Jonas Glombitza
jonas.glombitza@fau.de

9th Institute of Space Sciences Summer School:
Multimessenger Astrophysics

https://github.com/DeepLearningForPhysicsResearchBook/deep-learning-physics

Convolutional Neural Networks

I. Processing image-like data

II. Incorporating symmetries into DNNs

Time schedule for the next days

Tutorial: Introduction to deep learning
Monday 1h

- Training of deep neural networks

Hands-on
Monday 2:30h

- Interactive training of neural networks
- machine learning frameworks: Keras / TensorFlow
- Implementation of linear regression and fully-connected networks

Lecture
Tuesday 2:00h

- Questions + Convolutional neural networks

Advances in deep learning:
Tuesday 1:00h

- Transformer networks
- Large Language Models

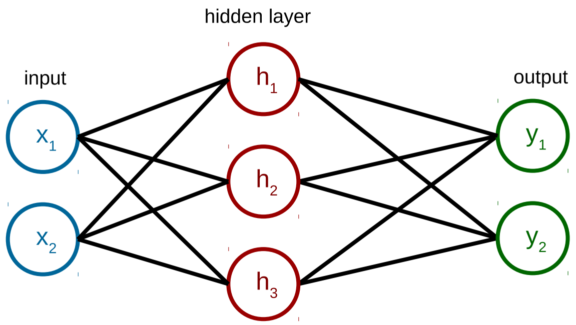
Set up & Requirements:
https://bitly.cx/iHcxS & https://bit.ly/3pyXRii
we will use **Jupyter Notebooks** and Keras / TensorFlow
we will use **Google Colab** → Google Account required

Recap: Neural Networks

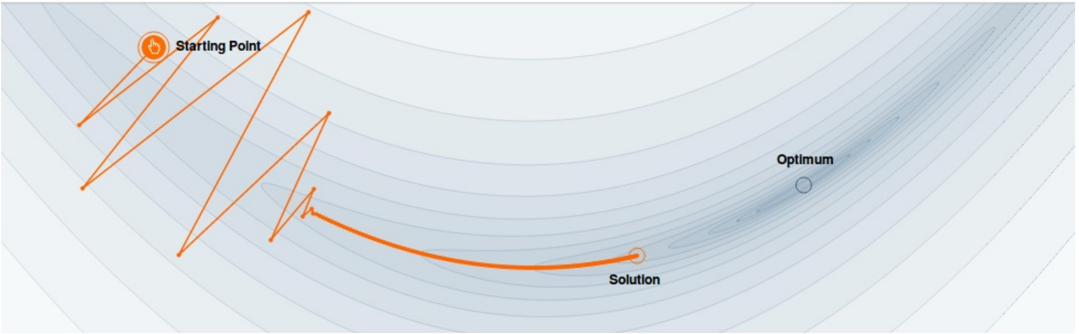
- Typical Machine Learning task
  - Labeled Data (x, y)
  - Model with adaptive weights $y_m(x, \theta)$
  - Objective $J(\theta)$
  - Optimization procedure

- Neural Network $y = \sigma(Wx + b)$
  - Matrix multiplication (adaptive superposition of features)
  - Add of bias
  - Nonlinear activation

- Deep Learning is form of representation learning
  - Stack multiple layers for increasing feature hierarchy



Recap: Optimization



Why Momentum Really Works, Distill

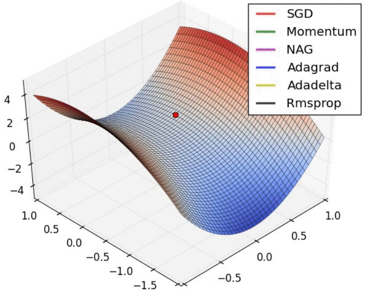
Mini-batch training: Use **stochastic gradient descent** algorithm (SGD)

Momentum: Use past gradients (velocity)

- Faster convergence by **damping oscillations** and increasing the step size for more informative gradients

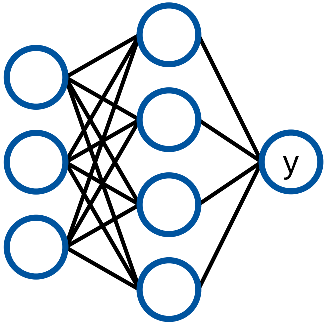
Adaptive learning rate: Scaling using past gradients

- Use adaptive learning rate for each parameter

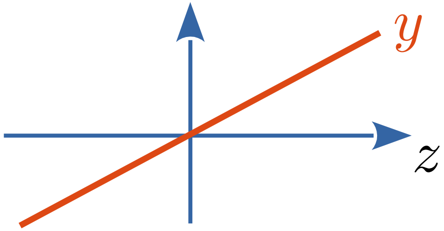


Classification vs. Regression

Regression



Linear

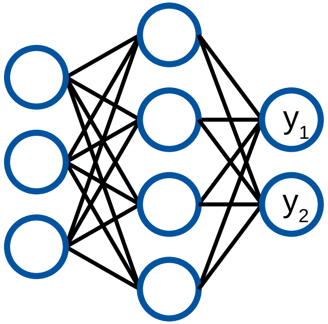


no activation function

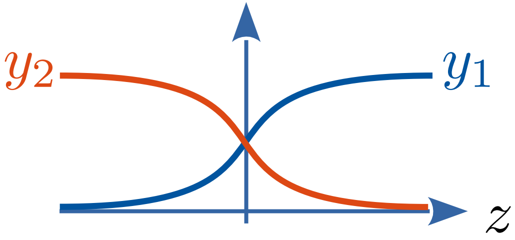
Minimize mean-squared-error

$$J(\theta) = \frac{1}{n} \sum_i [y_i - y_m(x_i)]^2$$

Classification



Softmax



$$y_j(z) = \frac{e^{z_j}}{\sum_i e^{z_i}}$$

Minimize cross entropy

$$J(\theta) = -\frac{1}{n} \sum_i y_i \log[y_m(x_i)]$$

Metrics: Classification

ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

FAU

Classification

$$J(\theta) = -\frac{1}{n}\sum_i y_i \log[y_m(x_i)]$$

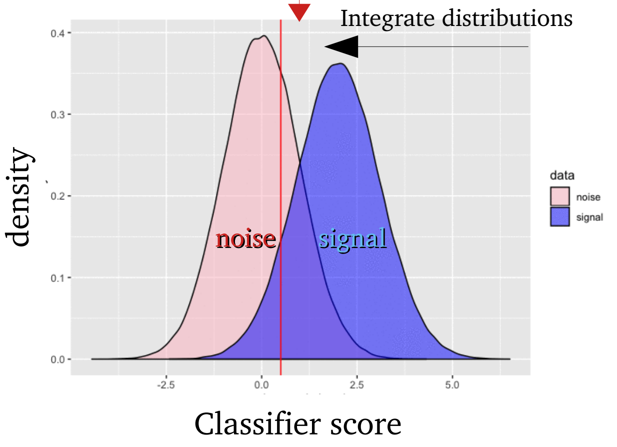
Minimize cross entropy

Metrics:

Accuracy: fraction of correction predictions

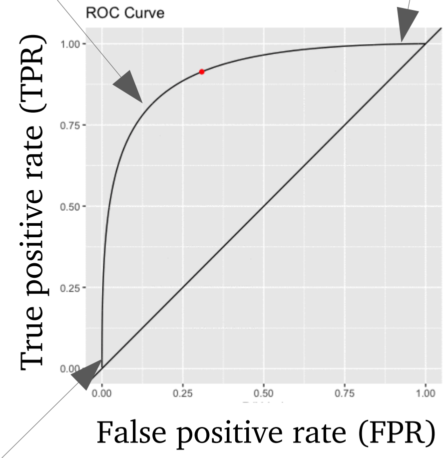
Better: Receiver Operation Characteristic (ROC)

Only accept events above threshold
scan threshold → RoC



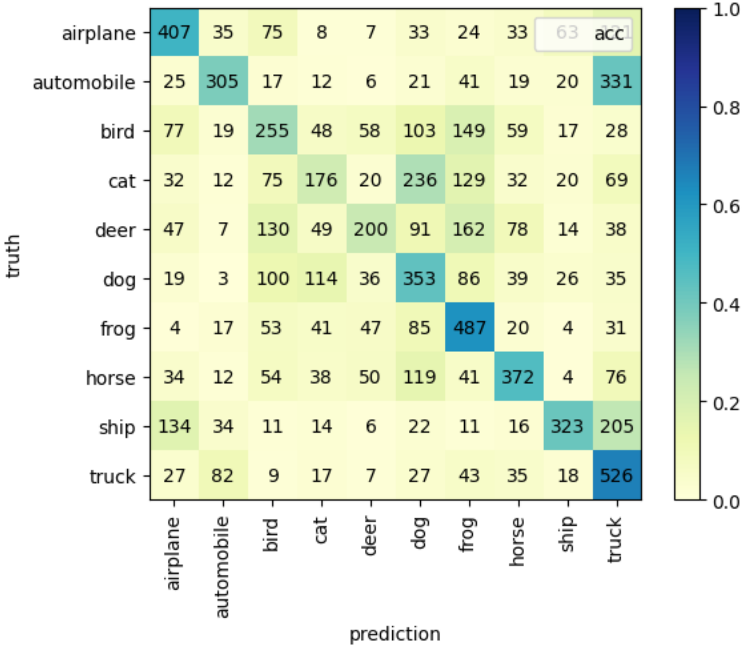
Select most signal / less noise
TPR= large, FPR=small

Select almost everything
TPR~1, FPR~1



Select almost nothing
TPR~0, FPR~0

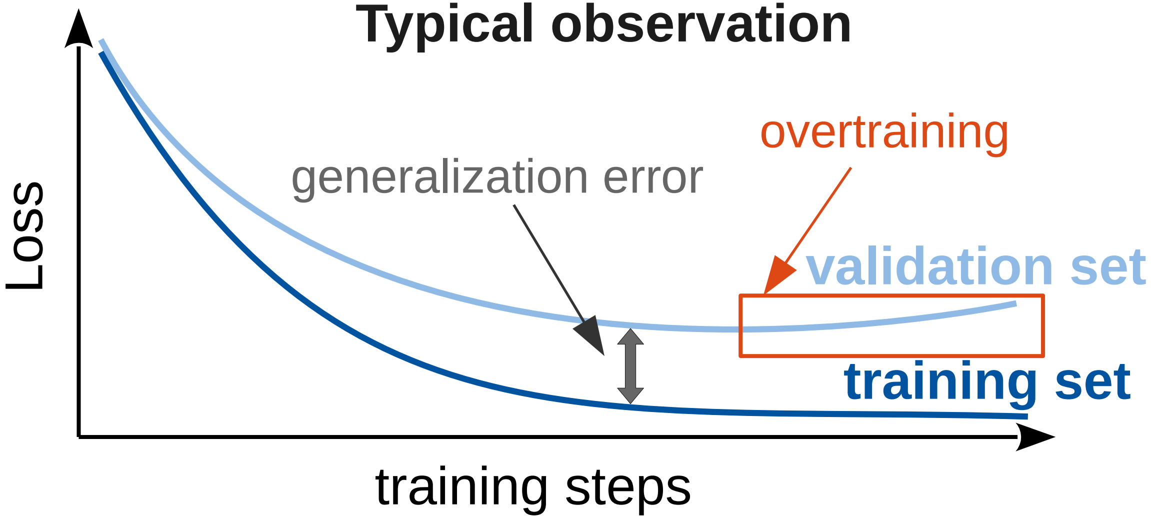
Multi-class classification
→ no ROC possible

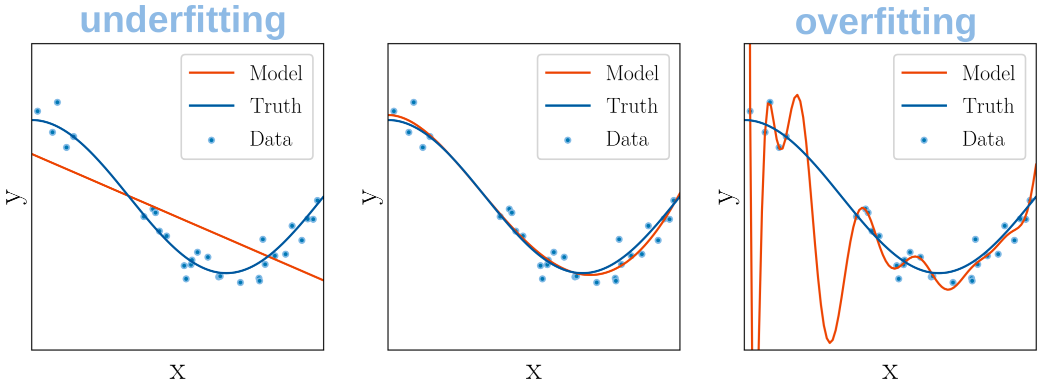


Recap: Under- and Overtraining

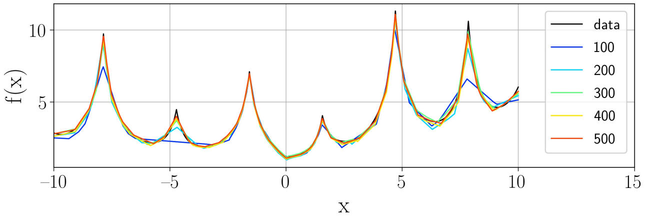
- Split data into 3 sets
  - training
  - test
  - validation
- What is a reasonably split?

- During training monitor the loss separately for training and validation set
  - check for overtraining!
  - if loss stops decreasing
    - reduce learining rate
    - stop training





“A feed-forward network with a linear with a finite number of nodes can approximate any reasonable function to arbitrary precision.”



Is the simple feed-forward network the ultimate AI building block?

(a) Yes, as proven by the universal approximation theorem

(b) Practically not; You know theory and theorists...

(c) Please don’t talk the next 3 hours about
feed-forward networks only...

Dedicated time for

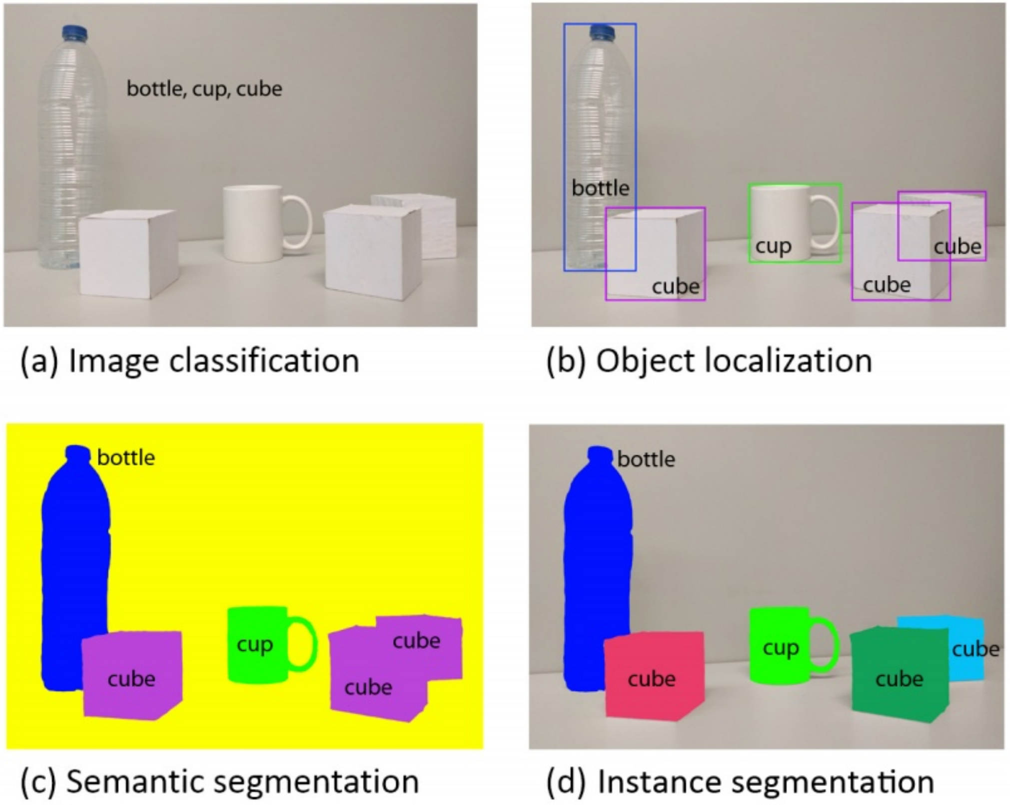
Your Questions

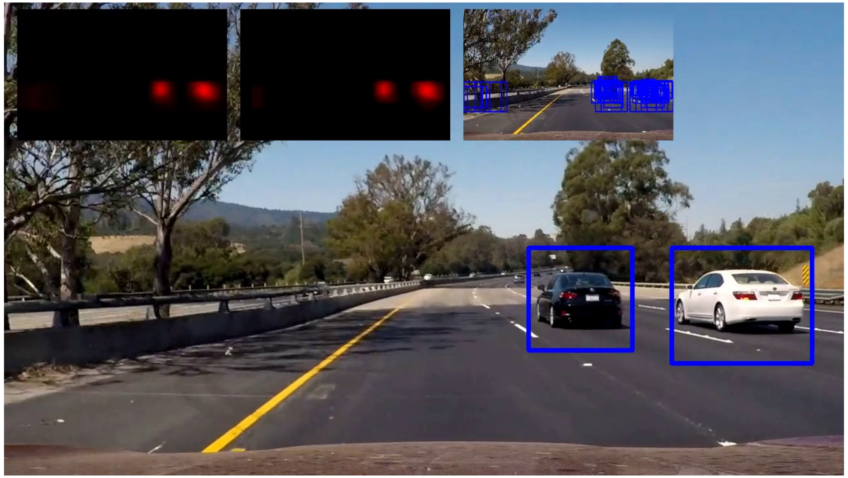
Natural Images

Automate task for humans, very challenging for machine learning models:

- High dimensional input (up to millions of pixels)
- Many possible classes depending on task
- Multiple variations
  - Viewing angle, light conditions, deformation, object variations, occlusions….

Computer Vision Tasks





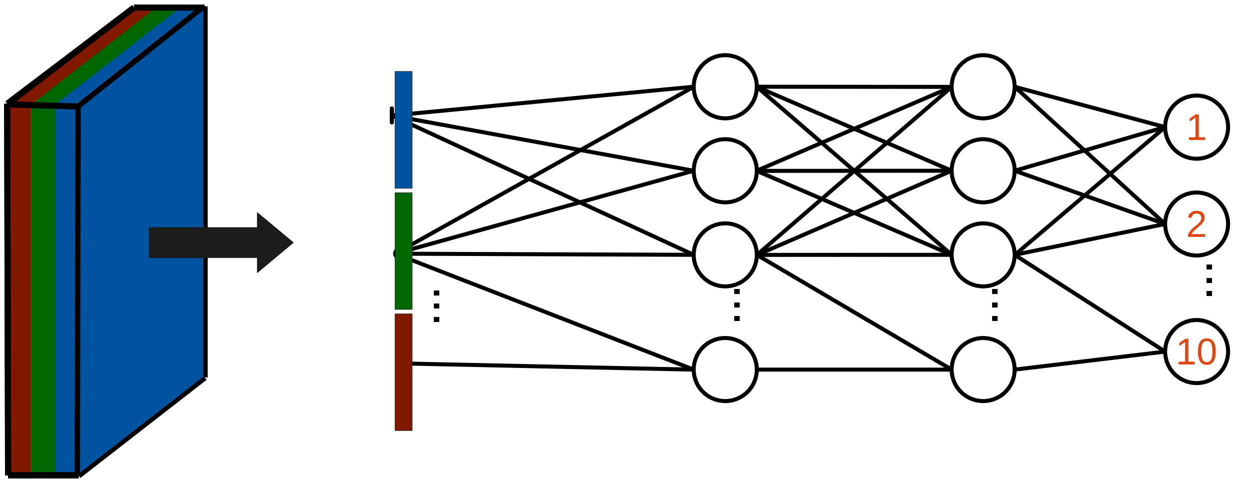
Fully Connected Network

- Input layer: Flatten image to 32 x 32 x 3 = 3072 vector
- Fully connected: every pixel connected with each other

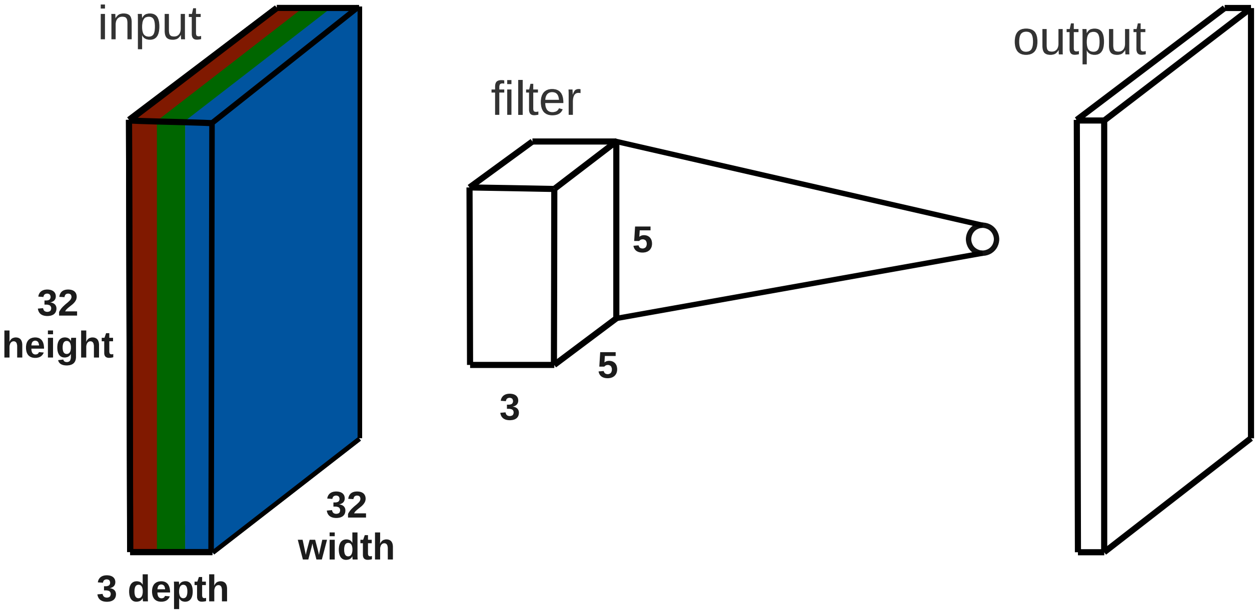
✗ Huge number of adaptive parameters per layer

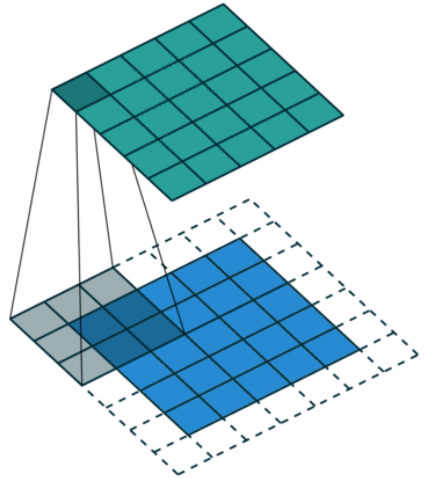
✗ No use of translational variance

✗ No prior on local correlations



2D Convolutional Neural Networks

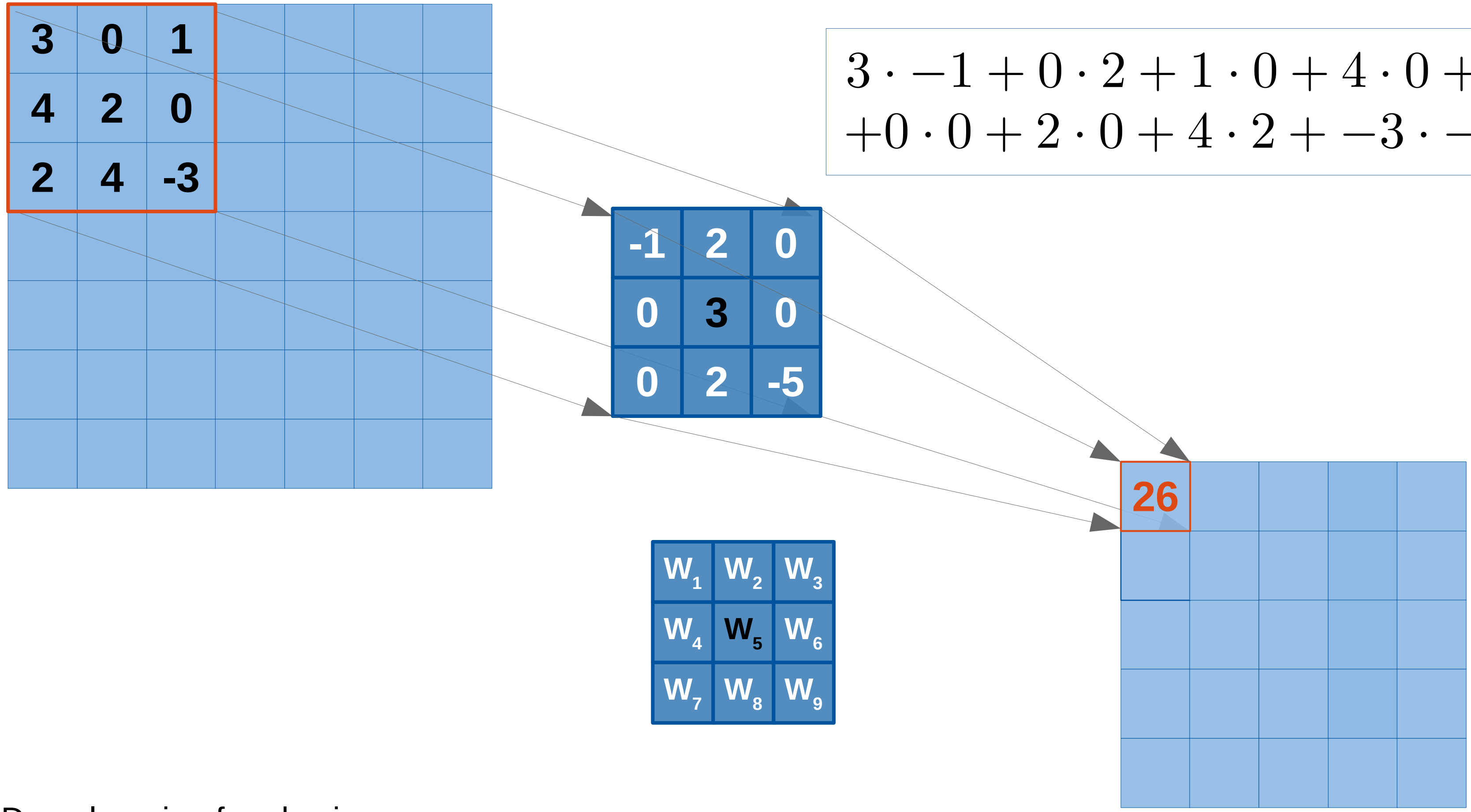




Paul-Louis Pröve,
Towards Data Science

- Consider input volume (width x height x depth), e.g., 3 color channels
- Use convolutional filter with smaller width and height but same depth
- Slide filter over the entire volume and calculate linear transformation to get one output value for each position

Convolutional Operation



$$3 \cdot -1 + 0 \cdot 2 + 1 \cdot 0 + 4 \cdot 0 + 2 \cdot 3$$
$$+0 \cdot 0 + 2 \cdot 0 + 4 \cdot 2 + -3 \cdot -5 = 26$$

Convolutional filters

hand-designed filters

Edge

-1	-1	-1
-1	8	-1
-1	-1	-1

Diagonal edge

-1	-1	2
-1	2	-1
2	-1	-1

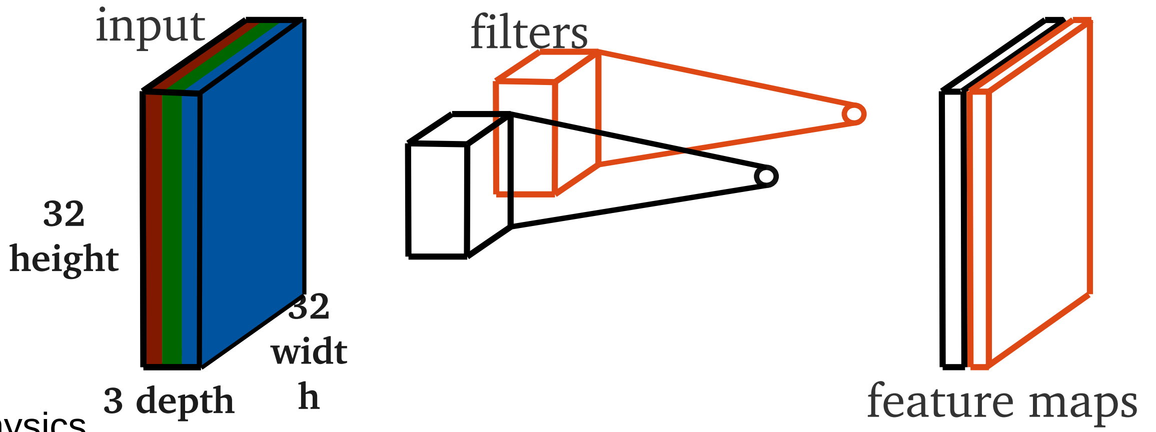
deep learning

Convolutional Networks

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

adaptive parameters

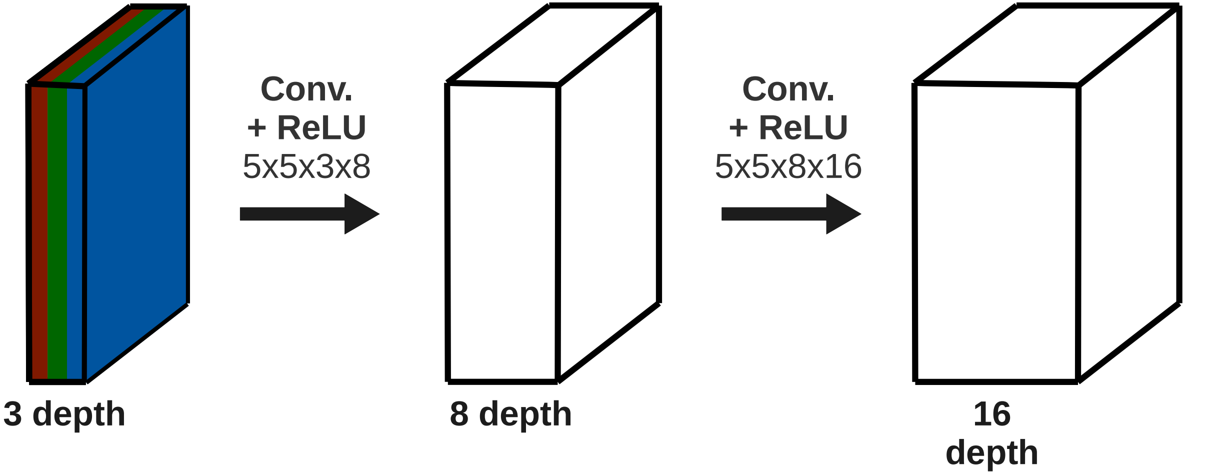
- scan input image for the presence of specific feature using **filters**
- use multiple filters and stack the results as **feature maps** (depth-wise stacking)

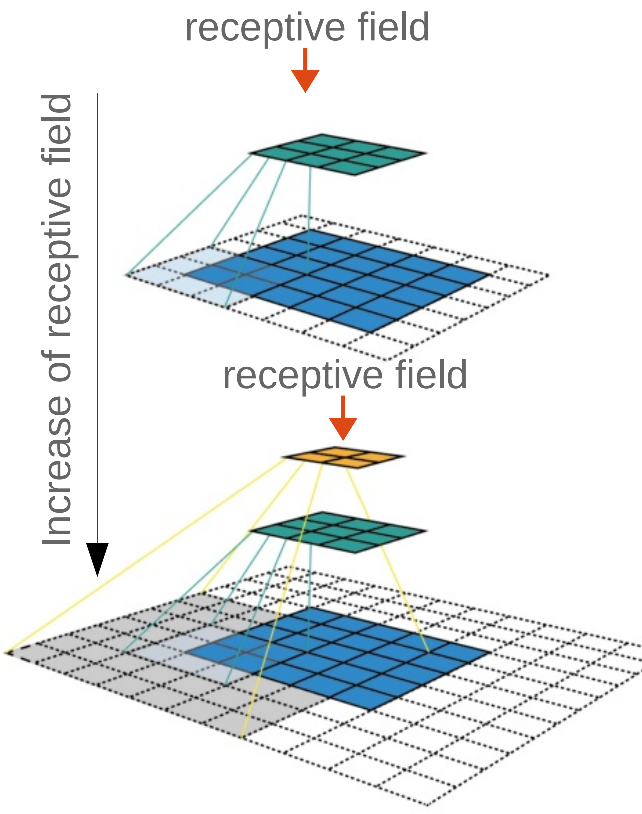


2D Convolutional Operation

Stack multiple convolutional layers + activations

- Each convolution acts on feature map of previous layer
- Increasing feature hierarchy
- Increasing of receptive field



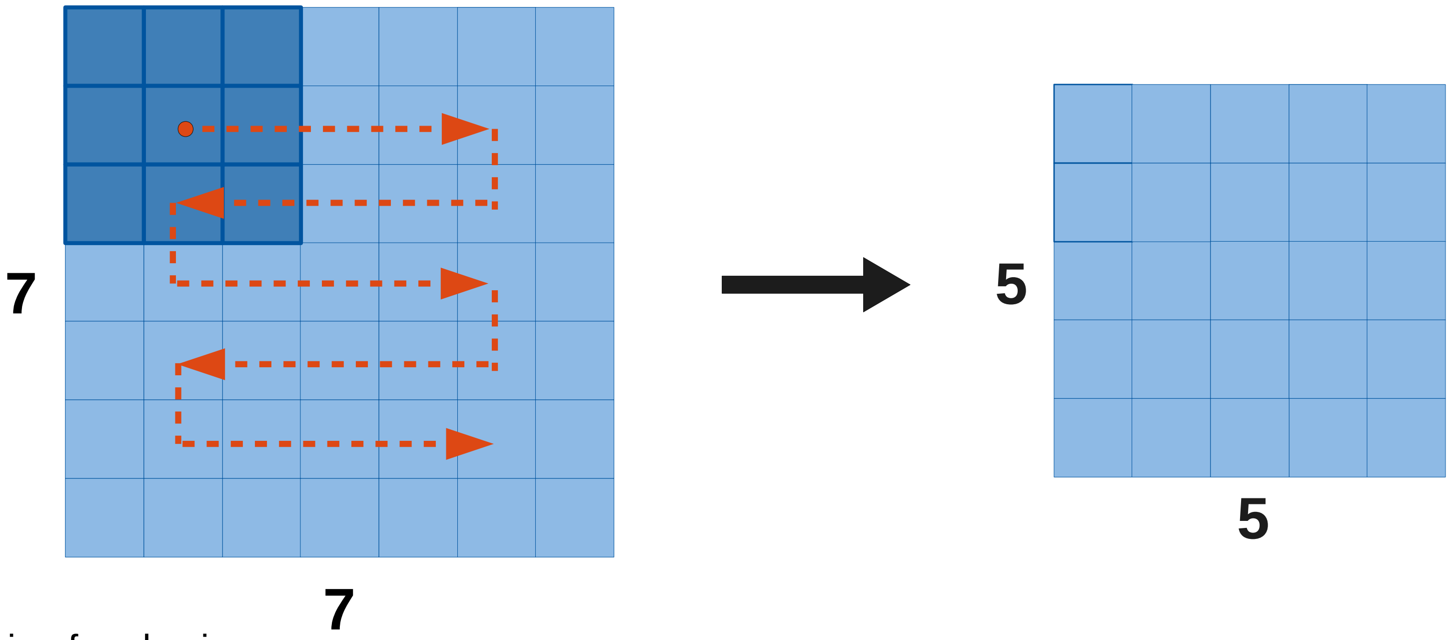


Spatial Output Size

Standard convolution reduces the output size due to extent of the filter

- Sets upper bound to the number of convolutional layers

- **Example:** Convolution with 3 x 3 filter

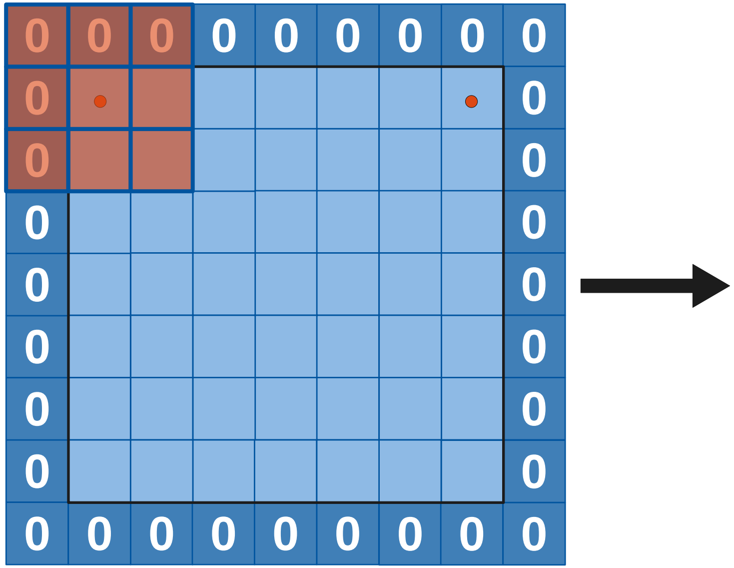


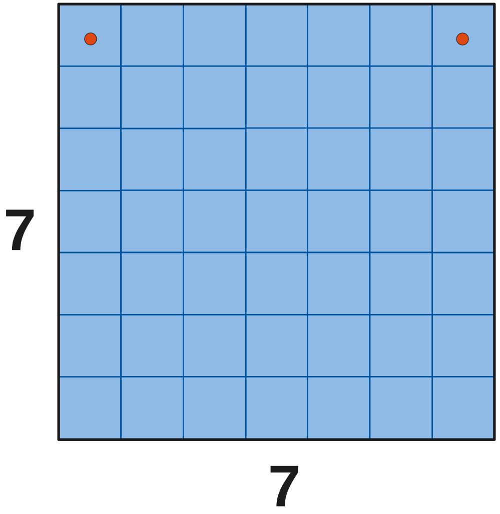
Padding

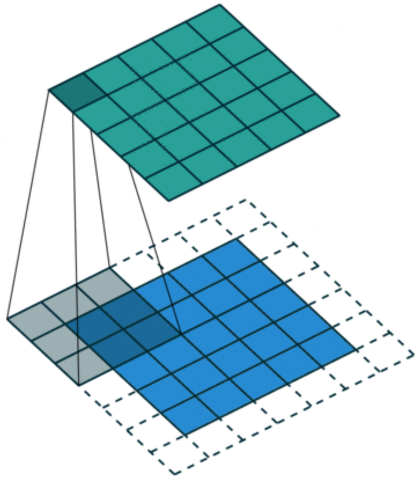
Add zeros around image borders to conserve the spatial extent of the input

- Prevents fast shrinking of the network input

- **Example:** Convolution with 3 x 3 filter and padding







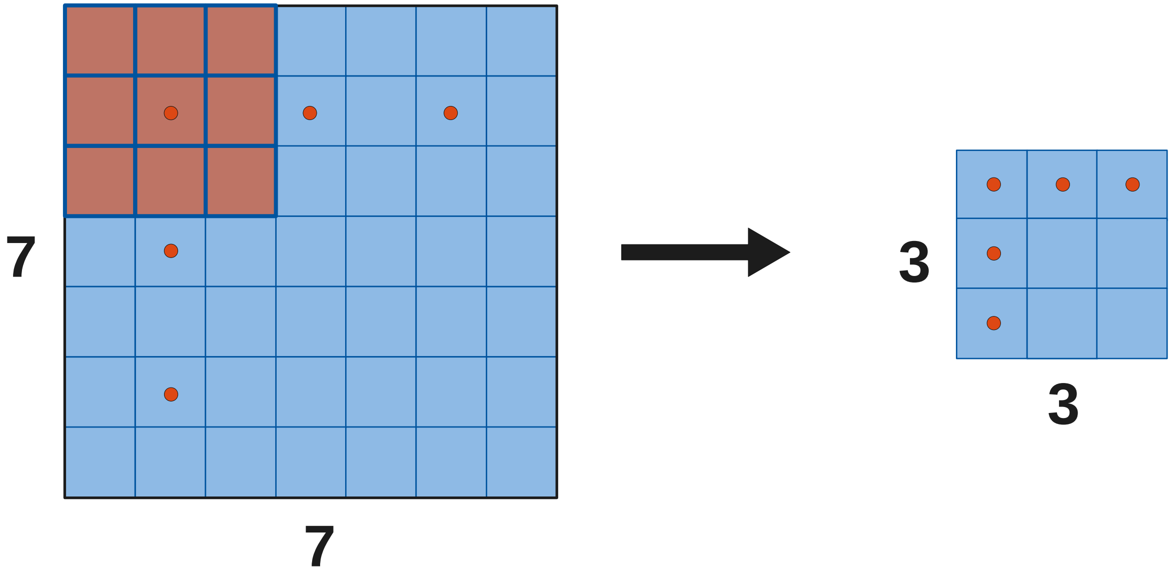
Paul-Louis Pröve,
Towards Data Science

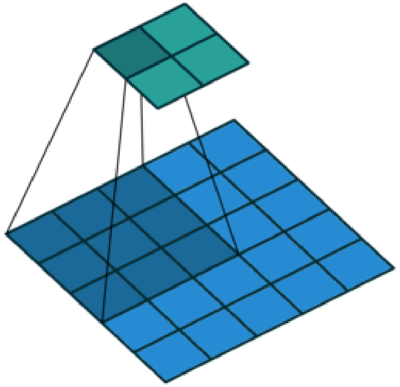
Striding

Using a larger stride when sliding over the input, reduces the output size

- Useful for switching to smaller image sizes / larger scales

- **Example:** Convolution with 3 x 3 filter and stride of 2





Paul-Louis Pröve,
Towards Data Science

Pooling

Sub-sample the input to reduce the output size

- Used to merge semantically similar features
- Make network invariant to small translations or perturbations

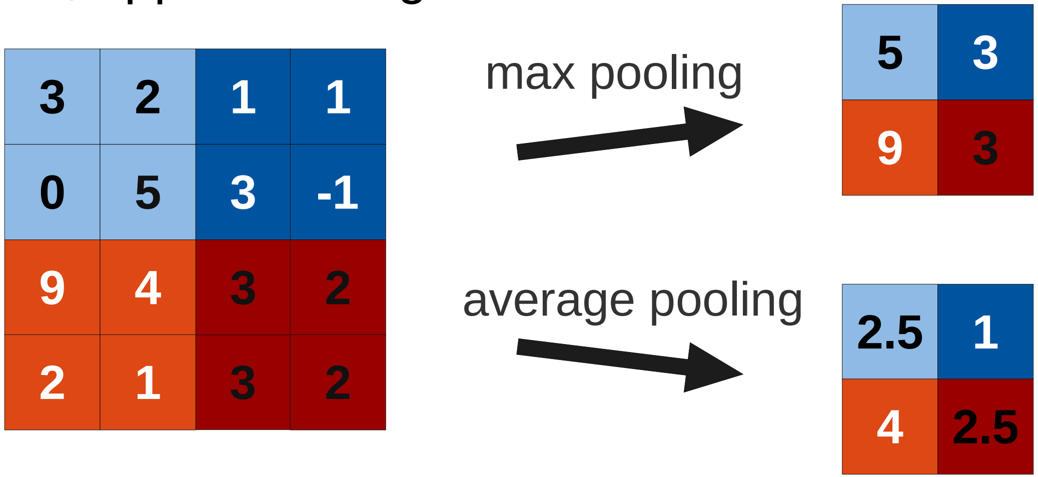
Average pooling: Take the mean of each patch → for some regressions preferable

Max pooling: Take the maximum of each patch

→ in practice often better performance, applies stronger constraint

- **Typical Pooling:**
  Pooling using 2 x 2 patches
  and a stride of 2

- **Overlapping Pooling:**
  3 x 3 patches with stride of 2

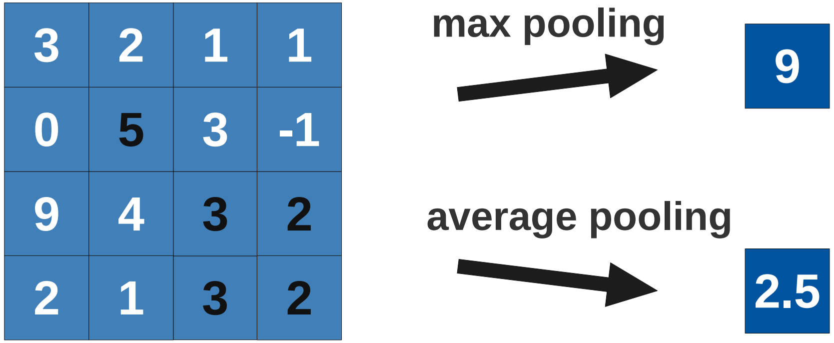


Global Pooling Operation

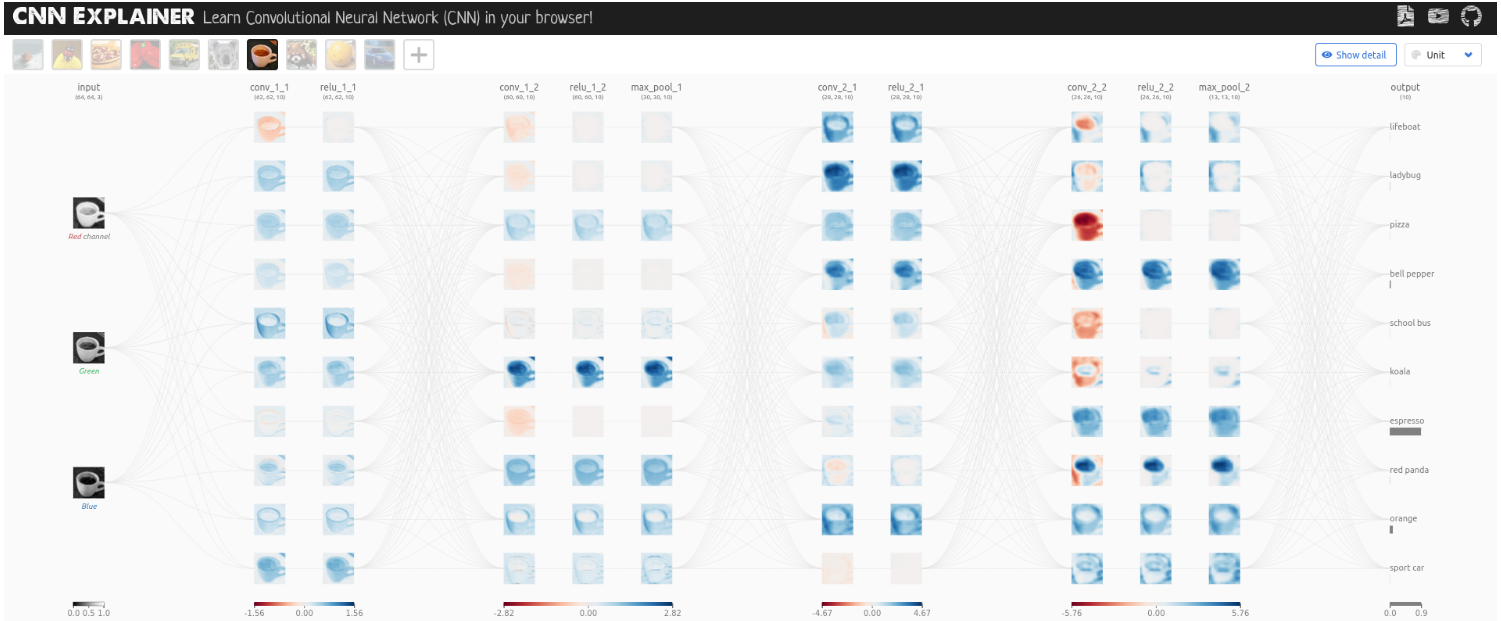
- Take maximum/average over complete image → usually second last layer
- Replace fully connected layers
  - Saves parameters in later layers of the models → prevent overfitting
- Can be seen as regularizer
  - Fully connected transformation matrix with diagonal shape
- Enforcing correspondences between feature maps and categories
- Allows object detection in the input space

“The pooling operation used in convolutional neural networks is a big mistake, and the fact that it works so well is a disaster”

- Geoffrey Hinton



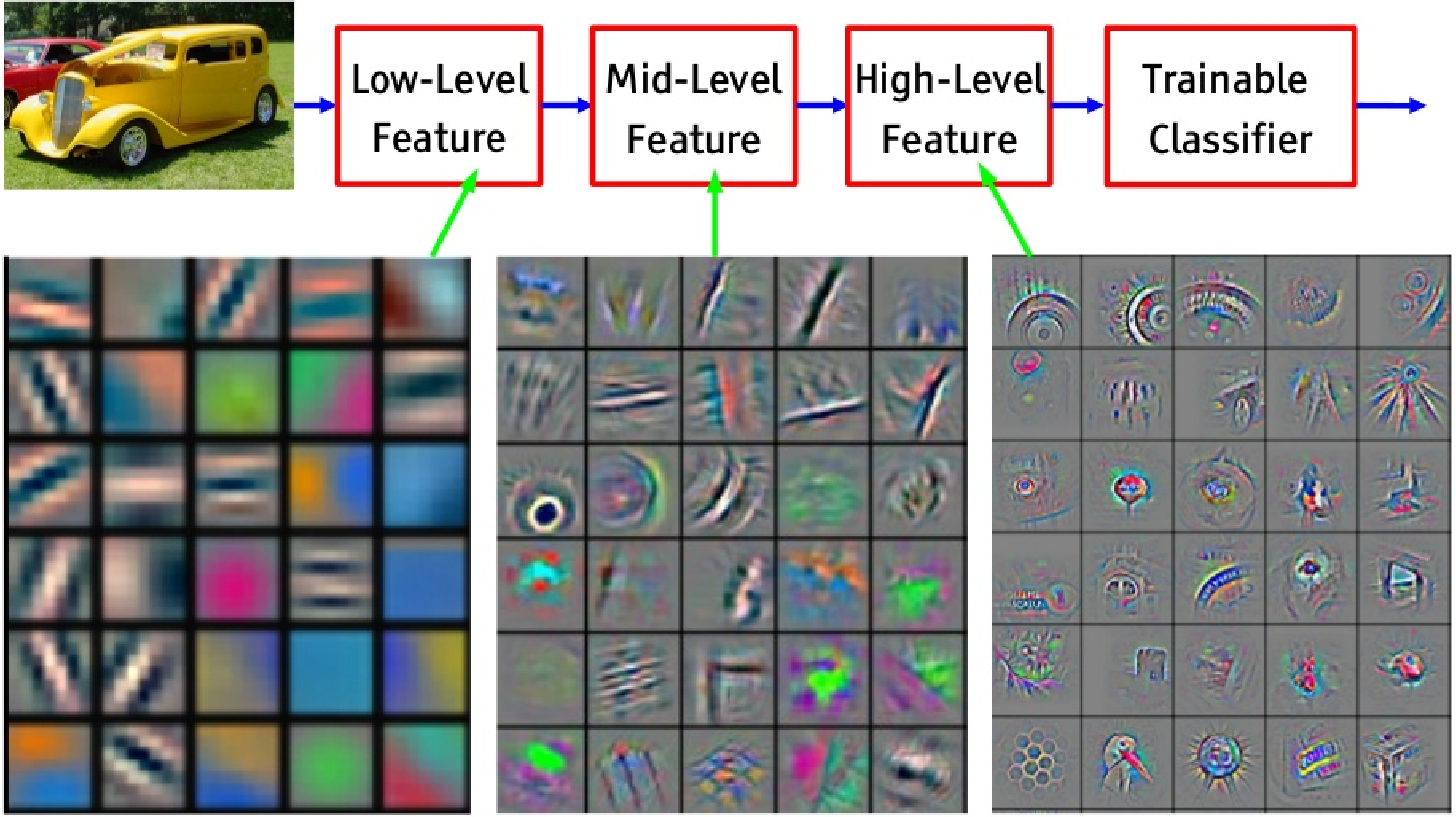
Visualization of CNNs – 5 mins



- Exercise: → please use the "espresso example"
  - What is the CNN (likelihood) prediction of the image to be classified as an orange?
  - Which feature in the first layer is almost useless for the espresso classification?
  - What kernel size is used? Is striding used?
  - Is padding used in the convolutional operation not?

https://poloclub.github.io/cnn-explainer/

Feature Hierarchy



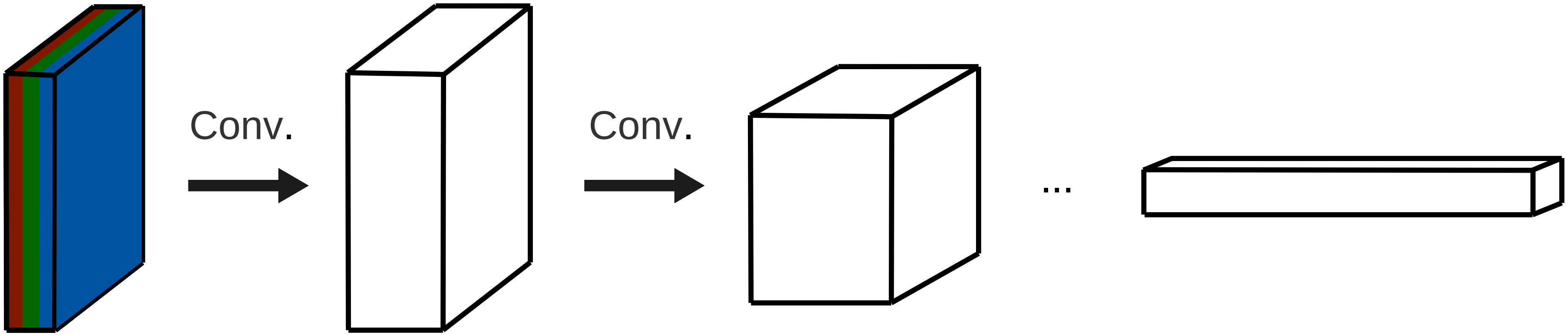
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

https://arxiv.org/abs/1311.2901

Convolutional Pyramid

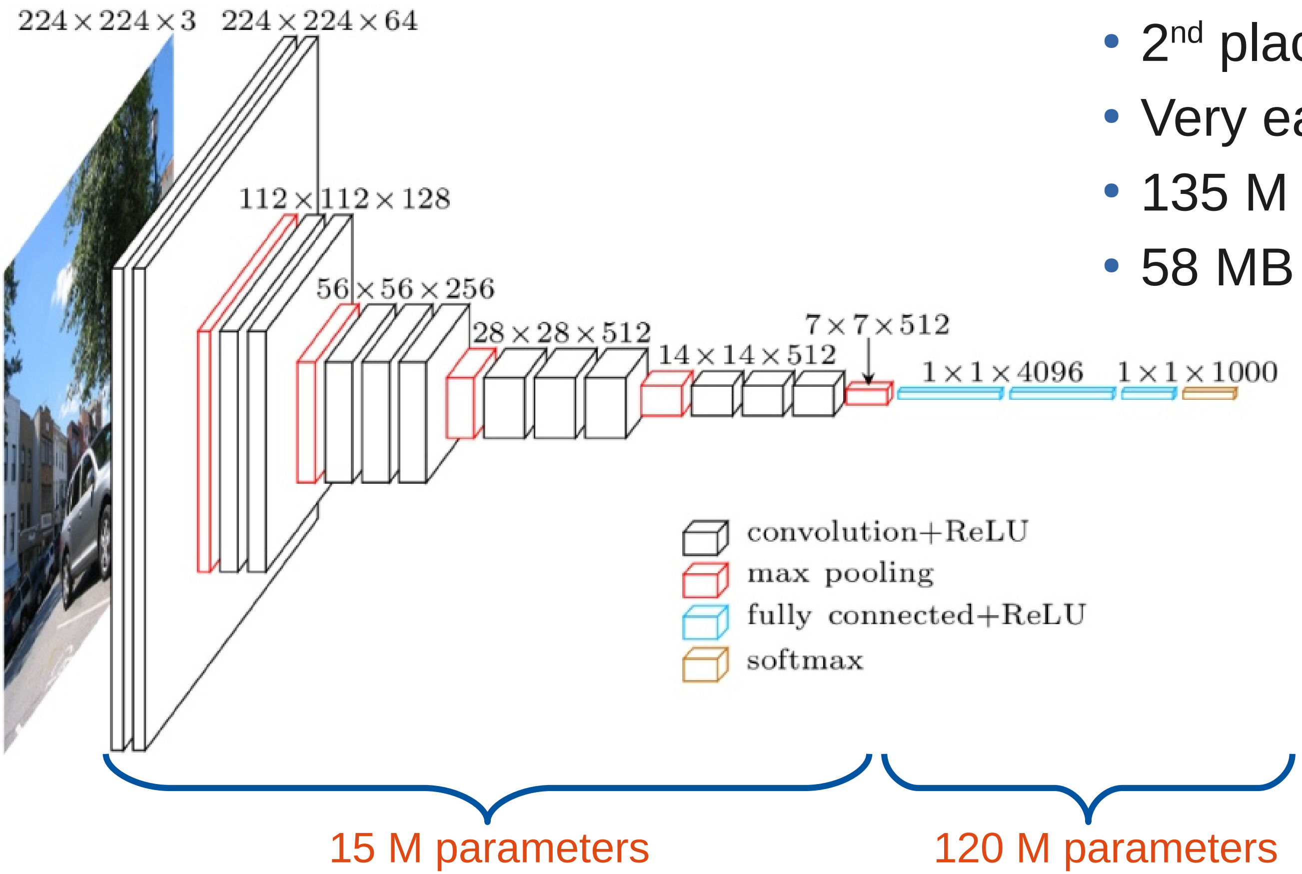
ConvNet architectures usually have a pyramidal shape. For deeper layers:

- Increasing of feature space
- Decreasing of spatial extent



➢ Spatial information is converted to representational features with increasing hierarchy

Example Architecture



VGGNet 16

- 2nd place ILSVRC2014
- Very easy structure
- 135 M parameters → 530 MB
- 58 MB memory for activations

Simonyan, Zissermann
https://arxiv.org/abs/1409.1556

5 minute break

Question: What is a good filter size?

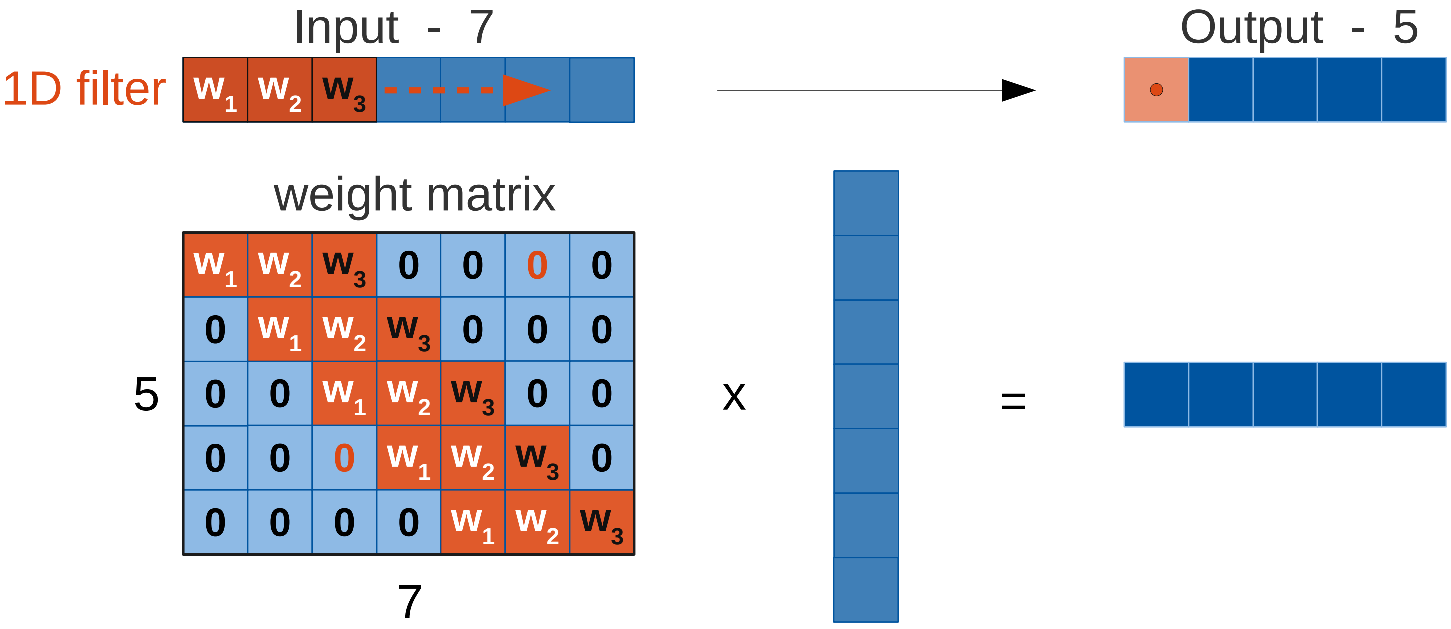
(a) Small (3 x 3)

(b) Large (50 x 50)

(c) Medium (20 x 20)

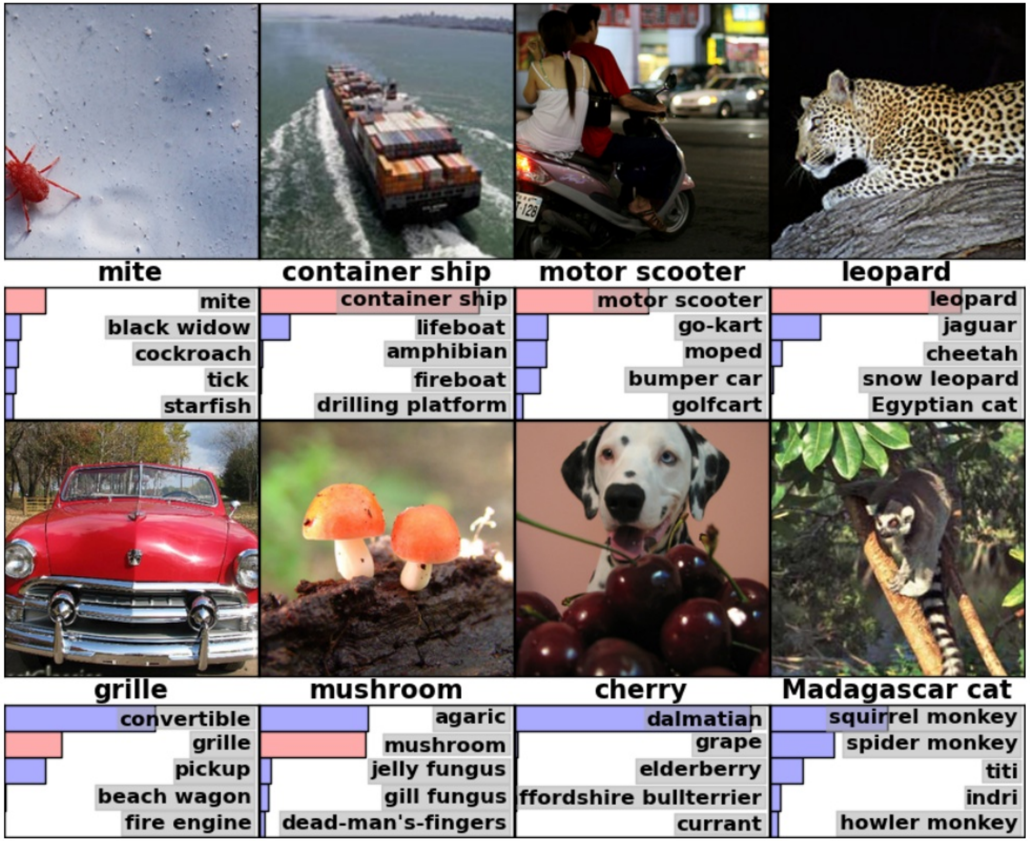
Convolutional Operation

- Fully connected layers are special case of convolutional layers

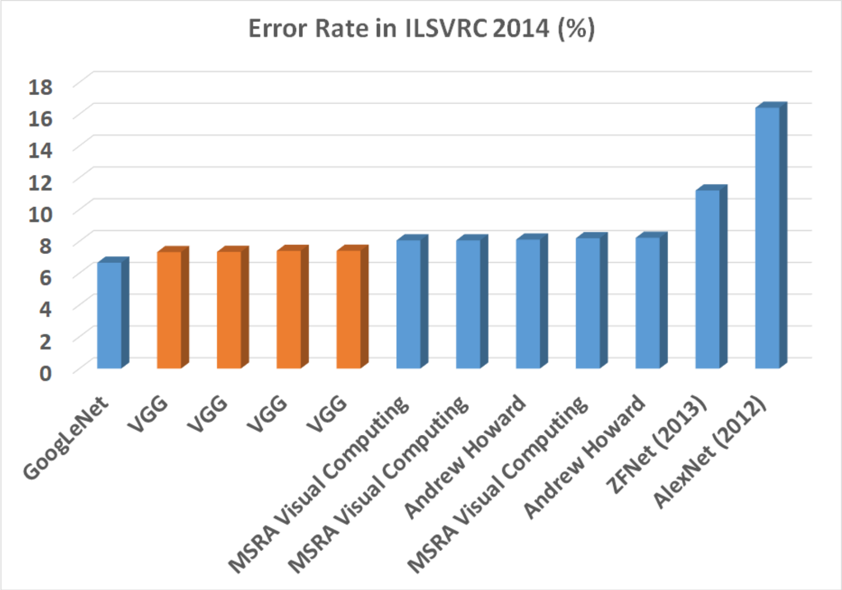


- Parameters greatly reduced due to **sparsity** and **weight sharing**
  - Strong prior on **local correlation** and **translational invariance**

Results on ImageNet

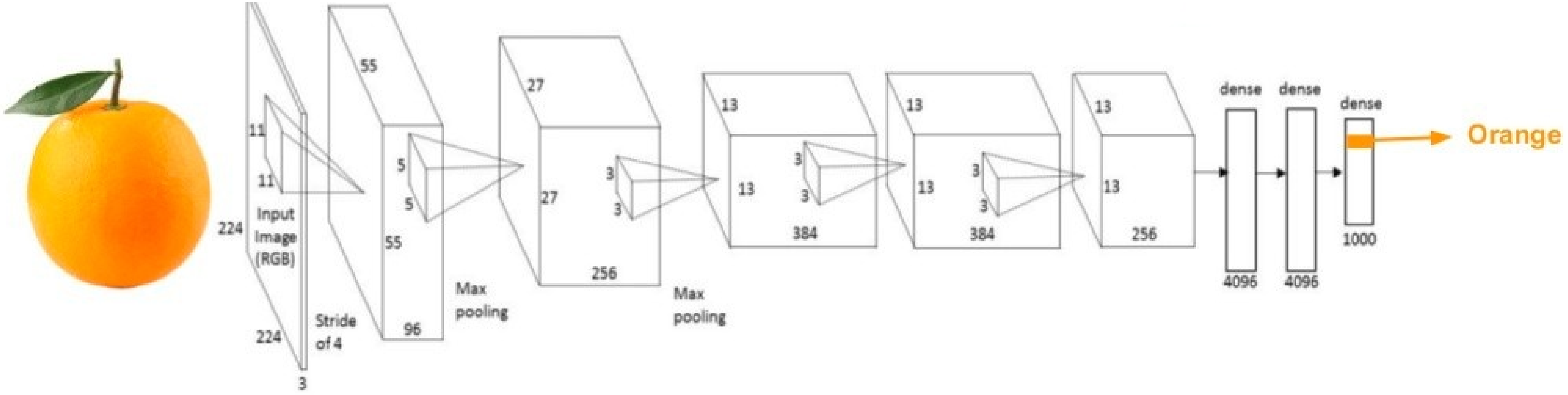


AlexNet (2010) - http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf



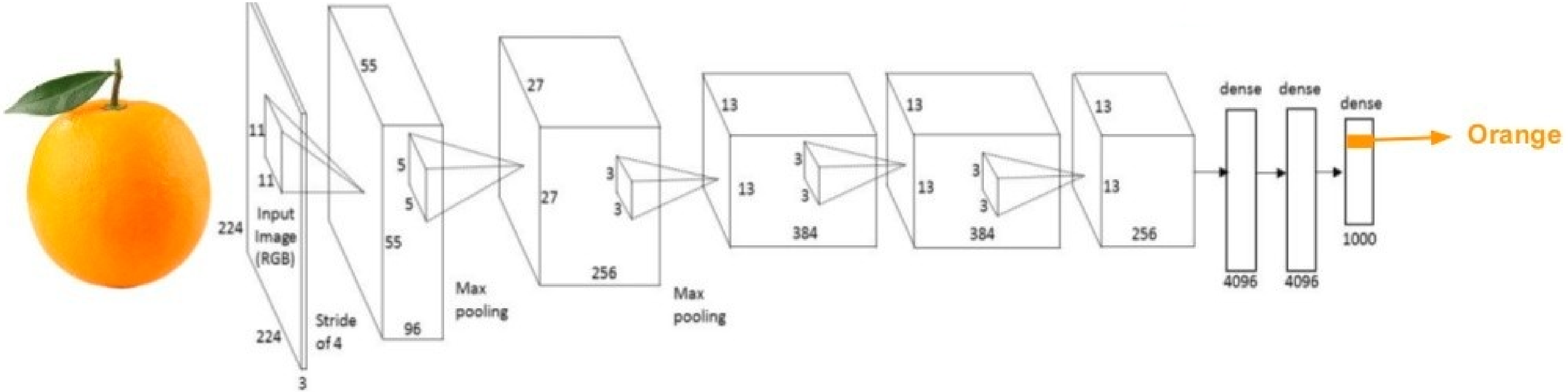
CV as basic task
→ considered as mostly solved

Dropout in CNNs



Where we have to apply dropout?

Dropout in CNNs



Where we have to apply dropout?

- Convolutional networks are less sensitive to over-fitting due to weight sharing
- Spatial Dropout: drop entire feature maps
- Use Dropout before MaxPooling layer
  - Make use of subdominant features

Clarifying frequent misunderstandings

- The **filters are not pre-defined** by the user → just width, depth, and number
  - filters are adapted / learned by the CNN during training

- **Number of filters define number of new feature maps**
  - ten 3x3 filter applied to RGB image → 10 feature maps

- **Filter has the depth of the input image** (e.g. depth 3 for RGB images)
  - two 3x3 filter applied to RGB image → 2 feature maps, i.e. 2 channels
    → number of adaptive parameters = 3 x 3 x 3 * 2 + 2 = 56

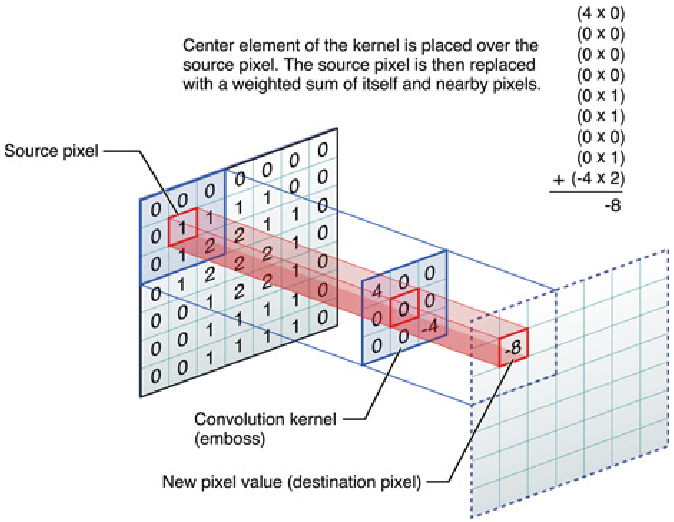
- **After each convolutional operation an activation is applied!** (usually)

- **CNN part is followed by a fully-connected part** (in most cases)
  → output is reshaped (flattened) to a vector → apply vanilla NN layer

Summary

- 2D Convolution acts on 3D input (width x height x depth)
- Slide small filter over input and make linear transformation (dot product + bias)
- Hyperparameter:
  - Size of filter, typically (1 x 1), (3 x 3), (5 x 5) or (7 x 7)
  - Number of filters (feature maps)
  - **Padding** (maintain spatial extent)
  - **Striding** or **pooling** (reduce spatial extent)

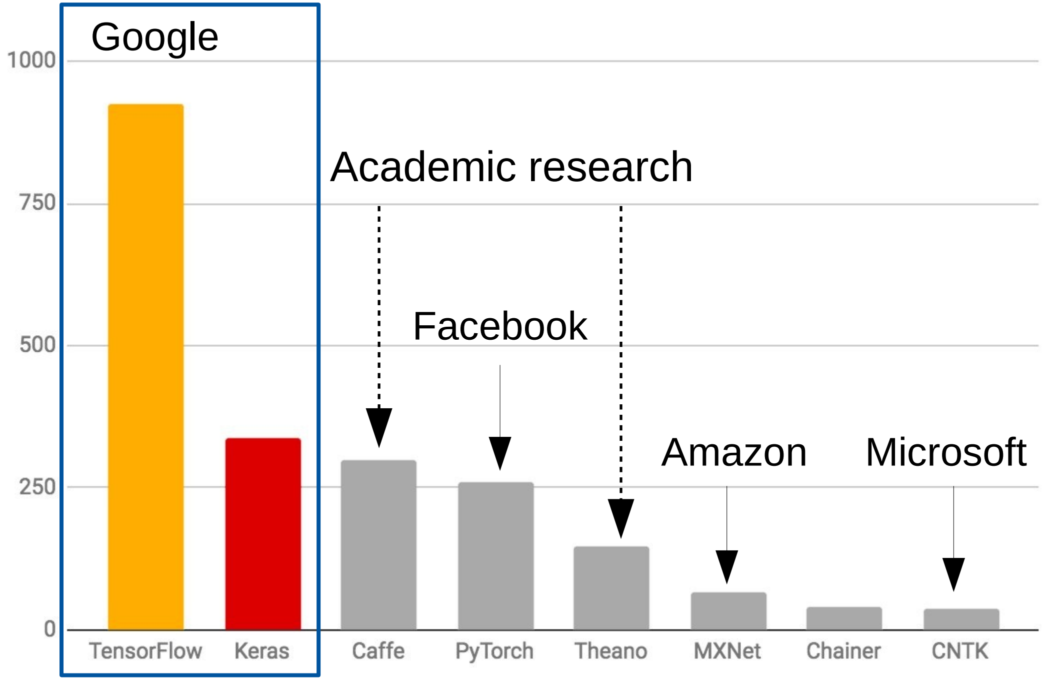
- Reduction of parameters using symmetry in data:
  - Prior on **local correlations** (use small filters)
  - **Translational invariance** (weight sharing)



References & Further Reading

- M. Erdmann, J. Glombitza, G. Kasieczka, U. Klemradt, Deep Learning for Physics Research, World Scientific, 2021, www.deeplearningphysics.org/
- I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, Chapter 7 / 8 / 9, MIT Press, 2016, www.deeplearningbook.org
- Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, arXiv:1502.03044
- Y. LeCun, Y. Bengio, G. Hinton: Deep Learning, Nature 521, pages 436–444
- K. Simonyan, A. Zissermann: Very Deep Convolutional Networks for Large-Scale Image Recognition - ArXiv 1409.1556
- Toy Simulation: M. Erdmann, J. Glombitza, D. Walz, Astroparticle Physics 97, 46-53

Practice II



arXiv mentions as of 2018/03/07 (past 3 months)

Convolutional Layers - Keras

- Same Syntax as for fully connected layers

```
layers.Convolution2D(32, kernel_size=(5, 5), padding='same', activation='relu', strides=(2, 2))
```

  - layer with 32 filters, size of filter 5x5 pixels, stride of 2 in both directions, and ReLU
  - Use padding='same' to keep spatial dimension (else padding='valid')

Pooling and transition to fully-connected networks

- Pooling layer with pooling size of 2x2 pixels and a stride of 2 in both dimensions

```
layers.MaxPooling2D((2,2), strides=(2, 2))    //    layers.AveragePooling2D((2,2), strides=(2, 2))
```

- Layer flattens output to vector → allows use of Dense layers after Convolutions

```
layers.Flatten()
```

- Pooling operation on complete feature map → (remove all pixel dimensions + Flatten)

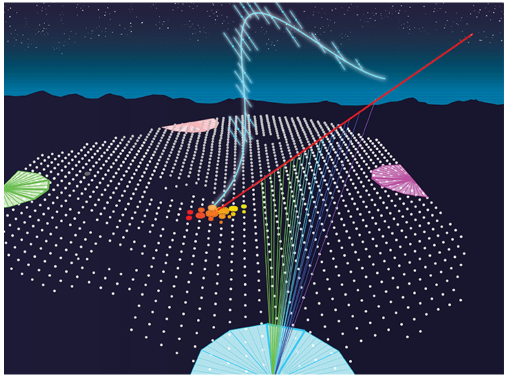
```
layers.GlobalMaxPooling2D()    //    layers.GlobalAveragePooling2D()
```

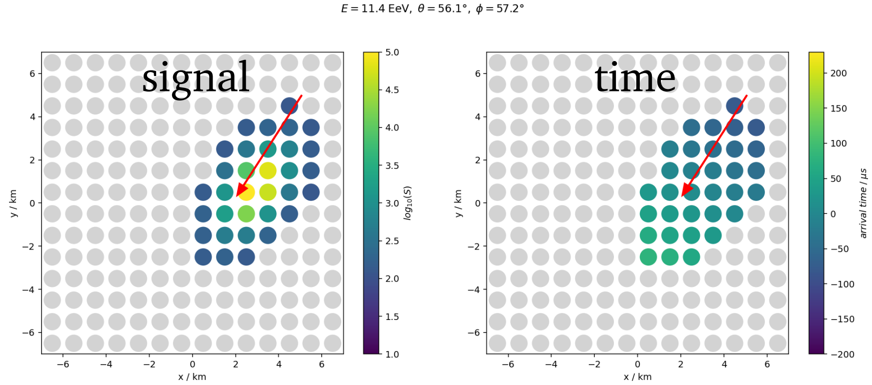
Air Shower Reconstruction - CNN

Now: OPEN tutorial at:

- https://github.com/jglombitza/tutorial_nn_airshowers
- or click

Open in Colab





Task

- Reconstruct energy of the shower
  - Footprint is 2D image
  - **can** directly be used as input
    → input shape: 14 x 14 x 2
  - **Try to reach a resolution better than 2 EeV! (try, e.g., CNN pyramid!)**

Results I

Model – add:

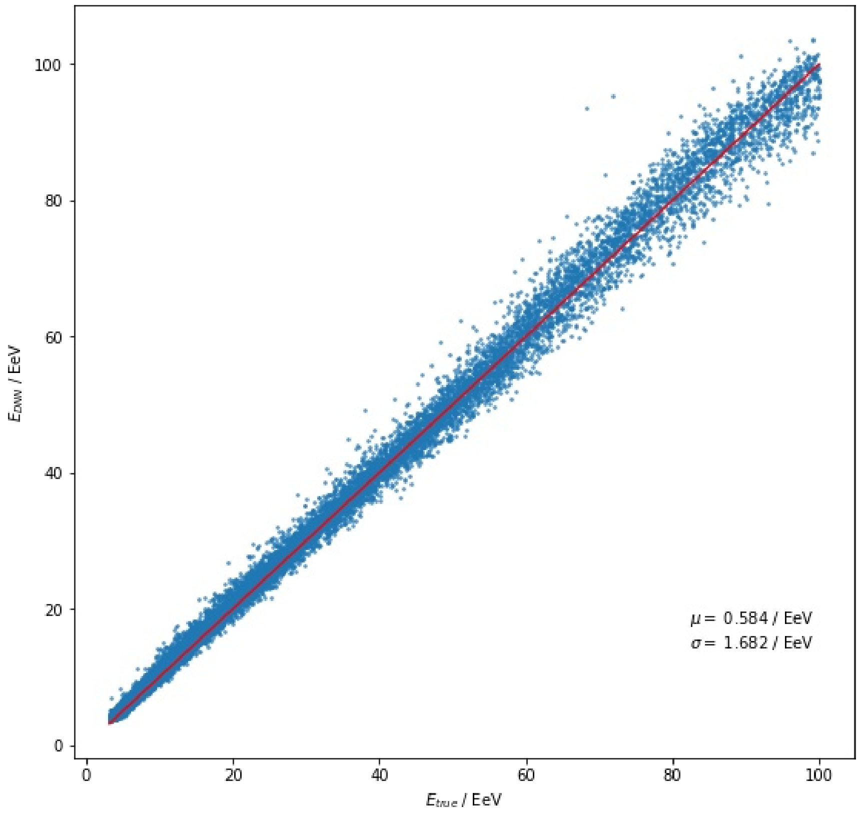
- Conv. layers and filters
- Pooling, Dense (FC) layers
- Regularization (after Flatten)

Model – modify:

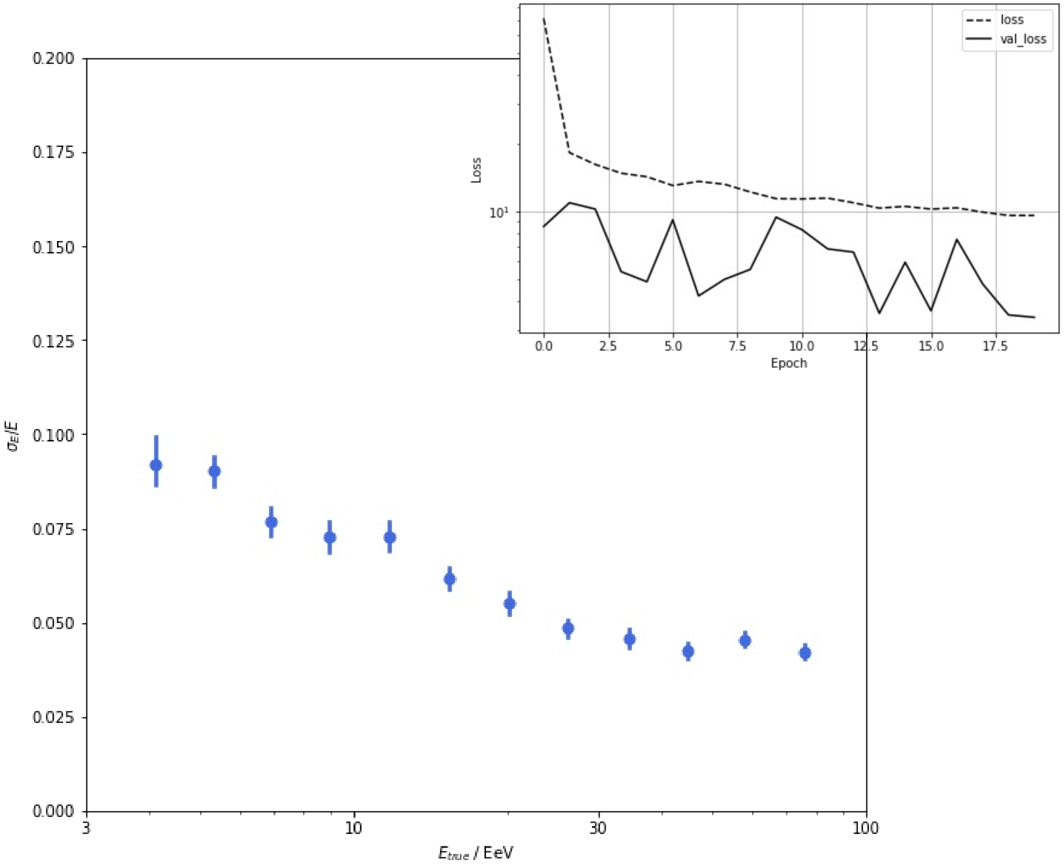
- Batch size, epochs
- Kernel size, strides
- Optimizer, learning rate

```
kwargs = dict(activation='elu', padding='same',)
model = models.Sequential()
model.add(layers.Conv2D(64, (3, 3),
input_shape=X_train.shape[1:], **kwargs))
model.add(layers.Conv2D(64, (3, 3), **kwargs))
model.add(layers.Conv2D(64, (3, 3), **kwargs))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), **kwargs))
model.add(layers.Conv2D(128, (3, 3), **kwargs))
model.add(layers.Conv2D(128, (3, 3), **kwargs))
model.add(layers.GlobalMaxPooling2D())
model.add(layers.Dropout(0.3))
model.add(layers.Dense(128, activation='elu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(1))
```

Results II



$\mu =$ 0.584 / EeV
$\sigma =$ 1.682 / EeV
E_{DNN} / EeV
E_{true} / EeV



loss
val_loss
Loss
Epoch
σ_E/E
E_{true} / EeV

CIFAR 10 – Convolutional Networks

Model – add:

- Conv. layers and filters
- Pooling, Dense (FC) layers
- Regularization (after Flatten)

Model – modify:

- Batch size, epochs
- Kernel size, strides
- Optimizer, learning rate

```
model = models.Sequential([
    layers.Convolution2D(32, kernel_size=(5, 5), padding='same',
                         activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2,2)),
    layers.Convolution2D(64, kernel_size=(3, 3), padding='same',
                         strides=(2, 2), activation='relu'),
    layers.Flatten(),
    layers.Dropout(0.3),
    layers.Dense(10, activation='softmax')
    ])
```

Open in Colab

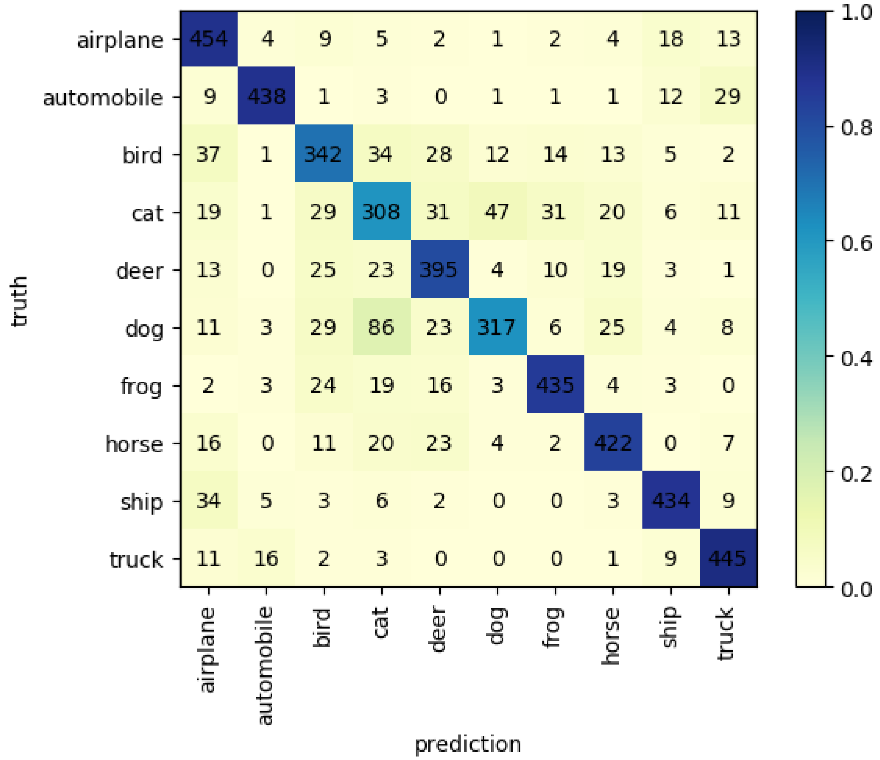
- **Can you achieve >75% validation accuracy?**

https://github.com/jglombitza/cifar_tutorial

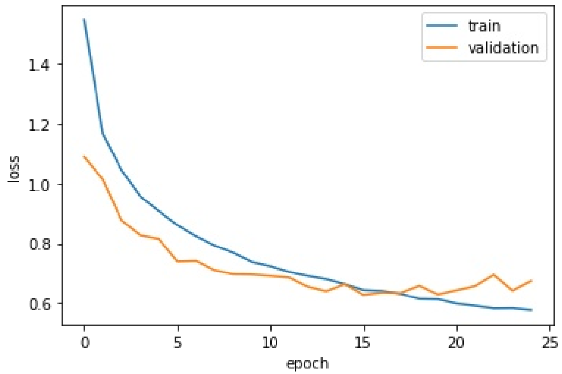
CIFAR 10 – Deep Convolutional Network

```
model = models.Sequential([
    layers.Convolution2D(16, kernel_size=(3, 3), padding='same', activation='elu',
                                                input_shape=(32, 32, 3)),
    layers.Convolution2D(32, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(32, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.MaxPooling2D((2,2)),
    layers.Convolution2D(32, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(64, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(64, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.MaxPooling2D((2,2)),
    layers.Convolution2D(64, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(128, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.GlobalMaxPooling2D(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')])
```

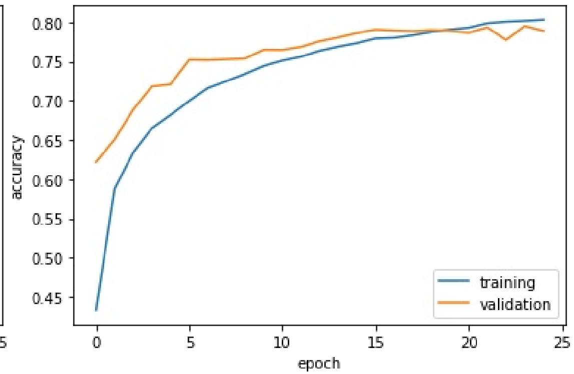
Results



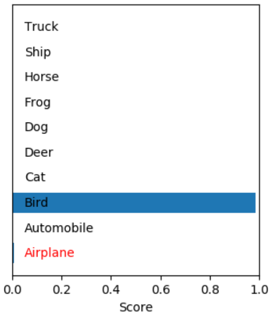
airplane: 454 4 9 5 2 1 2 4 18 13
automobile: 9 438 1 3 0 1 1 1 12 29
bird: 37 1 342 34 28 12 14 13 5 2
cat: 19 1 29 308 31 47 31 20 6 11
deer: 13 0 25 23 395 4 10 19 3 1
dog: 11 3 29 86 23 317 6 25 4 8
frog: 2 3 24 19 16 3 435 4 3 0
horse: 16 0 11 20 23 4 2 422 0 7
ship: 34 5 3 6 2 0 0 3 434 9
truck: 11 16 2 3 0 0 0 1 9 445
truth
prediction
airplane automobile bird cat deer dog frog horse ship truck
0.0 0.2 0.4 0.6 0.8 1.0



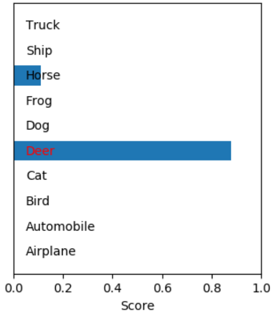
train
validation
loss
epoch



training
validation
accuracy
epoch



Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile
Airplane
Score



Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile
Airplane
Score

Jonas Glombitza
jonas.glombitza@fau.de

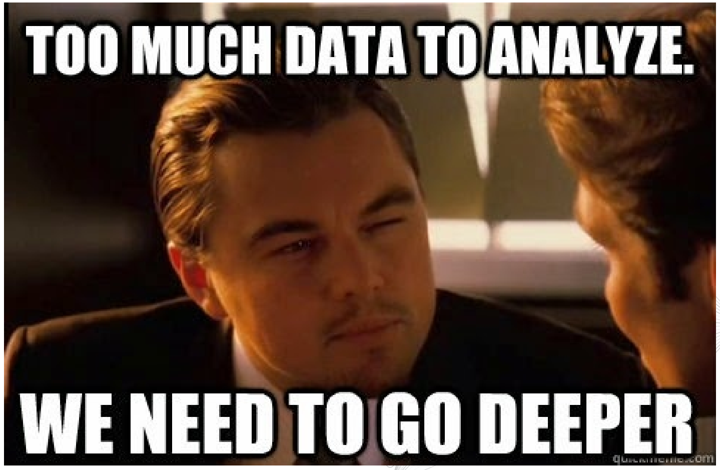
ISAPP School 2024
Bad Liebenzell, KIT

https://github.com/DeepLearningForPhysicsResearchBook/deep-learning-physics

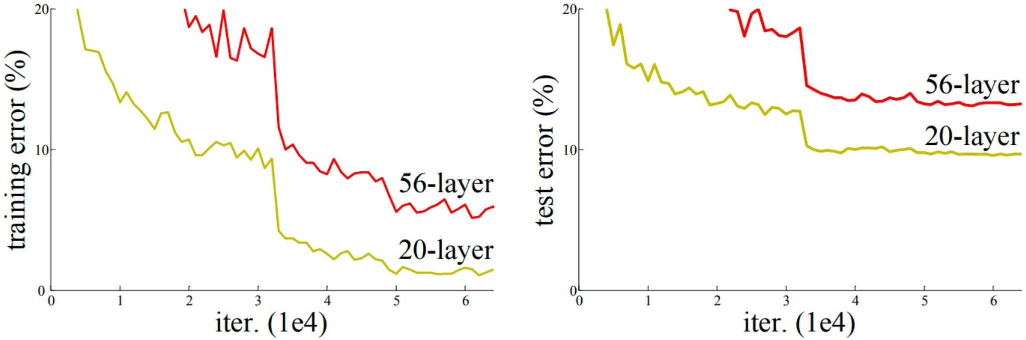
ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

Advanced CNN / DNN techniques

I. Vanishing gradients
II. Normalization layers
III. Short cuts

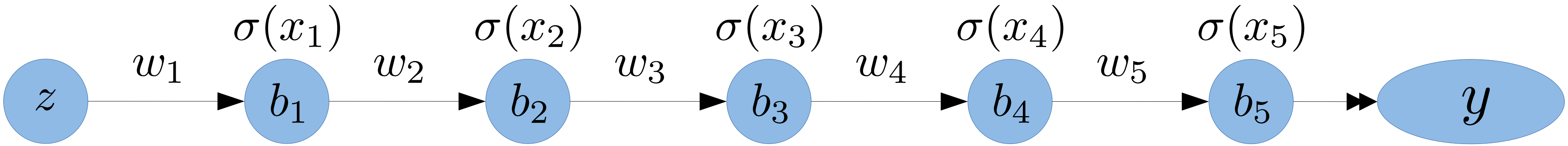


Obstacles when Going Deeper



- Neural networks should get monotonously better when adding more layers
- **Problems**
  - Bad initialization
  - Vanishing gradients – gradients become too small
  - Shattered gradients – gradients become white noise
  - Internal covariate shift – need to constantly adapt changes in earlier layer
  - Convolutional filter show redundant behavior → advanced CNN operations

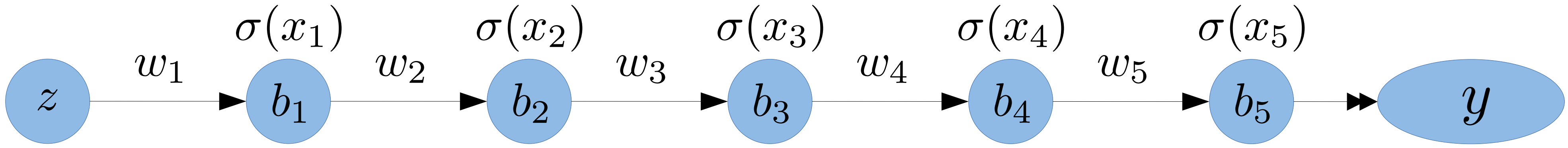
Vanishing Gradient Problem



$$y = \sigma(x_5) = \sigma(w_5 \cdot \sigma(x_4) + b_5) = \sigma(w_5 \cdot \sigma(w_4 \cdot \sigma(x_3) + b_4) + b_5)....$$

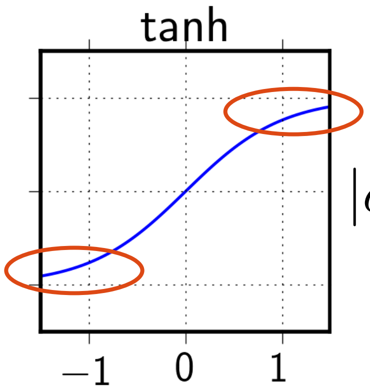
$$\frac{\partial y}{\partial w_1} = \frac{\partial \sigma(x_5)}{\partial x_5} \frac{\partial x_5}{\partial \sigma(x_4)} \frac{\partial \sigma(x_4)}{\partial x_4} \frac{\partial x_4}{\partial w_1} .... = \underline{\sigma'(x_5)} w_5 \cdot \underline{\sigma'(x_4)} w_4 .... \cdot \underline{\sigma'(x_1)} z$$

Vanishing Gradient Problem



$$y = \sigma(x_5) = \sigma(w_5 \cdot \sigma(x_4) + b_5) = \sigma(w_5 \cdot \sigma(w_4 \cdot \sigma(x_3) + b_4) + b_5)....$$

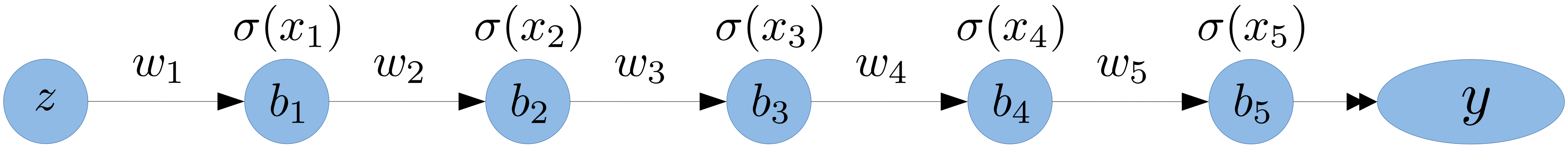
$$\frac{\partial y}{\partial w_1} = \frac{\partial \sigma(x_5)}{\partial x_5} \frac{\partial x_5}{\partial \sigma(x_4)} \frac{\partial \sigma(x_4)}{\partial x_4} \frac{\partial x_4}{\partial w_1} .... = \underline{\sigma'(x_5)} w_5 \cdot \underline{\sigma'(x_4)} w_4 .... \cdot \underline{\sigma'(x_1)} z$$



$$|\sigma'(x)| << 1$$

- Stacking many layers can lead to vanishing gradients
- Activation saturates
  - Updates in early layers become very tiny
    - **No learning**
  - Don’t use sigmoids / tanh only rarely → use shortcuts
    - still useful as last layer if data ranges [0;1] or [1:-1]

Vanishing Gradient Problem

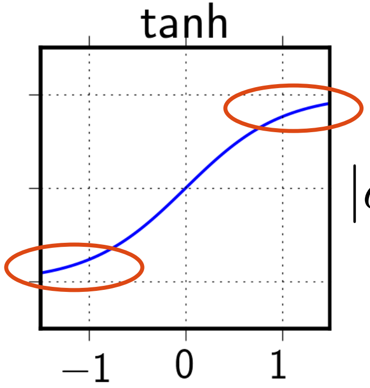


$$y = \sigma(x_5) = \sigma(w_5 \cdot \sigma(x_4) + b_5) = \sigma(w_5 \cdot \sigma(w_4 \cdot \sigma(x_3) + b_4) + b_5)....$$

$$\frac{\partial y}{\partial w_1} = \frac{\partial \sigma(x_5)}{\partial x_5} \frac{\partial x_5}{\partial \sigma(x_4)} \frac{\partial \sigma(x_4)}{\partial x_4} \frac{\partial x_4}{\partial w_1} .... = \underline{\sigma'(x_5)} w_5 \cdot \underline{\sigma'(x_4)} w_4 .... \cdot \underline{\sigma'(x_1)} z$$

VISUALIZATION

LINK



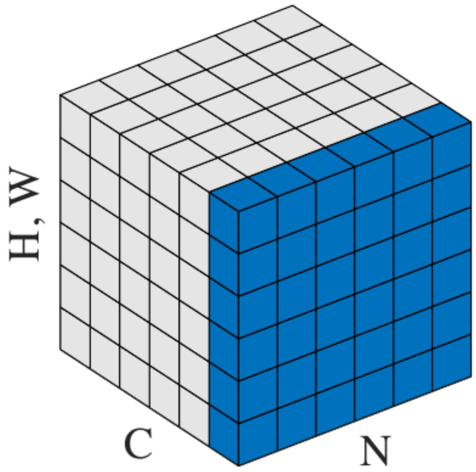
$$|\sigma'(x)| << 1$$

- Stacking many layers can lead to vanishing gradients
- Activation saturates
  - Updates in early layers become very tiny
    - **No learning**
  - Don't use sigmoids / tanh only rarely → use shortcuts
    - still useful as last layer if data ranges [0;1] or [1:-1]

Batch Normalization

- Calculate batch-wise for each channel:
  - Mean: μ_B and Variance: σ_B^2
  - Add free parameters $\gamma,\ \beta$
    - Easily control first moments of distribution

➢ $$y = \frac{x - \mu_B}{\sigma_B}\gamma + \beta$$

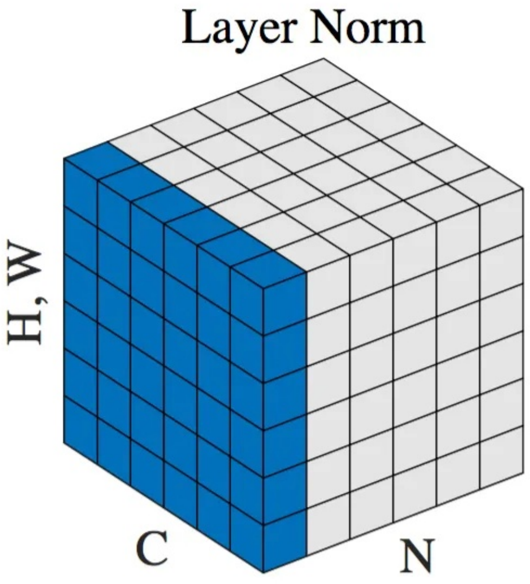


- Makes DNN robust against poor initializations
- Helps with vanishing gradient / less sensitive to high learning rates
- Has regularizing effect (no large weights, noise because of batch dependency)
- Reduce internal covariate shift

➢ **Very successful for convolutional architectures**

Layer Normalization

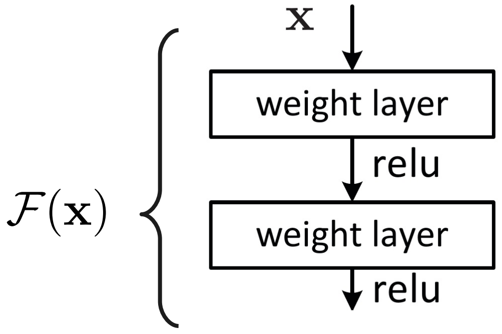
- Calculate sample-wise (across channels)
  - Mean: μ_l and Variance: σ_l^2
  - Add free parameters $g,\ b$ → control first moments
  - $$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l, \quad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$
- $$y = \frac{x - \mu_B}{\sigma_B} g + b$$



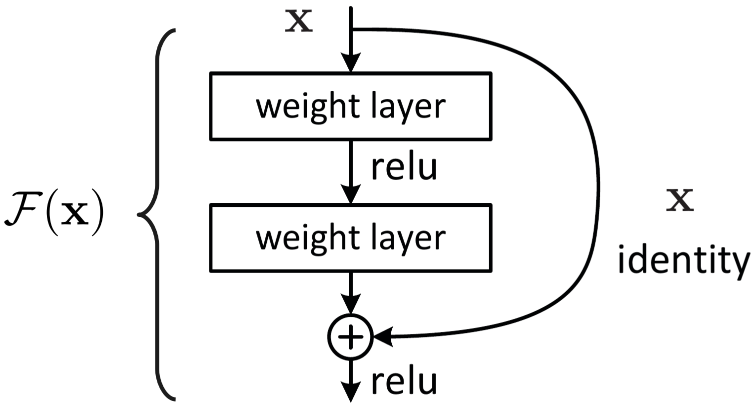
- Makes DNN robust against poor initializations
- Improves training behaviour → normalized gradients
- **Very successful for recurrent / transformer architectures** (small batches)

Residual Unit

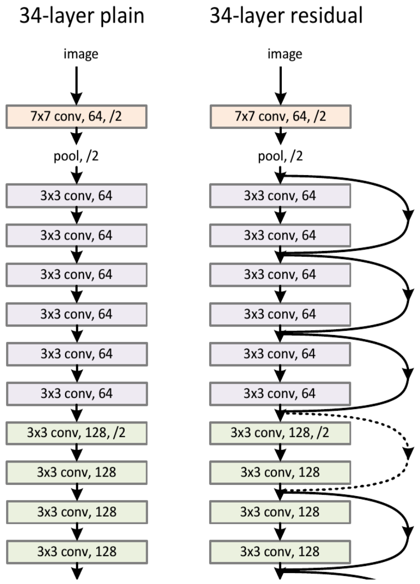
Idea: Residual unit consisting of small network and a shortcut (identity mapping)



$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x})$



$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$



- Weight block learns small residual $\mathcal{F}(\mathbf{x})$ on top of input $\mathbf{x}$
  - Output of residual unit $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$
- Shortcut let gradient propagate easily to earlier layers
- Later layers can easily turn weights to zero by $\mathcal{F}(\mathbf{x}) \to 0$

Up to several of hundreds layer deep!

5 minute break

Jonas Glombitza
jonas.glombitza@fau.de

9th Institute of Space Sciences Summer School:
Multimessenger Astrophysics

Transformers

I. Processing image-like data
II. Point cloud transformers

https://github.com/DeepLearningForPhysicsResearchBook/deep-learning-physics

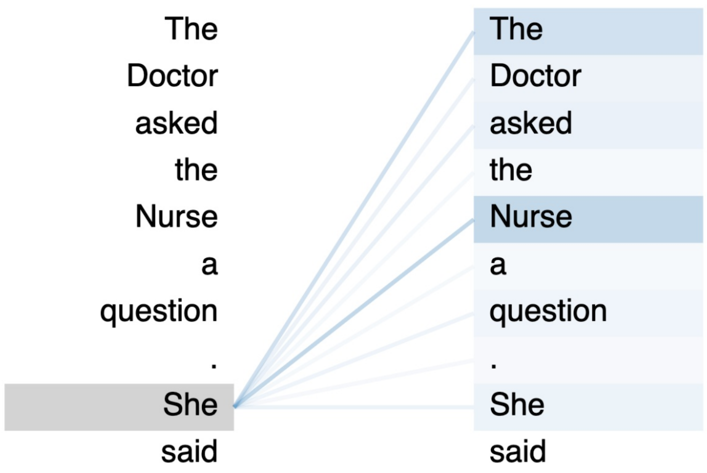
Transformers

- Transformers are **backbone of latest breakthroughs**: LLMs / Stable Diffusion
- Building blocks: DNNs with attention mechanism
  - Which parts of sequence semantically correlated → analyze together

ChatGPT

In a nutshell: extension of fully-connected DNNs

- listen to all inputs
- *but* focus on most important inputs
- → increase noise robustness



Analyze *sequences* (different lengths):

- $(x_1, x_2, x_3, x_4, x_5, \ldots, x_n)$
  - single element called *token* (e.g, word)

Simple case: Attention of 2 vectors

$$X = \begin{pmatrix} \vec{x_1}^T \\ \vec{x_2}^T \end{pmatrix} = \begin{pmatrix} {x_1}^{(1)} & {x_1}^{(2)} \\ {x_2}^{(1)} & {x_2}^{(2)} \end{pmatrix}$$

- Input: 2 unit vectors (arbitrary length), could, e.g., represent words / measurements

Dot product measures similarity: parallel = 1, orthogonal = 0

$$\mathrm{A} = \begin{pmatrix} \vec{x_1} \cdot \vec{x_1} & \vec{x_1} \cdot \vec{x_2} \\ \vec{x_2} \cdot \vec{x_1} & \vec{x_2} \cdot \vec{x_2} \end{pmatrix} = XX^T$$

Attention matrix: A

- Estimate attention (similarity of vectors)
- Calculate dot product

- Add learnable parameters to compare similarities in feature space (after filtering), use W^Q

✗ Attention matrix symmetric (not useful)

$$\mathrm{A} = \begin{pmatrix} \underbrace{W^Q\vec{x_1} \cdot W^Q\vec{x_1}}_{=1 \rightarrow \text{boring}} & W^Q\vec{x_1} \cdot W^Q\vec{x_2} \\ W^Q\vec{x_2} \cdot W^Q\vec{x_1} & \underbrace{W^Q\vec{x_2} \cdot W^Q\vec{x_2}}_{=1 \rightarrow \text{boring}} \end{pmatrix} = XW^Q(XW^K)^T$$

same!

Simple case: Attention of 2 vectors

$$X = \begin{pmatrix} \vec{x_1}^T \\ \vec{x_2}^T \end{pmatrix} = \begin{pmatrix} x_1{}^{(1)} & x_1{}^{(2)} \\ x_2{}^{(1)} & x_2{}^{(2)} \end{pmatrix}$$

- Input: 2 unit vectors (arbitrary length), could, e.g., represent words / measurements

$$\mathrm{A} = \begin{pmatrix} \vec{x_1} \cdot \vec{x_1} & \vec{x_1} \cdot \vec{x_2} \\ \vec{x_2} \cdot \vec{x_1} & \vec{x_2} \cdot \vec{x_2} \end{pmatrix} = XX^T$$

Attention matrix: A

- Estimate attention (similarity of vectors)
- Calculate dot product

- Add 2 learnable sets of parameters to compare in feature space, project using W^Q, W^K
  - ✔ Attention matrix <u>not</u> symmetric & diagonal entries <u>not</u> unity

$$\mathrm{A} = \begin{pmatrix} W^Q\vec{x_1} \cdot W^K\vec{x_1} & W^Q\vec{x_1} \cdot W^K\vec{x_2} \\ W^Q\vec{x_2} \cdot W^K\vec{x_1} & W^Q\vec{x_2} \cdot W^K\vec{x_2} \end{pmatrix} = XW^Q(XW^K)^T$$

Scaled dot product attention

- Create three projections of input, $W^i \in \mathbb{R}^{f \times d}$
  - perform scaled dot product attention

$$X = \begin{pmatrix} \vec{x}_i^{\,T} \\ \vdots \end{pmatrix}$$

Input: arbitrary number of vectors, of dimension f

queries

$$\mathrm{Q} = \mathrm{X} \cdot W^Q$$ → projection 1 for attention matrix

keys

$$\mathrm{K} = \mathrm{X} \cdot W^K$$ → projection 2 for attention matrix

Output shape: $n_{\text{inputs}} \times d$

values

$$\mathrm{V} = \mathrm{X} \cdot W^V$$ → project input to multiply with attention matrix

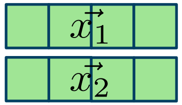
$$\text{Attention}(Q, K, V) \hat{=} AV = \underbrace{\text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)}_{\text{Attention matrix: } A_{ij} = \frac{e^{q_i \cdot k_j}}{\sum_j e^{q_i \cdot k_j}}} V = Z$$

$\frac{1}{\sqrt{d}}$ → ensure $\mathrm{Var}(QK^T) = 1$ after initialization

Scaled dot product attention

- Create three projections of input, $W^i \in \mathbb{R}^{4\times 3}$
  - perform scaled dot product attention

$$X = \begin{pmatrix} \vec{x_1}^T \\ \vec{x_2}^T \end{pmatrix} =$$

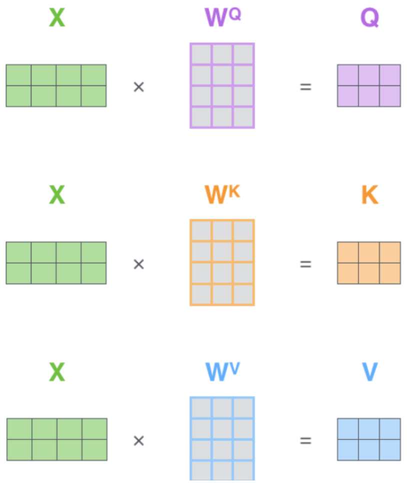


Input: 2 vectors, of dimension $f = 4$

queries
$$\mathrm{Q} = \mathrm{X}\cdot W^Q$$

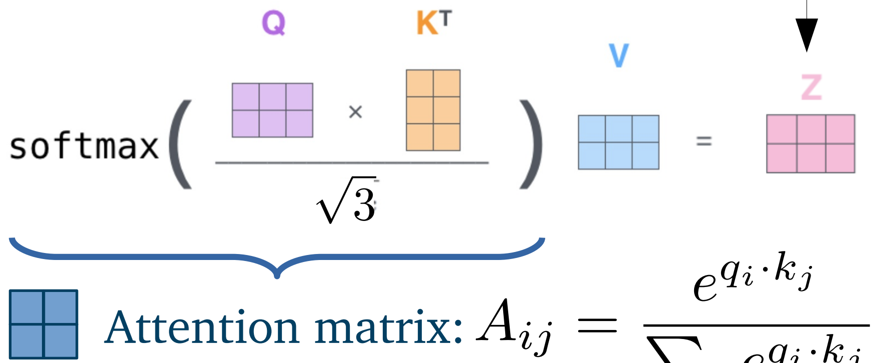
keys
$$\mathrm{K} = \mathrm{X}\cdot W^K$$

values
$$\mathrm{V} = \mathrm{X}\cdot W^V$$



Output shape: $n_{\text{inputs}} \times d = 2 \times 3$

$$\mathrm{Attention}(Q, K, V) \hat{=} AV = \mathtt{softmax}\left(\frac{QK^T}{\sqrt{3}}\right)V = Z$$



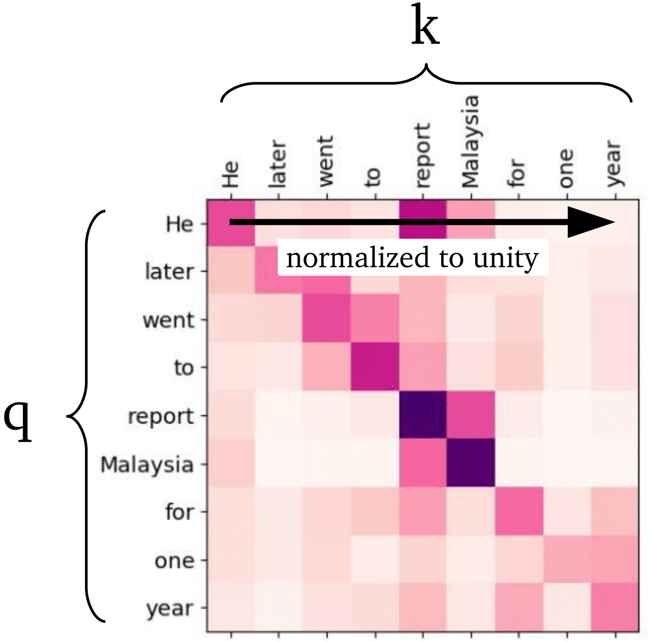
Attention matrix: $A_{ij} = \frac{e^{q_i \cdot k_j}}{\sum_j e^{q_i \cdot k_j}}$

Attention

- Enables to learn long range correlations
  - CNNs: prior on local correlations (small filters)
  - Correlates all inputs with each other (like in fully-connected networks)
  - Attention: clever way to tell DNN where to focus on
- Very memory expensive
  - Matrix scales with sequence length squared: n_{inp}^2

Attention mechanism very flexible

- Due to dot product: independent of sequence length

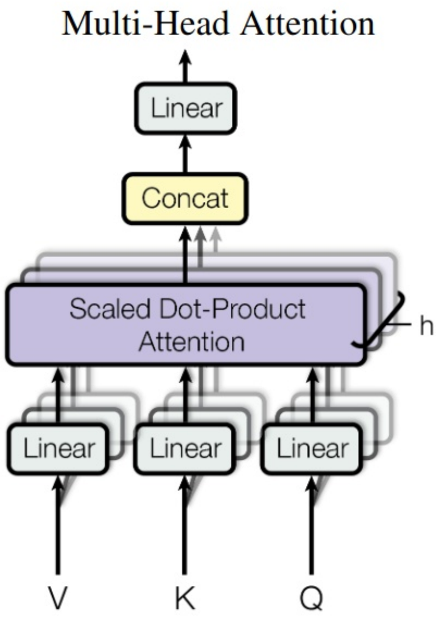


Visualized attention matrix

Multi-head attention

- Attention performed in parallel → called *heads*
  - Analyze h subspaces with different representation
- Keep complexity
  - dimensionality per head reduced: $d' = d/h$

    (Note: d needs to be dividable by h)
- Add final projection to combined heads
  - $\mathrm{V} = \mathrm{X} \cdot W^O$



$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}\left(\mathrm{head}_1, \ldots, \mathrm{head}_\mathrm{h}\right) \cdot W^O$$

$$\mathrm{head} = \mathrm{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

Inputs to the transformer

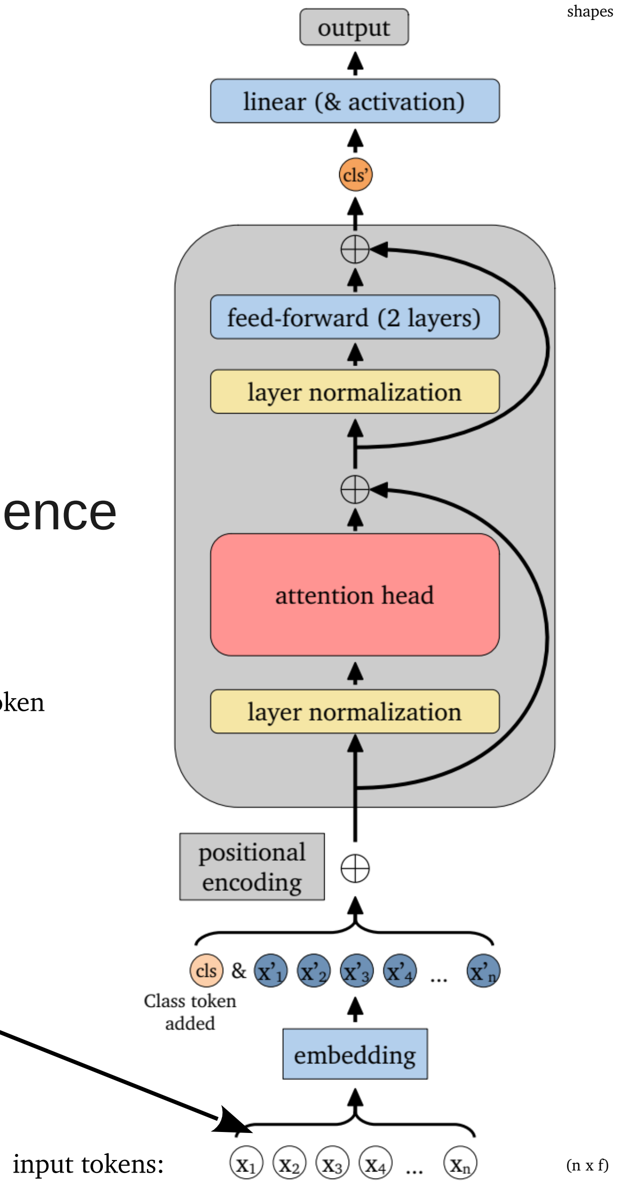
*For simplicity we consider batch size of 1
usually training is performed in batches

Inputs*

- Input is **a single** sequence (one sentence)
- Arbitrary number of tokens (words) per sequence
  - Each word / vector has dimension f

first token

$$\text{Length of sequence: } n \left\{ \begin{pmatrix} \vec{x}_1^{\,T} \\ \vec{x}_2^{\,T} \\ \vec{x}_3^{\,T} \\ \vdots \\ \vec{x}_n^{\,T} \end{pmatrix} \right.$$



Embedding

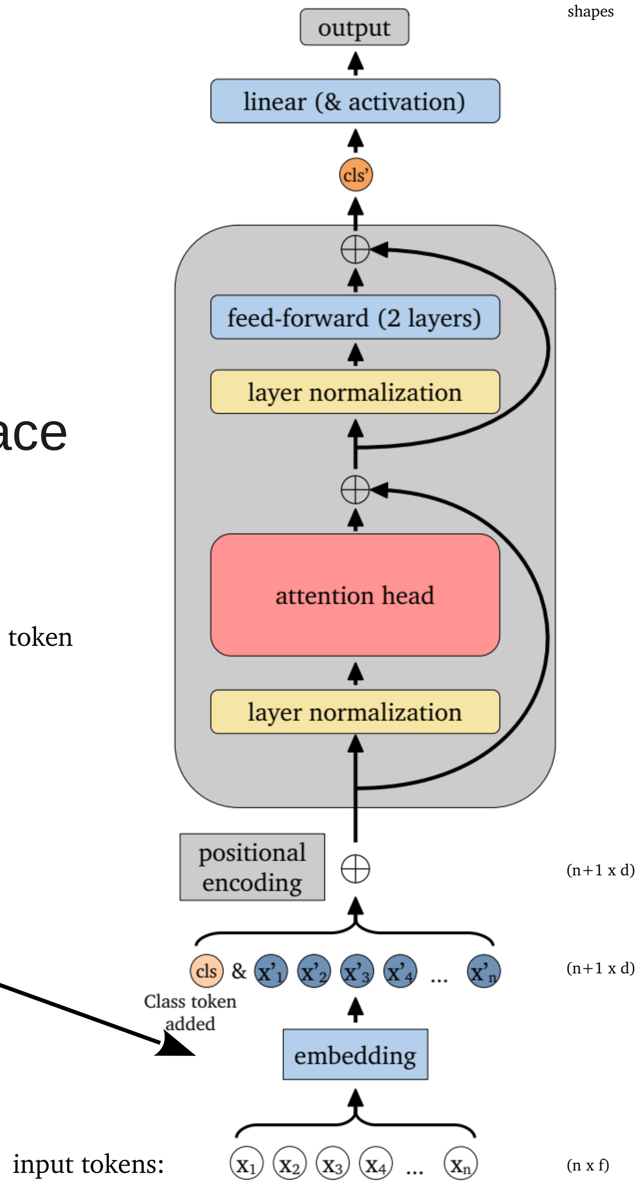
Embedding layer

- Project inputs vectors into high-dimensional space
  - Find suitable representation

Embedding matrix

$$X' = W_{\mathrm{embed}}\, X = \begin{pmatrix} \vec{x'}_1^{\,T} \\ \vec{x'}_2^{\,T} \\ \vec{x'}_3^{\,T} \\ \vdots \\ \vec{x'}_n^{\,T} \end{pmatrix}$$

transformed token



Does the attention matrix considers token ordering?

(a) Yes! How should large language models (LLMs) work if not?!

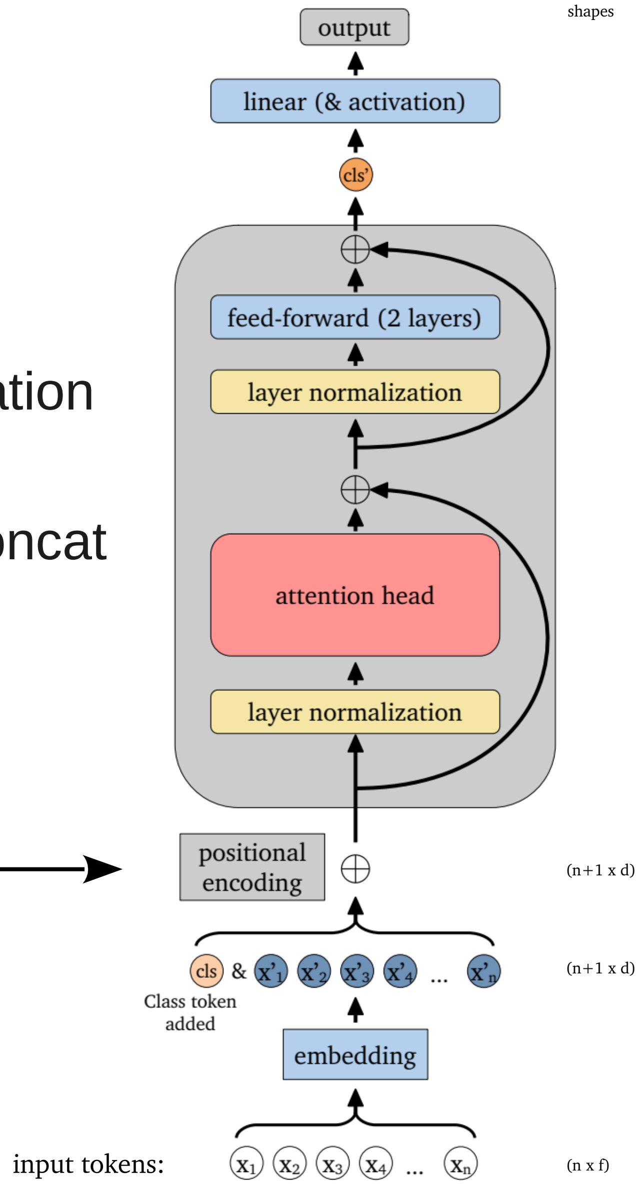
(b) No, the decomposition is permutational invariant

Positional encoding

Positional encoding

- Transformers are permutational invariant
- If positions / ordering contain important information
  - add positions as additional information
- Many options possible: encode & add / pure concat

A permutational invariant transformer would confuse
"I like physics more than ice cream"
with
"I like ice cream more than physics"



Positional encoding

Positional encoding

- Transformers are permutational invariant
- If positions / ordering contain important information
  - add positions as additional information
- Many options possible: <u>encode & add</u> / pure concat

Example positions

Coordinate of point:

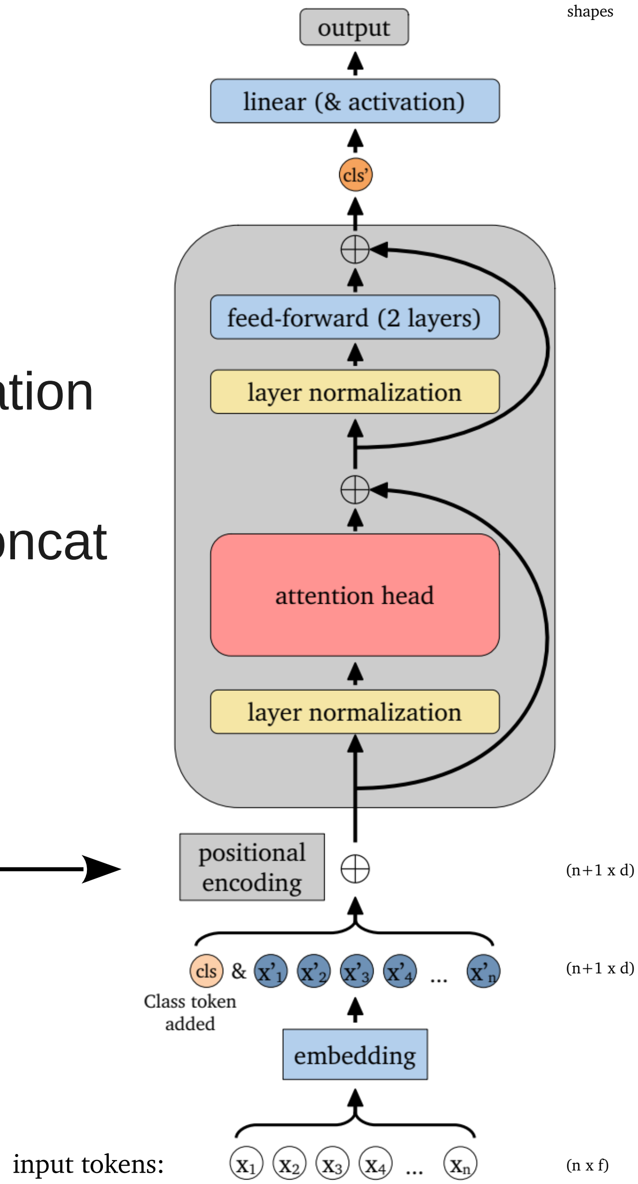
$$\vec{x}_{pos} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Index of sequence:

$$\vec{x}_{pos} = \begin{pmatrix} i \end{pmatrix}$$

Encoding matrix

$$\mathrm{X'} = \mathrm{P}_{\mathrm{enc}}\, X_{\mathrm{pos}} = \begin{pmatrix} \vec{x'}^{T}_{1,\mathrm{pos}} \\ \vec{x'}^{T}_{2,\mathrm{pos}} \\ \vec{x'}^{T}_{3,\mathrm{pos}} \\ \vdots \\ \vec{x'}^{T}_{n,\mathrm{pos}} \end{pmatrix}$$



Positional encoding

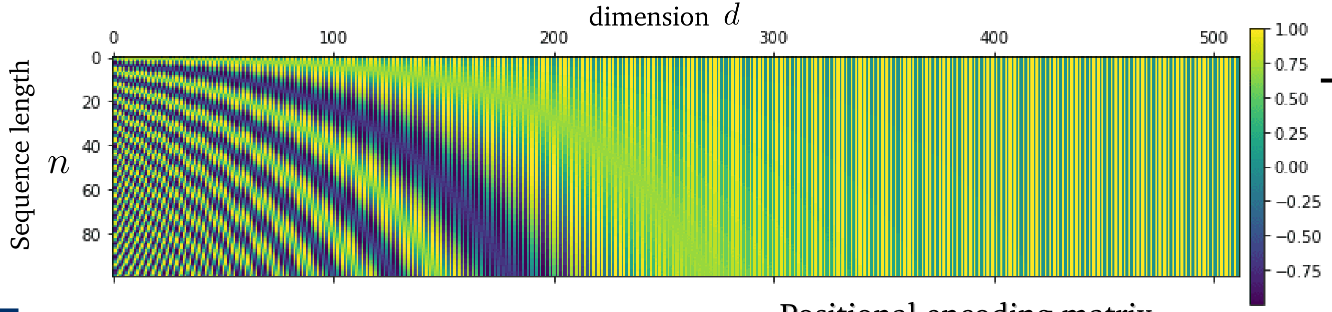
Positional encoding

- Transformers are permutational invariant
- If positions / ordering contain important information
  - add positions as additional information

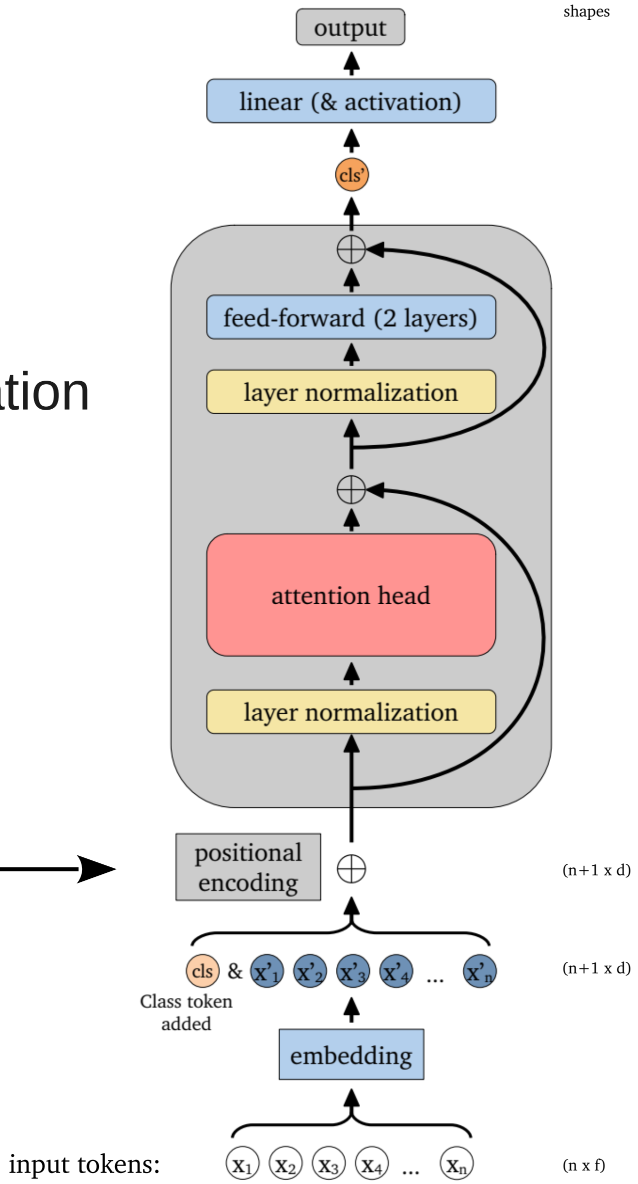
Positional encoding for vision transformer

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$



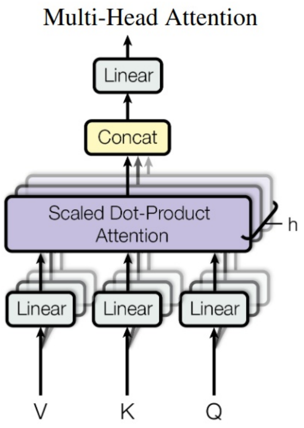
Positional encoding matrix

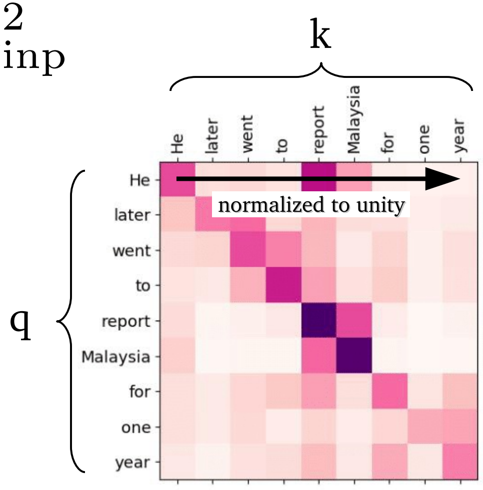


Attention block

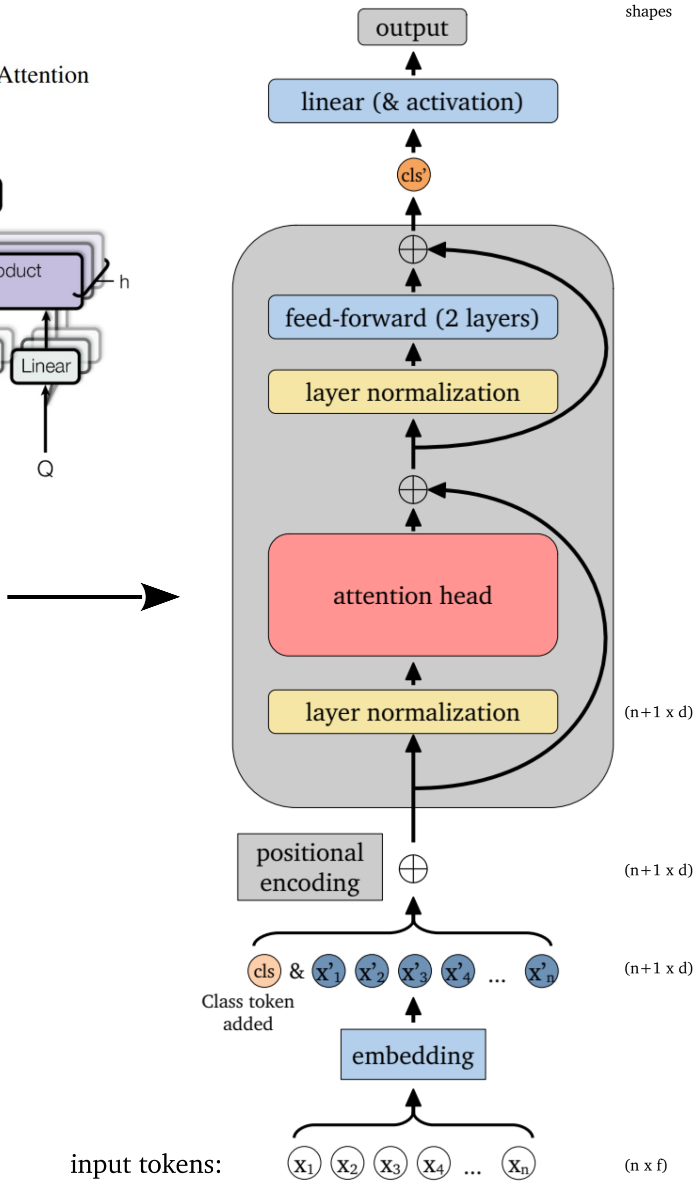
Attention block

- Multi-head attention performed
  - h heads run in parallel $d' = d/h$
- Learn long-range relation
- Expensive, scales n_{inp}^2





Visualized attention matrix

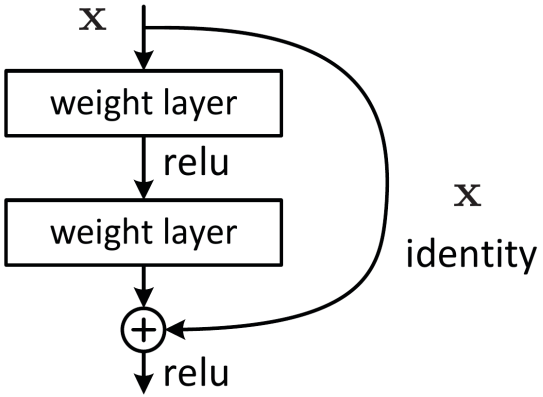


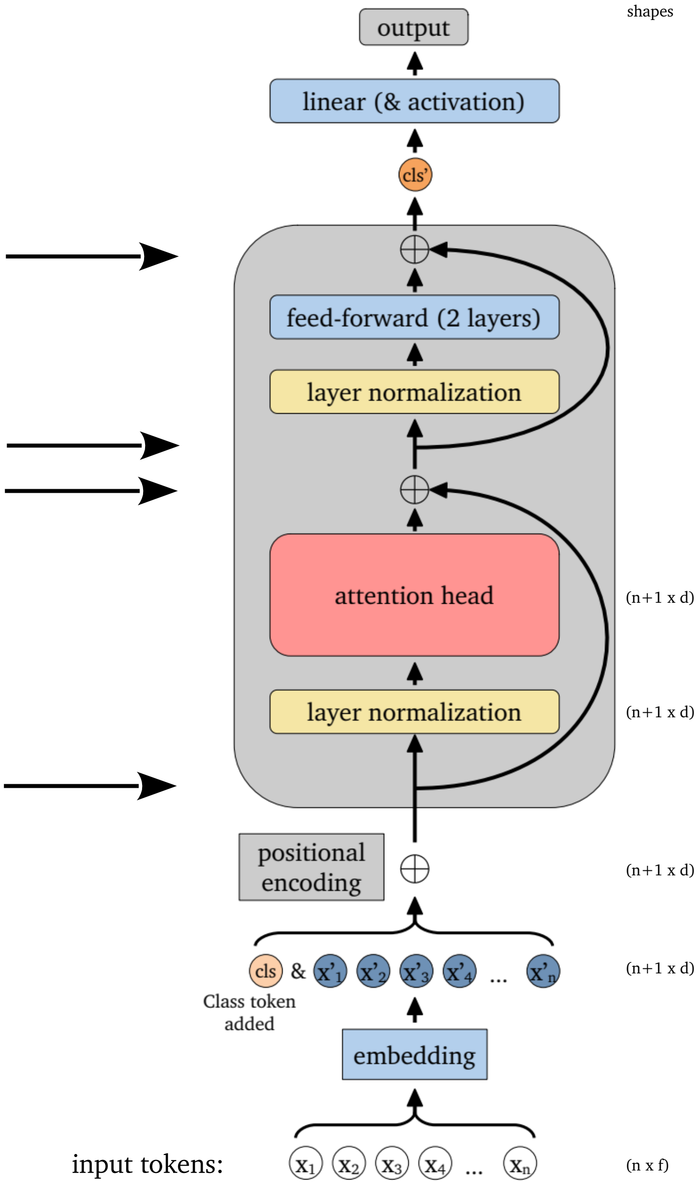
5 minute break

Residual connection

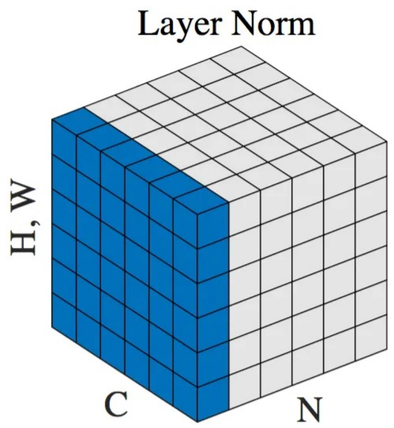
Residual connections

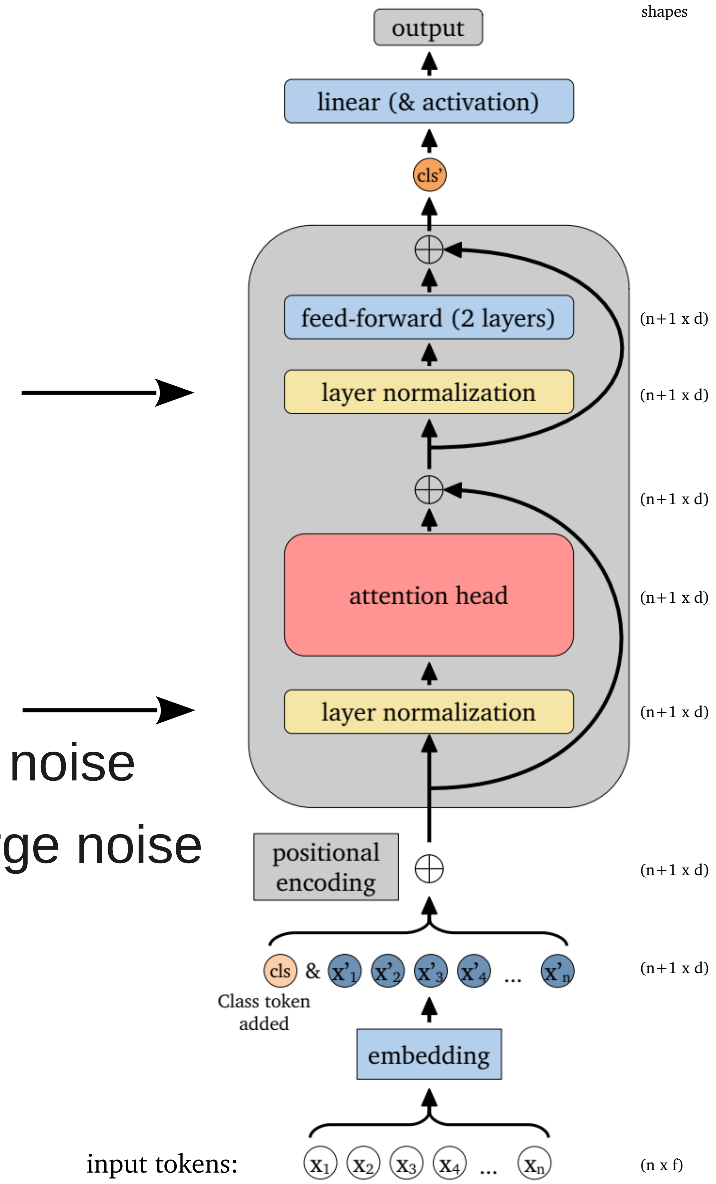
- Learn residual: $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$
- Improved gradient propagation
- Previous information easily "kept"





Layer Normalization





Layer normalization

- Stabilize training
- Normalize across features
- Useful for transformers
  - Usually small batches → batch norm: large noise
  - Varying sequence length → batch norm: large noise

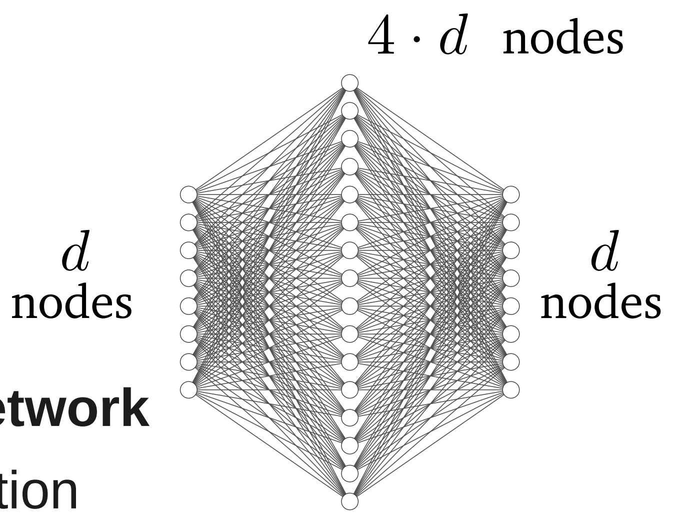
Why don’t we use batch normalization?

(a) Layer normalization causes less noise for varying sequence lengths

(b) We can also use batch normalization

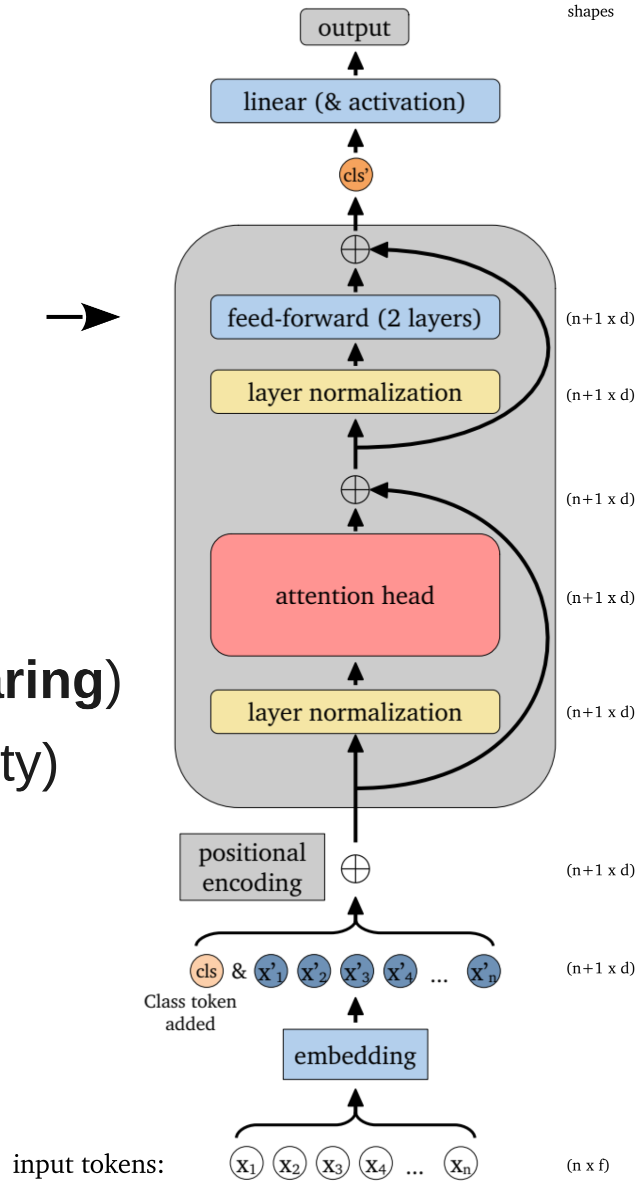
(c) In transformer training batches are usually too small

Feed-forward part



Fully-connected network

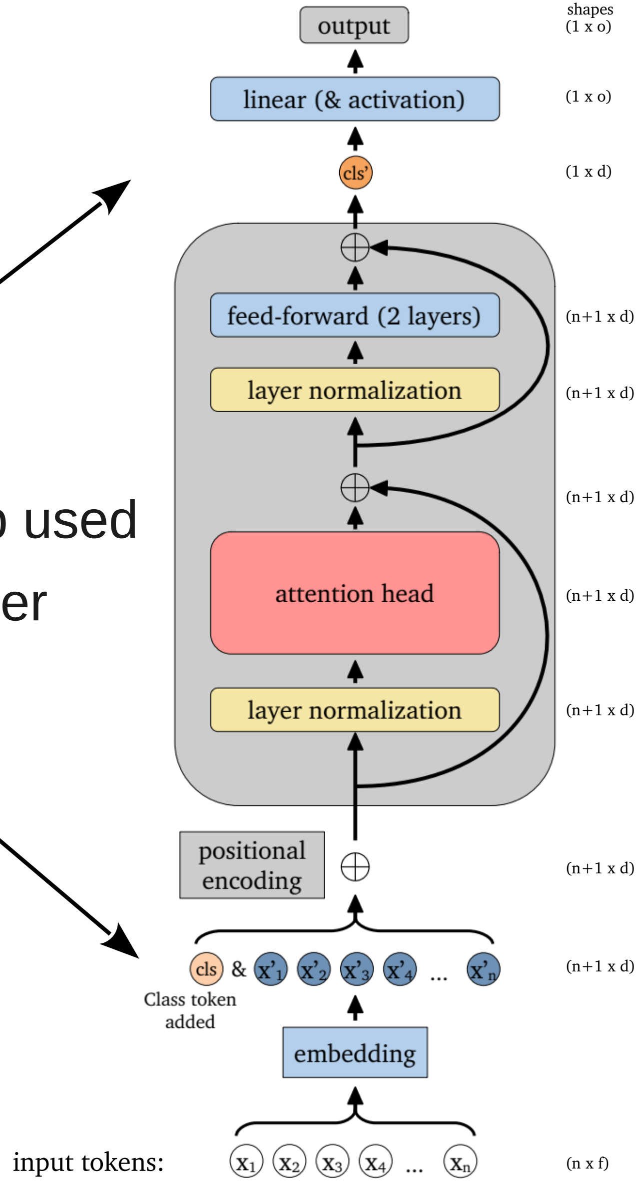
- Affine transformation
  - Includes bias (not only projections)
- Operates on each token in parallel (**weight sharing**)
- Often 2 layers (hidden layer holds larger capacity)



CLS (class) token

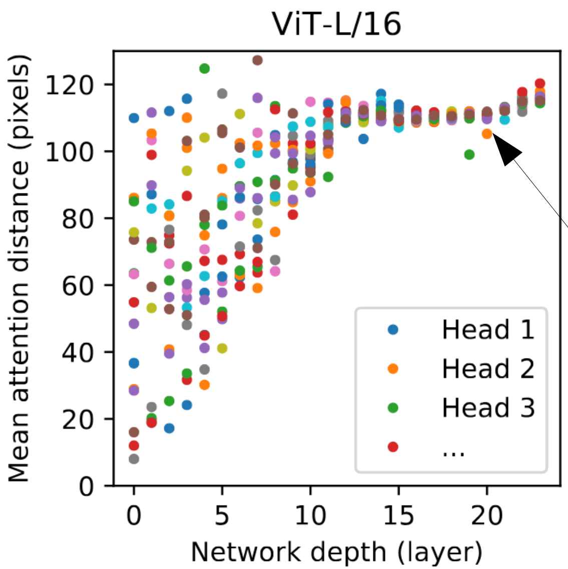
Class token

- Used for the final prediction
  - Different to RNNs where usually last time step used
- CLS token can be interpreted as memory register
  - Store and read from that token



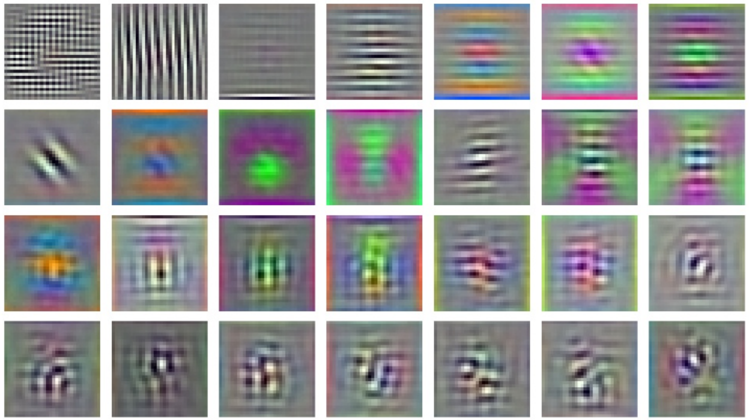
Transformer Networks

- Interestingly transformers show similar behavior as CNNs
  - Visualized filters in first layers very similar
- Deeper layers focus on long-range correlations



This is almost impossible to be extracted by CNNs

RGB embedding filters
(first 28 principal components)

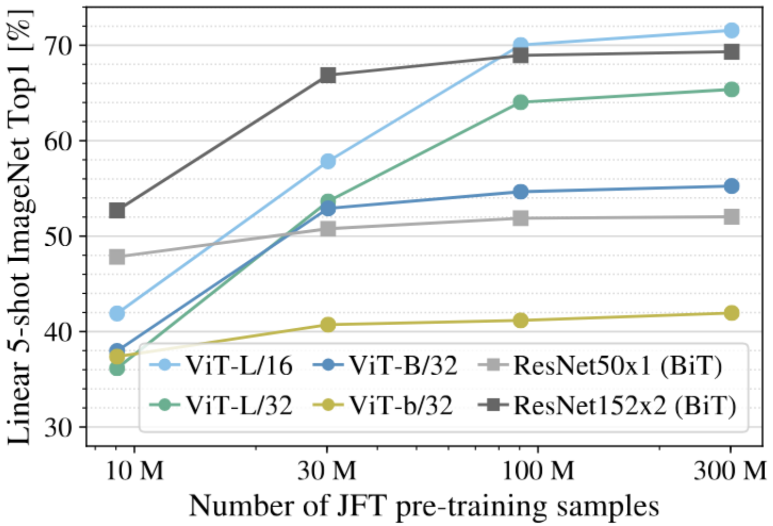


Similar to:
Gabor filters, edge detection, color detection

https://arxiv.org/abs/2010.11929

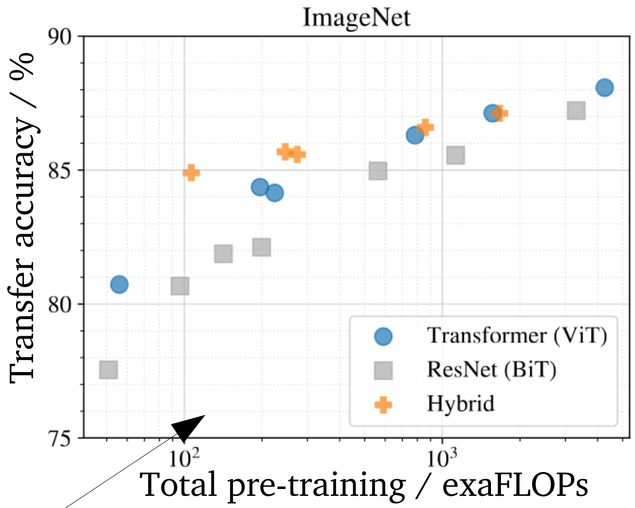
Scaling laws of transformer networks

- Transformers outperform CNNs on large data sets
- CNN prior (locality, translational invariance) powerful, if less information available
- Hybrid approach: first CNN layers followed by ViT backbone → performs best



JFT-300M
Internal Google data set with
300M images, 1B labels

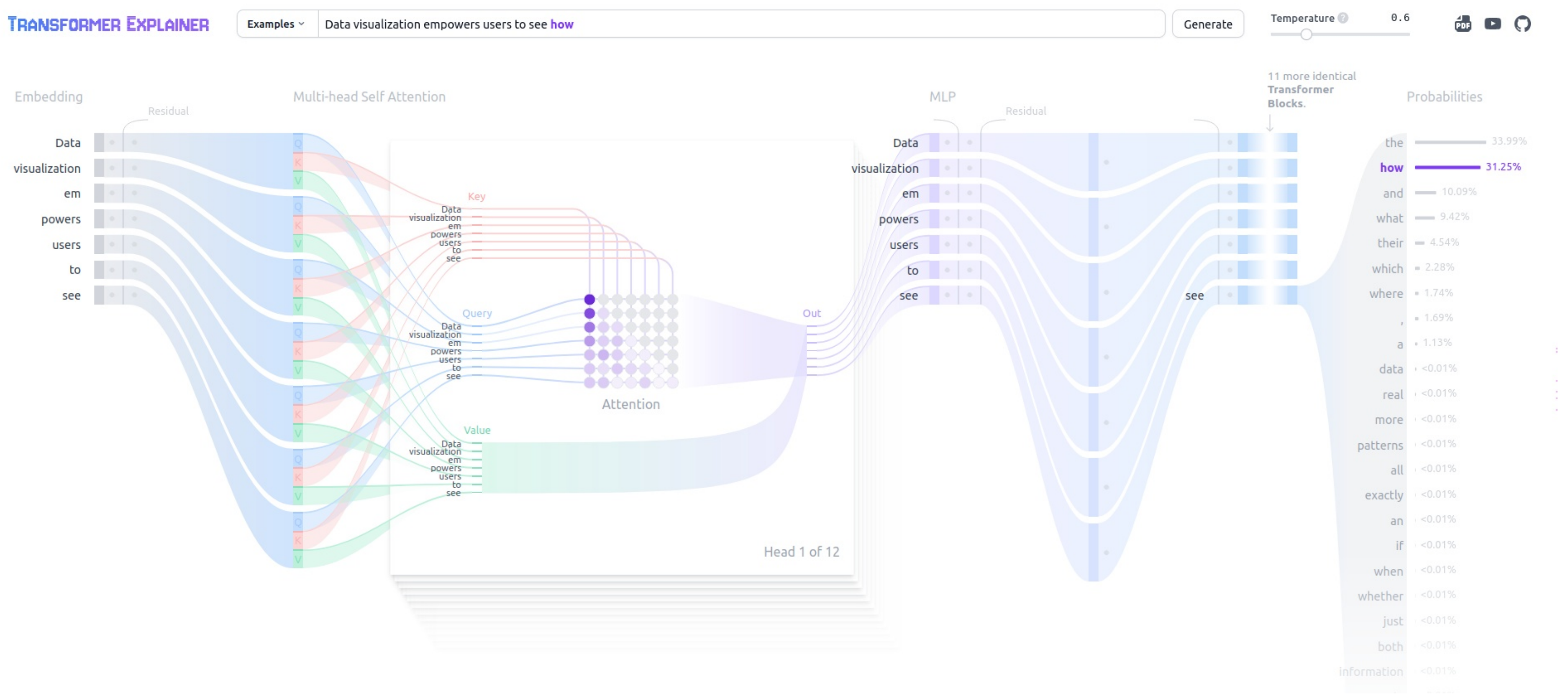
Transformers (usually)
always pre-trained!



Consumer GPU: (1k€ - 5k€)
Single GPU: ~ 50 terraFLOPs
→ 6h for 1 exaFLOPs
Second bin: 1 month training time!

https://arxiv.org/abs/2010.11929

Transformer explainer



https://poloclub.github.io/transformer-explainer/

Large Language Models (LLMs)

- Modern LLMs are decoder only architectures → text *generators*
  - Transformers trained at scale (compute, data, capacity)
  - Pre-trained O(10% internet)
- Pre-training via next token prediction (auto-regressive)

Gravity	→ makes	Mitochondria	→ produce
Gravity makes	→ apples	Mitochondria produce	→ ATP
Gravity makes apples	→ fall	Mitochondria produce ATP	→ through
Gravity makes apples fall	→ down	Mitochondria produce ATP through	→ respiration

  - human knowledge is shared and distributed via language
  - Models train on many tasks in parallel!

- This makes powerful text *generators* (next token predictors!)

Large Language Models (LLMs)

ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

- LLMs are one (few) shot learners!
  - Give one (few) examples and it will generalize well
- Predicting next tokens is hard → context helps a lot

Gravity [next token → hard (is, makes, defines …)

Gravity makes apples fall → easy (down)

- Massive pre-training creates powerful feature representations
  - Question answering, translation, classification, summarization
- Transformers are great in pattern matching
  - Simplified: jump to already learned similar example

Output you see

- State-of-the-art models: use scratch pad reasoning
  - [\think imitate reasoning, pull web info to context \think \answer … \answer]

Bonus: Try PointCloud Transformer

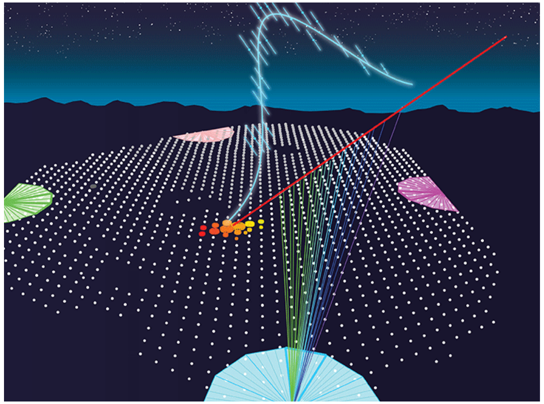
Model – add:

- Implement MLP
- Add MultiHeadAttention layer
- Add MLP to transformer block
- Add DNN output

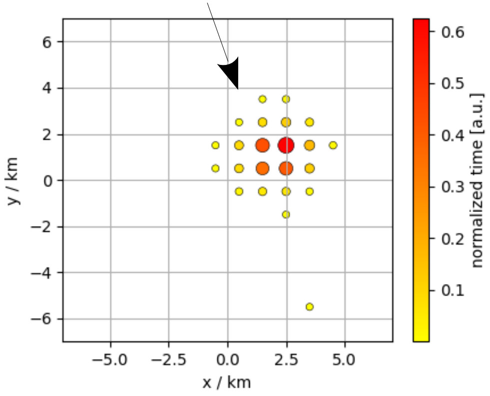
Model – modify:

- Numbers of transformer blocks
- Hidden layer size
- Optimizer, learning rate, epochs

➢ **Can you achieve a good energy resolution?**



Use only triggered stations



Open in Colab

https://github.com/jglombitza/tutorial_nn_airshowers/

Summary

- We only touched surface of deep learning in the last days
  - Field is rapidly evolving
- What's newer not always better for <u>your</u> task: tree learning still powerful *(if you have low dimensional data / small datasets)*
- You're now prepared for exploring advanced neural network stuff

Check out more advanced stuff

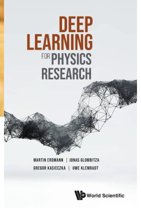
<u>Covered basics</u>

- DNN training
- Generalization, Regularization
- FCN, CNN, Transformers
- Keras / tensorflow basics
- First training experience

Tryout Deep Learning Yourself!

Find many physics examples at:
http://www.deeplearningphysics.org/

For example:
- CNNs, RNNs, GCNs
- GANs and WGANs
- Anomaly detection, Denosing AEs
- Visualization & introspection and more



https://github.com/DeepLearningForPhysicsResearchBook/deep-learning-physics

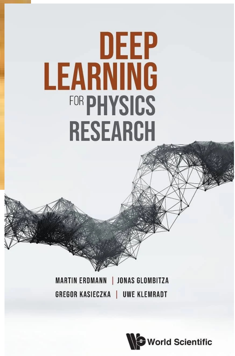
Tryout
Deep Learning
Yourself!

Find many physics examples at:

http://www.deeplearningphysics.org/

For example:

- CNNs, RNNs, GCNs
- GANs and WGANs
- Anomaly detection, Denosing AEs
- Visualization & introspection and more



Shapes during transformer operations

shape: 1 x d

shape: (n+1) x d

shape: (n+1) x d (after concat)

shape: (n+1) x d’ (each head)

shape: (n+1) x d (using add)

shape: (n+1) x d

shape: n x f