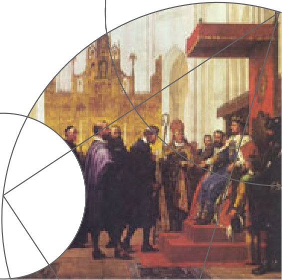
Faculty of Science

# Big Data Analysis (Applied Machine Learning)

## Convolutional Neural Networks (CNN) and images.

Aleksandar Topic
Niels Bohr Institute, University of Copenhagen

May 13th, 2020



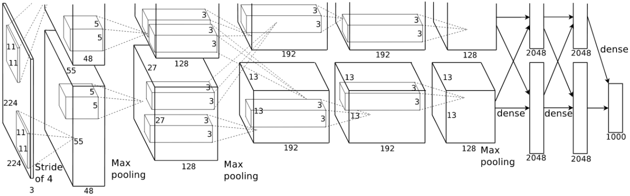
# Overview

- Brief recap of Artificial Neural Networks (ANNs)
- Images on a computer
- CNN architecture and building blocks
- Training process and inference
- Implementation in Python
- Examples and perspectivation

# Recap of Artificial Neural Networks (ANNs)

- A model used in both supervised and unsupervised learning
- Usecases include regression, classification, segmentation, compression, etc.
- Less interpretable compared to decision trees
- A black box model
- Require large amounts of data, often need data augmentation
- Good at dealing with natural variance in data

Convolutional neural networks work especially well for image data




ImageNet architecture from Alex Krizhevsky, et. al (*ImageNet Classification with Deep Convolutional Neural Networks*), 2012

# Images on a computer

- Discrete representation onto finite grid and resolution
- Channels (depth) expands representation
- Bitwidth and datatype in relation to dynamic range
- Contrast reflected by choice of colormap (linear vs non-linear)
- Typically we normalize values to be in range [0,1]

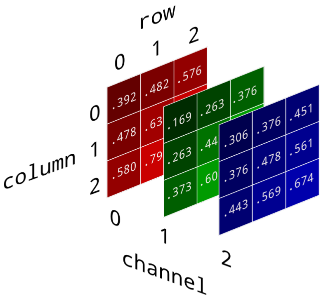



Image from `brohrer.github.io`

# Images on a computer

The convolution operation, a regional approach

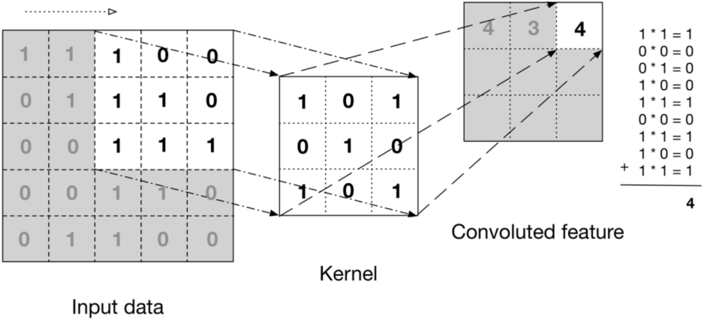
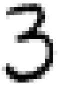



Image from Josh Patterson and Adam Gibson, (*Deep Learning, A Practitioner's Approach*), 2017

# Images on a computer

Three examples of convolution kernels on [28,28] px input image

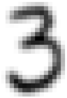
- original image



- $dx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

- $dy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

- $G = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$



# Images on a computer

Properties of convolutions

- Used to extract or manipulate image features
- A linear operator - but what about non-linear kernels?
- Input/output generalizes to N-dimensions and multiple channels
- Stride and padding
- Global vs local context in feature maps
- Convolution operator commutes with translation $\rightarrow$ translational invariance

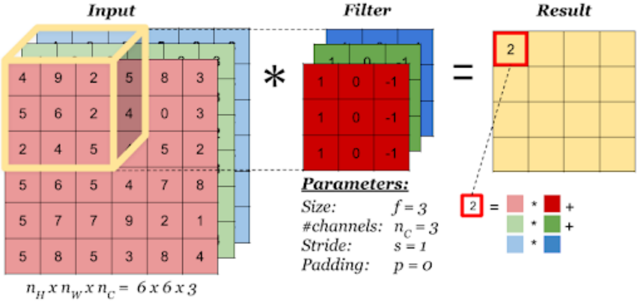



Image from `indoml.com`

# CNN architecture and building blocks

- Layers can roughly be divided into having one of two purposes: Feature extraction or pattern recognition
- One layer typically includes convolution followed by pooling
- Typically increase the number of feature maps, while decreasing kernel size
- Networks with few layers and/or feature maps are called shallow networks
- Shallow/Deep depends on data. We need to make sure that model capacity is sufficient!

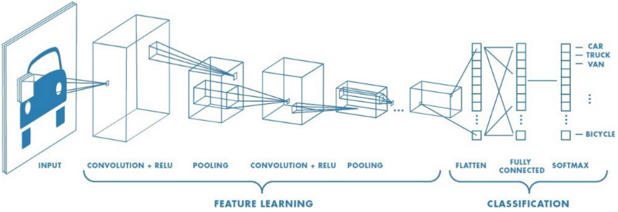



Image from (*Introducing Deep Learning with MATLAB*), eBook

# CNN architecture and building blocks

Why do we need convolution?

- CNNs contain many parameters, even for modest input sizes
- Convolution provides a tool for describing images in terms of individual features
- The output of each operation produces a *feature map*
- Captures image context
- Kernel values = weights, these are the trainable parameters
- Each filter has a unique scalar associated to it - the bias term
- Common filter sizes are about 3,5,7 - What happens if too large? Too small? (1x1)
- Weight sharing, more than one gradient can affect values
- General features $\rightarrow$ specific features the deeper we go

# Interactive questions!

- Q1: Given an input image of **W=[28,28]** pixels, we perform a convolution using a kernel size of **F=5**, a padding of **P=1** and a stride of **S=1**. What will the dimenions of the output become?

- Q2: We look at a single convolution layer in a CNN. We feed a RGB image with **C=3** channels as input. The layer contains **N=8** kernels each of size **K=5**. What is the total number of trainable parameters for this given layer?

# Interactive questions!

- A1:

$$\left(\frac{W-F+2P}{S}\right)+1 \quad = \quad \left(\frac{28-5+2\cdot 1}{1}\right)+1 \quad = \quad 26$$

where:
$W \rightarrow$ input size, $F \rightarrow$ kernel size, $P \rightarrow$ padding size, $S \rightarrow$ stride size

- A2:

$$K^2\cdot C\cdot N+B \quad = \quad 5^2\cdot 3\cdot 8+8 \quad = \quad 608$$

where:
$K \rightarrow$ kernel size, $C \rightarrow$ # of input channels,
$N = B \rightarrow$ # kernels/biases (output channels)

# CNN architecture and building blocks

## Pooling layer

- Reduces dimensions and localization accuracy
- Makes feature maps more manageable
- No trainable parameters
- Pooling operates on each activation map individually
- When downsampling, are we loosing information?
- Remember: Values represent the accumulation of all previous processes
- Pooling enhances what is prominent

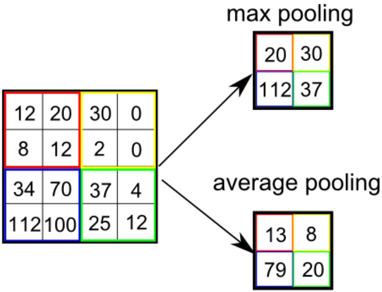
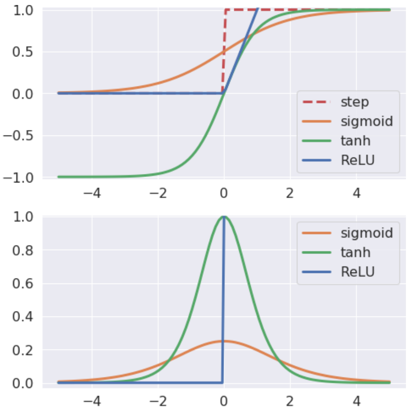



Image from `towardsdatascience.com`

# CNN architecture and building blocks

## Activations

- Recall: Convolution is linear $\rightarrow$ only linear data mappings
- Each feature map is "activated" by non-linearity activation function
- Monotonicity is not necessary but can help speed up optimization
- ReLU is most used, low computational cost
- Often choose functions which are cheaply differentiable (BP)
- Saturated vanishing gradient problem




Top image shows common activations with their derivatives below.

# CNN architecture and building blocks

Dense layers - almost a ”regular” feed forward ANN

- Fully connected layers
- Number of trainable parameters increase drastically
- Randomized dropout to avoid regional dominance
- Typically used towards end of network - and no shared parameters
- Responsible for pattern recognition and classification
- Dense layers are problem-specific and **not** necessary in a CNN

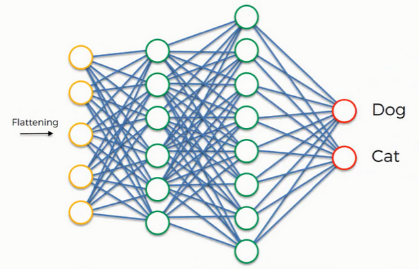



Image from *Fundamental Concepts of Convolutional Neural Network*, 2019

# CNN architecture and building blocks

How to choose architecture and what to consider?

- Well there is **no** one-fits-all approach ...
- The capacity of the network is a measure for what it can learn
- We control capacity through many (hyper)parameters
- How does the initialized parameters change, does it make sense?
- How many epochs are necessary? Early stopping?
- Often work in mini-batches when possible
- Pruning can tremendously increase performance
- Try things out: One change at one place can make other changes elsewhere redundant

# Training process and inference

While training is typically slow, inference is almost instantaneous.

Some data processing is necessary to make BP usable.

$$I_{\text{normalized}} = \frac{I - min(I)}{max(I) - min(I)} \qquad I_{\text{standardized}} = \frac{I - \mu_{\text{I}}}{\sigma_{\text{I}}}$$

This ensures small perturbations in weights/bias also yield small changes in output.

- Start with small portion of data - should be easy to overfit
- Monitor various parameters during training
- Then make adaptations accordingly: From coarse to fine tuning
- Is learning rate too low?
- Sanity check: Is loss behaving as expected?
- Try making the model intentionally worse - does it behave reasonably?
- Random search vs grid search
- Can initialize from trained network (transfer learning)

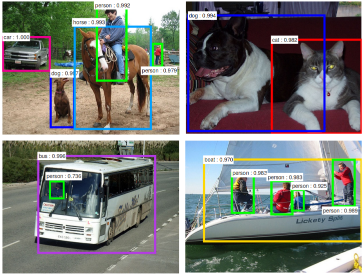
# Implementation in Python

- Very parallelizable process through vectorization $\rightarrow$ GPU
- Processing images in mini-batches not single image at a time
- Data-augmentation when necessary: Rotations, scaling, stretching, flipping, add noise, change lighting, etc.
- Need to be careful when data-augmenting (MNIST: 6 vs 9)
- Most popular: Tensorflow (Google), PyTorch (Facebook)
- Personal choice of high vs low level approach in both frameworks

# Implementation in Python

Live demo in Jupyter Notebook

# Examples and perspectivation




Left: Object detection using Faster RCNN (Shaoqing Ren et. al 2016). Right: Image segmentation using U-net (Olaf Ronneberger et. al 2015).

Interactive 3D visualization of CNN on MNIST data:
`https://www.cs.ryerson.ca/~aharley/vis/conv/`

More advanced interactive CNN visualization system:
`https://poloclub.github.io/cnn-explainer/`