Neural Networks 2 - Convolutional Neural Networks

18NES2 - Week 6, Winter semester 2025/26

Zuzana Petříčková

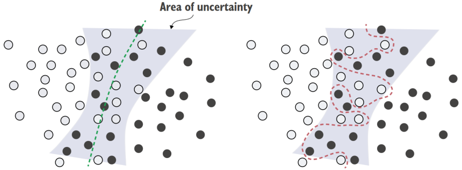
November 4, 2025

Neural Networks 2 - Convolutional Neural Networks

1. Review
2. Motivating Example: Bird Species Classification
3. Convolution Operation
4. Convolutional Neural Network
   - Classic CNN Architecture
5. Training a CNN Model from Scratch
   - CNNS and Regularization
6. Graded Homework

What We Covered Last Time

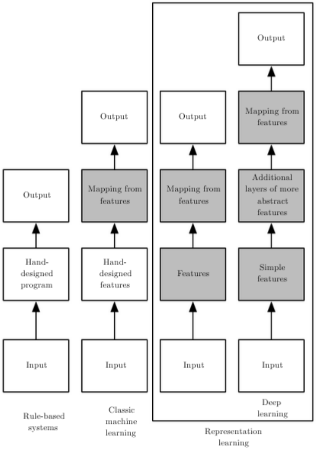
- **Generalization in Neural Networks**
  - About generalization, underfitting and overfitting
  - How to measure generalization
  - Techniques to improve generalization
    - Early stopping, Regularization techniques, Dropout, Normalization, Ensemble models,...
  - Practical examples



F. Chollet: Deep Learning with Python, Fig. 5.5

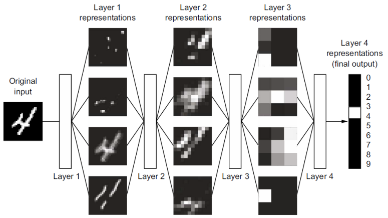
- **3rd homework assignment**
- **Introduction to Convolutional Neural Networks**

Recap: Deep Learning



I. Goodfellow, Y. Bengio, A. Courville: Deep Learning, 2016, Figure 1.5

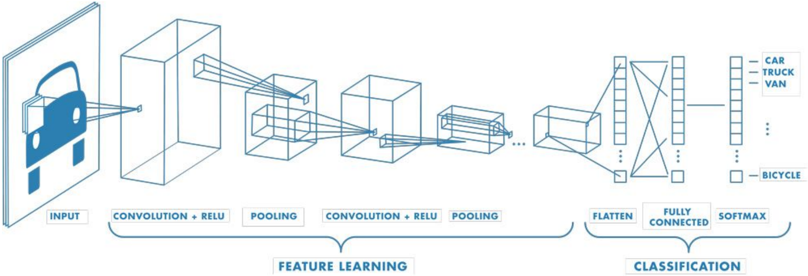
- Utilizes artificial neural networks with many layers (so-called deep networks)
- Models automatically learn to extract features from data – less manual preprocessing
- Architecture is often tailored to the specific data type (image, text, audio, ...)



F. Chollet: Deep Learning with Python, Fig. 1.6

Convolutional Neural Network

- A specialized type of deep neural network for processing image data
- Efficient feature extraction using convolutional layers (filters)

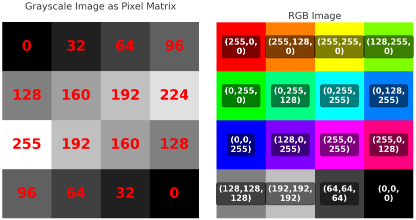


Source:

https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning

Reminder: Digital Image Representation

- A digital image is a matrix (tensor) of pixels.
- Each pixel (short for "picture element") describes the color at a specific position in the image.



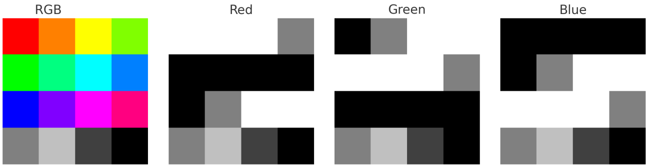
Grayscale Image

- Each pixel is a single value indicating brightness (e.g., $0 =$ black, $255 =$ white).
- For machine learning, pixel values are usually normalized to the interval $[0, 1]$.

Reminder: Digital Image Representation

Color Image (RGB)

- Each pixel consists of three components: R (red), G (green), B (blue).
- The image is represented as a 3D tensor of shape (height $\times$ width $\times$ 3).
- These components are called color channels.

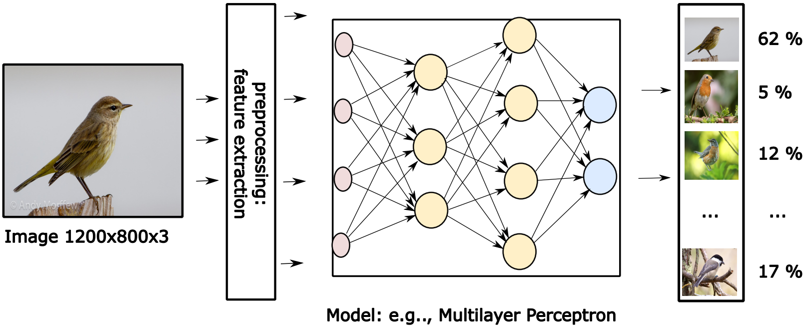


Example: convolution_introduction.ipynb

Motivating Example: Image Classification
Bird Species Recognition

Motivating Example: Bird Species Recognition

Classical Machine Learning Approach

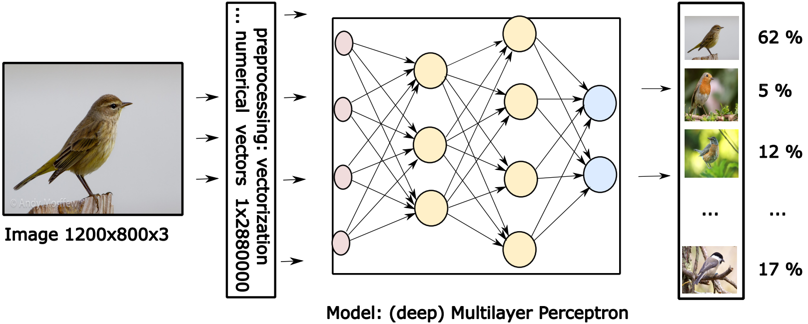


Thorough preprocessing of the data: Feature Extraction

- edge detection, LBP histograms, etc.
- information loss; requires careful feature design

Motivating Example: Bird Species Recognition

What if we train a neural network directly on the data?

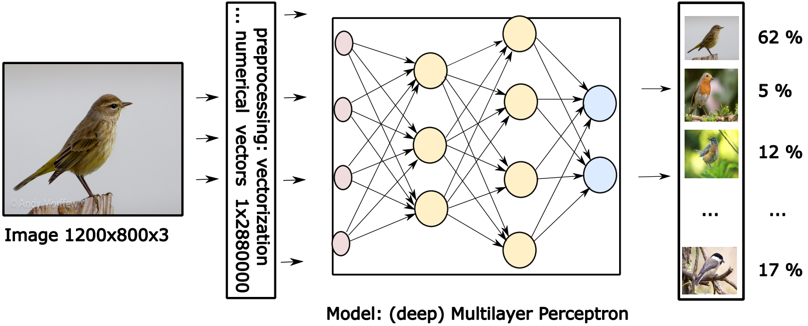


Deep Learning Principle

- let the model learn useful features from the data
- preprocess the data just slightly (e.g., vectorization, normalization)

Motivating Example: Bird Species Recognition

What if we train a neural network directly on the data?

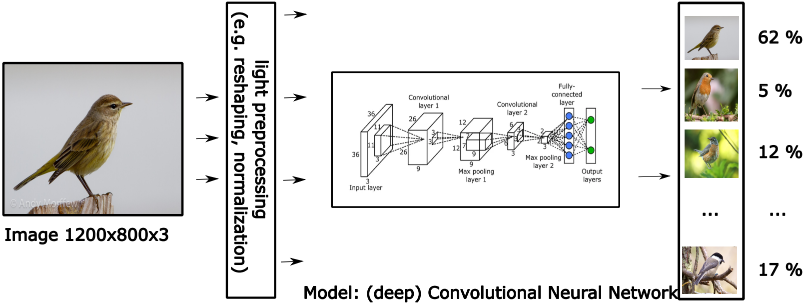


Drawbacks of the MLP approach (fully connected layers only):

- high number of parameters
- loss of spatial relationships between pixels
- it is difficult to train the model effectively

Motivating Example: Bird Species Recognition

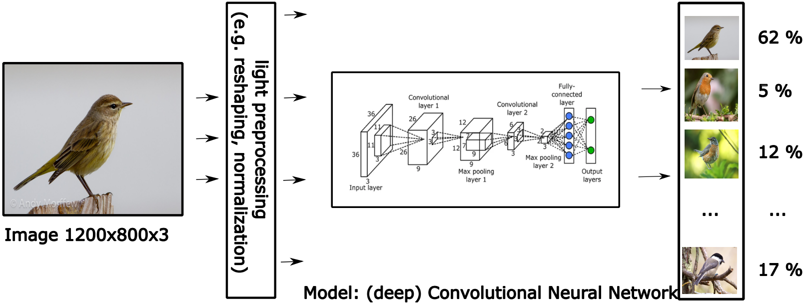
How could we improve this?



Convolutional Neural Network (CNN):

- A specialized type of neural network for processing image data
- It takes spatial arrangement of pixels into account
- Efficient feature extraction using convolutional layers (filters)

Motivating Example: Bird Species Recognition

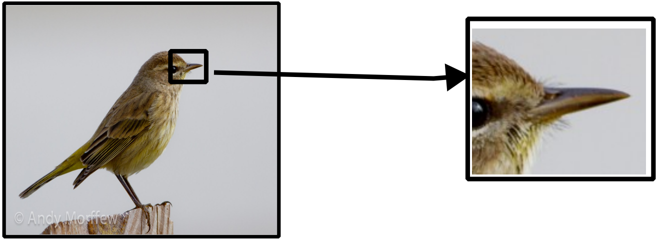


Advantages of Convolutional Networks

- Easier to train compared to fully connected networks, fewer parameters
- Preserve spatial relationships and local patterns in pixels
- Better scalability to large input images, robustness to translation and scale variations of objects

Motivating Example: Bird Species Recognition

Patterns in data: for example, beak shape



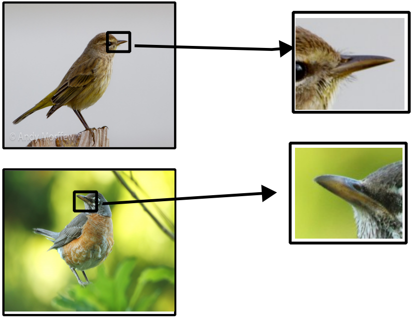
Let's create a beak detector:

- a simple model (e.g., a single-layer neural network) that detects beaks in images

But: the beak may appear in different locations within the image

Motivating Example: Bird Species Recognition

Patterns in data: for example, beak shape

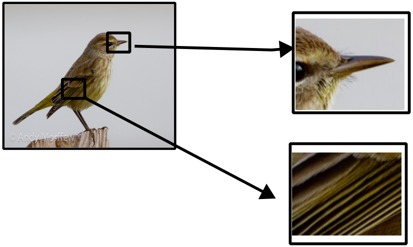


The beak may appear in different image regions

- the detector should find a beak in any image and at any location
  $\rightarrow$ the detector must slide over the input image

Motivating Example: Bird Species Recognition

There are multiple patterns in the data (e.g., beak, feather, eye):



The idea:

- create a set of detectors for different features (patterns)
- detectors should recognize features anywhere in the image
  $\rightarrow$ detectors slide over the image
- these detectors form the initial layers of a convolutional neural network

Convolutional Neural Network

- A neural network that includes convolutional layers

Convolutional Layer

- Consists of a set of filters (also called kernels or detectors)
- Each filter performs a convolution operation over the input image
- The output of the convolution (a feature map) is passed to the next layer

Convolution Operation

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Image 6x6 (black and white)

Convolutional Layer:

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1 (3x3)

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2 (3x3)

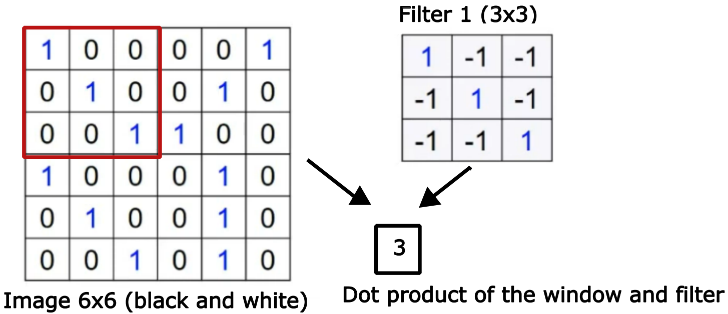
. . .

- The convolutional layer contains several filters
- Each filter detects a pattern (feature) of size 3 $\times$ 3 pixels (e.g., diagonal edge, vertical edge, etc.)

Example source: Petr Doležel – Convolutional Neural Network,

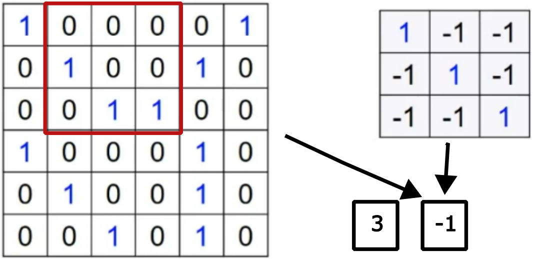
https://www.youtube.com/watch?v=-2vEi-Aa0FA

Convolution Operation



- We compute the dot product:
  $y = \sum_{i=1}^{9} w_i x_i + b$ (for flattened matrices)

Convolution Operation



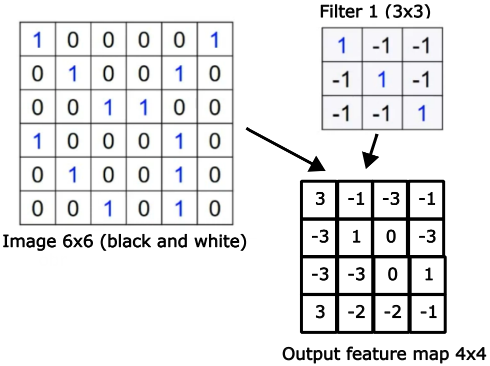
Filter 1 (3x3)

Image 6x6 (black and white)

Dot product of the window and filter

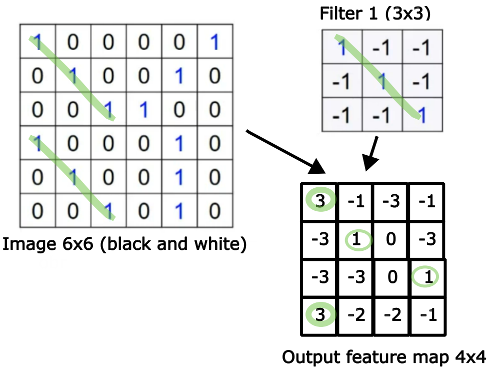
- Move the sliding window and compute another dot product

Convolution Operation



- By sliding the window over the image, we apply the filter to the entire input
- The result is a new 2×2 tensor – a **feature map**

Convolution Operation



Feature Map

- Indicates where (and how strongly) the pattern represented by the filter appears in the input image
- Example: diagonal edge filter

Convolution Operation

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Image 6x6 (black and white)

Filter 2 (3x3)

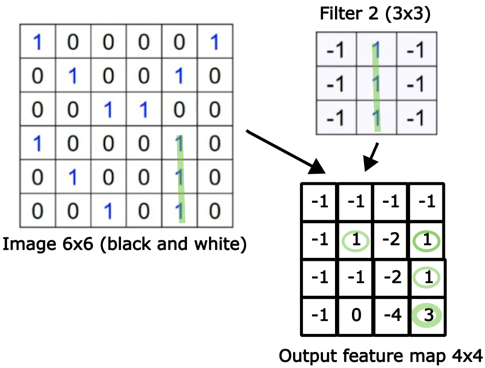
-1	1	-1
-1	1	-1
-1	1	-1

-1	-1	-1	-1
-1	1	-2	1
-1	-1	-2	1
-1	0	-4	3

Output feature map 4x4

- We can similarly apply a second filter

Convolution Operation

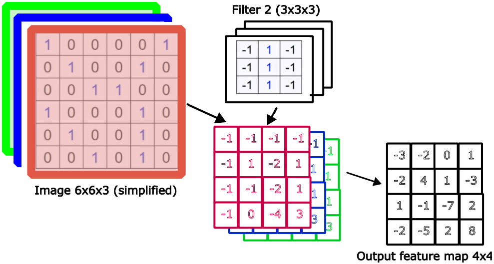


Second Feature Map

- Indicates where (and how strongly) the pattern represented by the second filter appears in the input image
- Example: vertical edge filter

Convolution Operation

Color Image: 3 input channels – R, G, B



- Each filter has weights for **all input channels (R, G, B)**
- Computation: convolution is performed separately on each channel and the results are **summed**
- Each filter produces **one aggregated** output feature map
- The number of filters defines the **number of output channels**

Convolution Operation

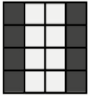
Example: Zebra

Source:

https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning

Convolution Operation

Example: Zebra

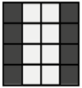


-1	1	1	-1
-1	1	1	-1
-1	1	1	-1
-1	1	1	-1

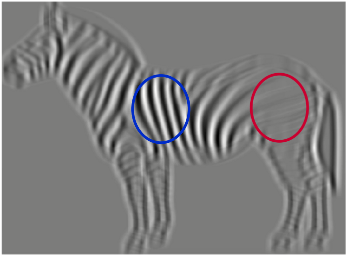
- Output after applying the vertical stripe filter

Convolution Operation

Example: Zebra



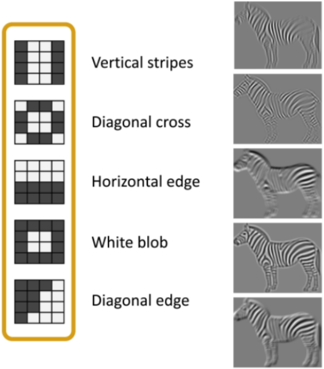
-1	1	1	-1
-1	1	1	-1
-1	1	1	-1
-1	1	1	-1



- Clearly shows where the pattern is strongly present and where it is not

Convolution Operation

Example: Zebra



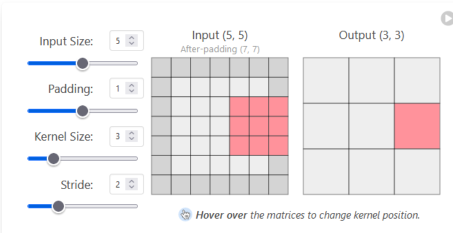
- Examples of other filters and their resulting feature maps

Convolution Operation

Convolution Operation Parameters

- Dimensions of the input image
- Padding – how borders are handled
- Filter size
- Stride – the step used to move the filter across the image

Understanding Hyperparameters



Great interactive visualization:
https://poloclub.github.io/cnn-explainer/

Convolution Parameters: Padding and Stride

Padding

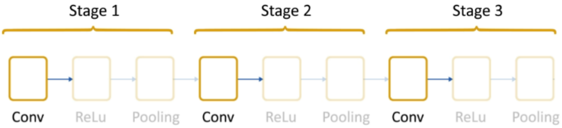
- Adds extra border pixels around the input image
- Common options:
  - **Valid** – no padding (output shrinks)
  - **Same** – padding added to keep output size the same as input
- Helps preserve spatial size and improve edge detection

Stride

- Controls how far the filter moves at each step
- **Stride = 1**: typical setting, dense coverage
- **Stride > 1**: reduces output size, performs downsampling

Tip: Try out different values in CNN Explainer

Classic CNN Architecture



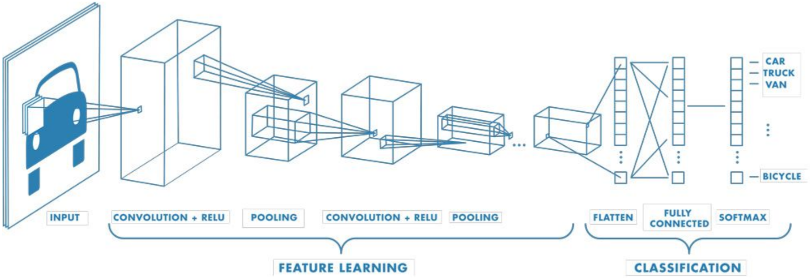
Core idea: stack convolutional layers (or blocks) on top of each other

- The first convolutional layer detects simple features (e.g., edges, blobs)
- Each following layer extracts higher-level features

Hierarchical structure of features:

edges → shapes → object parts → whole objects

Classic CNN Architecture



Main components of a CNN:

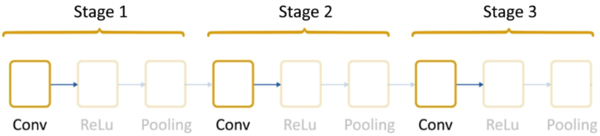
- **CNN base** - Convolutional blocks for feature extraction
- Flattening layer – converts the feature maps into a 1D vector
- **Head** - Fully connected neural network for classification

Image source: `https://`
`//matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning`

Classic CNN Architecture

Typical structure of a convolutional block:

- Convolutional layer
- Nonlinear activation function (e.g., ReLU)
- Pooling layer



Pooling (Subsampling) Layer

- Reduces spatial resolution while preserving most of the relevant information
- A sliding window (e.g., 2×2) moves across the feature map, often with stride $= 2$
- Common operations: MAX (max pooling), AVERAGE (average pooling); no weights involved

Why pooling?

- Condenses the information stored in the feature map
- Keeps track of where and how strongly a feature occurs
- Reduces the data size (e.g., $2 \times 2 \rightarrow 1$ value $= 75\%$ reduction)

Convolutional Block – Example: Convolutional Layer

vstupní obrázek (6x6x3)

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Filtr 1 (3x3)

1	-1	-1
-1	1	-1
-1	-1	1

Filtr 2 (3x3)

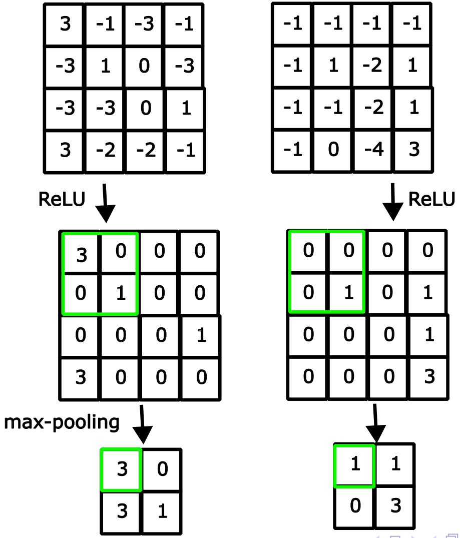
-1	1	-1
-1	1	-1
-1	1	-1

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

-1	-1	-1	-1
-1	1	-2	1
-1	-1	-2	1
-1	0	-4	3

feature maps (4x4xpočet))

Convolutional Block – Example: Pooling Layer



Convolutional Block

Why not just stack convolutional layers without pooling?

- The number of parameters grows with each added layer
- Image size stays (almost) the same, especially with "same" padding
  $\Rightarrow$ the size of the feature maps (and computation) keeps growing

Pooling layer:

- Reduces data size while preserving information about feature presence and strength
- e.g., $2 \times 2 \rightarrow 1$ value = quarter size

Alternating convolution and pooling – bipyramidal effect:

- Spatial size decreases, number of feature maps increases

Bipyramidal Architecture

- One of the oldest architecture types: wide and shallow, with a deeper fully connected part – close to the basic layered schema

Typical structure of a bipyramidal architecture:

- The number of filters typically doubles in deeper layers (e.g., 32, 64, 128, . . .)
- Most commonly used filter size: 3×3
- ReLu activation
- Max-pooling 2×2 is often paired with filter doubling
- When using several convolutional layers, we may not need many fully connected layers
- (Optionally) One or more fully connected layers are added for classification

CNN_fashion_mnist.ipynb

- Practical example: Fashion MNIST dataset

Training a Convolutional Neural Network

- Typically trained using a variant of backpropagation (e.g., SGD, Adam)
- A high number of trainable parameters
  $\rightarrow$
  - The model requires a large amount of training data
  - Mini-batch learning $\rightarrow$
- Mini-batch learning – the model requires a large amount of data

How to choose a suitable architecture in practice?

- We usually don't optimize the number of layers or neurons manually
- We pick a proven topology from the literature for the given type of task:
  - Bipyramidal architecture
  - One of the modern architectures (e.g., https://keras.io/api/applications/)

Examples

CNN_fashion_mnist.ipynb

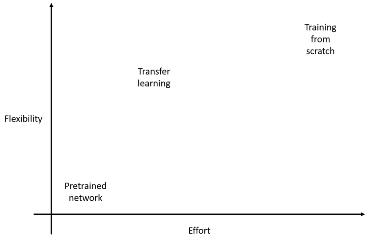
- Practical example: Fashion MNIST dataset

Useful links:

- Interactive CNN visualization: https://poloclub.github.io/cnn-explainer/
- MathWorks – activation visualization (face)

Ways to Build and Train a Convolutional Neural Network

- Training from scratch
- Using a pretrained model
- Transfer learning
- Fine-tuning a pretrained model



Source:

https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning

Example: Training a Model from Scratch on a Small Dataset (Cats vs Dogs)

cats_dogs_from_scratch.ipynb

- Dataset: **Kaggle Cats vs Dogs** (25000 images, 500 MB compressed)
- Two classes: **Cat** and **Dog**
- We use a smaller subset of 4000 images (balanced)
- Suitable for demonstrating CNN from scratch, overfitting on small data, and regularization/augmentation

Example of a CNN Trained from Scratch on a Small Dataset: Cats vs Dogs

Task:

- Build a simple bipyramidal CNN from scratch
- Train on the small dataset split; monitor training and validation curves

In this example, we demonstrate several practical techniques:

- Image preprocessing
- Efficient data loading using data loaders
- Visualization of filters and feature maps

cats_dogs_from_scratch.ipynb

Example of a CNN Trained from Scratch on a Small Dataset: Cats vs Dogs

Observations

- **Baseline (no regularization):** test accuracy around **70%**; the model **overfits quickly** (validation accuracy peaks early; validation loss rises after a few epochs).
- Compared to Fashion-MNIST, the model needs more inductive bias and variability in the data (natural images, backgrounds, poses).
- Saliency often highlights facial regions (eyes, muzzle), but can be distracted by background clutter.

How to improve generalization?

Convolutional Neural Networks and Generalization

- Compared to MLPs, CNNs typically learn more slowly, especially on CPUs
- Convolutional neural networks tend to suffer more from overfitting
- Overfitting is a major issue when only a small dataset is available (hundreds or a few thousand samples)
- How can generalization be improved?
  - **Standard regularization**: early stopping, dropout, normalization (for deep models)
  - **Data augmentation**: expanding the dataset *on the fly*
  - **Transfer learning**

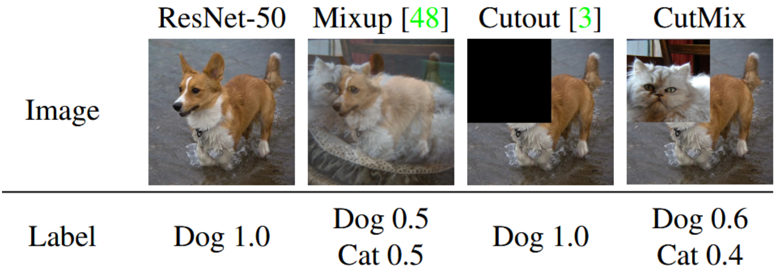
Data Augmentation

- Various (random) image transformations: rotation, shift, flip, skewing, resolution change, brightness and contrast adjustment, cropping, adding noise, blur, combinations
- In Keras, implemented via a dedicated layer

Source:

https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning

Data Augmentation – Popular Variants



Source: Yun et al., CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features, https://arxiv.org/pdf/1905.04899

Data Augmentation

Advantages:

- Artificially increases the size and diversity of the training dataset
- Helps prevent overfitting by exposing the model to a wider range of input variations
- Improves generalization to unseen data and robustness to distortions

Implementation in Keras:

- Easy to use via layers such as `RandomFlip`, `RandomRotation`, `RandomZoom`, etc.
- Augmentation happens **on the fly during training**, saving memory

CNNs and Generalization

Common Regularization Techniques

- **Early stopping**
- **L1/L2 regularization** – typically used with ReLU units in convolutional and fully connected layers
- **Dropout** – adding a special dropout layer after each fully connected layer (in the classification head)
- **Normalization** of inputs, weights, and layer outputs; a popular technique is Batch Normalization
- **Label smoothing** – adding noise to labels
- **Ensembling**

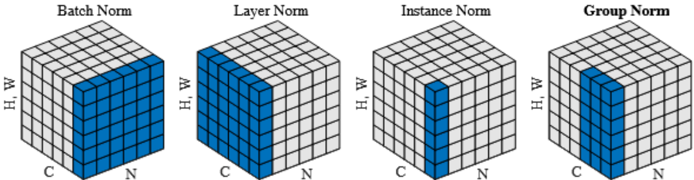
Especially relevant for CNNs:

- **Data augmentation**
- **Transfer learning**

CNN and Regularization

Normalization of Layer Inputs

- Aims to fix the mean and variance of each layer's output
- Helps address the vanishing gradients problem
- Implemented by adding an additional layer (e.g., after a convolutional layer)
- Results in faster training and reduced sensitivity to weight initialization
- Increases robustness to noise in the data (can replace Dropout for deep models)
- Different normalization variants exist:



Example of a CNN Trained from Scratch on a Small Dataset: Cats vs Dogs

cats_dogs_from_scratch.ipynb

- The model without regularization generalized very poorly.

With augmentation and regularization

- **Accuracy improves from about 0.65-0.7 to about 0.8-0.85 on the test set.**
- **Overfitting is delayed and weaker**: training and validation curves stay closer for longer, the validation peak occurs later.

Next Step

- How about using a pretrained model or transfer learning?

4th Graded Homework: CNN on a CIFAR-10 Subset

Goal: Train a small CNN on a subset of the **CIFAR-10** dataset and experiment with different techniques to improve generalization.

Dataset:

- Use a smaller subset of the data (e.g., 2000 training, 1000 validation, 1000 test samples).
- You can select 3–5 classes (e.g., *airplane*, *car*, *bird*, *cat*, *dog*) or use all 10 classes if resources allow.
- Each image is 32×32 px, RGB.
- The dataset can be easily loaded using `tf.keras.datasets.cifar10`.

4th Graded Homework: CNN on a CIFAR-10 Subset

Suggested workflow:

- Start with a small bipyramidal CNN and train it from scratch.
- Observe and describe signs of overfitting.
- Add data augmentation and regularization, then compare results.

You can start from the example notebooks shown in the lab:
fashion_mnist.ipynb cats_dogs_from_scratch.ipynb

4th Graded Homework: Requirements

Pipeline: load → preprocess → build → train → evaluate

Include in your notebook:

- Training and validation curves (loss + accuracy).
- Final test accuracy and a confusion matrix.
- Comparison of:
  - baseline CNN (no regularization),
  - CNN with data augmentation,
  - CNN with augmentation + Dropout / BatchNorm / other regularization.
- Short conclusion: what worked best and why.

Optional:

- Visualize misclassified images.
- Try reducing or increasing the number of classes and observe the effect.

4th Graded Homework: CNN on a CIFAR-10 Subset

Submission

- Submit the notebook by **Nov 10, 2025**.
- **Consultation required by Nov 14, 2025** to receive points (short discussion after lab or individually).
- **Points: 2**