Convolutional Neural Networks (CNNs)

Suvinava Basak

December 29, 2024

Abstract

Artificial intelligence has been making tremendous strides in closing the gap between human and computer capabilities. Both amateurs and researchers work on many facets of the area to achieve incredible results. The field of computer vision is one of several such fields. The goal of this field is to make it possible for machines to see and perceive the world similarly to humans. They will then be able to use this knowledge for a variety of tasks, including Natural Language Processing, Media Recreation, Image and Video Recognition, Image Analysis and Classification, Recommendation Systems, and more. Deep Learning's contributions to computer vision have been developed and refined throughout time, primarily using an algorithm, **Convolutional Neural Network**.

In this book, we will dig deeper into the realm of CNN and explore many things, starting from the **historical perspective** to the building blocks of a CNN along with their **mathematical foundations**. We will also look at different CNN architectures, e.g., **LeNet**, **AlexNet**, **VGGNet**, **GoogleNet**, **ResNet**, **MobileNet**, **EfficientNet** etc. and some of the real-life applications. Finally, we end this book with some of the current research and future trends of CNNs.

Contents

1 Introduction

Convolutional Neural Networks (CNNs) are a type of deep learning model **designed to recognize patterns in visual data**, such as images and videos. Unlike traditional neural networks, CNNs exploit the spatial and hierarchical patterns present in the data to efficiently learn features. CNNs consist of layers that perform convolutions, mathematical operations that filter and extract features such as edges, textures, and shapes from images. These networks typically have three main types of layers: **convolutional layers**, **pooling layers** (to reduce the image size), and **fully connected layers** (for final classification). CNNs are particularly powerful for tasks like image recognition, object detection, and facial recognition, as they automatically learn relevant features from the raw input data.

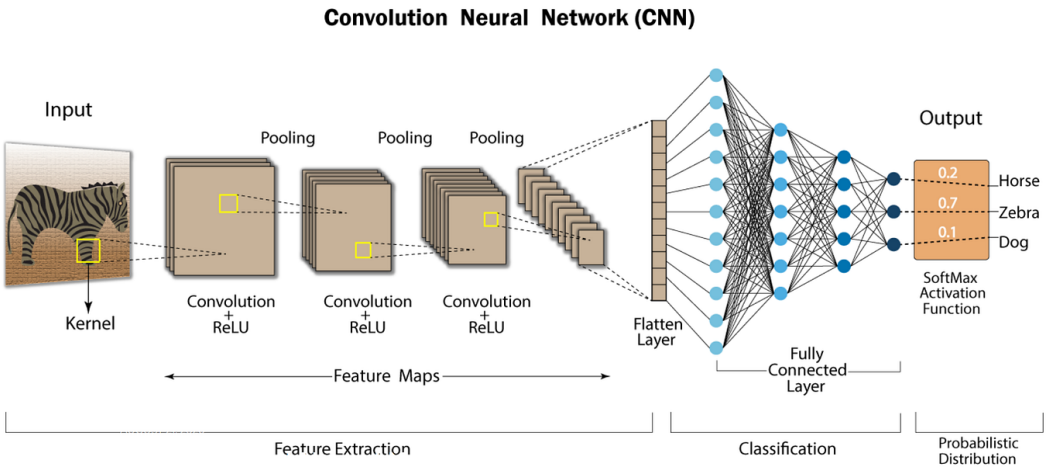

Figure 1.1: A typical Convolutional Neural Network

1.1 Key Characteristics of CNNs

- **Convolutions**: A mathematical operation that extracts patterns such as edges and textures.
- **Hierarchical Feature Learning**: CNNs learn low-level features (e.g., edges) in initial layers and complex patterns (e.g., objects) in deeper layers.
- **Parameter Sharing**: Convolutions reduce the number of parameters, making them computationally efficient.
- **Translation Invariance**: Recognize patterns regardless of their position in the input.

For instance, CNNs are capable of classifying an image of a cat even if the cat appears in different parts of the image.

1.2 Why CNNs are important in Machine Learing

A **Convolutional Neural Network (ConvNet/CNN)** can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. **The pre-processing required in a ConvNet is much lower as compared to other classification algorithms**. While in primitive methods, filters are hand-engineered; with enough training, ConvNets have the ability to learn these filters/characteristics.

CNNs have revolutionized the field of computer vision and beyond by enabling machines to:

- **Understand visual content**: Perform image classification, object detection, and segmentation.
- **Process structured data efficiently**: Learn features directly from raw input, reducing the need for manual feature engineering.
- **Achieve human-level accuracy**: Outperform traditional methods in tasks like face recognition, medical imaging, and natural language processing.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

1.3 Why ConvNets over Feed-Forward Neural Nets?

An image is nothing but a matrix of pixel values. So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes?

For extremely basic binary images, this method might show an average precision score while class prediction but would fail miserably for complex images with dependent pixels.

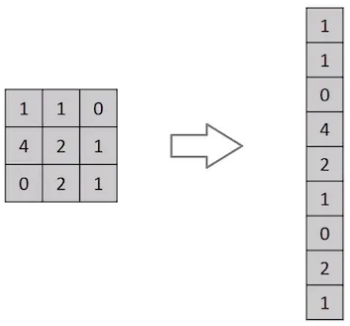

Figure 1.2: Flattening of a 3x3 image matrix into a 9x1 vector

A ConvNet can **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and the reusability of weights.

Spatial Dependency means a pixel's value is influenced by nearby pixel's value in image. This is because generally they all belong to same color because they are from same object. **Temporal dependency** comes in videos. When a frame changes to next, if there is not a lot of movement in objects, pixel's values remain same.

Aspect	ANN	CNN
Input Representation	Flat, requires feature extraction	Structured (e.g., images as grids of pixels)
Architecture	Fully connected layers	Convolutional + pooling layers + fully connected layers
Parameter Count	High (independent weights for each connection)	Low (shared weights through convolutions)
Feature Learning	Manual or less efficient	Automatic and hierarchical
Applications	Generic machine learning	Specialized for spatial and structured data

Table 1.1: Difference between ANNs and CNNs

1.4 Visual Representation of CNN workflow

The image illustrates a simple Convolutional Neural Network (CNN) designed to classify handwritten digits from the **MNIST dataset**. The MNIST dataset contains 28×28 grayscale images of handwritten digits, with the goal to classify them into one of the 10 output classes (from 0 to 9), i.e., to predict which digit (from 0 to 9) is represented in each image. Below are the sequence of layers the input images are passed through:

Input Layer: Each input image is 28×28 pixels, where each pixel represents a grayscale value ranging from 0 (black) to 255 (white). The input image is then fed into the CNN for processing.

Convolutional Layer: The first step in the CNN is the convolutional layer, where a set of small filters or kernels is applied to the input image. These filters scan the image and detect basic features like edges or corners. This process results in feature maps that highlight the presence of certain features in different parts of the image.

Pooling Layer: Next, a pooling layer reduces the spatial dimensions of the feature maps, typically using max pooling. This operation retains the most important information while reducing the amount of data, making the network more computationally efficient.

Flattening and Fully Connected Layers: After pooling, the output is flattened into a one-dimensional vector and passed through fully connected layers. These layers combine

the learned features and classify the image into one of the 10 classes (digits 0 to 9).

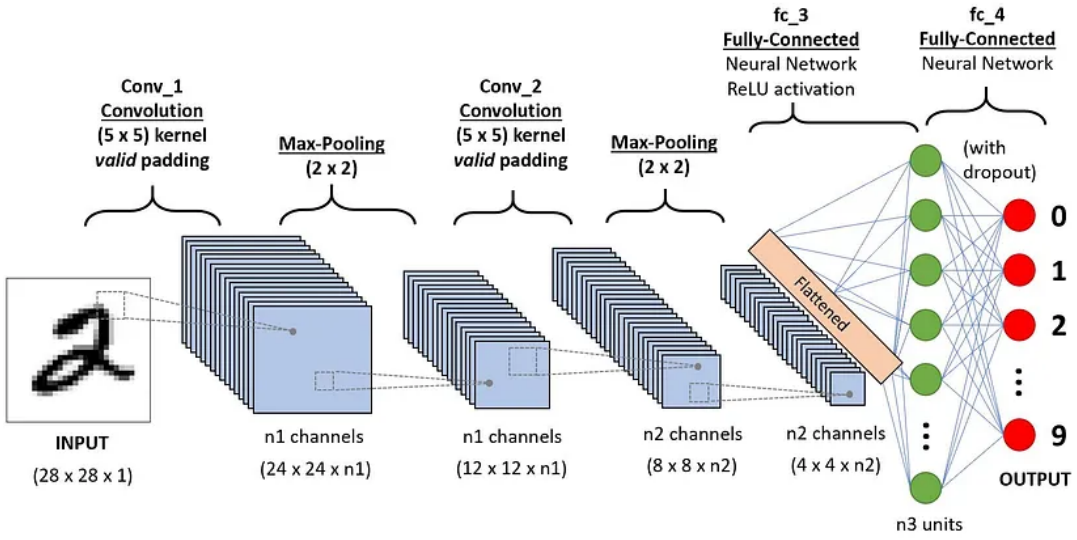

Figure 1.3: A CNN architecture to classify handwritten digits from the MNIST dataset

Output Layer: The final layer is the output layer, where a **softmax** activation function assigns probabilities to each class. The class with the highest probability is the predicted digit.

Training and Classification: The CNN is trained using labeled MNIST images and adjusts its internal parameters through backpropagation to minimize prediction errors. Once trained, the model can accurately classify unseen handwritten digits.

2 Historical Perspective of CNNs

2.1 Origin of CNNs

Convolutional Neural Networks (CNNs) trace their origins to early work in neural networks and pattern recognition. The concept of using convolutions for image analysis emerged from the broader study of artificial neural networks.

Early Pioneers and Inspirations

- **1970s-1980s: Neocognitron and Hebbian Learning**
  The Neocognitron, proposed by Kunihiko Fukushima in 1980, is often considered one of the first models that prefigured CNNs. It introduced the idea of hierarchical layers that could extract visual features, similar to the function of the human visual system. However, the Neocognitron lacked a mechanism for training via backpropagation.[1]

- **1986: Backpropagation and Neural Networks**
  The introduction of backpropagation by David Rumelhart, Geoffrey Hinton, and Ronald Williams in 1986 was a significant advancement. It provided a practical way to train multilayer networks by calculating gradients and adjusting weights, forming the foundation for modern CNN training.[2]

2.2 Key Milestones in CNN Development

2.2.1 LeNet (1998) - The First Major CNN

In 1998, **Yann LeCun** and his team introduced **LeNet-5**, a CNN architecture that successfully tackled the problem of digit recognition, particularly for the **MNIST dataset**. LeNet's architecture, as introduced by them, demonstrated the potential of CNNs in recognizing visual patterns.[3] [4]

Key components of LeNet-5:

- **Convolutional Layers**: Extracting low-level features such as edges.

- **Pooling Layers**: Reducing the size of the feature maps while retaining essential information.

- **Fully Connected Layers**: Final decision-making layers for classification.

2.2.2 AlexNet (2012) - A Breakthrough in Deep Learning

The introduction of **AlexNet** by **Alex Krizhevsky**, **Ilya Sutskever**, and **Geoffrey Hinton** in 2012 marked a breakthrough in deep learning, achieving a significant performance improvement in the **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**. This success was largely due to the deep architecture and the use of GPUs for training, which accelerated the computation process.[5]

Key features in AlexNet:

- **ReLU Activation**: Allowed faster convergence compared to traditional activation functions like sigmoid.
- **GPU Utilization**: Enabled the training of a large model on a massive dataset.

2.2.3 VGGNet (2014) - Deeper Architectures

VGGNet, developed by the **Visual Geometry Group (VGG)** at Oxford, demonstrated that deeper CNN architectures could provide better results. With 16 to 19 layers, VGGNet showed that increasing depth helped improve the model's ability to recognize more complex patterns.[6]

Key features of VGGNet:

- **Uniform 3x3 Convolutional Filters**: Simplified the architecture while maintaining performance.
- **Increased Depth**: Leveraging more layers enabled the model to capture more detailed features.

2.2.4 GoogleNet, InceptionV1 (2014)

GoogleNet (InceptionV1), introduced in 2014 by researchers at Google, is a groundbreaking convolutional neural network (CNN) architecture designed for efficient, large-scale image recognition tasks. Its key innovation is the **Inception module**, which performs multi-scale convolutions (1×1, 3×3, and 5×5) in parallel, allowing the network to capture features at various scales while maintaining computational efficiency. To reduce the model's complexity, 1×1 convolutions are used for dimensionality reduction before applying larger convolutions. With 22 layers, GoogleNet was deeper than previous architectures like AlexNet, yet more computationally efficient. This architecture won the 2014 ILSVRC (ImageNet Large Scale Visual Recognition Challenge) with state-of-the-art accuracy, paving the way for further advancements in deep learning.[7]

Key features of GoogleNet:

- Introduced the **Inception module**, which performs multi-scale convolutions in parallel (1×1, 3×3, 5×5).
- Used 1×1 convolutions to reduce dimensionality and computational cost.
- Much deeper (22 layers) but computationally efficient compared to VGGNet.

2.2.5 ResNet (2015) - Solving Vanishing Gradient Problem

The introduction of **ResNet** by **Kaiming He** and colleagues in 2015 solved the problem of training very deep networks. By using **skip connections**, ResNet alleviated the vanishing gradient problem, allowing for networks with up to 152 layers.[8]

Key features of ResNet:

- **Residual Learning**: Skip connections help in passing gradients efficiently during backpropagation.
- **Batch Normalization**: Improved the stability of the training process.

2.2.6 MobileNet (2017)

MobileNet is a family of lightweight convolutional neural network (CNN) architectures designed by Google for efficient image classification on mobile and embedded devices with limited computational resources. It is optimized to achieve a good trade-off between accuracy and computational cost, making it suitable for real-time applications on devices with constraints such as smartphones, drones, and IoT devices.[9]

Key features of MobileNet:

- **Depthwise Separable Convolutions**: Reduces computation by splitting convolutions into depthwise and pointwise operations.
- **Width Multiplier**: Scales the number of channels in each layer to adjust model size and performance.
- **Resolution Multiplier**: Adjusts the input image resolution for balancing accuracy and computational cost.
- **Efficient for Real-Time Applications**: Optimized for mobile devices, suitable for tasks like image classification and object detection.

2.2.7 EfficientNet (2019)

EfficientNet is a family of convolutional neural networks introduced by **Mingxing Tan** and **Quoc V. Le** in 2019, designed to achieve state-of-the-art performance while optimizing efficiency. It uses a novel **compound scaling method** that uniformly scales a network's depth, width, and input resolution in a balanced way to maximize accuracy and efficiency. Unlike previous models that scale only one dimension (e.g., depth or width), EfficientNet finds the best combination of all three. The base model, **EfficientNet-B0**, is built using a lightweight architecture with **MBConv blocks** (inspired by MobileNet), and larger variants (B1–B7) scale up for better accuracy while maintaining computational efficiency. EfficientNet, through proper scaling, achieves higher accuracy on benchmarks like ImageNet with significantly fewer parameters and FLOPs compared to models like ResNet and Inception, making it ideal for both resource-constrained devices and high-performance applications.[10]

Key features of EfficientNet:

- Introduced a **compound scaling method** to optimize the trade-off between network depth, width, and resolution.
- Achieved state-of-the-art performance on ImageNet with fewer parameters and FLOPs compared to ResNet and Inception-based models.
- Includes variants from EfficientNet-B0 to B7, each scaling up computational cost and accuracy.

2.3 CNNs in the Modern Era

The Rise of Transfer Learning and Pretrained Models
With the advancements in computational power and large-scale datasets, CNNs have continued to improve. Pretrained models such as **Inception**[7] and **EfficientNet**[10] have played a major role in enabling transfer learning, where models trained on large datasets are adapted for specific tasks.

Transfer Learning
Transfer learning allows practitioners to leverage models trained on large datasets (like ImageNet) and fine-tune them for new, domain-specific tasks. This approach saves time and computational resources while achieving high performance.

2.4 Relevance of CNNs to Artificial Neural Networks (ANNs)

CNNs are an extension of **traditional Artificial Neural Networks (ANNs)**, which generally rely on fully connected layers for classification. Unlike ANNs, CNNs use shared weights, convolutions, and pooling layers to process spatial information more efficiently.

Key differences from ANN:

- **Weight Sharing**: In CNNs, filters (kernels) are shared across the entire input, making them computationally more efficient compared to fully connected layers in ANNs.[11]
- **Hierarchical Feature Learning**: CNNs learn low-level features in the initial layers and combine them into more abstract concepts in deeper layers. This hierarchical feature learning is not a core principle of traditional ANNs.

3 Building Blocks of CNNs

To understand how CNNs work, it's crucial to break down their architecture into core components. These blocks — **Convolutional Layers**, **Pooling Layers**, **Activation Functions**, **Fully Connected Layers**, and **Dropout Layers** — work together to extract and classify patterns in data.

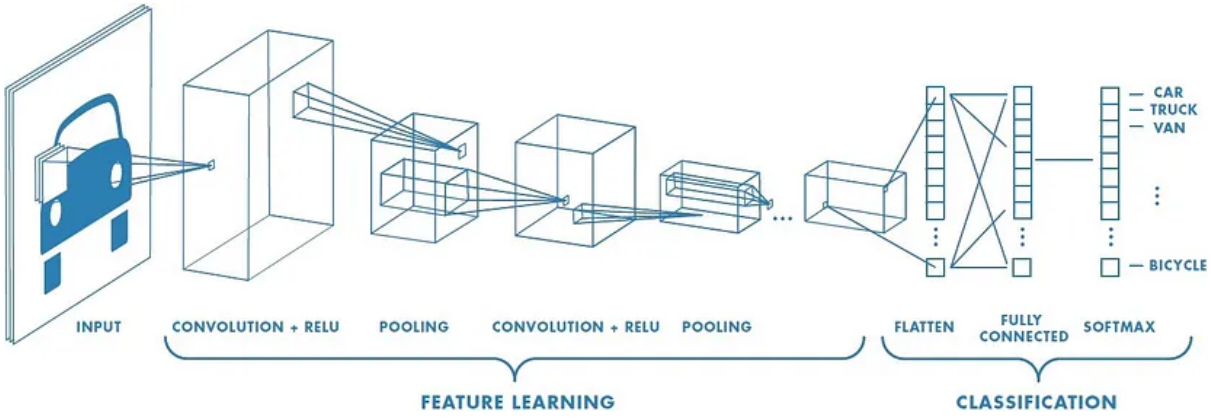

Figure 3.1: A typical Convolutional Neural Network architecture

An input image can usually be represented as a matrix with corresponding pixel values. It can have multiple such matrices depending on the number of color channels of the input image.

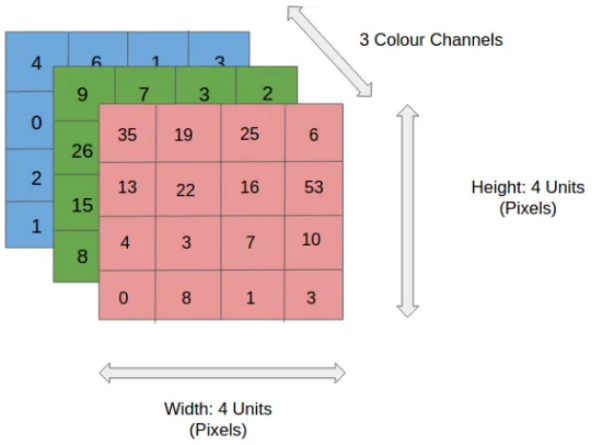

Figure 3.2: A sample $4 \times 4 \times 3$ RGB input image

In figure 3.2, we have an RGB image that has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist

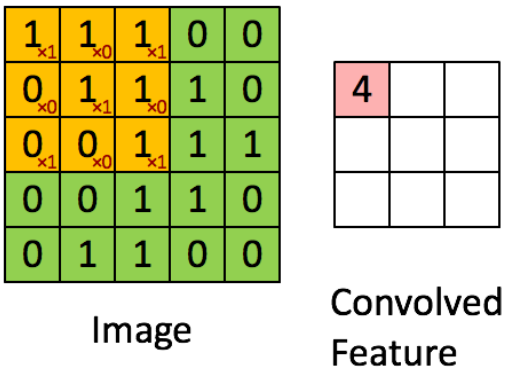

Figure 3.3: Convolving a $5 \times 5 \times 1$ image with a $3 \times 3 \times 1$ kernel to get a $3 \times 3 \times 1$ convolved feature

— Grayscale, RGB, HSV, CMYK, etc.

You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The primary function of a ConvNet is to transform images into a more manageable representation while preserving the essential features needed for accurate predictions. This is crucial when designing an architecture that is good at feature extraction as well as remains scalable to large datasets.

3.1 Convolutional Layers: The Core Component

The convolutional layer is the most fundamental building block of CNNs. It is responsible for detecting spatial features (e.g., edges, textures) by applying a **kernel** or **filter** to the input data.

3.1.1 How Convolution Works

Convolution is the heart of CNNs. It is a mathematical operation where a filter slides (or convolves) over the input matrix to compute a feature map. In other words, the filter or kernel is applied to the input image to compute feature maps. The operation captures local patterns, which are then combined to form higher-level patterns.

In figure 3.3, a kernel (filter matrix) of size 3×3 slides over the input matrix (image) of size 5×5 to produce an output feature map (convolved feature) of size 3×3.

In general, any image matrix can be represented as Height $\times$ Width $\times$ No. of channels. In figure 3.3, no. of channels is considered as 1. For an input image matrix of size $W_{in} \times H_{in}$, the corresponding dimension of the output feature map (convolved feature) after convolving with a square kernel of dimension k can be formularized as below:

$$W_{out} = W_{in} - k + 1 \tag{3.1}$$

$$H_{out} = H_{in} - k + 1 \tag{3.2}$$

In figure 3.3, the green section resembles our $5 \times 5 \times 1$ input image, I. The kernel/filter,

K, represented in color yellow, is selected as a $3 \times 3 \times 1$ matrix:

$$\mathrm{K} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

3.1.2 Is Kernel Always a Square Matrix?

The kernel in a Convolutional Neural Network (CNN) does not always have to be square. Although square kernels (e.g., 3×3, 5×5) are the most commonly used due to their simplicity and symmetry, kernels can also have rectangular or non-square shapes depending on the application:

- **Rectangular Kernels**:
  - 3×5: Used when more detailed feature extraction is needed in one dimension (e.g., in text or speech recognition).
  - $1 \times n$ or $n \times 1$: Common in 1D convolutions or when processing time-series data or text.
- **Horizontal/Vertical Feature Extraction**:
  - A 1×3 kernel emphasizes horizontal patterns, while a 3×1 kernel highlights vertical patterns.
- **Custom Shapes**:
  - For specialized use cases like detecting specific geometric patterns in images or reducing computational complexity in one dimension.

3.1.3 Stride and Padding

In figure 3.3, the kernel shifts 9 times, every time performing an element-wise multiplication operation (**Hadamard Product**) between kernel K and the portion P of the image matrix I over which the kernel is hovering.

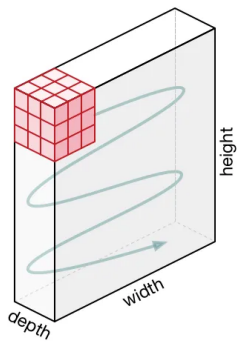

Figure 3.4: Typical movement of a kernel matrix during convolution

Figure 3.4 shows the typical movement of a kernel matrix. The filter moves to the right with a certain **Stride Length** until it parses the complete width. Moving on, it hops

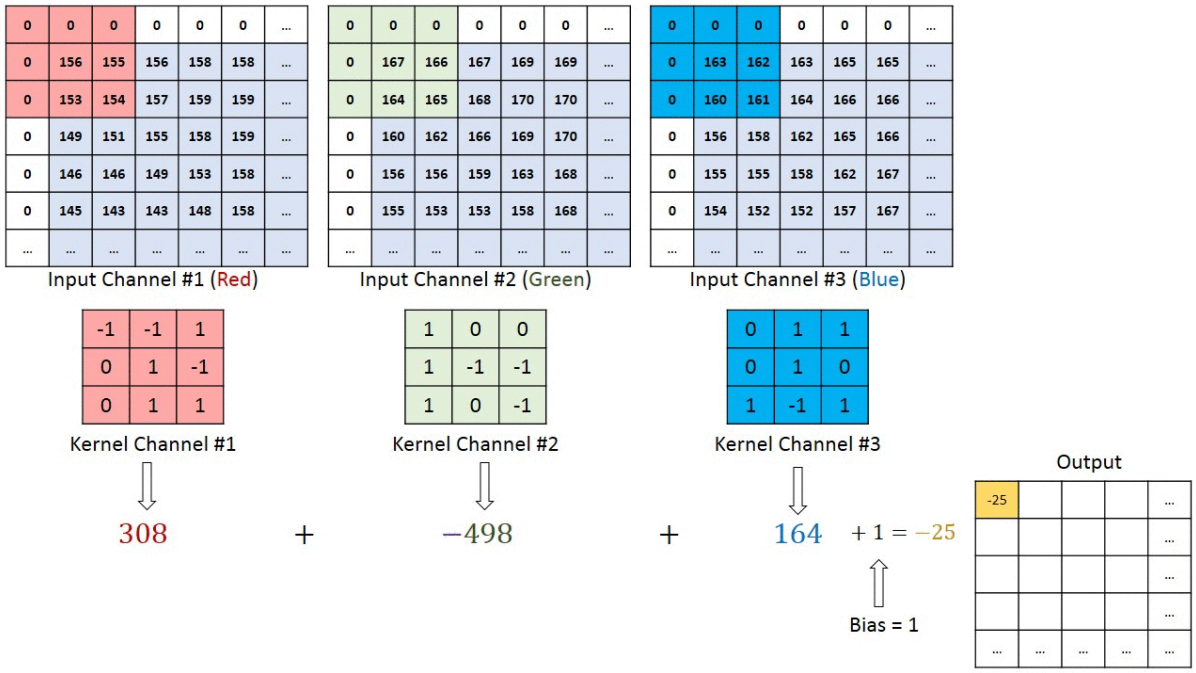

Figure 3.5: Convolution operation on a $M \times N \times 3$ image matrix with a $3 \times 3 \times 3$ Kernel

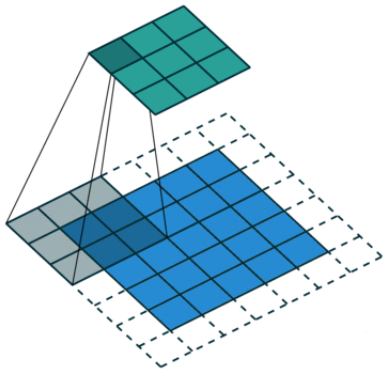

Figure 3.6: Convolution operation with stride length = 2

down to the beginning (left) of the image with the same Stride Length and repeats the process until the entire image is traversed.

As we can see in figure 3.5, in the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between K_n and I_n stack ($[K_1, I_1]$; $[K_2, I_2]$; $[K_3, I_3]$) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

The number of shifts of the kernel depends on **stride length**. In our example in figure 3.3 and figure 3.5, stride length = 1, which can also be called **non-strided**. Stride length denotes how many cells the kernel will skip before the next element-wise multiplication operation. Figure 3.6 shows a convolution operation with stride length = 2.

There are two types of results to the convolution operation — one in which the convolved feature is reduced in dimensionality as compared to the input (as explained by 3.1 and

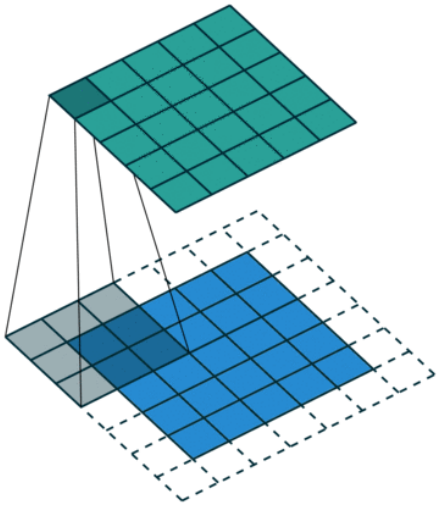

Figure 3.7: Same padding - $5 \times 5 \times 1$ image is padded with 0s to create a $6 \times 6 \times 1$ image

3.2), and the other in which the dimensionality is either increased or remains the same. This is done by applying **Valid Padding** in the case of the former, or **Same Padding** in the case of the latter.

When we augment the $5 \times 5 \times 1$ image into a $6 \times 6 \times 1$ image by applying a 0-vector at each end (**zero-padding**) and then apply the $3 \times 3 \times 1$ kernel over it, we find that the convolved matrix turns out to be of dimensions $5 \times 5 \times 1$. Hence the name — **Same Padding**.

On the other hand, if we perform the same operation without padding, we are presented with a matrix that has dimensions of the Kernel ($3 \times 3 \times 1$) itself — this is **Valid Padding**.

Considering stride length and padding, the dimension of the output feature map (convolved feature) after convolving an input image matrix of size $W_{in} \times H_{in}$ with a square kernel of dimension k, will be:

$$W_{out} = \frac{W_{in} - k + 2P}{S} + 1 \tag{3.3}$$

$$H_{out} = \frac{H_{in} - k + 2P}{S} + 1 \tag{3.4}$$

where, W_{in} and H_{in} are the dimensions of the input image, W_{out} and H_{out} are the dimensions of the convolved feature matrix, k is the kernel size, P is the padding size (amount of zero-padding applied to the input), and S is the stride length of the convolution operation.

3.1.4 Importance of Stride and Padding

Stride and padding are crucial parameters in convolutional neural networks (CNNs) as they impact the output size, feature extraction, and computational efficiency.

- **Stride determines the step size of the convolutional filter**; a stride of 1 is used when fine-grained feature extraction and preserving spatial details are impor-

tant, while larger strides (e.g., 2) are employed to reduce the spatial dimensions and computational load.

- Padding involves adding extra pixels (typically zeros) around the input to **control the spatial size of the output**. "Same" padding is used when maintaining the input size is desired, while "valid" padding is chosen to avoid adding any extra pixels, resulting in smaller outputs.

Selecting appropriate stride and padding values depends on the task: for applications like object detection, smaller strides and "same" padding help retain spatial details, whereas larger strides and "valid" padding may be preferred for classification tasks requiring reduced spatial dimensions.

In summary, the purpose of the convolution operation is to **extract high-level features**, such as edges, from the input image. Convolutional Neural Networks (ConvNets) are not restricted to a single convolutional layer. Typically, the initial convolutional layer focuses on capturing low-level features like edges, colors, and gradient orientations. As additional layers are added, the network progressively adapts to detect high-level features, ultimately creating an architecture capable of comprehensively understanding the images in the dataset, much like humans do.

3.1.5 Key Characteristics of Convolution Layer

A convolutional layer typically has the below characteristics:

1. **Parameter Sharing**: Each filter is applied uniformly across the input, drastically reducing the number of parameters compared to fully connected layers.[12]

2. **Local Receptive Field**: Filters capture patterns in small regions of the input, making the model computationally efficient.[13]

3. **Strides and Padding**:

   - **Stride** controls how much the filter shifts during convolution.
   - **Padding** adds extra pixels (e.g., zeros) around the input to preserve dimensions after convolution.

3.2 Pooling Layers: Down sampling

Similar to the Convolutional Layer, the Pooling layer also reduces the spatial dimensions of the Convolved Feature (output feature map). This dimensionality reduction helps decrease the number of computations required to process the data further. It also helps mitigate the risk of overfitting by extracting dominant features that are invariant to rotation and position, thus maintaining the process of effectively training the model.

3.2.1 Types of Pooling

While there are two types of pooling available, Max Pooling is mostly used in real-world applications.

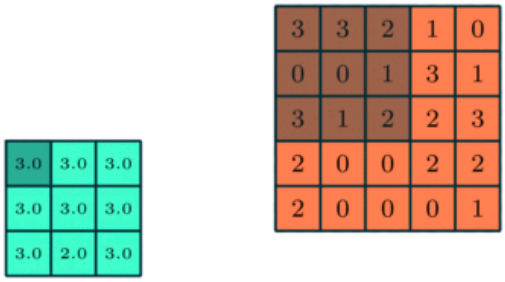

Figure 3.8: 3×3 pooling on 5×5 convolved feature

- **Max Pooling**: Takes the maximum value from each window of the input. It **captures the most prominent features.**[14]
- **Average Pooling**: Computes the average of values in the window. This method is less common in modern architectures.[3]

Figure 3.9 shows how different types of pooling will extract features from the output feature map (convolved feature).

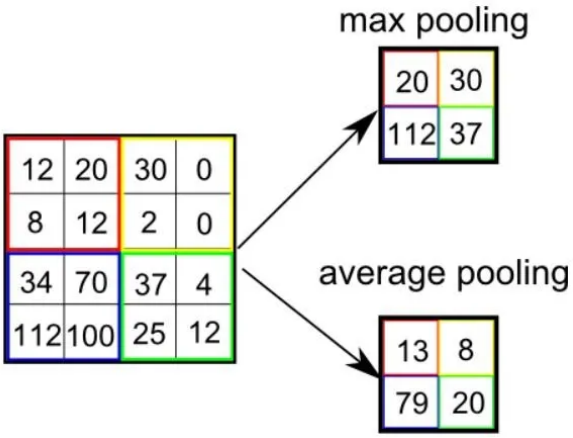

Figure 3.9: Types of Pooling

3.2.2 Mathematical Example of Max Pooling

Given a 2×2 pooling filter applied to a 4×4 input will be as below:

For an input matrix $\begin{bmatrix} 1 & 3 & 2 & 1 \\ 5 & 6 & 8 & 2 \\ 7 & 2 & 9 & 3 \\ 4 & 5 & 1 & 0 \end{bmatrix}$, max pooling output matrix will be $\begin{bmatrix} 6 & 8 \\ 7 & 9 \end{bmatrix}$.

3.2.3 Max Pooling v/s Average Pooling

In addition to dimensionality reduction, Max Pooling also performs as a **Noise Suppressant** by completely discarding the noisy activations. On the other hand, Average Pooling

simply performs dimensionality reduction as a noise-suppressing mechanism. That is why, in general, **Max Pooling performs a lot better than Average Pooling**. However, the choice between Max Pooling and Average Pooling depends on the specific use case and the nature of the task at hand.

In simpler words, **Max Pooling leads to sparse feature maps but emphasizes the most important activations** (by discarding noise). Since it highlights only the most prominent features (e.g., edges, textures, etc.) that are critical for decision-making, Max Pooling performs better in tasks requiring robustness to noise in input data and preservation of sharp, critical features (e.g., object detection and classification, etc.).

On the other hand, **Average Pooling produces dense feature maps with less emphasis on extreme activations**, but better captures overall patterns by smoothing the outputs by averaging activations. Therefore, if the goal is to generate smooth and generalized (balanced) feature representations and the input data is relatively clean and lacks significant noise, Average Pooling is a better choice.

The Convolutional Layer and the Pooling Layer come together and form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-level details even further, but at the cost of more computational power. For example, figure 1.3 in chapter 1 shows a CNN architecture (to classify handwritten digits from the MNIST dataset) using 2 Convolutional Layers and 2 Max Pooling layers one after another.

So far we have enabled the model to understand the input image and extract important features (e.g., edges, textures, angles, etc.) in a much simpler matrix form in lower dimensions. Moving on, we will flatten this output and feed it to a regular Neural Network for classification purposes. But before that, we will touch upon another important topic which is important for the Convolutional Layer - **Activation Functions**.

3.3 Activation Functions

Activation functions in the convolutional layer play a crucial role in Convolutional Neural Networks (CNN). Activation functions introduce non-linearity into the model, enabling it to learn complex patterns and relationships in the data. Without activation functions, the CNN would behave as a linear model, regardless of the number of layers, limiting its ability to solve non-linear problems such as image classification, object detection, and natural language processing.

By enabling the network to learn non-linear transformations and hierarchical features, activation functions are essential for the success of deep learning models in solving complex real-world problems.

Below are some commonly used activation functions:

- **ReLU (Rectified Linear Unit) function**: It ensures sparsity by retaining only positive activations, which not only introduces non-linearity but also improves com-

putational efficiency and helps avoid the vanishing gradient problem.[15]

$$\text{ReLU}(z) : \mathbb{R} \to \mathbb{R}_{\geq 0}, \quad \text{ReLU}(z) := \max(0, z) \tag{3.5}$$

- **Sigmoid function**: Commonly used in the output layer of a binary classification task, as it maps values to the range $(0, 1)$, representing probabilities. Suitable when the model needs to output probabilities or interpret values as likelihoods, but it can suffer from vanishing gradients.[16][2]

$$\sigma : \mathbb{R} \to (0, 1), \quad \sigma(z) := \frac{1}{1 + e^{-z}} \tag{3.6}$$

- **Hyperbolic Tangent (tanh) function**: Preferred in hidden layers when inputs are zero-centered, as it maps the input in the range $(-1, 1)$, which helps faster convergence during training. It has more efficient gradient flow than sigmoid function.

$$\tanh : \mathbb{R} \to (-1, 1), \quad \tanh(z) := \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.7}$$

- **Hard hyperbolic tangent (hardtanh) function**: Provides a faster approximation of the traditional tanh function, maintaining a balance between non-linearity and computational simplicity.

$$\text{hardtanh}(z) : \mathbb{R} \to [-1, 1], \quad \text{hardtanh}(z) := \max(-1, \min(1, z)) \tag{3.8}$$

- **Identity function**: While it does not introduce non-linearity, it is occasionally used in skip connections or output layers when no transformation is needed.

$$\text{id}(z) : \mathbb{R} \to \mathbb{R}, \quad \text{id}(z) := z \tag{3.9}$$

- **Softmax function**: Typically applied at the output layer for multi-class classification tasks, converting raw scores into normalized probabilities.

$$\text{softmax}(z) : \mathbb{R}^n \to \mathbb{R}^n, \quad \text{softmax}(z)_j := \frac{e^{z_j}}{\sum_{j=1}^{n} e^{z_j}} \tag{3.10}$$

Together, these activation functions allow CNNs to learn non-linear transformations, capture intricate data patterns, and perform complex real-world tasks effectively.

3.4 Fully Connected Layers

After the earlier layers of convolution and pooling extract local features from the input data, the output from the last pooling layer is flattened into a column vector. This flattened output is fed into a fully connected layer (FC) to aggregate the learned spatial features for classification purposes.

Fully connected layers play a crucial role in combining the high-level features extracted by the convolutional layers to identify global patterns and relationships. They enable the network to synthesize the extracted features into meaningful representations for decision-making, such as classifying an input image into one of several categories. The addition of

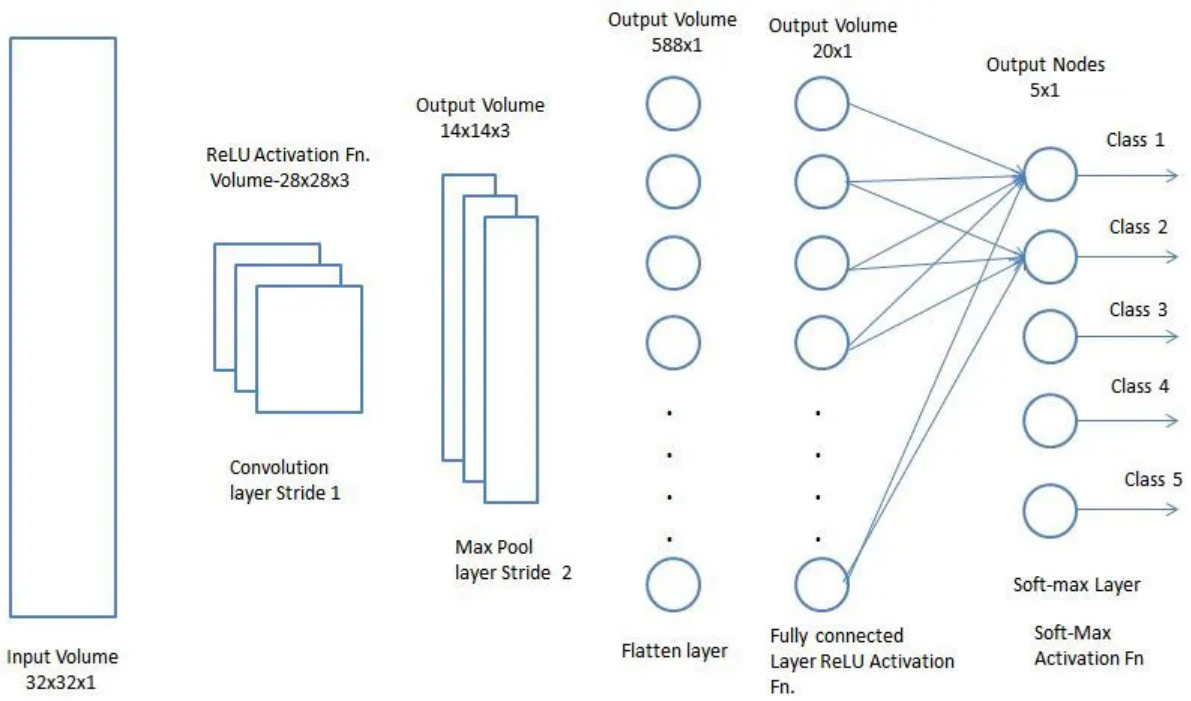

Figure 3.10: Feeding the flattened final output to regular NN

fully connected layers is a computationally efficient way to learn non-linear combinations of these high-level features.

After the input is flattened and passed through the fully connected layers, backpropagation is performed at each training iteration. Over multiple epochs, the model gradually learns to distinguish between prominent and subtle features in the data. This iterative process enables the network to accurately recognize patterns and classify inputs.

In a classification task, the fully connected layer produces a set of values that correspond to the likelihood of the input belonging to each class. These values are converted into probabilities using an activation function like softmax, enabling the network to predict the most probable class for the given input.[5]

By connecting spatially extracted features to the classification process, fully connected layers act as the final step in converting raw input data into meaningful decisions.

3.5 Dropout Layer

Dropout is a regularization technique to prevent overfitting by randomly "dropping out" (setting to zero) a fraction of neurons' outputs during training.

A dropout layer in a Convolutional Neural Network (CNN) uses this technique to force the network to learn redundant, robust features by not relying heavily on specific neurons. The dropout rate, typically between 0.2 and 0.5, determines the proportion of neurons to drop. During testing, all neurons are used, and their outputs are scaled by the dropout rate to maintain consistency with training. This helps improve the network's generalization capability on unseen data.[17]

4 Mathematics behind CNNs

4.1 Convolution Operation: The Core of Feature Extraction

The convolution operation is a key mathematical foundation of CNNs. It allows the network to detect patterns such as edges, corners, and textures in the input data.

Mathematical Definition

As stated earlier, convolution is a mathematical operation where a filter slides (or convolves) over the input matrix to compute a feature map. In a 2D convolution, a kernel (filter matrix) K of size $k \times k$ slides over the input matrix (image) I to produce an output feature map O. The convolution operation is mathematically represented as:

$$O(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i+m, j+n) \cdot K(m,n) \tag{4.1}$$

where (i,j) iterates over the image and $O(i,j)$ is the output pixel value of the feature map O at the pixel position (i,j).

4.2 Stride and Padding

4.3 Pooling: Dimensionality Reduction

4.4 Backpropagation in CNNs

4.5 Optimization Techniques

4.6 Regularization Techniques

5 Different CNN Architectures

6 Real-life Applications of CNNs

6.1 Image Classification

6.2 Object Detection

6.3 Semantic Segmentation

6.4 Generative Applications

6.5 Autonomous Systems

6.6 Natural Language Processing

6.7 Medical Imaging

6.7.1 Role of CNNs

6.7.2 Mathematical Foundations

6.7.3 Applications in Tumor Detection

6.7.4 Impact

6.7.5 Challenges

7 Current Research and Future Trends

7.1 Current Research Areas in CNNs

7.1.1 Lightweight CNN Architectures

7.1.2 Self-supervised Learning with CNNs

7.1.3 Hybrid CNN Architectures

7.2 Future Trends in CNNs

7.2.1 Explainable AI (XAI) for CNNs

7.2.2 Federated Learning for CNNs

7.2.3 Integration with Neuromorphic Computing

7.2.4 CNNs in Quantum Computing

7.2.5 Cross-disciplinary Applications

8 Conclusion

8.1 CNN's Impact on Deep Learning

8.2 Challenges and Future Directions

8.2.1 Challenges

8.2.2 Future Directions

8.3 Open Problems in the Domain

8.3.1 Dynamic and Continual Learning

8.3.2 Multimodal Integration

8.3.3 Scalable CNN Training

8.3.4 Handling Uncertainty in Predictions

8.3.5 Few-shot and Zero-shot Learning

Bibliography

[1] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by backpropagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[4] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Proceedings of the IEEE Conference on Neural Networks, 1990*, pp. 396–404, 1990.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, pp. 1097–1105, 2012.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015. arXiv:1409.1556.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 1–9, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv*, vol. abs/1704.04861, 2017.

[10] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[11] Y. Bengio, "Learning deep architectures for ai," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[13] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv*, 2016.

[14] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision (ECCV)*, pp. 818–833, Springer, 2014.

[15] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 807–814, 2010.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research (JMLR)*, vol. 15, pp. 1929–1958, 2014.

Further references

Apart from the above references, several YouTube videos (by Brandon Rohrer [1] [2], Serrano.Academy, 3Blue1Brown, deeplizard, The Julia Programmina Language, MIT, Standford University) are relevant to know more about Convolutional Neural Networks in detail.
There are also some blogs on `medium.com` which can serve as a good starting point to delve deeper into the realm of CNN:

- *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* by Sumit Saha
- *An Introduction to different Types of Convolutions in Deep Learning* by Paul-Louis Pröve
- *Gentle Dive into Math Behind Convolutional Neural Networks* by Piotr Skalski
- *Intuitively Understanding Convolutions for Deep Learning* by Irhum Shafkat

Happy Learning!