Neural Networks in Machine Vision

CTU, FS, U12110

Ing. Matouš Cejnek, Ph.D.

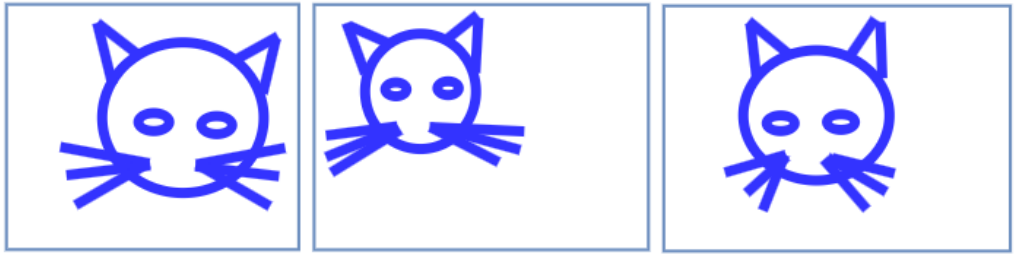
Motivational Example

How to Recognize a Cat-like Object in an Image?



Search for the Cat-like Features!

Why it is not easy?



- They are on different location
- They have different scale and rotation
- They actually look very different
- The colors and lighting are different

At least the features are somehow organized in cat-like patterns!

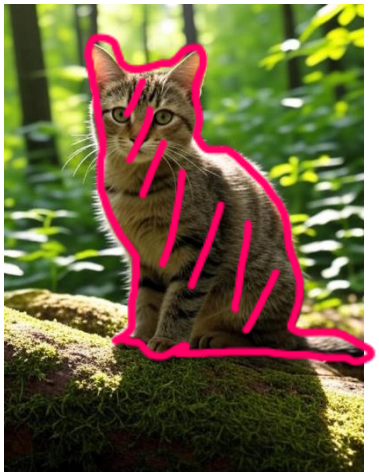
Neural Networks Review and Problem Introduction

Tasks Suitable for Deep Learning

Classification
Is this a cat image?

Detection
Where are the cats?

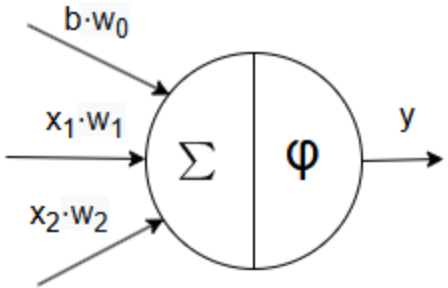
Segmentation
What pixels are cat?



Regression
How many cats?

Single Neural Unit

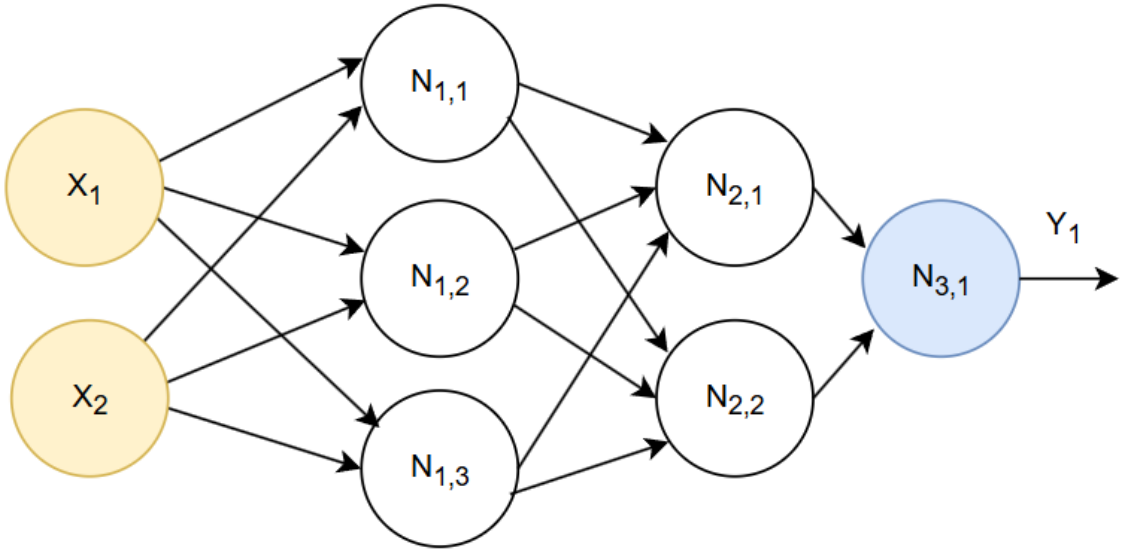
We search for ***W*** values to minimize the error of the output ***y*** :



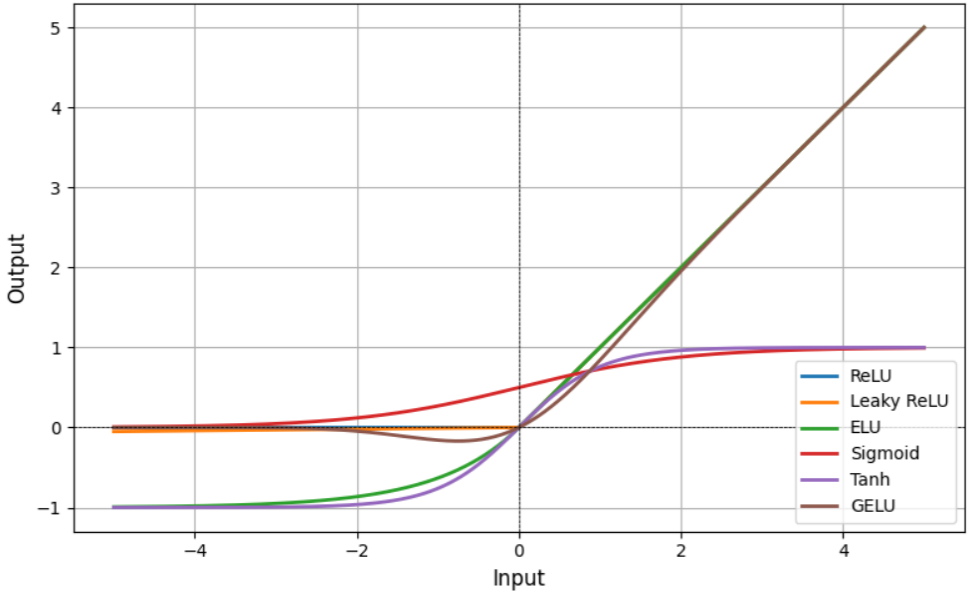
$$y = \varphi\left(\sum_{i=1}^{n} x_i w_i + b w_0\right)$$

Multi-layer Perceptron

Many neurons:



Activation Function

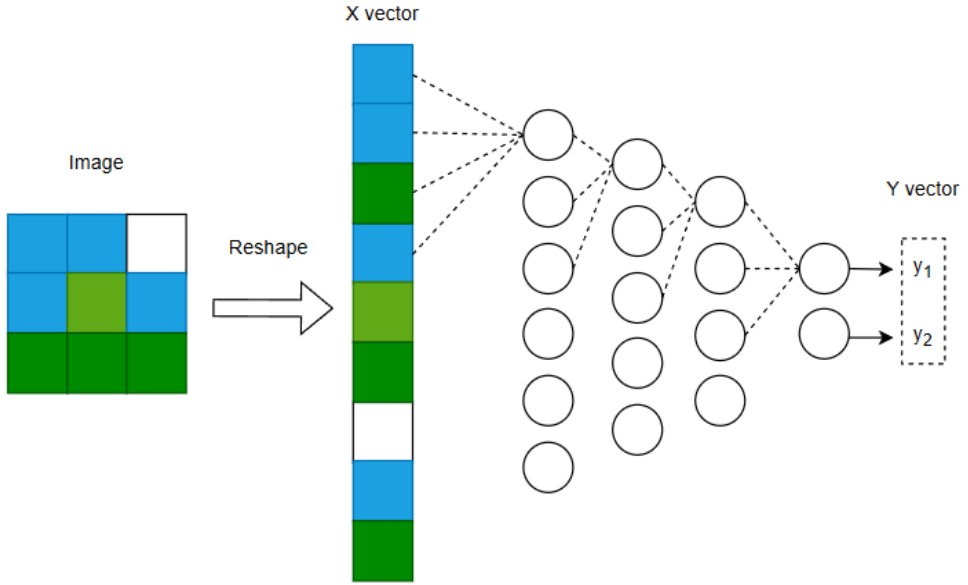


Why we use them:

- To introduce non-linearity, which allows the network to learn complex functions.
- Without them, MLPs (even deep ones) are no better than a single linear layer.

MLP use in Image Processing

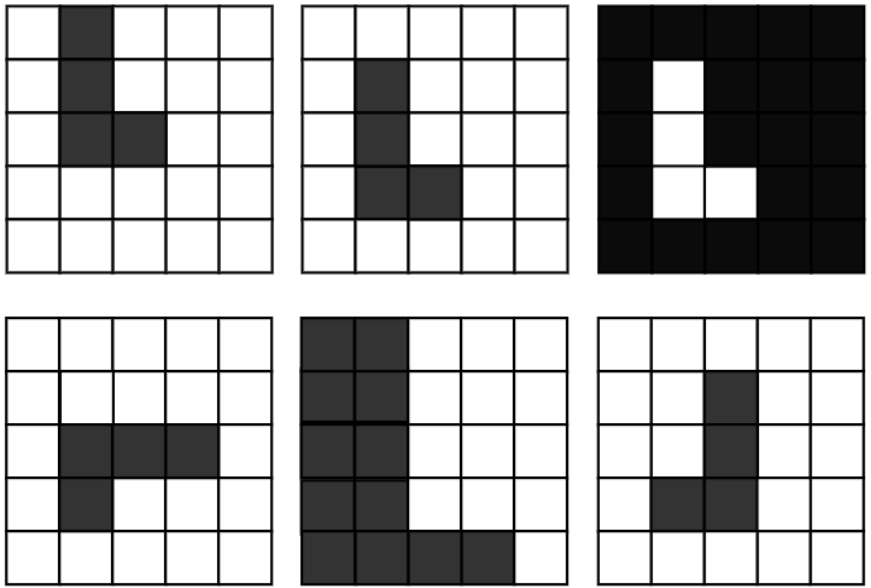
Naive example:



Is this a smart idea?

The Actual Challange

- Object can translate
- Object can rotate
- Object can scale
- Colors can change



Challange Summary

So what we actually need to find in images?

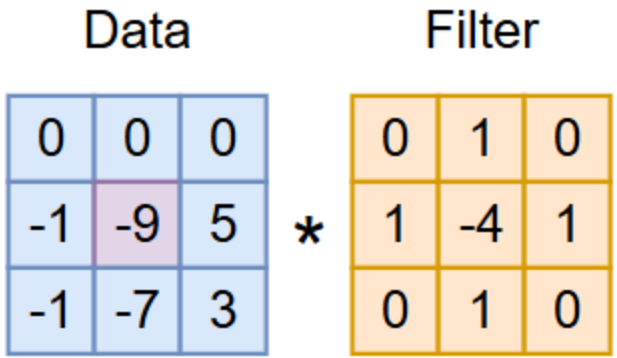
- Patterns
- Patterns of patterns
- Patterns of [Patterns of [Patterns, ...], ...]

How to learn patterns in an effective way?

2D Convolution (Simplified)

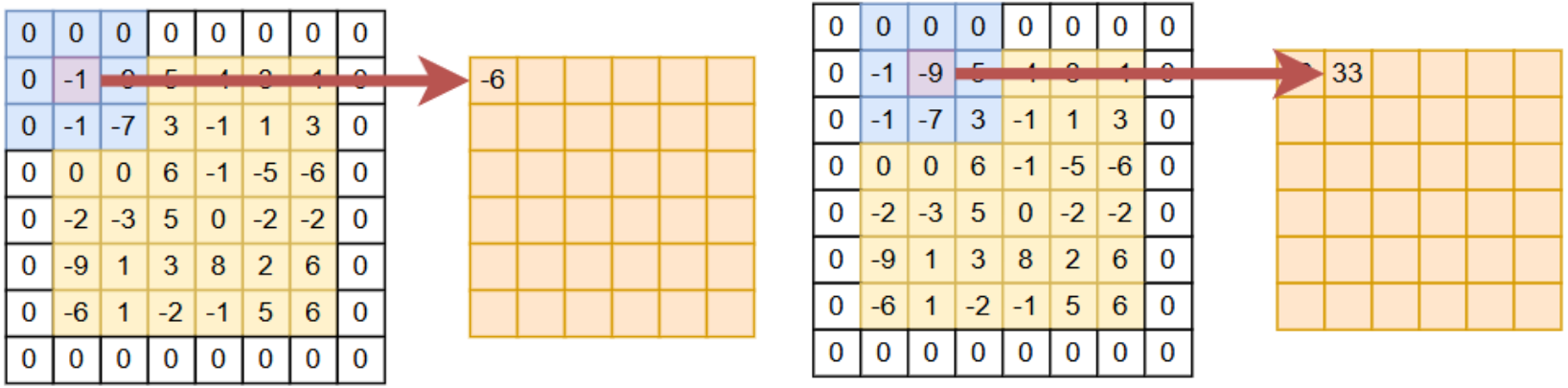
2D Convolution

- Computation for a single point (x,y) - the surrounding is used.
- Kernel (filter) can have various size



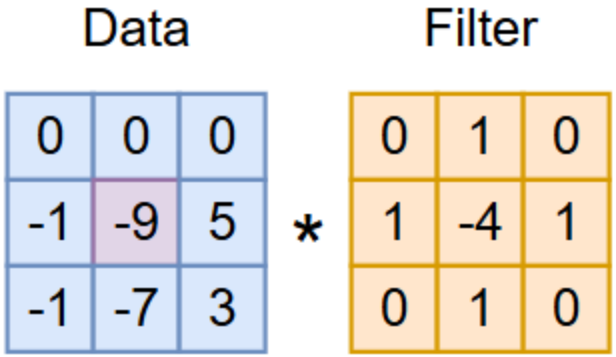
2D Convolution

This process is repeated for all points – sliding window like.



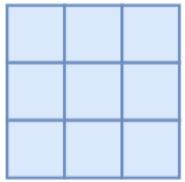
2D Convolution

We already know this:

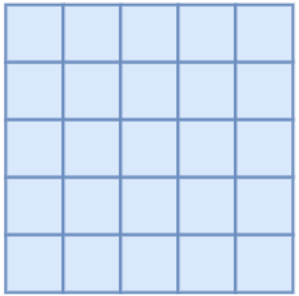


What about **sizes**, **dilations** and **strides**?

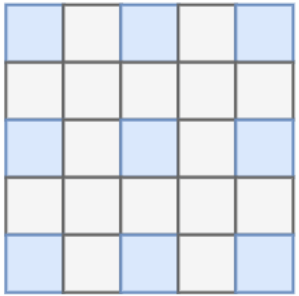
Kernel (Filter) Size



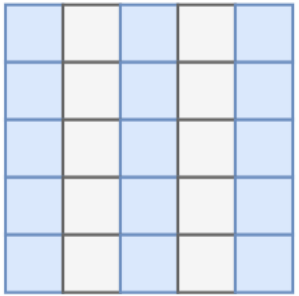
Size: (3,3)
Dilation: (1, 1)
Receptive field: (3x3)



Size: (5,5)
Dilation: (1,1)
Receptive field: (5x5)

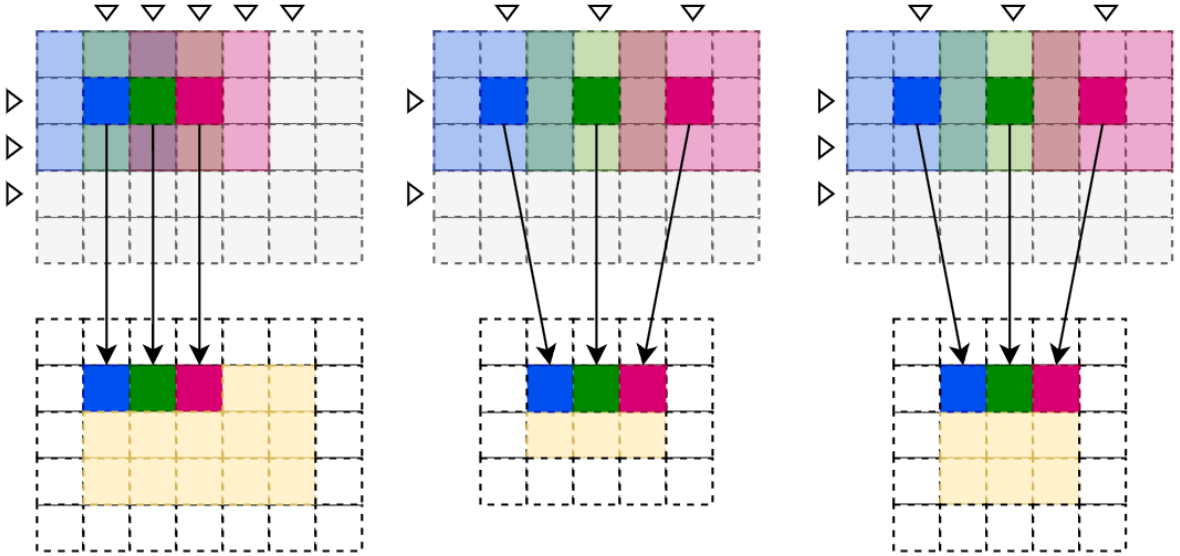


Size: (3,3)
Dilation: (2,2)
Receptive field: (5x5)



Size: (5,3)
Dilation: (1,2)
Receptive field: (5x5)

Convolution Stride



Stride: (1, 1)

Stride: (2, 2)

Stride: (1, 2)

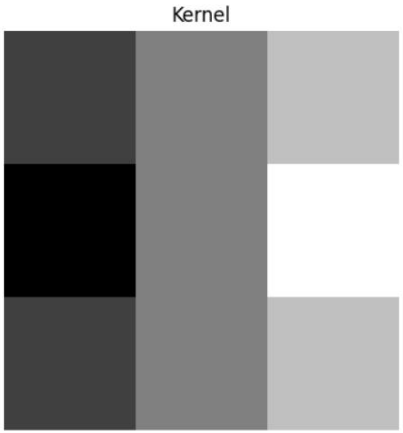
2D Convolution Output Meaning

- It computes a **cross-correlation** between the input and the kernel.
- **Each kernel** is designed to detect a specific **pattern or feature**.
- A **high output value** indicates a strong presence of the kernel's pattern in the input.
- We can adjust the kernel values to detect the desired pattern.

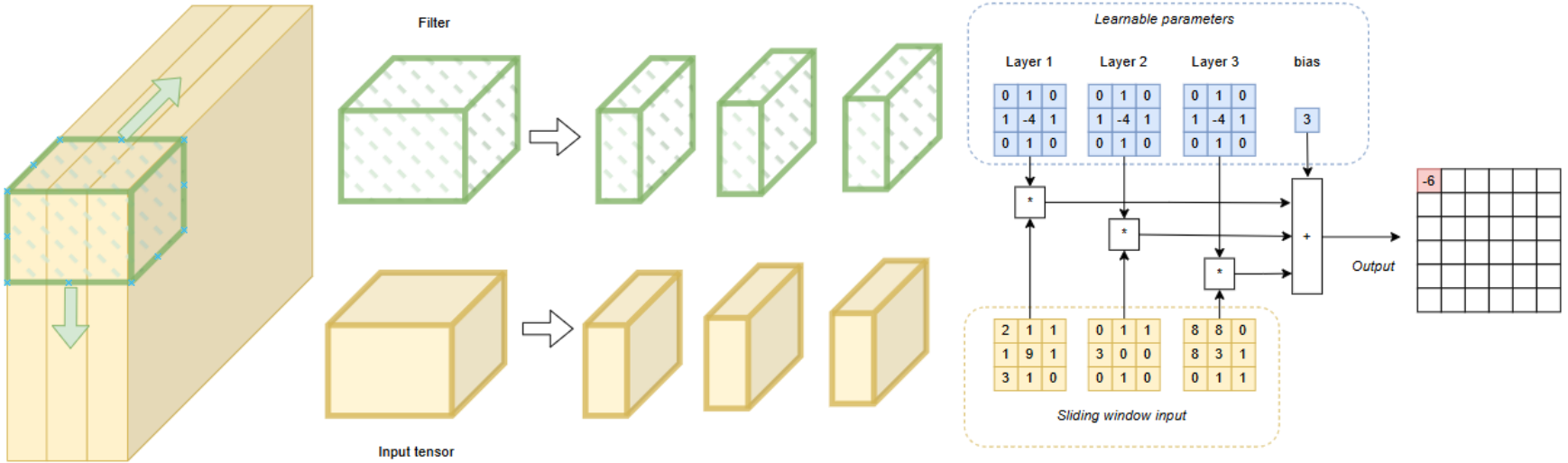
Example Feature Map

Kernel values:

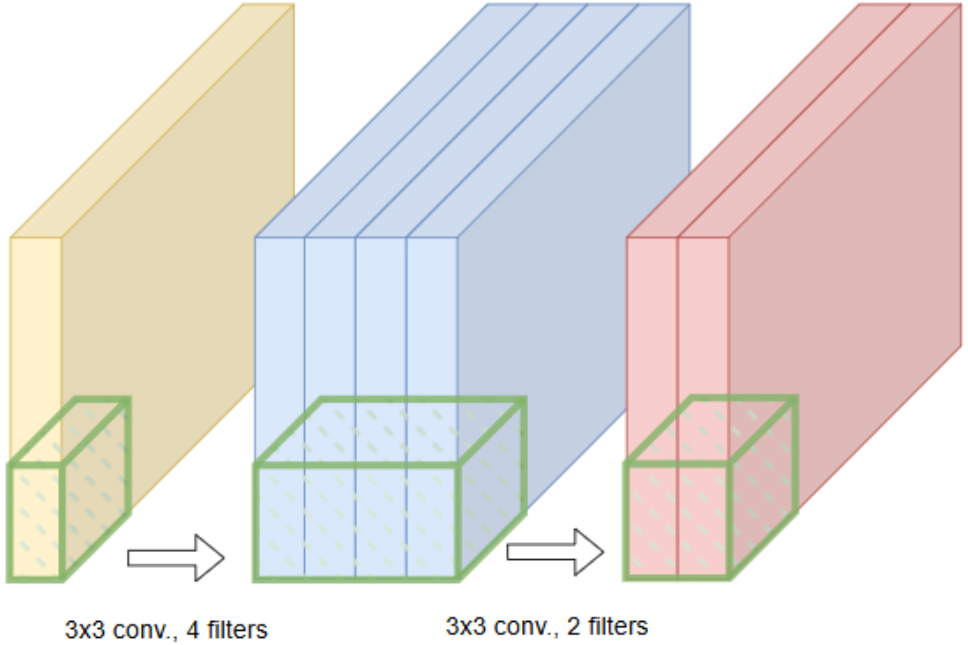
-1, 0, 1
-2, 0, 2
-1, 0, 1



Multi-channel Convolution Simplified

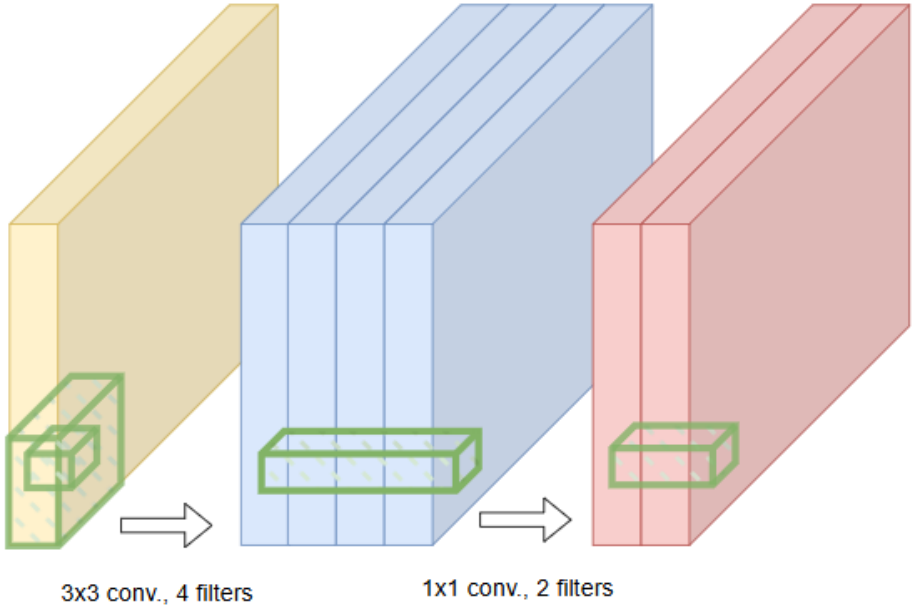


Multiple Channels



- Kernel can have another dimmension (depth)
- Kernel still moves only in ***x*** and ***y*** – 2D convolution

Overview



- We can have filters with various sizes **to catch patterns.**
- We can catch patterns **over stacks of images** (channels).
- We can use **multiple filters to catch multiple patterns**!

Convolution Pipeline Summary

We can nest the convolutions to finally catch complex

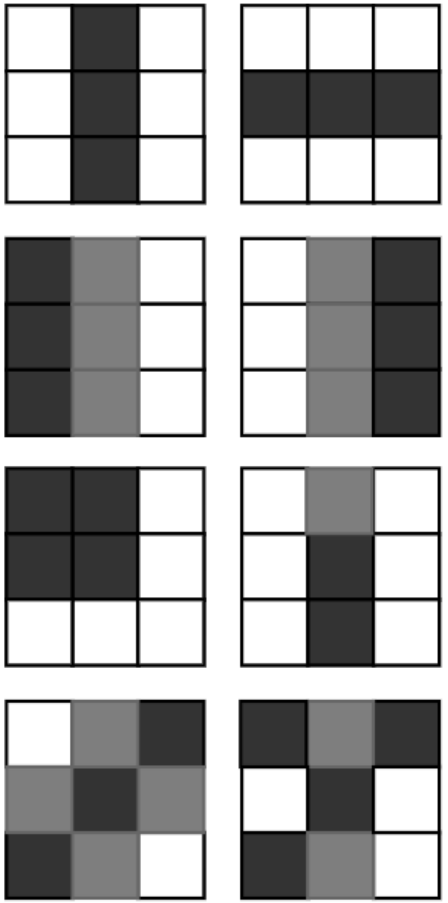
Patterns of [Patterns of [Patterns of, ...], ...]

The ouput of this convolution is called **feature map**.

But what are the actual patterns?

Low-level Patterns (early layers)

- Edges (horizontal, vertical, diagonal)
- Corners & lines
- Color gradients
- Simple textures



Mid-level Patterns (middle layers)

- Contours
- Shapes or parts of objects
- Textures or repeated motifs
- Combinations of edges (e.g. curves, junctions)

High-level Patterns (final layers)

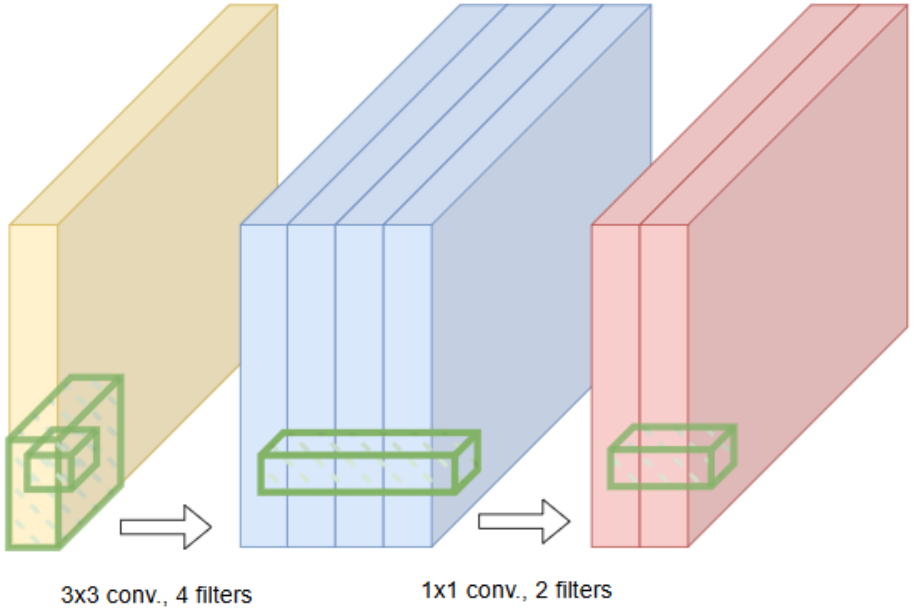
- Full object parts (e.g. wheel, eye, window)
- Whole object configurations (e.g. faces, cars)
- Class-discriminative features (e.g. dog vs. cat characteristics)

Feature Map Summary

- We can catch patterns with **feature maps**.
- We know what the patterns mean.
- **What we can do with them?**

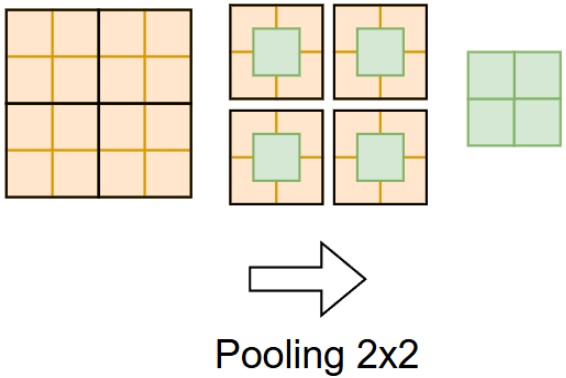
Utilization of Feature Maps

Feature maps



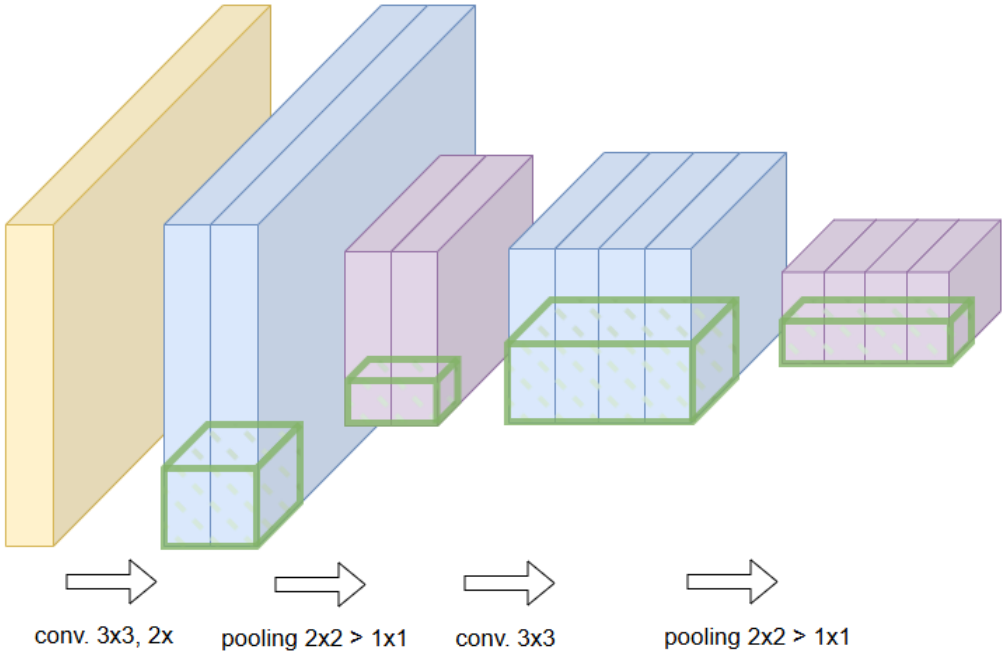
We can generate multiple feature maps and refine them using subsequent 1×1 convolutions to reduce dimensionality or enhance specific features. **But what to do with them?**

Pooling



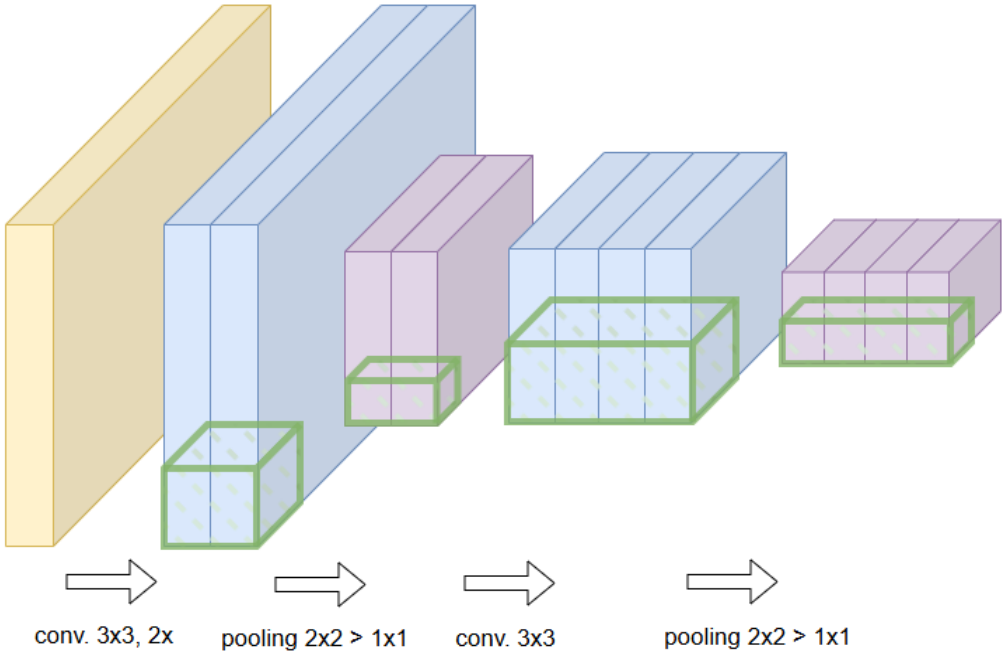
- **We can downsample the feature maps** by replacing regions with their maximum or average value.
- It is commonly known as max pooling or average pooling.

Pooling



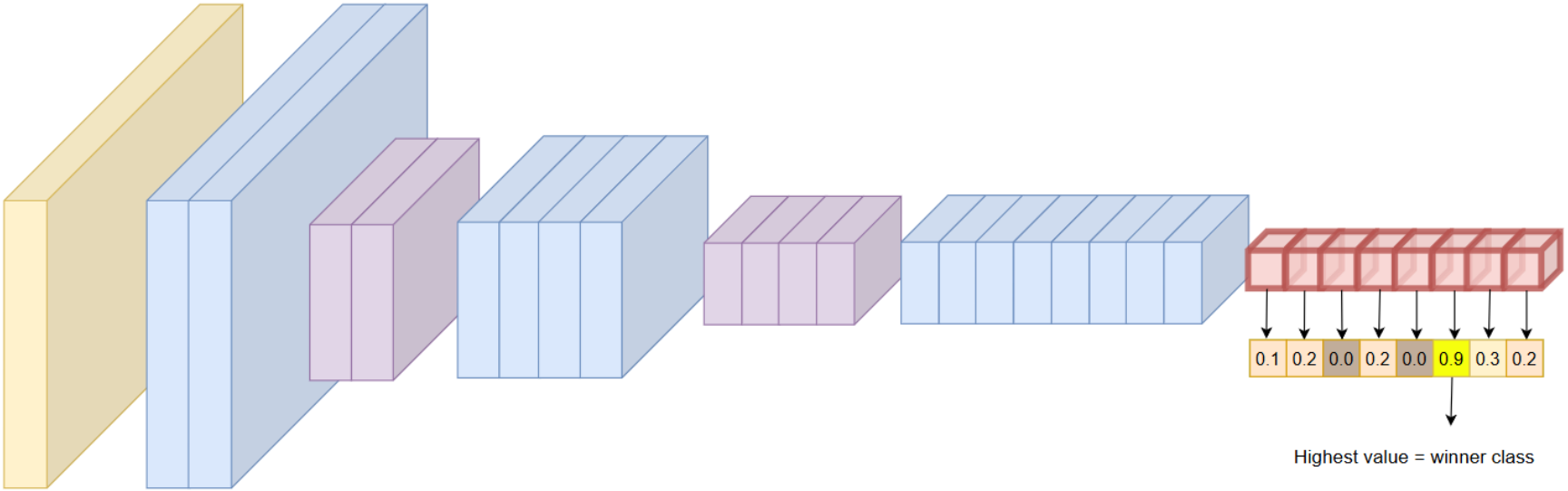
- We now create a funnel-like structure by alternating convolutional and pooling layers.
- **How does it help?**

Pooling

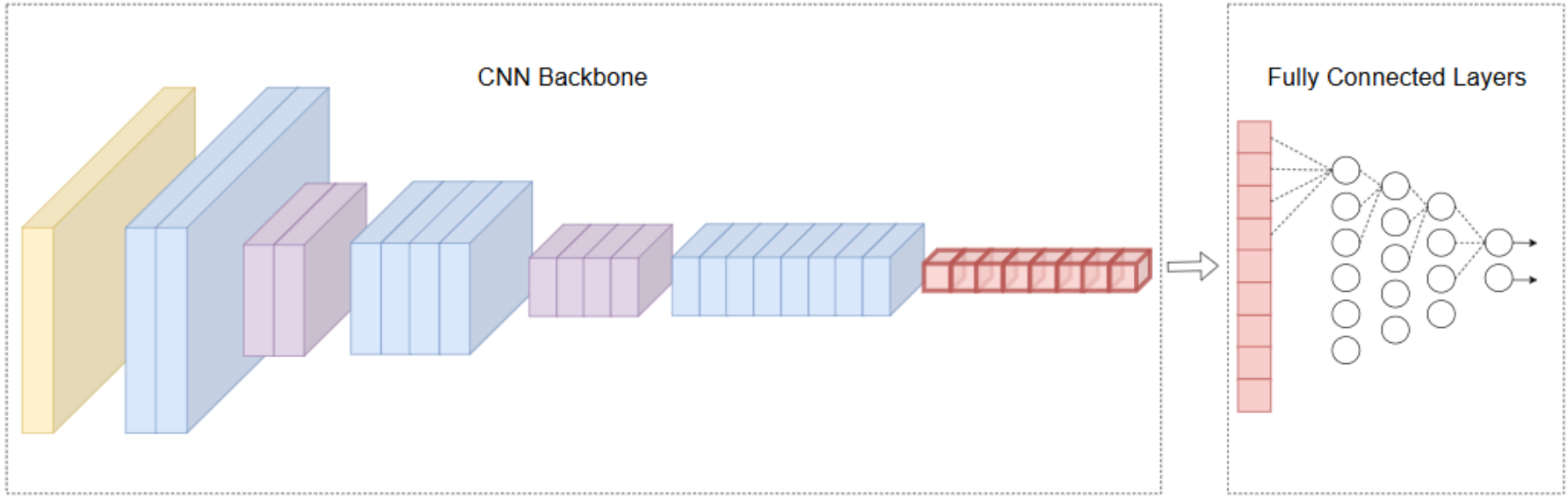


- We exchange spatial resolutions for feature descriptors (channels).
- **These channels can actually contain the information we need!**

Pooling and Convolutions – Example Architecture



Example Architecture – Fully Connected Head



Practical Questions

What Numbers We Have to Use?

- Common GPUs (e.g., using CUDA) are highly optimized for 32-bit floating point (FP32) operations
- Inputs, weights, and activations are usually kept in FP32.
- This is the default, and there's no need to round or quantize manually.

Should We Normalize the Input Images?

- If inputs (or filters) are too large or have large variance, the outputs of layers can:
  - Explode (very large values),
  - Vanish (very small values)
- **We shold normalize the values** (for example 0-1).

How the Convolution is Actually Done?

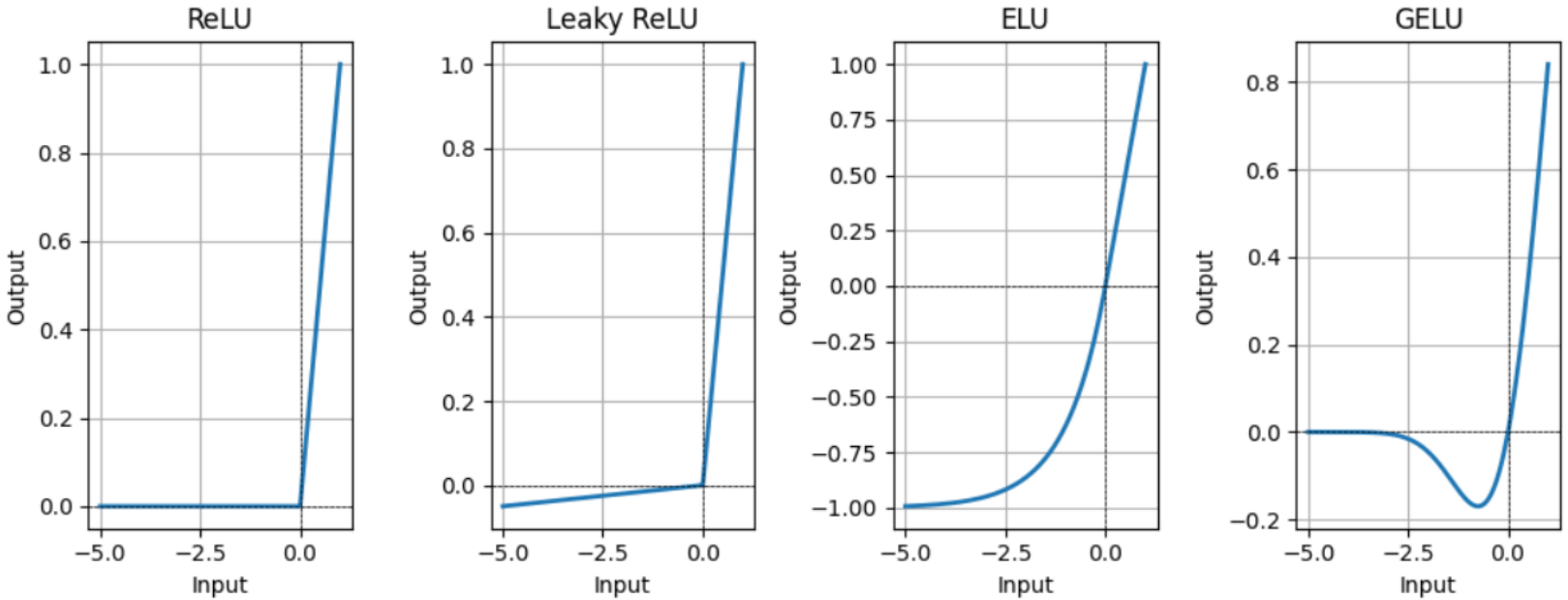
- The entire input tensor is unfolded into a collection of all possible windows (patches),
- These windows are arranged into rows of a big 2D matrix,
- The filters are flattened into a matrix as well,
- **Then a big matrix multiplication is done all at once** (in parallel).

What Activation Function We Use in CNNs?

Mainly **ReLU** and similar alternatives, because:

- Simple & fast to compute
- Introduces non-linearity while preserving positive values
- Promotes sparsity (many neurons become inactive → less overfitting)
- Avoids the vanishing gradient problem for positive inputs

How Those Functions Looks?



Training

Training Overview

CNN training is very similar to MLP:

1. **Forward pass**: input image flows through the network
2. **Loss computation**: measure prediction error
3. **Backpropagation**: compute gradients for all weights
4. **Weight update**: adjust weights using the gradients

Loss Computation

- **Measures how far** the network's predictions are from the ground truth.
- **Produces a single scalar value** representing the overall error.
- **Depends on the task and ground truth type**, e.g., regression vs. classification.

Table: Loss functions according to task

Task	Ground truth shape	Common loss functions
Regression	Single value or vector	Mean Squared Error (MSE), Mean Absolute Error (MAE)
Binary classification	Single probability	Binary Cross-Entropy (BCE),
Multi-class classification	One-hot vector	Cross-Entropy Loss
Image segmentation	2D class map	Cross-Entropy, Dice Loss
Object detection	Bounding boxes + class labels (structured)	Smooth L1 (boxes), Cross-Entropy (labels)

Cross Entropy

$$L = -\sum_{i=1}^{C} y_i \cdot \log(p_i)$$

C = number of classes,

y = ground truth (1 for the correct class, 0 for others — *one-hot encoded*),

p = predicted probability for class

Example ground truth (5 classes): [0, 0, 0, 0, 1]

Example predicted probability: [0.1, 0.0, 0.2, 0.1, 0.9)

Dice Loss

$$\text{Dice} = \frac{2 \cdot |P \cap G|}{|P| + |G|}$$

$$\text{Dice Loss} = 1 - \text{Dice}$$

- |P∩G| = number of pixels where **prediction and ground truth are both 1**
- |P| = number of pixels predicted as positive
- |G| = number of pixels in the ground truth positive region

Backpropagation

- Calculates how much each weight contributed to the overall error.
- Starts from the output layer and propagates the error **back through the network**.
- **Uses chain rule:** Computes gradients layer by layer using calculus.
- **Updates all parameters:** Provides the gradients needed for the optimizer to adjust weights.

Weight Update

- **Driven by gradients:** Uses gradients from backpropagation to determine how each weight should change.
- **Shared weights:** A single convolution filter is updated based on **all positions** where it was applied.
- **Optimizers** smartly adjust how weights are updated, controlling the step size and direction to make training faster, more stable, and more efficient.

Table: Popular optimizers

Optimizer	Key idea	Effect
SGD	Basic gradient descent	Simple but may be slow or unstable
SGD + Momentum	Adds memory of past gradients	Smooths updates, faster convergence
RMSprop	Scales updates by recent gradient magnitudes	Balances learning rate automatically
Adam	Momentum + adaptive scaling	Fast, stable, widely used

Training Data

Quality of Data

- **Data diversity:** Include **all relevant scenarios** the model must handle to avoid **overfitting**.
- **Resolution too high:** Model becomes **slow or untrainable**.
- **Resolution too low:** Important **features may be lost**.
- **Balanced dataset:** Ensure **fair representation** of all classes.

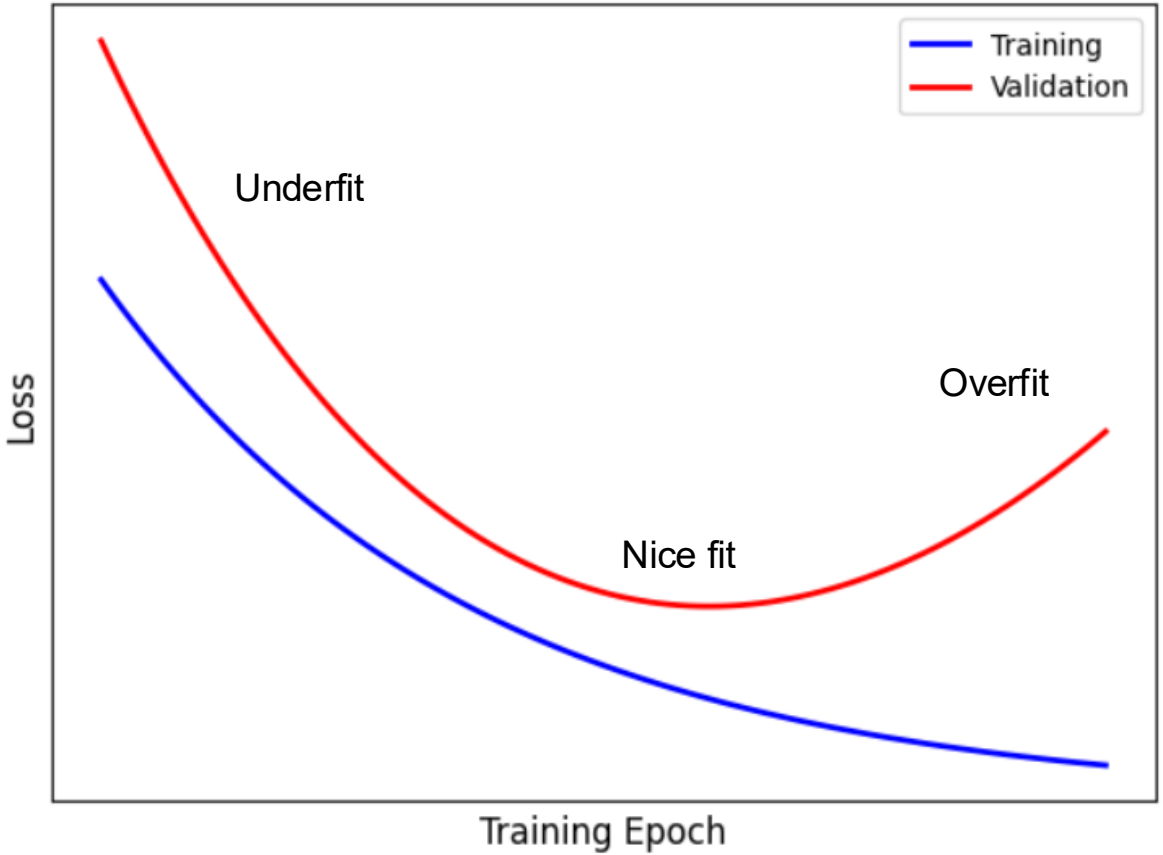
Quantity of Data – How Much is Enough?

- **Too little data:** High risk of **overfitting** and poor generalization.
- **More data = better model**, but with **diminishing returns** after a certain point.
- Dataset for **specifics tasks** commonly have 500-10000 images
- **Augmentation is often used** (synthetic modifications of data)

How to Split the Data?

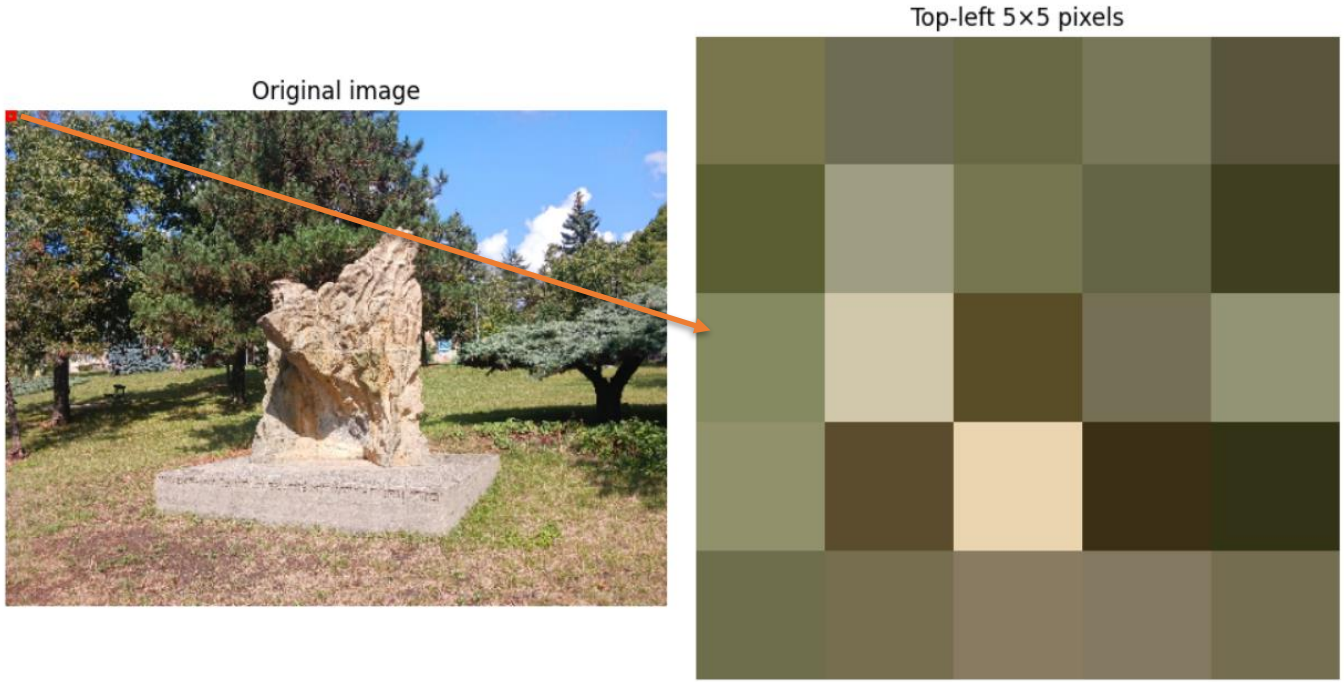
- **Epoch:** One complete pass through the **entire training dataset**.
- **Validate after each epoch** to monitor performance.
- **Validation loss is for evaluation only**
- Typical **training/validation split** is **80/20** or **90/10**
- There is **no universal rule** for choosing the split ratio – it depends on the **problem and available data**.

Overfitting vs Underfitting



Overfitting – an Intuitive Example

If there's a **fixed or unique pattern** (like the same 5×5 corner) in every training image, the network can **memorize this exact patch** rather than learning a generalizable concept.



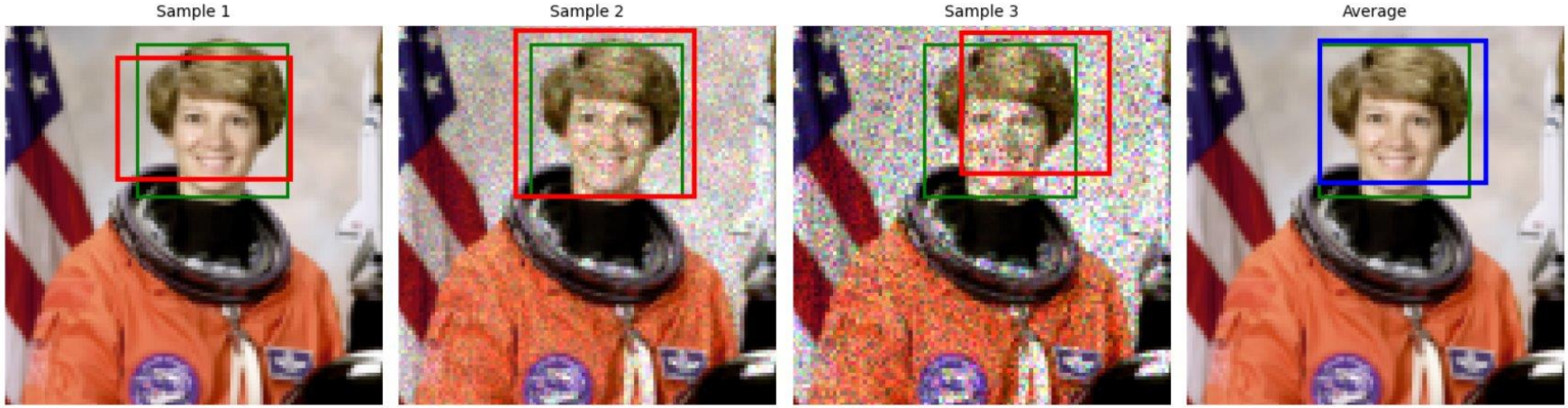
Data augmentation

- Augmentation is modification of the training data (e.g., color, noise, blur, flipping, etc.).
- If possible, apply augmentations **on the fly** so the model never sees the exact same image twice.

Batch Training

- Processes multiple samples together in each training step.
- Speeds up computation through parallelism (e.g., on GPUs).
- **The loss is computed per sample**.
- **Then summed or averaged over the batch.**

Batch Training



Practical Tips on Training

Dataset Suggestions

- **Minimize labeling errors:** Ensure your dataset has accurate and consistent labels
- Use augmentation for small datasets.
- Match model complexity to dataset size.

How to Setup the First Training?

- Use **the biggest batch** your device can fit in memory.
- Use Adam or AdamW optimizer with learning rate 1e-4 or 1e-3
- Use learning rate scheduler from beginning
- Use augmentation every time

Transfer Learning

- It is **reuse of pretrained models** trained on large datasets.
- Speeds up training and **reduces data requirements**.
- Commonly **freeze deeper layers.**

Conclusion

What we have learned?

- CNNs can **quantify specific features** and patterns in image.
- These quantifications (feature maps) can be used for reasoning.
- The performance of CNN is **limited by training data.**
- The computational cost of CNNs training can be significant.
- The core idea of CNNs is very simple, but technical details of implementation are rather complicated.