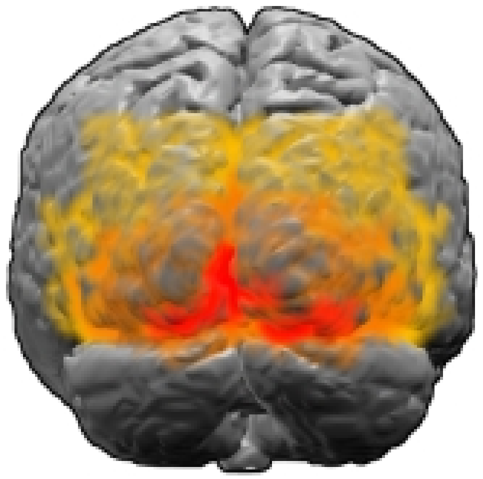
Introduction to Data Analysis and Machine Learning in Physics: 5. Convolutional Neural Networks (CNN)

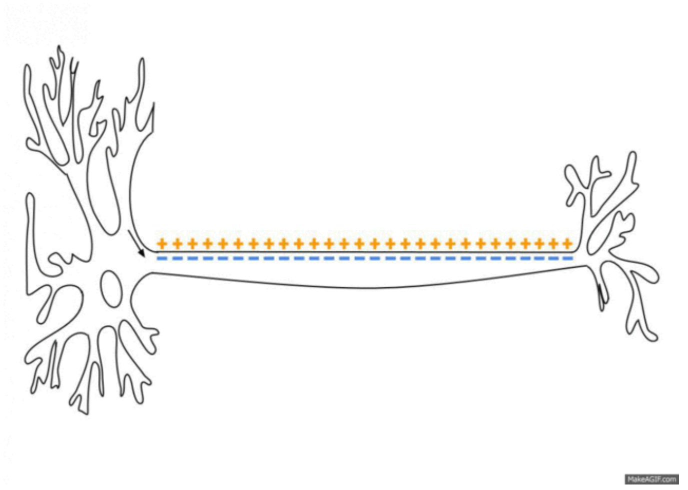
Jörg Marks

Studierendentage, 7-11 April 2025



Visual cortex, Wikipedia

CNN Overview

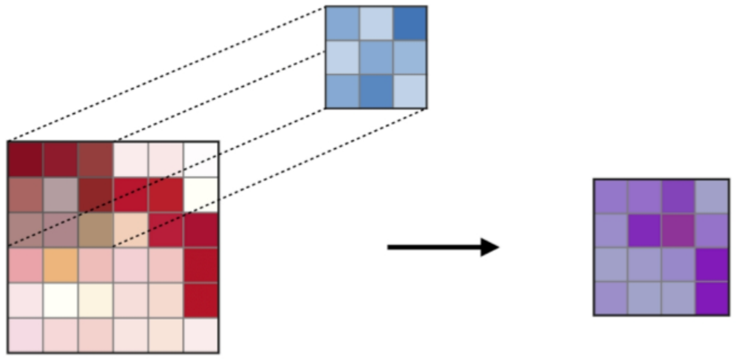


- Neurons in the visual cortex fire signals (action potential) in case of visual stimuli in the field of vision (receptive field). Each neuron responds only to a subset of the receptive field and passes the signals to the neurons of the next layers with increasing complexity from layer to layer.

- Convolutional Neural Networks were inspired by this type of biological models. They are feed forward neural networks which learns feature engineering via filters (also called kernels). They lead to regularized weights and create less connections in the propagating network. The intention was to prevent vanishing and exploding gradients during back propagation and to overcome over training.

- The convolution filters (kernels) slide along the input features and provide new feature (multiple) maps (convolutional layers) of reduced size. The neurons in one layer are fully connected to the neurons in the next layer (feed-forward neural network)
  → leads to reduced set of parameters which typically appear in the structure finding of large images

- Key component of CNNs: Convolutional layers
  - Set of learnable filters
  - Low-level features at the first layers → high level features towards the end



CNN Overview

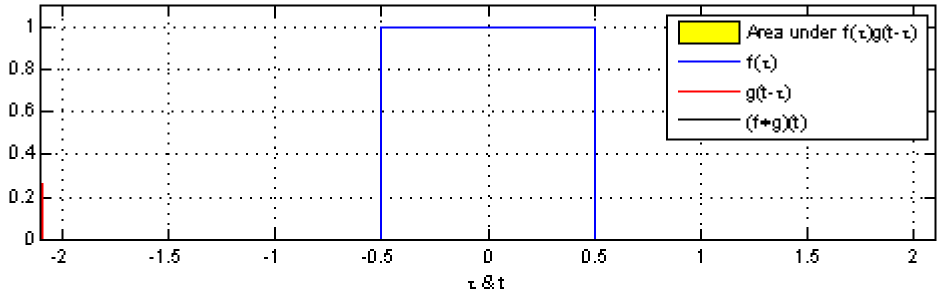
- Typical applications are
  - audio recognition and language processing
  - image and video recognition
  - medical image analysis
  - financial time series
- CNNs learn to optimize the filters (kernels) and extract specific image features by applying a small matrix of numbers (called filtering)

Convolutional Neural Networks

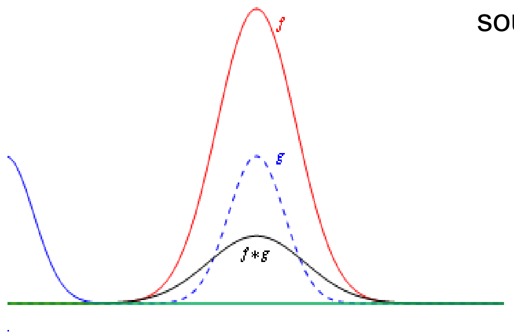
- A Convolutional Neural Network (CNN) learns features directly from data without specifying the features explicitly. CNNs are useful for pattern recognition in images and computer vision, but also for classification tasks in audio data and signal processing.
- Learning the features happens via convolution filter, which are activated by scanning over the pixel of an image. Edges or structure and color changes are transported to another filter layer. A Rectified Linear Unit (ReLU) is applied as activation function. Via a pooling technique the number of parameters is reduced from layer to layer.
- Convolution $$(f * g)(x) = \int_{\mathbb{R}^n} f(\tau) g(x - \tau) d\tau$$

$f(x)$: input distribution $g(x - \tau))$: kernel or filter function

The function value of the weight function f at the position τ tells us how strong the contribution of g(x – τ) in f at the position x is.



source Wikipedia



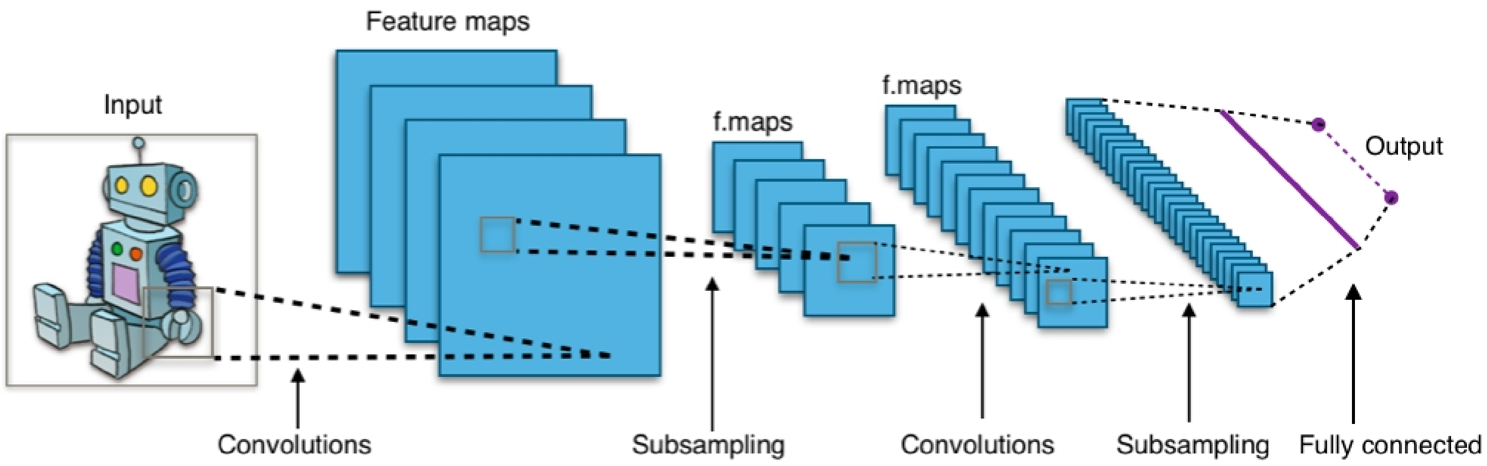
- Discrete convolution $$S(i,j) = (I * K)(i,j) = \sum_m \sum_n \underbrace{I(i-m, j-n)}_{\text{image}} \underbrace{K(m,n)}_{\text{kernel}}$$

Play with images and and various kernel for image processing: https://setosa.io/ev/image-kernels/

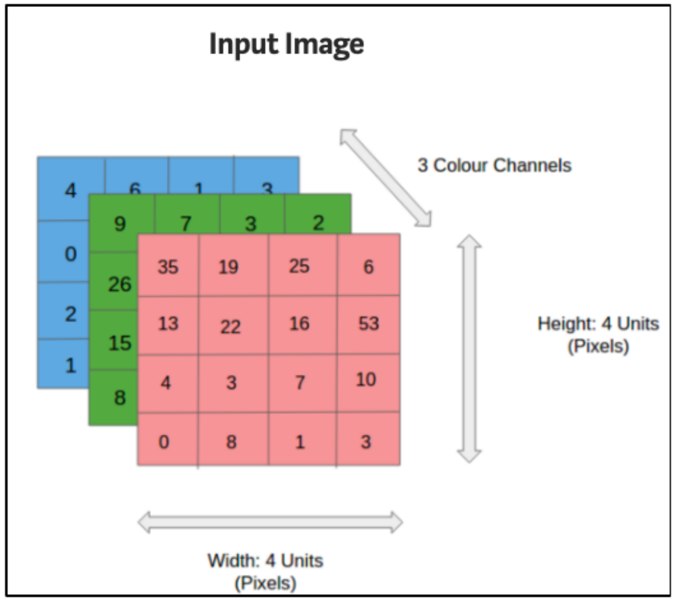
Convolutional Neural Networks

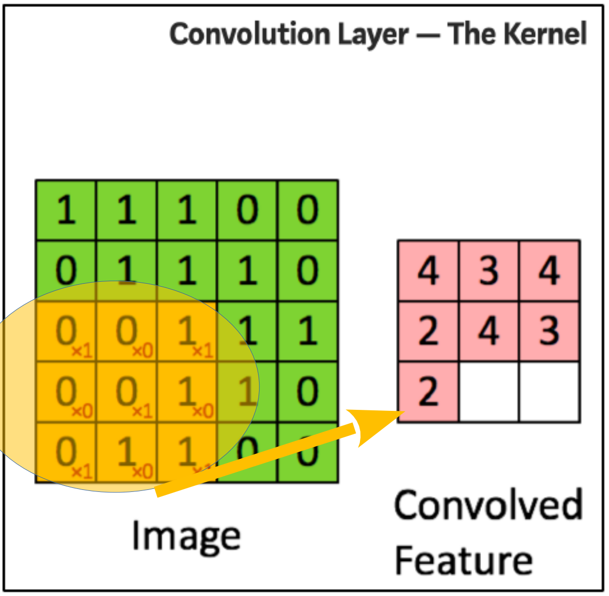
More information can be found in the Stanford course https://cs231n.github.io/convolutional-networks/

- Structure of a typical CNN used in image classification



- Main idea is to extract particular localized features of data, eg. an image, using a convolution as filter mechanism. The input is a data structure [xPix,yPix,3] of raw pixel values with three color channels R,G,B. Each color channel is treated independently.





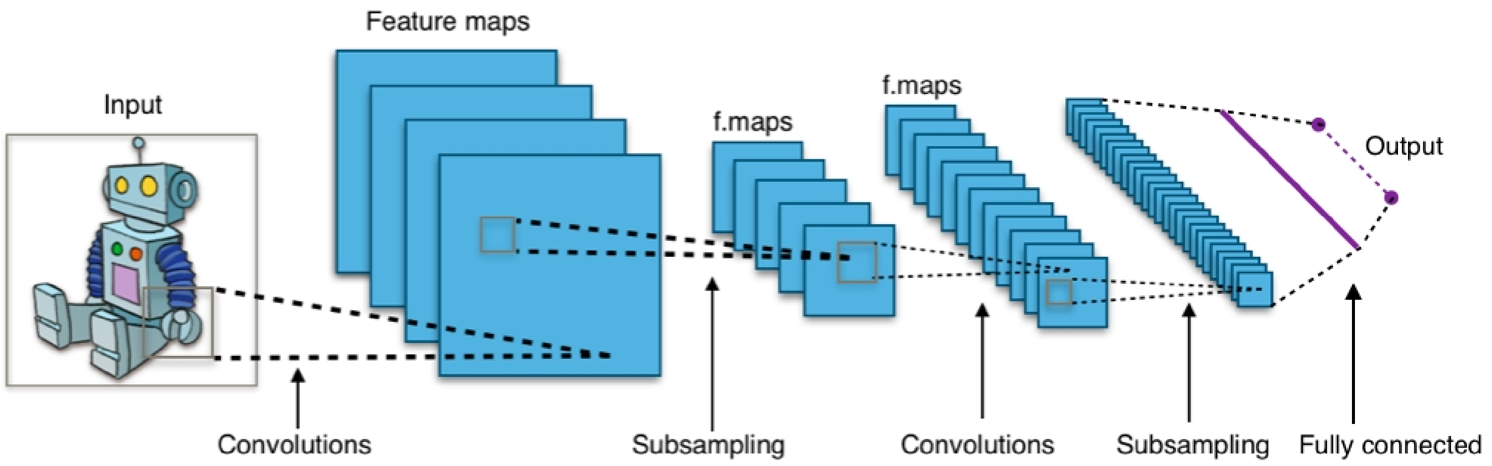
```
Kernel/Filter, K =

1  0  1
0  1  0
1  0  1
```

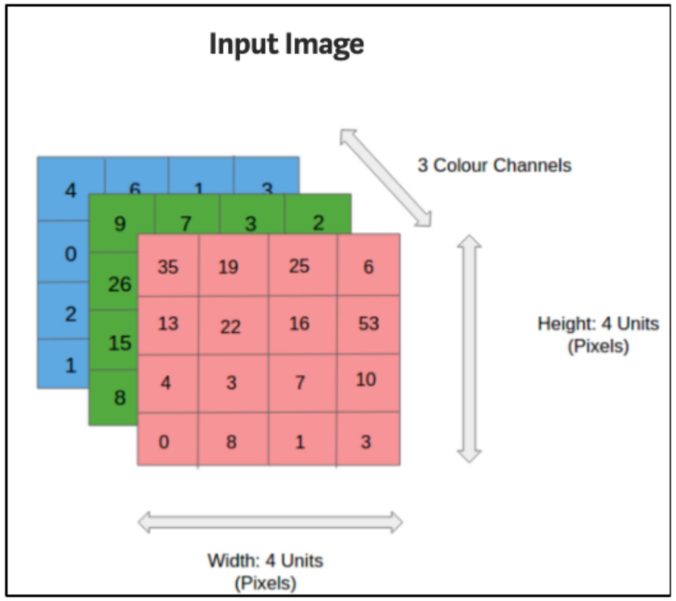
Convolutional Neural Networks

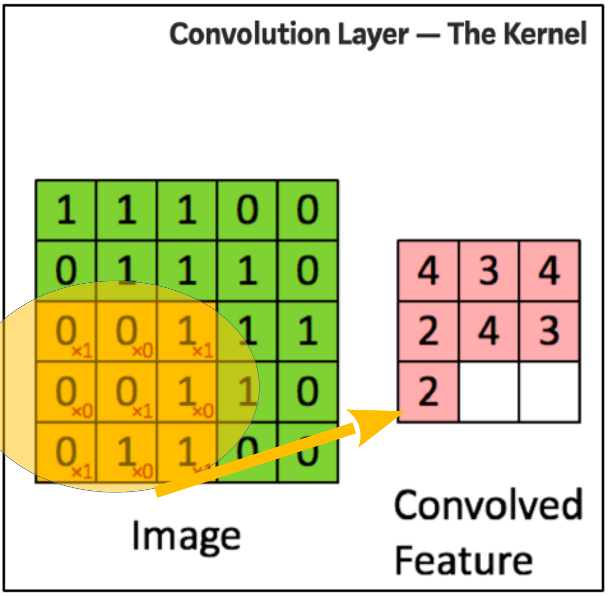
More information can be found in the Stanford course https://cs231n.github.io/convolutional-networks/

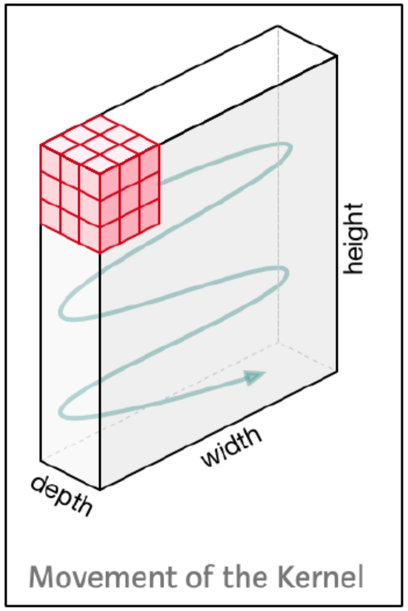
- Structure of a typical CNN used in image classification



- Main idea is to extract particular localized features of data, eg. an image, using a convolution as filter mechanism. The input is a data structure [xPix,yPix,3] of raw pixel values with three color channels R,G,B. Each color channel is treated independently.



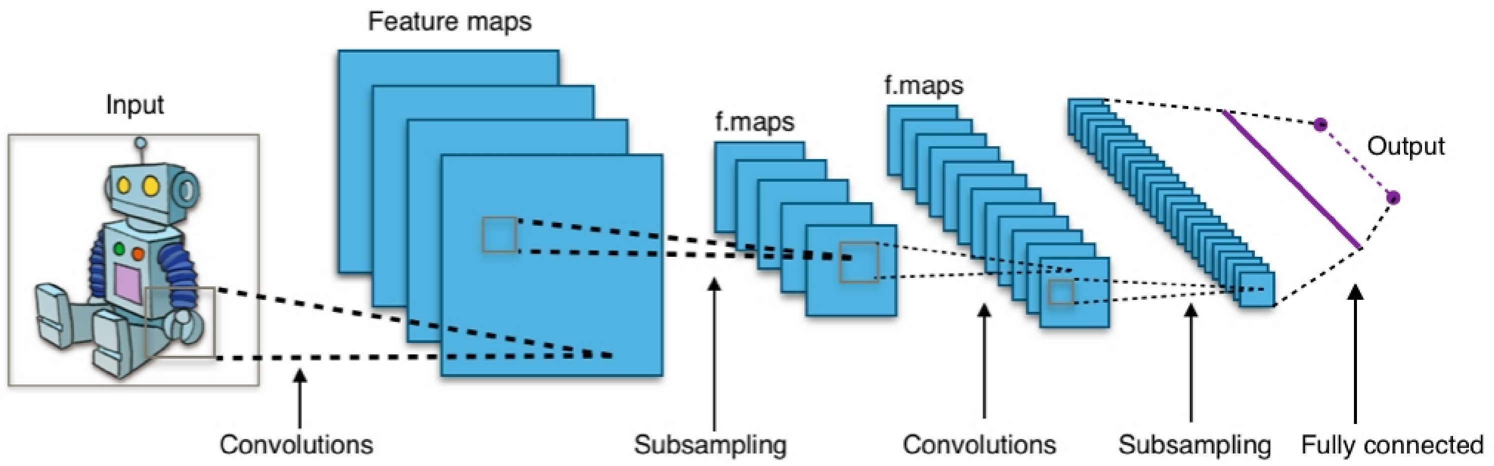




Convolutional Neural Networks

More information can be found in the Stanford course https://cs231n.github.io/convolutional-networks/

- Structure of a typical CNN used in image classification



- Main idea is to extract particular localized features of data, eg. an image, using a convolution as filter mechanism. The input is a data structure [xPix,yPix,3] of raw pixel values with three color channels R,G,B. Each color channel is treated independently.

- There are 3 main building blocks:

  I) Convolutional layer

  ○ Obtain a weight matrix by computing the dot product of the weight matrix and a small sub region of the input data and scanning over the whole image. This results in a weight matrix which transports certain features of the image to further layers.

  ○ To the weights activation functions like ReLU are applied. The convolution operation (weight matrix · sub input structure) behaves like a filter.

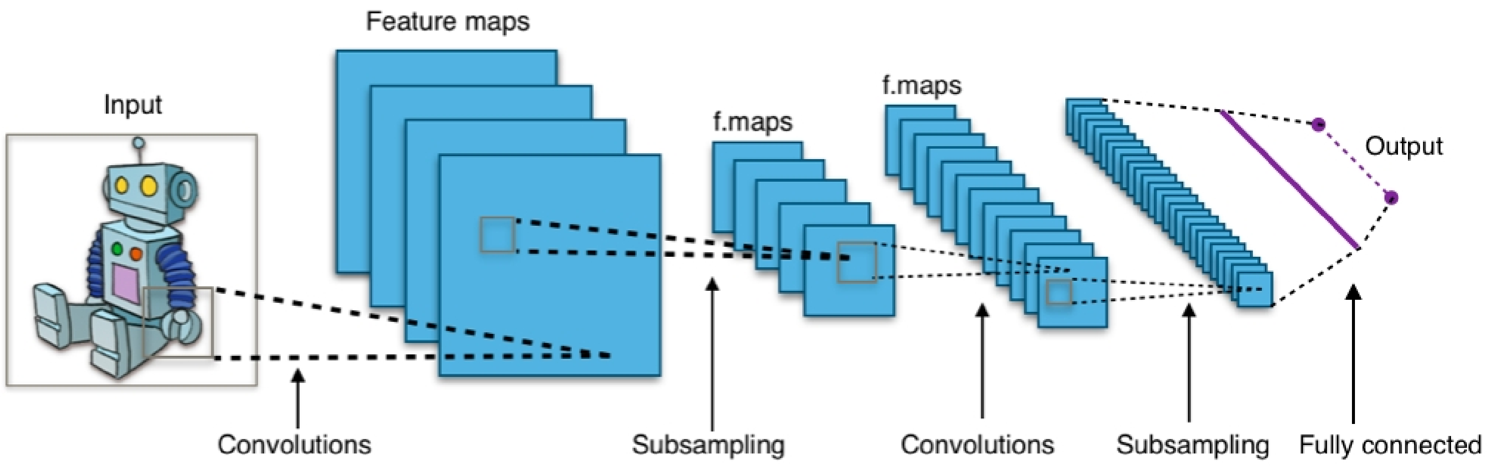
  ○ The weight matrix is determined by a loss function.

  ○ Multiple convolutional layers extract with increasing depth more and more complex features

Convolutional Neural Networks

More information can be found in the Stanford course https://cs231n.github.io/convolutional-networks/

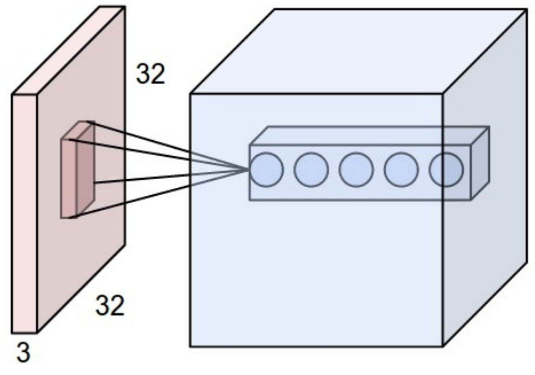
- Structure of a typical CNN used in image classification

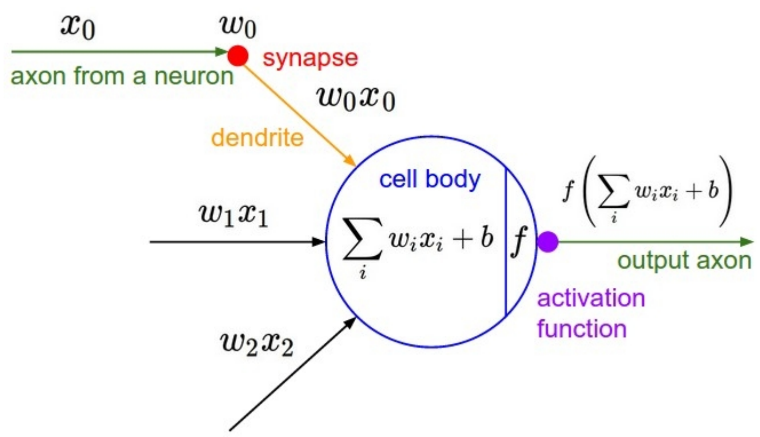


- Main idea is to extract particular localized features of data, eg. an image, using a convolution as filter mechanism. The input is a data structure [xPix,yPix,3] of raw pixel values with three color channels R,G,B. Each color channel is treated independently.

- There are 3 main building blocks:

I) Convolutional layer

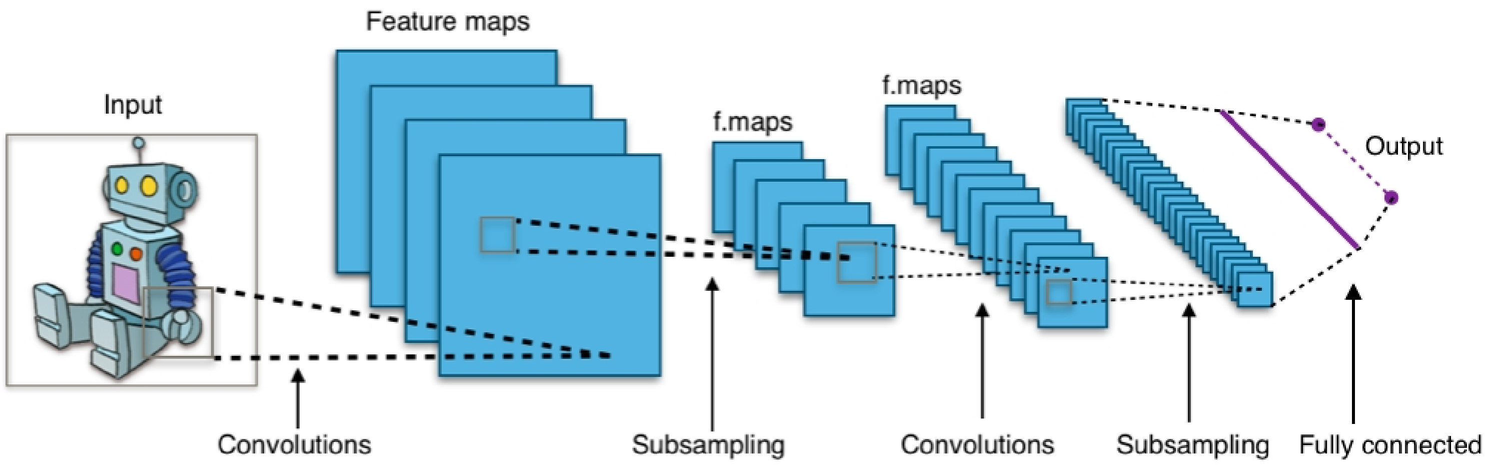




More information can be found in the Stanford course https://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks

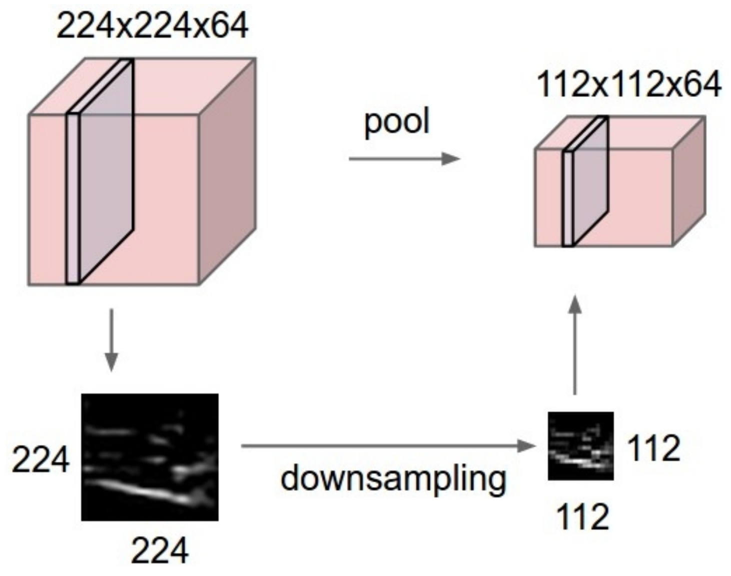
- Structure of a typical CNN used in image classification



II) Pooling layer

- Several neighboring pixel are pooled together by averaging or by taking their maximum.
- Max Pooling algorithm: Take a 4x4 input (W), step (S) with a 2x2 filter (F) over the input, take the maximum entry
- Zero Padding: to the edges of the picture pixel are added (P) and filled with zeros
- Output size OP :

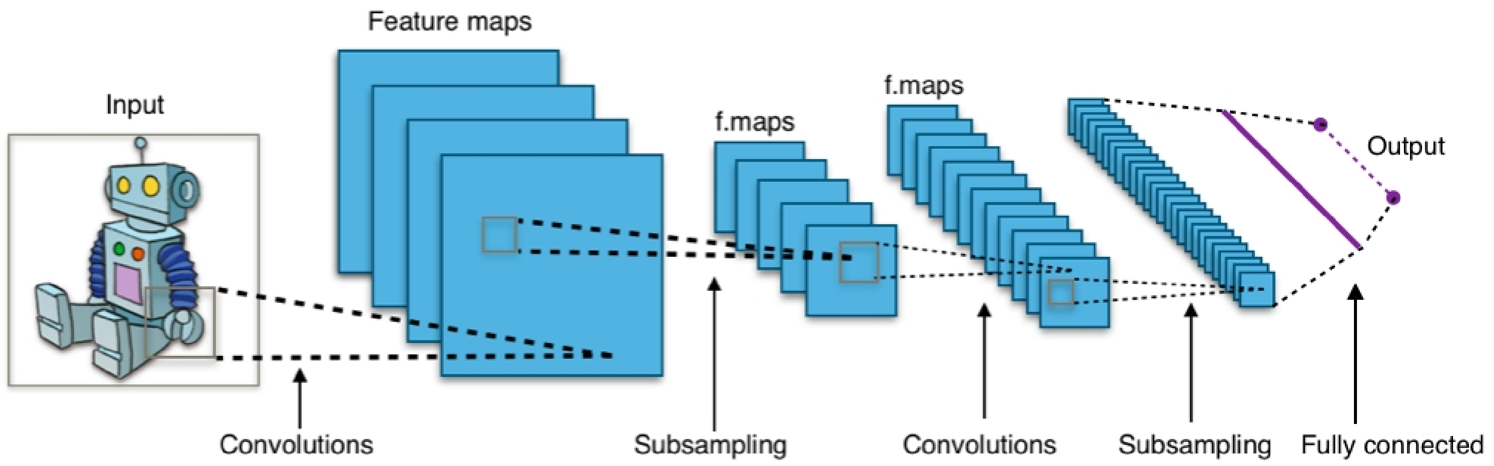
$$OP_{x,y} = \frac{W - F_{x,y} + 2P}{S_{x,y}} + 1$$



More information can be found in the Stanford course https://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks

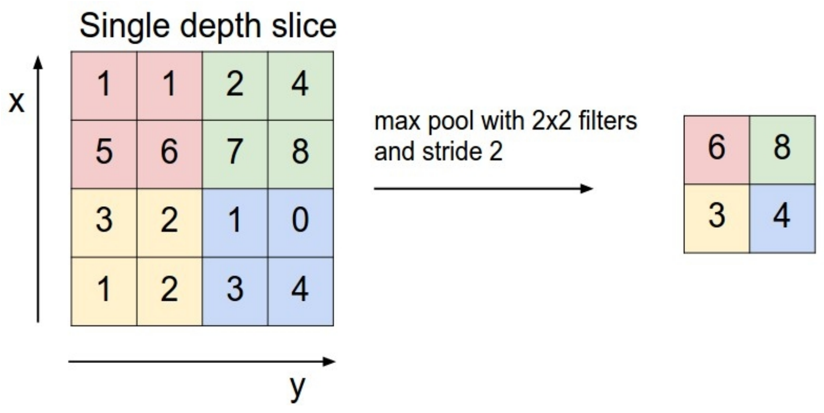
- Structure of a typical CNN used in image classification



II) Pooling layer

- Several neighboring pixel are pooled together by averaging or by taking their maximum.
- Max Pooling algorithm: Take a 4x4 input (W), step (S) with a 2x2 filter (F) over the input, take the maximum entry
- Zero Padding: to the edges of the picture pixel are added (P) and filled with zeros
- Output size OP :

$$OP_{x,y} = \frac{W - F_{x,y} + 2P}{S_{x,y}} + 1$$



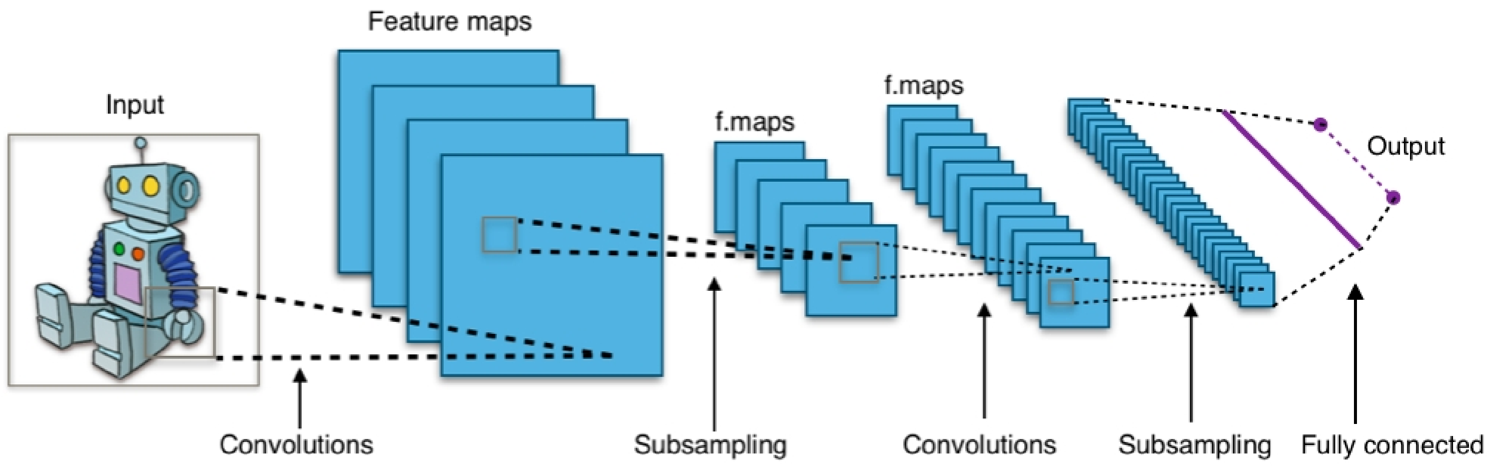
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

More information can be found in the Stanford course https://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks

- Structure of a typical CNN used in image classification



III) Output layer

- The final stage is a fully connected layer which works as MLP to generate an output
- The very last layer is equal to the number of output classes.
  In order to obtain probabilities for each class
  SoftMax is applied
- A loss function determines all the weights

Input Layer

Convolutional Layer

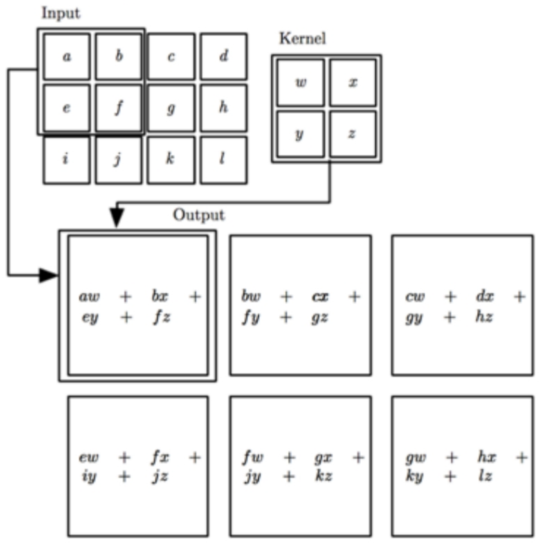
- Convolution Stage
- Activation Stage
- Pooling Stage

Output Layer

Convolutional Neural Networks

- 2D Convolution as matrix multiplication

Matrix multiplication for 2D Convolution



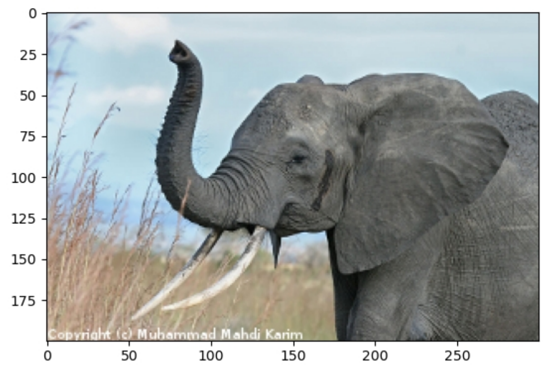
Fewer weights needed than for the full matrix multiplication

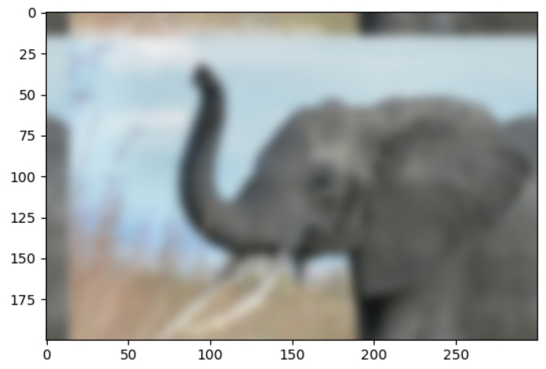
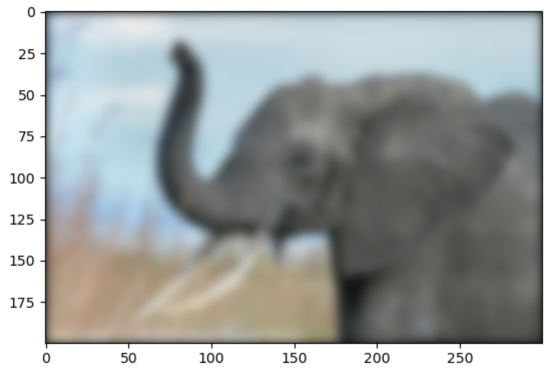
- Example: blur an image by convolution with a Gaussian kernel
  - the bluring increases with increasing width of the gaussian

  (here an FFT is used instead of the matrix multiplication)

plot_image_blur.ipynb

lectures.scientific-python.org



MNIST classification with a CNN in TensorFlow / Keras

read and process data

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Normalize the pixel values to be between 0 and 1
x_train = x_train / 255
x_test = x_test / 255

# Reshape the images into 4D arrays for use with a CNN
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

# Convert the labels into one-hot encoded arrays
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)
```

Model definition

```
# conv layer with 32 3x3 filters
model = Sequential(
    [
    Input(shape=input_shape),
    Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(32, activation="relu"),
    Dense(10, activation="softmax"),
    ]
)

model.summary()
```

Compile Model

Using Keras, you have to `compile` a model, which means adding the loss function, the optimizer algorithm and validation metrics to your training setup.

```
model.compile(loss="categorical_crossentropy",
        optimizer="adam",
        metrics=["accuracy"])
```

Model training and displaying of the weights

```
# Train the model and record the history
history = model.fit(x_train, y_train, epochs=3, batch_size=64,
          validation_data=(x_test, y_test))


# Get the weights of the Dense layer
# plot the weights as a heatmap or image, where the weights are represented
# as pixel values.
# model.layers[2].get_weights()[0] returns only the weights of the third
# layer. If you wanted to get the biases, you would use
# model.layers[2].get_weights()[1].
#dense_weights = model.layers[2].get_weights()[0]
last_layer_weights = model.layers[-1].get_weights()[0]
# Plot the weights as a heatmap
plt.imshow(last_layer_weights, cmap='coolwarm')
plt.colorbar()
plt.title('weights in the output layer')
plt.show()
```

Examples - Deep Neural Networks

- Two extreme cases of training results

  - If the model does not reflect the data content or the training is insufficient
    → bad network performance

  - If the model allows for to much complexity it learns features of the training data sample
    → network can be applied to other samples (overtraining effect)

    test overtraining in the example clothing dataset by including the method Dropout in TF or changing the number of nodes in the hidden layer

- Other classification examples with Tensorflow and Keras

  - uses the Fashion MNIST dataset of Zalando, which contains 60,000 gray scale images in 10 categories each showing low resolution clothing pictures.

  - sequential model with two dense layers → significant overtraining with test accuracy = 87.7%

    clothingSequential.ipynb

  - sequential model with two dense layers adding Dropout and softmax activation
    → no over training with test accuracy = 88.2 %

    clothingSequentialDropout.ipynb

  - CNN model where the input data is shaped to a 4D tensor of shape (samples, height, width, channels). MaxPooling2D and 3x3 Filters and ReLu activation and the Dropout feature is used → no over training with test accuracy = 91.4 %

    clothingCNN_4D.ipynb

Exercise: Train a digit-classification neural network on the MNIST dataset using TensorFlow/Keras

a) Plot training and validation loss as well as training and validation accuracy as a function of the number of epochs

b) Determine the accuracy of the fully trained model using the test set

c) Determine a confusion matrix plotting the true label versus the found label of the test set

d) Try to improve the performance of the network by changing the number of filters and by adding a second convolutional layer

e) Can you plot some examples from the test set and their predictions

CNN_MNIST_digits_tf.ipynb