Neural Networks 2 - CNN Applications

18NES2 - Week 8, Winter semester 2025/26

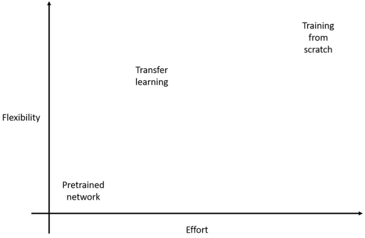
Zuzana Petříčková

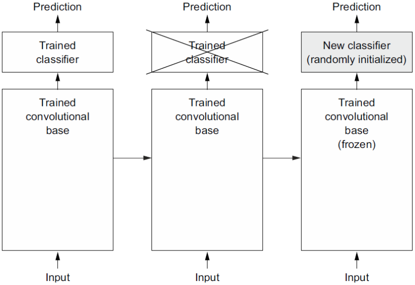
November 18, 2025

What We Covered Last Time

Ways to Build and Train a Convolutional Neural Network

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

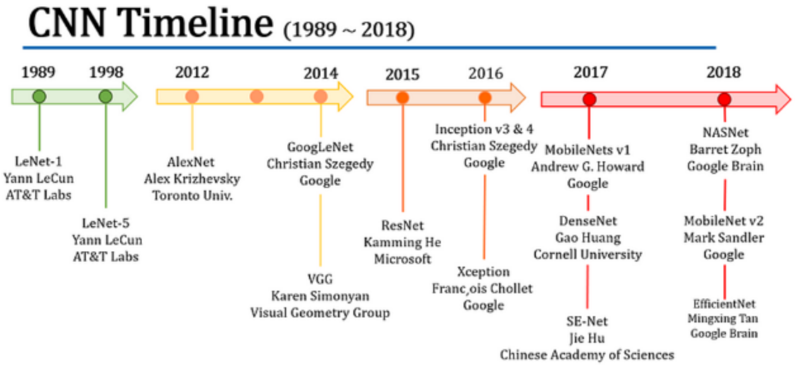




Sources of the images:
https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning
F. Chollet: Deep Learning with Python, Fig. 8.2

What We Covered Last Time

Evolution of CNN Architectures



Source of the image:

https://miro.medium.com/v2/resize:fit:640/format:webp/0*1SDlsJ7snNv_deec.png

What We Covered Last Time

Evolution of CNN Architectures

- **Early CNNs (LeNet, 1998)** — simple, shallow models
- **Deep CNNs (AlexNet, VGG, 2012–2014)** — bipyramidal architecture, increased depth, use of GPUs, ReLU activation, data augmentation.
- **Efficient and scalable designs (GoogLeNet, ResNet, 2014–2016)** — inception modules and skip connections enable very deep networks.
- **Lightweight and optimized models (Xception, MobileNet, EfficientNet, 2017-2019)** — separable convolutions, architectural scaling, and automated design (NAS).
- **Hybrid and Transformer-based models (ViT, ConvNeXt, 2020+)** — combine convolution and self-attention for superior global context understanding.

What We Covered Last Time

CNN architectual patterns

- The enhanced CNN architectures share common **design patterns** — repeated building blocks that make CNNs efficient and powerful
- Understanding these patterns helps you:
  - design better models for your own tasks,
  - fine-tune pretrained architectures more effectively,
  - and understand why modern CNNs work so well.

Remains for today:

- Practical examples of CNN design patterns in Keras (residuals, stacking, bottlenecks, separable convolutions)
- Construction of a **small custom model** inspired by modern architectures

This Week

1. CNN architectures and design patterns
   - Practical Examples
2. Applications of Convolutional Neural Networks
   - Encoder–Decoder Architectures and Semantic Segmentation
   - Object Detection
3. Graded Homework

Practical example: CNN Architecture Patterns

Common design patterns in modern CNNs:

- **Residual connections** — add shortcut links to ease training of deep networks (ResNet family).
- **Bottleneck blocks** — compress and expand feature maps to reduce computation (ResNet, Inception).
- **Depthwise separable convolutions** — factorize convolutions to make models lighter (Xception, MobileNet).
- **Inception-style modules** — parallel multi-scale feature extraction.
- **Batch Normalization and Activation ordering** — stabilize and speed up training.

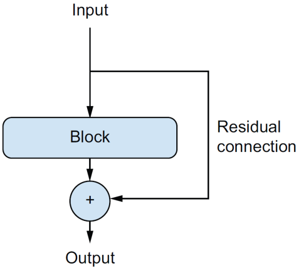
cnn_architecture_patterns.ipynb

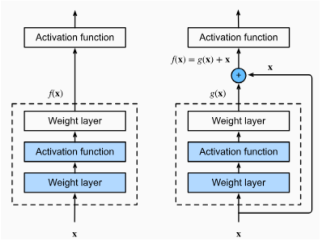
- includes residual blocks, bottlenecks, separable convolutions, and a small Xception-like model.

Practical example: CNN Architecture Patterns

Residual Block:

- Introduced in **ResNet (2015)**.
- Adds a **shortcut (skip) connection** that bypasses one or more layers.
- The output is the **sum of the input and transformed input:** $y = F(x) + x$



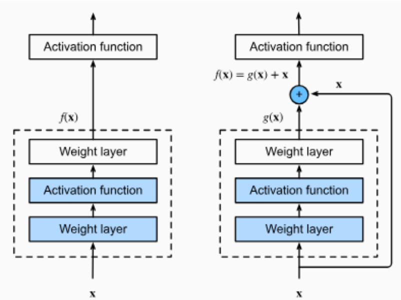


Source: F. Chollet, "Deep Learning with Python," Figure 9.3

Source: A. Zhang et al.: Dive into Deep Learning, Figure 8.6.2

Practical example: CNN Architecture Patterns

Residual Block:



Source: A. Zhang et al.: Dive into Deep Learning, Figure 8.6.2

Used in:

- **ResNet**, **ResNeXt**, **DenseNet (conceptually similar)**, and many modern CNNs.

Practical example: CNN Architecture Patterns

Bottleneck Block:

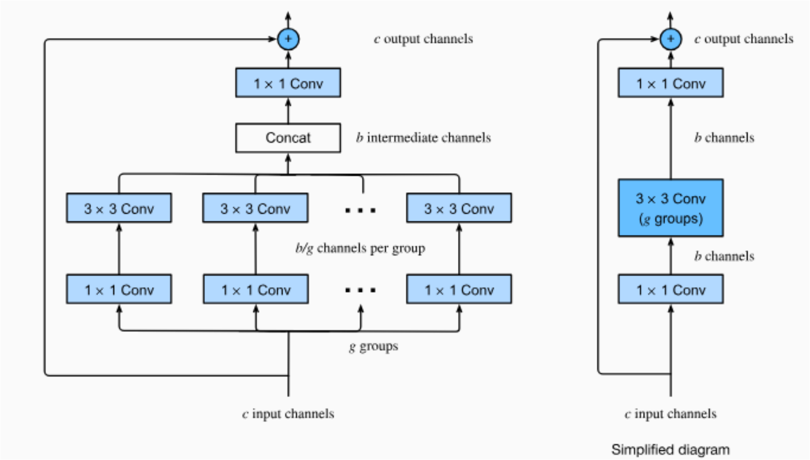
- Introduced in **ResNet-50/101 (2015)** to reduce number of channels to reduce computational cost.
- Consists of three convolutions:
  1. 1×1 convolution — **reduces** number of channels,
  2. 3×3 convolution — performs main spatial processing,
  3. 1×1 convolution — **restores** original dimensionality.
- Keeps representational power while reducing the number of parameters.
- This design enables networks with hundreds of layers.

Used in:

- **ResNet-50/101/152**, **Inception-v2/3**, **EfficientNet**.

Practical example: CNN Architecture Patterns

Bottleneck Block:



Source: A. Zhang et al.: Dive into Deep Learning, Figure 8.65

Practical example: CNN Architecture Patterns

Depthwise Separable Convolution:

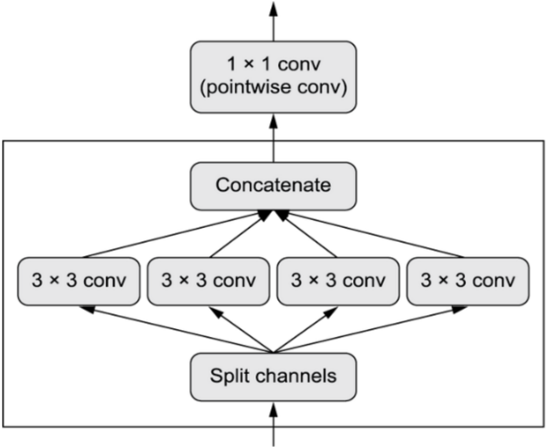
- Another breakthrough to improve efficiency and slim down layers
- Factorizes a standard convolution into two steps:
  1. **Depthwise convolution** — applies a single filter per input channel.
  2. **Pointwise convolution (1×1)** — combines the results across channels.
- Reduces the number of parameters and computations drastically.

Used in:

- **Xception** (2017) — "Extreme Inception": a deep CNN built entirely from separable convolutions.
- **MobileNet** and later efficient models.

Practical example: CNN Architecture Patterns

Depthwise Separable Convolution:



Source: F. Chollet, "Deep Learning with Python," Figure 9.10

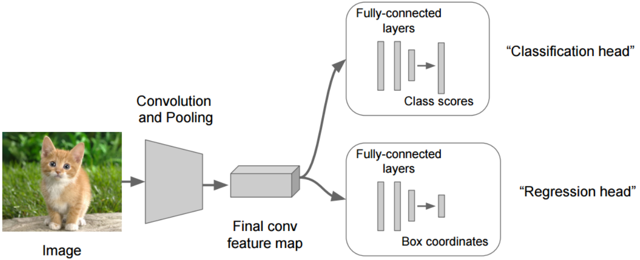
Practical Example: Custum CNN Model In Practice

Small Xception-like model on Cats vs. Dogs

- **cats_dogs_small_Xception.ipynb**
- The architecture captures Xception's design philosophy in a smaller form.
- **Observations:** The model learns more slowly but overfits less, it achieves higher test accuracy (compared to the classical bipyramidal architecture)
- Regularization and data augmentation are still important.

Applications of Convolutional Neural Networks

- The classification head of a neural network can be replaced by a different head to solve a different task on the same (or similar) data
  - Different classification tasks — classification head
  - Regression tasks — regression head
  - ...



Source: https://i.stack.imgur.com/FGrD1.png

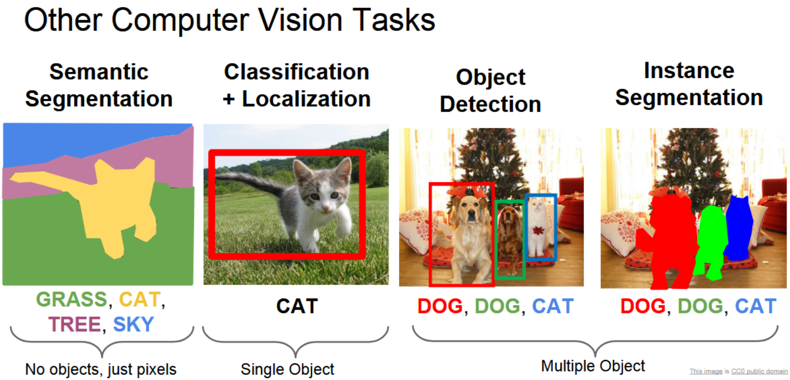
Applications of Convolutional Neural Networks

- Image classification — classification head
- Regression tasks — regression head

Example of Regression: Predicting the Angle of Digits

Source: https://matlabacademy.mathworks.com

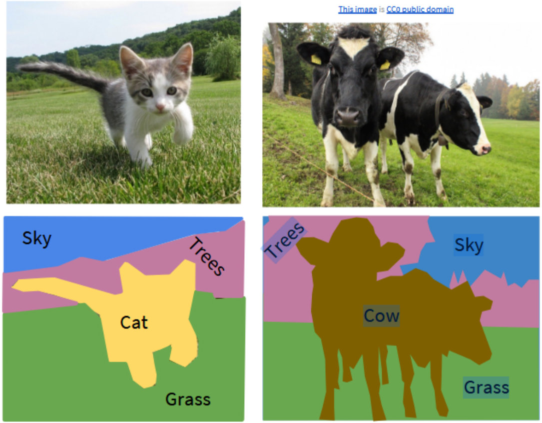
Applications of Convolutional Neural Networks



Source: `https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`

Semantic Segmentation

- Goal: assign a **class label to each pixel** in the image.
- Each pixel belongs to one of the predefined object categories.



Source: `https://cs231n.stanford.edu/slides/2024/lecture_9.pdf`

Encoder–Decoder Architectures

- General neural network design pattern for tasks that transform one structured input into another structured output.
- **Encoder:** compresses the input into a compact latent representation (feature extraction).
- **Decoder:** reconstructs the target structure (image, sequence, etc.) from this latent space.
- Widely used in:
    - Image restoration and denoising
    - Autoencoders
    - Semantic segmentation
    - Neural machine translation

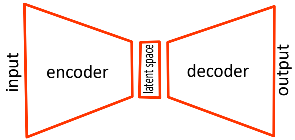
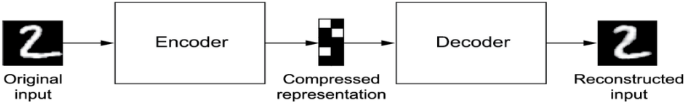


Image-to-Image Architectures

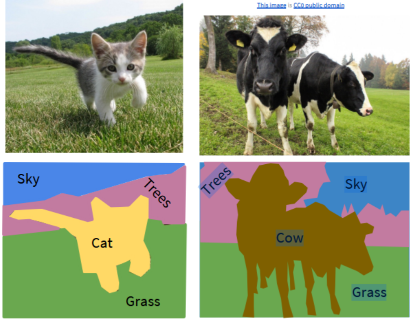
- Encoder–decoder networks can reconstruct or transform images:
  - Image restoration and denoising
  - Super-resolution
  - Inpainting (filling in missing parts)
- Typical design:
  - Encoder: convolution + pooling (feature extraction)
  - Decoder: upsampling using **transposed convolution** or **unpooling**



F. Chollet: Deep Learning with Python, Fig. 12.4

Encoder–Decoder for Semantic Segmentation

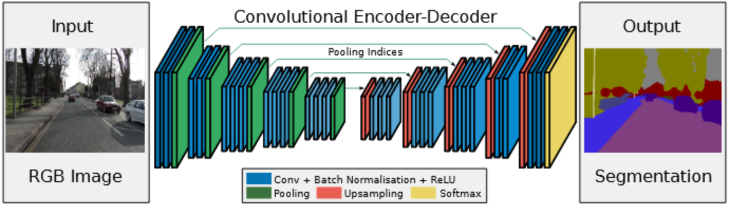
- Goal: assign a **class label to each pixel** in the image.
- Each pixel belongs to one of the predefined object categories.
- Typical architecture: **encoder–decoder CNN** (e.g., SegNet, U-Net).



Source: `https://cs231n.stanford.edu/slides/2024/lecture_9.pdf`

Encoder–Decoder for Semantic Segmentation

- **Encoder:** classic CNN that extracts multi-scale features (**downsampling using greater stride rather than poolng**).
- **Latent space:** compact feature representation.
- **Decoder:** reconstructs spatial resolution using **upsampling:**
  - **Transposed convolutions (a.k.a. deconvolutions)**
  - **Unpooling** (less common in segmentation)

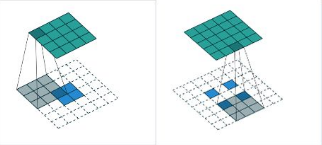


Source: SegNet — A Deep Convolutional Encoder–Decoder Architecture for Image Segmentation `https://arxiv.org/pdf/1511.00561`

Transposed Convolution (Deconvolution)

- Performs the inverse of a standard convolution: increases spatial resolution.
- Commonly used for image reconstruction, segmentation, and generative models.
- Great interactive visualization:
  github.com/vdumoulin/conv_arithmetic
  deeplizard.com/resource/pavq7noze4

N.B.: Blue maps are inputs, and cyan maps are outputs.

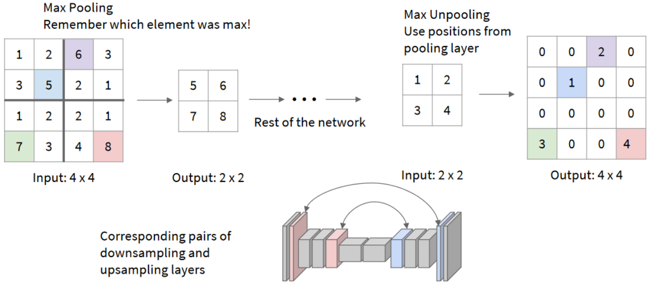


No padding, no strides, transposed	No padding, strides, transposed

Source: `https://github.com/vdumoulin/conv_arithmetic`

Alternative Upsampling: Unpooling Layer

- Restores spatial structure by reversing pooling operations.
- Often used in generative or visualization models (e.g., feature inversion).
- Less suitable for semantic segmentation, as it doesn't learn spatial refinement.



Source: `https://cs231n.stanford.edu/slides/2024/lecture_9.pdf`

Practical example: Encoder–Decoder for Semantic Segmentation

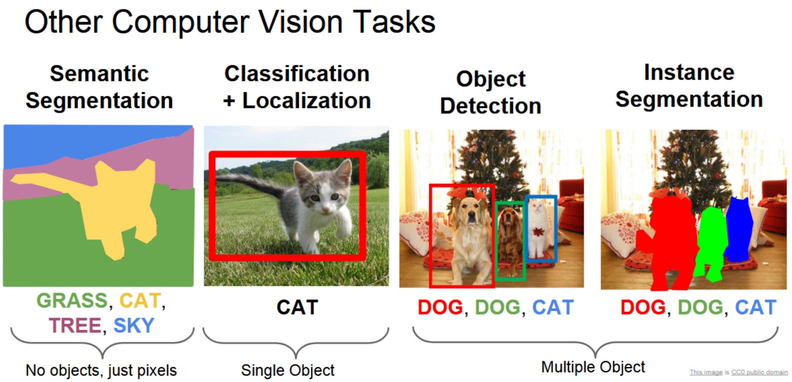
Simple Encoder–Decoder Architectures (Keras): simple_encoder_decoder_architectures.ipynb

- Minimal working examples illustrating key architecture patterns:
  - **Strided-convolution** encoder–decoder,
  - **Pooling–unpooling** architecture,
  - **U-Net–style skip connections**.

Segmentation Example (Oxford Pets): pets_segmentation.ipynb

- Training a simple strided-convolution encoder–decoder model from scratch for multi-class semantic segmentation.

Applications of Convolutional Neural Networks

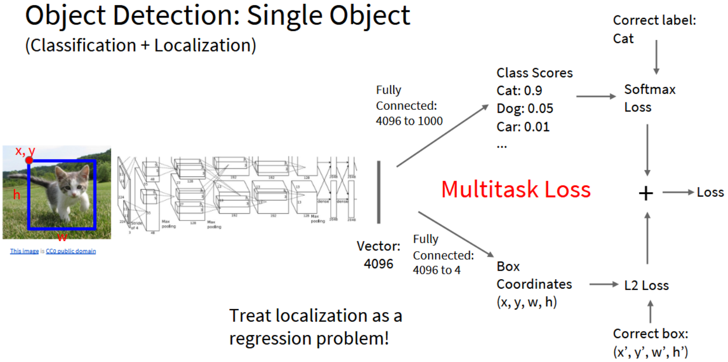


Source: `https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`

CNNs and Object Detection

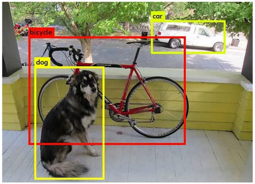
Single object – two heads:

- **Classification head** – predicts the object class.
- **Regression head** – predicts the bounding box coordinates.

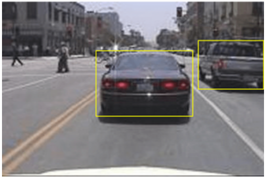


Source: `https://cs231n.stanford.edu/slides/2024/lecture_9.pdf`

CNNs and Multi-Object Detection



Source: Blog by Matthijs Hollemans



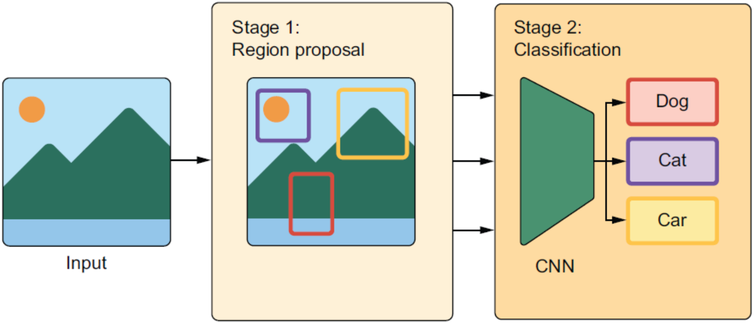
Sources: `https://matlabacademy.mathworks.com`

Triantafyllidou, D. et al.: *A Fast Deep Convolutional Neural Network for Face Detection in Big Visual Data.*

R-CNN: Region-based Convolutional Neural Network

Two-stage detector

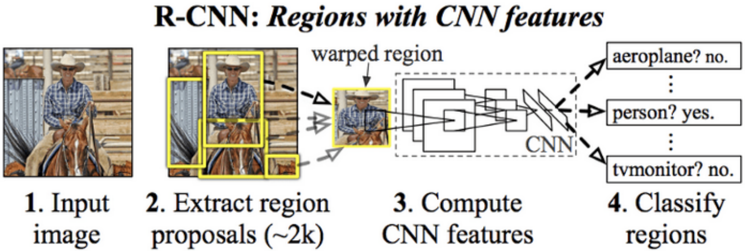
1. Extract candidate regions (ROIs, regions of interest)
2. Run classification on each region



Source: F. Chollet, "Deep Learning with Python," Figure 12.2

R-CNN: Region-based Convolutional Neural Network

- The input image is first divided into candidate regions (ROIs) using **selective search** heuristics.
- Each region is resized to a fixed size and passed through a pretrained CNN (e.g., VGG-16 trained on ImageNet in the original paper).
- Two heads are used:
  - a classifier (originally SVM) to classify each region,
  - a regressor to refine the bounding box.



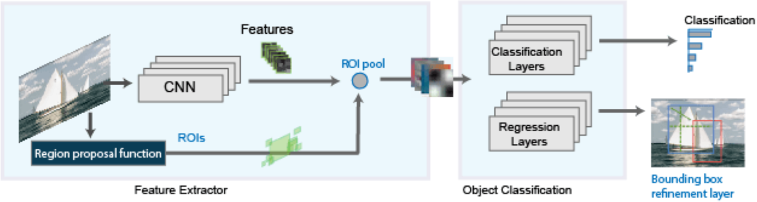
Fast R-CNN and Faster R-CNN

The original R-CNN

- Very computationally expensive (many independent forward passes for each image)

Fast R-CNN

- Speeds up computation by **sharing convolutional features** between all regions of interest (ROIs).
- The whole image is passed through a CNN base once; ROIs are pooled from the feature map.

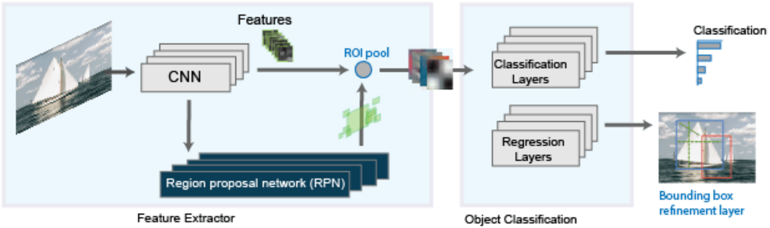


Source: *https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-*

Fast R-CNN and Faster R-CNN

Faster R-CNN

- Introduces a **Region Proposal Network (RPN)** that learns to propose ROIs directly inside the model.
- Replaces the slow selective search step.



Source: *https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html*

One-Stage Detectors: YOLO, RetinaNET, after 2015

Key idea:

- Predict **bounding boxes and classes in a single forward pass**, without a separate region proposal stage.

YOLO (You Only Look Once)

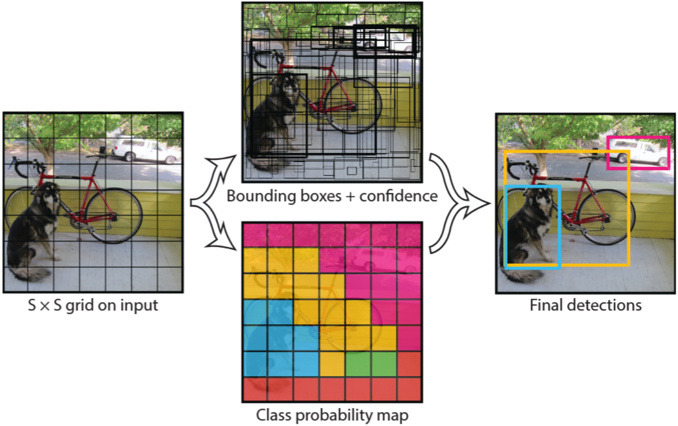
- Divides the image into a grid and predicts bounding boxes + class probabilities for each cell.
- Extremely fast – suitable for real-time applications.

SSD (Single Shot MultiBox Detector)

- Uses multiple feature maps at different scales to detect objects of various sizes.
- Also a one-stage detector, balancing accuracy and speed.

Modern detectors (e.g. YOLOv5/8, RetinaNet, DETR) further improve speed–accuracy trade-offs.

YOLO model



Source: Redmon et al. You only look once: unified, real-time object detection, 2015

Practical example: Object detection

Original source of the example

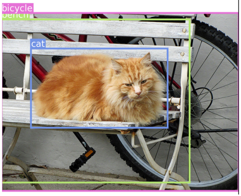
- **Cholett: Deep learning with Python**

coco_object_detection.ipynb

- Two examples:
  - Training a YOLO model from scratch on the COCO dataset
  - Using a pretrained RetinaNet detector

COCO (Common Objects in Context)

- Dataset designed for object detection, segmentation, and image captioning



6th Graded Homework: Image Segmentation

Goal: Compare several encoder–decoder architectures for **semantic segmentation** on the **Oxford-IIIT Pets** segmentation dataset.

Architecture variants to compare:

- **Model 1:** Strided-convolution encoder–decoder (baseline, no skip connections, as in the lecture notebook).
- **Model 2:** Pooling–unpooling model (max-pooling + unpooling / nearest-neighbour upsampling).
- **Model 3:** U-Net style encoder–decoder (skip connections between matching levels).

You can start from the example notebooks:

- **simple_encoder_decoder_architectures.ipynb**
- **pets_segmentation.ipynb**

6th Graded Homework: Requirements

Pipeline: load $\rightarrow$ preprocess $\rightarrow$ build $\rightarrow$ train $\rightarrow$ evaluate

Include in your notebook:

- Training curves (loss + IoU / pixel accuracy).
- Final segmentation metrics on the test set.
- Visual comparison of predictions for all three models.
- Short discussion:
  - Which architecture performs best and why?
  - Does the use of skip connections affect results?
  - Compare also the training time.

Optional:

- Try adding regularization: e.g., data augmentation, dropout, weight decay, or label smoothing.

6th Graded Homework: Submission

Submission

- Submit the notebook by **Dec 1, 2025**.
- **Consultation required by Dec 5, 2025** to receive points (short discussion after lab or individually).
- **Points: 1–2**
    - +1 point for implementing the required three models, their comparison and visualizations.
    - +1 point for additional variants or improvements