Convolution Neural Networks
Common Architectures

CTU, FS, U12110

Ing. Matouš Cejnek, Ph.D.

Short Review of CNNs

What are CNNs?

- CNNs are neural networks popular for image processing.
- Convolution captures patterns like edges or textures to help models understand visual structures.
- **CNNs are suitable to learn hierarchical patterns in images—in order to solve visual recognition tasks.**

Why We Use CNNs?

- They are easy to implement and deploy in production.
- They can solve very complex problems suprisingly well.
- We use them for all big tasks:

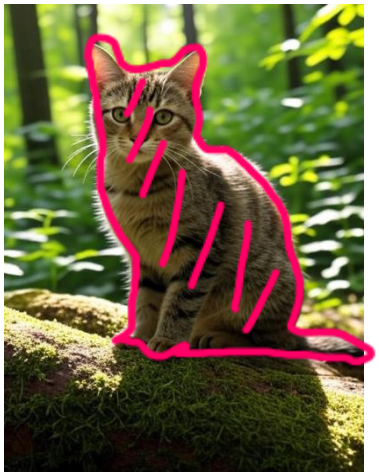
  Classification, Detection, Segmentation, Regression

Tasks Suitable for CNNs

Classification
Is this a cat image?

Detection
Where are the cats?

Segmentation
What pixels are cat?



Regression
How many cats?

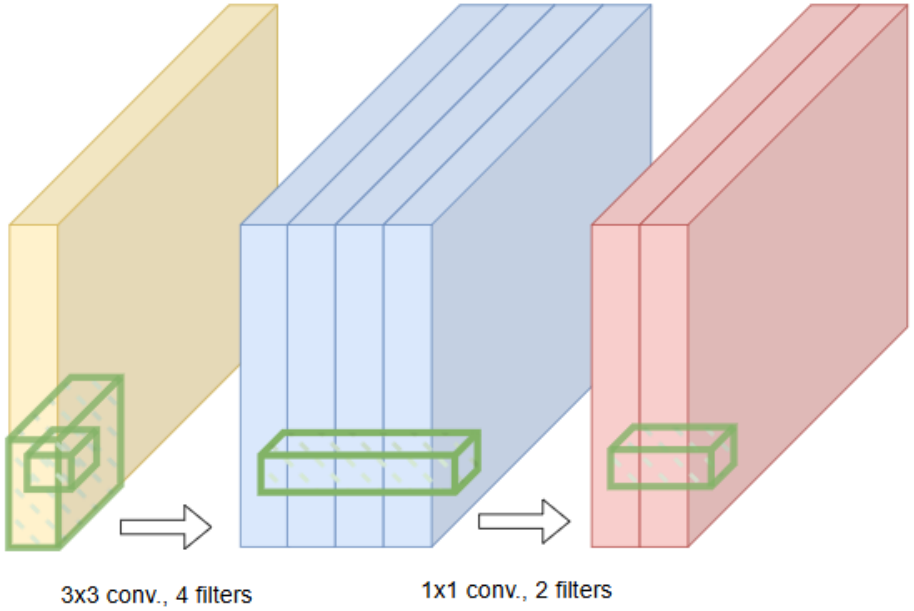
What is the Limitation?

- CNNs can solve very complex problems.
- But only if we have data that is **excellent in both quality and quantity**.
- We use data (observations) instead of prior knowledge – **blackbox vs. whitebox issue.**

How CNNs Work?

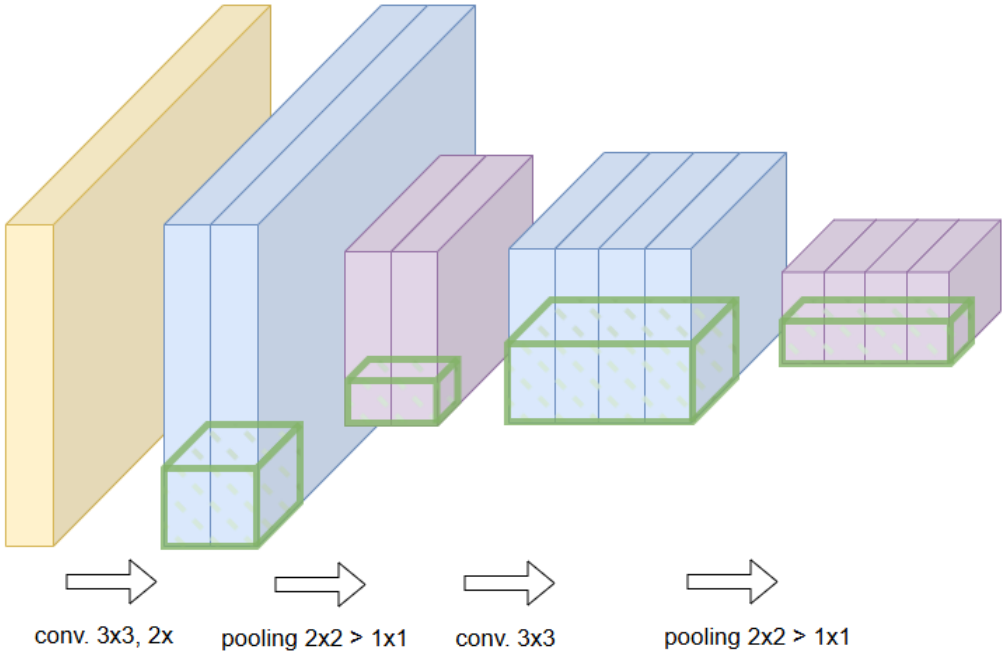
- Convolution **filters** can quantify patterns.
- We create a complex structure (**model**) consisting of convolution filters.
- We search for **optimal filter parameters** to make the model working.

Feature Maps



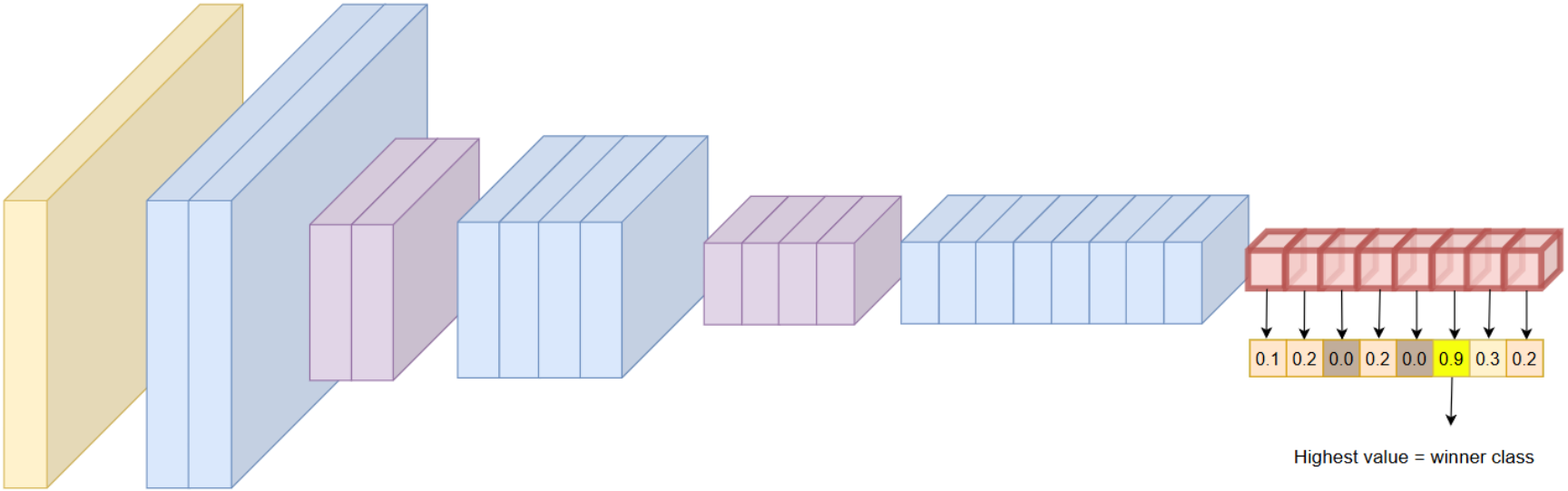
We can generate multiple feature maps and refine them using subsequent 1×1 convolutions to reduce dimensionality or enhance specific features. **But what to do with them?**

Pooling



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

Pooling and Convolutions – Example Architecture



Modular Design

What Do We Actually Need?

Different outputs shapes for similar images:

- **Detection** – we do not how many objects are there
- **Segmentation** – output is feature map
- **Classification** – number of classes can change

We would like to **avoid costly training** if possible.

What We Know

- Deep CNN layers are hard to train.
- Good CNN architectures can work over multiple domains.

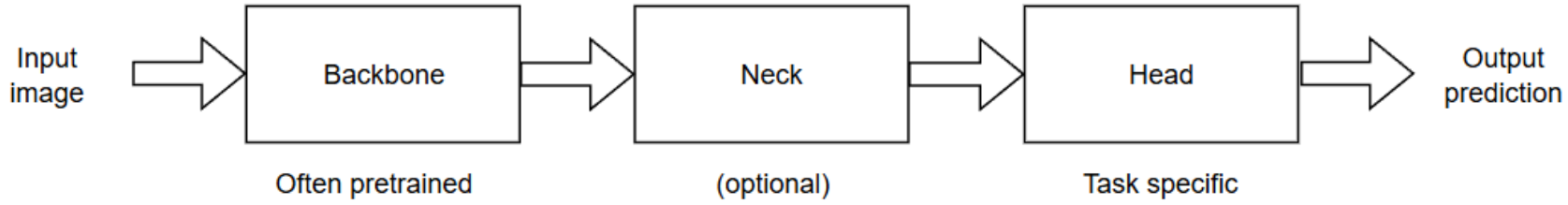
How to recycle the CNNs for different tasks?

Modular Design Pattern

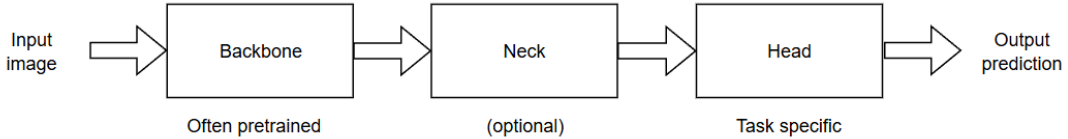
Backbone – General pretrained CNN

Neck – Specific features highlighter

Head – Task-specific predictor

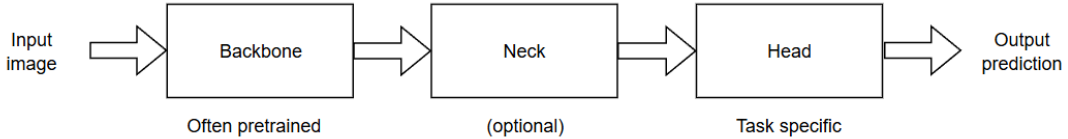


Backbone



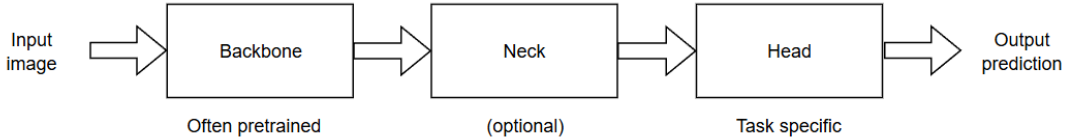
- A stack of convolutional layers (usually deep) responsible for generic feature extraction from input data (e.g., edges, textures, shapes).
- Can be **pretrained** on large datasets.
- Outputs **feature maps** with spatial and semantic information.

Head



- The final layers responsible for producing the **task-specific output**.
- Examples:
  - Fully connected layers for classification.
  - Bounding box regressors
  - Mask predictors for segmentation.

Neck



- Intermediate layers that process and refine the features from the backbone.
- **Highlight or fuse features** that are relevant to the specific task.
- Examples: **attention modules** and **bottlenecks**

Popular Backbones

Purely Sequential Backbones

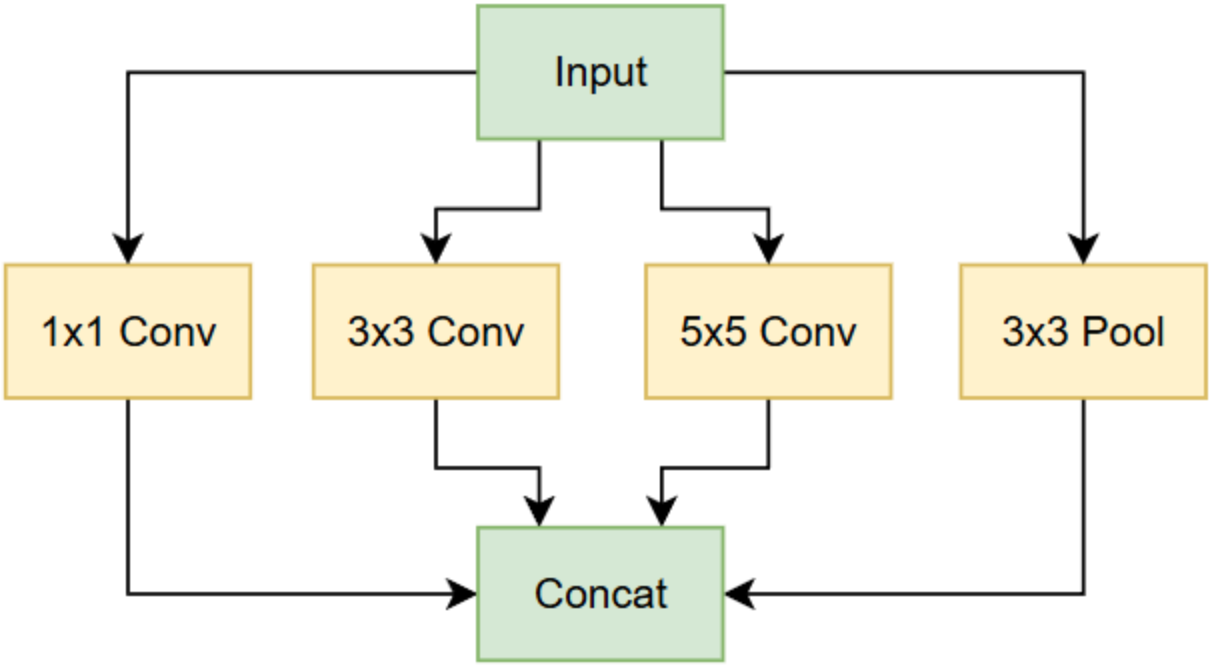
- Layers are stacked one after another
- Use **simple, uniform blocks**, e.g., 3×3 conv + 2×2 pooling.
- Easy to **understand and implement.**
- **VGG** family are classic examples, built only from conv and pooling layers.
- **Limited depth** — training deep models is hard without residuals.

VGG Based Networks (2014+)

- **VGG-11** – 8 conv layers + 3 fully connected layers
- **VGG-13** – 10 conv layers + 3 fully connected layers
- **VGG-16** – 13 conv layers + 3 fully connected layers *(most popular)*
- **VGG-19** – 16 conv layers + 3 fully connected layers *(deepest)*

These networks are often **used without the fully connected layers**, serving purely as a **feature extraction backbone**.

Inception (2014)



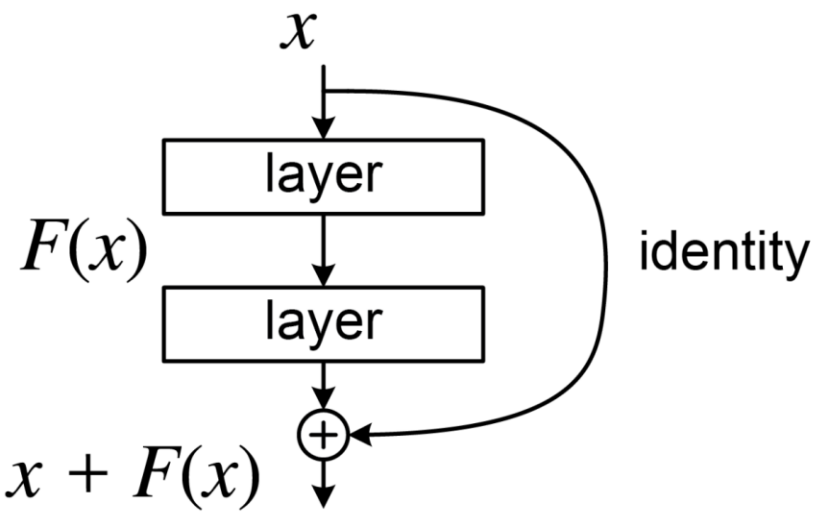
Pooling stride = 1

It does not reduce the dimensions.

Inception Based Networks

- **GoogLeNet / Inception v1 (2014)**
- **Inception v2 (2015)**
- **Inception v3 (2015)**
- **Inception v4 (2016) -** Utilizing residual connections

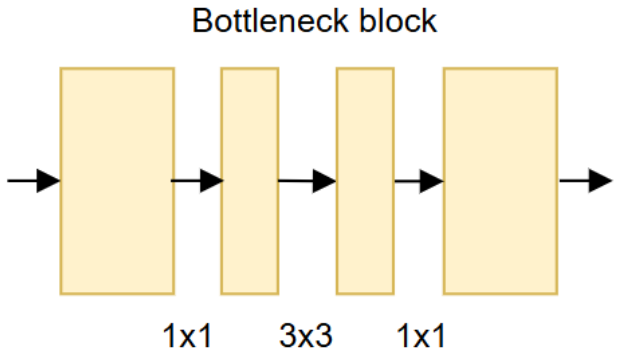
Residual Connections (2015)

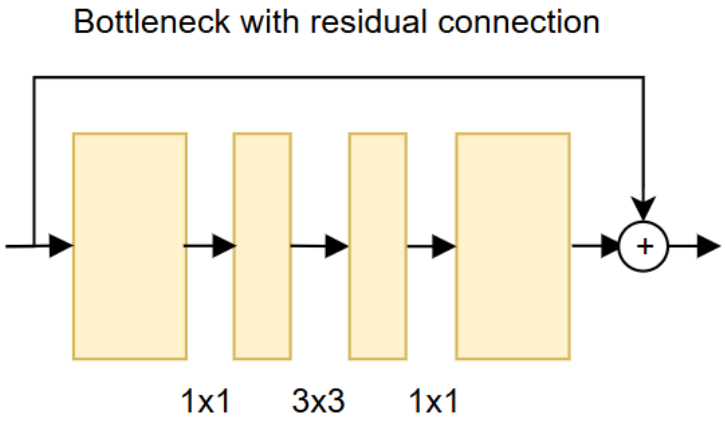


- Solves **vanishing gradients** in very deep networks.
- Output is **y=F(x)+x** (skip connection).
- Improves **training stability** and gradient flow.
- Introduced in **ResNet (2015)**, enabling >100-layer networks.

Bottleneck

- Three-layer block: **1×1 → 3×3 → 1×1**
- Middle layer has **reduced channels** (¼ of original).
- Cuts **computational cost** while keeping depth.
- Enables **very deep networks** without huge FLOPs.





Few Notable ResNets

- **ResNet-18 / ResNet-34** – lightweight models for smaller tasks.
- **ResNet-50** – common backbone using bottleneck blocks.
- **ResNet-101 / ResNet-152** – deeper models for high-accuracy tasks.
- **ResNeXt-50 / ResNeXt-101** – adds *cardinality* (grouped convolutions).

DenseNet (2016)

- In a DenseNet block, **each layer receives inputs from *all previous layers***
- Instead of adding features like in ResNet, DenseNet **concatenates them**, preserving all earlier features.
- **Strong accuracy** with less computation.

Popular DenseNets

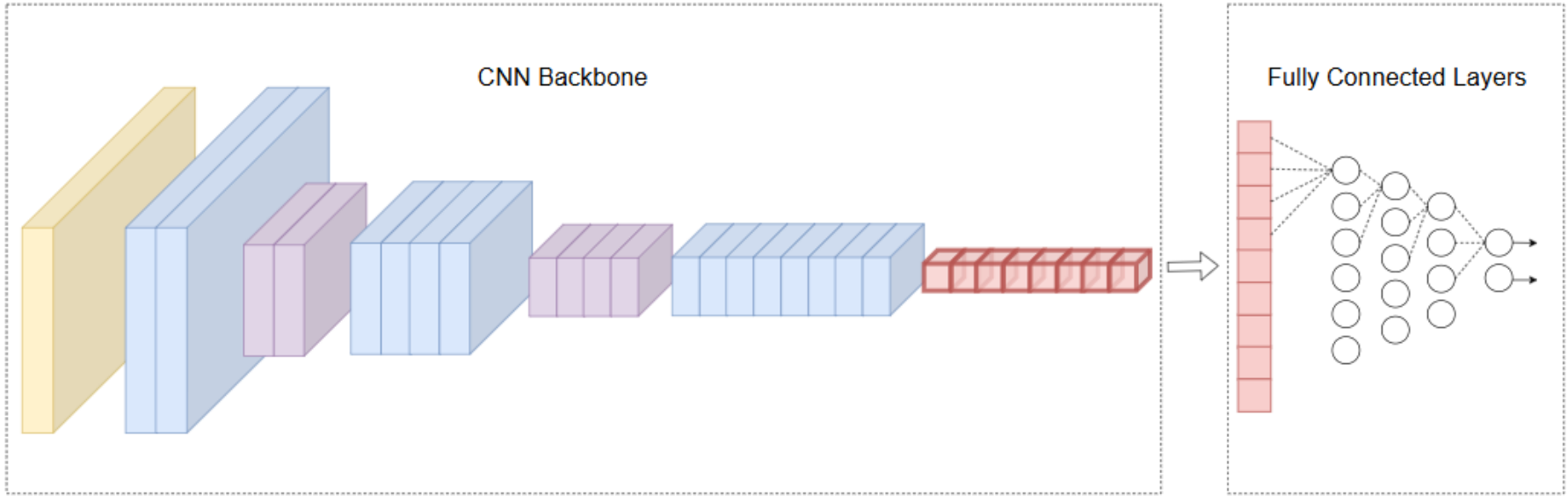
- **DenseNet-121:** 121 total layers, mobile-friendly applications.
- **DenseNet-169**
- **DenseNet-201**
- **DenseNet-264:** Largest standard version

Classification and Regression Heads

Common Head Design

- Feature maps are **reshaped** before prediction, often using **GAP** or **flattening**.
- A **Multi-Layer Perceptron (MLP)** is applied to the resulting vector for **classification or regression**.
- **Softmax** is commonly used on the output for **classification tasks**.
- **Regression heads** usually have a **linear output.**

Backbone and Head Example



Backbone Output Reshaping

- Global Average Pooling (GAP):
  - For each channel, computes the **average value across all spatial positions**. Example: 7×7×512 → 1×1×512 → 512
- Flattening:
  - Takes the **entire feature map** (H × W × C) and flattens it into **one long vector**. Example: 7×7×512 → 25,088

Softmax Output Activation

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Raw model output (logits): `[2.0, 1.0, 0.1]`

1. Step: exponentiate → `[e², e¹, e⁰·¹] = [7.39, 2.72, 1.11]`
2. Step: normalize → `[7.39/11.22, 2.72/11.22, 1.11/11.22] ≈ [0.66, 0.24, 0.10]`

   Interpretation → **Class 1:** 66%, **Class 2:** 24%, **Class 3:** 10%

Detection Heads

Detection Head Design

- It predicts **object class** and **bounding box coordinates.**
- Often operate on **multiple spatial locations** to detect several objects.
- Can be **single-stage** (direct predictions) or **two-stage** (region proposals first).
- Use **multi-scale features** to handle objects of different sizes.

Simple Detection Taxonomy

1. **Two-Stage Detectors** (High Accuracy)
2. **One-Stage Detectors** (High Speed)
3. **Modern Trends** (Transformers and other complex models)

Two-Stage Detectors

- **First stage:** generate **region proposals** where objects may be located.
- **Second stage:** classify each proposal and refine its bounding box.
- Typically achieve **higher accuracy**, especially for **small objects**.
- Slower than one-stage detectors due to **two-step processing**.

Popular Two-Stage Detectors

- **R-CNN (2014)**
- **Fast R-CNN (2015):** Shared backbone for proposals, much faster than R-CNN.
- **Faster R-CNN (2015)**
- **Mask R-CNN (2017)**

(Note: R-CNN → Region-based Convolutional Neural Network)

R-CNN (2014) – Not end-to-end

Three separate steps:

- Generate **region proposals** (e.g., Selective Search).
- Run a **CNN** on each region to extract features.
- Train **SVM classifiers** and bounding box regressors separately.
- Very **slow** and complex pipeline.

Fast R-CNN (2015) – Partially end-to-end

- Uses a **shared backbone CNN** for the whole image → faster feature extraction.
- Still **relies on external region proposals** (Selective Search).
- Classification and bounding box regression **are trained together**, but region proposal step is **not learned**.

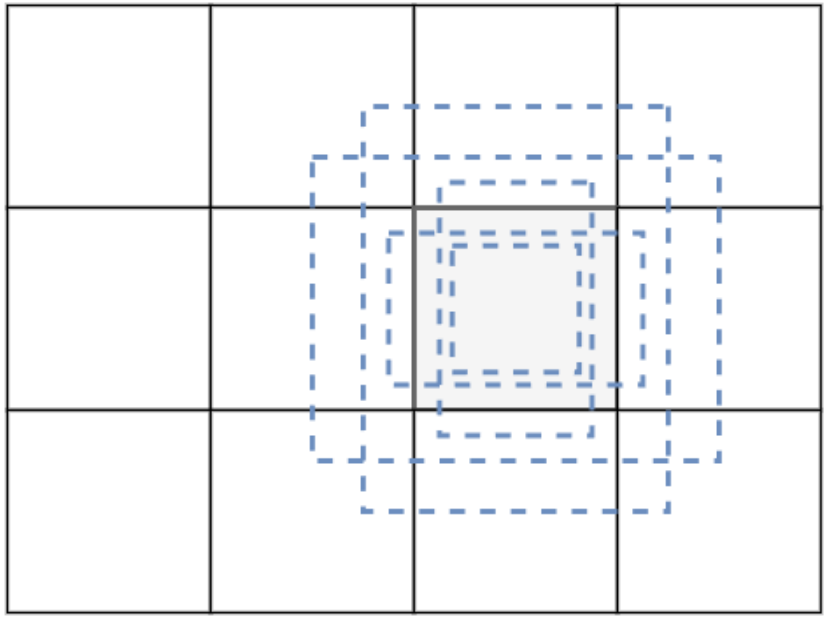
Faster R-CNN (2015) – Fully end-to-end

- Introduces **Region Proposal Network (RPN)** that **learns to generate proposals**.
- Entire pipeline — backbone, RPN, classification, and box regression — is trained **jointly**.
- First truly **end-to-end two-stage detector**.

Popular One-Stage Detectors

- **YOLO (2016)** – "You Only Look Once," real-time object detection.
- **SSD (2016)** – "Single Shot Detector," popular for balanced speed and accuracy.
- **YOLOv3/v4/v5+** – Successive YOLO improvements for accuracy and efficiency.

The Concept of Grid and Anchors



Grid Cell: (x_c, y_c)

Coords: (x, y, w, h)

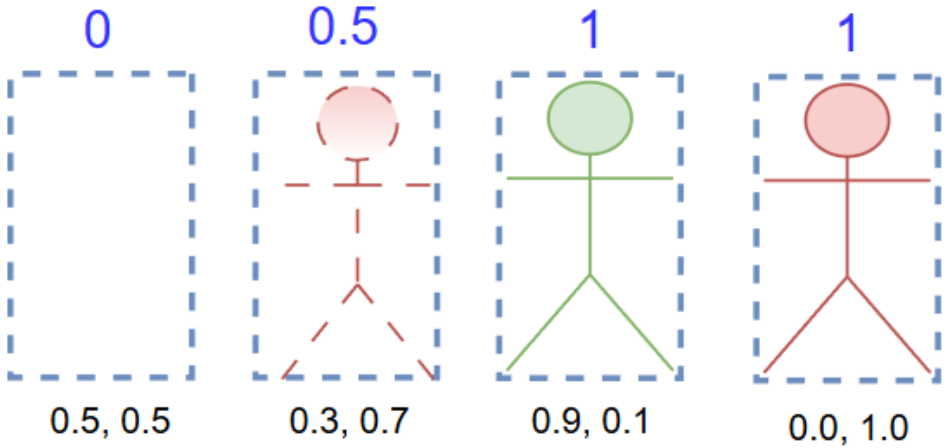
Class: $(c_1, ..., c_n)$

Grid and Anchors

- **Multiple anchors per grid cell**, with **multiple grid scales** per image.
- Total number of detections is **limited by the number of anchors**.
- **Filtering step** to keep only valid detections:
  - **YOLO:** objectness score, **SSD:** background class.
- **Non-Maximum Suppression (NMS)** to remove duplicates.

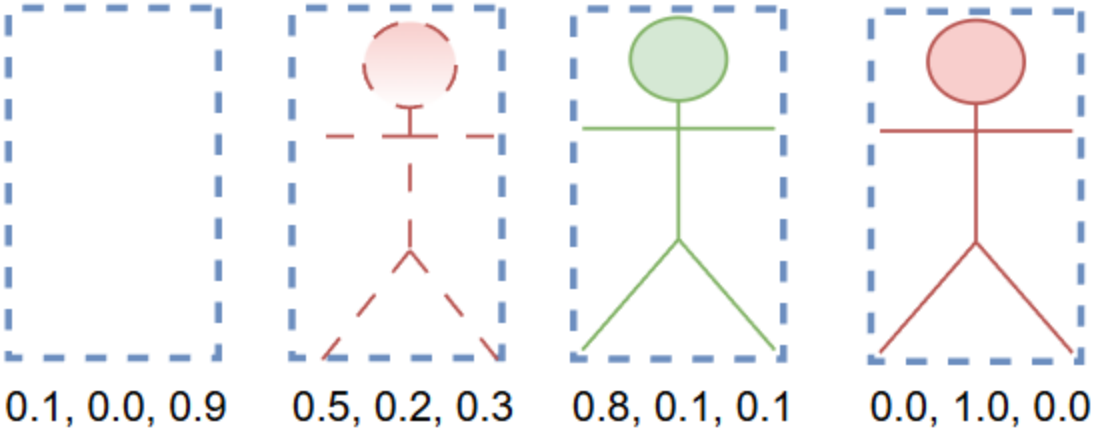
YOLO: Objectnes Score

- Additional output describing how likely the box is contain object.



SSD: Background class

- One extra class for background.



Detection Summary

- Generate candidate boxes across the image.
- Refine boxes and classify potential objects.
- Filter and keep meaningful detections.

Segmentation Models

Segmentation Models

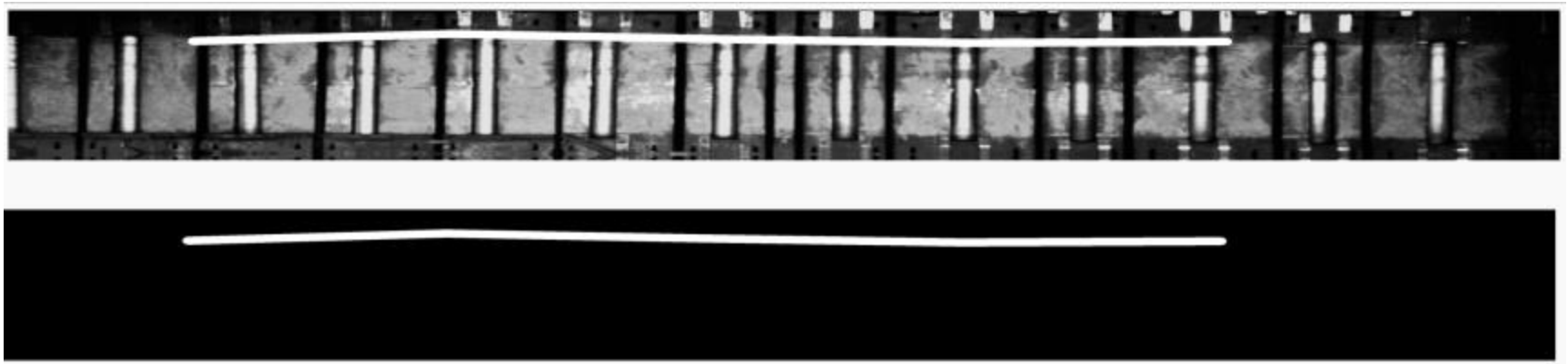
- Predict a **class label for each pixel** instead of a single class for the whole image.
- Often use an **encoder–decoder structure.**
- Common outputs are **per-pixel probability maps** processed with softmax or sigmoid.

Segmentation

- **Semantic segmentation** assigns class to each pixel.
- **Instance segmentation** separates individual object instances.

Segmentation Example – Steel Billet

One class – one pixel-wise classification map:



Segmentation Example - Apples

Segmentation Example - Apples

Input

Target

Output

Segmentation Example - Apples

Input

Target

Output

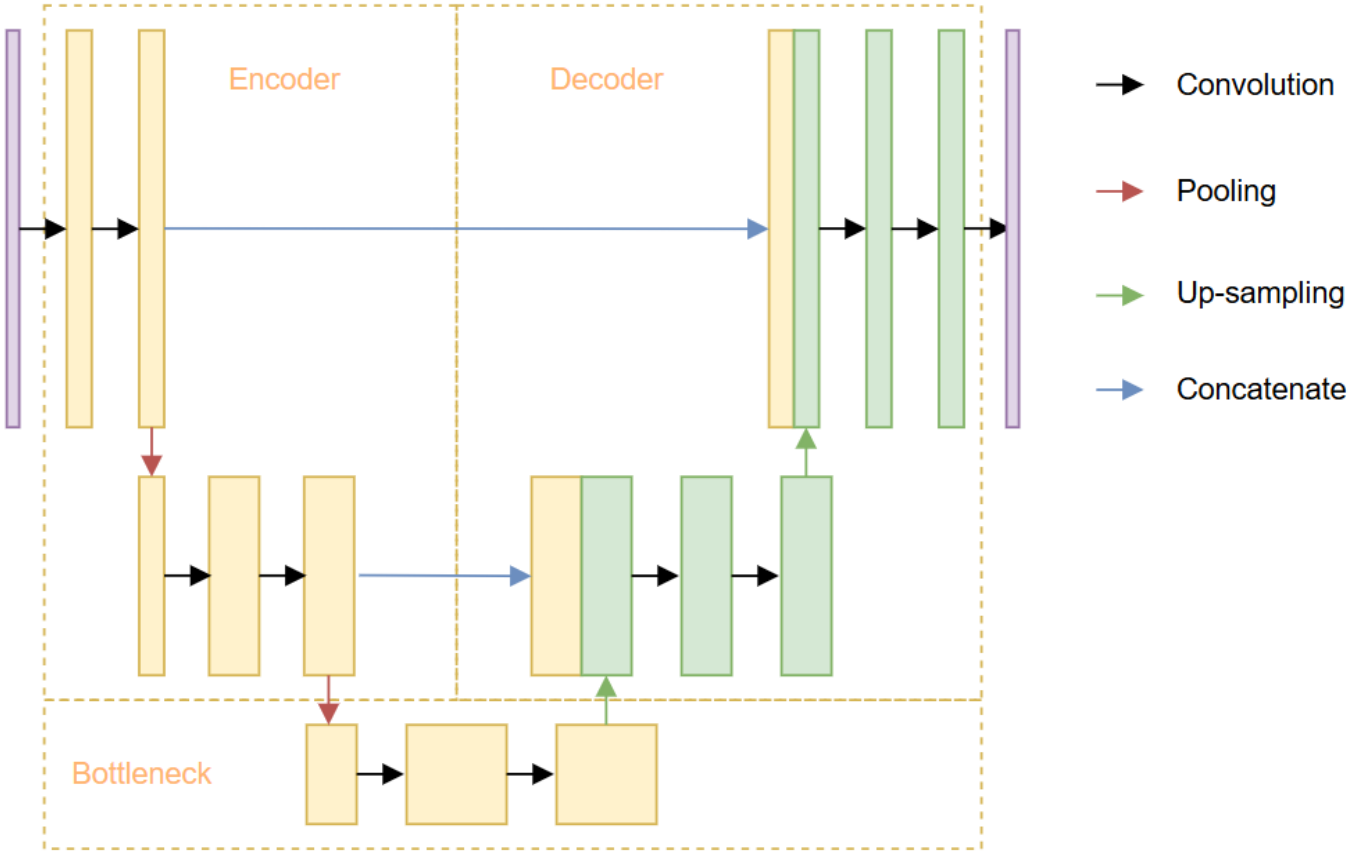
Encoder

- Acts like a **backbone**, using convolution and pooling to **gradually reduce spatial resolution**.
- The encoder is often **built on a popular backbone** (e.g., VGG, ResNet) to leverage pretrained feature extraction.
- Captures **high-level semantic features** while losing fine details.

Decoder

- **Upsamples** low-resolution feature maps back to the **original image size**.
- Uses **transposed convolutions**, **bilinear upsampling**, or **unpooling**.
- Often combines encoder features via **skip connections** to recover **spatial detail and sharp boundaries**.

U-Net Architecture



Few Popular Segmentation Models

- **FCN (Fully Convolutional Network, 2015):** First end-to-end trainable semantic segmentation network.
- **U-Net (2015):** Encoder-decoder with skip connections, popular in medical imaging.
- **Mask R-CNN (2017):** Extends Faster R-CNN for **instance segmentation**.

Segmentation Output

- Segmentation models output **pixel-wise classification maps**.
- Each map corresponds to a **specific class**.
- Output maps can be the **same size as the input** or **smaller** (upsampled later).
- The maps are often processed with **softmax across channels** to produce per-pixel class probabilities.

Similarity and Clustering Heads

Similarity Measure (Siamese Network Style)

Shared backbone, one head.

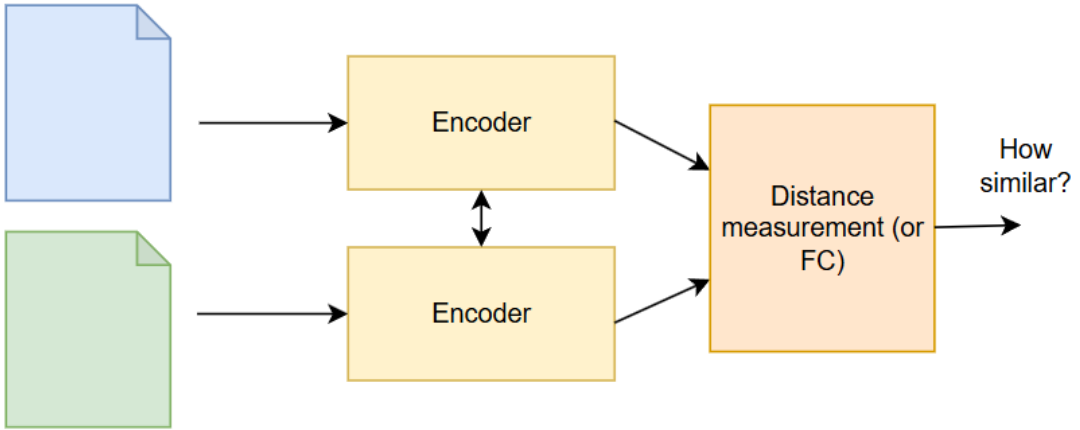
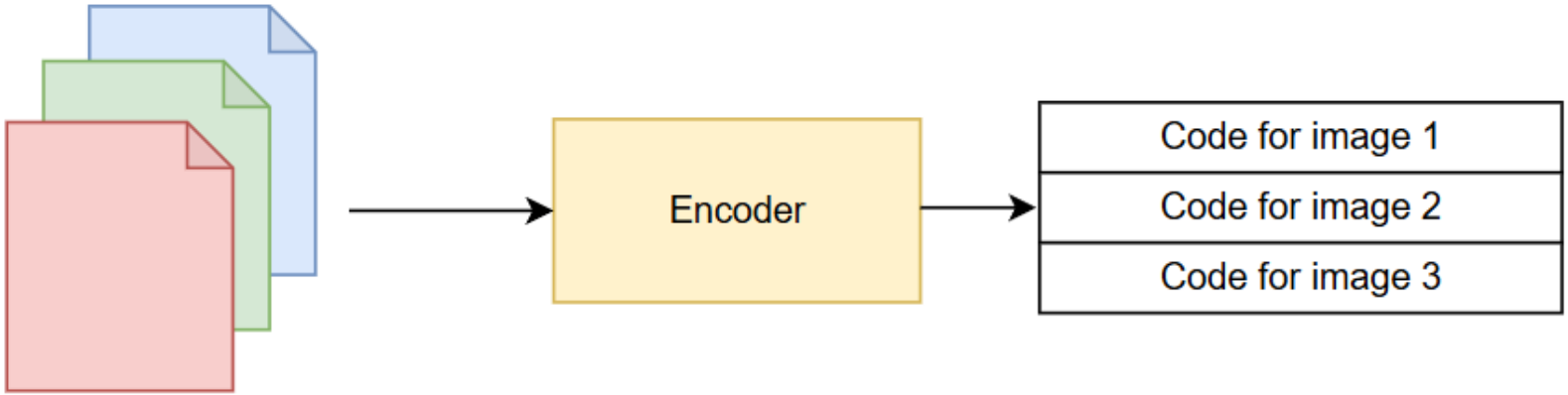


Image Encoding with Backbone

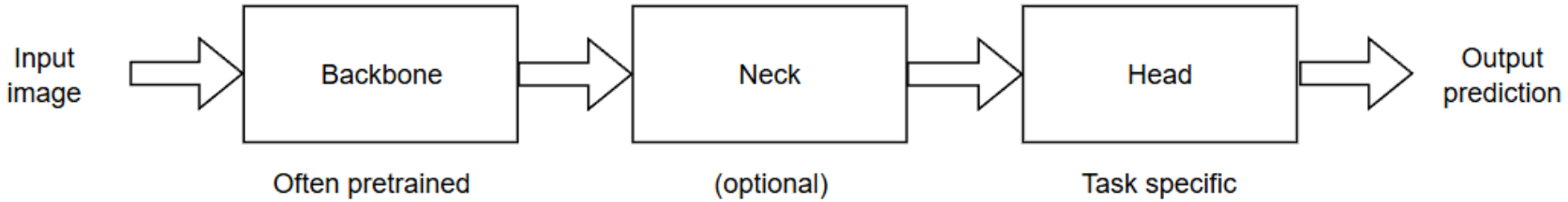
Codes can be used for storage, clustering, etc.



Necks and Attention

Neck

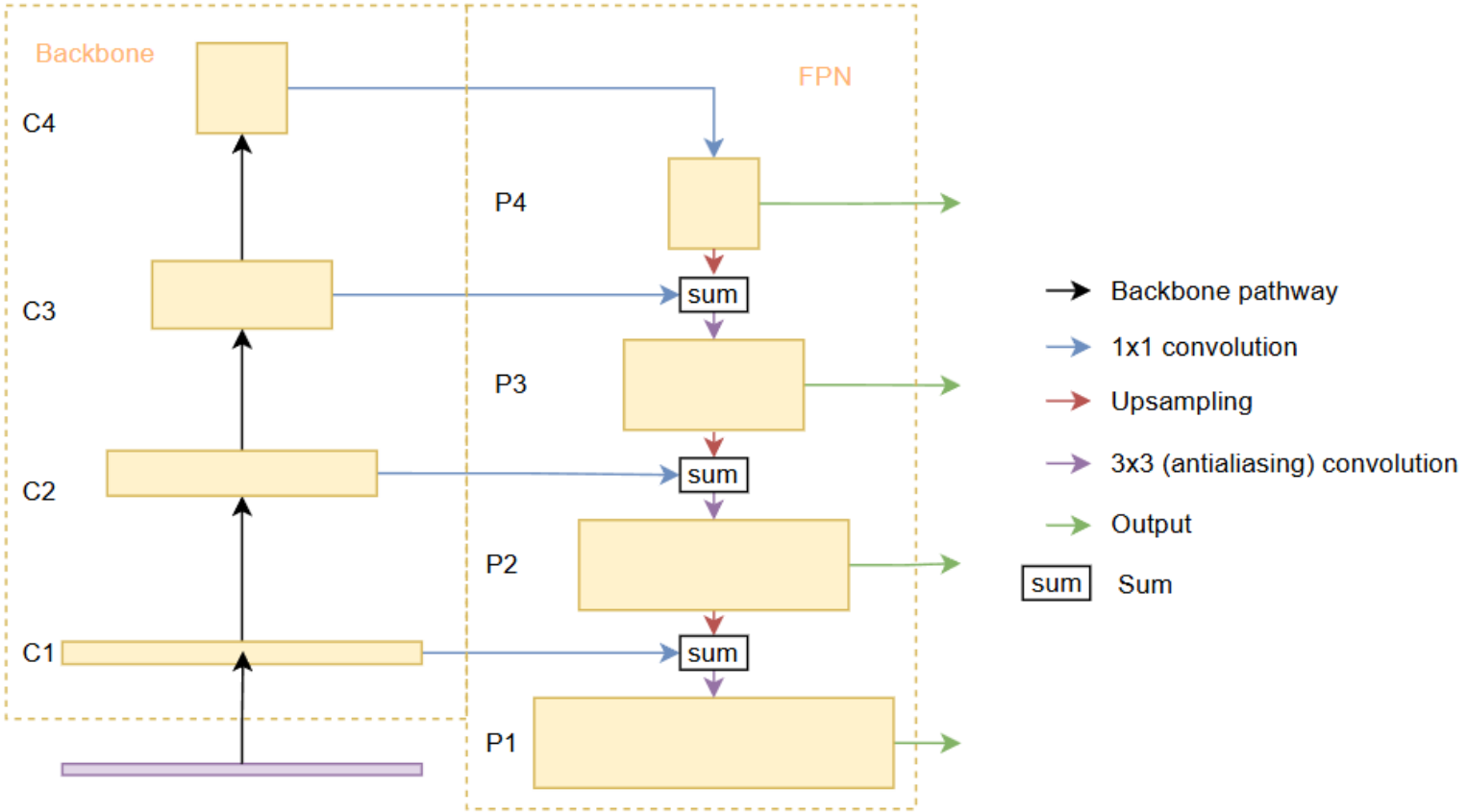
- **Purpose:** Refine, fuse, and reweight features from the backbone.
- **Key benefit:** Improves detection, segmentation, and other tasks, especially for **objects at different scales**.



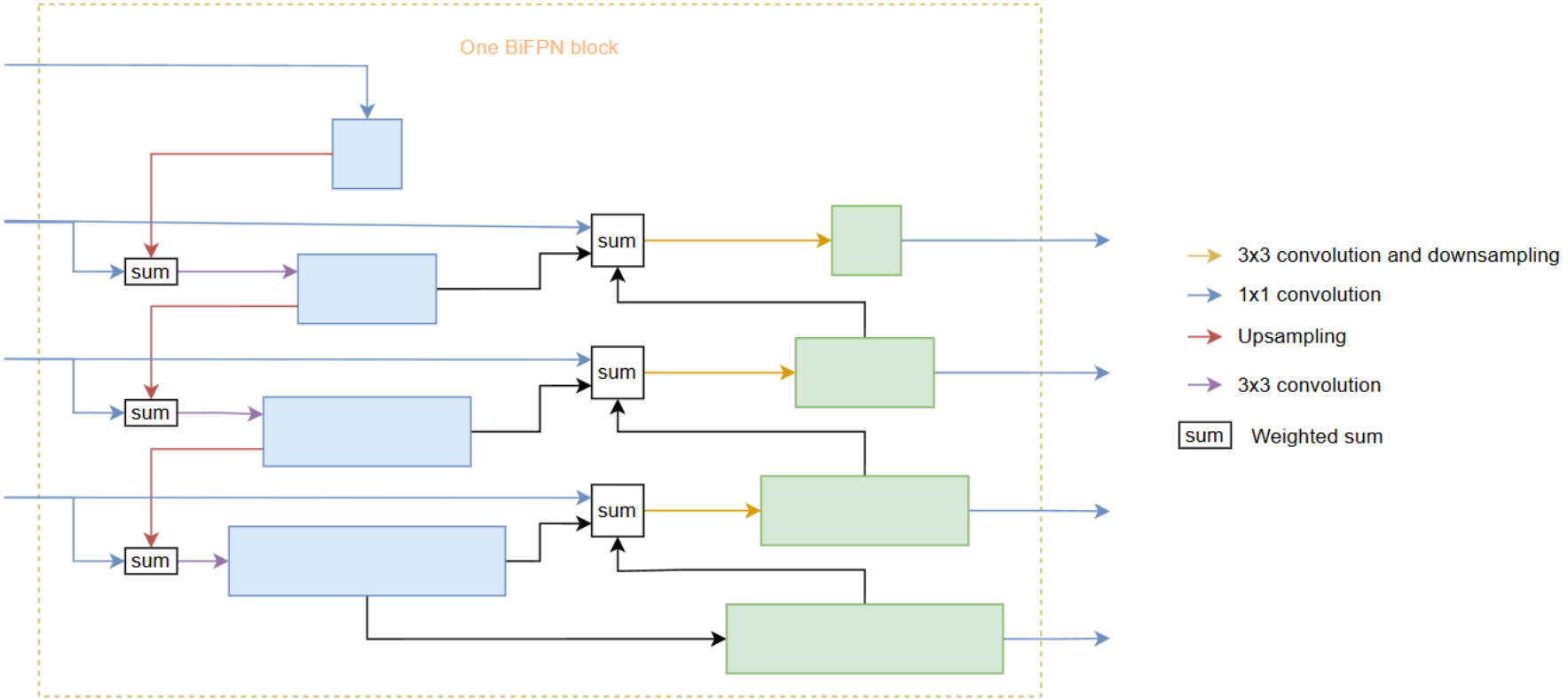
Popular Necks

- **Feature Pyramid Networks (FPN)**
- **PANet (Path Aggregation Network)**
- **BiFPN (Bidirectional FPN)**
- **SE blocks (Squeeze-and-Excitation)**: channel-wise attention.
- **CBAM (Convolutional Block Attention Module)**: spatial + channel attention.

FPN with Backbone

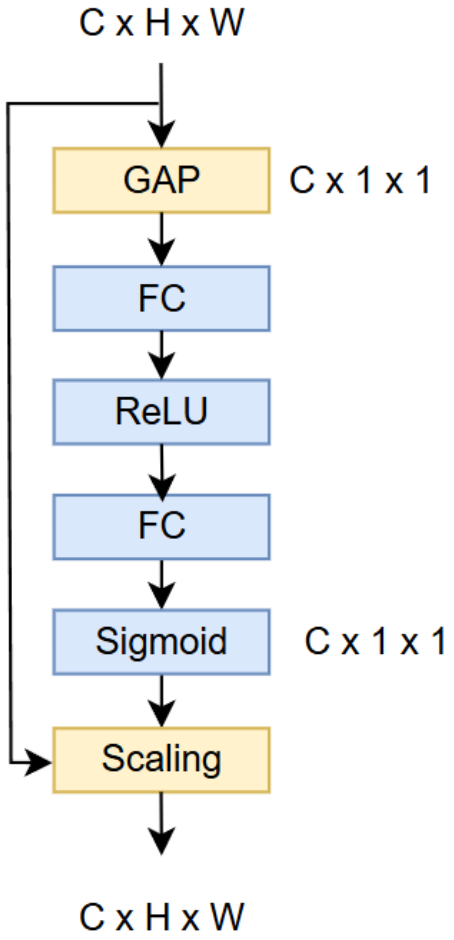


BiFPN


One BiFPN block
sum
sum
sum
sum
sum
3x3 convolution and downsampling
1x1 convolution
Upsampling
3x3 convolution
sum
Weighted sum

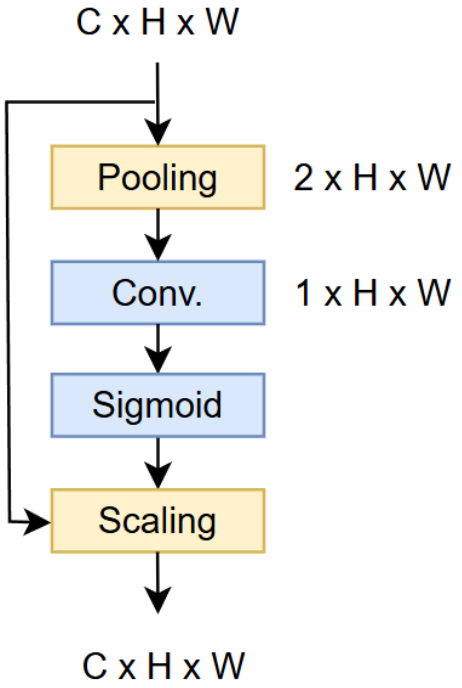
Squeeze and Excitation Block

- Weight for every channel is calculated with fully connected (FC) block.
- Channels are re-weighted (scaled according to the SE Block output.



Spatial Attention

- Both average and max pooling is used (two channels)
- Highlights the important regions



Convolutional Block Attention Module (CBAM)

- CBAM = **Channel Attention** (like SE) + **Spatial Attention**
- CBAM is useful because it helps a CNN learn what and where to focus — without adding much computational cost.
- Introduced in 2018

Practical Tips

How to Choose the Architecture

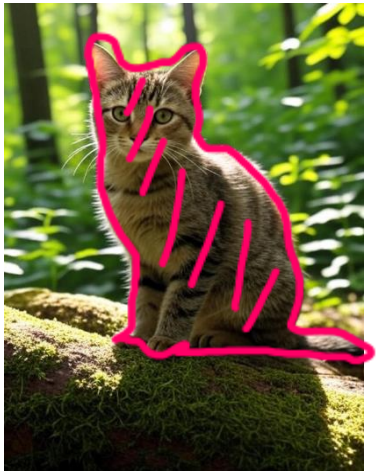
1. Understand your task
2. Choose backbone
3. Create the head
4. Add attention mechanisms if necesarry

Do you need **spatial resolution** (decoder)?

Classification
Is this a cat image?

Regression
How many cats?

Segmentation
What pixels are cat?



Detection
Where are the cats?

Choose a Backbone

- Dataset size?
- Size of the features?
- Computational capacity?
- Problem complexity?
- Image resolution?

Backbone Suggestions

- Small dataset - ResNet
- Small features - Modify ResNet, or use VGG
- Computational restrictions – MobileNet

Note: Use pretrained backbone whenever possible.

Build Your Head

- Use MLP head or CNN decoder
- Use more complex head if you have enough data
- Shape the output properly for your task (size and activation function)

Add Attention Mechanisms

Attention mechanisms can simplify the tasks for the head:

- Highlighting the most salient spatial regions
- Emphasizing the most informative feature channels
- Preserving fine details that might otherwise be lost

Conclusion

Final Points

- CNNs are composed of **backbones, necks, and heads.**
- **Major developments occurred between 2012 and 2018**, establishing the foundations of modern CNNs.
- New architectures are largely **extensions and refinements** of core ideas, focusing on **efficiency, multi-scale feature handling, and attention mechanisms**.

Elharrouss, Omar, et al. "Backbones-review: Feature extraction networks for deep learning and deep reinforcement learning approaches." arXiv preprint arXiv:2206.08016 (2022).

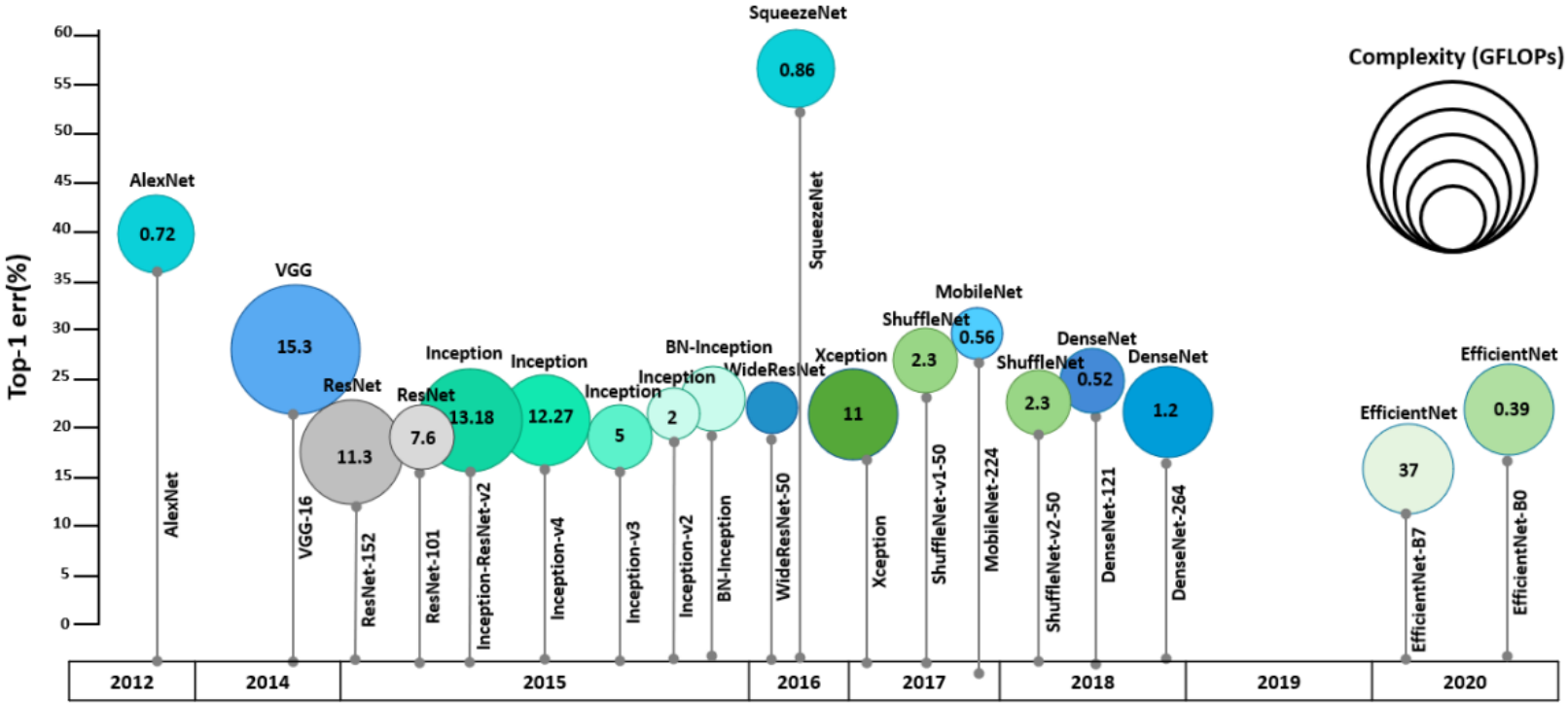


Figure 8: Timeline and performance accuracies of the the proposed networks on ImageNet.