OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

INFORMATIK UND
MATHEMATIK

Lecture 5
Neural Networks II

Dr.-Ing. Maike Stern | 11.11.2021

Recap
Feedforward neural networks

Number of layers: 3

Number of neurons 1. layer: 6

Number of neurons 2. & output layer: 4

Activation function hidden layers: ELU

Activation function output: Softmax

Initialisation: He initialisation, biases with 0.1

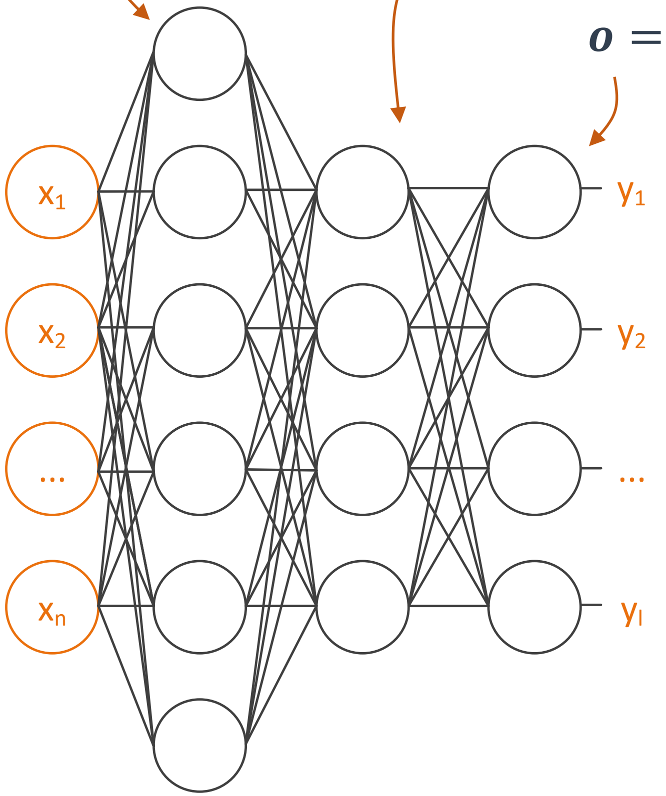
Optimiser: Momentum

Learning rate schedule: Exponential decay

$$\boldsymbol{h}^{(1)} = \mathrm{ELU}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$$

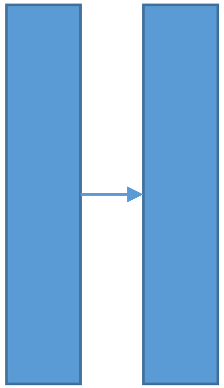
$$\boldsymbol{h}^{(2)} = \mathrm{ELU}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\boldsymbol{o} = \mathrm{softmax}\left(\boldsymbol{W}^{(3)\top}\boldsymbol{h}^{(2)} + \boldsymbol{b}^{(3)}\right)$$



Batch Normalisation

The problem:



- Internal covariate shift: The distribution of each layer's input changes during training, because the previous layer's parameters change
  → lower learning rate & careful initialisation necessary
- Gradient updates work under the assumption that the other weights do not change (while in practice, we update all layers simultaneously)

Sergey Ioffe, Christian Szegedy: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015.

Batch normalisation (Batchnorm) during training

- Batchnorm is added just before or after the activation function of every hidden layer
  $\boldsymbol{h} = \mathrm{BN}\big(\phi(\boldsymbol{W}^{\top}\boldsymbol{x} + \boldsymbol{b})\big)$ or $\boldsymbol{h} = \phi\big(\mathrm{BN}(\boldsymbol{W}^{\top}\boldsymbol{x} + \boldsymbol{b})\big)$
- Batchnorm zero-centers and normalises each input and then scales and shifts the results *with learned parameters* → the model learns the optimal scale and mean of each of the layer's inputs

$$\widehat{\boldsymbol{x}}^{(i)} = \frac{\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \varepsilon}}$$

$\boldsymbol{\mu}_B = \frac{1}{m_B}\sum_{i=1}^{m_B}\boldsymbol{x}^{(i)}$, where m is the number of samples in the mini-batch, and $\boldsymbol{\mu}_B$ is the vector of input means

$\boldsymbol{\sigma}_B = \frac{1}{m_B}\sum_{i=1}^{m_B}(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_B)^2$, where is the vector of input standard deviations

$\widehat{\boldsymbol{x}}_i$ is the vector of zero-centered and normalised inputs for sample i

$$\boldsymbol{z}^{(i)} = \boldsymbol{\gamma} \otimes \widehat{\boldsymbol{x}}^{(i)} + \boldsymbol{\beta}$$

γ is the learned output scale parameter vector, β is the learned output shift parameter vector, and z_i is the rescaled and shifted version of the input features

$\otimes$ represents element-wise multiplication

Sergey Ioffe, Christian Szegedy: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015.

Batch normalisation (Batchnorm) during test time

- Compute the moving average of the layer's input means **μ** and standard deviations **σ** during training
- At test time, use **μ** and **σ** to compute batchnorm
- Usually, deep learning libraries do this automatically (but you should check the default settings to avoid frustration) ☺

Batch normalisation (Batchnorm) summary

+ Controls the internal covariate shift and reduces the vanishing gradient problem
+ Smoothes the optimisation landscape → more predictive and stable behaviour of the gradients
- Adds complexity to the model

Design rules:
When using batchnorm, the β parameter replaces the bias.
The ELU activation function may render batchnorm unnecessary.
Using a batchnorm layer as first layer of the network standardises the training set for you.

Santurkar et al.: How does Batch Normalisation help optimization?, 2018.
https://en.wikipedia.org/wiki/Batch_normalization

Recap
Feedforward neural networks

Number of layers: 3

Number of neurons 1. layer: 6

Number of neurons 2. & output layer: 4

Activation function hidden layers: ELU

Activation function output: Softmax

Initialisation: He initialisation, biases with 0.1

Optimiser: Momentum

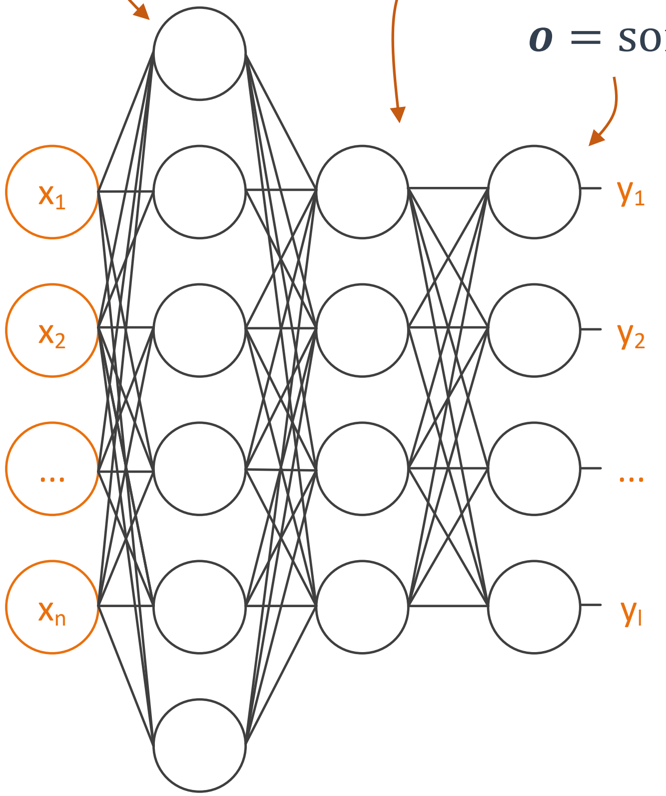
Learning rate schedule: Exponential decay

Batchnorm

$$\boldsymbol{h}^{(1)} = \mathrm{ELU}\left(\mathrm{BN}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)\right)$$

$$\boldsymbol{h}^{(2)} = \mathrm{ELU}\left(\mathrm{BN}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)\right)$$

$$\boldsymbol{o} = \mathrm{softmax}\left(\mathrm{BN}\left(\boldsymbol{W}^{(3)\top}\boldsymbol{h}^{(2)} + \boldsymbol{b}^{(3)}\right)\right)$$

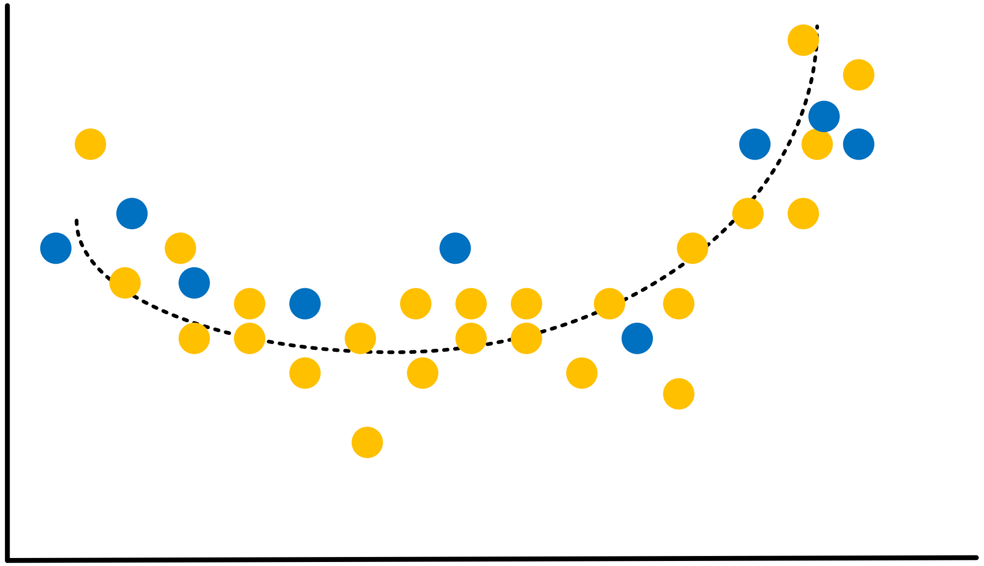


Machine Learning Research

Network Training

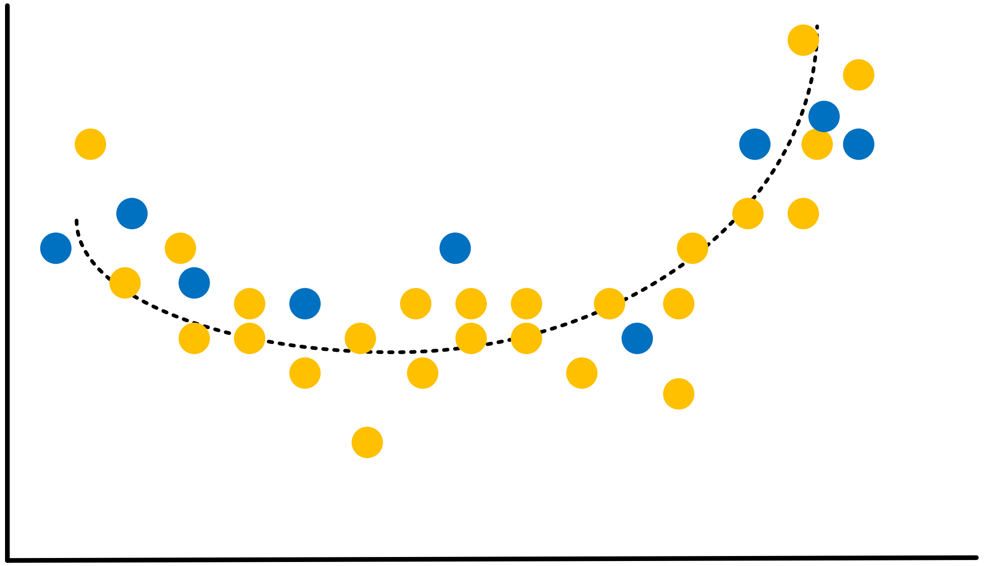
The data perspective

- To find the optimal network parameterisation, we train the network using a training dataset
- This training dataset $\mathcal{D}_{\sim P^*}$ (empirical distribution) is sampled from the data-generating distribution P^* (true distribution) and is, necessarily, a rough estimate of P^*
- As the number of training examples increases, the empirical distribution of $\mathcal{D}$ approaches the true distribution P^*



The data perspective

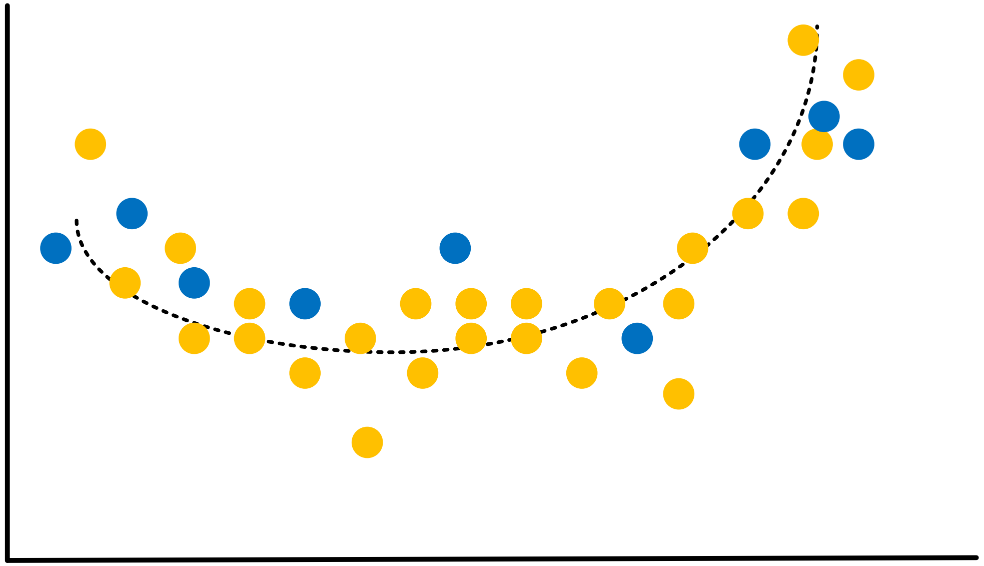
- To find the optimal network parameterisation we train the network using a training dataset
- This training dataset $\mathcal{D}_{\sim P^*}$ (empirical distribution) is sampled from the data-generating distribution P^* (true distribution) and is, necessarily, a rough estimate of P^*
- As the number of training examples increases, the empirical distribution of $\mathcal{D}$ approaches the true distribution P^*



Design rules:
The more data the better.
Training and validation sets should be a good representation of the true distribution.

The data perspective
Bias-variance trade-off

- In other words, we have an unobserved true function f that we want to approximate using a machine learning method and the training dataset $\mathcal{D}_{\sim P^*}$
- The measure of fit is determined by computing the mean squared error (MSE) between the true function f and the estimated function $\hat{f}$ based on a hold-out test dataset



The data perspective
Bias-variance trade-off

- We have an unobserved true function f that we want to approximate using a machine learning method and a number of samples (the training dataset)
- The measure of fit is determined by computing the mean squared error (MSE) between the true function f and the estimated function $\hat{f}$ based on a hold-out test dataset $\{x_{n+1}, y_{n+1}\}$
- Generally, the test error has the following components:

error terms

$$\mathbb{E}\left(y_0 - \hat{f}(x_{n+1})\right)^2 = \underbrace{\mathrm{Var}\left(\hat{f}(x_{n+1})\right) + \left[\mathrm{Bias}\left(\hat{f}(x_{n+1})\right)\right]^2}_{\text{(potentially) reducible error}} + \mathrm{Var}(\epsilon)$$

Expected test MSE: estimate the true function f using a large number of training sets and test each $\hat{f}$ at x_{n+1}

Variance

Bias

(potentially) reducible error

Irreducible error, introduced because $\hat{f}$ is not a perfect estimate for f, due to measurement errors, etc.

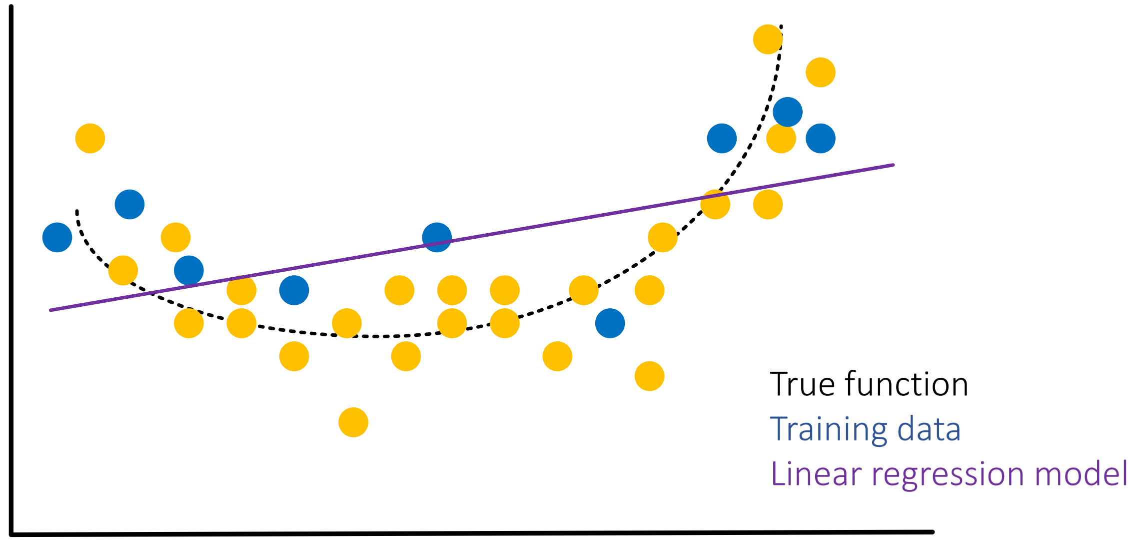
Machine Learning Research

Bias

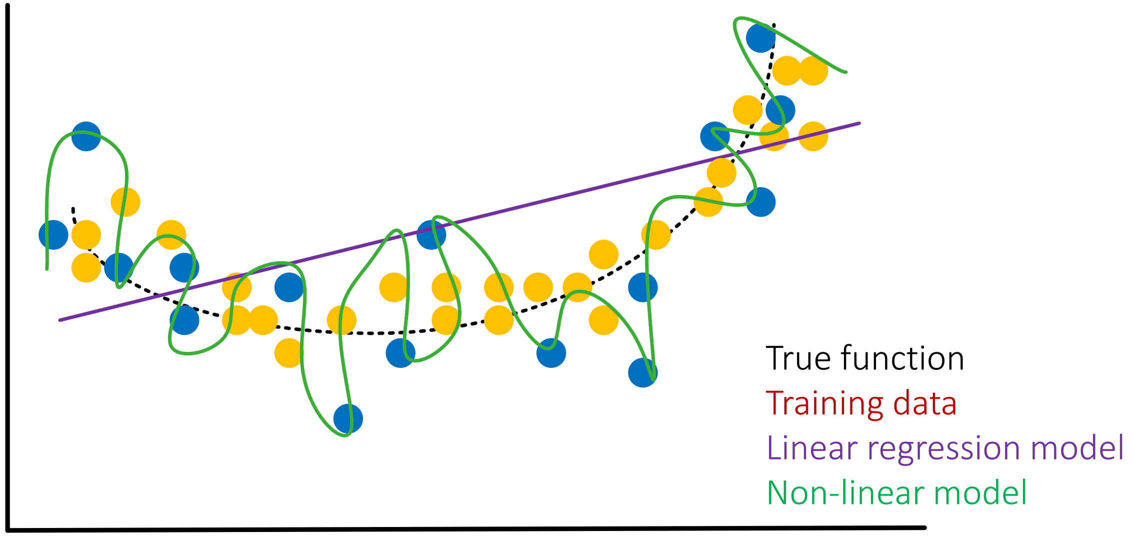
Note that the term bias is also used elsewhere:

- $wx + b$
- Algorithmic bias: If an algorithm produces results that are systemically prejudiced (https://queue.acm.org/detail.cfm?id=3466134)

Bias is the error introduced by approximating a problem with a model that is too simple, e.g. approximating a non-linear problem with linear regression

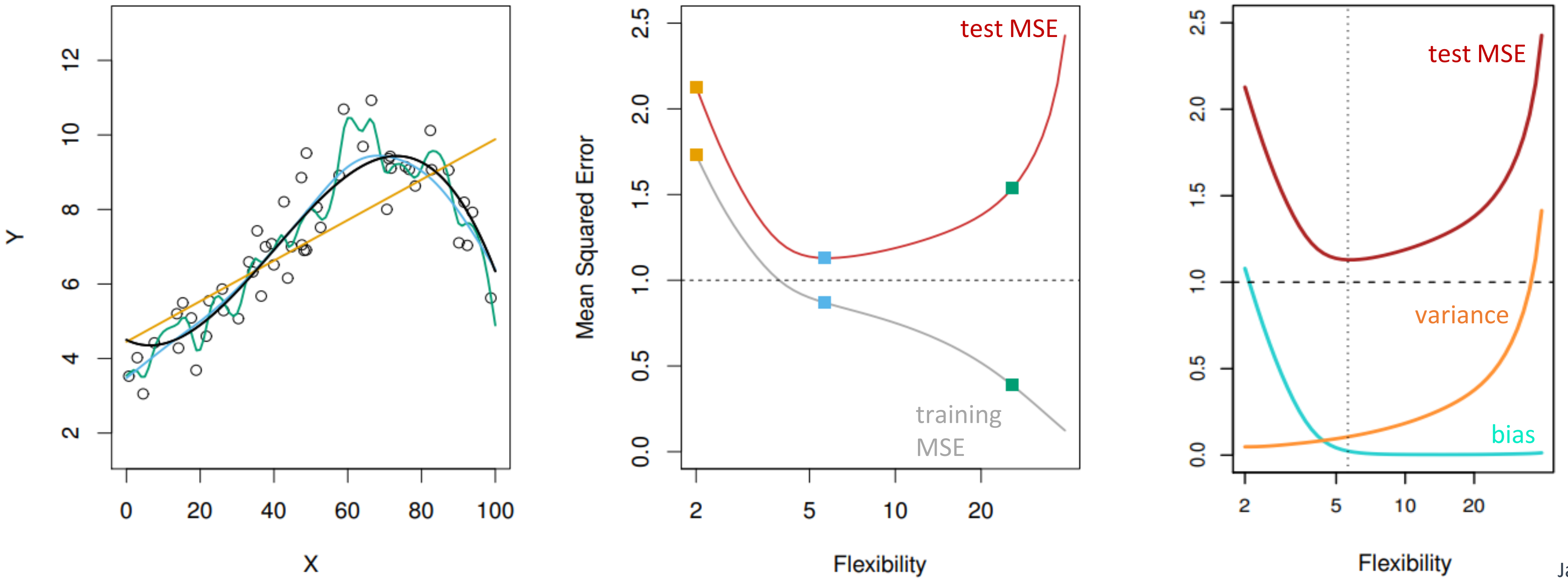


Variance is a measure of how much the learned function would change if we estimated it using a different training dataset



Bias-variance trade-off

- Bias: The less flexible a model that models a substantially non-linear problem, the higher both training error and test error
- Variance: The more flexible a model (the more degrees of freedom) the lower the training error but the higher the test error → also called overfitting

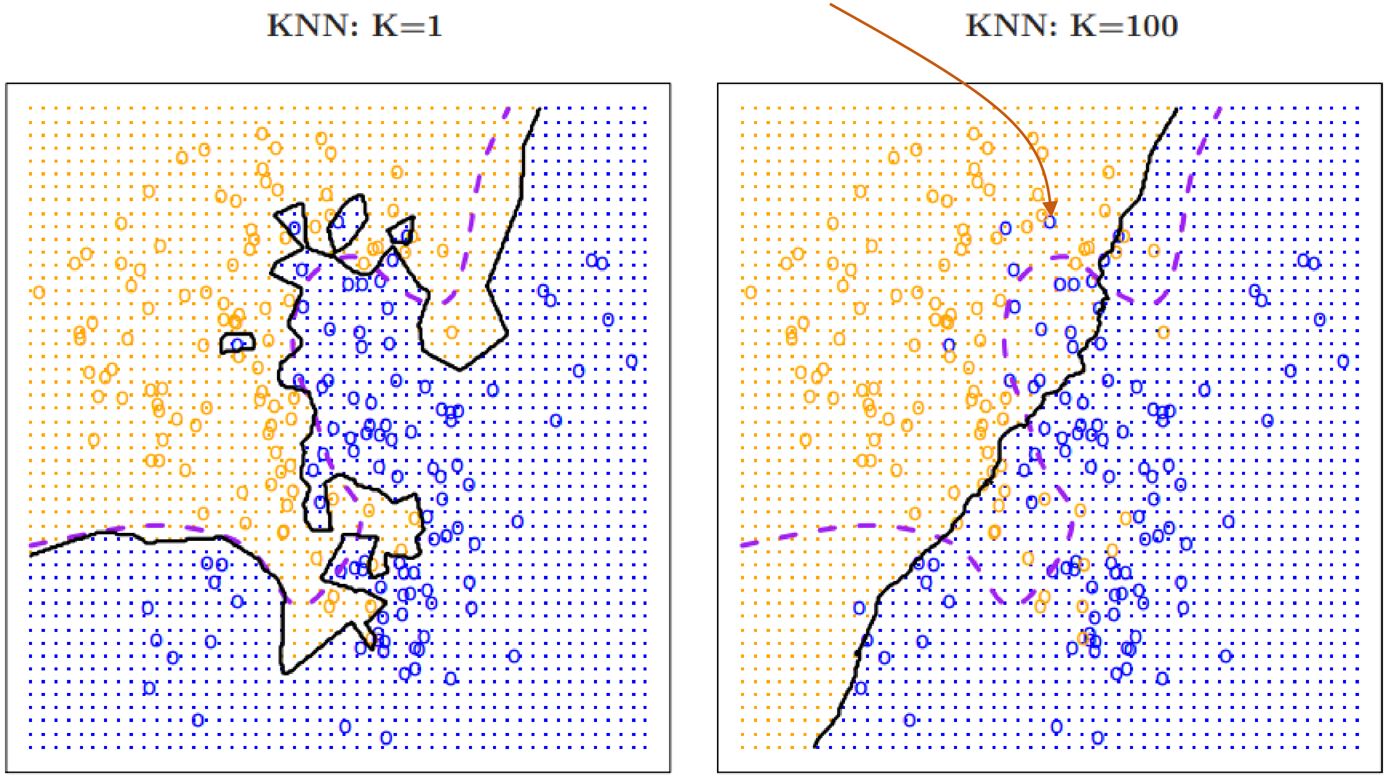


James et al.: An Introduction to Statistical Learning.

Bias and variance in classification

Bayes error /
irreducible error

high variance



high bias

Bayes decision boundary

James et al.: An Introduction to Statistical Learning.

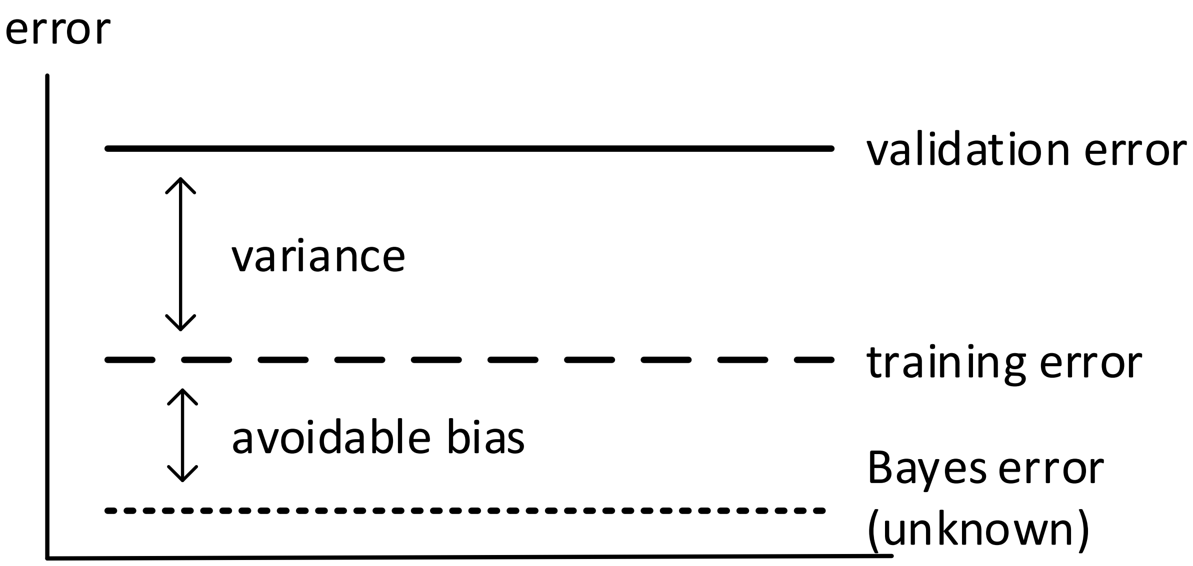
Bias-variance trade-off in neural networks

- Neural networks are universal function approximators that are extremely flexible
- Thus, neural networks typically do not suffer from errors dominated by bias but variance

James et al.: An Introduction to Statistical Learning.

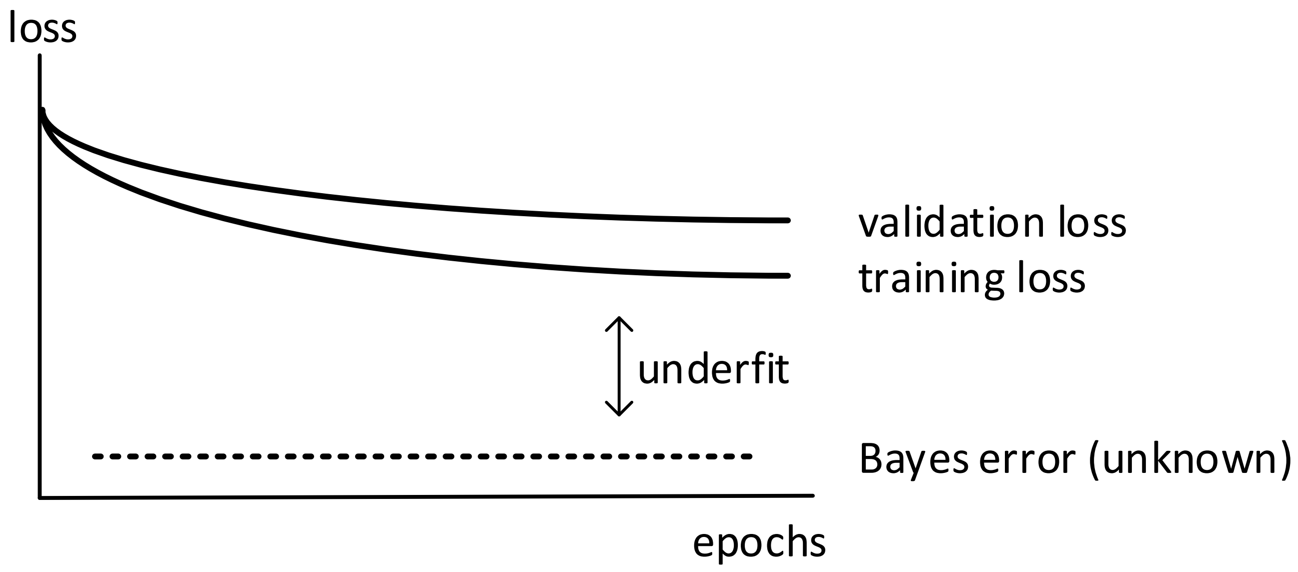
How to spot bias and variance

- Bias: The training error is higher than the Bayes error (irreducible error). Since the Bayes error is generally unknown, bias may only be indicated by a high training error
- Variance: The validation error is (much) higher than the training error, which indicates that the network memorises the training samples and does not generalise well with respect to the validation samples



Spotting & decreasing a high bias

- When observing the training progress by plotting the loss over epochs, a high training and validation loss indicate a high bias
- The increasing difference in validation loss and training loss also indicates increasing variance: high bias and high variance may occur in high-dimensional data

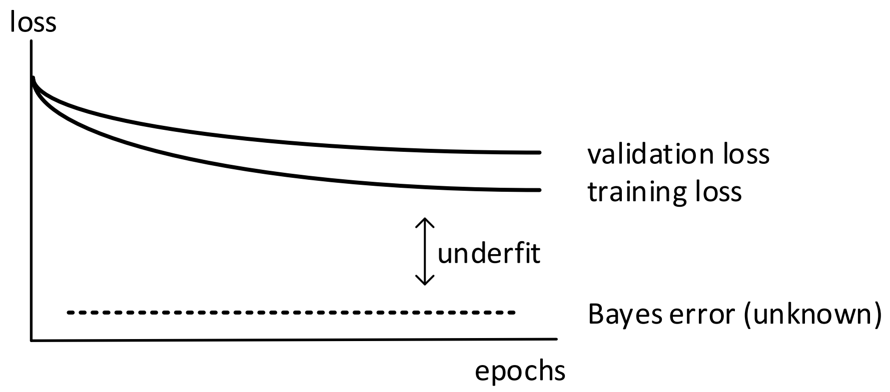


Spotting & decreasing a high bias

- When observing the training progress by plotting the loss over epochs, a high training and validation loss indicate a high bias
- The increasing difference in validation loss and training loss also indicates increasing variance: high bias and high variance may occur in high-dimensional data

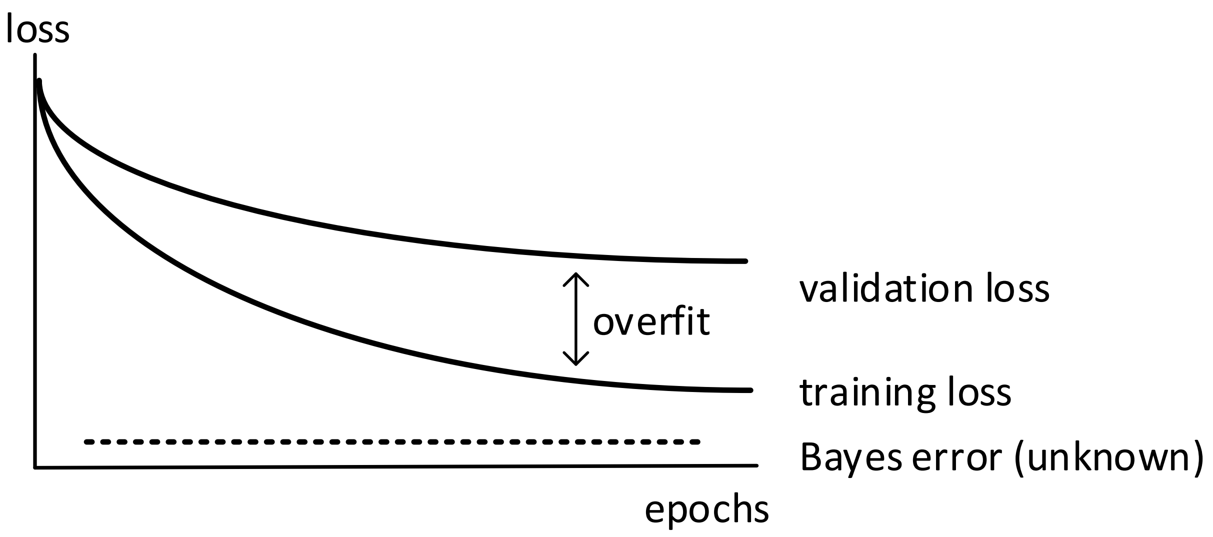
Approaches to diminish a high bias:

- Use a larger network architecture (more layers, more neurons)
- Train longer
- Try other optimisation methods
- Use a different architecture type
  (e.g. convolutional nets instead of forward neural networks)



Spotting & decreasing high variance

- When observing the training progress by plotting the loss over epochs, a low training loss but high validation loss indicates a high variance
- Also, test set performance will be bad

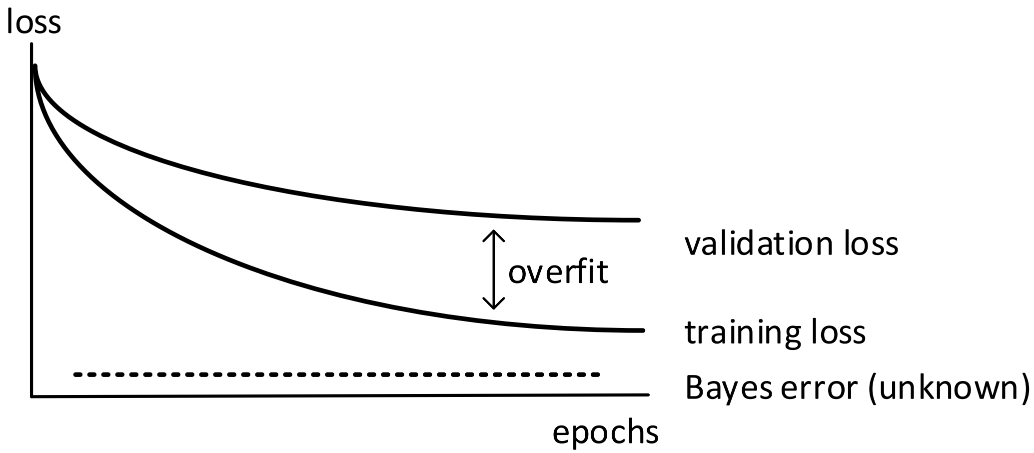


Spotting & decreasing high variance

- When observing the training progress by plotting the loss over epochs, a low training loss but high validation loss indicates a high variance
- Also, test set performance will be bad

Approaches to diminish high variance:

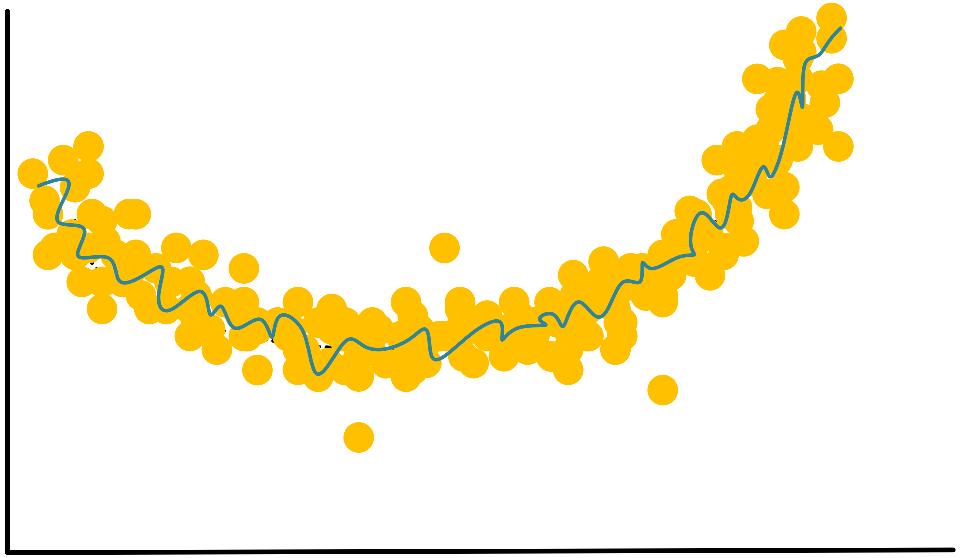
- Use more training data (if possible)
- Early stopping
- Optimise the network architecture
- Add regularisation



Decreasing high variance
More data

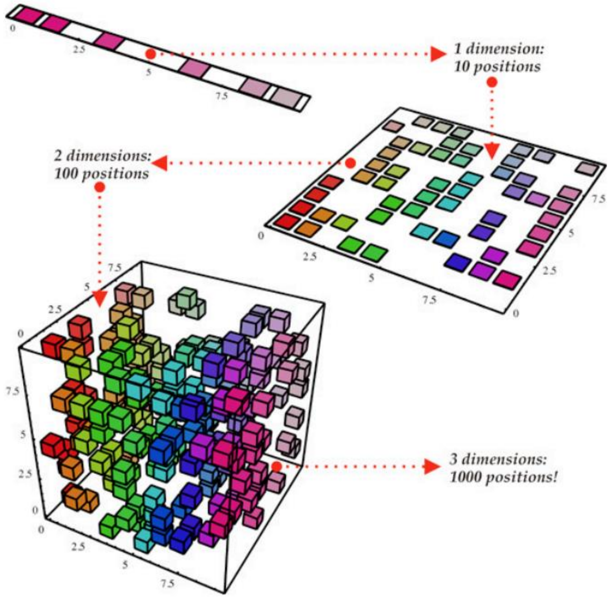
Decreasing variance

- More data: A model with 100 degrees of freedom perfectly fits 100 training examples, thus adding more data fosters a more general solution



Decreasing variance

- More data
- Generally, the more input features, the more data is required (curse of dimensionality)



Decreasing high variance
Early stopping

Decreasing variance

- More data
- Early stopping: Monitor the validation loss and stop network training if the validation loss didn‘t drop after a given number of epochs

number of epochs to wait

```
tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', min_delta=0, patience=0, verbose=0,
    mode='auto', baseline=None, restore_best_weights=False
)
```

minimum change in the monitored quantity to qualify as an improvement

```
val_loss: 2.5561
val_loss: 4.1238
val_loss: 2.7290
val_loss: 2.9921
val_loss: 2.5698
val_loss: 2.7733
```

val_loss isn‘t lower than five epochs before, so network training was stopped

Decreasing high variance
Early stopping

Decreasing variance

- More data
- Early stopping: Monitor the validation loss and stop network training if the validation loss didn‘t drop after a given number of epochs
- Optimise the network architecture

The bias-variance trade-off from a hypothesis space perspective

→ A model's flexibility is given by the expressiveness (or lack thereof) of its hypothesis space of candidate models

- Bias: A model with a very limited hypothesis space might not be able to represent the true distribution of the data P^* (even with a large training dataset $\mathcal{D}_{\sim P^*}$ whose distribution is close to P^*)
- Variance: A model with a highly expressive hypothesis space is more likely able to correctly represent P^*, however, we might not be able to select the „right" model given our limited training dataset $\mathcal{D}_{\sim P^*}$ and thus overfit to $\mathcal{D}$

Neural networks have a highly expressive hypothesis space

Neural networks have a highly expressive hypothesis space

Hard constraint

Restrict the hypothesis space of possible models, e.g. by choosing a simpler method, a smaller network architecture, etc.

→ Imposes a hard constraint that prevents a model that precisely captures the training data

Decreasing variance
Imposing constraints

Neural networks have a highly expressive hypothesis space

Hard constraint

Restrict the hypothesis space of possible models, e.g. by choosing a simpler method, a smaller network architecture, etc.

→ Imposes a hard constraint that prevents a model that precisely captures the training data

Soft constraint

Change the training objective so as to incorporate a soft preference for simpler models

→ Combines components that seek a model that fits well with the training data with a regularising component

Regularisation

Regularisation

- Regularisation: Any modification we make to a learning algorithm that is intended to reduce its generalisation error but not its training error → reduce variance
- One way: Adding a regularisation term to the loss function

$$\hat{f} = \underbrace{\hat{f}}_{} \; \arg\min_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} L(f(x_i), y_i) + R(f)$$

minimal loss function

hypothesis space

loss, e.g. cross-entropy or mean squared error

regularisation term

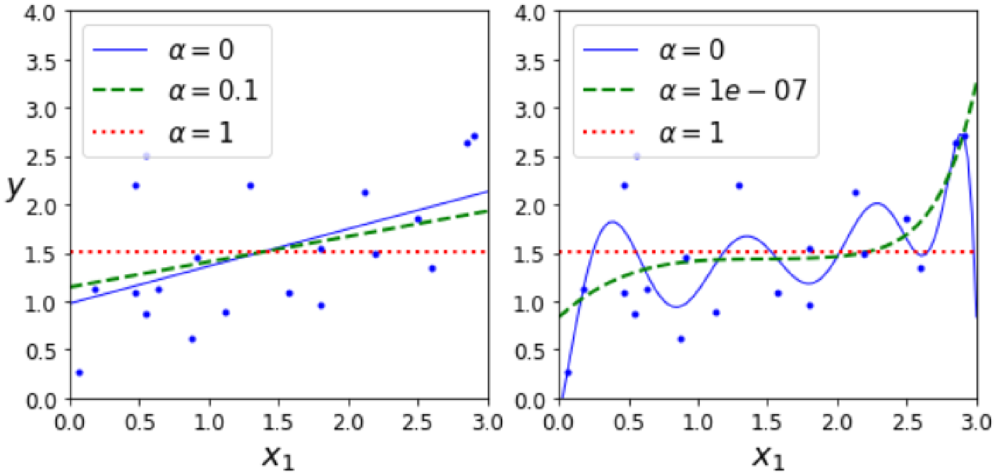
- Typically, regularisation incorporates the weights but not the biases

L_1 regularisation

- Also known as Lasso regression (least absolute shrinkage and selection operator regression)
- Adds the $\ell 1$ norm to the training loss function, with $R(f) = \alpha\|\boldsymbol{w}\|_1 = \sum_{i=1}^{n} |w_i|$
- Linear penalty: Parameters shrink independently of their current value
- Given a large enough regularisation strength α, L_1 regularisation performs variable selection by forcing some of the parameters to equal zero

α is the regularisation strength (typically $1*10^{-7}$ to 0.01)

→ Results in sparse models



A. Geron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow

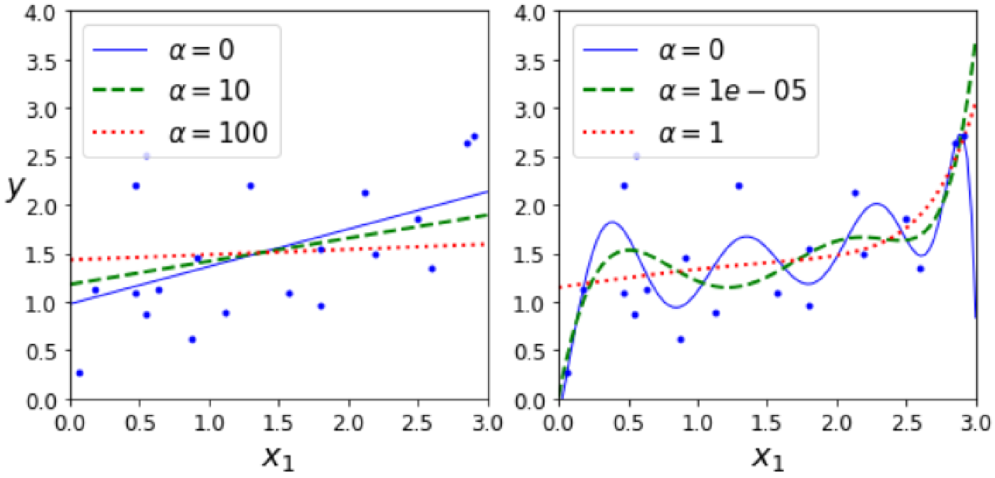
Machine Learning Research

L_2 regularisation

regularisation strength (typically $1*10^{-6}$ to 0.1)

- Also known as Ridge regression or Tikhonov regularisation
- Adds the Euclidean norm ($\ell 2 = \|\boldsymbol{w}\|_2 = \sum_{i=1}^{n} \sqrt{w_i^2}$) to the training loss function, with $R(f) = \alpha \frac{1}{2} \|\boldsymbol{w}\|_2^2$
- Here, the regularisation adds a quadratic penalty on the magnitude of the weights or, in other words, it prefers weights with small values, and hereby implicitly restricts the hypothesis space of candidate models
- The penalty grows quadratically with the parameter magnitude: an increase in w_i from 0 to 0.1 is penalised less than an increase from 3 to 3.1

→ results in a dense, smoothed distribution of the parameter values, where all parameters contribute



A. Geron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow

Machine Learning Research

Dropout

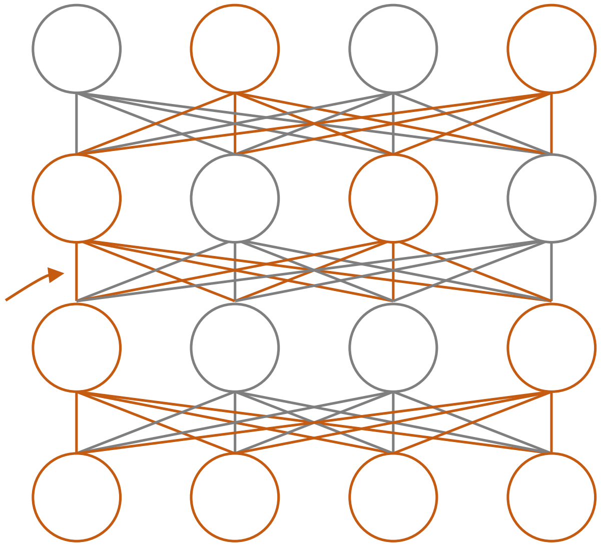
- At every *training* step, each neuron (including inputs, excluding outputs) has a probability p of being temporarily dropped out

Dropout rate, typically between 10% and 50%

keep probability

- For the remaining neurons, each output is divided by $(1 - p)$, to keep the magnitude of the inputs stable
- During test time, all neurons are active

\+ Increases the network‘s robustness and generalisation ability



Design rules:
L2 regularisation + dropout is a good starting point
The optimal regularisation strength changes with the learning rate

Conclusion
Regularisation

- Regularisation helps prevent overfitting, a common problem of neural networks
- Regularisation methods:
  - Batch normalisation
  - Early stopping
  - L_1, and L_2
  - Dropout

Recap
Feedforward neural networks

Number of layers: 3

Number of neurons 1. layer: 6

Number of neurons 2. & output layer: 4

Activation function hidden layers: ELU

Activation function output: Softmax

Initialisation: He initialisation, biases with 0.1

Optimiser: Momentum

Learning rate schedule: Exponential decay

Batchnorm

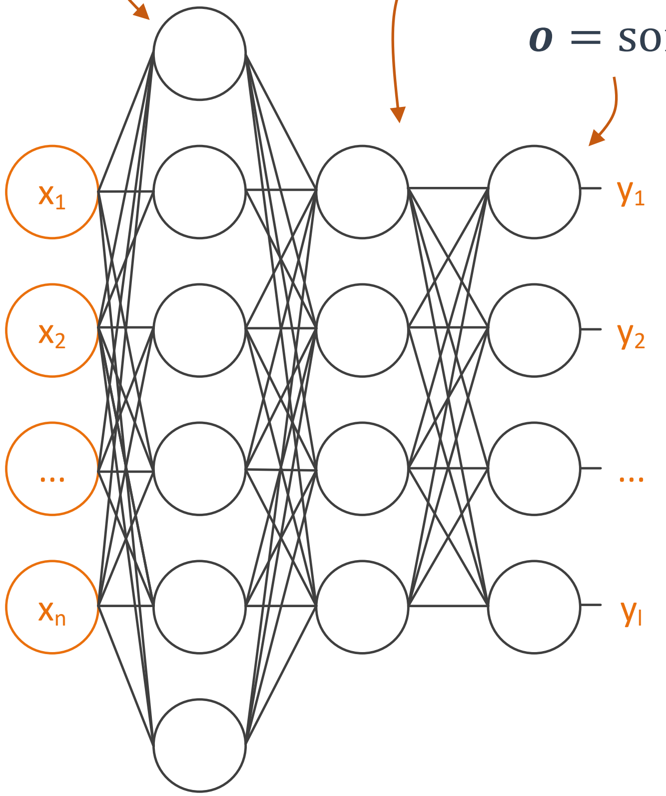
Early stopping

L_2 regularisation

$$\boldsymbol{h}^{(1)} = \mathrm{ELU}\left(\mathrm{BN}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)\right)$$

$$\boldsymbol{h}^{(2)} = \mathrm{ELU}\left(\mathrm{BN}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)\right)$$

$$\boldsymbol{o} = \mathrm{softmax}\left(\mathrm{BN}\left(\boldsymbol{W}^{(3)\top}\boldsymbol{h}^{(2)} + \boldsymbol{b}^{(3)}\right)\right)$$



$$\text{loss} = L(f(x), y) + \alpha\|w\|_2^2$$

Questions?

Neural Networks for Computer Vision

Neural networks for computer vision
Classification

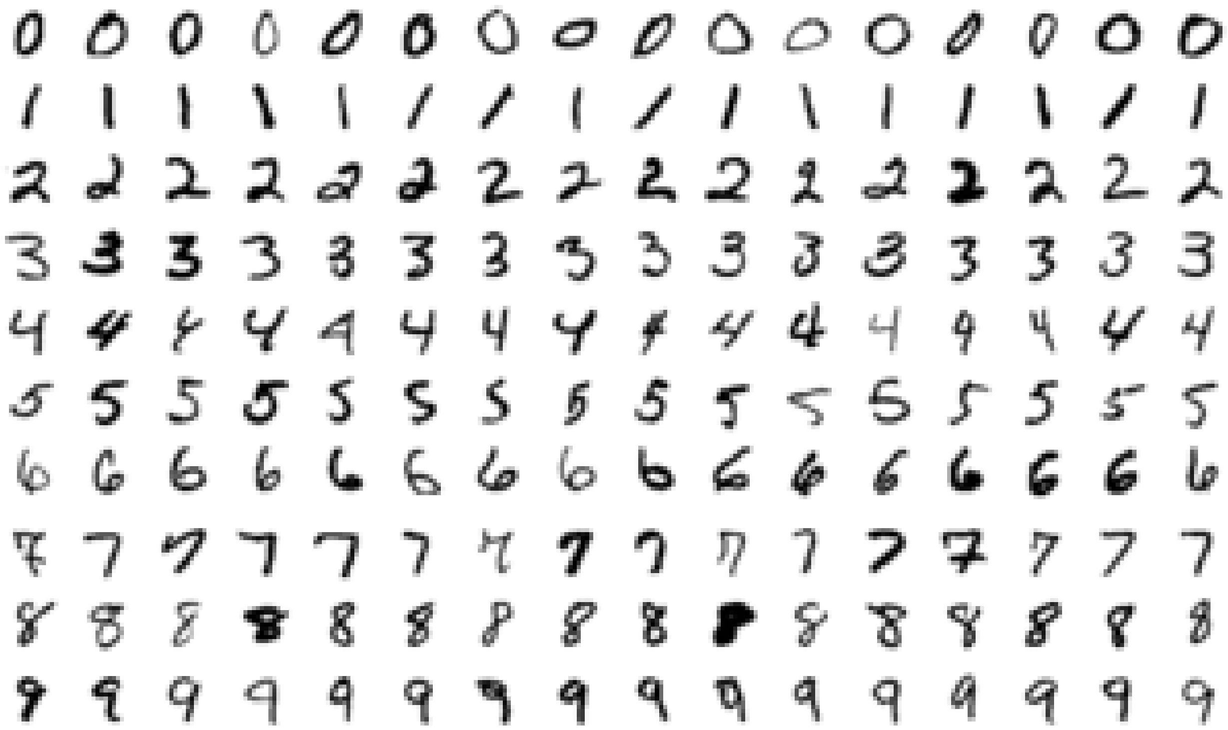
input

Neural network

Classification output:

- Coffee cup
- Cookie
- Cat paw
- Spoon
- Bench

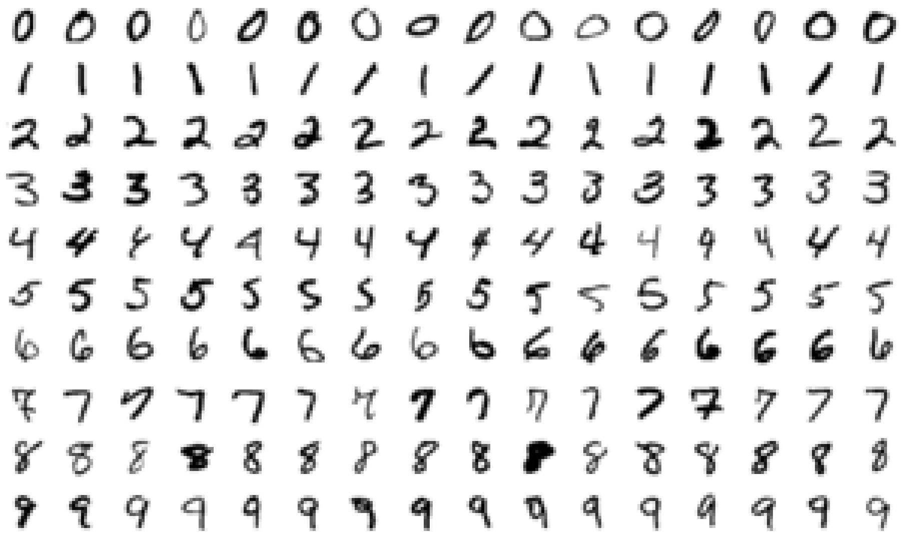
Why don‘t we use feedforward neural networks for computer vision?



Neural networks for computer vision

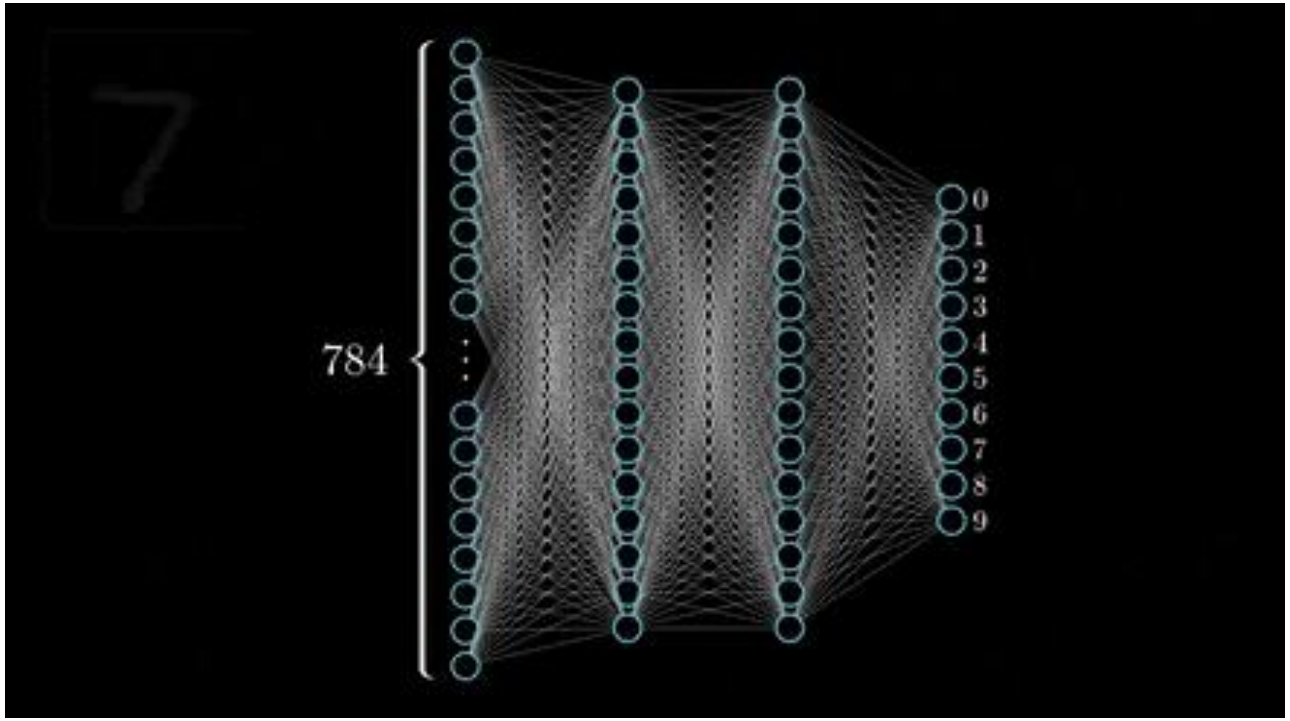
Why don‘t we use feedforward neural networks for computer vision?

- Fully-connected networks do not scale well to images (e.g. 28*28 = 784 weights per neuron)
- Transformations in the image produce significant changes in the raw data, e.g.
  - changes in the position
  - changes in the object size



The MNIST dataset is already size-normalised

Why don‘t we use feedforward neural networks for computer vision?



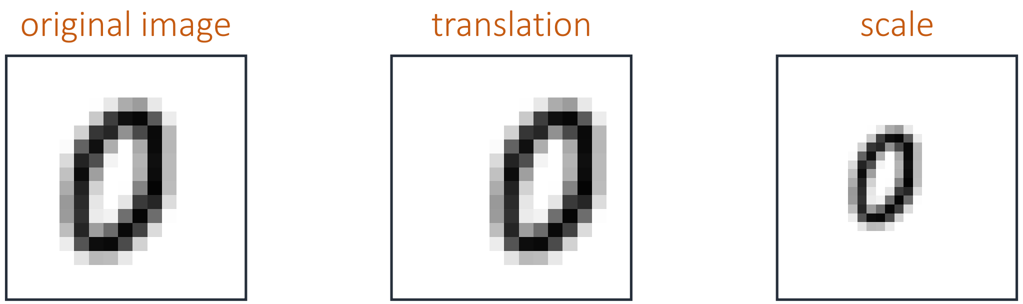
Neural networks for computer vision

Why don‘t we use feedforward neural networks for computer vision?

- Transformations in the image produce significant changes in the raw data, e.g.
  - changes in the position
  - changes in the object size

→ These changes should still yield the same predictions, hence we need:
  - Translation invariance: An image object should be assigned the same classification irrespective of its position within the image
  - Scale invariance: The same is true for objects of different size

Invariance: when the effect of a transformation in the input is not detectable in the output



These changes should still yield the same predictions, hence we need:

- Translation invariance: An image object (e.g. a handwritten digit) should be assigned the same classification (e.g. “9”) irrespective of its position within the image
- Scale invariance: The same is true for objects of different size

Solutions:

- Image preprocessing: extract features that are invariant under the required transformations
- Augment the training set with respect to the desired invariances
- Build the invariance properties into the network structure

equip the network with prior knowledge about the task

How to equip neural networks with a prior knowledge about computer vision?

Neural networks for computer vision

How to equip neural networks with a prior knowledge about computer vision?

- Images have a strong 2D local structure
- Structures of neighbouring pixels can be classified into a small number of categories, e.g. edges

local features

Spatially close pixels are highly correlated

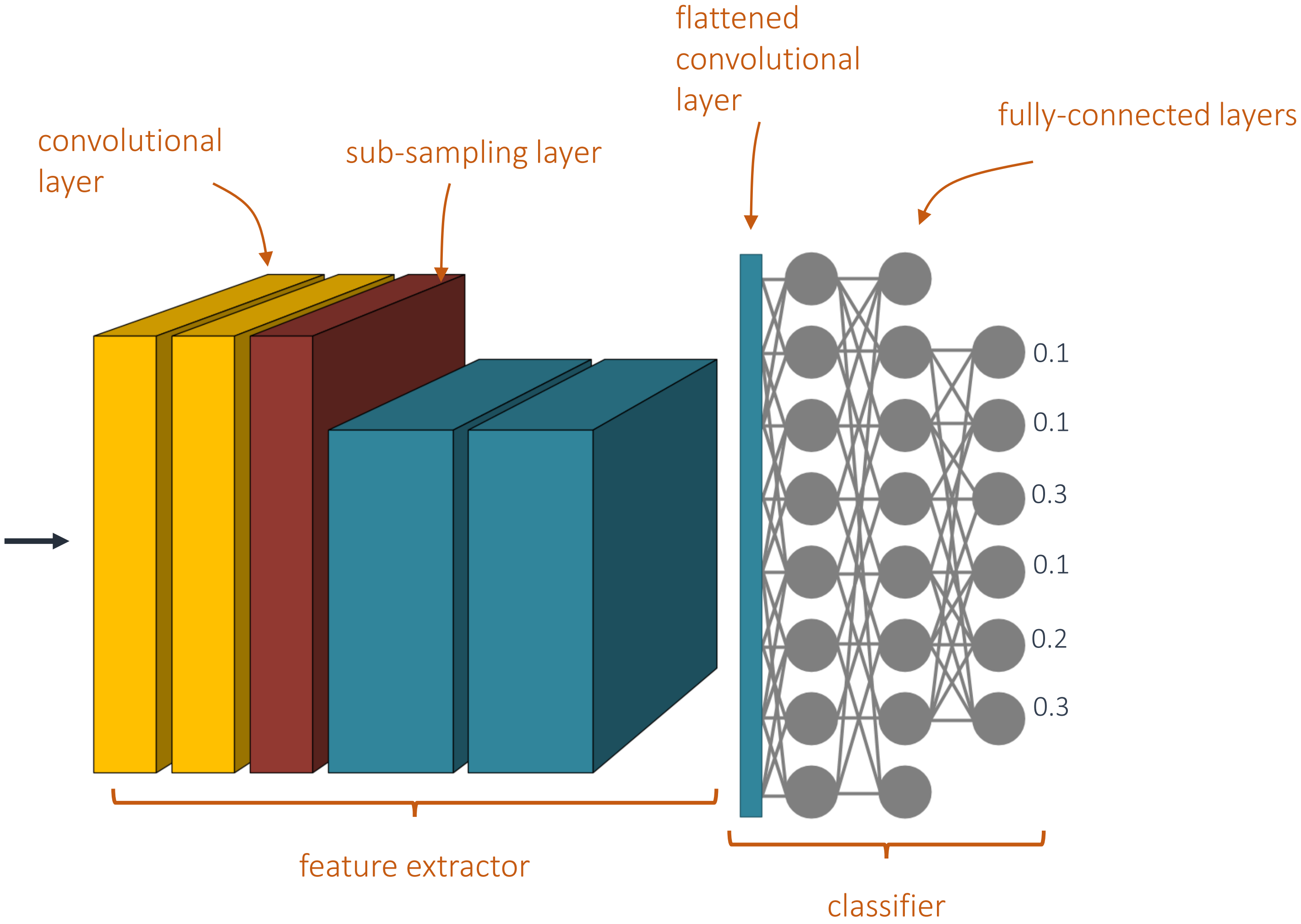
Why don‘t we use feedforward neural networks for computer vision?

- Images have a strong 2D local structure
- Structures of neighbouring pixels can be classified into a small number of categories, e.g. edges

Solution:

- Convolutional layers, which use local receptive fields and shared weights
- Sub-sampling layers, which reduce the sensitivity to shifts and distortions

Convolutional neural networks



What is convolution?

Extract image features by applying filters

(and further processing the extracted information)

What is convolution?

kernel / filter

-1	0	+1
-2	0	+2
-1	0	+1

image

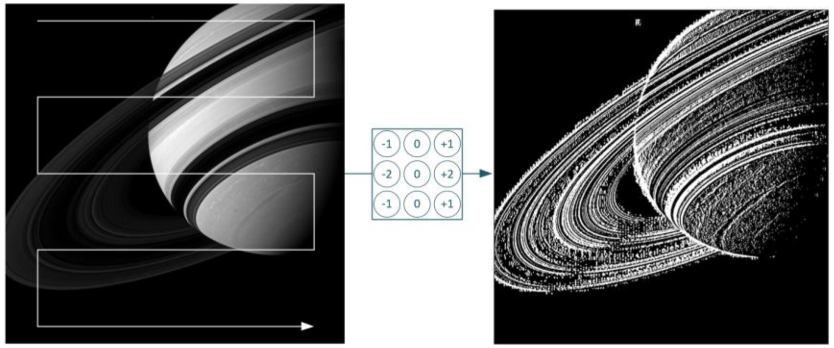
0	0	0	0	0
0	1	2	3	0
0	1	2	3	0
0	1	2	3	0
0	0	0	0	0

padding: to keep the image dimensions in the output

-1*0 + 0*0 + 1*0 + … -1*0 + 0*1 +1*2

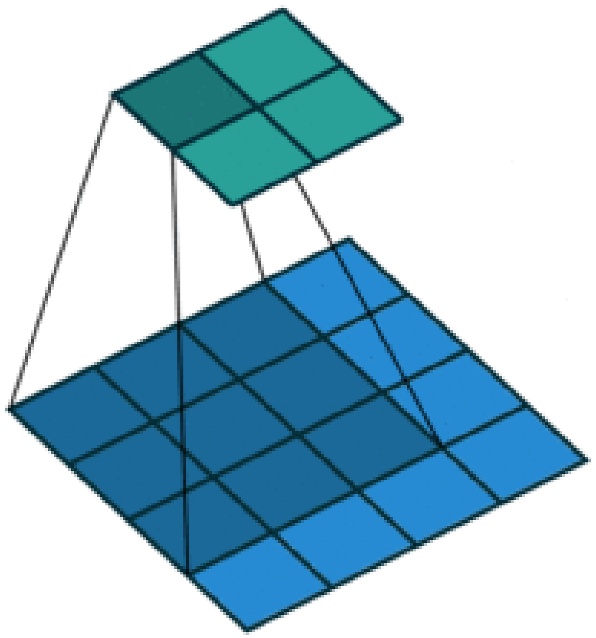
convoluted image / feature map

6	6	-6
8	6	-8
8	6	-6

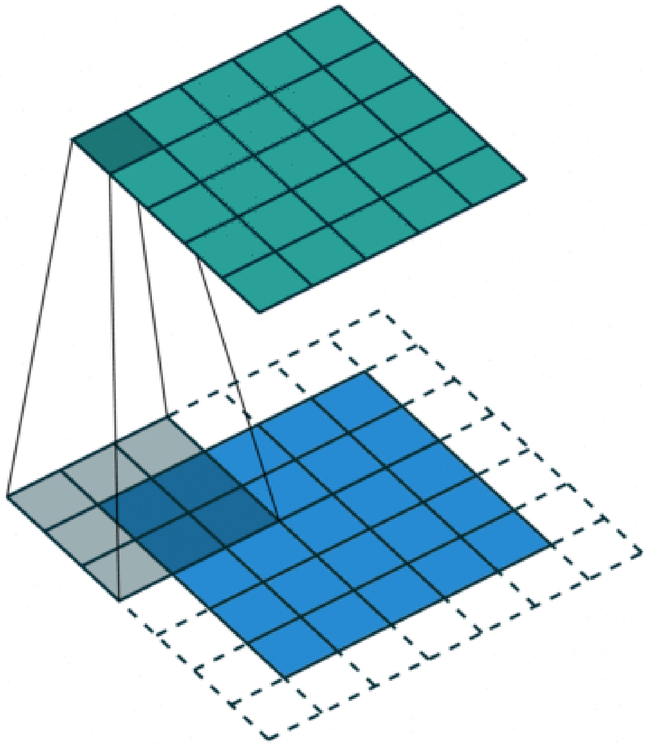


What is convolution?

Convolution with a 3*3 filter

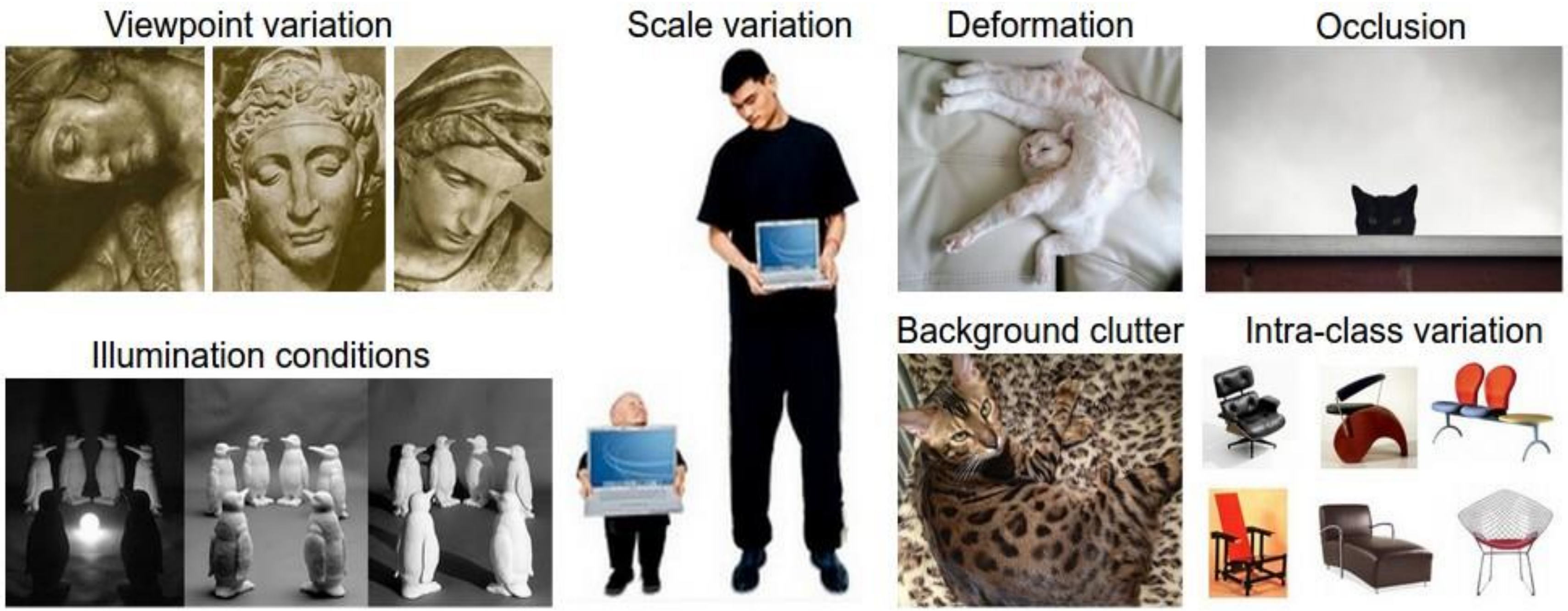


Convolution with padding to keep the dimensions



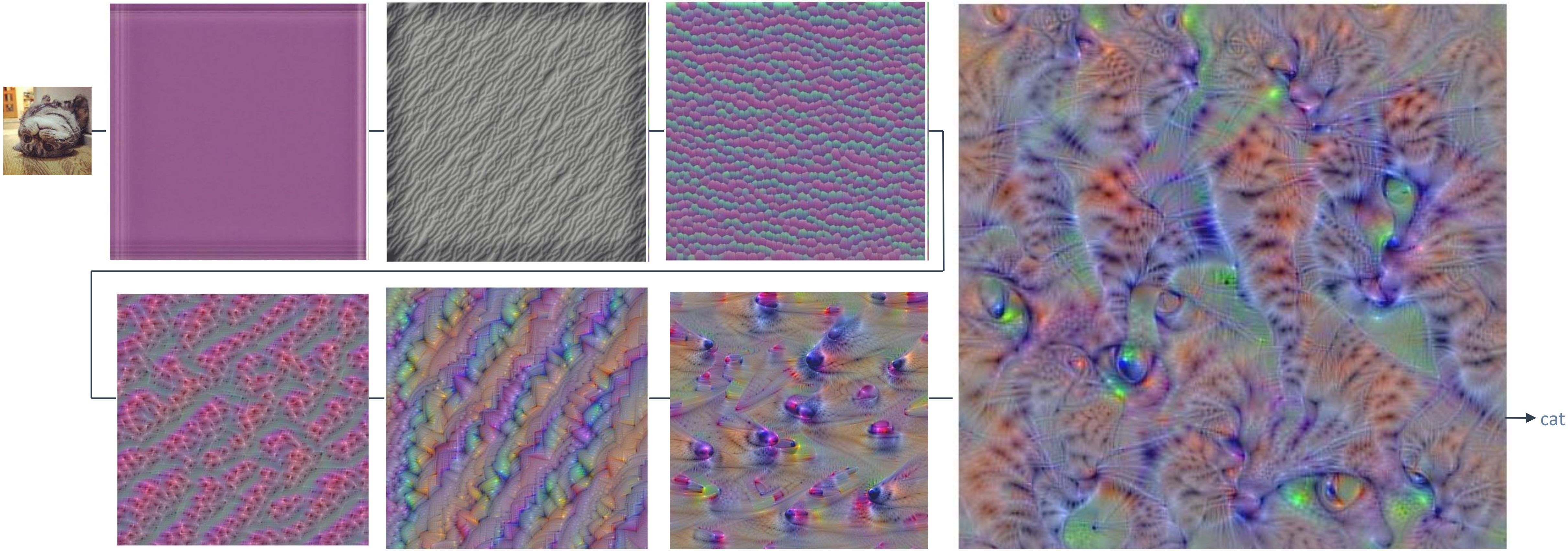
Dumoulin & Visin. A guide to convolution arithmetic for deep learning (2018) https://github.com/vdumoulin/conv_arithmetic

Image variations exacerbate the manual extraction of features



https://cs231n.github.io/convolutional-networks/

Convolutional neural networks employ thousands of self-learned filters



Convolutional neural networks

Convolution: $z_{i',j'} = \text{ReLU}\left(b + \sum_{i=1}^{H} \sum_{j=1}^{W} x_{i'+i-1,j'+j-1} \cdot w_{ij}\right)$, with i', j' feature map indices and i, j kernel indices

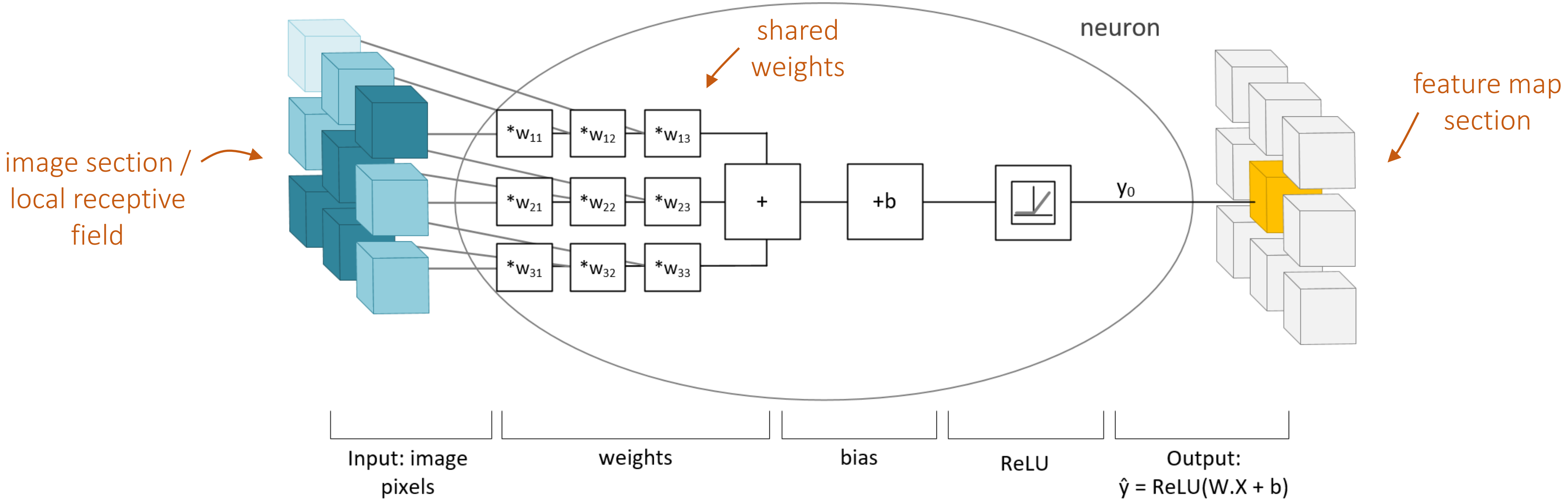
output, located at the kernel‘s center

bias

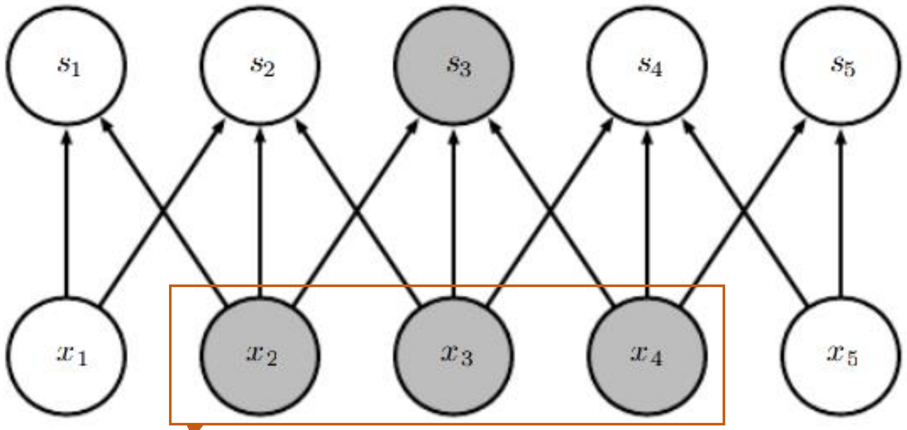
kernel height H und width W

input image pixel

kernel weight

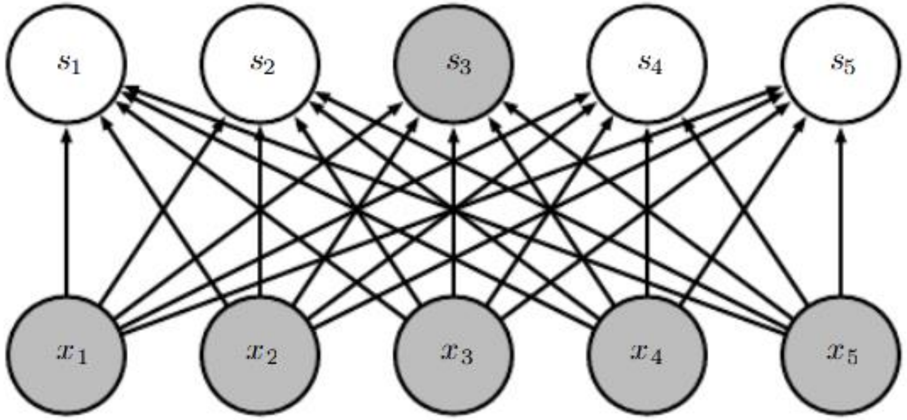


Sparse connectivety



receptive field

In CNNs, a neuron / unit is only affected by a small number of inputs



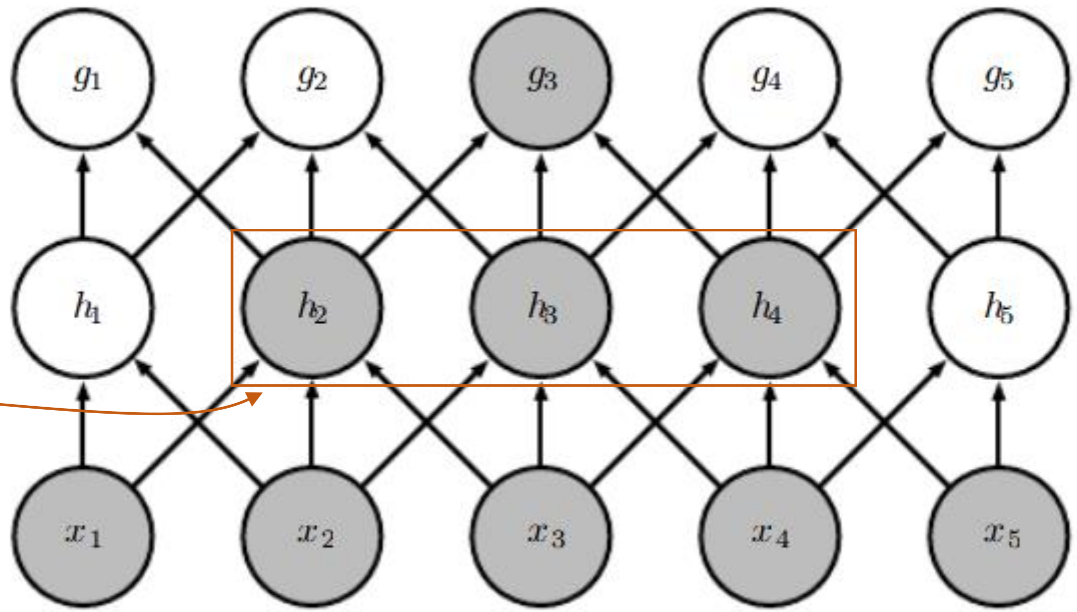
In fully-connected networks, a neuron / unit is affected by all inputs

Receptive field

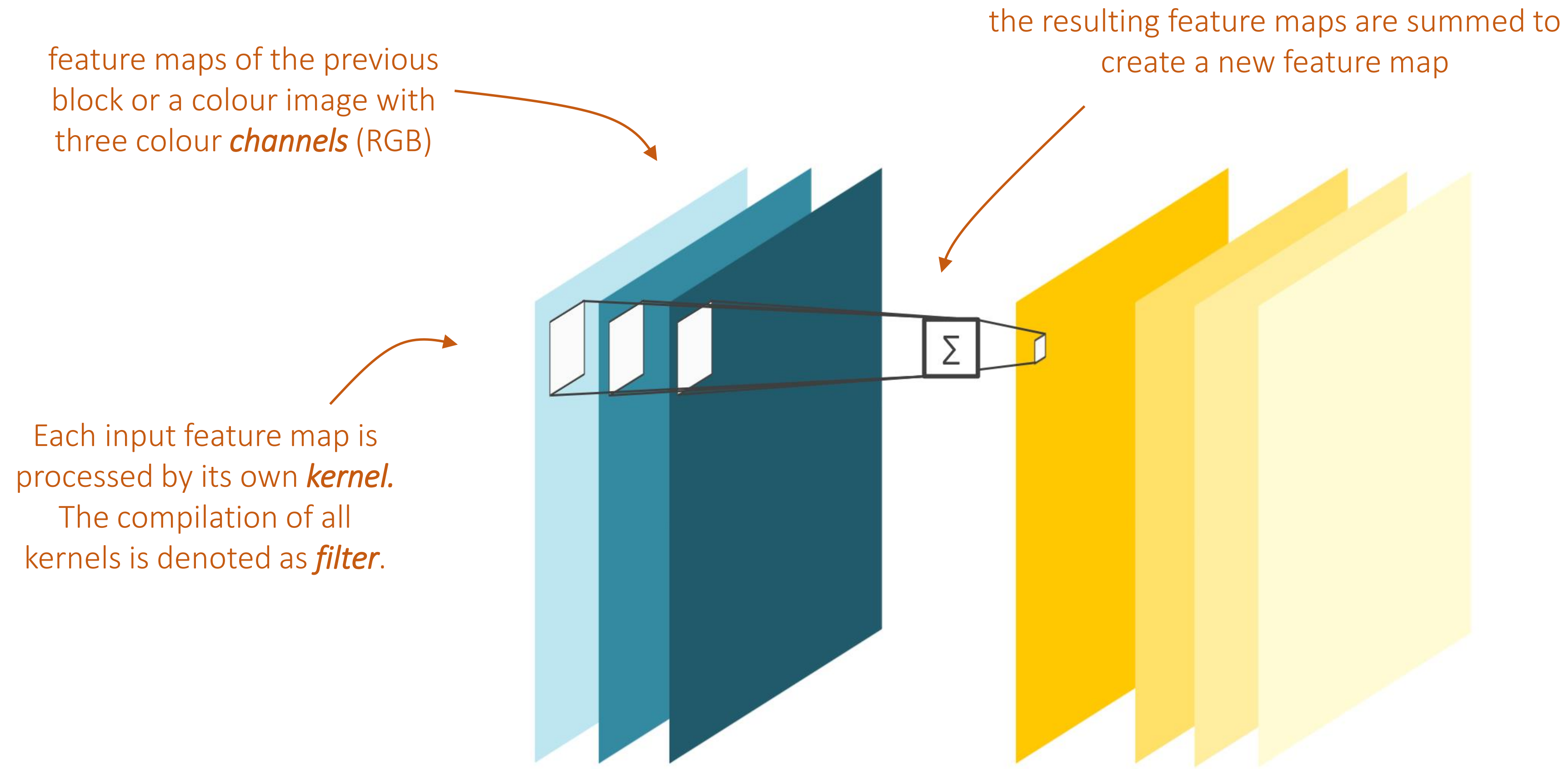
While the initial receptive field is quite small, e.g. 3*3, it increases with increasing network depth

→ neurons deep in the network can indirectly be connected to all or most of the input image

receptive field



Each feature map is composed of the sum of the previous filtered feature maps



Input size: $W_1\ x\ H_1\ x\ D_1$
New size: $W_2\ x\ H_2\ x\ D_2$, with:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

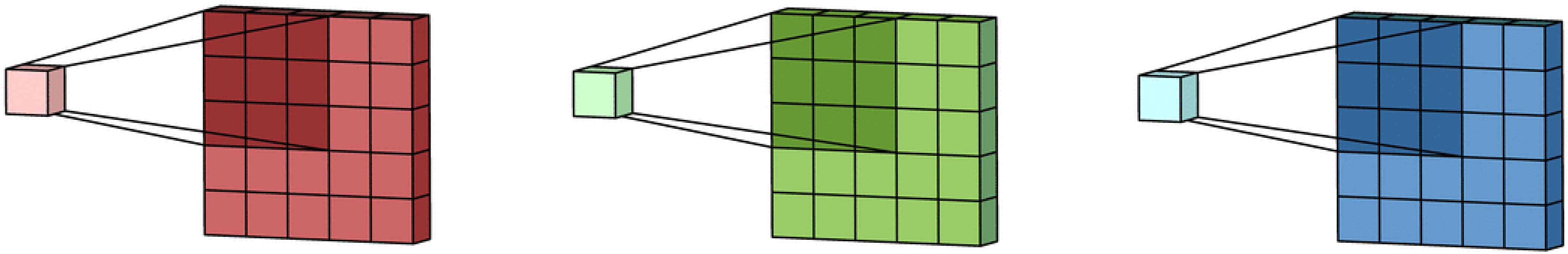
$$D_2 = K$$

Where K is the number of filters, F is the filter size, S is the stride, and P is the amount of zero padding.

Number of weights per filter = $F \cdot F \cdot D_1$

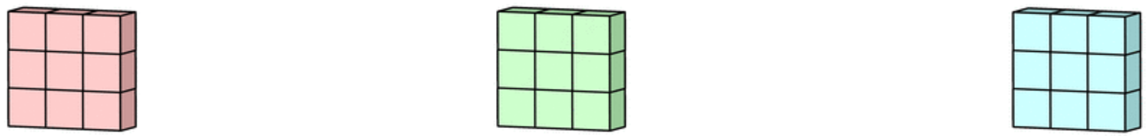
Total number of weights = $(F \cdot F \cdot D_1) \cdot K$ plus K biases

Each kernel processes a different input feature map or a different input channel



Kernels & filters

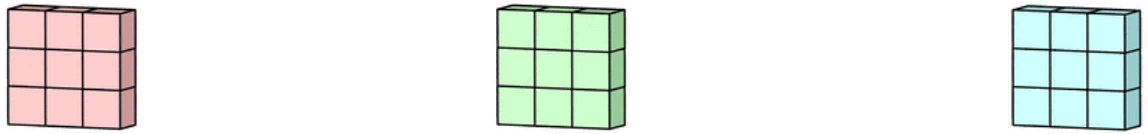
- All separately processed versions are then summed together, to form output feature map.
- The kernels of a filter each produce one version of each input feature map, and the filter as a whole produces one overall output feature map.



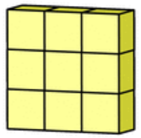
https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Kernels & filters

- All separately processed versions are then summed together, to form output feature map.
- The kernels of a filter each produce one version of each input feature map, and the filter as a whole produces one overall output feature map.



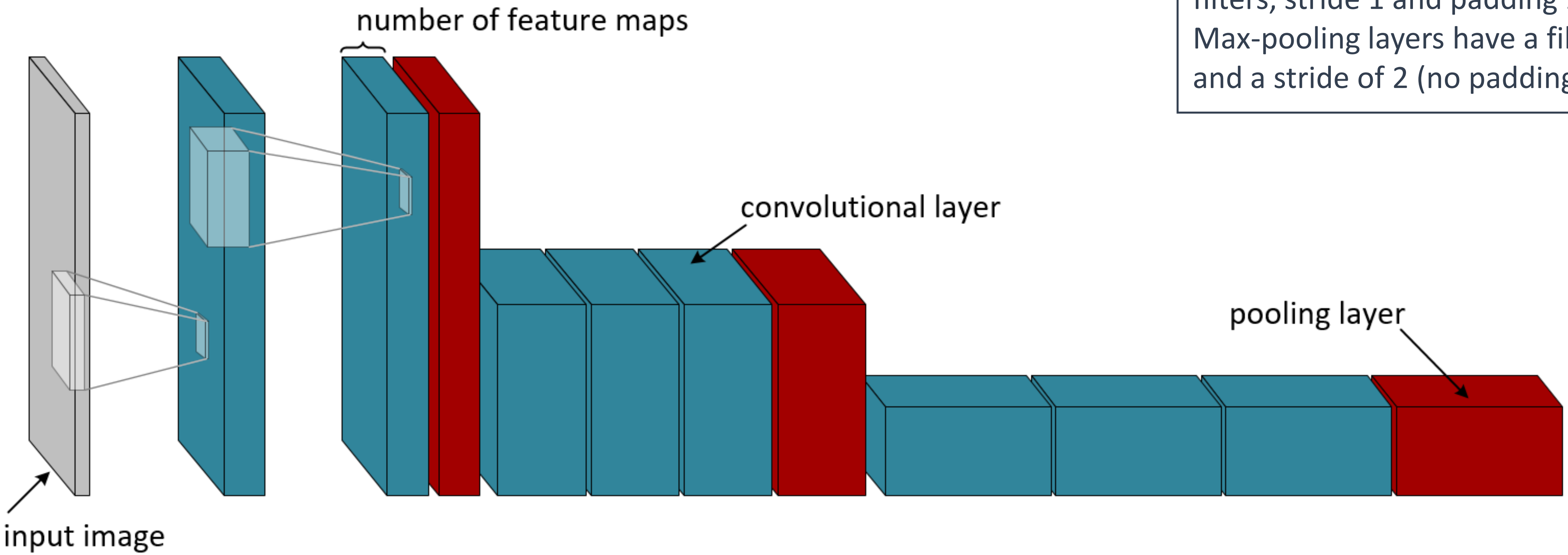
- Finally, the bias is added to each pixel



https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

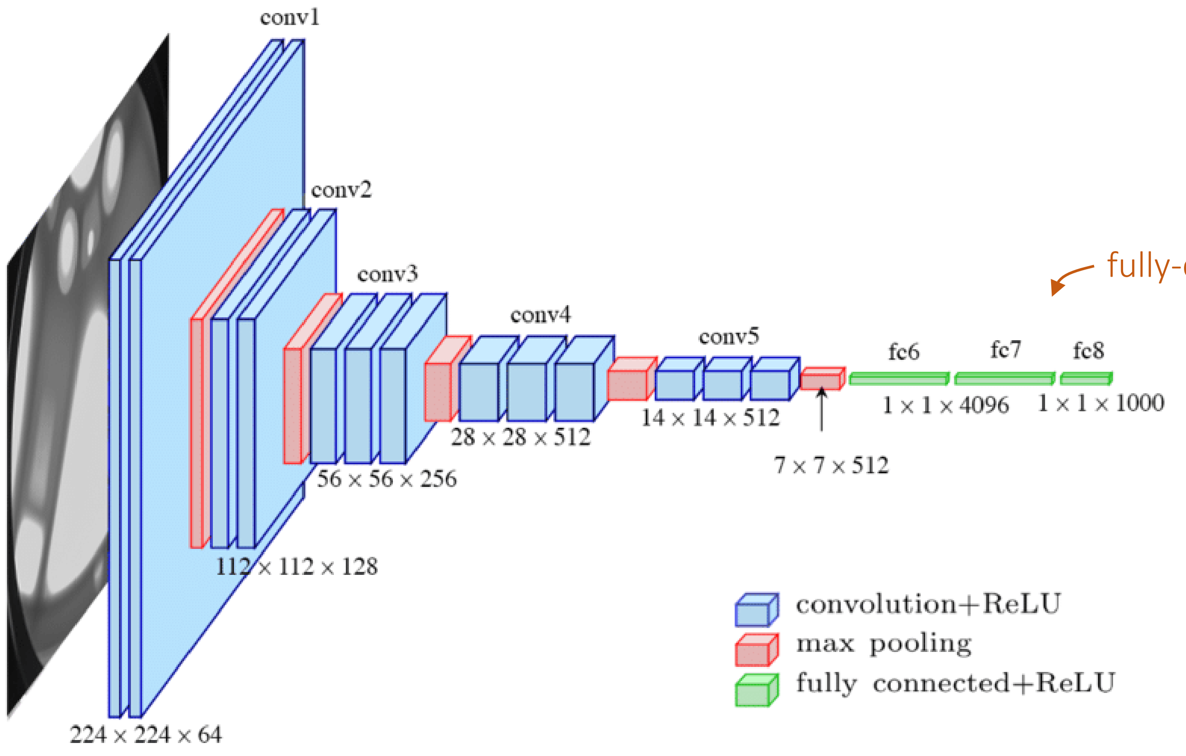
Convolutional neural networks

Design rules:
Blocks of two to three convolutional layers followed by a max-pooling layer.
Each layer has 64 to 512 filters, with 3x3 filters, stride 1 and padding 1.
Max-pooling layers have a filter size of 2x2 and a stride of 2 (no padding).



Setup of a convolutional neural network

The VGG-16 network architecture



fully-connected layers

feature map height x width x number of filters

VGG 16: Simonyan & Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition (2014)
Figure source: Ferguson et al. Automatic localization of casting defects with convolutional neural networks (2017)

Sub-sampling | Max-pooling

Sub-sampling reduces the feature map size

- Reduces the sensitivity of the output to shifts and distortions by introducting invariance to small translations
- Reduces the computational cost and thus allows the typical bi-pyramid:
  decreasing feature map sizes & increasing number of filters with increasing network depth
- Mostly: max-pooling, also possible: average pooling, L2 pooling

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 1 & 1 \\ 2 & 1 & 4 & 1 \\ 3 & 1 & 1 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix}$$

Input size: $W_1 \ x \ H_1 \ x \ D_1$
New size: $W_2 \ x \ H_2 \ x \ D_2$, with:

$$W_2 = \frac{W_1 - F}{S} + 1$$

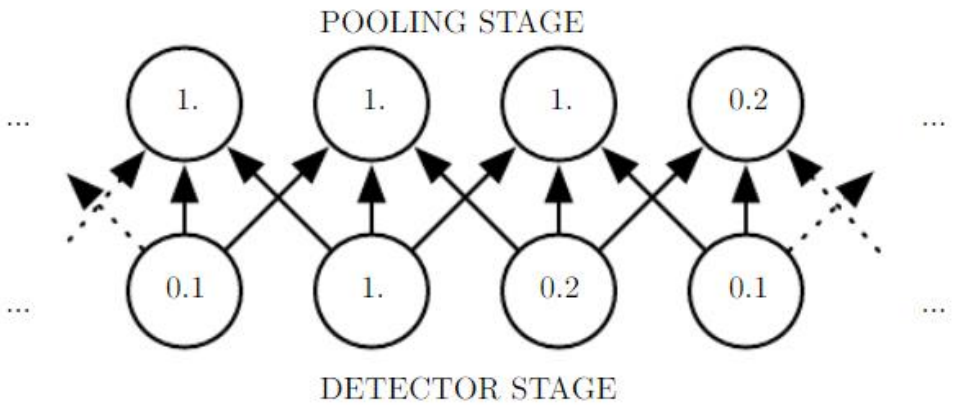
$$H_2 = \frac{H_1 - F}{S} + 1$$

$$D_2 = D_1$$

Where F is the pooling kernel size, here 2 and S is the stride, here 2

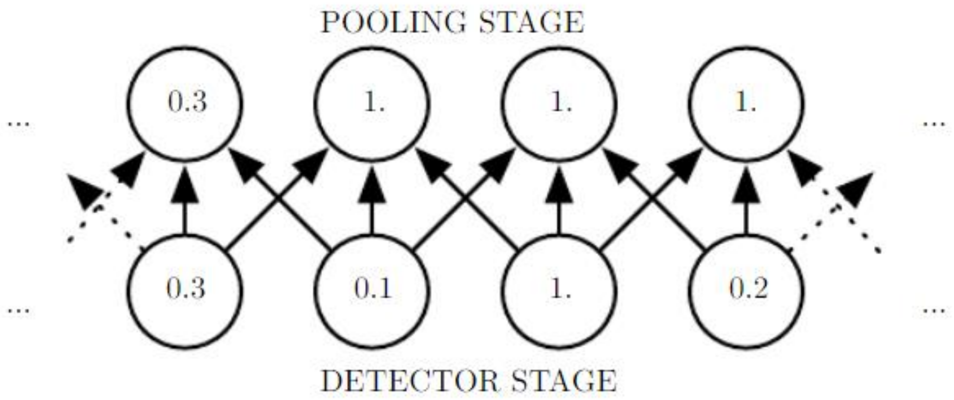
Sub-sampling | Max-pooling

Shifting the inputs by one changes every input pixel, but only half the values in the max-pooling output
→ Adding max-pooling introduces a prior that the learned function must be invariant to small translations



result of the max-pooling operation

input to the max-pooling layer coming from a convolutional layer



result of the max-pooling operation

the same input, but shifted by one pixel

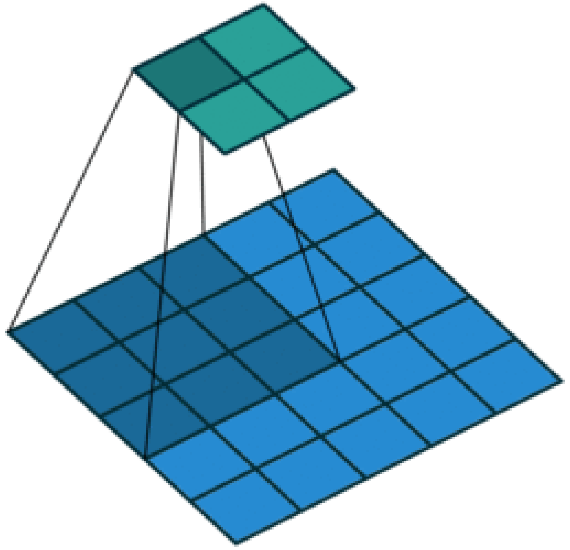
Goodfellow et al. Deep Learning.

Machine Learning Research

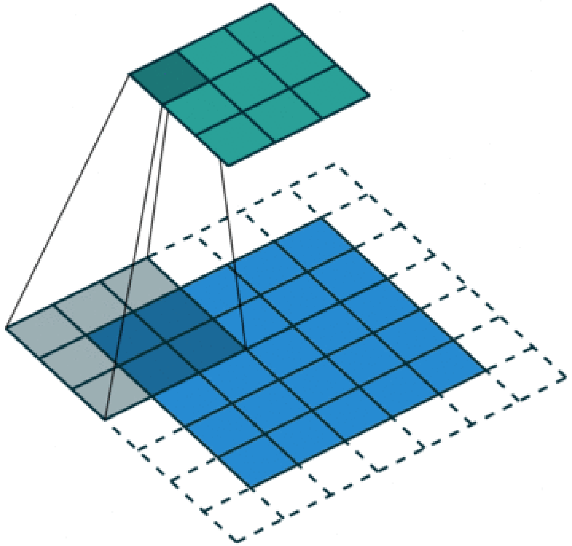
Getting rid of max pooling

- Max-pooling may result in a loss of accurate spatial information
- Instead of sub-sampling with max pooling, we can also use convolutional layers, with a stride of 2 (or bigger)
- Often used in GANs and variational autoencoders

Convolution with a 3*3 filter and stride 2



Convolution with a 3*3 filter, padding, and stride 2



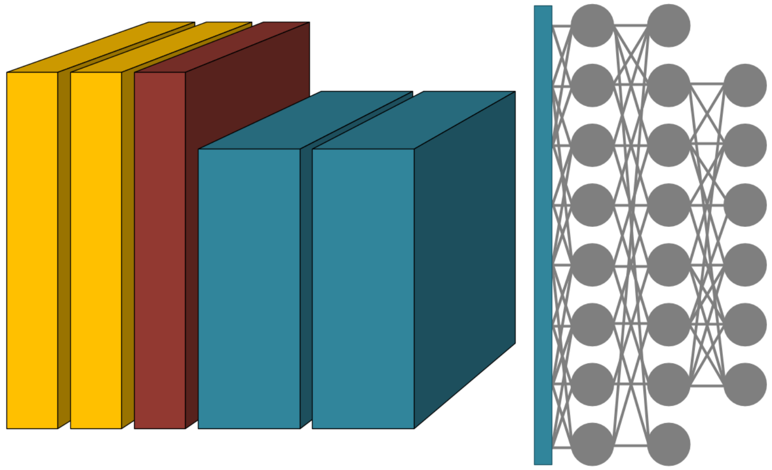
Springenberg et al. Striving for Simplicity: The All Convolutional Net (2014)

From convolution to fully-connected layers

To output a classification, convolutional neural networks transition from the convolutional part (feature detection) into the classifier part, consisting of fully-connected layers

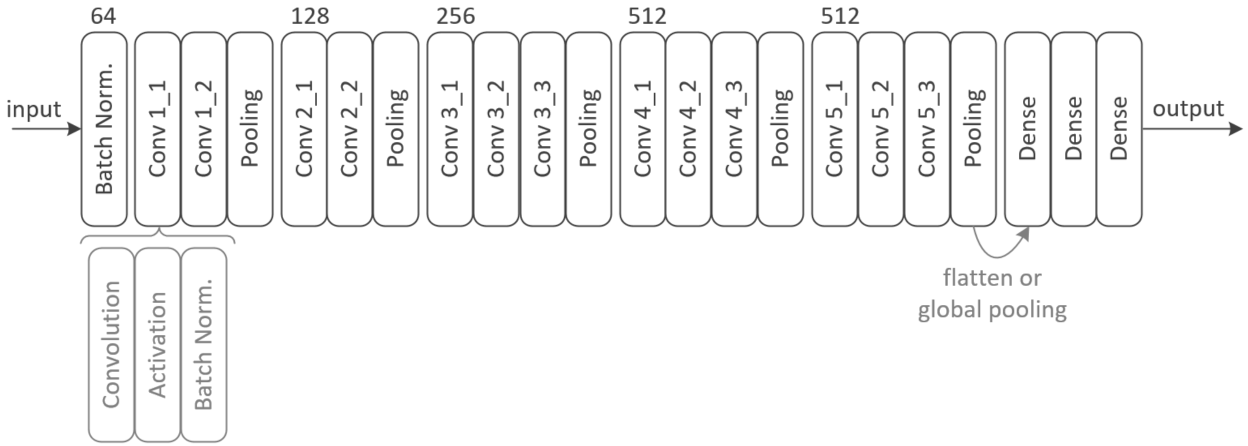
There are two ways to connect both layer types:

- Flatten
  - Use the flattened last convolutional layer as input for the first fully-connected layer
    (batch_dim, H, W, num_channels) → (batch_dim, H*W*num_channels)
- Global average pooling
  - Computes the mean of each feature map
    (batch_dim, H, W, num_channels) → (batch_dim, num_channels)
  - Also available as max operation



Conclusion

- Convolutional neural networks combine a feature detection part (convolutional layers) with a classification part (fully-connected layers)
- Since 2014, CNNs are the state-of-the-art for image classification
- Next time, we will look into more advanced CNNs



Machine Learning Research


ANY QUESTIONS?
memecrunch.com