Filters In CNN are same as weights in ANN, they are learnt by forward and backward propagation.

But in the pooling layer, that filter is fixed and doesn’t learn anything. It saves computation and detect only major features. Padding is not or rarely used in pooling because it zero and nothing will be gotten in max pooling because it try to get only max value.

There are two conventions, some consider conv layers is one and pooling as other layer, and some consider conv and pooling as single layer.

CNN are faster, every output only depends on few inputs like in filter of 3x3 one output location is depends only on 9 values but in fully connected layer, one output depend on every input.

LeNet, AlexNet, VGG

Very deep networks vanishing and exploding gradients, so in ResNet is better in deep networks. In this after every two layers, previous layers results are added in second next layer just before applying activation.

Network in Network is 1X1 convolution. Used as bottle neck layer in Inception to reduce computational cost.

In Inception module, we take multiple types of filters in a single layer and stack them up in single volume. In Inception network there are many inception modules.

In MobilNet Network. Depth wise separable convolution include Depth following pointwise convolution.

MobileNet v1 there is depthwise and pointwise convolution and in MobileNet v2 in addition to, there are residual connection and expansion layer. Expansion and pointwise convolutions are similar, difference is that, in expansion it increases the volume by using greater number of filters and in pointwise convolutions it reduces by using lesser filter quantity, it also called projection.

They are very less in computational cost.

EfficientNet.

ImageNet is a dataset that has almost 14 million images and 1000 classes.

Ensembling, training several different networks and averaging their y hat or output.

**Sliding Window** is a **brute-force method** for object detection. It moves a fixed-size window across the image and applies a classifier (e.g., CNN) to check for objects.

### **How It Works:**

1. Start with a **small window**.
2. Move the window **horizontally** across the image.
3. After one row, move **vertically down**.
4. Repeat at **different scales** (multi-scale detection).

### **Disadvantages:**

* **Computationally expensive** ⚡
* **Slow** for real-time applications.

**Anchor Boxes** are **predefined bounding boxes** of different sizes and aspect ratios, used in **region-based object detection** methods like **YOLO and Faster R-CNN**.

### **Why Use Anchor Boxes?**

* Objects have **different sizes** and **shapes**.
* Helps in detecting **multiple objects** in the same location.
* Avoids **misalignment problems**.

**Non-Maximum Suppression** **(NMS)** is a technique used in **object detection** to remove **multiple overlapping bounding boxes** and keep only the most relevant one.

Object detection models (like **YOLO, Faster R-CNN, SSD**) generate **multiple bounding boxes** for the same object.  
NMS ensures that:

* We keep **only one bounding box** per detected object.
* We **eliminate redundant boxes** with lower confidence scores.

**YOLO** is a **real-time object detection** algorithm that processes an image in a **single forward pass** of a neural network.

### **2️⃣ Each Grid Cell Predicts:**

For each grid cell, YOLO predicts:  
✔ **Bounding Boxes (Bx, By, Bw, Bh)** → Adjusted from Anchor Boxes  
✔ **Confidence Score (Pr(Object) × IoU)** → How sure the model is about an object  
✔ **Class Probabilities (P1, P2, ..., Pn)** → Probability for each object category

💡 **Final Predictions = Grid Cell Output × Number of Grid Cells**

## **🔹 Example: How It Works in YOLO**

1. The image is divided into a **grid**.
2. Each grid cell has **multiple anchor boxes** (e.g., square, tall, wide).
3. The model **adjusts the best-fitting anchor box** to match the actual object.
4. The refined box becomes the **final bounding box**.
5. **Non-Maximum Suppression (NMS)** removes redundant boxes.

## **🔹 Architecture of YOLO**

YOLO uses a **single convolutional neural network (CNN)** to process the image and detect objects.  
✔ **Backbone CNN (e.g., Darknet, CSPNet)** extracts features.  
✔ **Detection Head** generates bounding boxes & class probabilities.

| **Version** | **Improvements** |
| --- | --- |

|  |  |
| --- | --- |
| YOLOv1 | First version, grid-based prediction |

|  |  |
| --- | --- |
| YOLOv2 | Improved accuracy, added anchor boxes |

|  |  |
| --- | --- |
| YOLOv3 | Multi-scale detection, Darknet-53 backbone |

|  |  |
| --- | --- |
| YOLOv4 | CSPDarknet53, faster & more accurate |

|  |  |
| --- | --- |
| YOLOv5 | PyTorch-based, easier training & deployment |

|  |  |
| --- | --- |
| YOLOv8 | State-of-the-art, transformer-based |

**Semantic Segmentation** is a **pixel-wise classification task** where each pixel in an image is assigned a **class label** (e.g., "road," "car," "tree," etc.). Unlike object detection, which provides **bounding boxes**, semantic segmentation provides a **detailed mask** of each object.

📌 What is Transpose Convolution?

Transpose Convolution (also called **Deconvolution**) is a type of convolution operation used to **upsample** feature maps in neural networks. It is the **opposite of a standard convolution**, where instead of reducing spatial dimensions, it increases them.

💡 **Used in:**  
✔ **Semantic Segmentation** (U-Net, DeepLabV3+)  
✔ **Image Generation** (GANs)  
✔ **Super-Resolution**

1️⃣ Standard Convolution (Downsampling)

In a normal **convolution operation**, a kernel (filter) slides over the input, **reducing** the spatial dimensions of the feature map.

### **2️⃣ Transpose Convolution (Upsampling)**

Transpose convolution **reverses** the downsampling process by expanding the input size. It **spreads the values** of the smaller feature map into a **larger output space**.

## **📌 1. Face Recognition (Identification)**

✔ **Definition:** Identifies **who** a person is from a set of known faces.  
✔ **Task:** Compares a given face to a database of multiple faces and finds the best match.  
✔ **Example:** Unlocking your phone using Face ID, tagging friends in Facebook photos.  
✔ **Algorithm:** Uses **feature extraction & classification** (e.g., ResNet, FaceNet, ArcFace).

💡 **Example Flow:**  
1️⃣ Input: A face image  
2️⃣ Extract facial features (embedding)  
3️⃣ Compare with stored embeddings in a database  
4️⃣ Return the best match (or "unknown" if no match is found)

✔ **Real-world use cases:**  
🔹 Security systems (CCTV)  
🔹 Airport check-ins  
🔹 Biometric attendance systems

## **📌 2. Face Verification (Authentication)**

✔ **Definition:** Confirms whether **two face images** belong to the same person.  
✔ **Task:** A binary classification problem: **"Is this the same person?" (Yes/No)**  
✔ **Example:** Verifying identity for banking apps or login authentication.  
✔ **Algorithm:** Uses **similarity measurement** (e.g., Cosine Similarity, Euclidean Distance).

💡 **Example Flow:**  
1️⃣ Input: Two face images  
2️⃣ Extract facial embeddings using a model  
3️⃣ Compute the similarity score (e.g., cosine similarity)  
4️⃣ If similarity > threshold → "Same person," else "Different person"

✔ **Real-world use cases:**  
🔹 Two-factor authentication  
🔹 Login verification (e.g., Microsoft, Apple Face ID)  
🔹 Border control (passport verification)

## **🔹 Summary of Challenges & Solutions**

| **Challenge** | **Problem** | **Solution** |
| --- | --- | --- |
| **Filters** | Too many filters increase computation | Use filter pruning, optimal size (3×3 is standard) |
| **Pooling** | Loss of spatial information | Use strided convolutions instead of pooling |
| **Padding** | Zero-padding introduces artifacts | Use reflection padding |
| **Training** | Overfitting, slow convergence | Use data augmentation, dropout, batch norm |
| **Deep CNNs** | Vanishing gradients | Use ResNet (skip connections) |
| **Memory Usage** | High RAM usage | Use quantization, transfer learning |
| **Object Detection** | Small objects ignored | Use Feature Pyramid Networks (FPNs) |
| **Overlapping Boxes** | Duplicates in detection | Use Non-Maximum Suppression (NMS) |
| **Deployment** | Too slow for real-time | Use MobileNet, EfficientNet, model pruning |

### **8️⃣ What is the difference between FaceNet and DeepFace?**

| **Feature** | **FaceNet (Google)** | **DeepFace (Facebook)** |
| --- | --- | --- |
| **Model Type** | Triplet Loss-based | Deep Learning-based |
| **Training Data** | Uses triplet loss to learn embeddings | Trained on millions of faces |
| **Accuracy** | High | Also high |
| **Embedding Output** | 128-dimensional vector | 4096-dimensional vector |
| **Use Case** | Face verification, clustering, recognition | Face verification (mainly) |

### **9️⃣ How can you speed up inference for real-time applications?**

🔹 **1. Model Optimization**  
✅ **Quantization** – Convert 32-bit floating point weights to **8-bit integers** (reduces model size).  
✅ **Pruning** – Remove unnecessary weights (reduces computation).  
✅ **Knowledge Distillation** – Train a smaller "student" model to mimic a larger "teacher" model.

🔹 **2. Hardware Acceleration**  
✅ Use **TensorRT** (NVIDIA) for GPU-based inference.  
✅ Use **OpenVINO** (Intel) for CPU optimization.  
✅ Use **ONNX Runtime** to optimize model deployment across different platforms.

🔹 **3. Efficient Architectures**  
✅ Use **MobileNetV2, EfficientNet, or YOLOv5** instead of heavier models like ResNet-101.  
✅ Reduce input image size to minimize computational load.

🔹 **4. Batch Processing & Parallelization**  
✅ Use **batch inference** instead of processing single images one by one.  
✅ Use **multi-threading** to parallelize inference on multiple inputs.

📌 **Example: Running optimized YOLOv5 on TensorRT**