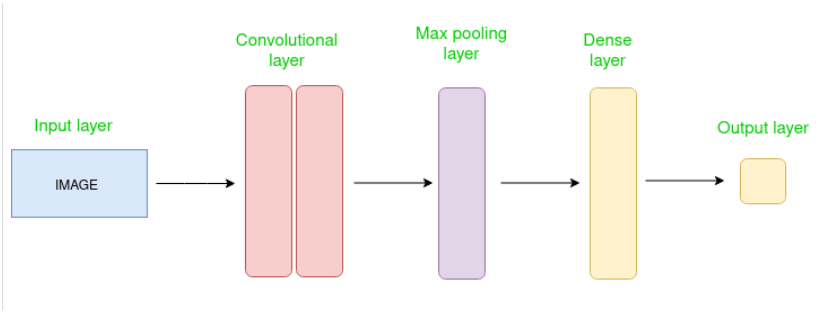
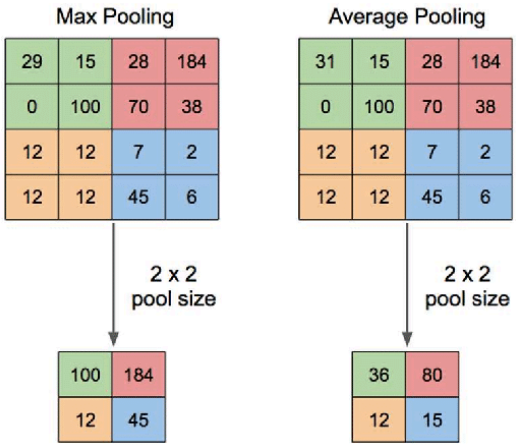
Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are predominantly used to extract features from grid-like data, such as images. The word "convolution" refers to a mathematical operation where one function is applied across another to produce a third function, and in the context of CNNs, it is used to extract meaningful features from input data.

A CNN consists of several key layers:



1. **Convolutional Layer**: This layer applies filters (or kernels) to the input image. Each filter focuses on a different aspect of the image (like edges, textures, etc.) and extracts relevant features from it. The goal is to detect patterns in smaller regions of the image, which helps identify key visual features.
2. **Pooling Layer**: After the convolutional layers, pooling layers are used to downsample the image. This reduces the spatial size of the data, minimizing the computational load while preserving the most important features. Common types of pooling are **max pooling**, where the largest value is taken from a group of pixels, and **average pooling**, where the average is taken.

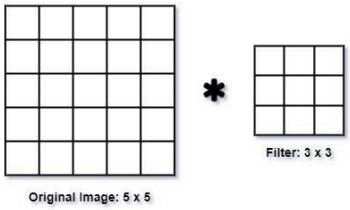


3. **Fully Connected Layer(Dense layer)**: In the final part of the CNN, fully connected layers are used to make predictions based on the extracted features. These layers connect every neuron from the previous layer to the next, allowing the network to make classifications or regressions.

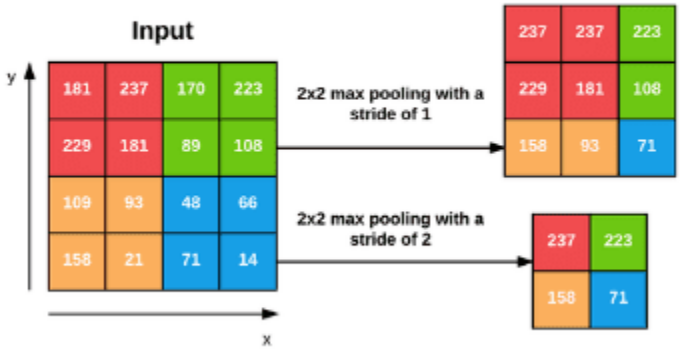
   Backpropagation and Learning: Throughout the learning process, the network adjusts its filters using a method called backpropagation. This allows CNN to learn optimal filters that best extract meaningful features from the data.

Key Concepts in CNNs:

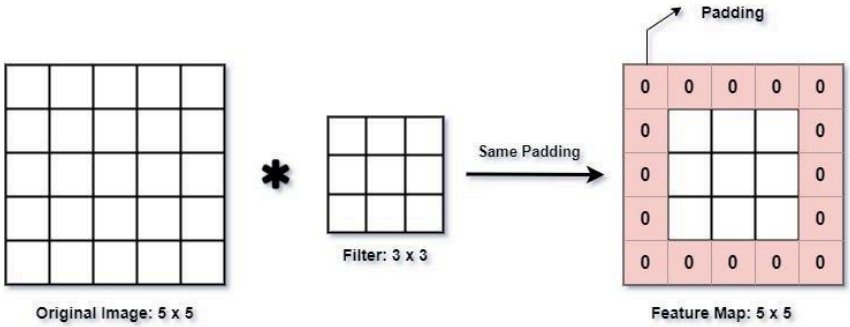
- **Filters/Kernels**: These are small matrices that slide over the input image, performing the convolution operation. Different filters capture different features such as edges, corners, or textures.



- **Convolution**: The process of applying a filter over an image to produce a feature map.
- **Sliding**: Refers to how the filter moves across the input image to cover different regions, extracting features from each part.
- **Strides**: Strides determine how much the filter moves across the input. A stride of 1 means the filter moves one pixel at a time, while larger strides mean faster coverage of the input with fewer computations.



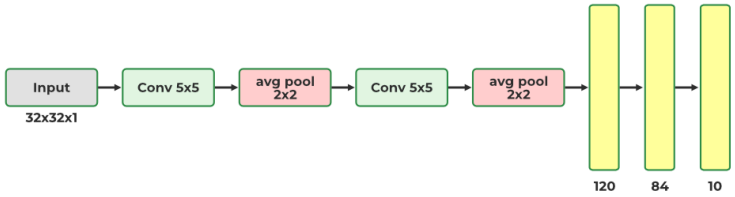
- **Padding**: Padding involves adding extra pixels (usually zeros) around the borders of the input image. This ensures that filters can be applied to the edges of the image without losing information.



Some CNN Architectures

❖ **LeNet-5**

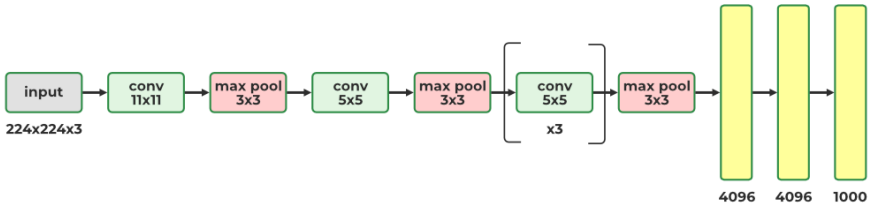
- ➢ Developed for **handwritten digit recognition** (MNIST dataset).
- ➢ Utilizes average pooling to reduce dimensionality.
- ➢ One of the earliest successful CNNs, showcasing the potential of convolutional networks for image classification tasks.
- ➢ **Architecture**:



- Consists of **2 convolutional layers** followed by **3 fully connected layers**.

❖ **AlexNet**

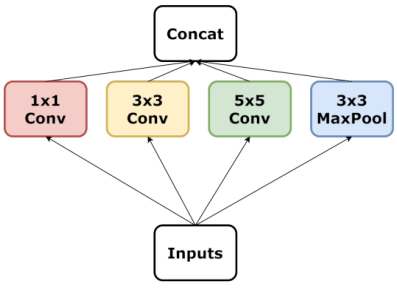
- ➢ The first model to successfully use a **deep CNN** on the **ImageNet dataset**.
- ➢ **Architecture**:



  - ■ Stacks convolutional layers directly on top of each other, allowing for a deep learning structure.
- ➢ **ReLU activation function**: Speeds up training compared to traditional activation functions like sigmoid or tanh.
- ➢ **Dropout technique**: First introduced in this model to prevent overfitting by randomly deactivating neurons during training.

❖ **GoogleNet (Inception Network)**

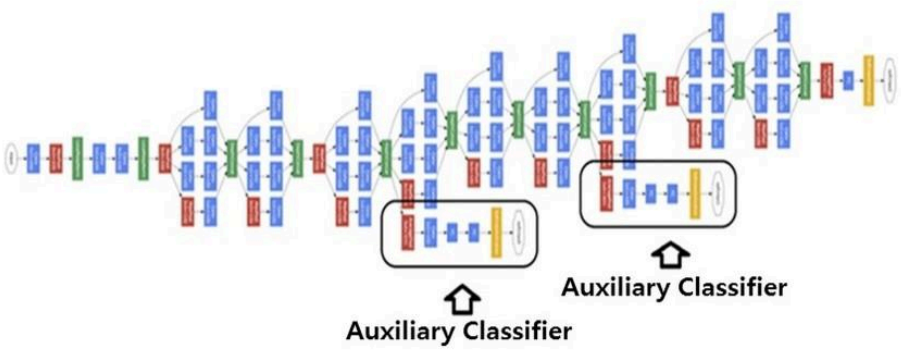
- ➢ Introduced the **Inception module**:



Inception Module

- Applies filters of various sizes (1x1, 3x3, 5x5) within the same layer.
- **Capture**:
  - **1x1 filters**: Used for dimensionality reduction.
  - **3x3 filters**: Capture medium-scale patterns in the input data.
  - **5x5 filters**: Detect high-scale patterns.
- Outputs from different filters are concatenated depth-wise, enhancing the model's ability to capture diverse features.

➢ **Auxiliary Classifiers**:



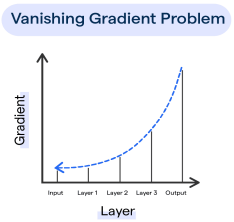
- Added during training to provide extra supervision, assisting in gradient flow and improving convergence.
- Only utilized during the training phase, not during prediction, preventing model complexity during inference.

❖ **ResNet (Residual Network)**

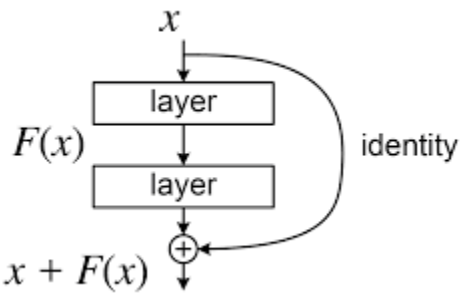
➢ ResNet (Residual Network) is a deep learning architecture designed to tackle the **vanishing gradient problem**, which occurs in very deep networks.

Vanishing Gradient Problem:

In deep neural networks, as the number of layers increases, the gradients used in backpropagation (the process that updates the weights of the network) tend to get smaller as they move backward through the layers. This leads to minimal updates for earlier layers, making it difficult for the network to learn properly. As a result, the network may stop improving, or its performance may stagnate, especially during the training of deep models. The vanishing gradient problem can prevent deep networks from effectively learning, particularly in earlier layers, limiting the depth of networks that can be trained successfully.



➢ **Residual Connections (Skip Connections)**:

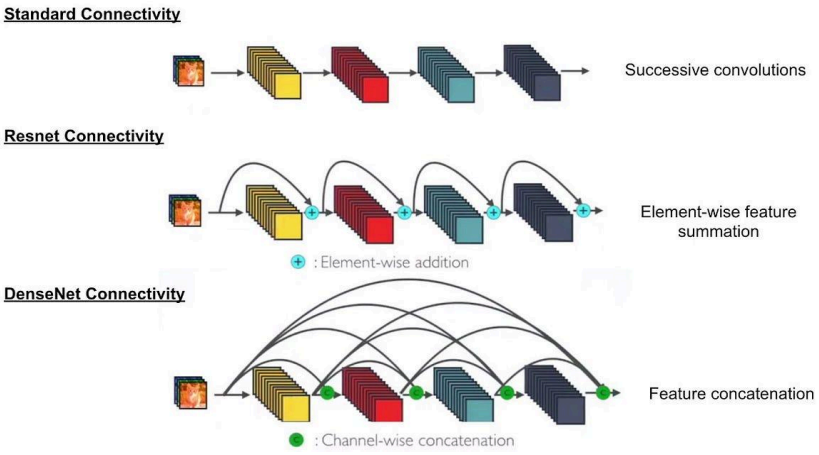


- To solve this(**vanishing gradient problem)**, ResNet introduces **residual connections**, also known as **skip connections**. These connections implement the formula:

$$y=f(x)+x$$

- Here, the input x is passed directly to the output y, bypassing one or more layers . This means that instead of learning a full mapping of the input x to the output y, the network only needs to learn the residual (or difference), f(x), between the input and the output.

❖ **DenseNet**



➢ Features **dense connections**, where every layer is connected to every other layer in a feed-forward fashion.

➢ **Feature Reuse**:
- Each layer utilizes feature maps generated by all preceding layers, promoting efficient learning and reducing redundancy.

➢ **Transition Layers**:
- Introduced between dense layers to downsample feature maps (compression), reducing their size.
- This layer effectively halves the number of parameters, enhancing model efficiency while maintaining performance.