DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SUBJECT CODE: 23CSX502 **SEM: VI**

SUBJECT NAME: DEEP LEARNING AND NEURAL NETWORKS **YEAR: III**

UNIT II- CONVOLUTION NEURAL NETWORKS

MODULE

Foundations of Convolutional Neural Networks – Architecture – Simple Convolution Network – Deep Convolutional Models –AlexNet, GoogLeNet, ResNet, region-Based CNNs

2.1 FOUNDATIONS OF CONVOLUTIONAL NEURAL NETWORKS:

Human vision quickly identifies objects, perceives depth, contours, and separates objects from backgrounds. The eyes capture raw pixel data, which the brain converts into primitives like lines and shapes, enabling recognition (e.g., a house cat). Neurons in the eye capture light, pre-process it, and send it to the visual cortex for analysis. Convolutional Neural Networks (CNNs) mimic this process to create effective computer vision models. Digital images are matrices of pixel intensities. Grayscale images have one intensity per pixel; color images have three channels (RGB), forming a 3D volume (width, height, depth). Neighbouring pixels are spatially correlated, and CNNs leverage this structure to extract meaningful features automatically.

2.1.1 Why CNNs Are Needed for Image Data & Limitations of Traditional ANNs

Images are high-dimensional, spatially structured data. Each image is made up of pixels, and neighbouring pixels are strongly related to each other. For example, in a photo of a cat, nearby pixels together form edges, textures, and shapes. Preserving this spatial relationship is very important for understanding visual content.

Limitations of Traditional Artificial Neural Networks (ANNs)

Traditional ANNs and fully connected networks treat every input neuron as independent and connect it to every neuron in the next layer. When applied to images, this creates several problems:

- **Huge Number of Parameters**: Even a small colour image (e.g., 200 × 200 × 3) has 120,000 input values. Connecting each of these to neurons results in millions of parameters, making the model:
  - Very slow
  - Memory intensive
  - Hard to train
- **Loss of Spatial Information:** Fully connected networks do not preserve spatial structure.
  - A pixel next to another pixel is treated the same as a pixel far away.
  - This makes it difficult to learn local patterns such as edges, corners, and textures.

- **Overfitting Problem:** Because of too many parameters, the network tends to memorize training data instead of learning meaningful features, especially when training data is limited.
- **Manual Feature Extraction**: Earlier systems required hand-crafted features (edges, shapes, textures).
    - This process is time-consuming
    - Task-specific
    - Poor at generalization
- **Poor Scalability**: Fully connected networks work for small images like MNIST (28 × 28), but they fail for real-world large, high-resolution images.

Why Convolutional Neural Networks (CNNs) Are Needed

CNNs were designed specifically to overcome these limitations and work efficiently with image data. They introduce three key ideas:

- Local Receptive Fields

  Instead of looking at the entire image at once, each neuron focuses on a small region (e.g., 3×3 or 5×5).

  Benefits:

    - Captures local patterns like edges and corners
    - Preserves spatial relationships
    - Reduces number of connections
- Weight Sharing

  The same filter (kernel) is applied across different parts of the image.

  Benefits:

    - Detects the same feature anywhere in the image
    - Drastically reduces parameters
    - Improves efficiency
- Pooling: Pooling reduces the size of feature maps by summarizing small regions.

  Benefits:

    - Reduces computation
    - Prevents overfitting
    - Provides translation tolerance
    - Keeps important information

2.1.2 CORE CONCEPTS OF CNNS:

2.1.2.1 Hierarchical Feature Learning in CNNs

Hierarchical feature learning enables Convolutional Neural Networks (CNNs) to build a structured and meaningful understanding of images by learning features in a layered manner.

Instead of recognizing everything at once, CNNs extract information step by step, moving from simple visual patterns to complex objects. Each layer builds upon the output of the previous layer, allowing the network to recognize intricate patterns that shallow models or manual feature extraction methods cannot.

A. Early Layers – Low-Level Features

The first few layers of a CNN focus on detecting **basic visual elements**, such as:

- Edges
- Lines
- Corners
- Simple textures
- Colour contrasts

These layers act like basic vision detectors. They do not understand objects but only capture simple local patterns from small regions of the image.

B. Middle Layers – Mid-Level Features

The middle layers combine the simple patterns detected by early layers to form more meaningful structures, such as:

- Curves
- Shapes
- Repeated textures
- Parts of objects (eyes, wheels, windows, etc.)

These layers begin forming relationships between features, helping the network understand different parts of an object.

C. Deep Layers – High-Level Features

The deeper layers combine mid-level features to recognize:

- Complete objects (cars, faces, animals, people)
- Object categories
- Complex visual patterns
- Semantic meaning (what the object is)

At this stage, the CNN no longer looks for simple shapes but instead understands **what the object represents**.

2.1.2.2 TRANSLATION INVARIANCE VS TRANSLATION EQUIVARIANCE IN CNNS

Translation Equivariance: A system is said to be translation equivariant if a shift in the input causes a corresponding shift in the output.

- Convolution layers in CNNs are translation equivariant
- When an object in an image moves, the feature map activation also moves
- The detected pattern remains the same, but its location changes

Example: If a face moves from left to right in an image, the activation detecting the face also shifts from left to right in the feature map.

Translation Invariance

A system is translation invariant if shifting the input does not change the output.

- The network produces the same response regardless of object position
- Particularly useful for image classification tasks

Example: A face classifier outputs "FACE PRESENT" irrespective of where the face appears in the image.

How CNNs Achieve Translation Invariance

- **Convolution Layers:** Provide translation equivariance, not invariance. Feature maps move with the object.
- **Pooling Layers (especially Max Pooling)**
  Reduce spatial resolution and ignore exact position within a local region, providing local translation invariance.
- **Deeper Layers with Repeated Pooling** Gradually increase robustness to larger spatial shifts.
- **Fully Connected Layers or Global Pooling** Discard spatial location information and convert equivariant feature maps into approximately invariant outputs.

2.2 ARCHITECTURE

CNN architecture is designed as a structured flow of CNN layers, convolution, activation, pooling, fully connected, and output, to analyse visual information with exceptional accuracy. This architecture of CNN ensures that every stage transforms raw image pixels into rich feature representations, enabling efficient pattern discovery and class prediction.

The progressive and hierarchical learning of features, from simple patterns to complex structures, allows CNNs to recognize important patterns in images and accurately identify what the image represents. This capability enables CNNs to achieve high accuracy in tasks such as image classification, object detection, and medical image analysis, especially in critical applications where reliability and precision are essential.

2.2.1 ARCHITECTURE OF CNN: AN OVERVIEW

CNNs work differently from traditional neural network architectures. Older models connect every input to every neuron. This creates heavy computation. A CNN avoids this by scanning small regions with filters. This reduces parameters and improves speed. It also helps the model notice the same pattern even when it appears in a different part of the image.

CNNs are widely used because they understand spatial patterns. They can read medical scans, identify faces, classify images, and detect objects. These tasks depend on local details, and CNNs capture those details well. Most CNN designs share a common set of parts:

- Filters
- Activation units
- Down sampling layers

- Dense layers
- Output functions

Each of these layers in CNN plays a specific role, contributing to effective feature extraction and accurate predictions within the CNN architecture in deep learning. Below is a CNN architecture diagram in deep learning:

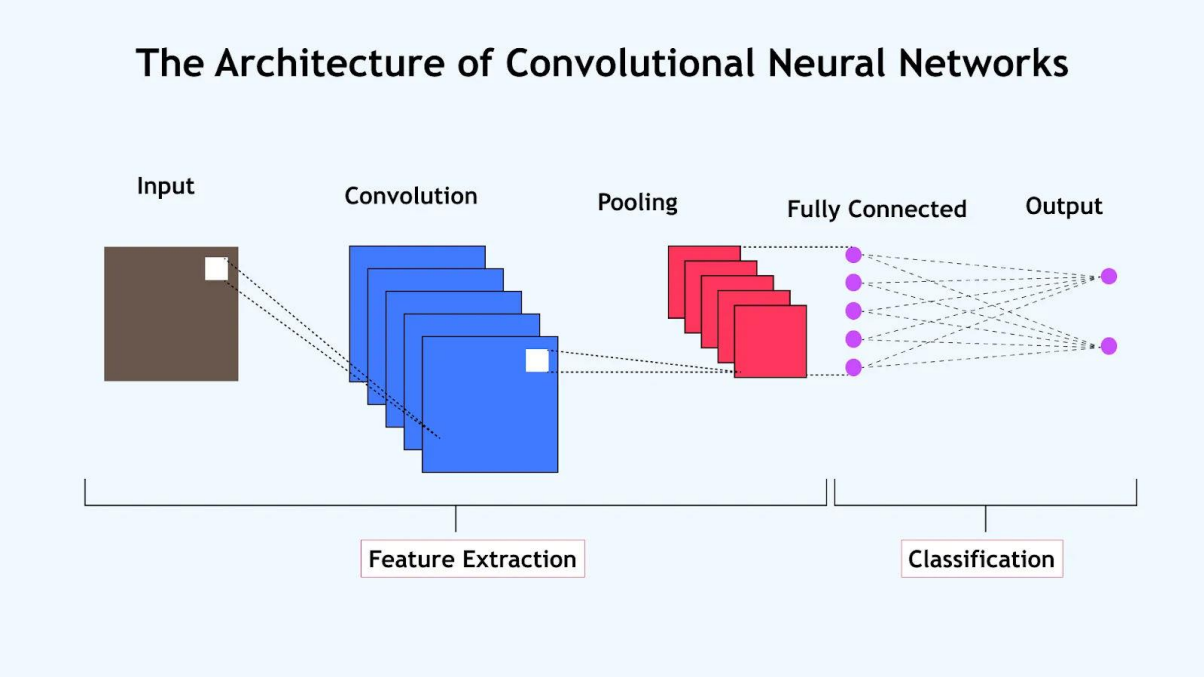


Figure 2.1: The Architecture of Convolutional Neural Network

The above Figure 2.1 shows:

Component	Purpose
Filters	Extract edges, textures, and shapes
Activation units	Add nonlinearity to learn complex patterns
Down sampling	Reduce data size while keeping key details
Dense layers	Combine features for final decisions
Output functions	Convert scores into probabilities

Table 2.1 Key Components and Their Purposes in a CNN

CNN builds understanding step by step. The layers of CNN sharpen what the model sees. Simple features grow into meaningful patterns. This is why the CNN architecture remains a strong choice for most vision problems.

2.2.2 Five Core CNN Layers in Architecture

A basic architecture of convolutional neural network works in five clear steps. Each layer in CNN has a simple job. Together, they turn raw images into class scores. These layers of CNN appear in almost every CNN you study, no matter how small or deep.

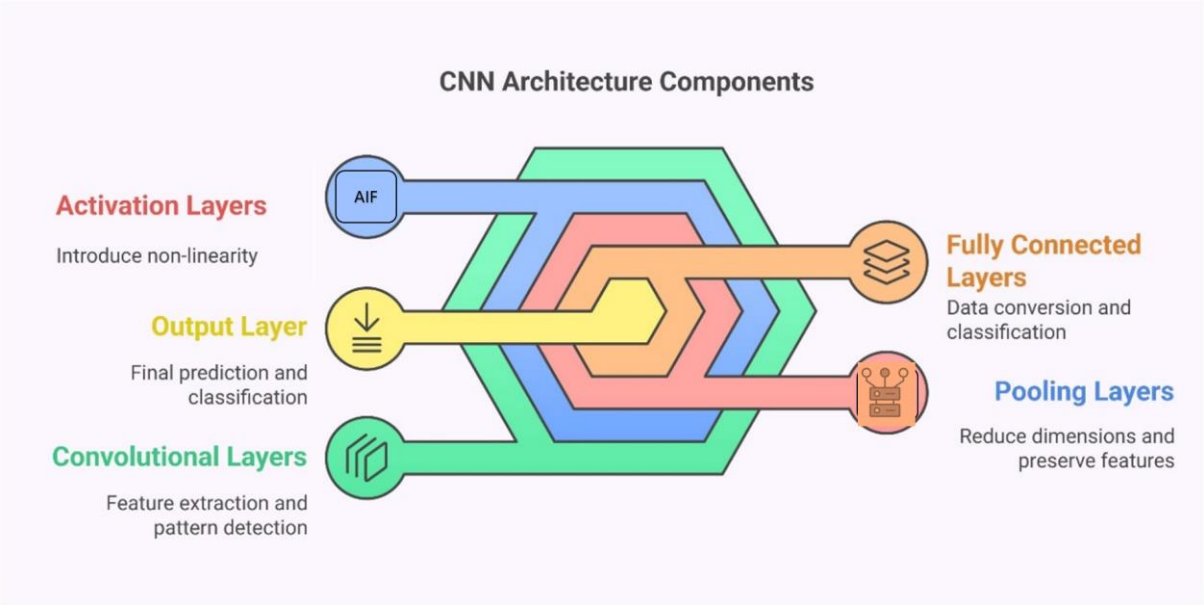


Figure 2.2: CNN Architecture Components

A) Convolution Layer

The convolution layer is the entry point of the simple CNN architecture. Instead of looking at the whole image at once, the model focuses on small blocks of pixels. This helps it notice local patterns before building a full understanding of the image. It uses multiple filters, and each one learns a different visual pattern such as an edge, corner, or texture.

How It Works:

- A filter slides across the image one small step at a time. At each step, it multiplies its values with the pixel values under it.
- The results are added and stored in a new grid called a feature map. This feature map shows where the learned pattern appears in the image.
- During training, the CNN adjusts the values inside each filter. Over time, the filters become good at detecting useful patterns. Early filters catch simple shapes. Deeper filters catch more complex textures and visual details.

Key actions in this layer

- Scanning small regions to understand local structure
- Extracting edges, corners, textures, and simple shapes
- Producing feature maps that highlight important visual details

B) Activation Layer

After the convolution layer picks up basic patterns, the model needs a way to understand more complex shapes and relationships. The activation layer makes this possible by adding nonlinearity. Without this step, an architecture of CNN in deep learning would behave like a simple linear model and would miss many details found in real images.

ReLU is the most widely used activation in convolutional neural network architecture. It keeps positive values and removes negative ones. This helps the model focus on strong signals and

train faster. Sigmoid and Tanh are also used, mainly when the model needs smoother transitions between values.

How It Works

- The activation function is applied to every value in the feature map.
- If you use ReLU, negative values become zero while positive values stay as they are.
- This simple step helps the model learn curves, textures, and layered patterns.

Common activation functions

- ReLU
- Sigmoid
- Tanh

C). Pooling Layer

The pooling layer reduces the size of the feature maps created by the previous convolutional neural network layers. It selects the most important values from small regions and leaves out the rest. This keeps essential details while removing noise and extra information. The result is a lighter and faster model that still retains the core features needed for learning.

Pooling also helps the CNN stay stable when objects shift slightly within an image. A feature that appears a little to the left or right will still be captured after pooling. This makes the model more reliable and improves its ability to generalize.

How It Works

- The layer divides each feature map into small blocks.
- Depending on the pooling type, it either picks the strongest value or takes the average of the block.
- This reduces the spatial size but keeps the important signal intact.

Two common types

- Max pooling: Strongest value in the region
- Average pooling: Average value in the region

D) Fully Connected Layer

After the earlier CNN layers extract and refine features, the CNN needs a way to combine everything into a final understanding. This is where the fully connected layer comes in. The model first flattens all feature maps into one long vector. This vector represents every detail learned so far.

The flattened vector is then passed into dense units. Each unit connects to every value in the vector. These connections help the model understand how different features relate to each other. Early layers of CNN focus on edges and shapes. This layer focuses on the big picture, such as identifying whether the image shows a digit, an object, or a face.

How It Works

- Each dense unit receives all input values and multiplies them with learned weights.

- The results are added and then passed through an activation function.
- This helps the model capture high-level patterns that simpler layers cannot detect.

What this layer handles

- Combining all learned features
- Understanding high-level patterns
- Passing values to the final stage

E) Output Layer

The output layer is the final step in the architecture of Convolutional Neural Network. This is where the model makes its decision. After the fully connected layer processes all features, the output layer converts those values into clear probabilities. These probabilities tell you which class the model believes the image belongs to.

For multi-class problems, the model uses Softmax. It assigns a probability to every class and ensures the values add up to one. For binary problems, the model uses Sigmoid. It produces a single probability that represents a yes or no outcome.

How It Works

- The final values from the dense units are passed into the chosen output function.
- The function transforms raw numbers into interpretable scores.
- The class with the highest score becomes the predicted label.

Role of this layer

- Producing class scores
- Returning the final prediction

These five layers in CNN complete the full flow of a simple CNN architecture. The model begins by detecting simple shapes and ends by delivering a clear prediction.

2.2.3 Variations and Extensions of Basic CNN Architecture

A basic architecture gives a strong starting point, but many tasks need deeper or more efficient designs. Over the years, several variations have been introduced to improve feature learning, speed, and accuracy.

A. Deep CNNs

Deep CNNs add more convolution and pooling layers. More convolutional neural network layers let the model learn detailed and complex patterns. Early layers capture edges, while deeper layers understand shapes and object parts. This design is common in large image classification tasks.

B. Residual Networks

Residual networks help when models grow very deep. They include skip connections that pass information forward without losing it. This makes training stable and prevents the model from forgetting earlier patterns.

3. Dilated Convolutions

Dilated convolutions widen the filter's field of view. They help the model capture broader context without increasing parameters. This works well for tasks like segmentation and depth estimation.

4. Depth wise Separable Convolutions

This variation breaks the convolution process into two smaller steps. It reduces computation and keeps the model fast. Lightweight CNNs and mobile models often use this approach.

Comparison Table

Variation	Main Benefit
Deep CNNs	Better feature depth
Residual networks	Stable training for deep models
Dilated convolutions	Wider context understanding
Depthwise separable convolutions	Faster and lighter models

Table 2.2: Variations of Convolutional Neural Networks and Their Main Benefits

These extensions build on the core architecture and make it flexible for different needs, from mobile devices to large-scale computer vision systems.

2.2.4 Real-World Applications of CNN Architecture

Application Area	How CNNs Are Used
Image classification	Classifies images into labels like animals, objects, or scenes by learning visual patterns.
Object detection	Locates multiple objects in an image and draws bounding boxes to identify each one.
Medical imaging	Helps detect tumors, fractures, and diseases by analyzing X-rays, MRIs, and CT scans.
Face recognition	Identifies and verifies faces in security systems, phones, and attendance tools.
Autonomous systems	Reads roads, signs, pedestrians, and obstacles for safe navigation.
Text extraction	Converts handwritten or printed text into digital text in OCR systems.
Quality inspection	Finds product defects on manufacturing lines by spotting texture or shape irregularities.

Table 2.3: Application Areas of CNNs and Their Practical Uses

2.3 Simple Convolution Network

In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyse visual imagery. The CNN architecture uses a special technique called Convolution instead of relying solely on matrix multiplications like traditional neural networks. Convolutional networks use a process called convolution, which combines two functions to show how one changes the shape of the other.

But we don't need to go behind the mathematics part to understand what a CNN is or how it works. The bottom line is that the role of the convolutional networks is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction.

2.3.1 Working of Convolutional Neural Network (CNN)

The basic architecture of a simple Convolutional Neural Network (CNN) includes an input image, convolution operations, feature maps, pooling layers, and outputs passed to subsequent stages. The CNN pipeline begins by applying convolutional layers to detect fundamental features like edges and textures. Activation functions (e.g., ReLU) add non-linearity, enabling the network to model complex patterns. Pooling layers down sample feature maps, reducing computational load and controlling overfitting while retaining essential information. These convolution and pooling steps are stacked repeatedly to progressively capture higher-level abstractions such as shapes and objects. Finally, fully connected layers integrate all learned features, and the output layer uses Softmax or Sigmoid functions to generate class probabilities, facilitating classification tasks. Additionally, techniques like dropout and batch normalization are often incorporated to improve training stability and generalization performance.

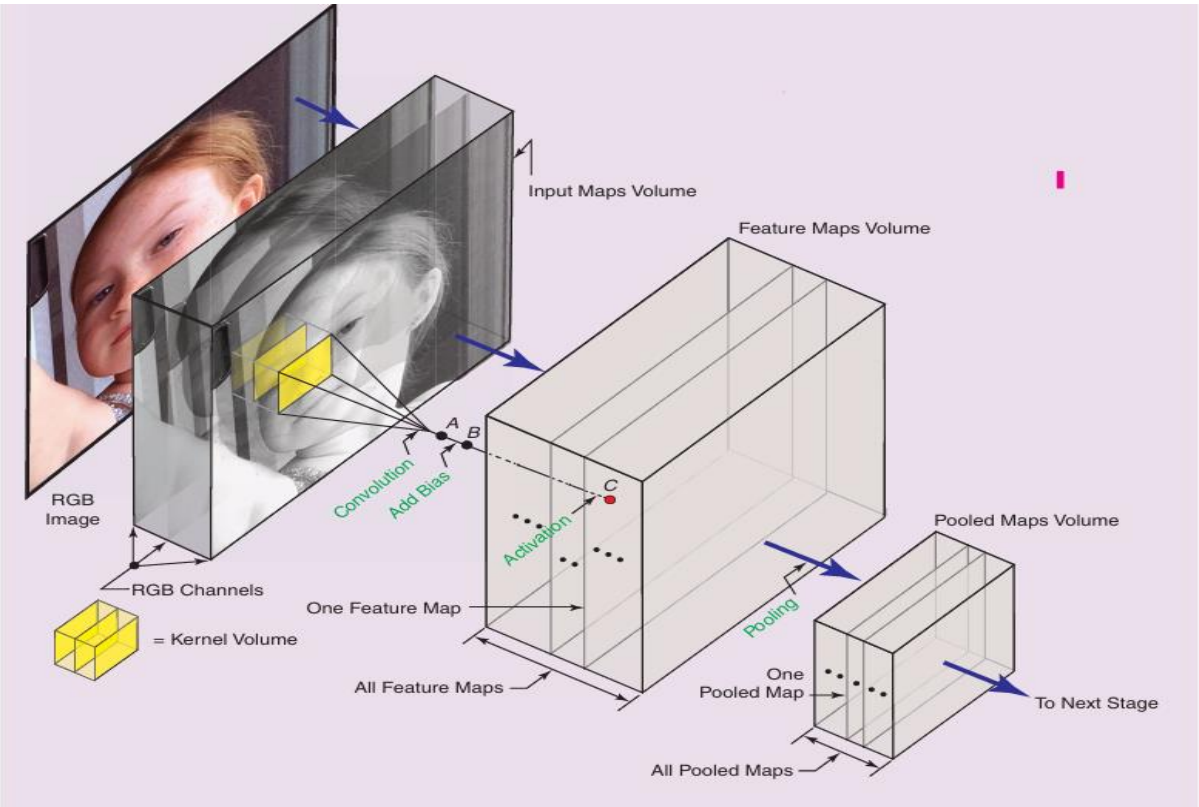


Figure 2.3: Basic Architecture and Workflow of a Convolutional Neural Network

How Does CNN Work?

Before we go to the working of Convolutional neural networks (CNN), let's cover the basics, such as what an image is and how it is represented. An RGB image is nothing but a matrix of pixel values having three planes, whereas a grayscale image is the same, but it has a single plane. Take a look at this image to understand more.

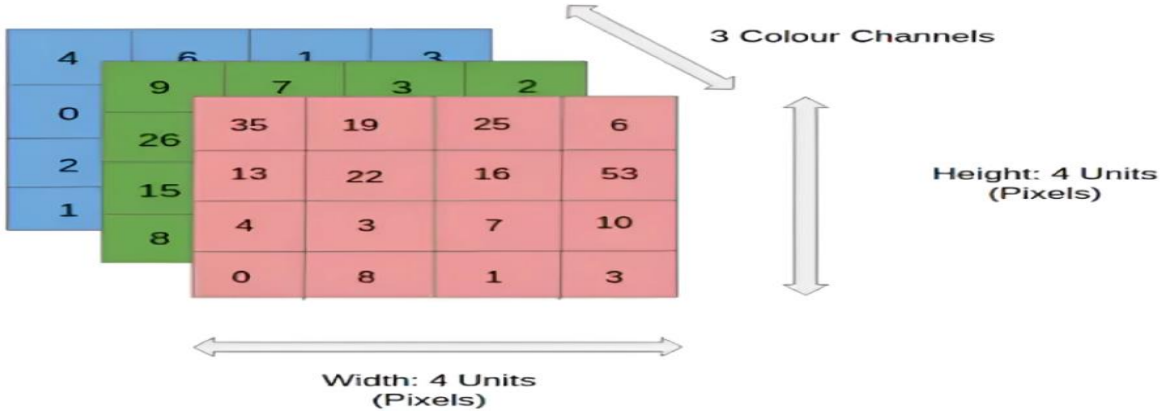


Figure 2.4: **Representation of a Color Image as a 3D Pixel Matrix with RGB Channels**

For simplicity, let's stick with grayscale images as we try to understand how CNNs work.

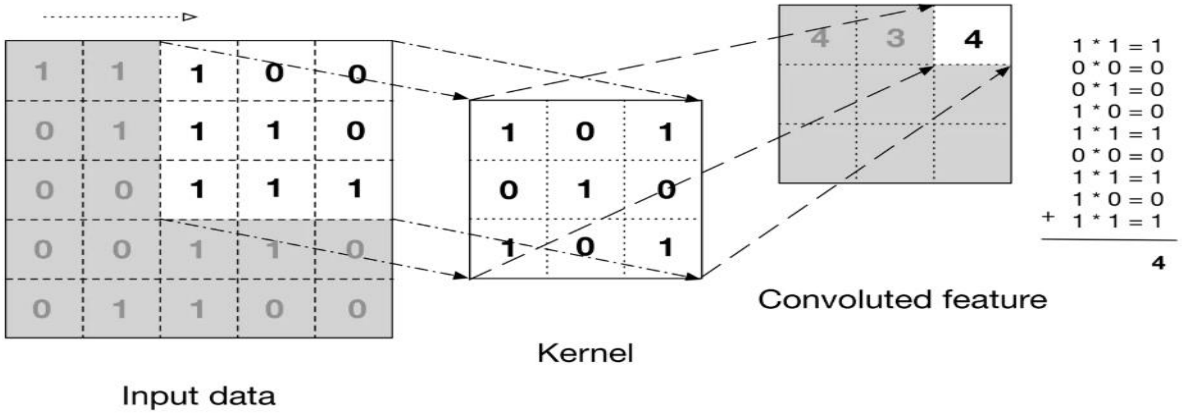


Figure 2.5: Representation of a grayscale image

The 3 parts in this image?

- **Input data** (left) → the image (numbers)
- **Kernel / Filter** (middle) → small box of numbers
- **Convoluted feature** (right) → output

Step 1: Input Data (Left) This is your image represented as **numbers**.

Example:

1 0 0

1 1 0

1 1 1

... Each number = pixel value.

Step 2: Kernel / Filter (Middle) This small box is called a **kernel** or **filter**.

Example shown:

1 0 1

0 1 0

1 0 1

This filter is designed to detect a **specific pattern**.

Step 3: Sliding the Filter The filter **slides** over the image like a scanner. At each position, CNN does: **Multiply → Add → Get one number**

Example of One Calculation: From the right side of your image:

1 * 1 = 1

0 * 0 = 0

0 * 1 = 0

1 * 0 = 0

1 * 1 = 1

0 * 0 = 0

1 * 1 = 1

1 * 0 = 0

1 * 1 = 1

Total = 4

So the output is **4**.

Step 4: Output (Convoluted Feature)

Each time the filter moves, it produces **one number**.

All these numbers form a new matrix called **Feature Map**

This shows **where the pattern exists** in the image.

Why This Finds Edges & Patterns?

- Edges = sudden number changes
- Shapes = combination of edges
- Objects = combination of shapes

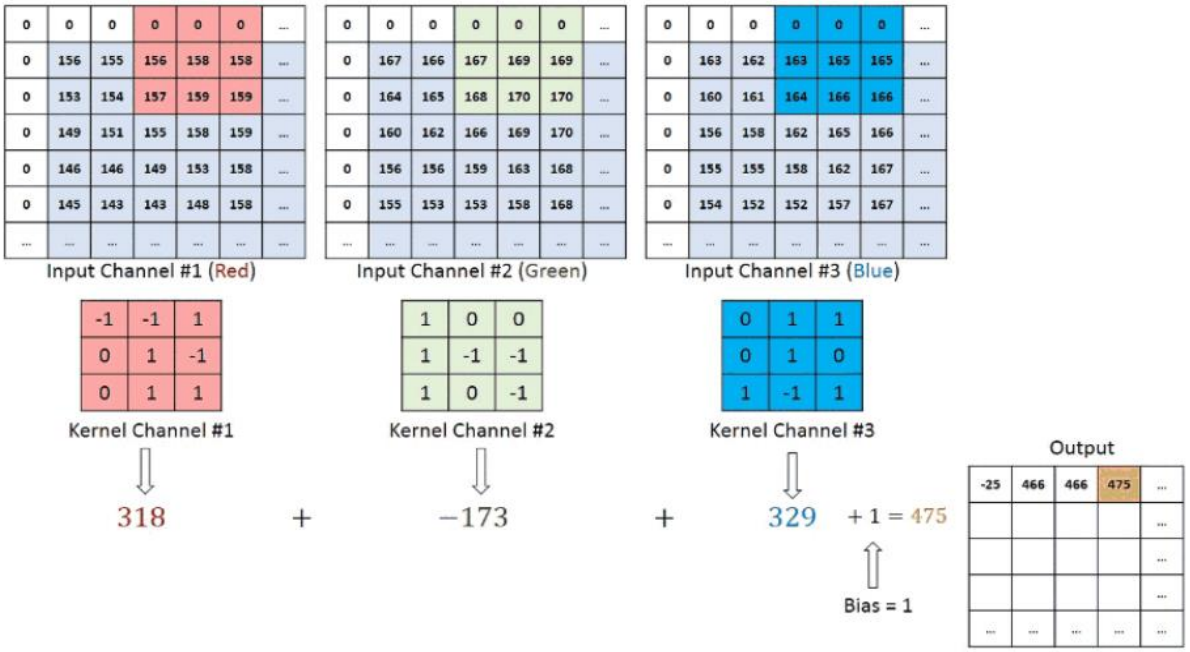


Figure 2.6: Convolution Works on RGB Image Channels with Kernel Filters and Bias

An RGB image consists of three channels: Red, Green, and Blue. Each channel is represented as a separate matrix of numbers, so instead of one image, a CNN processes three images stacked together.

The input image is divided into three channels:

- Channel 1 for Red
- Channel 2 for Green
- Channel 3 for Blue

Each channel contains pixel values of the image. For each channel, there is a corresponding kernel (filter): one for Red, one for Green, and one for Blue.

The CNN performs convolution separately on each channel. This means:

- The Red channel is convolved with the Red kernel, producing a number (e.g., 314).
- The Green channel is convolved with the Green kernel, producing a number (e.g., -175).
- The Blue channel is convolved with the Blue kernel, producing a number (e.g., 326).

CNN **adds them all together**:

314 + (-175) + 326 = 465

Then it adds **bias = 1**

465 + 1 = 466

Bias is an additional value in a neural network that helps the model adjust its output, enabling it to make accurate predictions even when the input is zero.

The number of parameters in a CNN layer depends on the size of the receptive fields (filter kernels) and the number of filters. Each neuron in a CNN layer receives inputs from a local region of the previous layer, known as its receptive field. The receptive fields move over the input, calculating dot products and creating a convolved feature map as the output. Usually, this map then goes through a rectified linear unit (ReLU) activation function. Classic CNN architectures like LeNet and more modern ones like ResNet employs this fundamental principle. Convolutional neural networks are composed of multiple layers of artificial neurons

2.3.2 Feature Extraction in CNN

CNNs learn features hierarchically: early layers detect basic patterns like edges, middle layers detect complex shapes, and deeper layers recognize complete objects such as faces or cars. CNNs are **feedforward networks**, meaning data flows in one direction only: Input → Hidden layers → Output, without any memory or feedback loops like RNNs. After feature extraction, the final layer performs **classification** by producing probability scores for each class using **Softmax** (multi-class) or **Sigmoid** (binary). The class with the highest probability is chosen as the output.

2.3.3 Pooling Layer:

The pooling layer reduces the size of feature maps, lowering computation and preventing overfitting while keeping important features. Max pooling selects the strongest values, while average pooling smoothens the output. Although CNNs are powerful, they lack human-like understanding and require large data and high computation, but they remain essential in modern AI applications.

2.3.4 Advantages and Disadvantages of Convolutional Neural Networks (CNN)

Advantages

- Automatically learn features without manual extraction.
- Use shared weights, reducing parameters, and improving efficiency.
- Recognize patterns regardless of their position in the input.
- Capture both low-level and high-level features effectively.
- Versatile and applicable to images, audio, video, and text.

Disadvantages

- Require high computational power and resources.
- Need large amounts of labelled data for training.
- Difficult to interpret how decisions are made.
- Can overfit, especially with small datasets.
- Require fixed input sizes, limiting flexibility.

2.4 Deep Convolutional Neural Networks (Deep CNNs)

Deep CNN consists of **many stages (layers)** connected in series. In practice, a CNN can have **tens of such stages**. While architectures may differ in the number of stages and internal design, the **basic structure of each stage remains the same**.

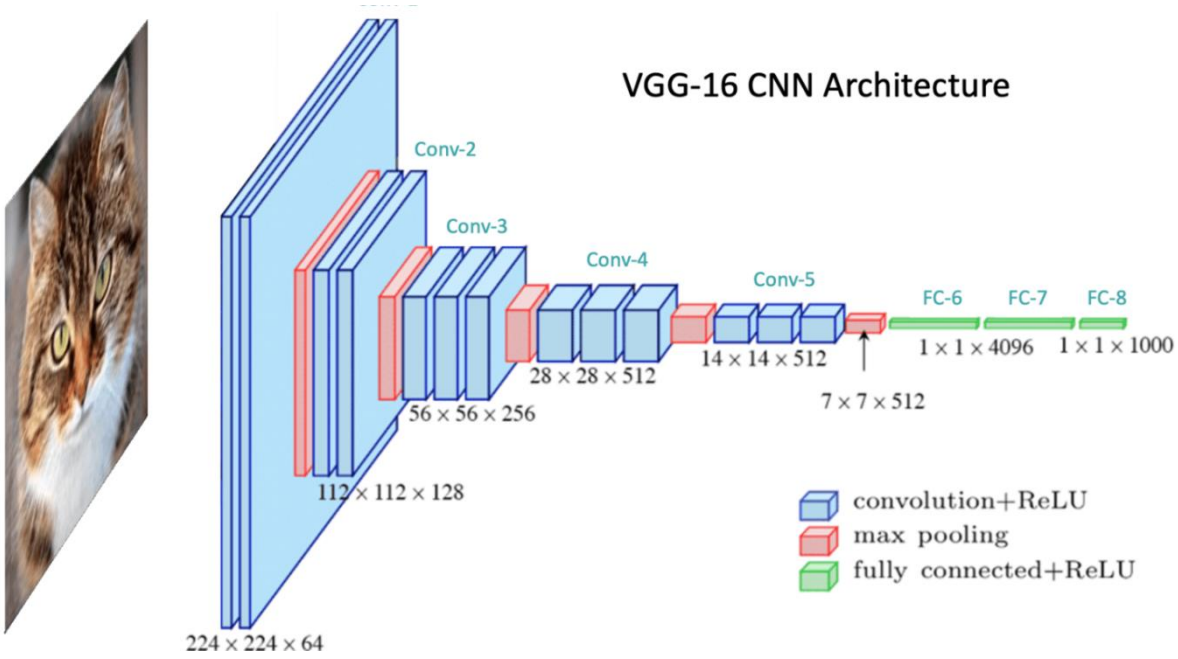


Figure 2.7: VGG-16 CNN Architecture

- Deep CNN with many layers connected in sequence.
- Uses blocks of **Convolution + ReLU → Pooling**.
- Feature map size shrinks (224×224 to 7×7).
- Number of feature maps grows (64 to 512).
- Ends with Fully Connected layers for classification.

Structure of One Stage in a Deep CNN

Each stage of a Deep CNN consists of **three volumes**:

1. **Input Maps Volume**
    - Contains the input feature maps from the previous stage.
    - For an RGB image, this volume has **three maps** (Red, Green, Blue).
    - All maps inside a volume are of the **same size**.
2. **Feature Maps Volume**
    - Generated after convolution, bias addition, and activation.
    - Each feature map detects a specific pattern.
3. **Pooled Maps Volume (Optional)**
    - Produced using pooling.
    - Reduces the spatial size of feature maps.
    - Not used in every stage.

Volume Convolution in Deep CNNs

The fundamental operation in each stage is **convolution**.

In Deep CNNs, the convolution performed is called **volume convolution**:

- A kernel is a **3D volume**, not just a 2D filter.
- The depth of the kernel = depth of the input volume.
- There is **no sliding in the depth dimension**.
- Convolution is performed separately on each 2D map.
- The results are **summed together**.

This sum forms a single value in the feature map.

Convolution Operation

- The kernel slides over the input maps spatially.
- Element-wise multiplication is performed.
- The results are summed.
- This produces one scalar output at each position.

This is called **volume convolution**.

Bias Addition

- A scalar bias is added to the convolution result.
- Helps improve learning flexibility.

Formula: $$z = \sum(w \cdot v) + b$$

- **z** → Final output value after convolution and bias addition
- **w** → Weights of the filter (kernel values)
- **v** → Input values (pixel values or feature map values)
- **b** → Bias (a constant added to shift the result and improve learning)

Activation Function:

Formula: $$a = h(z)$$

means that the biased output **z** is passed through an activation function **h( )** to produce the final activated output **a**, which introduces nonlinearity and helps the network learn complex patterns.

- **z** → Input value to the activation function (after convolution + bias)
- **h( )** → Activation function (e.g., ReLU, sigmoid, tanh)
- **a** → Final activated output passed to the next layer

Feature Maps Volume: A collection of feature maps where each map highlights specific patterns like edges, textures, colours, or shapes from the input image.

Pooling Layer: Reduces the size of feature maps to improve speed, lower computation, and make the model more stable.

Pooled Maps Volume: A smaller version of feature maps that retains important features and is passed to the next CNN layer.

Flow of One CNN Stage: Input Maps → Convolution → Bias Addition → Activation → Feature Maps → Pooling → Pooled Maps → Next Stage

Fully Connected Layers (End of CNN)

After all convolution and pooling stages, the final set of feature maps is converted into a one-dimensional vector using a process called **flattening**. This vector is then passed to one or more **fully connected (dense) layers**. These layers combine the extracted features to make decisions and perform the final classification. The output layer typically uses the **Softmax** activation function for multi-class classification or the **Sigmoid** function for binary classification, producing probability values for each class.

2.5 AlexNet

2.5.1 Introduction

AlexNet is a deep convolutional neural network (CNN) introduced in 2012 by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. It won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2012) by a huge margin, marking the beginning of the modern deep learning revolution in computer vision.

Before AlexNet, image recognition systems relied heavily on manual feature engineering, where humans had to design features by hand. This process was time-consuming, difficult to scale, and not very reliable. AlexNet showed that deep neural networks could automatically learn meaningful features directly from raw images. By combining deep architectures, ReLU activation, dropout, and GPU acceleration, AlexNet achieved unprecedented accuracy and proved that deep learning could outperform traditional machine learning methods.

2.5.2 Background and Motivation

Traditional machine learning methods relied heavily on **handcrafted features**, where engineers manually designed features such as edges, corners, and textures. This process required domain expertise, was time-consuming, and often failed to capture complex patterns in images.

As image datasets grew in size and complexity, classical algorithms like SVMs and shallow neural networks struggled to scale. They were not powerful enough to handle variations in lighting, orientation, background, and object appearance.

The release of the **ImageNet dataset**, containing over **1 million labeled images across 1000 categories**, created a major challenge for existing methods. This large-scale dataset demanded models that could automatically learn rich and hierarchical features.

AlexNet addressed these limitations by using **deep convolutional neural networks** along with **GPU acceleration**, making it possible to train very deep models on massive datasets. This combination enabled faster training, better feature learning, and significantly higher accuracy, proving that deep learning was the future of computer vision.

2.5.3 What is ALexNet?

AlexNet is a convolutional neural network (CNN) that was introduced in 2012. It was designed to recognize images and has been used in many fields, including robotics, medical imaging, and autonomous vehicles.

AlexNet was created to handle challenging picture recognition jobs effectively. For Example, Imagine teaching a child to recognize objects like apples and oranges. Initially, you'd show them many pictures of these objects.

Before AlexNet, we had to manually describe the characteristics of each object to the 'machine child'. AlexNet changed this by allowing the machine to learn these features on its own, similar to how children learn.

2.5.4 AlexNet Architecture

Let's dissect the design and operation of AlexNet to discover what made it revolutionary.

AlexNet has **eight layers**: three fully connected layers for classification and five convolutional layers for feature extraction. Each layer gradually converts the input image into insightful predictions.

ReLU activation, dropout, and overlapping pooling are among the innovations that set AlexNet apart and addressed the shortcomings of previous models.

Layer-by-layer Explanation:

A. Input Layer:

- **Size**: AlexNet takes colour images (RGB) as input with a size of **227 × 227 × 3**. Here, 227 × 227 represents the height and width of the image, and 3 represents the three-color channels: Red, Green, and Blue.
- **Preprocessing:** Before feeding images into the network, they are **normalized**. This means pixel values are adjusted to a common scale. Normalization helps the network learn faster, improves stability, and reduces unnecessary variations in the data.

B. Convolutional Layers:

Convolutional layers act like **pattern detectors**. They scan the image using small filters (kernels) to find important features.

Feature extraction in CNNs happens progressively across layers. The early (shallow) layers detect simple features such as edges, lines, corners, and basic colors. In models like AlexNet, the large 11×11 filter in the first convolutional layer captures broader patterns such as big shapes and coarse textures.

As we move deeper into the network, these simple features are combined to form more complex and meaningful representations like eyes, noses, wheels, faces, and entire objects. This process is automatic each filter learns to recognize a specific pattern on its own, such as vertical edges, curves, or textures, allowing the network to understand images from basic details to complete objects.

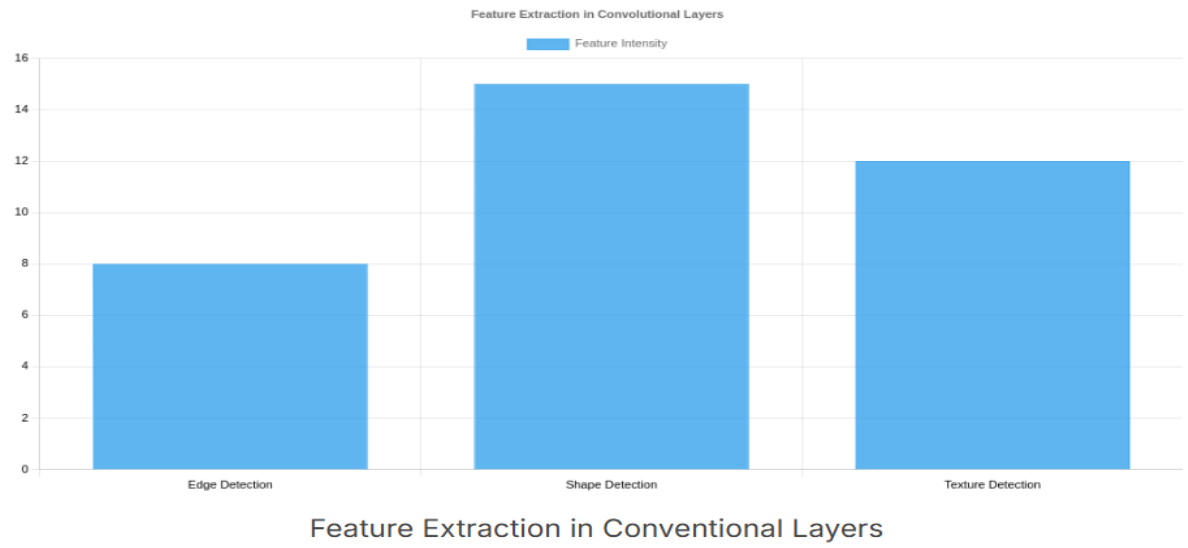


Figure 2.8: Feature Extraction in Convolutional Layers

The image is a bar chart titled *"Feature Extraction in Convolutional Layers."* It visually represents how different types of features are extracted by CNN layers with varying intensities. The three bars show Edge Detection, Shape Detection, and Texture Detection. Edge detection has the lowest value, indicating that simple features like lines, corners, and boundaries are captured first in the shallow layers. Shape detection has the highest value, showing that mid-level layers focus more on identifying complex patterns such as curves and object parts. Texture detection lies in between, representing the recognition of surface patterns and repeated structures in deeper layers. Overall, the image illustrates that CNNs learn features progressively, moving from simple visual details to more complex and meaningful patterns.

C. ReLu Activation:

After the convolution operation, the output feature maps are passed through an activation function called ReLU (Rectified Linear Unit). ReLU introduces non-linearity into the network by converting all negative values to zero while keeping positive values unchanged. This is important because real-world images contain complex patterns that cannot be learned using only linear operations. In AlexNet, ReLU was used instead of traditional activation functions like sigmoid or tanh because it is much faster to compute and helps the network train more efficiently.

ReLU also plays a crucial role in enabling deeper networks by reducing the vanishing gradient problem, where gradients become too small and stop learning. Since ReLU does not squash values into a small range, it allows gradients to flow better through the network, making learning faster and more stable. As a result, ReLU helps CNNs learn complex features more effectively and makes deep architectures like AlexNet practical.

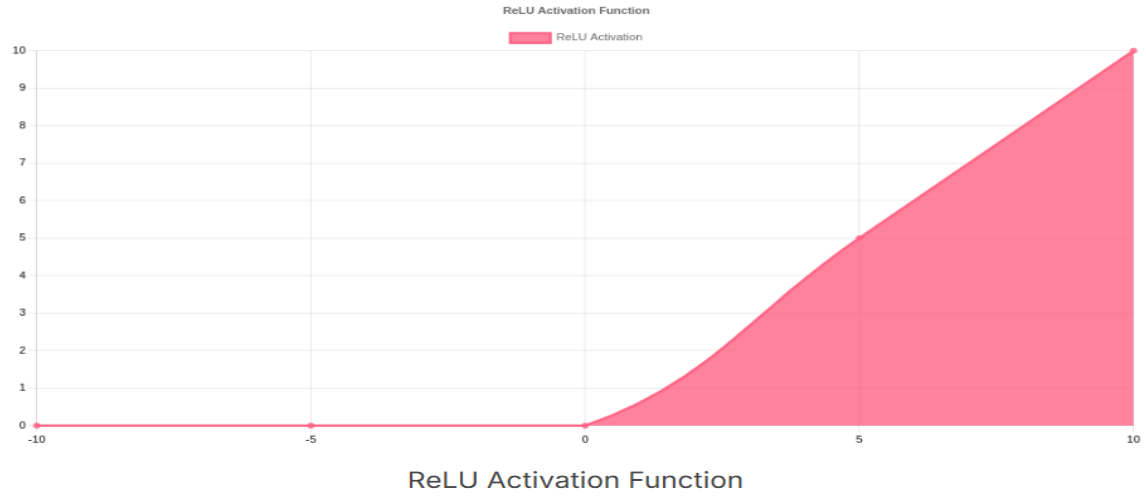


Figure 2.9: ReLU Activation Function

This image illustrates the behavior of the ReLU (Rectified Linear Unit) activation function, which is widely used in convolutional neural networks such as AlexNet. The graph shows that when the input value is negative, the output becomes zero, meaning ReLU completely removes negative activations. When the input is positive, the output increases linearly and remains unchanged. Mathematically, ReLU is defined as $f(x) = max(0, x)$. This simple behavior helps the network focus only on important features while ignoring weak or irrelevant signals. By setting negative values to zero, ReLU introduces non-linearity into the model, allowing it to learn complex patterns. Additionally, since ReLU does not compress values into a small range like sigmoid or tanh, it helps reduce the vanishing gradient problem and enables faster, more stable training of deep networks.

D. Pooling Layers:

Pooling layers reduce the spatial size of feature maps, lowering computational cost, memory usage, and making CNNs faster. In max pooling, the strongest value from each region is selected, preserving important features like edges and textures while removing less useful details. AlexNet used overlapping max pooling, which helped reduce overfitting and improved generalization. Overall, pooling makes CNNs more efficient, robust to small image changes, and better at focusing on key features.

Figure 2.10 illustrates the effect of pooling on feature maps in a convolutional neural network. It compares three important aspects: the **original feature map**, the **preserved features**, and the **reduced feature map size** after pooling. The original feature map represents the detailed spatial information extracted by convolution layers. After pooling, the size of the feature map is significantly reduced, which lowers computational cost and memory usage. However, the most important features such as strong edges, shapes, and textures are preserved.

This shows that pooling removes unnecessary details while keeping essential information intact. In max pooling, the strongest activations are selected from each region, ensuring that important patterns are not lost. As a result, pooling makes the network more efficient, more robust to small changes in the input image, and helps prevent overfitting by reducing complexity.

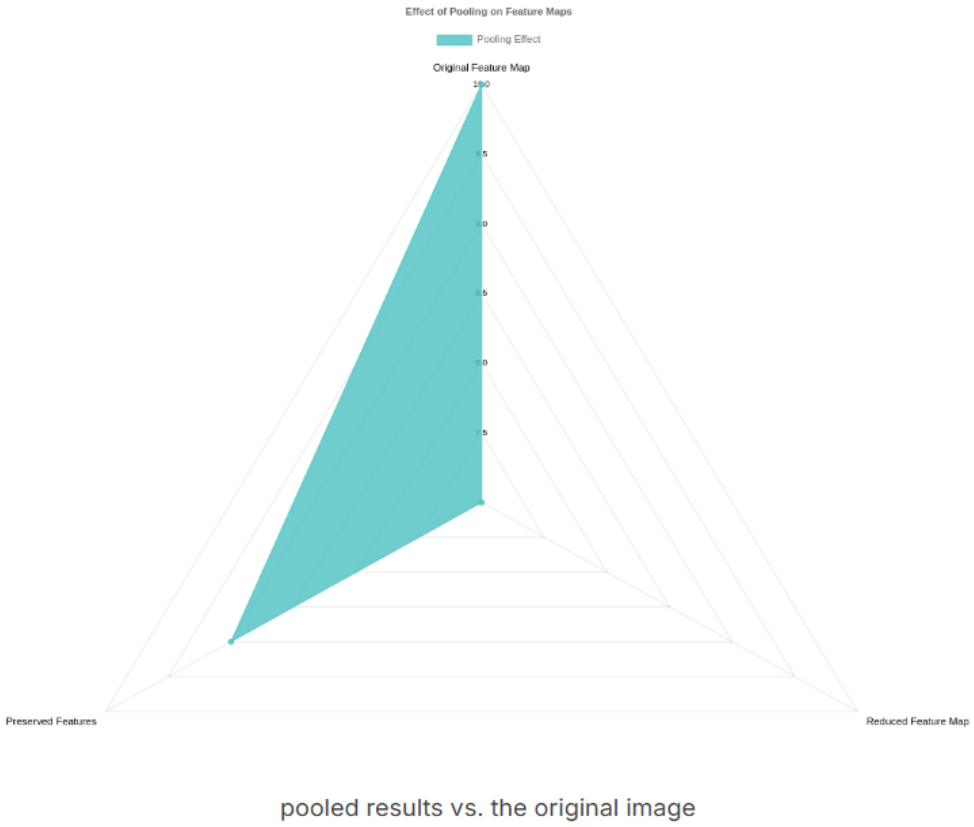


Figure 2.10: ReLU Activation Function

E. Fully Connected Layer

The fully connected (FC) layers are the final stages of AlexNet, where the high-level features extracted by the convolution and pooling layers are converted into meaningful predictions. In these layers, every neuron is connected to all neurons in the previous layer, allowing the network to combine all learned features such as shapes, textures, and object parts. This helps the model recognize complex patterns and relationships in the data, such as how different features together form a complete object. In simple terms, FC layers act like a decision-maker that interprets the extracted features and prepares them for classification.

D. Output Layer (Softmax)

The output layer of AlexNet uses the **softmax** activation function to convert the final values into probabilities. Since AlexNet was trained on the ImageNet dataset, which has **1000 classes**, the output layer contains 1000 neurons. Each neuron represents one class, and the softmax function ensures that all outputs lie between 0 and 1 and sum up to 1. The class with the highest probability is chosen as the final prediction.

By combining convolution layers (for feature extraction), ReLU (for non-linearity), pooling layers (for dimensionality reduction), fully connected layers (for decision-making), and the softmax output layer (for classification), AlexNet achieved outstanding performance. It outperformed its competitors in the ImageNet competition with a record-breaking **top-5 error rate of 15.3%**, proving the effectiveness of deep convolutional neural networks.

2.5.5 Key Innovations and Their Significance

Modern deep learning was made possible by its groundbreaking innovations:

- **ReLU Activation:** In contrast to sigmoid or tanh functions, ReLU's introduction of non-linearity solved the vanishing gradient issue and greatly accelerated training.
- **Dropout:** one of the main issues with deep networks was overfitting. Dropout, a technique used by AlexNet to randomly deactivate neurons during training, enhanced generalization and added a layer of regularization**.**
- **GPU Acceleration:** Due to CPU limitations, AlexNet could not have been trained on the large ImageNet dataset. However, AlexNet significantly decreased training time and illustrated the significance of hardware in AI by utilizing GPUs for parallel processing.
- **Overlapping Pooling:** In contrast to conventional pooling, overlapping pooling added a small amount of regularization, which enhanced the model's capacity to generalize to fresh data.
- **Data Augmentation:** To increase resilience and decrease overfitting, AlexNet artificially expanded the dataset using data augmentation techniques like flipping and random cropping**.**

These developments expanded the potential of deep learning and went beyond simple technological advancements.

2.5.6 Impact of AlexNet's Success

AlexNet's breakthrough signalled a sea change in AI history:

- **Catalyst for Deeper Architectures:** AlexNet's success sparked the creation of deeper and more intricate structures, such as VGGNet, ResNet, and Inception, which all pushed the limits of neural networks.
- **Revolutionizing other fields:** AlexNet's advancements went beyond picture identification, including natural language processing, driverless cars, and medical imaging. Deep learning made seemingly impossible tasks possible**.**
- **Hardware Acceleration Revolution:** AlexNet sparked the creation of TPUs (Tensor Processing Units) and other AI accelerators by demonstrating the effectiveness of GPUs in training deep networks.
- **Shift in AI Research:** AlexNet's success signalled a shift in AI research from conventional machine learning methods to deep learning, making CNNs the preferred method for image-related tasks**.**
- **Reshaping AI Education:** AlexNet inspired a new generation of researchers and educators to study deep learning. Its methods were included in AI courses at universities, inspiring the upcoming generation of AI professionals and enthusiasts**.**

2.5.7 Challenges faced by AlexNet

Even though AlexNet was incredibly successful, there were some challenges:

- **Computational Demands:** AlexNet's training required high-end GPUs, which were costly and out of reach for many researchers at the time**.**
- **Overfitting:** While data augmentation and dropout helped to reduce overfitting, generalization on smaller datasets remained difficult**.**
- **Data Dependency:** The significance of data curation and labelling was underestimated because AlexNet's success was dependent on sizable, annotated datasets such as ImageNet.
- **Energy Consumption:** The computational energy required to train big models, such as AlexNet, raised concerns about scalability and environmental effects.
- **Interpretability**: In spite of its achievements, AlexNet brought to light a crucial problem in deep learning: comprehending the reasoning behind models' decisions. This sparked research into explainable AI.

These difficulties highlighted the necessity of ongoing innovation in the accessibility and effectiveness of the concept.

AlexNet's win in the 2021 ImageNet competition was a significant turning point in artificial intelligence. Its cutting-edge architecture, which included GPU acceleration, dropout, and ReLu activation, helped close the gap between theoretical research and actual application.

In addition to winning a contest, AlexNet laid the groundwork for the AI revolution. The methods it introduced continue to influence contemporary deep learning, from generative models to object detection.

2.6 GoogLeNet Model - CNN Architecture

GoogLeNet (Inception V1) is a deep convolutional neural network architecture designed for efficient image classification. It introduces the Inception module, which performs multiple convolution operations (1x1, 3x3, 5x5) in parallel, along with max pooling and concatenates their outputs. The architecture is deep, yet optimized for speed and performance, which makes it suitable for large-scale visual recognition tasks. It brought forward innovative architectural choices such as 1×1 convolutions, global average pooling and the Inception module, all aimed at improving depth and computational efficiency**.**

2.6.1 Key Features of GoogLeNet

The GoogLeNet architecture is very different from previous architectures such as AlexNet. It uses many different kinds of methods such as:

A. 1×1 Convolutions

One of the core techniques employed in GoogLeNet is the use of 1×1 convolutions, primarily for dimensionality reduction. These layers help decrease the number of trainable parameters while enabling deeper and more efficient architectures**.**

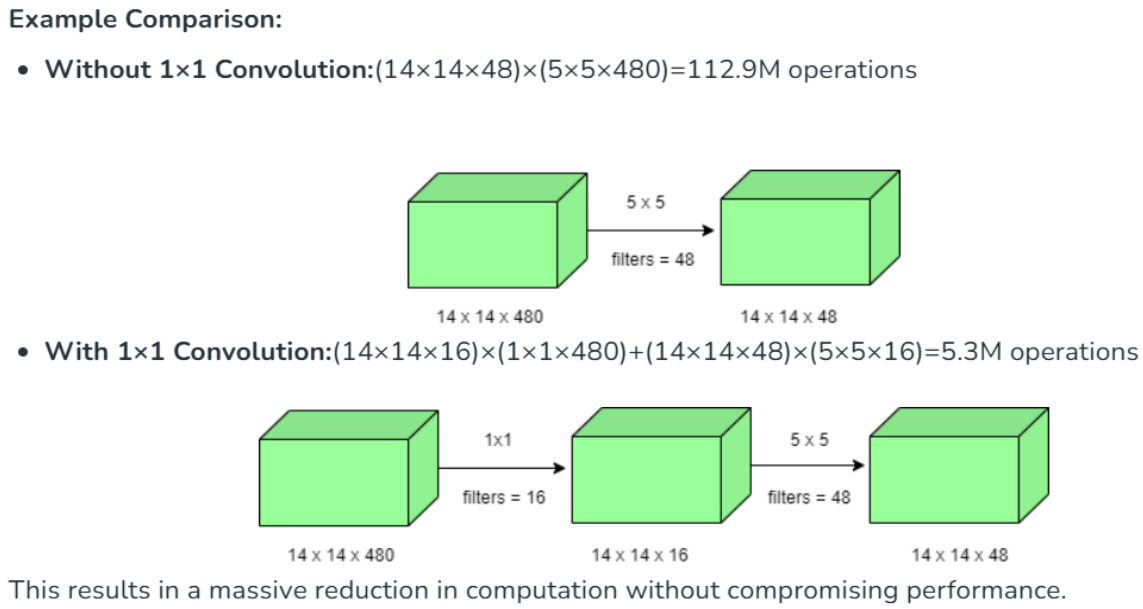


Figure 2.11: 1*1 Convolution

B. Global Average Pooling

In traditional architectures like AlexNet, fully connected layers at the end introduce a large number of parameters. GoogLeNet replaces these with Global Average Pooling, which computes the average of each feature map (e.g. converting 7×7 maps to 1×1), this significantly reduces the model's parameter count and solves overfitting.

Benefits:

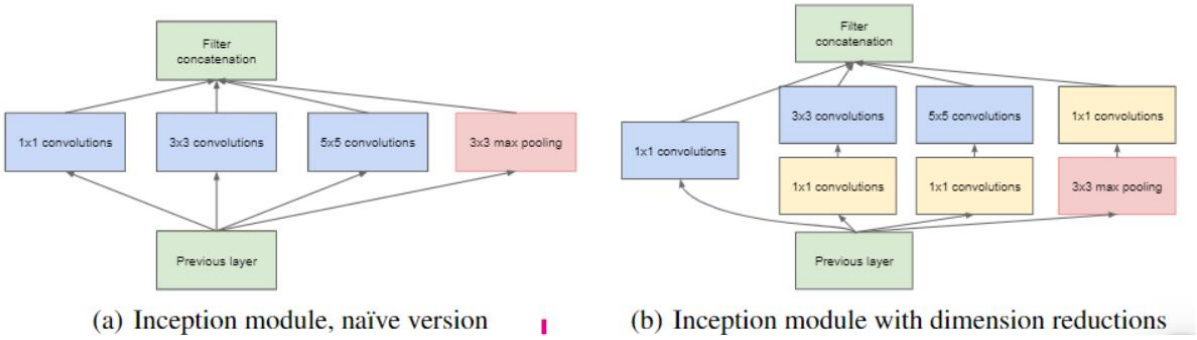
- Zero additional trainable parameters
- Reduces overfitting
- Improves top-1 accuracy by approximately 0.6%

C. Inception Module

The **Inception module** is the architectural core of GoogLeNet and is designed to extract rich features efficiently. Instead of applying a single type of filter at each layer, it processes the same input using multiple operations **in parallel**, including **1×1, 3×3, 5×5 convolutions, and 3×3 max pooling**. Each of these operations captures different kinds of information: smaller filters focus on fine details, while larger filters capture broader patterns. After these parallel operations, all their outputs are **concatenated depth-wise** to form a single combined feature map

the main **purpose** of this design is to allow the network to capture features at **multiple scales** simultaneously, making it more flexible and powerful. Its key **advantage** is that it improves the network's representational ability without dramatically increasing computational cost. By using techniques like 1×1 convolutions for dimensionality reduction, the Inception module

keeps the model efficient while still learning complex and meaningful features



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2.12: Inception Module

In the naïve Inception module, the input is processed in parallel using 1×1, 3×3, 5×5 convolutions and 3×3 max pooling, and their outputs are concatenated. However, this approach is computationally expensive because large filters operate directly on high-dimensional input, leading to many parameters, high memory usage, and slow computation.

To solve this, the optimized Inception module introduces 1×1 convolutions before 3×3 and 5×5 convolutions and after max pooling. These 1×1 layers reduce dimensionality, decrease computation, add non-linearity, and preserve important features. As a result, the optimized version is much more efficient and allows GoogLeNet to be deep and powerful without being too heavy or slow

D. Auxiliary Classifiers

To address the vanishing gradient problem during training, GoogLeNet introduces auxiliary classifiers(intermediate branches that act as smaller classifiers). These are active only during training and help regularize the network**.**

Structure of Each Auxiliary Classifier:

- Average pooling layer (5×5, stride 3)
- 1×1 convolution (128 filters, ReLU)
- Fully connected layer (1024 units, ReLU)
- Dropout layer (dropout rate = 0.7)
- Fully connected softmax layer (1000 classes)
- The auxiliary losses are added to the main loss with a weight of 0.3 to stabilize training.

E. GoogLeNet Model Architecture:

GoogLeNet is a 22-layer deep convolutional neural network (excluding pooling layers) designed with a strong focus on computational efficiency and high performance. Unlike earlier networks that became deeper by simply stacking convolution layers, GoogLeNet introduced the Inception module, which allows the network to grow deeper and wider without a huge increase in computation.

The network begins with initial convolution and max-pooling layers that extract low-level features such as edges and textures while reducing spatial size. After this, the core of the network consists of a series of Inception modules, stacked one after another. Each Inception module processes the same input using parallel paths of 1×1, 3×3, and 5×5 convolutions, along with max pooling. These outputs are then concatenated depth-wise, allowing the model to capture both fine and coarse features at the same time.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 2.4: Overall GoogLeNet Architecture)

To keep the network efficient, 1×1 convolutions are used inside the Inception modules for dimensionality reduction, which significantly reduces the number of parameters and computations. GoogLeNet also includes auxiliary classifiers connected to intermediate layers. These help reduce the vanishing gradient problem and act as regularizers during training.

Finally, instead of using large fully connected layers, GoogLeNet applies global average pooling, which converts each feature map into a single value. This drastically reduces parameters and prevents overfitting. The final layer uses a softmax classifier to produce class probabilities.

- **Inception V1 architecture** is a design for a deep neural network used mainly for image recognition tasks. It was created by Google researchers to make the network both **deep** and **efficient**.

Instead of just stacking regular convolution layers one after another, it uses **Inception modules** that do several operations in parallel (like looking at the image with different filter sizes all at once), and then combines the results.

This way, the network can understand patterns of different sizes (small details and big shapes) efficiently, without making the network too big or slow.

2.6.2 Key highlights of the architecture:

- **Input Layer**: Accepts a 224×224 RGB image as input.
- **Initial Convolutions and Pooling**: Applies a series of standard convolutional and max pooling layers to downsample the input and extract low-level features.
- **Local Response Normalization (LRN)**: Normalizes the feature maps early in the network to improve generalization.
- **Inception Modules**: Each module processes the input through 1×1, 3×3, and 5×5 convolutions, as well as 3×3 max pooling, all in parallel. The outputs are concatenated along the depth dimension, allowing the network to capture both fine and coarse features.
- **Auxiliary Classifiers**: Appear as smaller branches connected to intermediate layers of the network. Include average pooling, 1×1 convolutions, fully connected layers, and softmax outputs.
- **Final Layers**: Uses global average pooling (7×7) to reduce each feature map to a single value. Followed by a fully connected layer and a softmax activation to produce the final classification output.

2.6.3 Performance and Results

- Winner of ILSVRC 2014 in both classification and detection tasks
- Achieved a top-5 error rate of 6.67% in image classification
- An ensemble of six GoogLeNet models achieved 43.9% mAP (mean Average Precision) on the ImageNet detection task

Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Table 2.5: GoogLeNet Classification Performance (Top-5 Error)

Team	Year	Place	mAP	external data	ensemble	approach
UvA-Euvision	2013	1st	22.6%	none	?	Fisher vectors
Deep Insight	2014	3rd	40.5%	ImageNet 1k	3	CNN
CUHK DeepID-Net	2014	2nd	40.7%	ImageNet 1k	?	CNN
GoogLeNet	2014	1st	43.9%	ImageNet 1k	6	CNN

Table 2.6: GoogLeNet Detection Performance

2.7 Residual Networks (ResNet)

To overcome the challenges of training very deep neural networks, Residual Networks (ResNet) was introduced, which uses skip connections that allow the model to learn residual mappings instead of direct transformations making deep neural networks easier to train.

- It helps prevent vanishing gradient problems in very deep models.
- Skip connections let information flow directly across layers.
- ResNet enables building networks with hundreds or even thousands of layers.
- It is widely used in computer vision tasks like image classification and object detection.

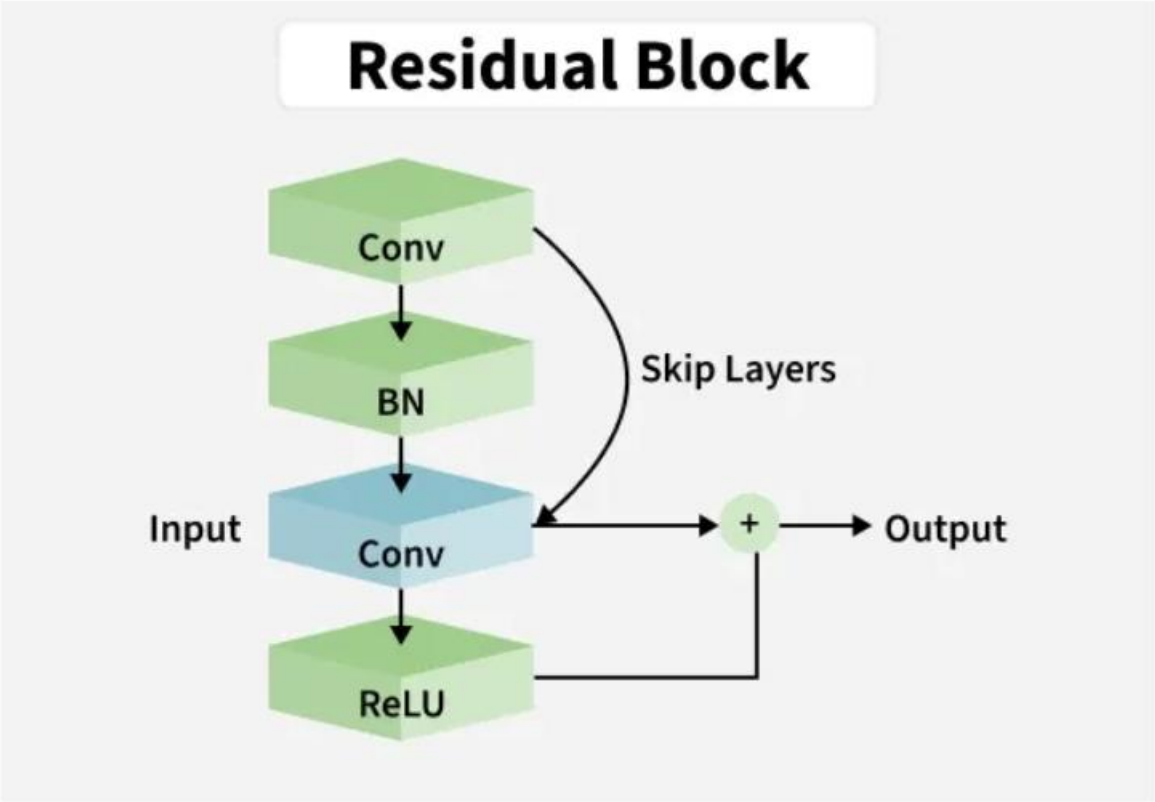


Figure 2.13: Residual Block

A residual block lets the network skip layers by adding the original input to the processed output, making deep networks easier to train.

2.7.1 Challenges in Deep Neural Networks

Deep Neural Networks are useful models but they also come with several training challenges, especially as the network depth increases.

Two major issues are

1. Vanishing/Exploding Gradient Problem: As the number of layers in a neural network increases, the gradients of the loss function with respect to the weights can become extremely small or excessively large during backpropagation.

2. Degradation Problem: The degradation problem occurs when increasing the network depth does not improve performance and may even worsen it. This problem has two aspects:

- **Performance Plateau:** Training error saturates after a certain depth meaning additional layers do not significantly reduce the error.
- **Accuracy Degradation:** Beyond a certain depth validation error increases and the model performs poorly on unseen data.

2.7.2 Understanding ResNet

ResNet is a deep learning architecture designed to train very deep networks efficiently using residual connections. Here are the key features of ResNet:

- **Residual Connections:** Enable very deep networks by allowing gradients to flow through identity shortcuts, reducing the vanishing gradient problem.
- **Identity Mapping:** Simplifies training by learning residual functions instead of full mappings.
- **Depth:** Supports extremely deep architectures for improved image recognition performance.
- **Fewer Parameters:** Achieves high accuracy with fewer parameters hence improving computational efficiency.
- **Results:** Delivers top performance on benchmark image recognition tasks.
- **Effective Approach:** Residual connections provide a reliable way to train deeper networks effectively, enabling networks to learn more complex features.

Here, the graph compares the training and test error of a 20-layered and 56-layered network across iterations showing how deeper networks struggle without proper residual connections.

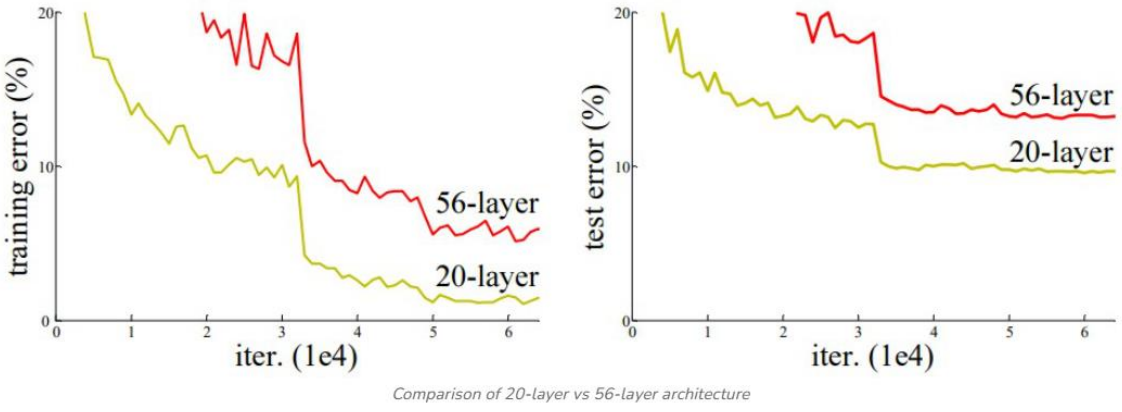


Figure 2.14: Comparison of 20-layer vs 56-layer architecture

- **Left graph (training error):** The 56 layer network reduces error slowly and shows strong fluctuations due to vanishing gradients, whereas the 20-layer network learns smoothly and reaches a much lower training error.

- **Right graph (test error):** The 56-layer network maintains a higher test error (degradation problem), while the 20-layer network generalizes better showing why ResNet skip connections are essential for training deep models.

2.7.3 ResNet-34

ResNet-34 is a deep residual network built on a 34-layer plain network inspired by VGG-19, with shortcut connections forming 16 residual blocks.

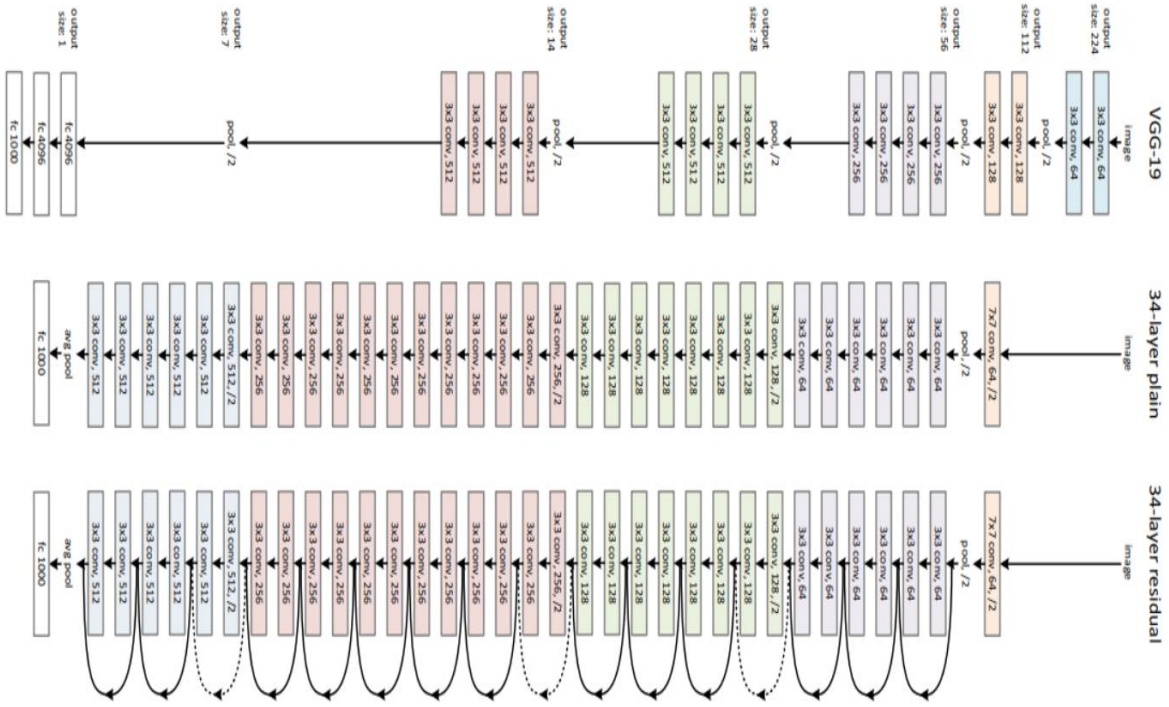


Figure 2.15: ResNet-34

Here are the different stages of the ResNet-34 architecture, showing its structured arrangement of residual blocks.

- **First set:** 3 residual blocks each with 2 convolution layers of 64 filters and identity skip connections.
- **Second set:** 4 residual blocks each with 2 convolution layers of 128 filters uses zero-padding or 1x1 projections for dimension changes.
- **Third set:** 6 residual blocks, each with 2 convolution layers of 256 filters.
- **Fourth set:** 3 residual blocks with 2 convolution layers of 512 filters each.
- **Feature map:** Passed through Global Average Pooling a dense layer with 1000 neurons and softmax for classification.

2.7.4 How ResNet Works

Conventional networks try to learn the full mapping $H(x)$. ResNet instead learns a residual function and combines it with the input via a skip connection

$$H(x) = F(x) + x$$

where:

- x: input to the block
- $H(x)$: desired mapping
- $F(x)$: residual function to be learned

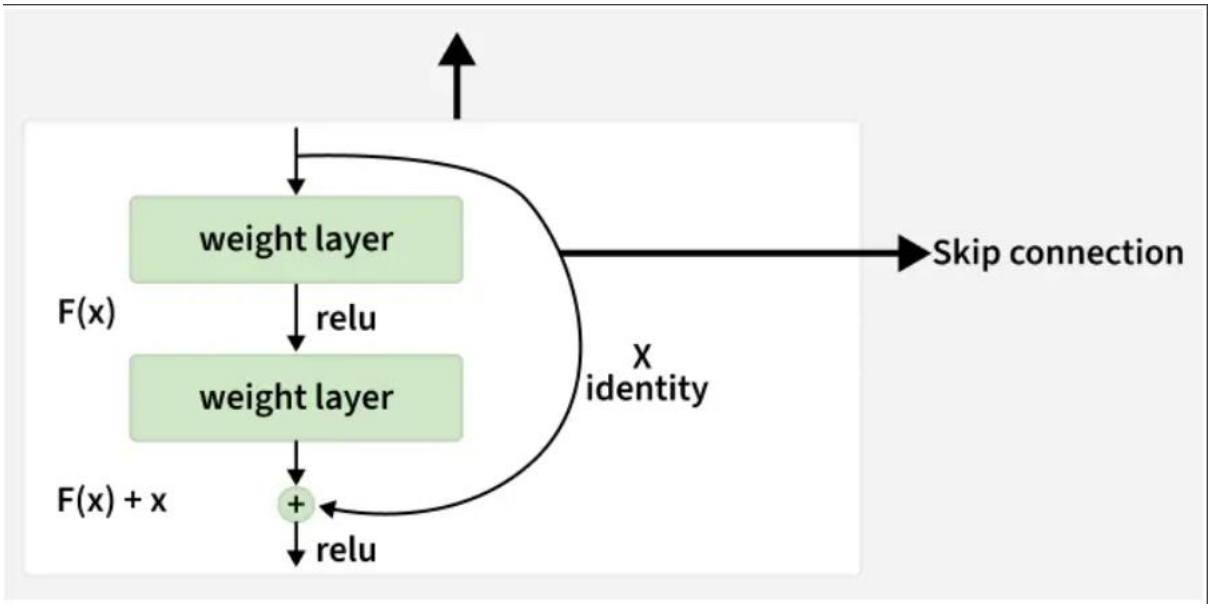


Figure 2.16: ResNet Working

2. Skip (Shortcut) Connection

- Bypasses one or more layers
- Adds input directly to output
- Prevents vanishing gradients
- Improves parameter updates

3. Handling Dimension Mismatch: When input and output dimensions differ

- **Zero Padding:** Adds extra zeros to the input to match output dimensions in a residual block
- **Linear Projection:** Uses a learnable 1x1 convolution to match input and output dimensions for the skip connection.

3. Stacking Residual Blocks: Multiple residual blocks can be stacked to create deep architectures. This allows networks to go very deep without suffering from degradation.

4. Global Average Pooling (GAP): Before the final fully connected layer ResNet uses GAP:

- Converts each feature map to a single value by averaging
- Reduces parameters less overfitting

- Produces compact feature representation

2.7.5 Advantages

- **Eases Training of Deep Networks:** Skip connections allow gradients to flow directly through the network, reducing vanishing gradient problems.
- **Enables Very Deep Architectures:** ResNet can train networks with 50, 100 or even 152+ layers effectively.
- **Improves Accuracy:** Residual learning helps the network achieve higher performance on tasks like image classification and object detection.
- **Reduces Degradation:** Adding more layers does not increase training error unlike plain deep networks.
- **Fewer Parameters for Better Efficiency:** Deep ResNets can have fewer parameters than traditional deep networks but performing better.

2.7.6 Challenges

- **High Computation:** Requires high computational power due to deep architecture.
- **Dimension Mismatch:** Dimension mismatch in skip connections needs extra projection layers.
- **Overfitting Risk:** Risk of overfitting on small datasets because of large model capacity.
- **Training Instability:** Training can become unstable without proper batch normalization.
- **Redundant Updates:** Residual blocks may learn only small or redundant updates.
- **Deep Network Degradation:** Gradient flow improves but may still degrade in extremely deep networks.

2.8 R-CNN - Region-Based Convolutional Neural Networks

Traditional Convolutional Neural Networks (CNNs) with fully connected layers often struggle with object detection tasks, especially when dealing with multiple objects of varying sizes and positions within an image. A brute-force method like applying a sliding window across the image to detect objects is highly computationally expensive, as it fails to scale efficiently when object frequency and variation increase.

To overcome these challenges, R-CNN (Regions with CNN features) was introduced. R-CNN presents a smarter approach by using a selective search algorithm to generate around 2,000 region proposals from an image. These proposals are likely to contain objects and are individually processed to detect and localize them more efficiently. R-CNN marked a significant advancement in the field of object detection and laid the foundation for faster and more accurate object detection models.

2.8.1 R-CNN Working

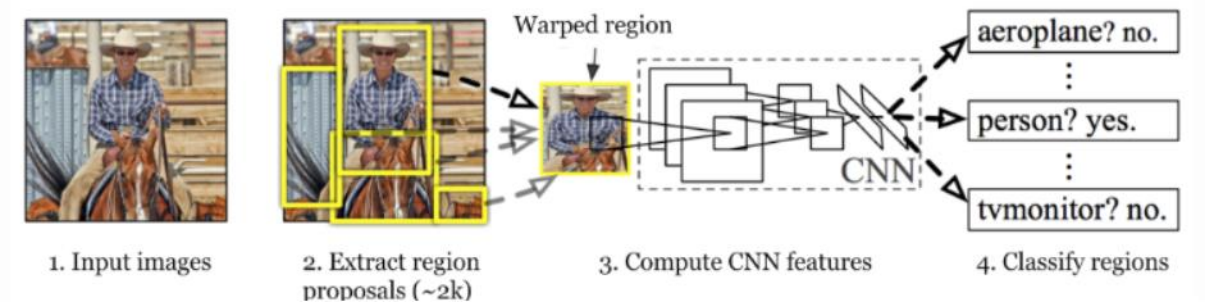


Figure 2.17: R-CNN Working

- **Input Image**: Start with a single input image containing one or more objects.
- **Region Proposal Generation**: Use Selective Search to generate around 2,000 region proposals (potential object locations).
- **Warp & Feature Extraction**: Each proposed region is cropped and resized (warped) to a fixed size. Then pass each region through a CNN to extract feature vectors.
- **Region Classification**: Use the extracted features to classify each region using SVMs into object categories (e.g. person, car) or background.

2.8.2 Key Features of R-CNNs

A. Region Proposals:

R-CNNs begin by generating region proposals, which are smaller sections of the image that may contain the objects we are searching for. The algorithm employs a method called selective search, a greedy approach that generates approximately 2,000 region proposals per image. Selective search effectively balances the number of proposals while maintaining high object recall, ensuring efficient object detection.

By limiting the number of regions for detailed analysis, this method enhances the overall performance of the R-CNN in detecting objects within images.

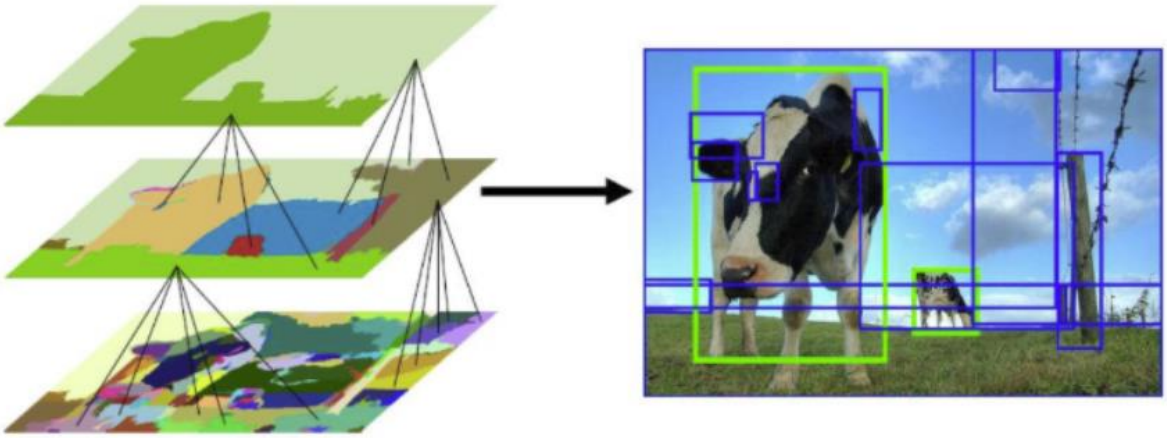


Figure 2.18: Object Detection using Feature Extraction

B. Selective Search

Selective Search is a greedy algorithm that generates region proposals by combining smaller segmented regions. It takes an image as input and produces region proposals that are crucial for object detection. This method offers significant advantages over random proposal generation by limiting the number of proposals to approximately 2,000 while ensuring high object recall.

Algorithm Steps:

1. Generate Initial Segmentation: The algorithm starts by performing an initial sub-segmentation of the input image.
2. Combine Similar Regions: It then recursively combines similar bounding boxes into larger ones. Similarities are evaluated based on factors such as color, texture, and region size.
3. Generate Region Proposals: Finally, these larger bounding boxes are used to create region proposals for object detection.

The selective search algorithm provides an efficient way to identify potential object regions, enhancing the overall effectiveness of the detection process.

C. Input Preparation in R-CNN

After generating the region proposals, these regions are warped into a uniform square shape to match the input dimensions required by the CNN model. In this case, we use the pre-trained **AlexNet model**, which was considered the state-of-the-art CNN for image classification at the time.

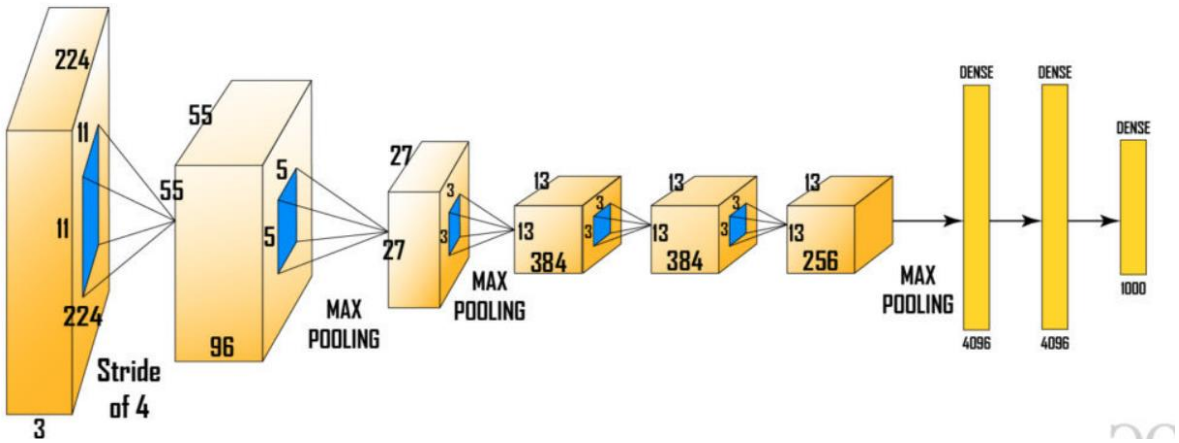


Figure 2.19: Warping and Resizing Region Proposals for CNN Input

The input size for AlexNet is (227, 227, 3), meaning each input image must be resized to these dimensions. Consequently, whether the region proposals are small or large, they need to be adjusted accordingly to fit the specified input size.

From the above architecture, we remove the final softmax layer to obtain a (1, 4096) feature vector. This feature vector is then fed into both the Support Vector Machine (SVM) for classification and the bounding box regressor for improved localization.

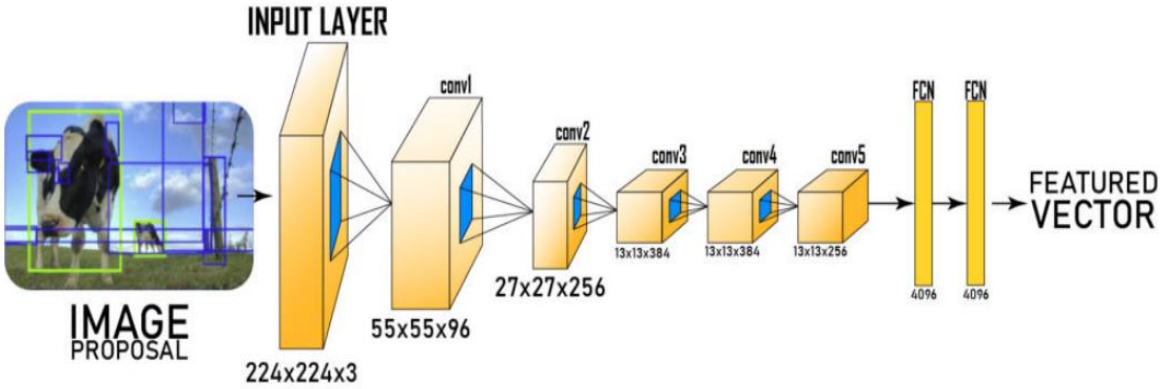


Figure 2.20: CNN Feature Extraction from Region Proposals

D. SVM (Support Vector Machine)

The feature vector generated by the CNN is then utilized by a binary **Support Vector Machine (SVM)**, which is trained independently for each class. This SVM model takes the feature vector produced by the previous CNN architecture and outputs a confidence score indicating the likelihood of an object being present in that region.

However, a challenge arises during the training process with the SVM: it requires the AlexNet feature vectors for each class. As a result, we cannot train AlexNet and the SVM independently and in parallel.

E. Bounding Box Regressor

To accurately locate the bounding box within the image, we utilize a scale-invariant linear regression model known as the **bounding box regressor**. For training this model, we use pairs of predicted and ground truth values for four dimensions of localization: (x, y, w, h). Here, x and y represent the pixel coordinates of the center of the bounding box, while w and h indicate the width and height of the bounding boxes, respectively.

This method enhances the **Mean Average Precision (mAP)** of the results by 3-4%.

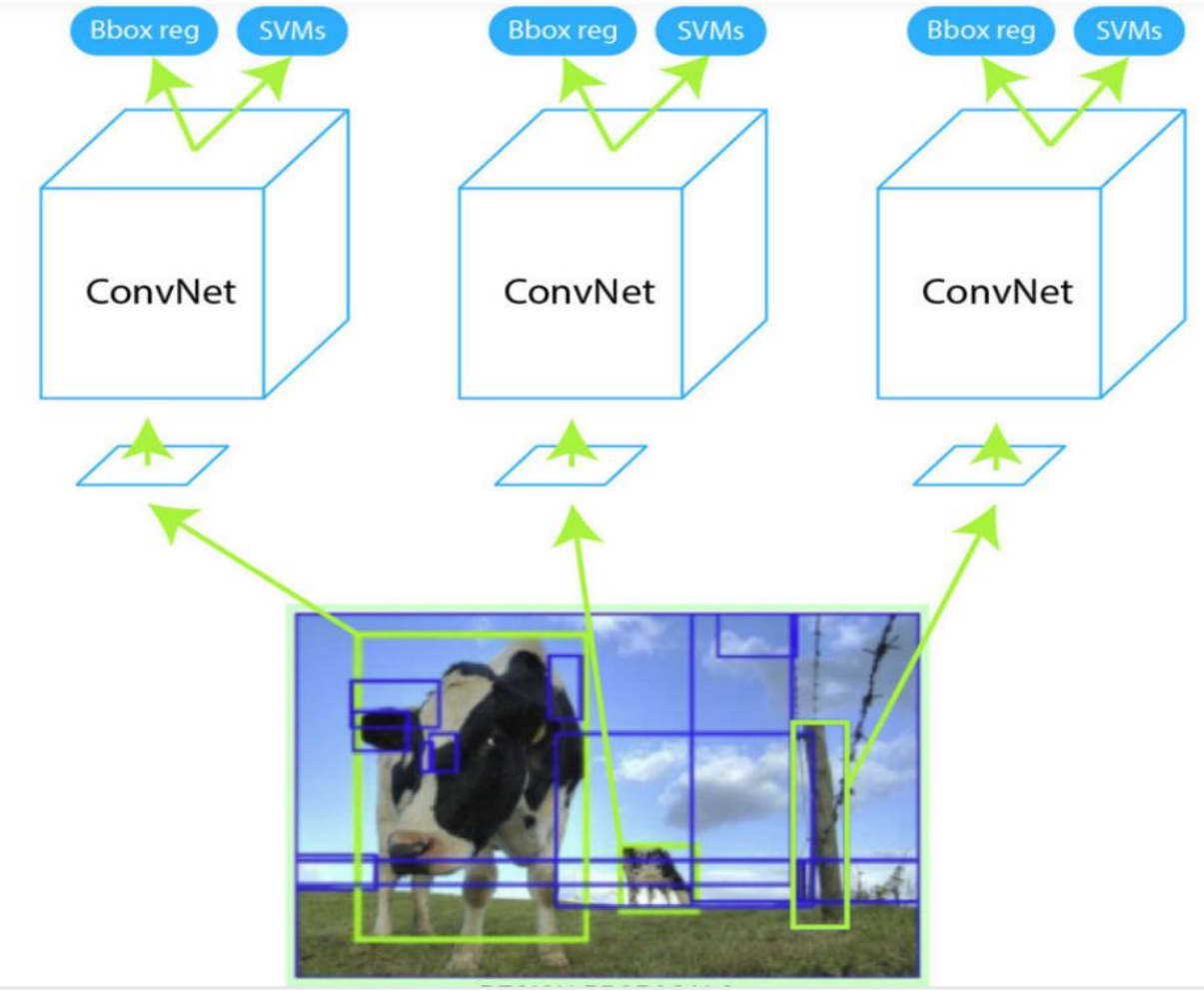


Figure 2.21: Bounding Box Regression for Object Localization in R-CNN

To further optimize detection, R-CNNs apply **Non-Maximum Suppression (NMS)**:

1. Remove proposals with confidence scores below a threshold (e.g., 0.5).
2. Select the highest-probability region among candidates for each object.
3. Discard overlapping regions with an IoU (Intersection over Union) above 0.5 to eliminate duplicate detections, where IoU is defined as:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

By combining region proposals, selective search, CNN-based feature extraction, SVM classification, and bounding box refinement, R-CNN achieves high accuracy in object detection, making it suitable for various applications.

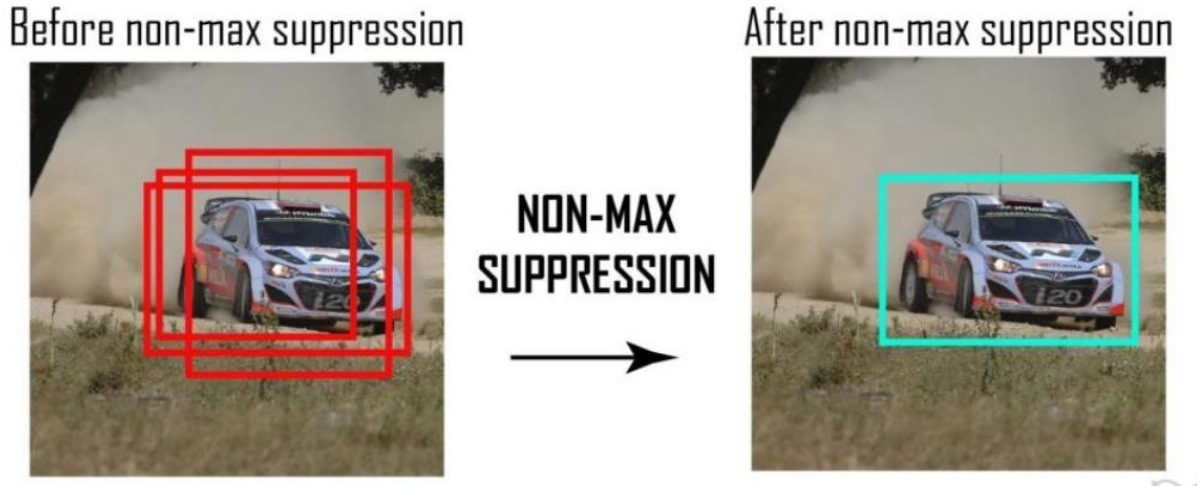


Figure 2.22: Final Object Detection after Non-Maximum Suppression

After that, we can obtain output by plotting these bounding boxes on the input image and labeling objects that are present in bounding boxes.

Results of R-CNN Model

The R-CNN gives a Mean Average Precision (mAPs) of 53.7% on VOC 2010 dataset. On 200-class ILSVRC 2013 object detection dataset it gives an mAP of 31.4% which is a large improvement from the previous best of 24.3%. However, this architecture is very slow to train and takes ~ 49 sec to generate test results on a single image of the VOC 2007 dataset.

2.8.3 Evolution of R-CNN: Fast R-CNN and Mask R-CNN

Following the introduction of R-CNN, several variations emerged to address its limitations:

A. Fast R-CNN

Fast R-CNN optimizes the R-CNN architecture by sharing computations across proposals. Key improvements include:

- **Single Stage Processing**: Instead of extracting features for each region proposal independently, Fast R-CNN processes the entire image once through the CNN to generate a feature map. The region proposals are then extracted from this shared feature map.
- **Softmax Classifier**: Fast R-CNN replaces the SVM with a softmax classifier, allowing for end-to-end training of the network.
- **Improved Bounding Box Regression**: Fast R-CNN enhances the bounding box regression process, leading to better localization accuracy.

B. Faster R-CNN

Faster R-CNN further advances the R-CNN framework by incorporating a Region Proposal Network (RPN). Key features include:

- **Region Proposal Network**: The RPN generates high-quality region proposals directly from the feature maps produced by the CNN, eliminating the need for selective search.

- **Shared Convolutional Features**: Both the RPN and the detection network share the convolutional features, significantly reducing computation time.
- **Improved Speed**: Faster R-CNN achieves real-time processing speeds of around 0.1 seconds per image while maintaining high detection accuracy.

C. Mask R-CNN

Building upon Faster R-CNN, Mask R-CNN was introduced to extend the model to perform instance segmentation. Key features include:

- **Segmentation Masks**: In addition to bounding boxes, Mask R-CNN predicts a segmentation mask for each detected object, providing pixel-level accuracy.
- **Feature Pyramid Networks (FPN)**: Mask R-CNN incorporates FPNs to improve performance on objects at different scales, enhancing detection accuracy for small objects.
- **RoIAlign**: This technique replaces RoIPooling to address misalignment issues, ensuring better feature extraction for each region of interest.

D. Cascade R-CNN

Cascade R-CNN implements a multi-stage object detection framework to improve detection performance. Key aspects include:

- **Multi-Stage Detection**: Cascade R-CNN employs a series of detectors operating at different stages, progressively refining the proposals and improving localization accuracy.
- **Improved Recall and Precision**: By addressing the trade-off between recall and precision at each stage, the model enhances overall detection performance, especially on challenging datasets.

2.8.4 Applications of R-CNN

- **Autonomous Vehicles**: R-CNN can detect and classify various objects on the road, such as pedestrians, other vehicles, and traffic signs, contributing to safer navigation.
- **Surveillance Systems**: In security applications, R-CNN can identify suspicious activities by detecting and classifying individuals and objects in real-time.
- **Medical Imaging**: R-CNN is used in medical applications to identify anomalies in medical scans, assisting in early diagnosis and treatment.
- **Augmented Reality**: R-CNN can enable object recognition in augmented reality applications, enhancing user experiences by overlaying digital information on the real world.

2.8.5 Challenges of R-CNN

R-CNN faces several challenges in its implementation:

- **Rigid Selective Search Algorithm**: The selective search algorithm is inflexible and does not involve any learning. This rigidity can result in poor region proposal generation for object detection.
- **Time-Consuming Training**: With approximately 2,000 candidate proposals, training the network becomes time-intensive. Additionally, multiple components need to be trained separately, including the CNN architecture, SVM model, and bounding box regressor. This multi-step training process slows down implementation.
- **Inefficiency for Real-Time Applications**: R-CNN is not suitable for real-time applications, as it takes around 50 seconds to process a single image with the bounding box regressor.
- **Increased Memory Requirements**: Storing feature maps for all region proposals significantly increases the disk memory needed during the training phase.