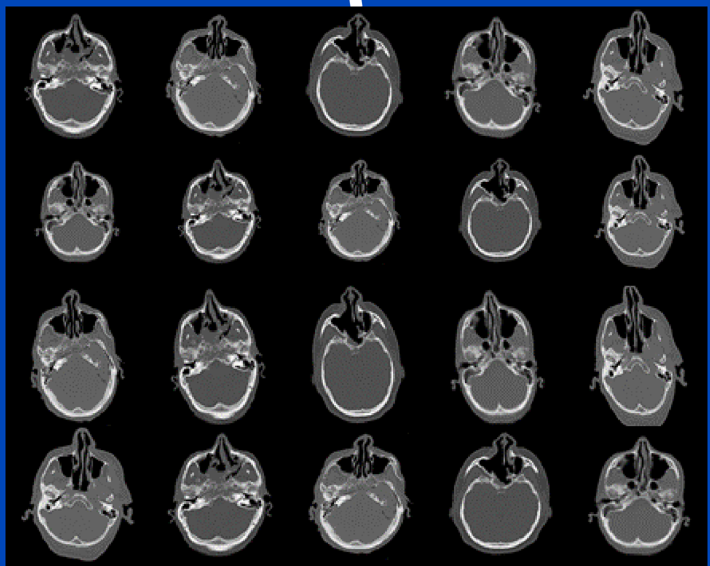
Network architecture & scatter estimation framework



dkfz.

Training the DSE Network



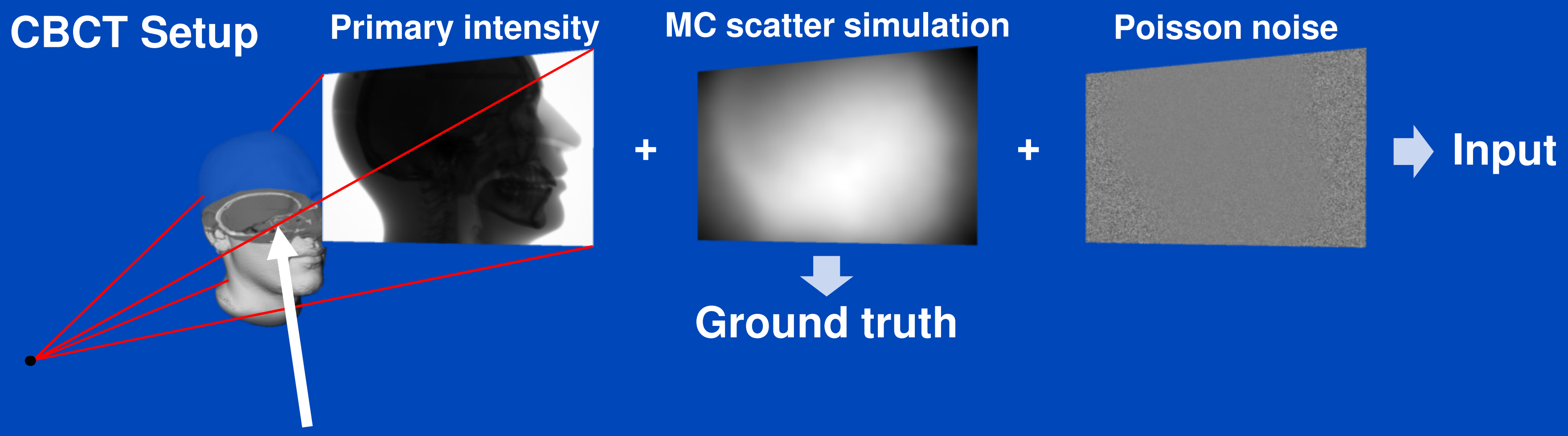


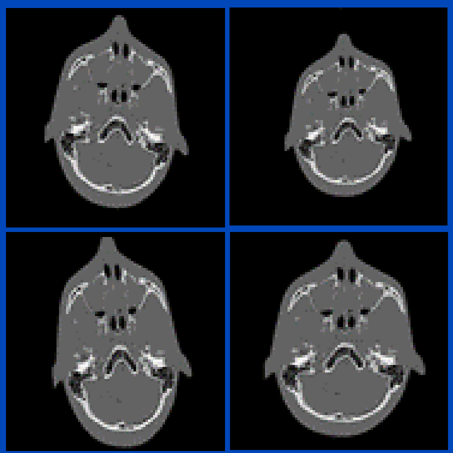
- Simulation of 12000 flat detector projection using data of different heads.
- Simulate different tube voltages.
- Splitting into 80% training and 20% validation data.
- Optimize weights of the CNN to reproduce the Monte Carlo scatter estimates:

$$(\boldsymbol{w}, \boldsymbol{b}) = \arg\min_{\boldsymbol{w},\boldsymbol{b}} \|\mathrm{DSE}_{\boldsymbol{w},\boldsymbol{b}}(T(p)) - I_{\mathrm{MC}}\|_2^2$$

- Training on a GeForce GTX 1080 for 80 epochs.

Testing of the DSE Network for Simulated Data (at 120 kV)

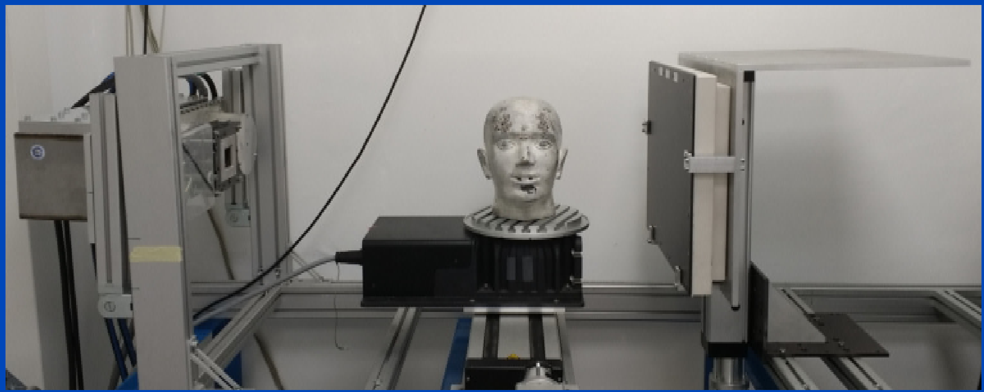




- Application of the DSE network to predict scatter for simulated data of a head (different from training data).

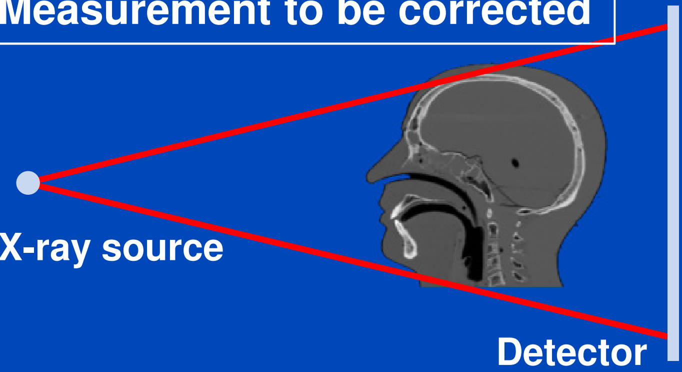
Testing of the DSE Network for Measured Data (120 kV)

DKFZ table-top CT

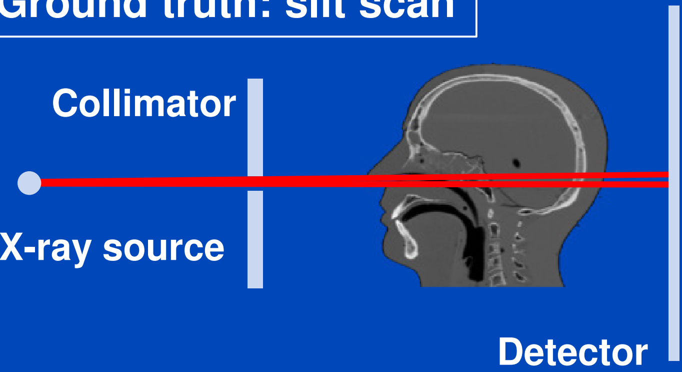


- Measurement of a head phantom at our in-house table-top CT.
- Slit scan measurement serves as ground truth.

Measurement to be corrected

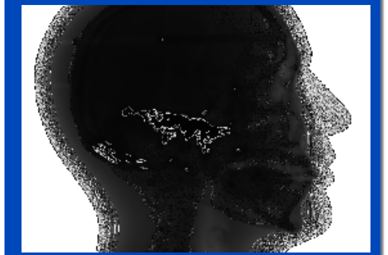


Ground truth: slit scan

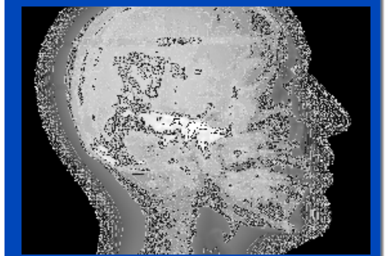


Performance for Different Inputs

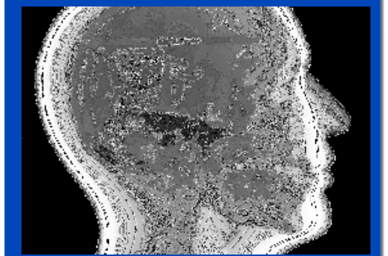
$T(p) = e^{-p}$

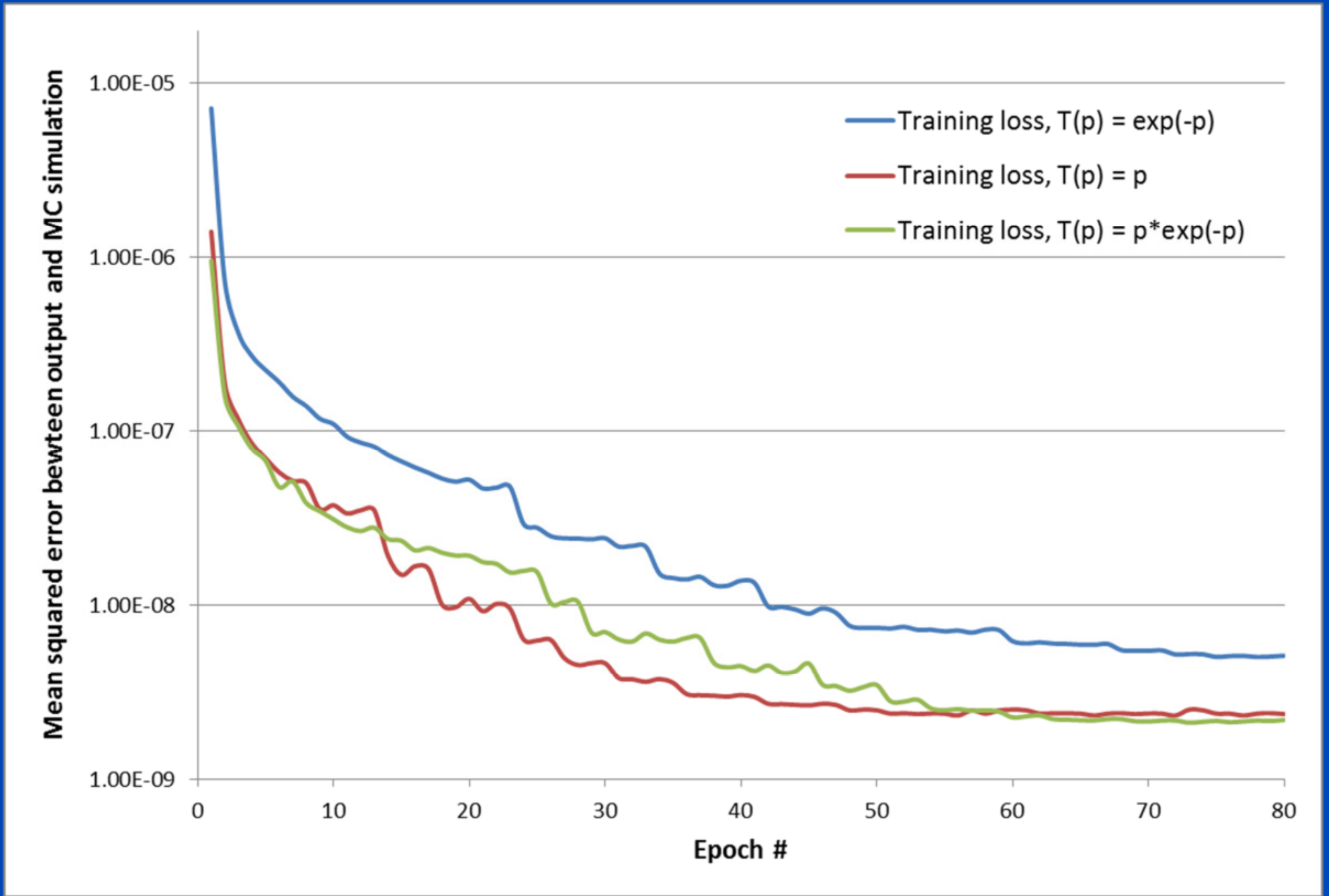


$T(p) = p$



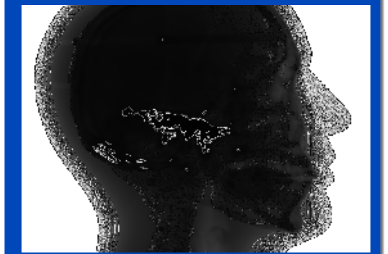
$T(p) = p \cdot e^{-p}$



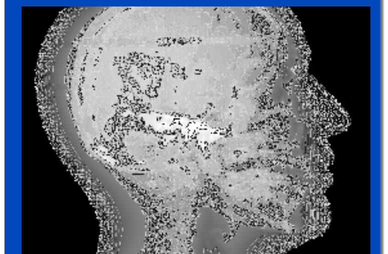


Performance for Different Inputs

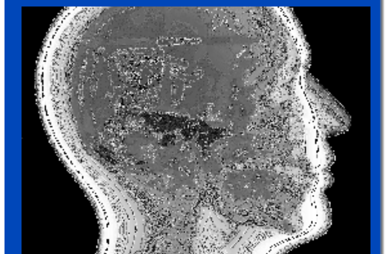
$T(p) = e^{-p}$

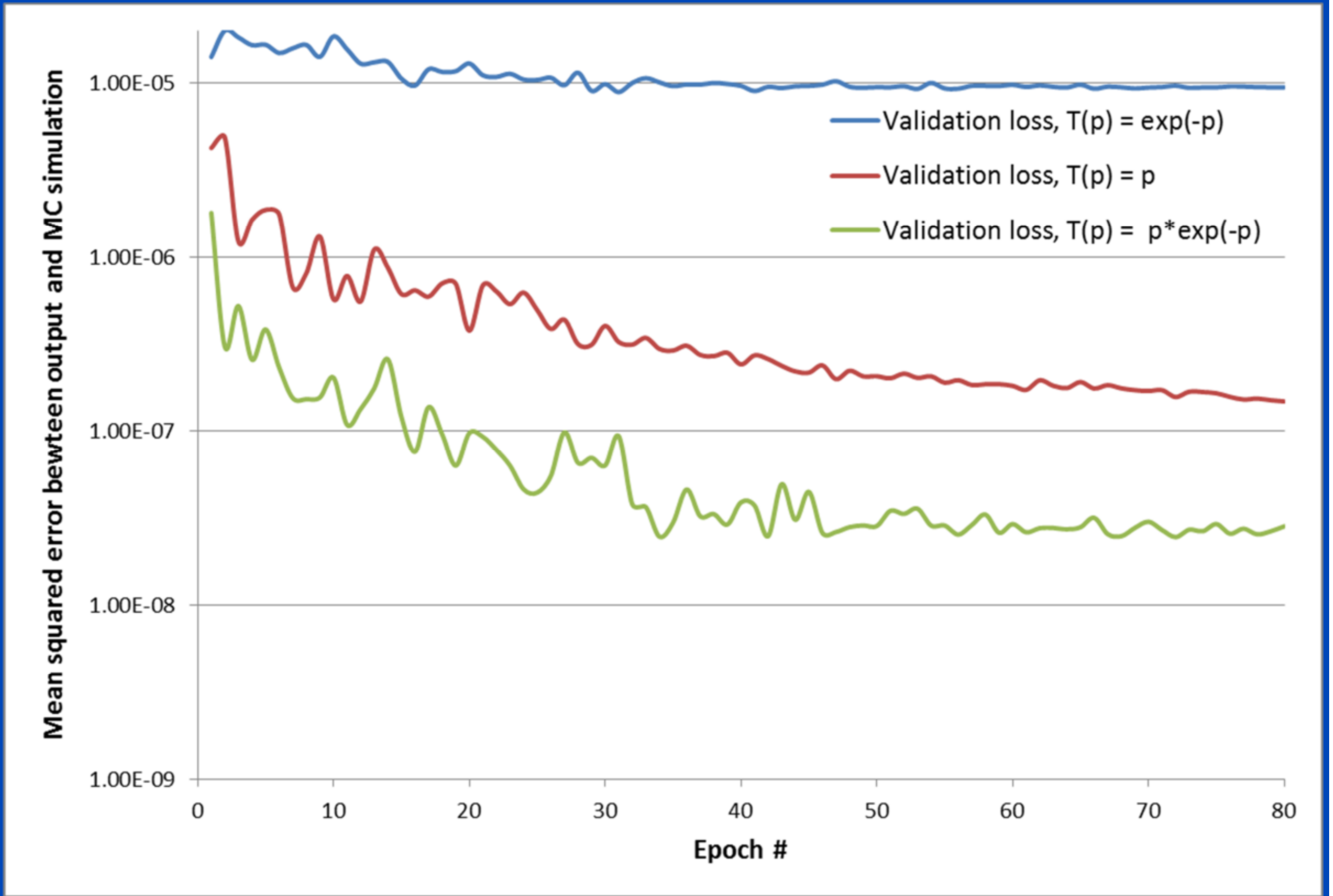


$T(p) = p$



$T(p) = p \cdot e^{-p}$

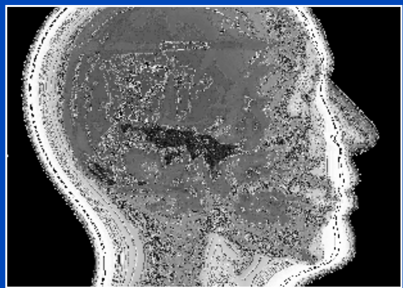




Ref 1: Kernel-Based Scatter Estimation

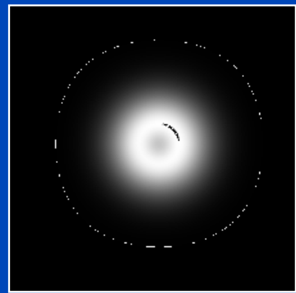
- **Kernel-based scatter estimation[1]:**
  - Estimation of scatter by a convolution of the scatter source term $T(p)$ with a scatter propagation kernel $G(\boldsymbol{u},\boldsymbol{c})$:

$$I_{\mathrm{s,\ est}}(\boldsymbol{u}) = \underbrace{\left(c_0 \cdot p(\boldsymbol{u}) \cdot e^{-p(\boldsymbol{u})}\right)}_{} * \underbrace{\left(\sum_{\pm} e^{-c_1(\boldsymbol{u}\hat{\boldsymbol{e}}_1 \pm c_2)^2} \cdot \sum_{\pm} e^{-c_3(\boldsymbol{u}\hat{\boldsymbol{e}}_2 \pm c_4)^2}\right)}_{}$$



$T(p)(\boldsymbol{u})$

Open parameters: c_0



$G(\boldsymbol{u},\boldsymbol{c})$

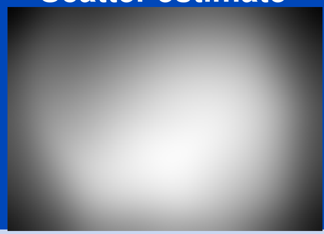
Open parameters: c_1, c_2, c_3, c_4

$$\{c_i\} = \operatorname{argmin} \sum_{n} \sum_{\boldsymbol{u}} \| I_{\mathrm{s,\ est}}(n,\boldsymbol{u},\{c_i\}) - I_{\mathrm{s}}(n,\boldsymbol{u}) \|_2^2,$$

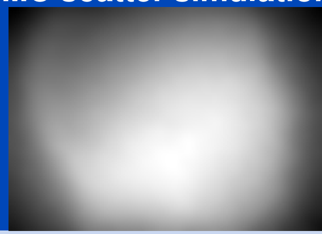
Samples of the training data set

Detector coordinate

Scatter estimate



MC scatter simulation



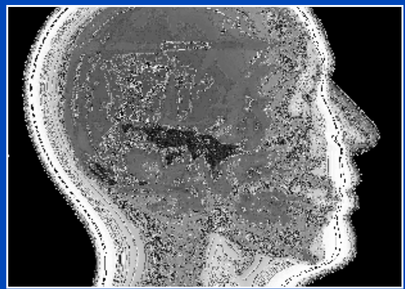
[1] B. Ohnesorge, T. Flohr, K. Klingenbeck-Regn: Efficient object scatter correction algorithm for third and fourth generation CT scanners. Eur. Radiol. 9, 563–569 (1999).

dkfz.

Ref 2: Hybrid Scatter Estimation

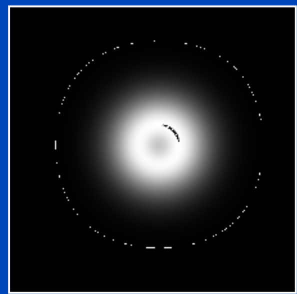
- **Hybrid scatter estimation[2] :**
  - Estimation of scatter by a convolution of the scatter source term $T(p)$ with a scatter propagation kernel $G(\boldsymbol{u}, \boldsymbol{c})$:

$$I_{\mathrm{s,\ est}}(\boldsymbol{u}) = \underbrace{\left(c_0 \cdot p(\boldsymbol{u}) \cdot e^{-p(\boldsymbol{u})}\right)}_{T(p)(\boldsymbol{u})} * \underbrace{\left(\sum_{\pm} e^{-c_1(\boldsymbol{u}\hat{\boldsymbol{e}}_1 \pm c_2)^2} \cdot \sum_{\pm} e^{-c_3(\boldsymbol{u}\hat{\boldsymbol{e}}_2 \pm c_4)^2}\right)}_{G(\boldsymbol{u},\boldsymbol{c})}$$



$T(p)(\boldsymbol{u})$

Open parameters: c_0



$G(\boldsymbol{u}, \boldsymbol{c})$

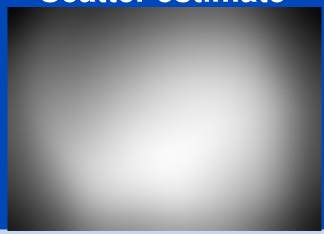
Open parameters: c_1, c_2, c_3, c_4

$$\{c_i\}_n = \mathrm{argmin} \sum_{\boldsymbol{u}} \| I_{\mathrm{s,\ est}}(n, \boldsymbol{u}, \{c_i\}) - I_{\mathrm{s}}(n, \boldsymbol{u}) \|_2^2,$$

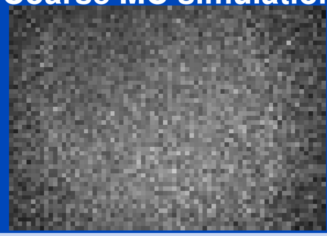
Samples of the test data set

Detector coordinate

Scatter estimate

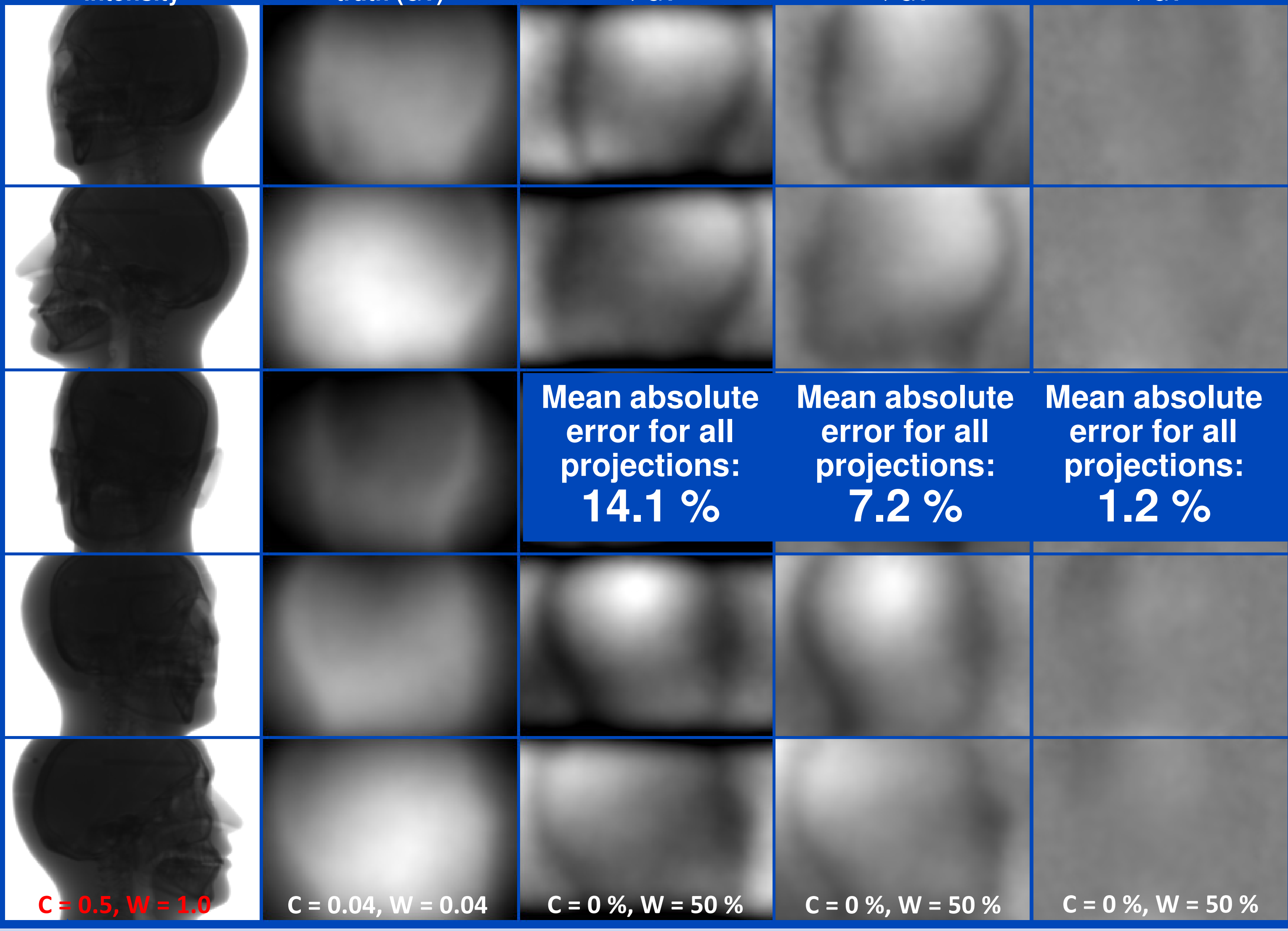


Coarse MC simulation



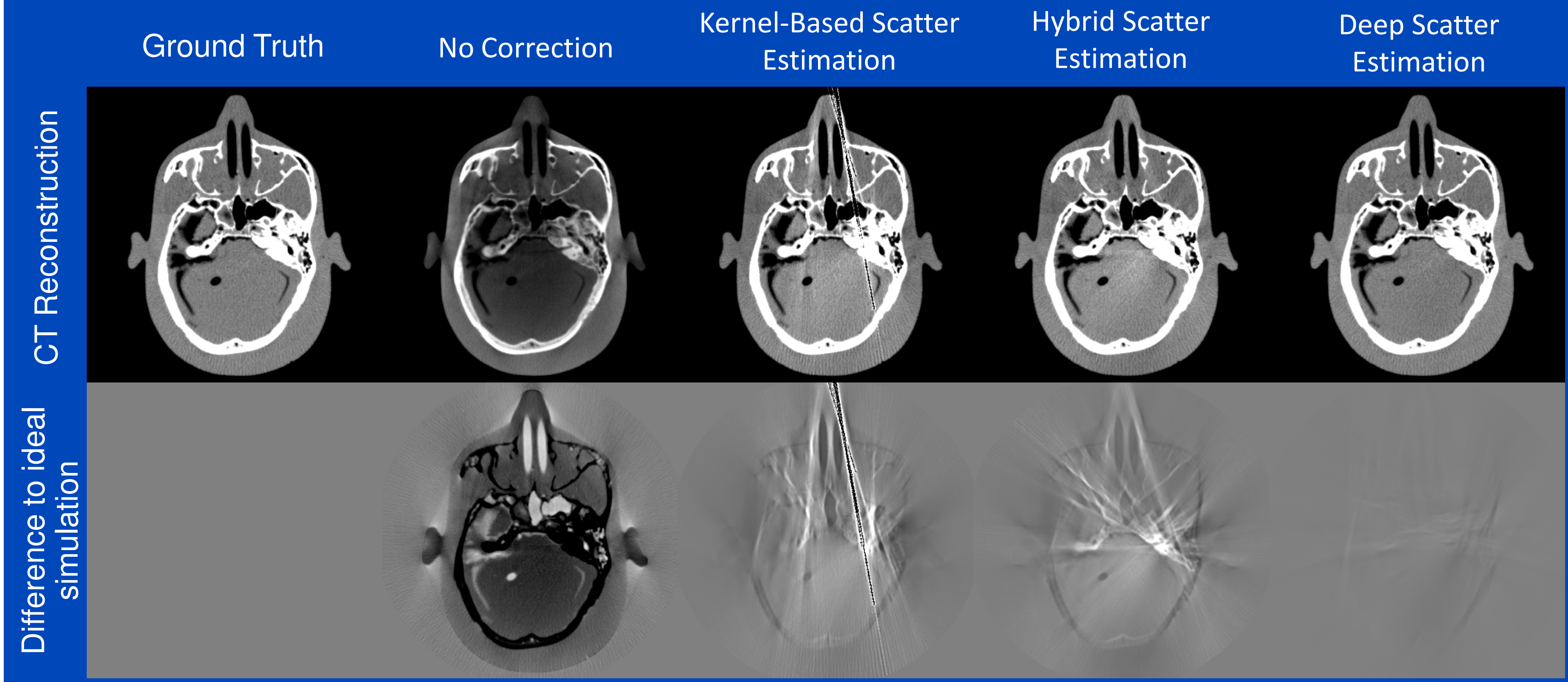
[2] M. Baer, M. Kachelrieß: Hybrid scatter correction for CT imaging. Phys. Med. Biol. 57, 6849–6867 (2012).

Results – Simulated Projection Data



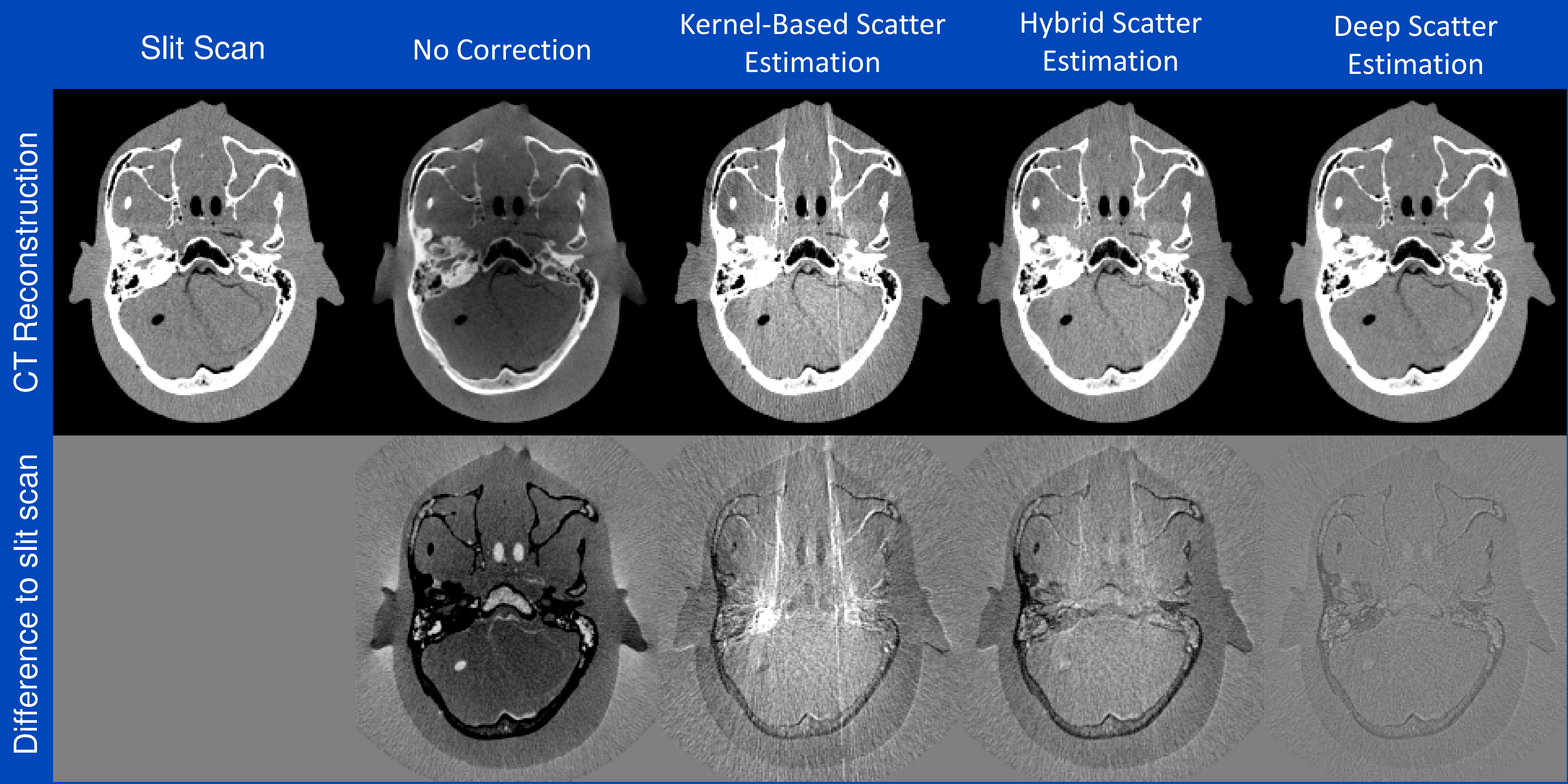
dkfz.

Results – CT Reconstructions of Simulated Data



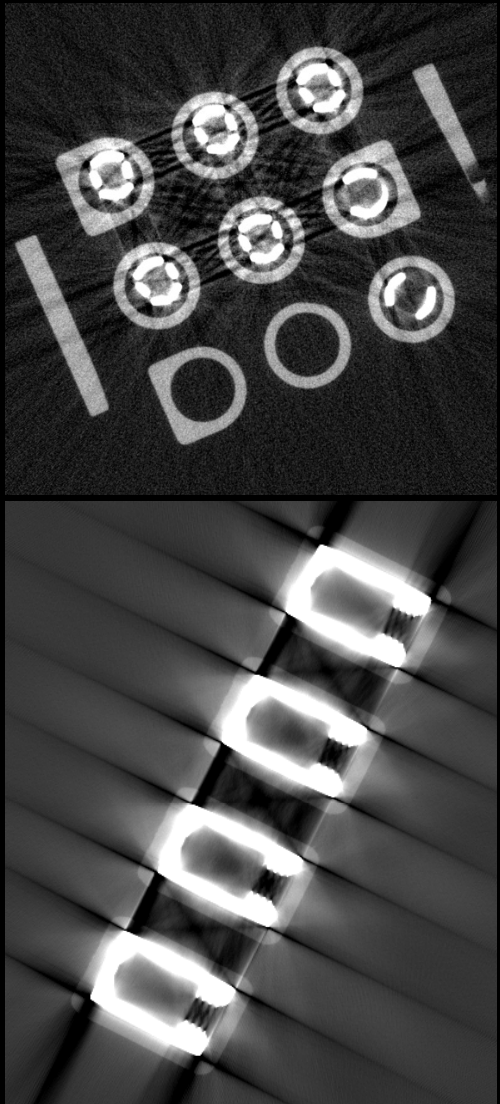
C = 0 HU, W = 1000 HU

Results – CT Reconstructions of Measured Data



C = 0 HU, W = 1000 HU

Standard reconstruction

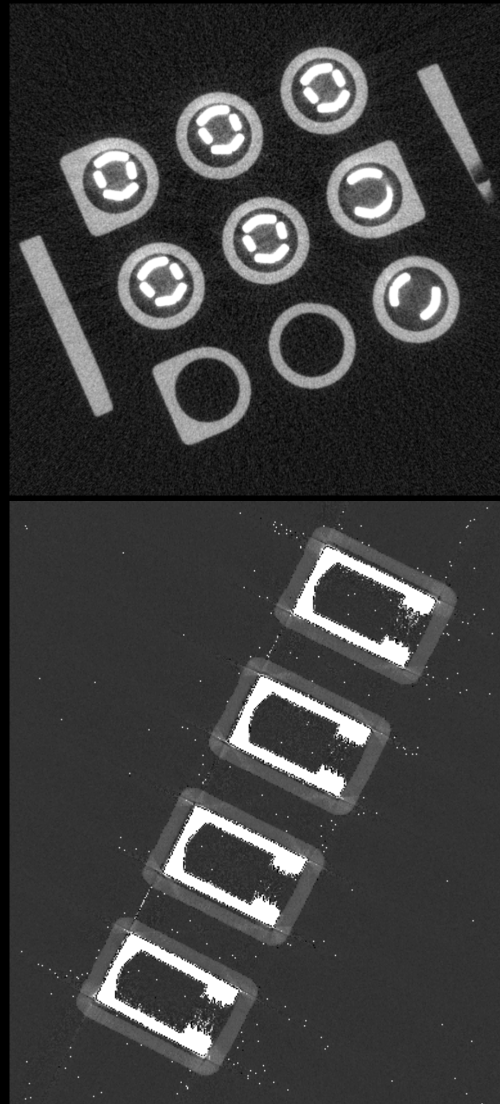


Simulation-based removal of

- beam hardening artifacts
- off-focal radiation artifacts
- focal spot blurring artifacts
- detector blurring artifacts
- **scatter artifacts**
- …

Presented at Wels 2016

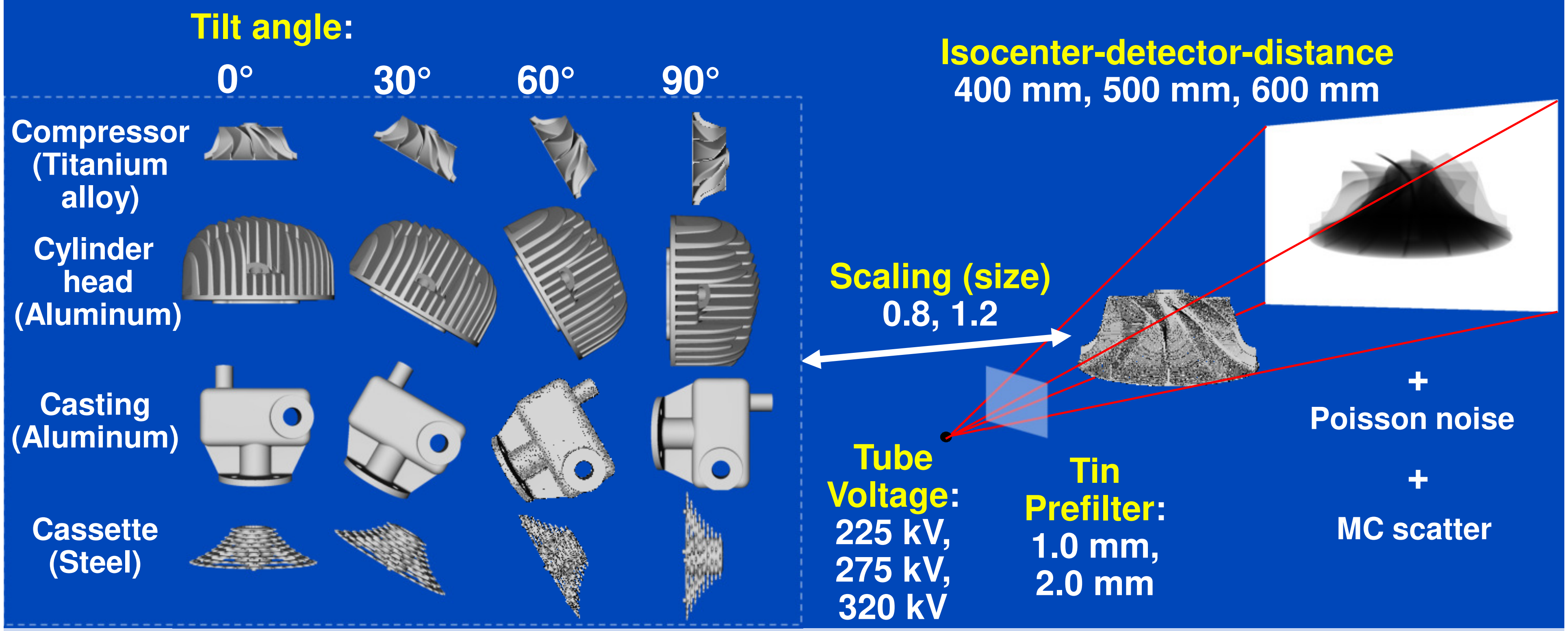
Simulation-based artifact correction



J. Maier, M. Kachelrieß et al. Simulation-based artifact correction (SBAC) for metrological computed tomography. Meas. Sci. Technol. 28(6):065011, May 2017.

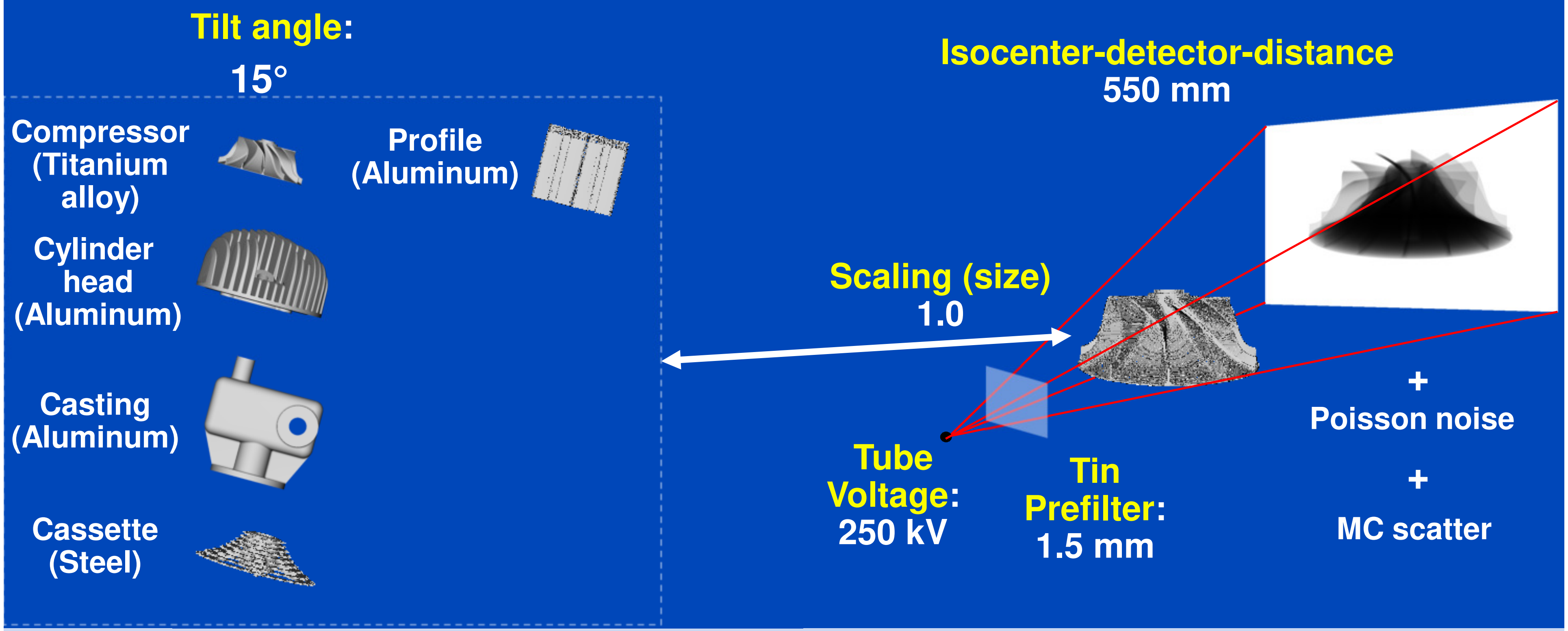
Simulation Study: Training Data

- Simulation of 16416 projections using different objects and parameter settings to train the DSE network.
- Training on a GeForce GTX 1080 for 80 epochs using the Keras framework, an Adam optimizer and a mini-batch size of 16.

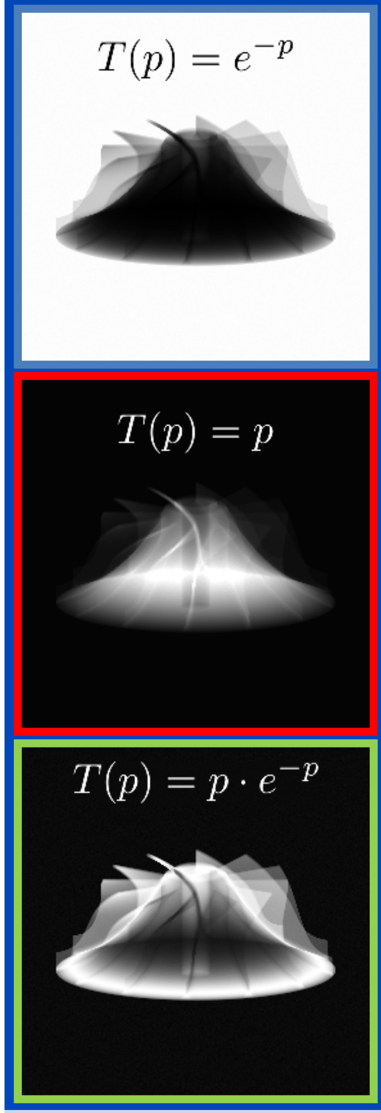


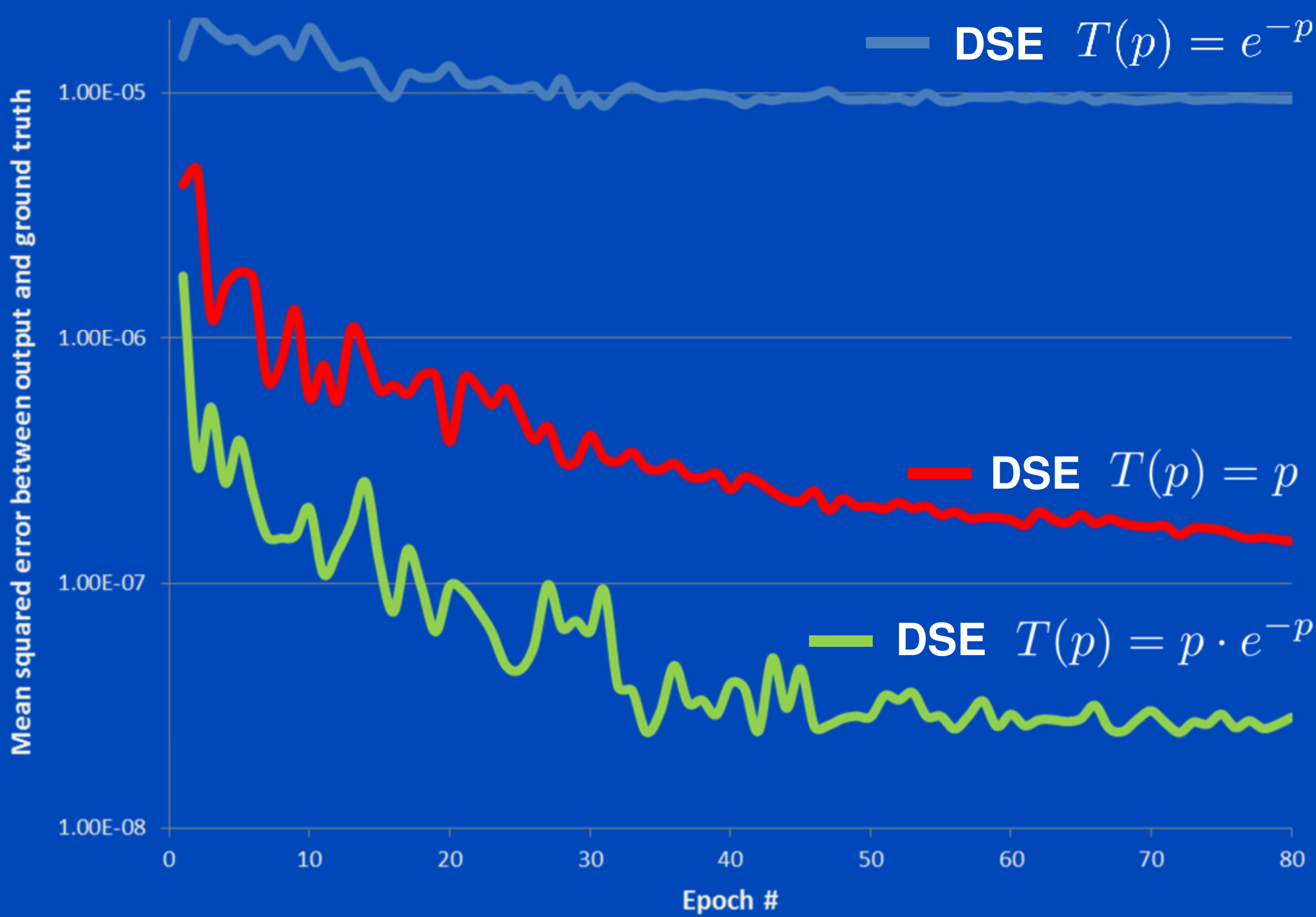
Simulation Study: Testing Data

- Simulation of a tomography (720 projection / 360°) of five components using acquisition parameters that differ from the ones used to generate the training data set.



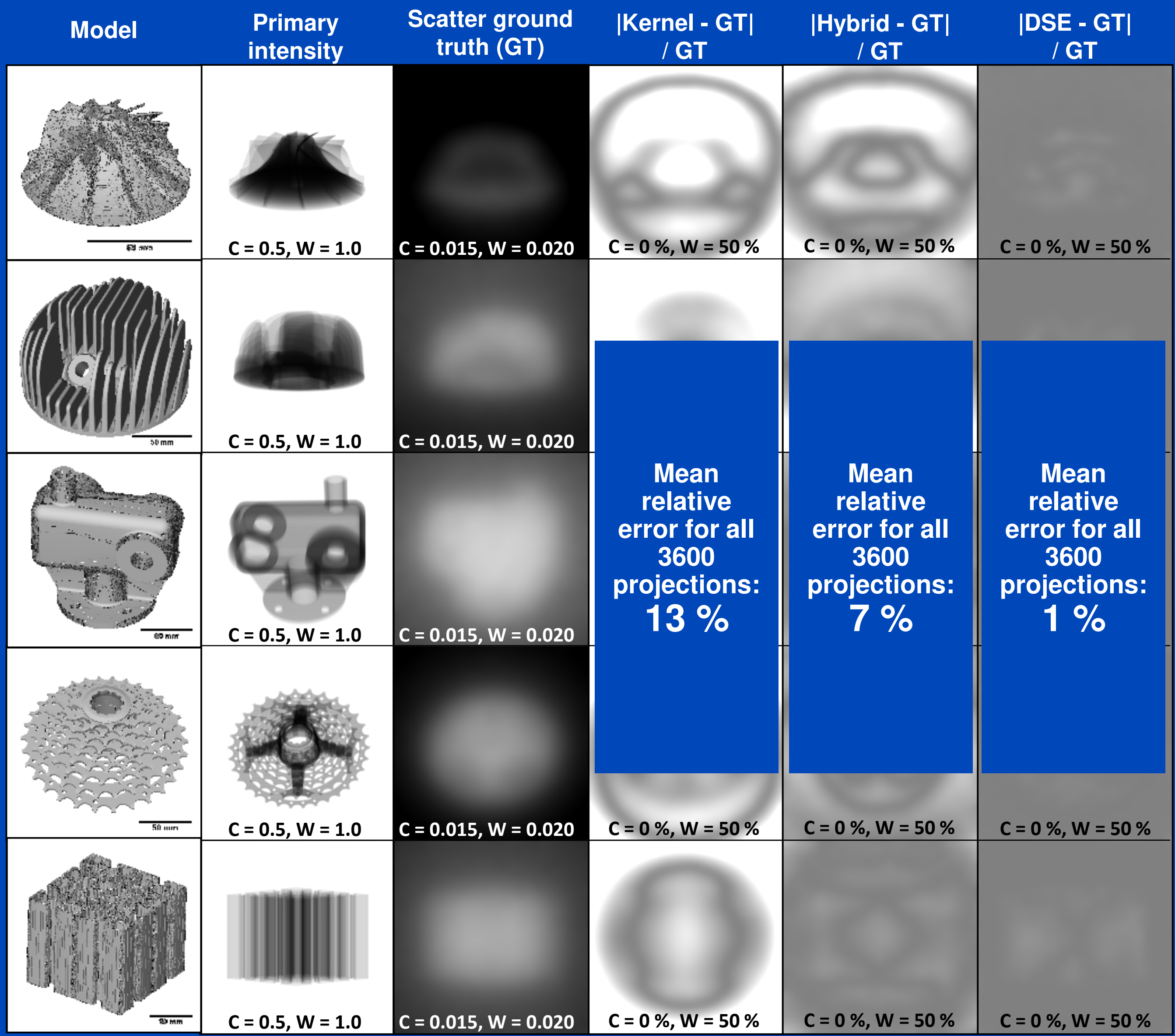
Performance on Testing Data for Different Inputs





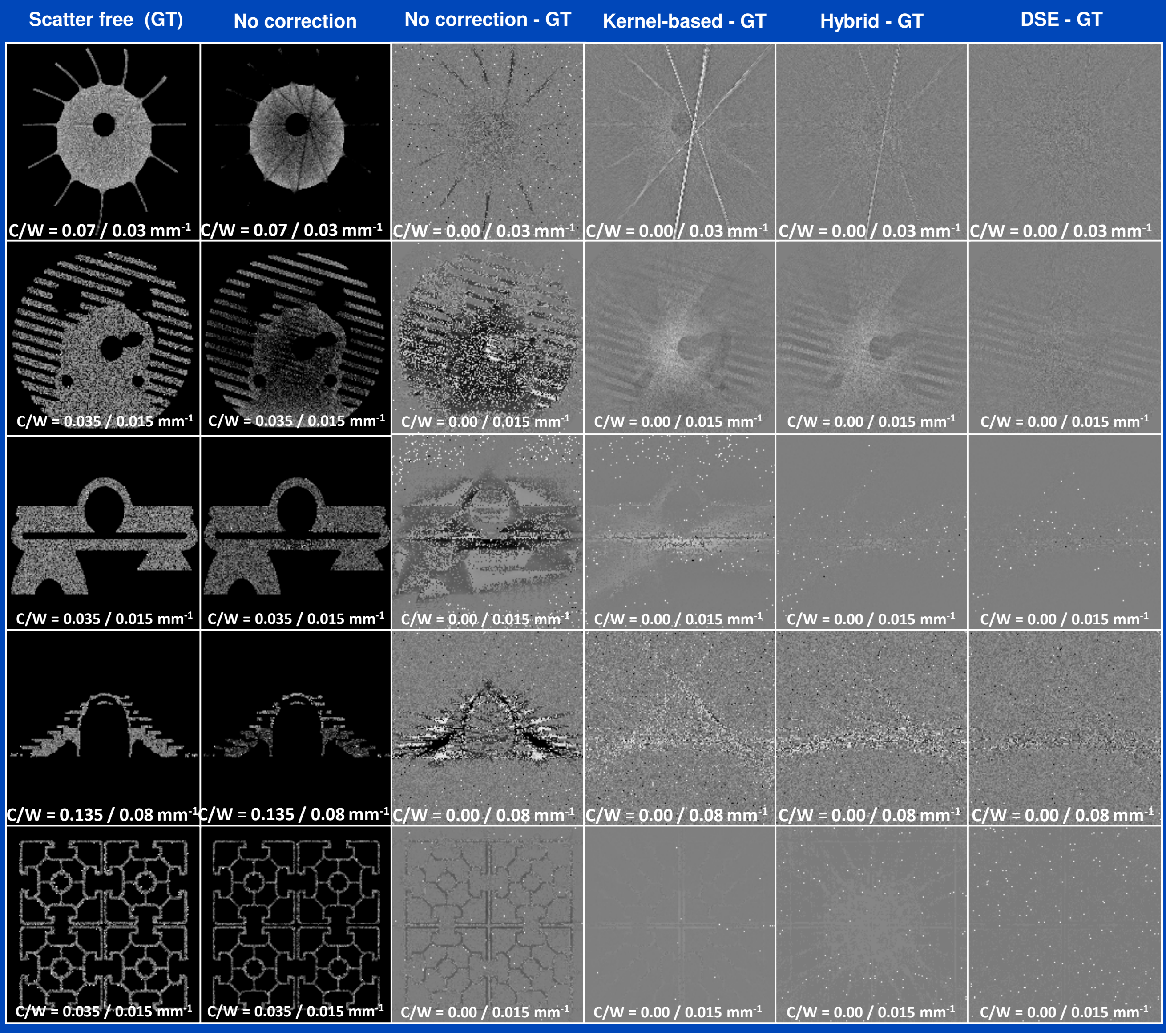
Results

Scatter estimates for simulated testing data



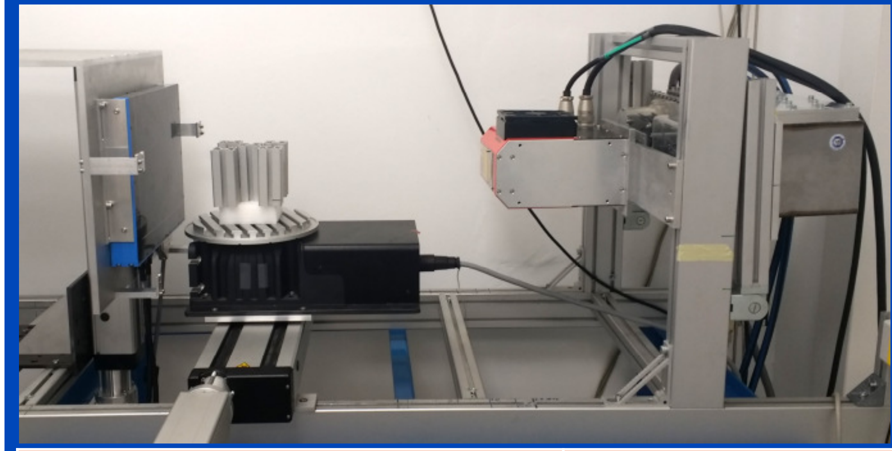
Results

CT reconstructions of scatter corrected testing data

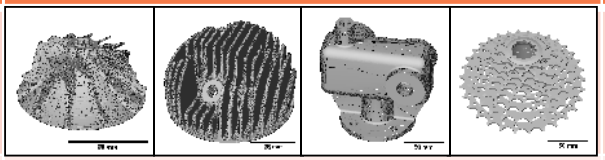
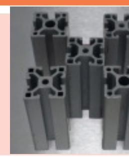


dkfz.

Application to Measured Data



- Measurement at DKFZ table-top CT
- Tomography of aluminum profile (720 projections / 360°)
- 110 kV Hamamatsu micro-focus x-ray tube
- Varian flat detector

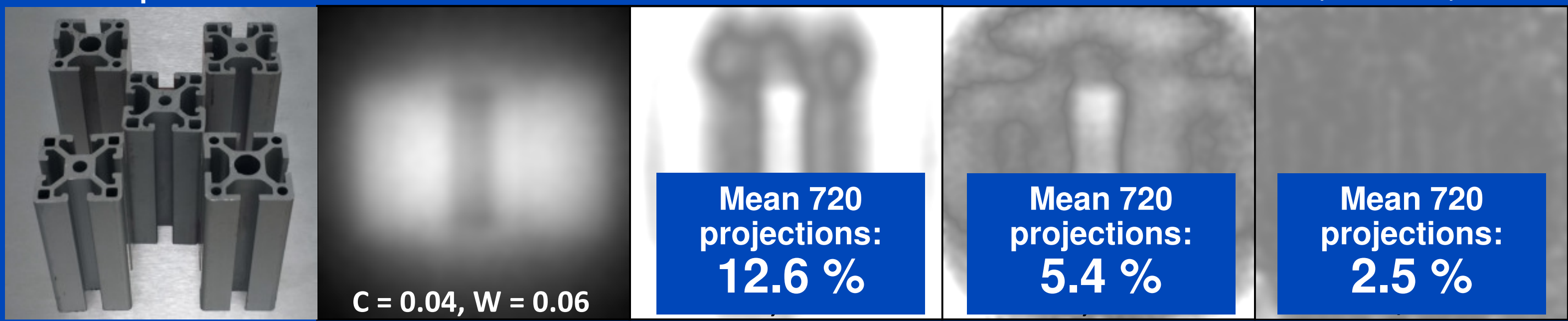
	Training	Testing
Components		
Detector elements	768×768	768×768
Source-detector distance	580 mm	580 mm
Source-isocenter distance	100 mm, 110 mm, 120 mm	110 mm
Tilt angle	0°, 30°, 60°, 90°	0°
Tube voltage	100 kV, 110 kV, 120 kV	110 kV
Copper prefilter	1.0 mm, 2.0 mm	2.0 mm
Scaling	1.0	-
Number of projections	8208	720

Results

Performance of DSE for measured data

Projection data

Component | MC scatter | |Kernel-based - MC| / MC | |Hybrid - MC| / MC | |DSE - GT| / MC

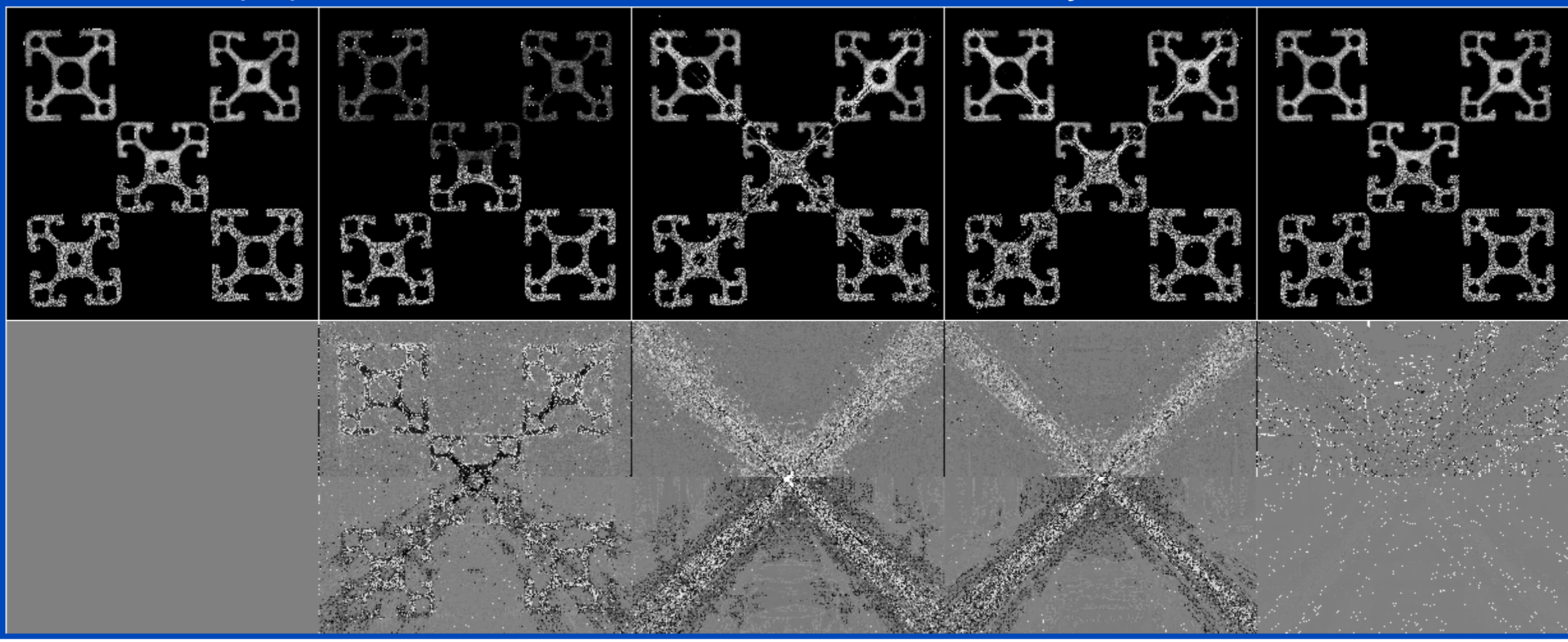


CT reconstructions

Monte Carlo (GT) | No correction | Kernel-based | Hybrid | DSE

CT reconstruction

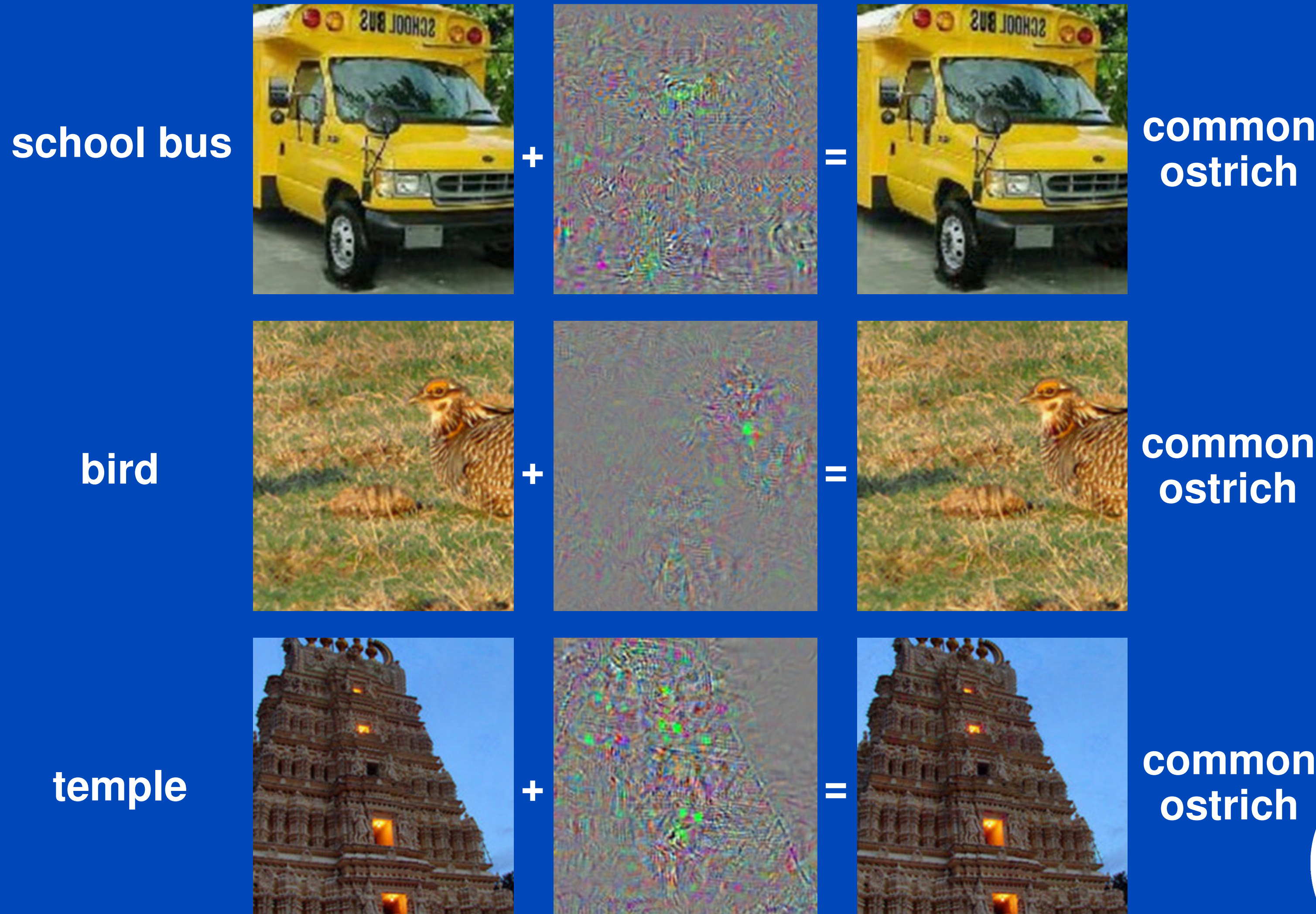
Difference to Monte Carlo



Conclusions

- DSE is a fast (~ 20 ms / projection) and accurate alternative to Monte Carlo simulation.
- DSE outperforms conventional kernel-based approaches in terms of accuracy.
- DSE is not restricted to reproduce only Monte Carlo scatter estimates but can be used with any other scatter estimate.

Adversarial Example



C. Szegedy et al., Intriguing properties of neural networks, arXiv 2014

Thank You!

This presentation will soon be available at www.dkfz.de/ct.

Job opportunities through DKFZ’s international PhD or Postdoctoral Fellowship programs (marc.kachelriess@dkfz.de).

Parts of the reconstruction software were provided by RayConStruct® GmbH, Nürnberg, Germany.