Jonas Glombitza
jonas.glombitza@fau.de

Astroparticle School 2025
Waischenfeld, Fraunhofer Research Campus
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 — **Tuesday 1:15h**

- Training of deep neural networks
- Interactive training of neural networks

Hands-on — **Friday 1:30h**

- Convolutional neural network
- machine learning frameworks: Keras
- Implementation of deep neural networks

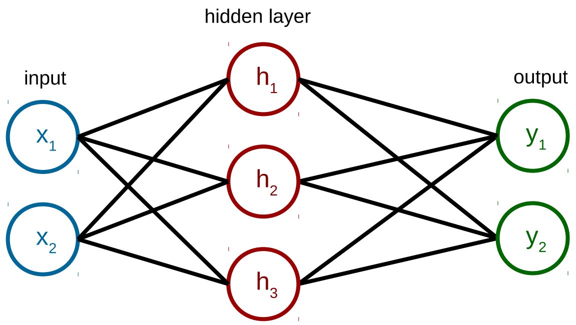
Set up & Requirements:
https://bitly.cx/iHcxS & https://bit.ly/3pyXRii
we will use **Jupyter Notebooks** and Keras / TensorFlow
we will use **Google Colab** → Google Account required

Recap: Neural Networks

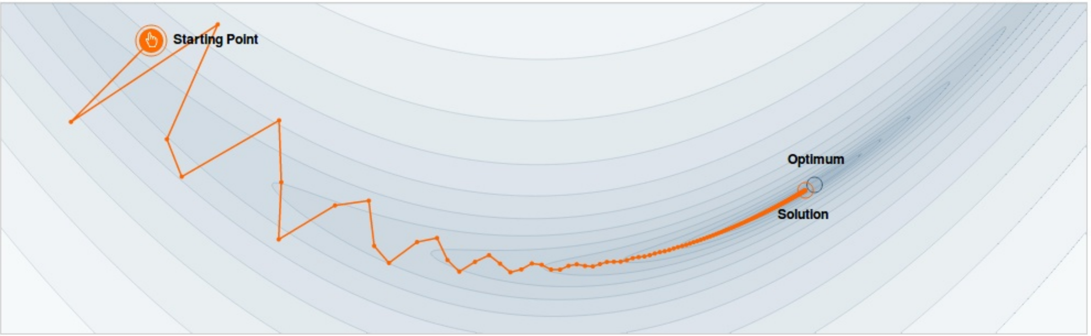
- Typical Machine Learning task
  - Labeled Data (x, y)
  - Model with adaptive weights $y_m(x, \theta)$
  - Objective $J(\theta)$
  - Optimization procedure

- 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

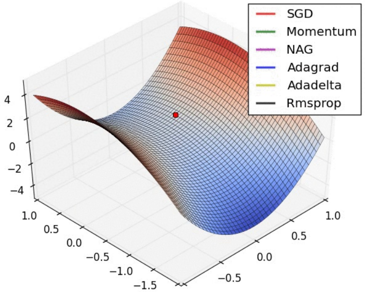
Mini-batch training: Use **stochastic gradient descent** algorithm (SGD)

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

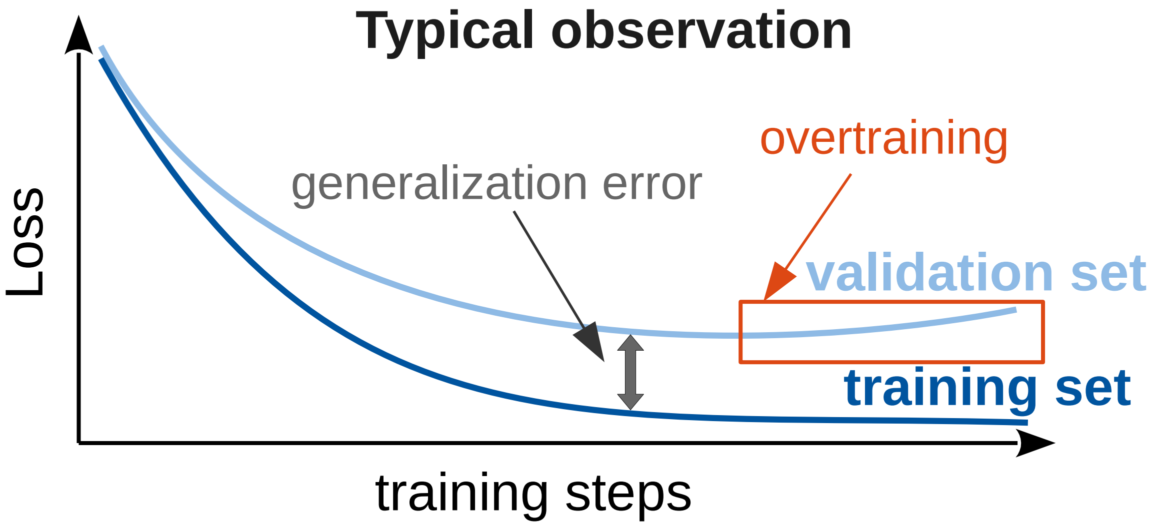
- Use adaptive learning rate for each parameter

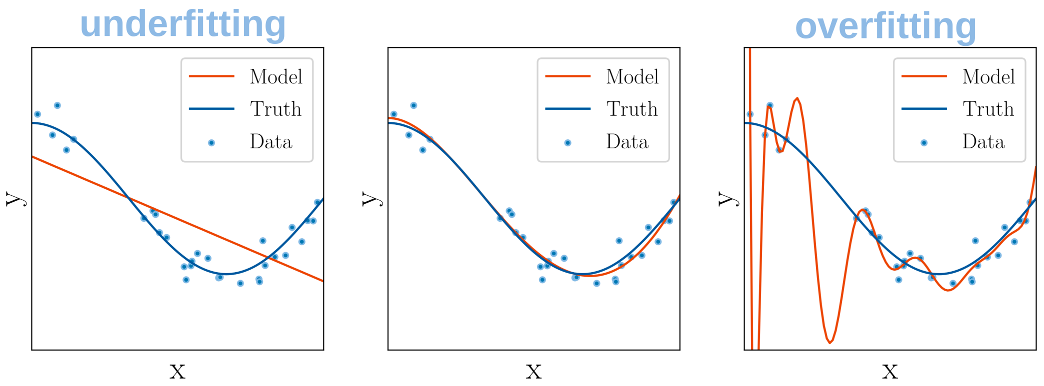


Recap: Under- and Overtraining

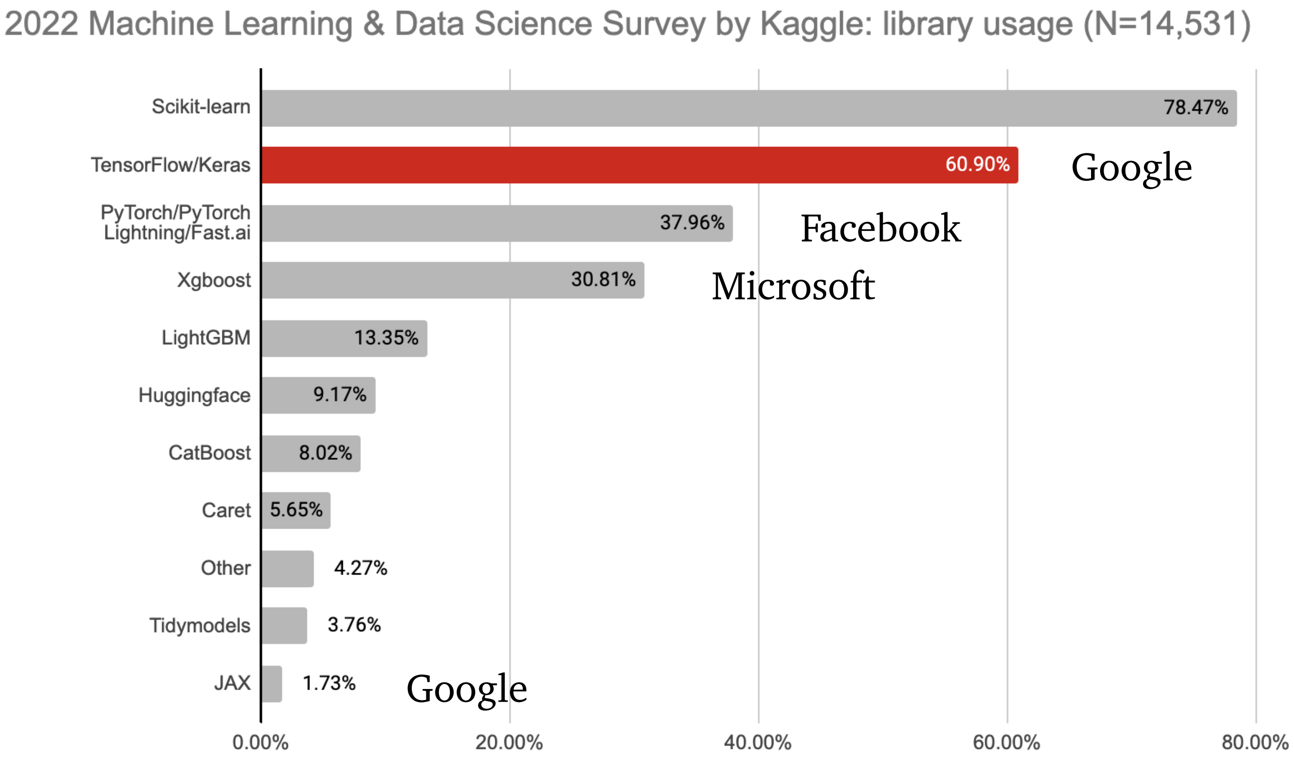
- Split data into 3 sets
  - training
  - test
  - validation
- What is a reasonably split?

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





Practice I



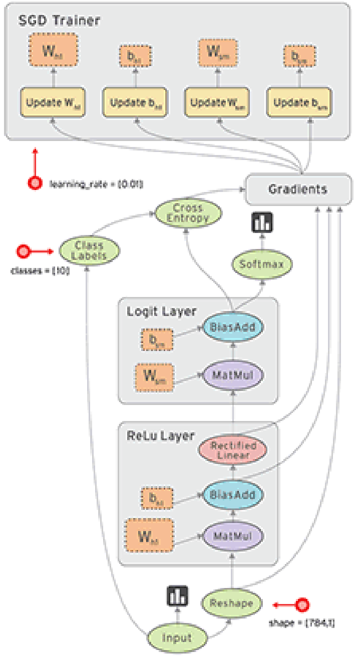
2022 Machine Learning & Data Science Survey by Kaggle: library usage (N=14,531)
Scikit-learn
78.47%
TensorFlow/Keras
60.90%
Google
PyTorch/PyTorch Lightning/Fast.ai
37.96%
Facebook
Xgboost
30.81%
Microsoft
LightGBM
13.35%
Huggingface
9.17%
CatBoost
8.02%
Caret
5.65%
Other
4.27%
Tidymodels
3.76%
JAX
1.73%
Google
0.00%
20.00%
40.00%
60.00%
80.00%

TensorFlow

“Open source software library for numerical computation using data flowing graphs”

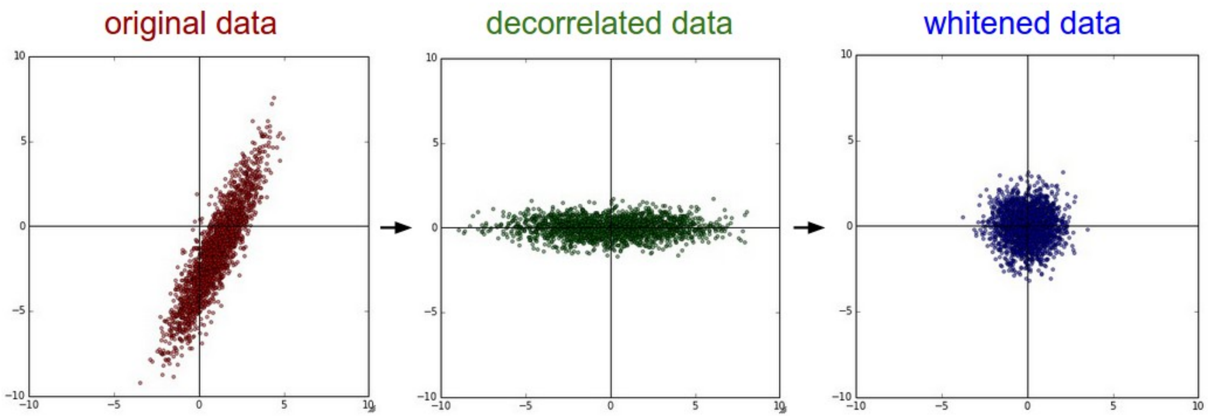
- **Nodes** represent mathematical operations
- **Graph edges** represent multi dimensional data arrays (**tensors**) which **flow** through the graph

- Supports:
  - CPUs and **GPUs**
  - Desktops and mobile devices
- Released 2015, stable since Feb. 2017
- Developer: Google Brain



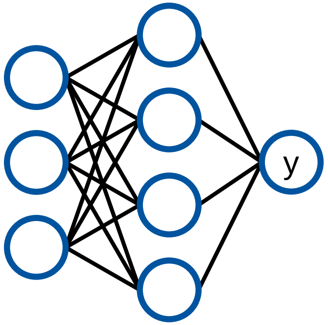
Data Preprocessing

- Input features of data set should be on same scale
  - Prevent particular sensitivity to few features
- Common normalization strategies
  - Limit range between [0, 1] or [-1,1]
  - Standard normalization: $\mu(x_i) = 0 \ \& \ \sigma(x_i) = 1$
  - Whitening: standard normalization + decorrelation

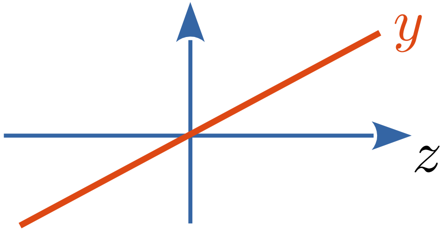


Classification vs. Regression

Regression



Linear

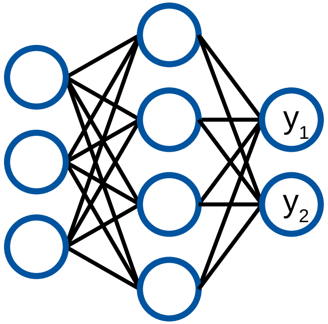


no activation function

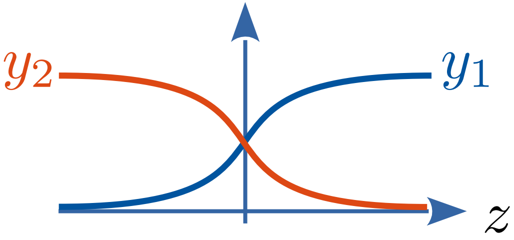
Minimize mean-squared-error

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

Classification



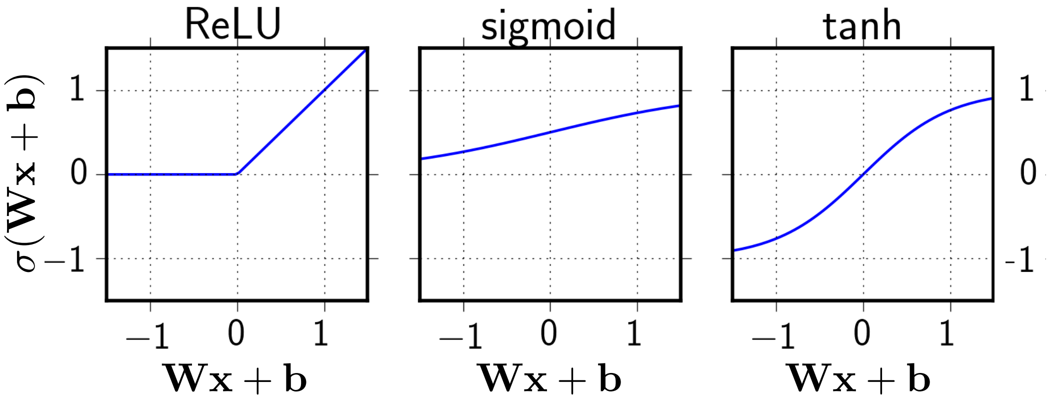
Softmax



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

Minimize cross entropy

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



What is a nice activation function?

(a) ReLU → since it’s so simple and has a simple and constant gradient

(b) Sigmoid → it’s very complex and inspired by biology

(c) Tanh → it is complex and also permits negative values

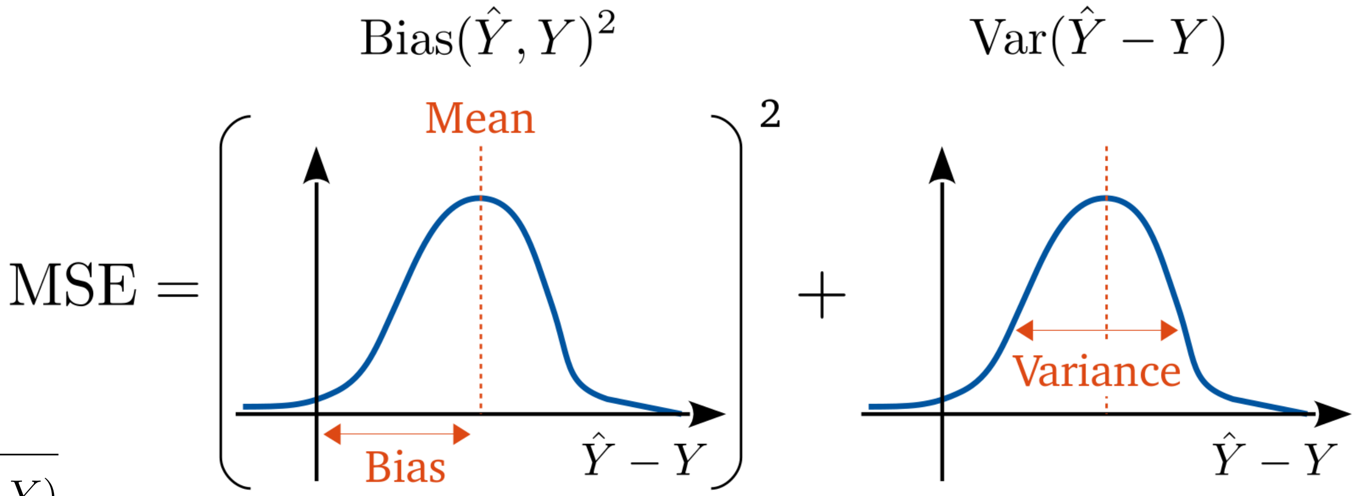
Metrics: Regression

Regression

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

Minimize mean-squared-error

Metrics:



Bias

→ often part of systematic uncertainty in experiments

Resolution: $\sigma_{\mathrm{res}} = \sqrt{Var(\hat{Y} - Y)}$

→ Resolution of an algorithms

Often used to quantify precision

"Minimizing the MSE, minimizes both bias and resolution of an estimator (your DNN)"

Metrics: Regression

Classification

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

Minimize cross entropy

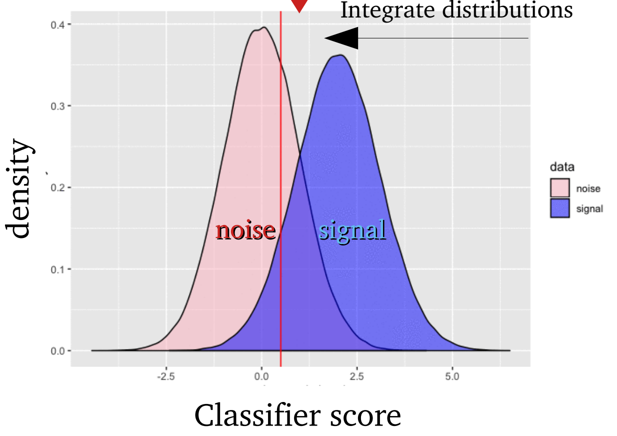
Metrics:

<u>Accuracy</u>: fraction of correction predictions

<u>Better</u>: Receiver Operation Characteristic (ROC)

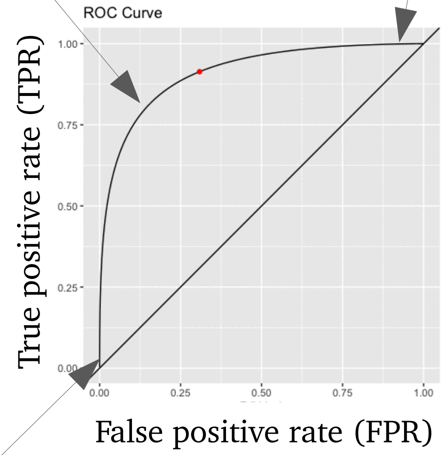
Only accept events above threshold
scan threshold → RoC

Integrate distributions



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

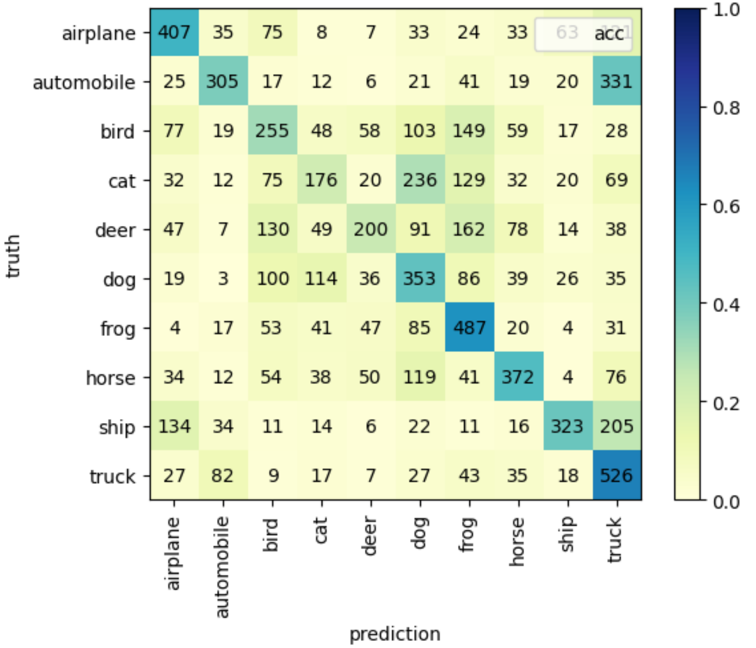
Select almost everything
TPR~1, FPR~1



Select almost nothing
TPR~0, FPR~0

Multi-class classification
→ no ROC possible

Confusion Matrix



Keras

- Will use Keras in this tutorial (TensorFlow backend) - https://keras.io
- High-level neural networks API, written in Python
- Concise syntax with many reasonable default settings
- Useful callbacks / metrics for monitoring the training procedure
- Nice Documentation & many examples and tutorials
- Comes with TensorFlow

How to train your Model?

I. Define Model
- Add layers, nodes, regularization, activation functions, ….)

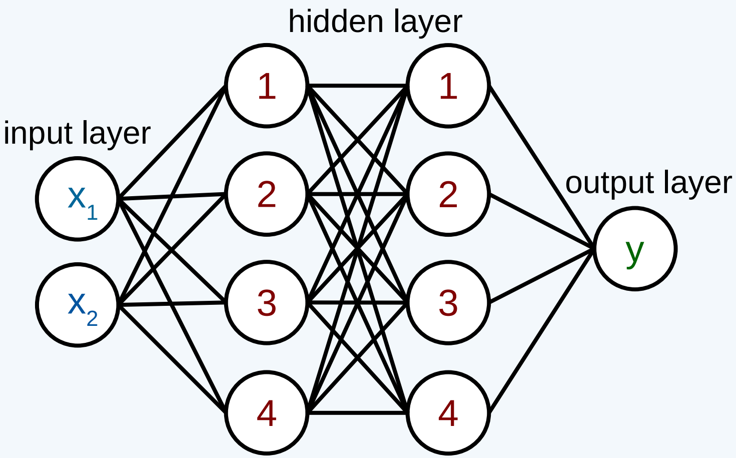
II. Compile Model
- Set Loss, optimizer settings and useful metrics

III. Fit Model
- Set number of iterations and train model on given data

```
from tensorflow import keras
layers = keras.layers
models = keras.models

# setup and train a 3-layer regression network with Keras
model = models.Sequential()
model.add(layers.Dense(4, activation='relu', input_dim=2))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(1, activation='linear'))
model.compile(loss='MSE', optimizer='SGD')
model.fit(xdata, ydata, epochs=200)
```

I
II
III



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

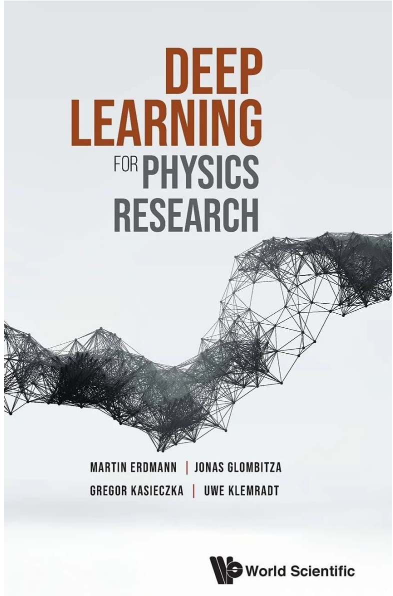
Tryout Deep Learning Yourself!

Find many physics examples at:

http://www.deeplearningphysics.org/

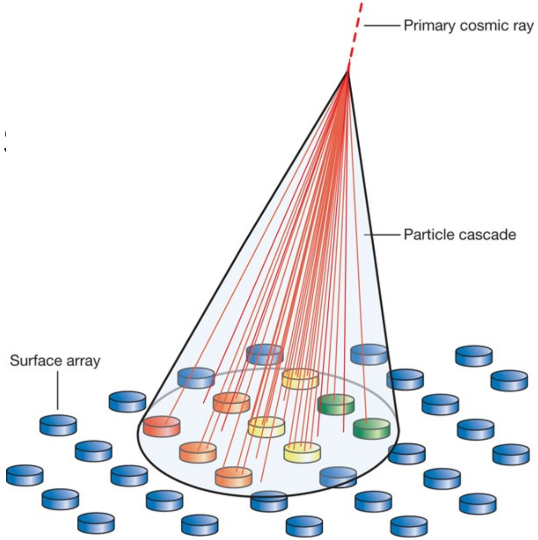
For example:

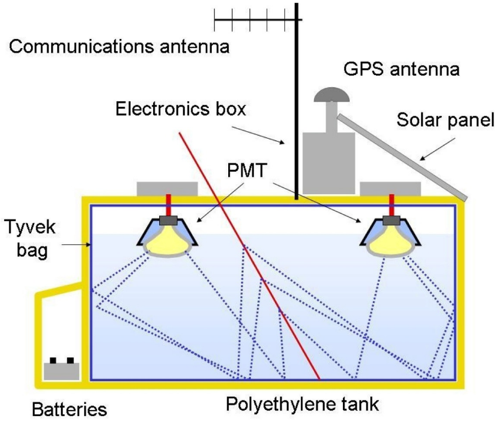
I. CNNs, RNNs, GCNs
II. GANs and WGANs
III. Anomaly detection, Denosing AEs
IV. Visualization & introspection and more

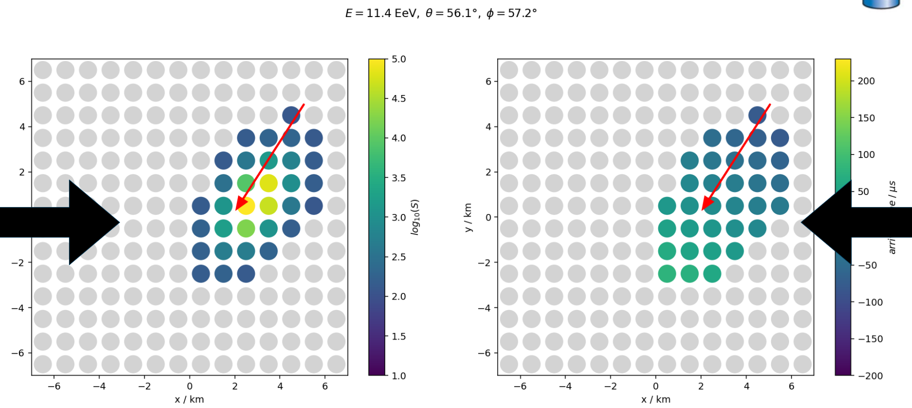


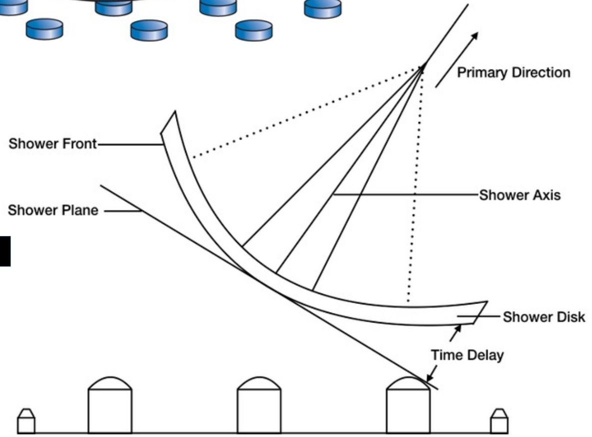
Air Shower Reconstruction

- Observatory for measuring cosmic-ray-induced air
  - 14 x 14 particle detectors, arranged in Cartesian grid (altitude of 1400 m)
  - particle detectors measure arrival time of the shower and deposited energy

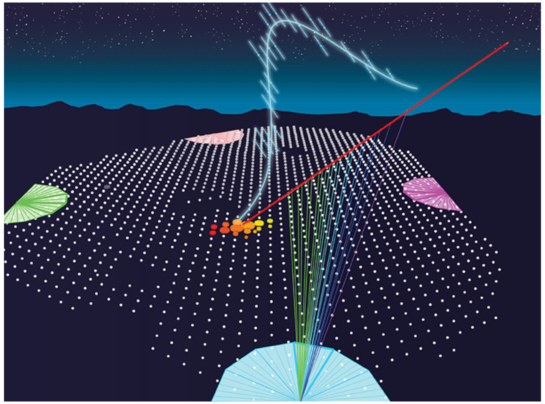








Air Shower Reconstruction - FCN

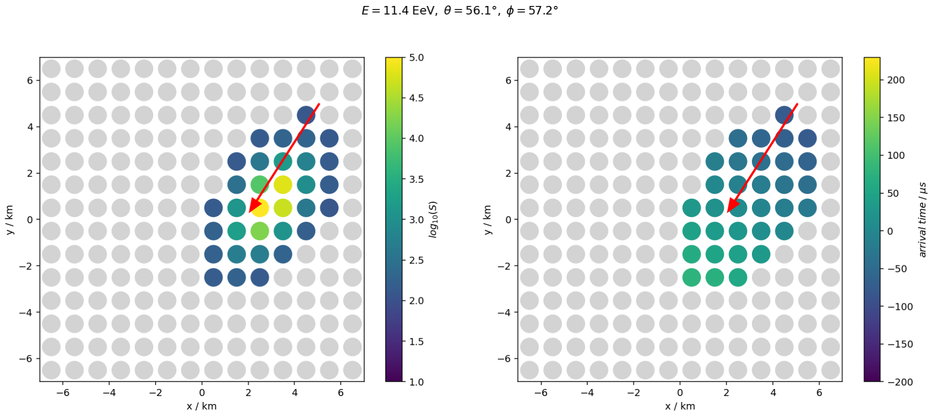


Now: OPEN tutorial at:

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

Open in Colab

Task



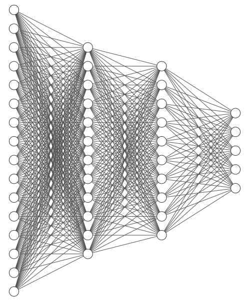
- Reconstruct energy of the shower
  - footprint is 2D image
  - cannot directly be used as input
    → reshape to a vector with length
    (14 x 14 x 2 = 392)
  - **Try to reach a resolution better than 4 EeV**

Results I

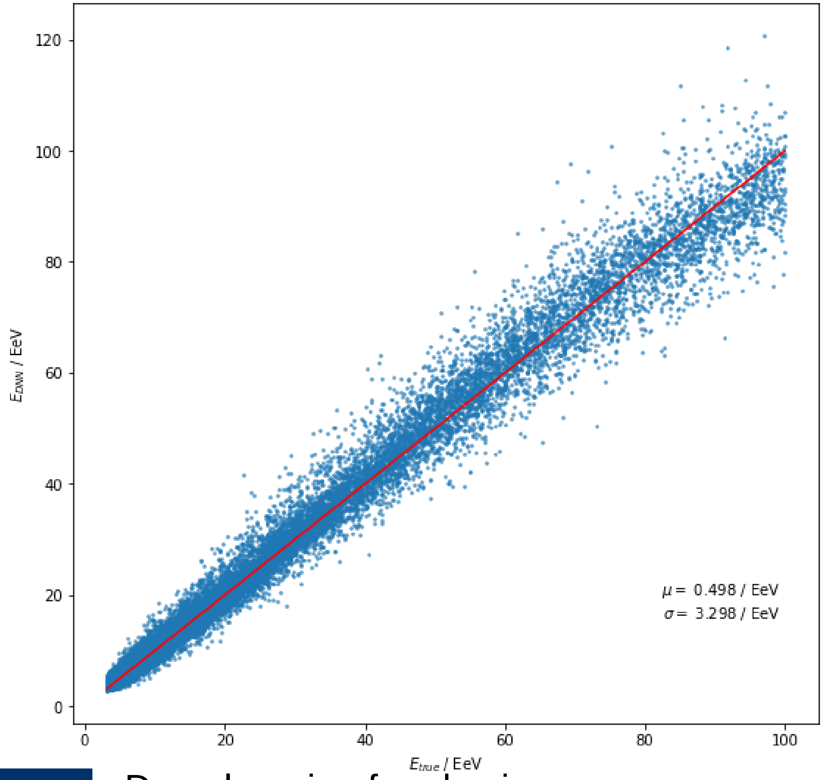
- Train fully-connected network as benchmark

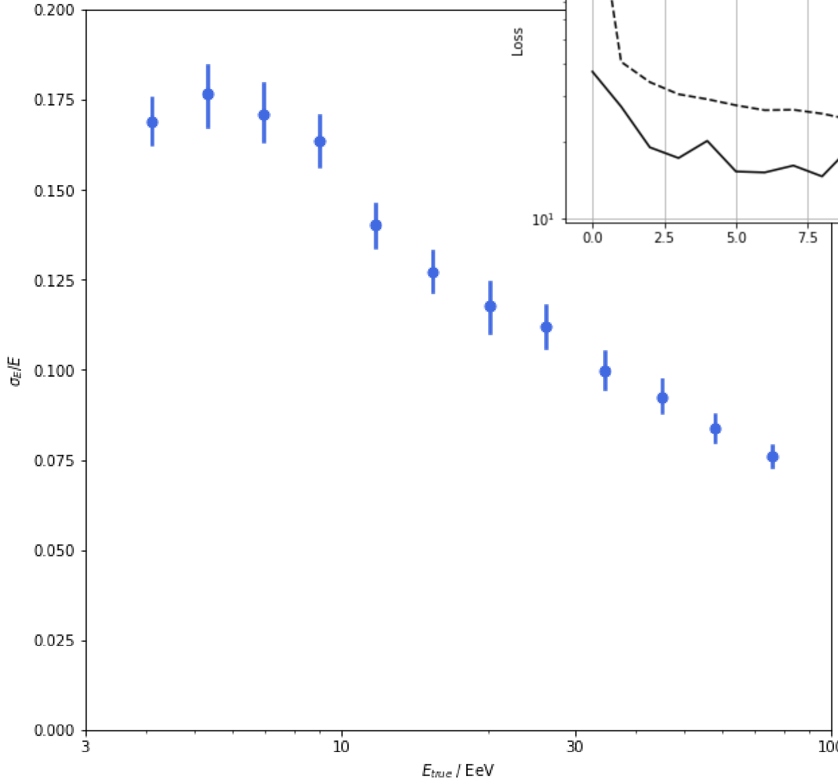
- Model – **add**:
  - additional layers
  - more nodes
  - regularization (Dropout)

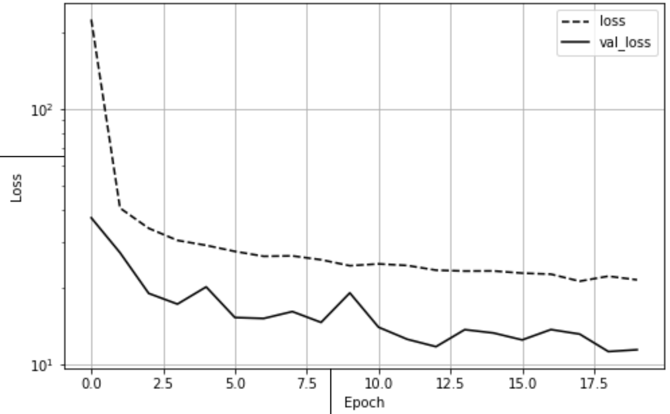
```
model = keras.models.Sequential()
model.add(layers.Flatten(input_shape=X_train.shape[1:]))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dense(100, activation="elu"))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(1))
```



Results II

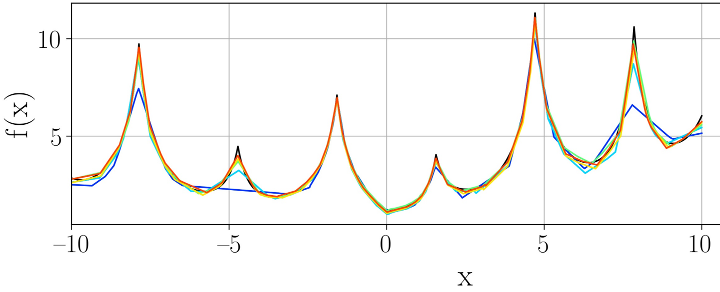

$\mu =$ 0.498 / EeV
$\sigma =$ 3.298 / EeV
E_{DNN} / EeV
E_{true} / EeV


σ_E/E
E_{true} / EeV


loss
val_loss
Loss
Epoch

“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 4 hours about
feed-forward networks only...

Dedicated time for

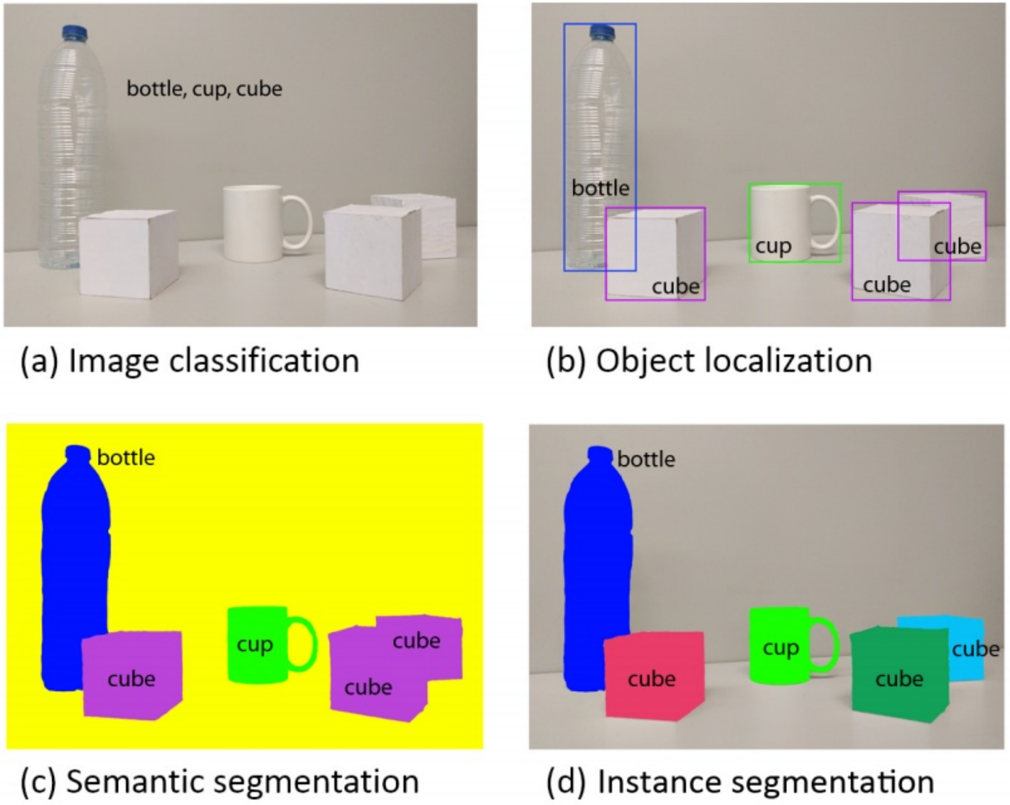
Your Questions

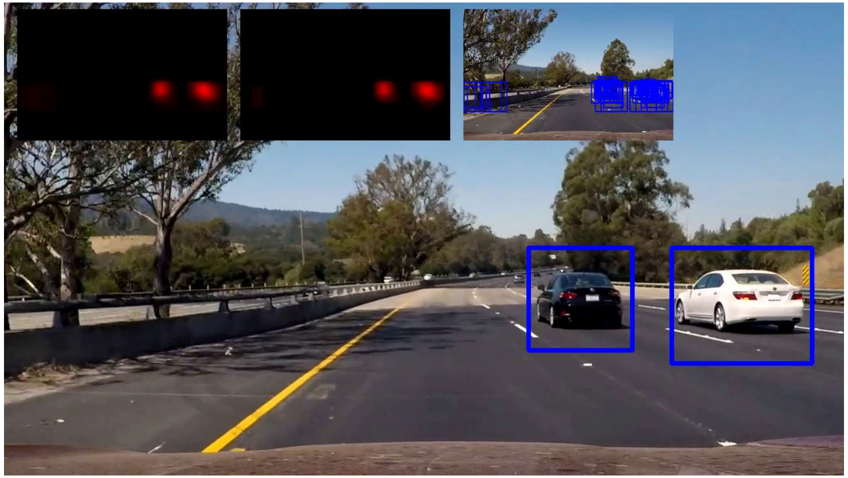
Natural Images

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….

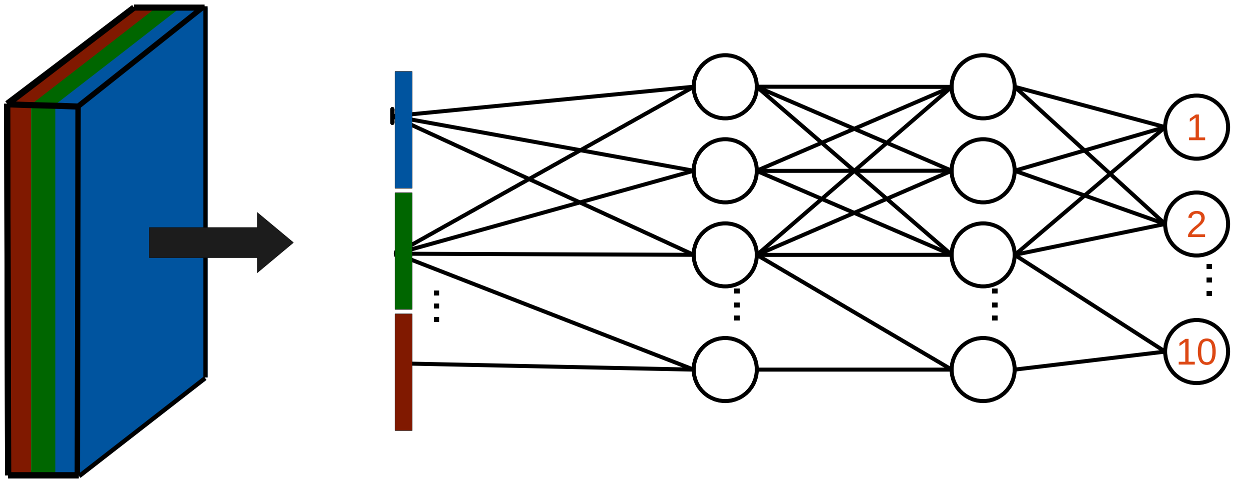
Computer Vision Tasks



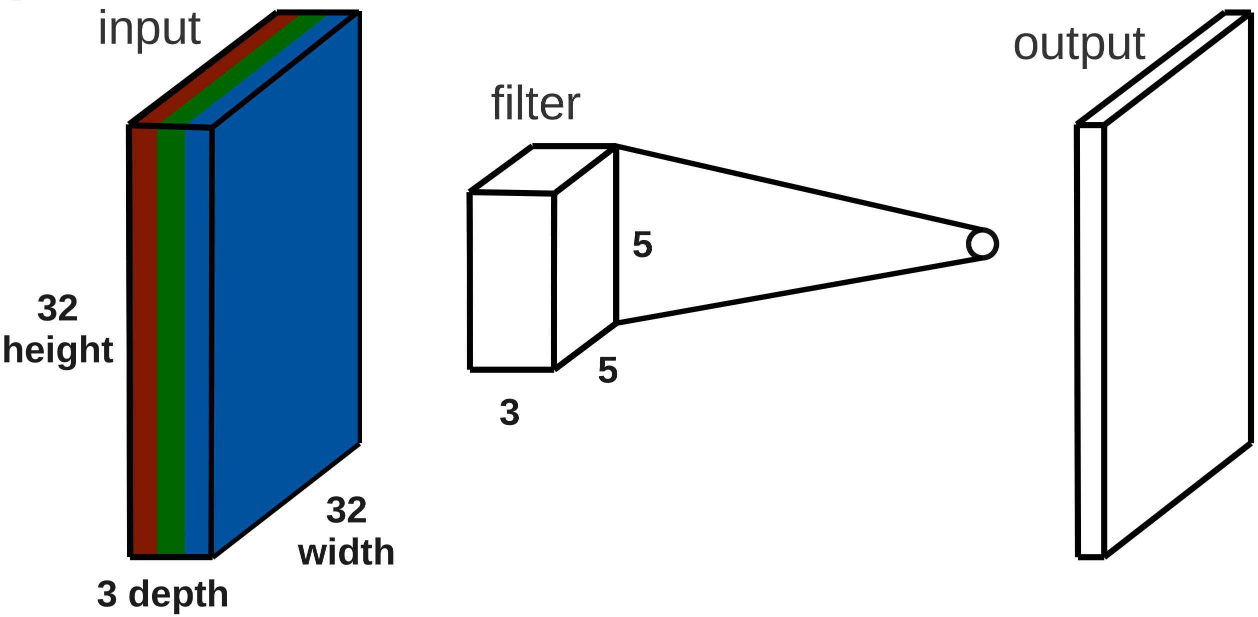


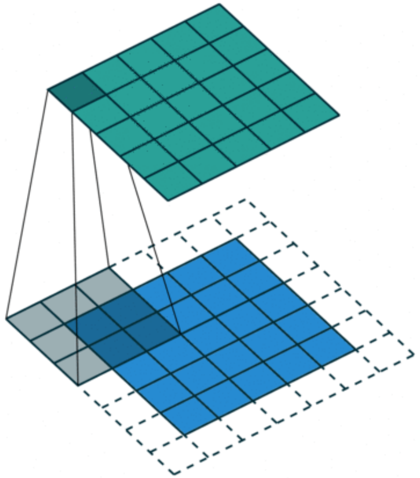
Fully Connected Network

- 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



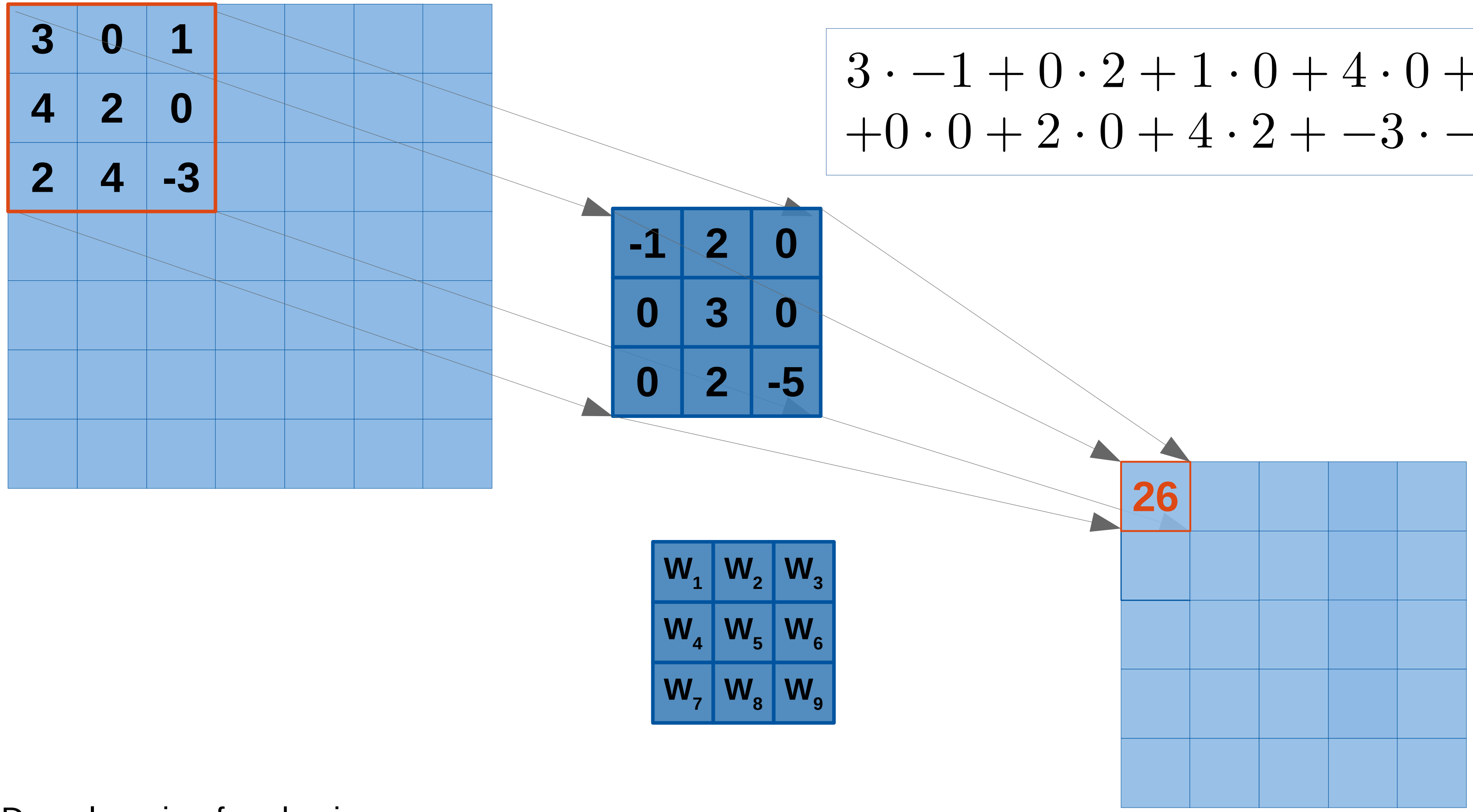


Paul-Louis Pröve,
Towards Data Science

- 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

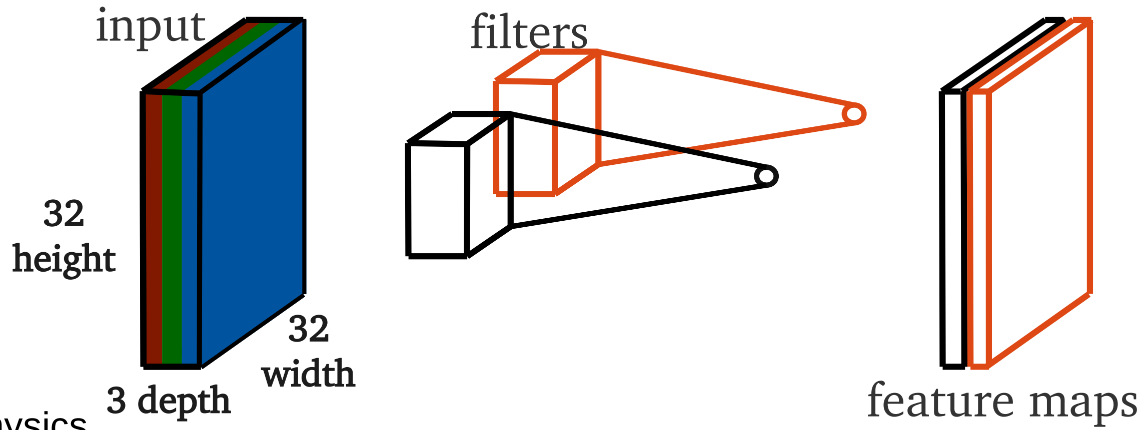
deep learning

Convolutional Networks

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

adaptive parameters

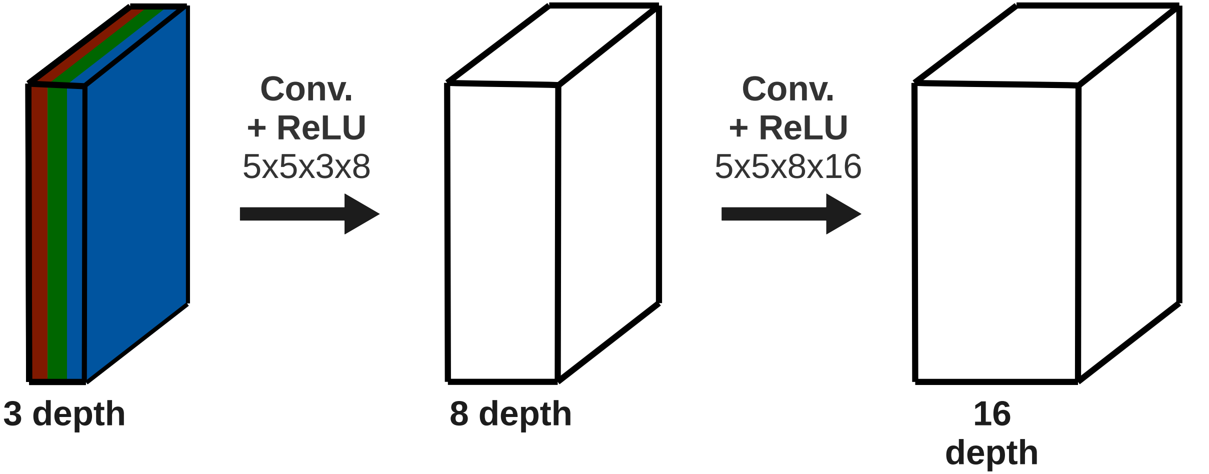
- scan input image for the presence of specific feature using **filters**
- use multiple filters and stack the results as **feature maps** (depth-wise stacking)

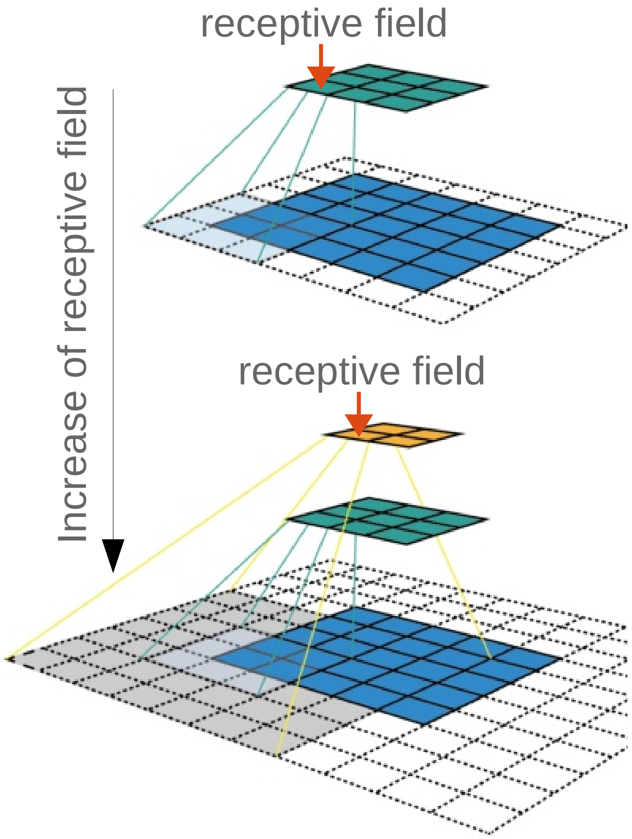


2D Convolutional Operation

Stack multiple convolutional layers + activations

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



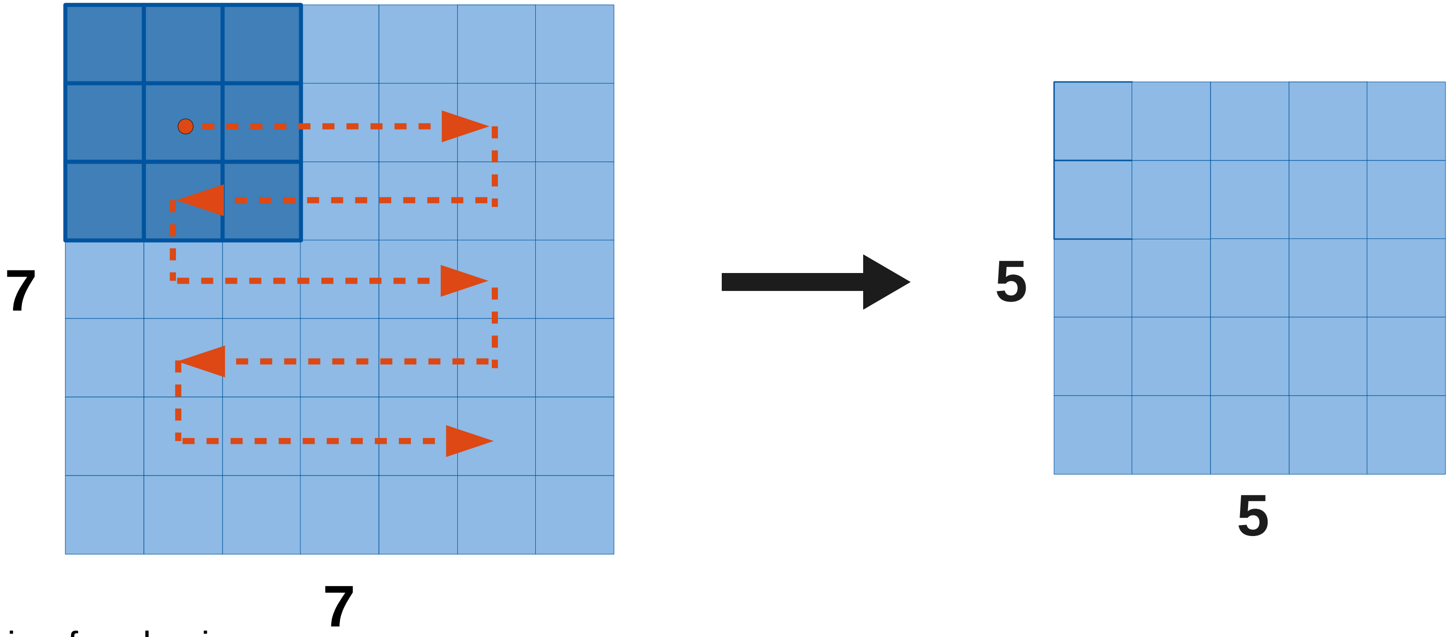


Spatial Output Size

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

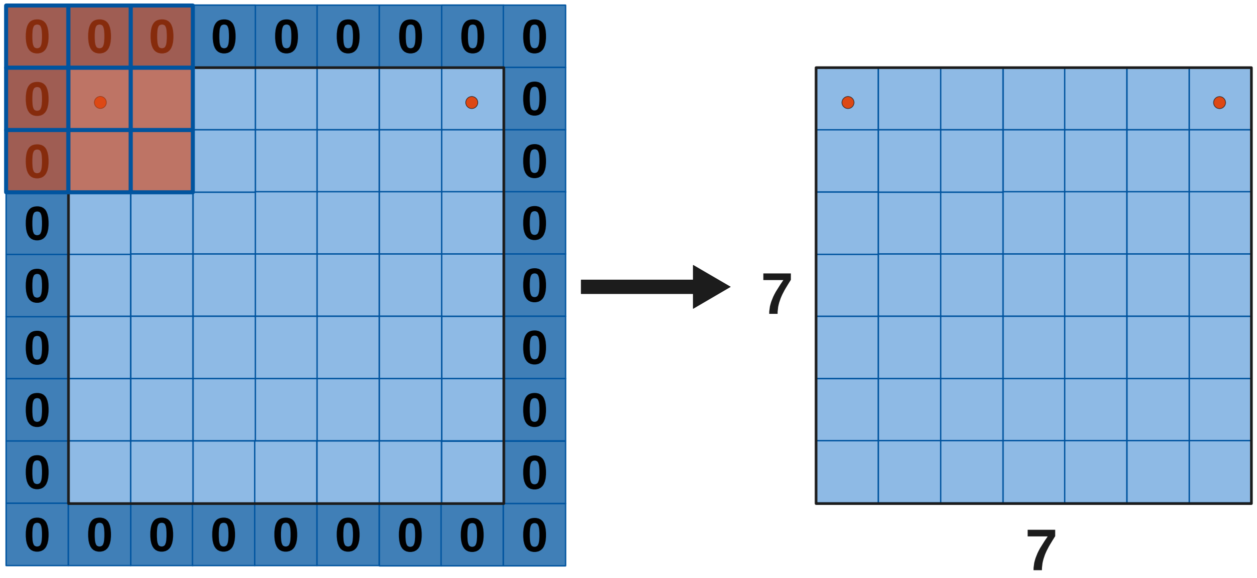


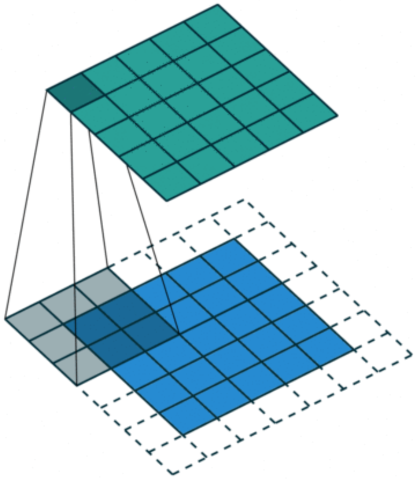
Padding

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





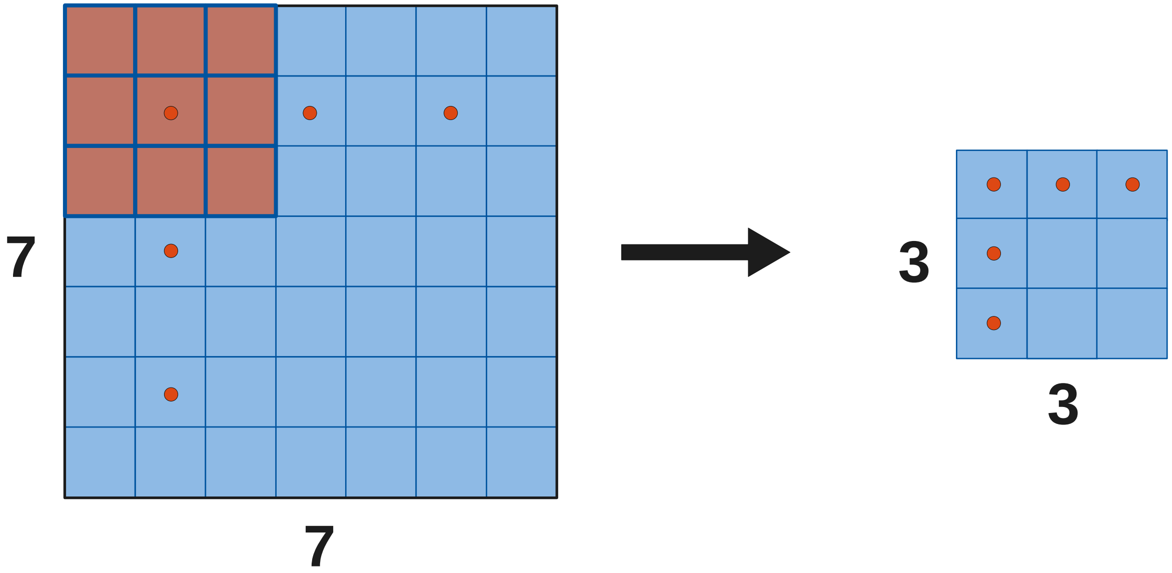
Paul-Louis Pröve,
Towards Data Science

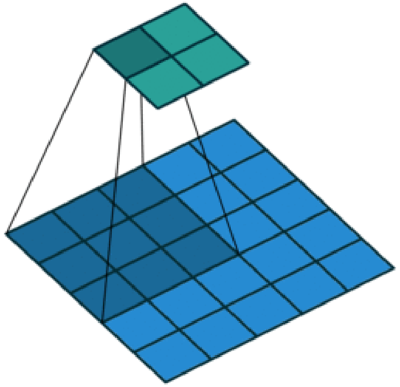
Striding

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

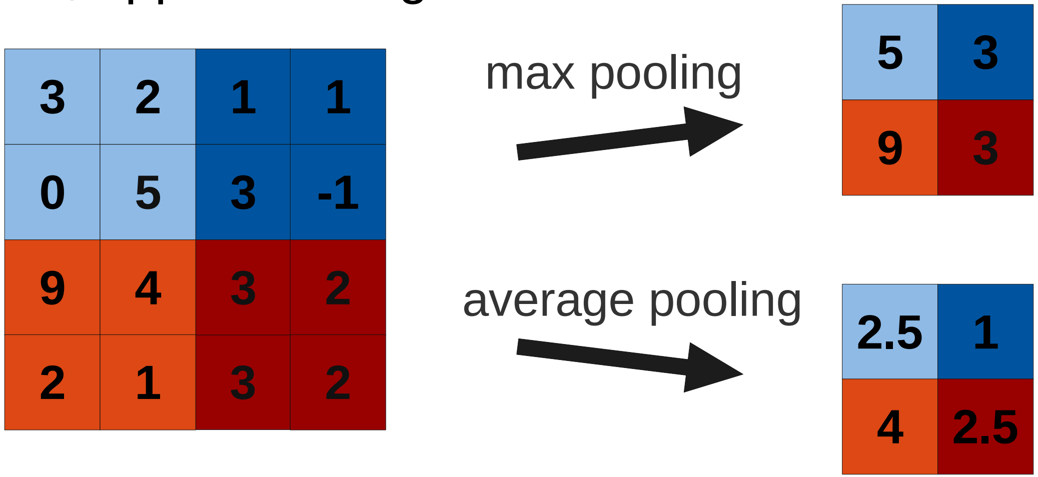
Average pooling: Take the mean of each patch → for some regressions preferable

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

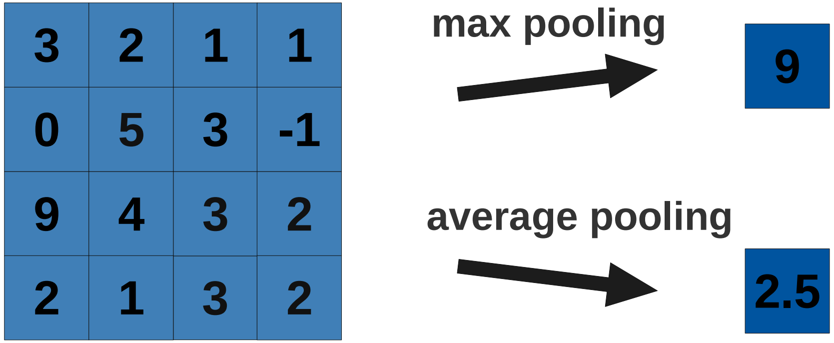


Global Pooling Operation

- 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

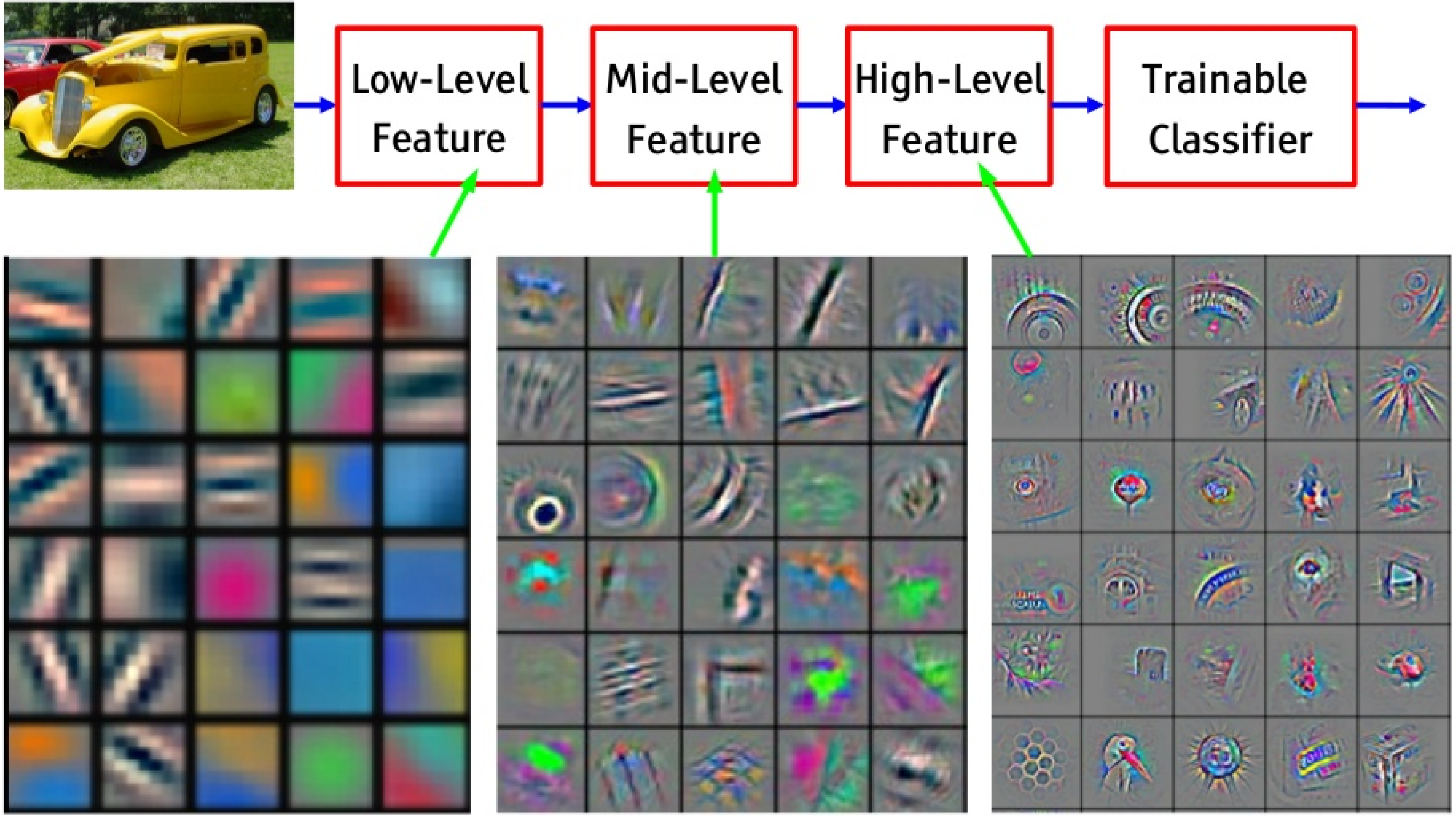


Visualization of CNNs – 5 mins

- 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



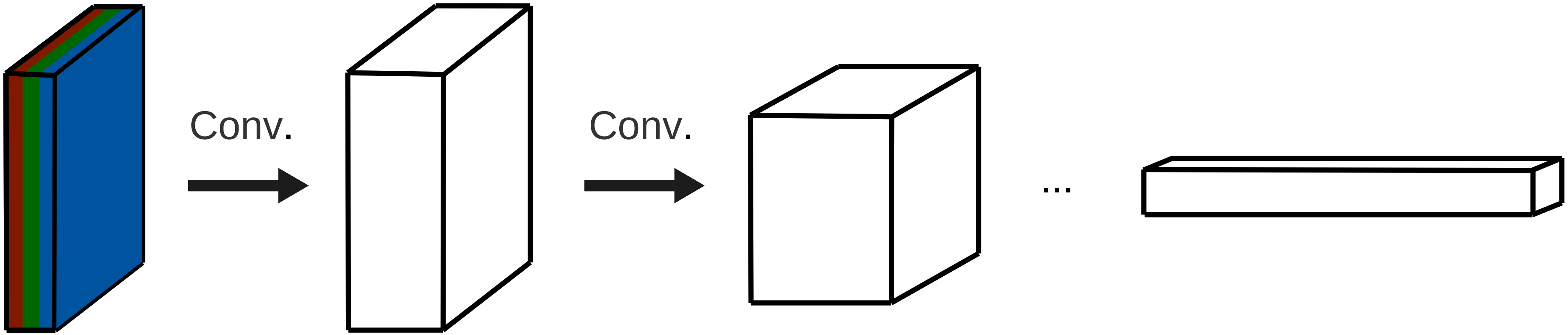
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

https://arxiv.org/abs/1311.2901

Convolutional Pyramid

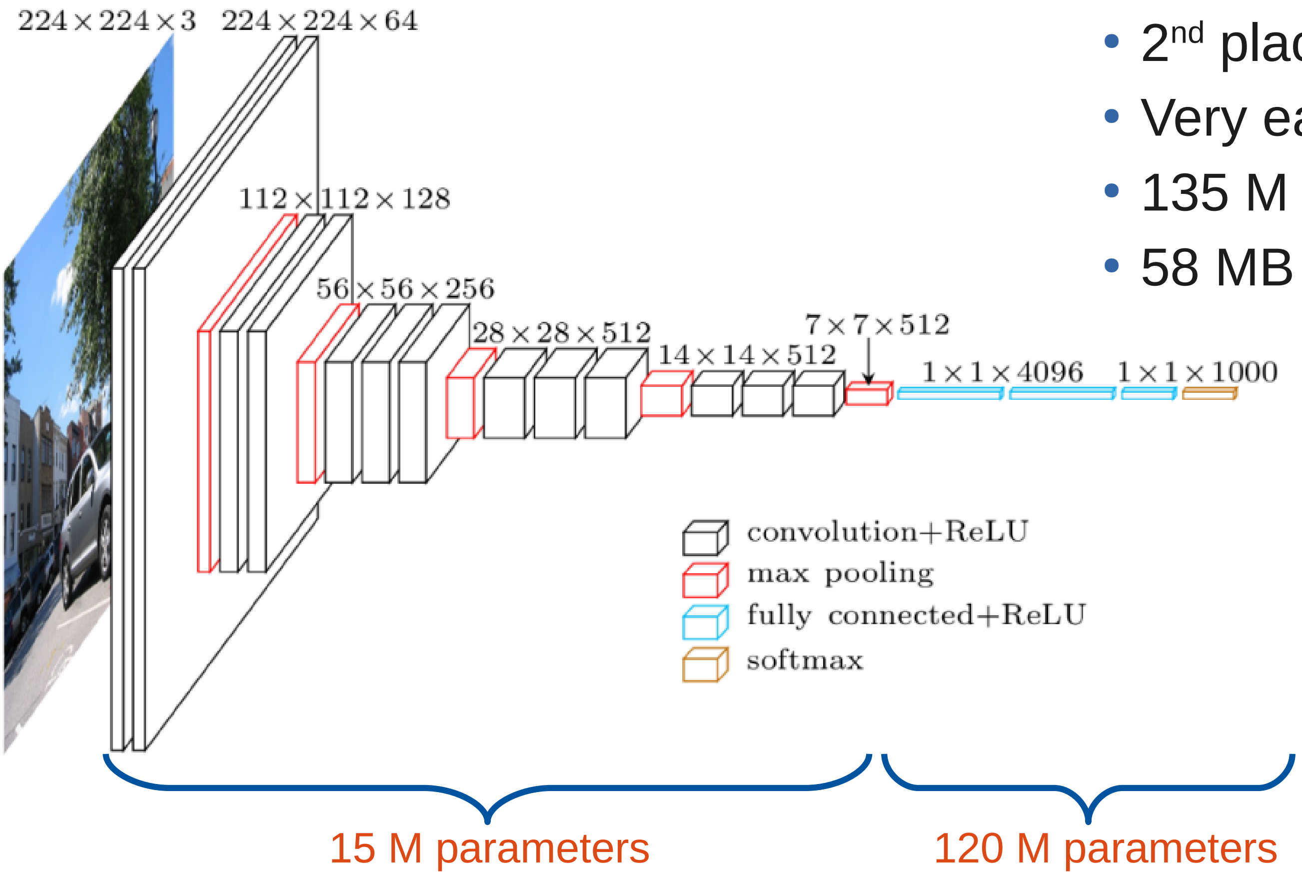
ConvNet architectures usually have a pyramidal shape. For deeper layers:

- 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

Question: What is a good filter size?

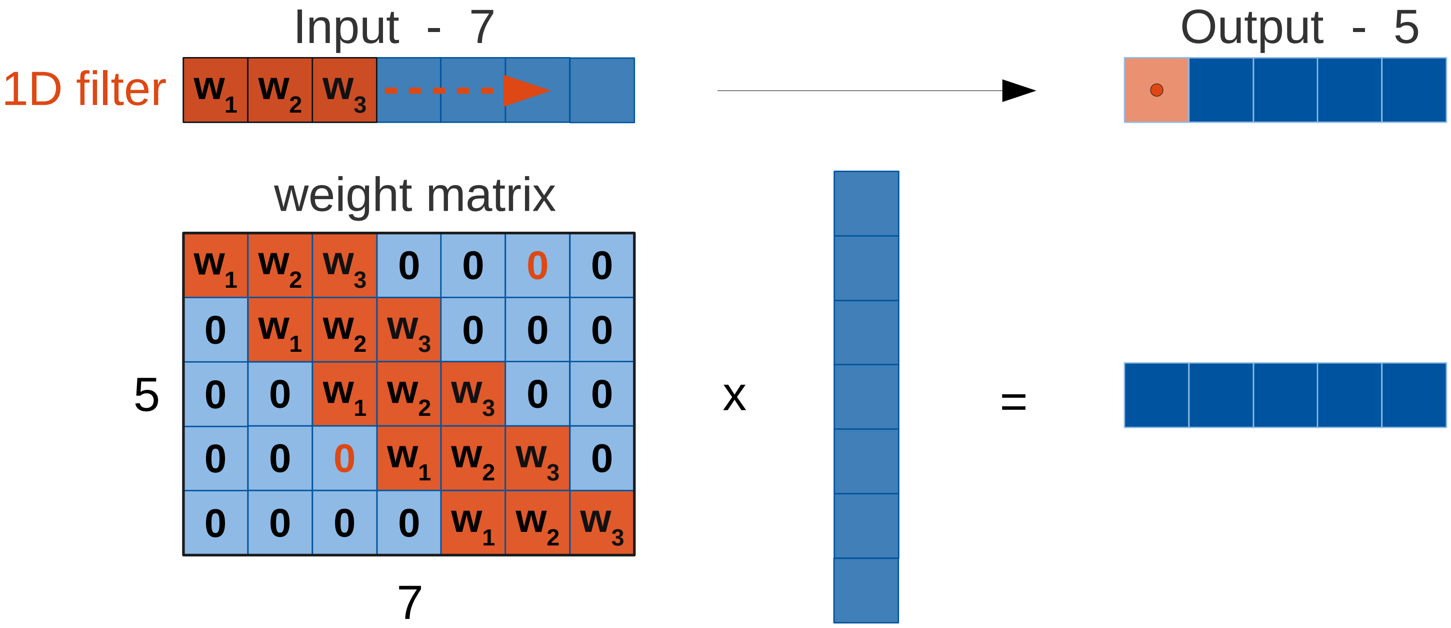
(a) Small (3 x 3)

(b) Large (50 x 50)

(c) Medium (20 x 20)

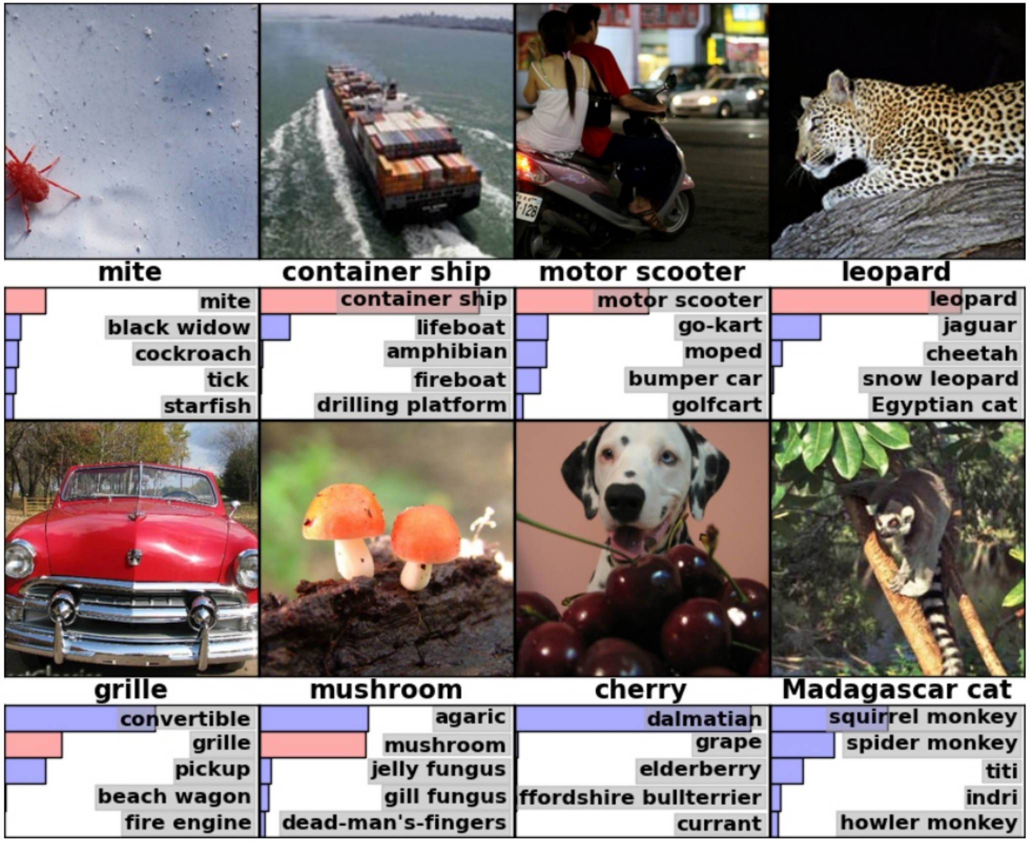
Convolutional Operation

- Fully connected layers are special case of convolutional layers

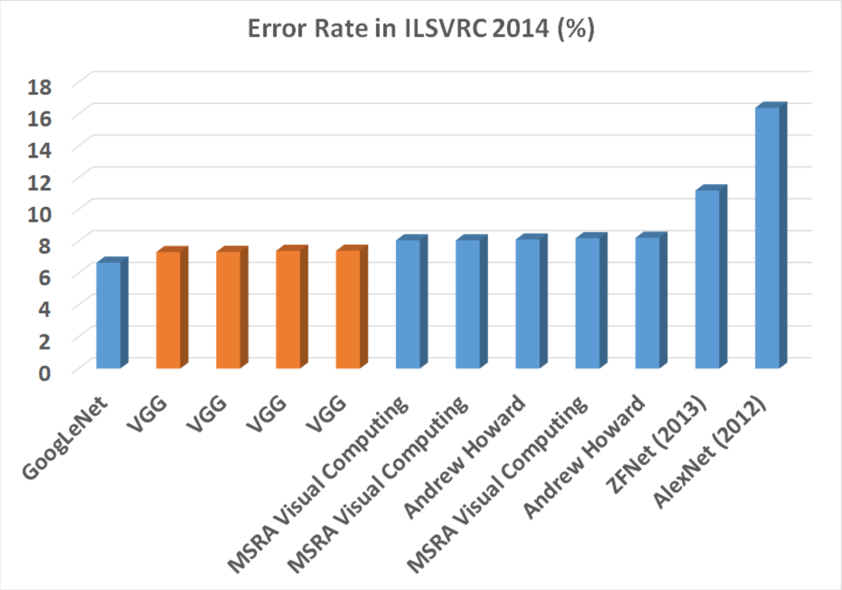


- Parameters greatly reduced due to **sparsity** and **weight sharing**
  - Strong prior on **local correlation** and **translational invariance**

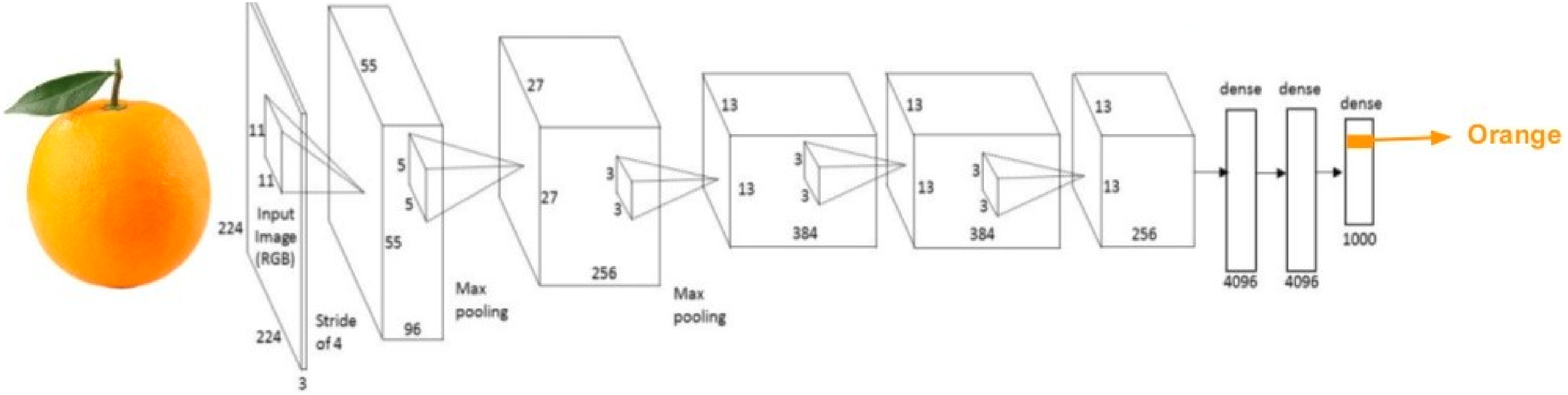
Results on ImageNet



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

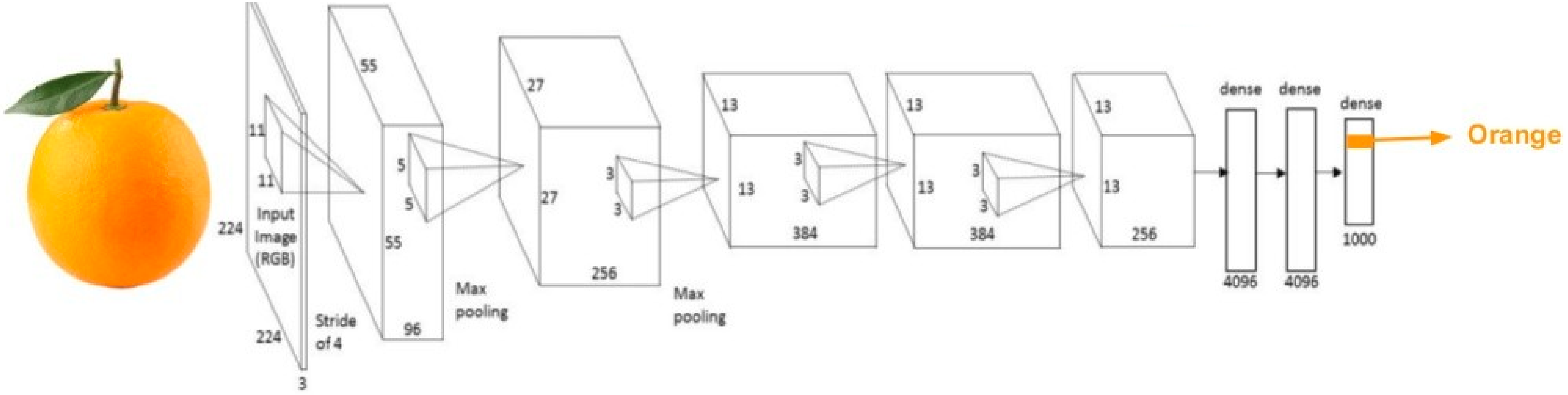


Dropout in CNNs



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

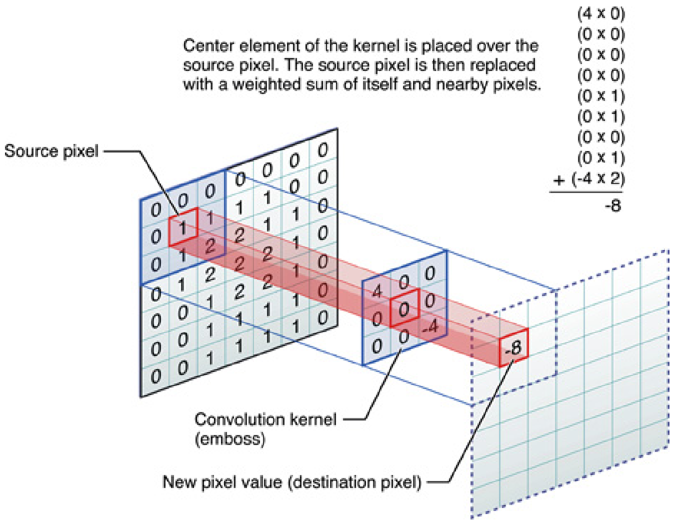
- **Filter has the depth of the input image** (e.g. depth 3 for RGB images)
  - two 3x3 filter applied to RGB image → 2 feature maps, i.e. 2 channels
    → number of adaptive parameters = 3 x 3 x 3 * 2 + 2 = 56

- **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

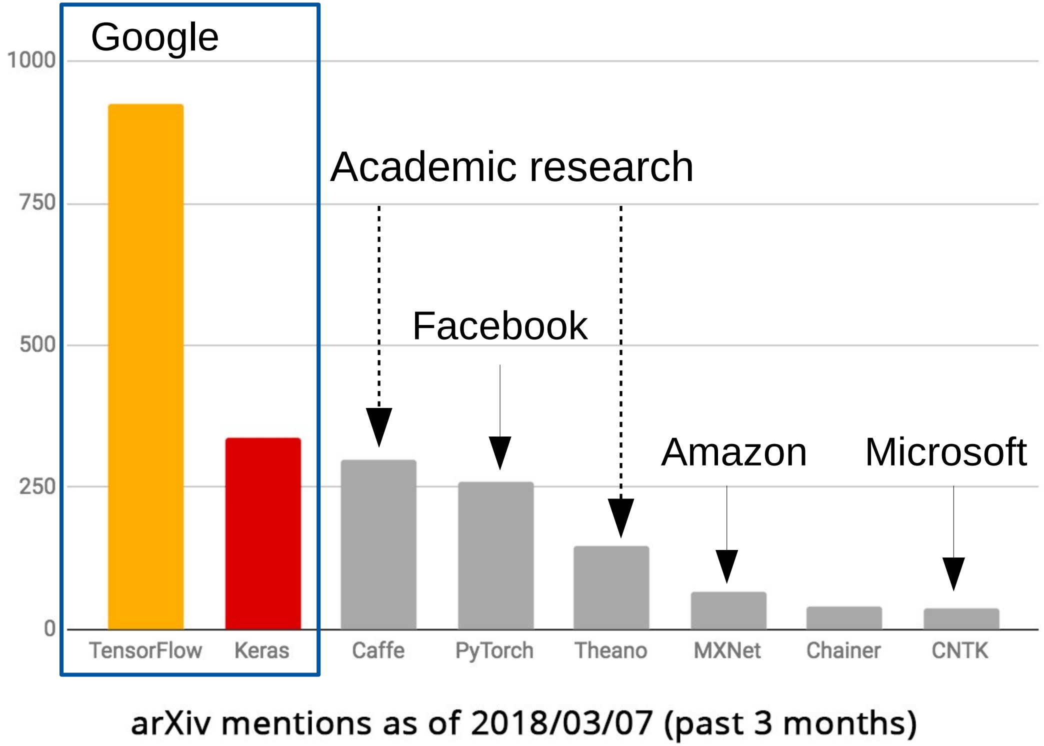
- 2D Convolution acts on 3D input (width x height x depth)
- Slide small filter over input and make linear transformation (dot product + bias)
- Hyperparameter:
  - Size of filter, typically (1 x 1), (3 x 3), (5 x 5) or (7 x 7)
  - Number of filters (feature maps)
  - **Padding** (maintain spatial extent)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


Google
1000
750
500
250
0
Academic research
Facebook
Amazon
Microsoft
TensorFlow
Keras
Caffe
PyTorch
Theano
MXNet
Chainer
CNTK
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))
```

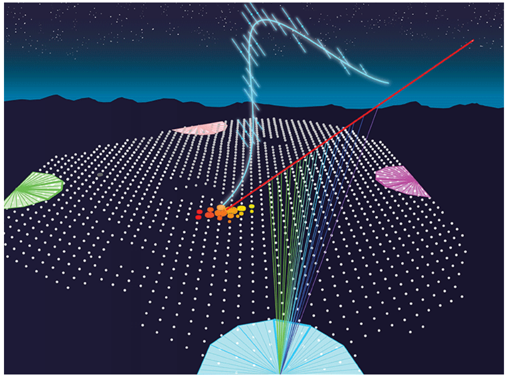
- Layer flattens output to vector → allows use of Dense layers after Convolutions

```
layers.Flatten()
```

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

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

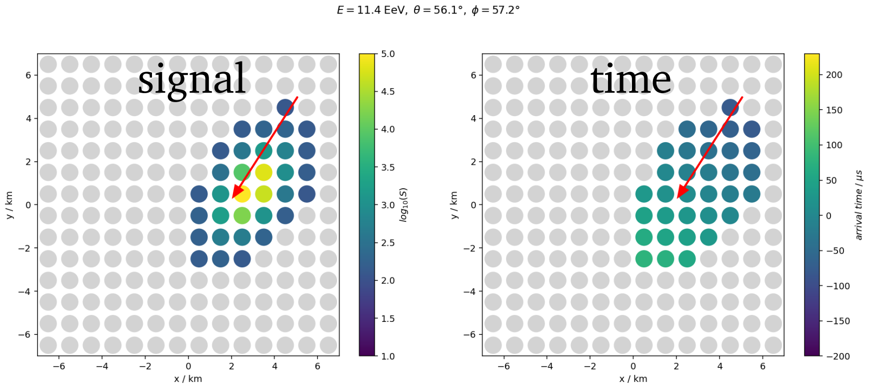
Air Shower Reconstruction - CNN



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:

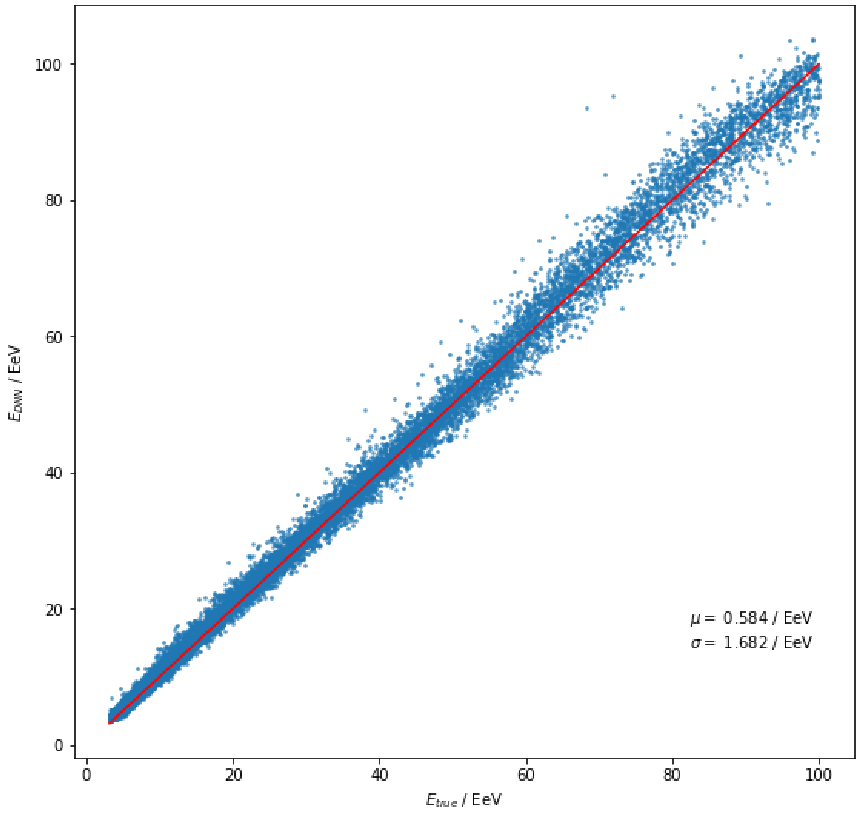
- Conv. layers and filters
- Pooling, Dense (FC) layers
- Regularization (after Flatten)

Model – modify:

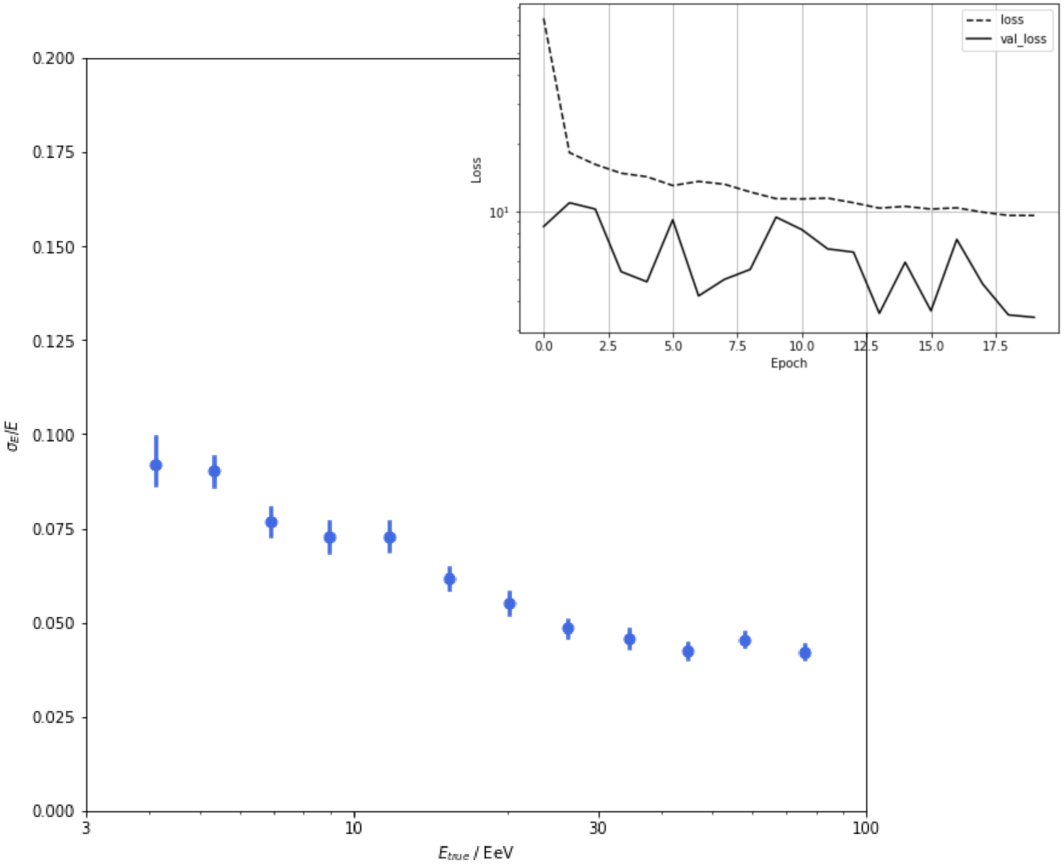
- Batch size, epochs
- Kernel size, strides
- Optimizer, learning rate

```
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



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

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

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

```
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')
    ])
```

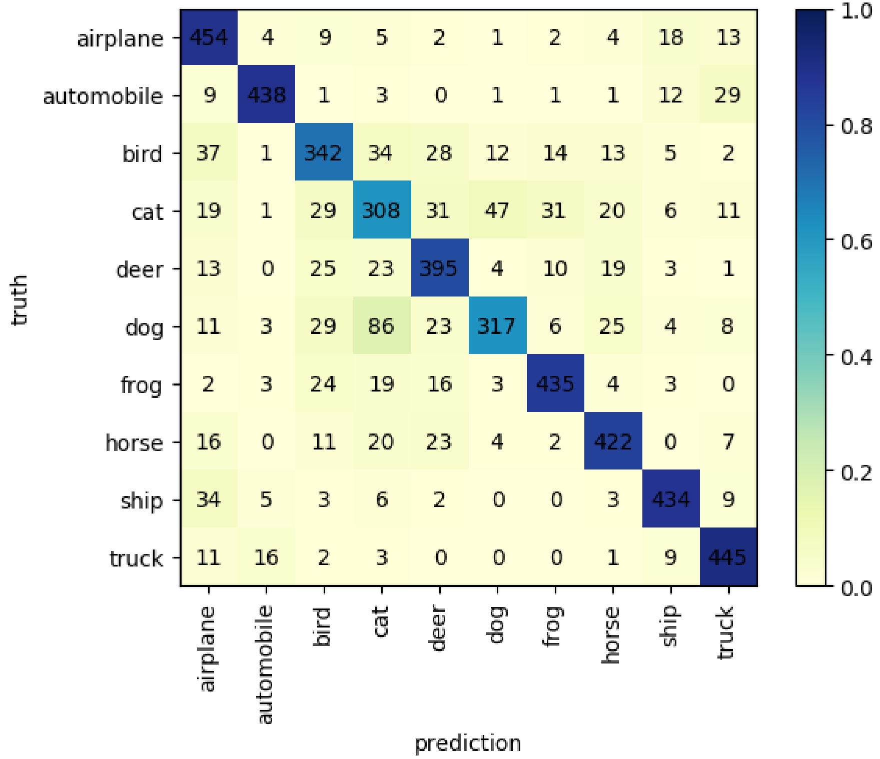
Open in Colab

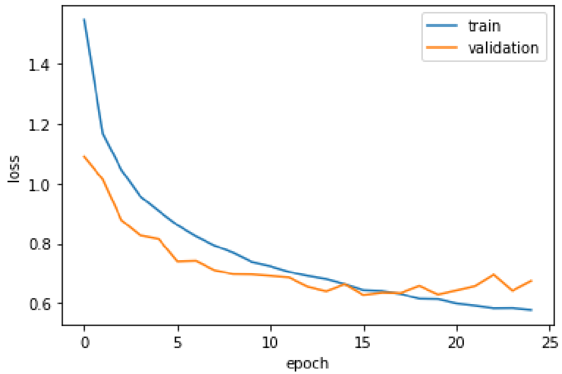
https://github.com/jglombitza/cifar_tutorial

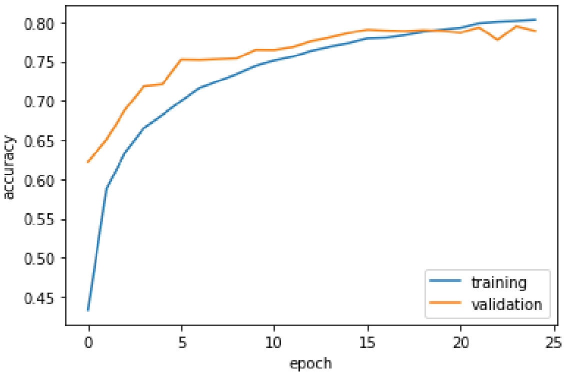
CIFAR 10 – Deep Convolutional Network

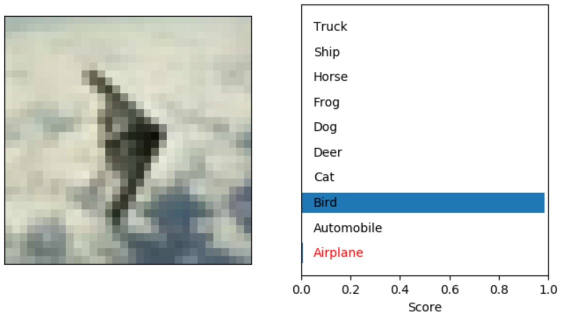
```
model = models.Sequential([
    layers.Convolution2D(16, kernel_size=(3, 3), padding='same', activation='elu',
                         input_shape=(32, 32, 3)),
    layers.Convolution2D(32, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(32, kernel_size=(3, 3), padding='same', 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')])
```

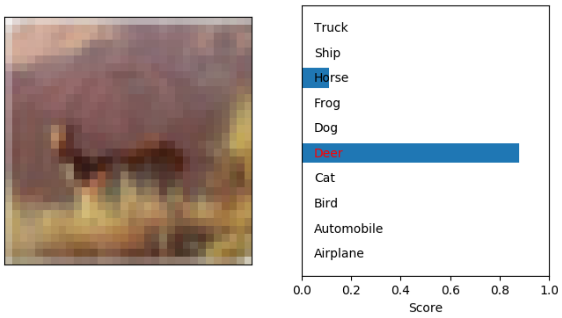
Results


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


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