

Introduction to Convolution Neural Network

A **Convolutional Neural Network (CNN)** is a type of Deep Learning Neural Network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.

When it comes to Machine Learning, [Artificial Neural Networks](#) perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use [Recurrent Neural Networks](#) more precisely an [LSTM](#), similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

Using Artificial Neural Networks (ANNs) for image-related tasks is not ideal due to the following reasons:

1. High Number of Parameters

Images usually have a lot of pixels. For example, a 100x100 grayscale image has 10,000 pixels, and each would be a separate input for the ANN. If the image is colored, this number increases threefold. ANNs need a huge number of parameters to handle this, leading to:

- High computational cost
 - Overfitting due to too many learnable parameters
-

2. Loss of Spatial Information

ANNs treat every input feature independently. In images, the relative position of pixels is crucial (e.g., edges, shapes), but ANNs cannot capture this spatial information effectively.

3. Inefficiency in Detecting Patterns

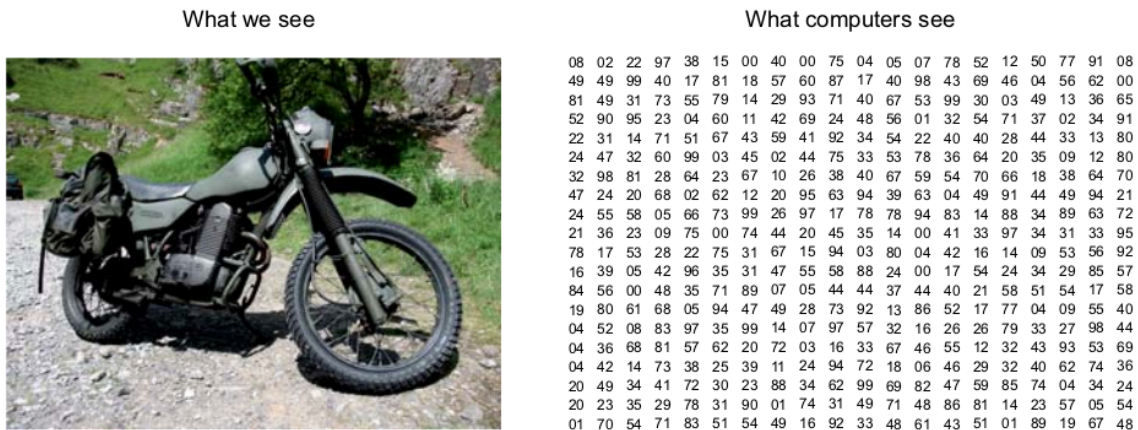
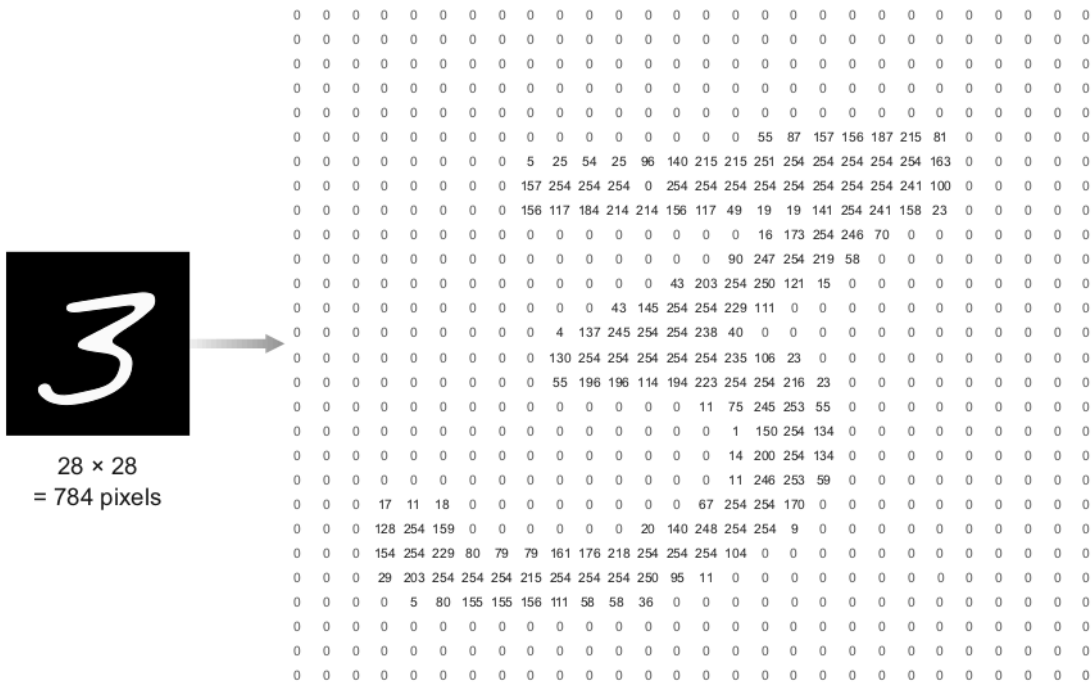
Images have patterns like edges, textures, and objects that are localized. ANNs do not have the mechanisms (like convolutional layers in CNNs) to efficiently detect and process these patterns.

4. Lack of Hierarchical Feature Extraction

In image tasks, we often need to detect simple features (e.g., edges) first and then combine them into more complex structures (e.g., shapes). Convolutional Neural Networks (CNNs) excel at this hierarchical feature extraction, but ANNs lack this capability.

How computers see images

When we look at an image, we see objects, landscape, colors, and so on. But that's not the case with computers. Consider figure 1.16. Your human brain can process it and immediately know that it is a picture of a motorcycle. To a computer, the image looks like a 2D matrix of the pixels' values, which represent intensities across the color spectrum. There is no context here, just a massive pile of data.



The image in figure 1.16 is of size 24×24 . This size indicates the width and height of the image: there are 24 pixels horizontally and 24 vertically. That

means there is a total of 576 (24×24) pixels. If the image is 700×500 , then the dimensionality of the matrix will be (700, 500), where each pixel in the matrix represents the intensity of brightness in that pixel. Zero represents black, and 255 represents white.

Color images

In grayscale images, each pixel represents the intensity of only one color, whereas in the standard RGB system, color images have three channels (red, green, and blue). In other words, color images are represented by three matrices: one represents the intensity of red in the pixel, one represents green, and one represents blue (figure 1.17).

As you can see in figure 1.17, the color image is composed of three channels: red, green, and blue. Now the question is, how do computers see this image? Again, they see the matrix, unlike grayscale images, where we had only one channel. In this case, we will have three matrices stacked on top of each other; that's why it's a 3D matrix.

The dimensionality of 700×700 color images is (700, 700, 3). Let's say the first matrix represents the red channel; then each element of that matrix represents an intensity of red color in that pixel, and likewise with green and blue. Each pixel in a color image has three numbers (0 to 255) associated with it. These numbers represent intensity of red, green, and blue color in that particular pixel.

If we take the pixel (0,0) as an example, we will see that it represents the top-left pixel of the image of green grass. When we view this pixel in the

color images, it looks like figure 1.18. The example in figure 1.19 shows some shades of the color green and their RGB values.

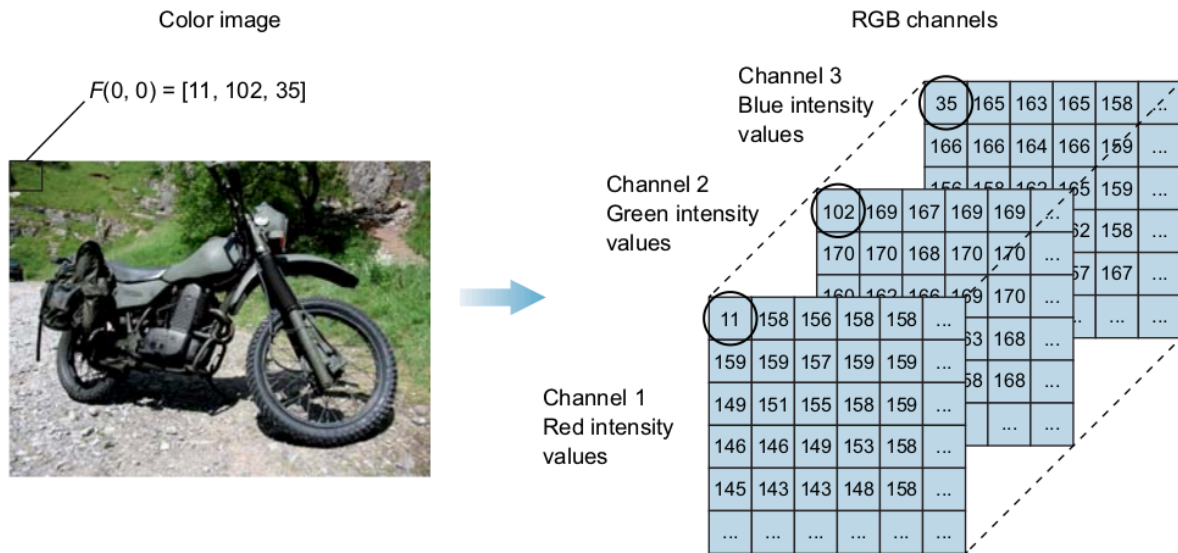


Figure 1.17 Color images are represented by red, green, and blue channels, and matrices can be used to indicate those colors' intensity.

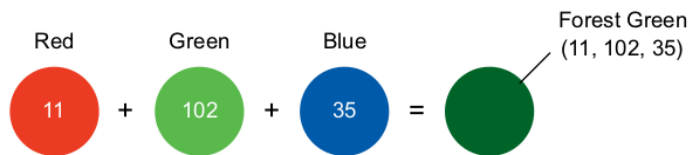


Figure 1.18 An image of green grass is actually made of three colors of varying intensity.

Forest HEX #0B6623 RGB 11 102 35	Forest green Codes: HEX #0B6623 RGB 11 102 35	Mint HEX #98FB98 RGB 152 251 152	Mint green Codes: HEX #98FB98 RGB 152 251 152
Olive HEX #708238 RGB 112 130 56	Olive green Codes: HEX #708238 RGB 112 130 56	Lime HEX #C7EA46 RGB 199 234 70	Lime green Codes: HEX #C7EA46 RGB 199 234 70
Jungle HEX #29AB87 RGB 41 171 135	Jungle green Codes: HEX #29AB87 RGB 41 171 135	Jade HEX #00A86B RGB 0 168 107	Jade green Codes: HEX #00A86B RGB 0 168 107

Figure 1.19 Different shades of green mean different intensities of the three image colors (red, green, blue).

Applications of Convolutional Neural Networks (CNNs) with Examples

CNNs are widely used for tasks involving images and patterns because they can understand visual data efficiently. Here are some practical applications explained in simple terms:

1. Image Classification

- **What It Does:** Identifies and labels images into categories.
 - **Example:**
 - An app that scans photos and tells if it's a dog, cat, or a car (e.g., Google Photos).
 - Medical diagnosis: Classifying X-rays as "healthy" or "disease present."
-

2. Object Detection

- **What It Does:** Identifies objects in an image and marks their location.
 - **Example:**
 - Autonomous vehicles: A self-driving car detects pedestrians, traffic signs, and other vehicles on the road.
 - Security systems: Detects unauthorized people in restricted areas using camera feeds.
-

3. Image Segmentation

- **What It Does:** Divides an image into segments to locate objects or boundaries more precisely.
- **Example:**

- Medical imaging: Highlighting tumors in an MRI scan for precise analysis.
 - Photo editing tools: Selecting and separating objects like a person from the background for edits.
-

4. Face Recognition

- **What It Does:** Identifies and verifies faces in images.
 - **Example:**
 - Unlocking phones with face recognition (e.g., Face ID on iPhones).
 - Social media tagging: Automatically tagging friends in photos.
-

5. Video Analysis

- **What It Does:** Processes video frames to analyze activities or objects.
 - **Example:**
 - Sports: Tracking players' movements and actions during a match.
 - Surveillance: Detecting unusual activities in security footage.
-

6. Handwriting Recognition

- **What It Does:** Reads and interprets handwritten text from images.
 - **Example:**
 - Digitizing written forms or notes (e.g., Google Lens or OCR tools).
-

7. Recommendation Systems

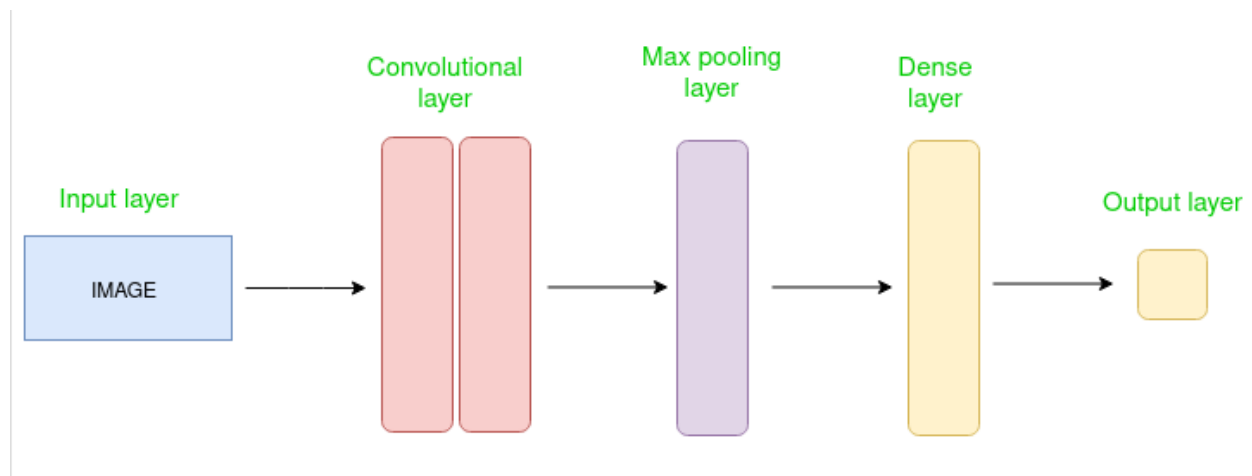
- **What It Does:** Analyzes visual content to suggest similar items.
- **Example:**
 - E-commerce: Suggesting products like clothes or shoes by analyzing images you like (e.g., Amazon, Flipkart).

Convolution Neural Network

Convolutional Neural Network (CNN) is the extended version of artificial neural networks (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example visual datasets like images or videos where data patterns play an extensive role.

CNN Architecture

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.



Simple CNN architecture

The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

Layers Used to Build ConvNets

A complete Convolution Neural Networks architecture is also known as convnets. A convnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

Types of layers: datasets

Let's take an example by running a convnets on of image of dimension $32 \times 32 \times 3$.

- **Input Layers:** It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.
- **Convolutional Layers:** This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2 , 3×3 , or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension $32 \times 32 \times 12$.

- **Activation Layer:** By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are **RELU**: $\max(0, x)$, **Tanh**, **Leaky RELU**, etc. The volume remains unchanged hence output volume will have dimensions $32 \times 32 \times 12$.
- **Pooling layer:** This layer is periodically inserted in the convnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2×2 filters and stride 2, the resultant volume will be of dimension $16 \times 16 \times 12$.

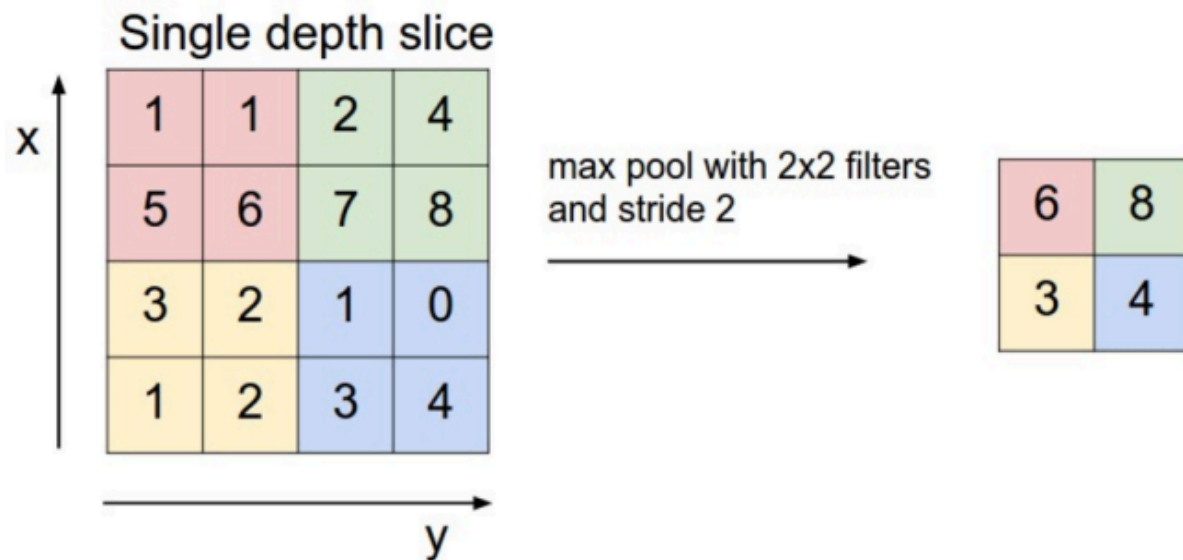


Image source: cs231n.stanford.edu

- **Flattening:** The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.
- **Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.

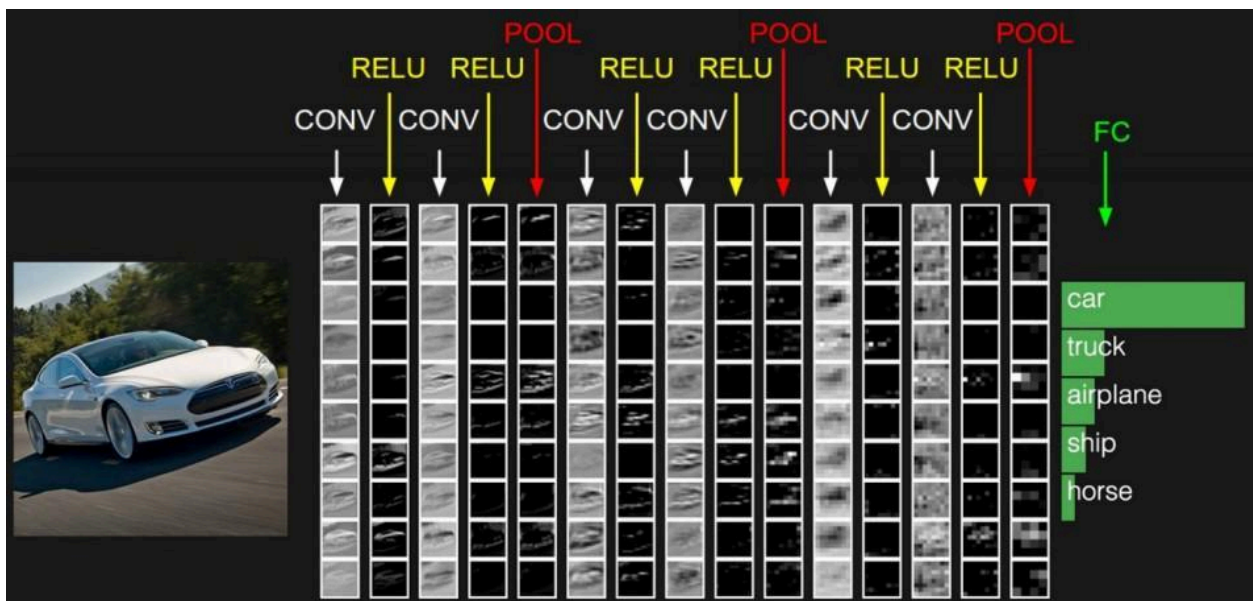


Image source: cs231n.stanford.edu

- **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

Convolution Operation in Convolutional Neural Networks (CNNs)

Convolution is the core operation in Convolutional Neural Networks (CNNs). It is designed to extract features such as edges, textures, and patterns from images while preserving spatial relationships. This operation makes CNNs highly effective for image and pattern recognition tasks.

1. What is Convolution?

Convolution is a mathematical operation that combines two functions to produce a third function. In the context of CNNs, it involves applying a small matrix (called a filter or kernel) to an input (such as an image) to create a feature map.

2. Key Components of Convolution

a. Input

- The image or data being processed.
- Represented as a matrix of pixel values. For a grayscale image, each pixel value is an intensity value (e.g., 0-255).

b. Filter (Kernel)

- A small matrix (e.g., 3x3 or 5x5) with specific values used to detect certain features.
- Examples:
 - **Edge detection filter:** Highlights edges in the image.
 - **Sharpening filter:** Enhances fine details in the image.

c. Stride

- The number of steps the filter moves across the input image.

- A stride of 1 moves the filter one pixel at a time, while a stride of 2 skips every alternate pixel.

d. Padding

- Adding extra pixels around the input image to control the output size.
 - **Same Padding:** Adds padding to ensure the output size matches the input size.
 - **Valid Padding:** No padding is added; the output size is smaller than the input.

e. Output (Feature Map)

- The result of the convolution operation. It highlights the features detected by the filter.

3. How Convolution Works

1. **Place the Filter on the Input Image:** Align the top-left corner of the filter with the top-left corner of the input.
2. **Element-Wise Multiplication:** Multiply each value in the filter with the corresponding value in the input image.
3. **Summation:** Add all the results of the multiplication.
4. **Assign the Result:** Place the summed value in the corresponding position in the feature map.
5. **Move the Filter:** Shift the filter by the stride value and repeat the process.

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

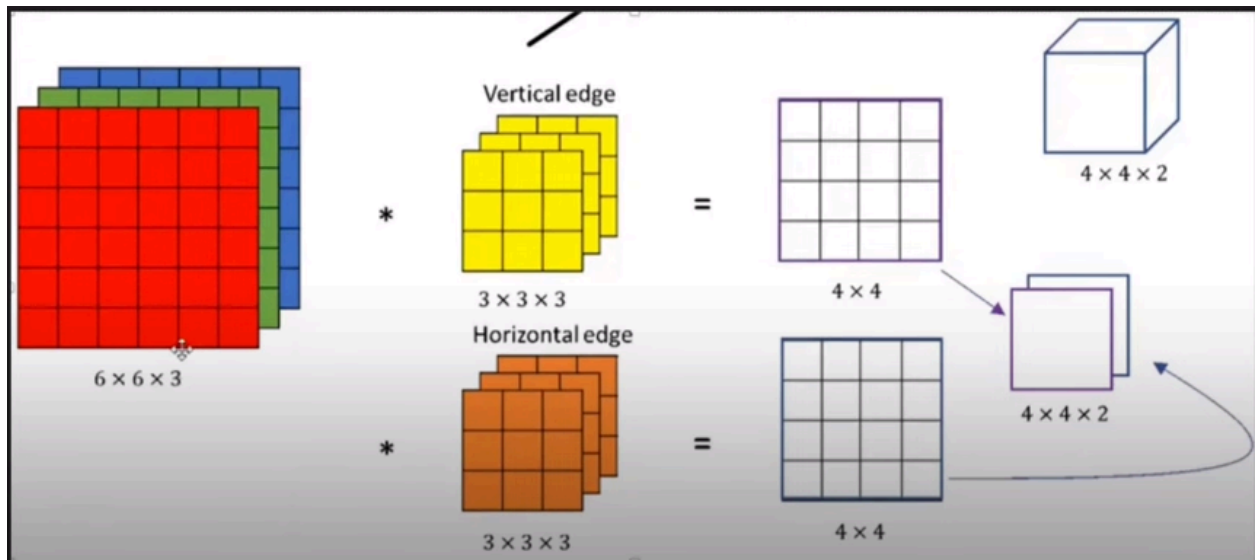
1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

*

=



Visualization_tool: <https://deeplizard.com/resource/pavq7noze2>

Padding in Convolutional Neural Networks (CNNs)

Padding refers to the process of adding extra pixels (usually zeros) around the border of an input image before applying the convolution operation. It helps control the spatial dimensions of the output feature map.

Why Padding is Important

1. Preserves Spatial Dimensions:

- Without padding, the spatial dimensions of the input reduce with each convolution.
- For example, a 5×5 input with a 3×3 filter reduces to a 3×3 output.
- Padding ensures the output dimensions are the same as the input dimensions when needed.

2. Prevents Information Loss:

- Convolution near the edges of an image loses some information because fewer pixels are covered.
- Padding includes these edge pixels, ensuring no loss of critical features.

3. Allows Deeper Networks:

- Maintaining consistent dimensions makes it easier to stack multiple convolutional layers.

Types of Padding

1. Valid Padding:

- No padding is added.
- The output size is smaller than the input size.
- Formula for output dimensions: suppose if we have a image of dimension $n \times n$ and kernel of size $m \times m$ then output dimension is $(n-m+1) \times (n-m+1)$
- Example: A 5x5 image with a 3x3 filter and no padding results in a 3x3 feature map.

2. Same Padding:

- Padding is added to keep the output size equal to the input size.
- The amount of padding depends on the filter size and stride.
- Formula for padding: $(n+2p-m+1) \times (n+2p-m+1)$
- Example: A 5x5 image with a 3x3 filter and same padding produces a 5x5 feature map.

3. Custom Padding:

- Specific padding values are chosen, not necessarily to preserve dimensions.
- Used in tasks with unique requirements.

Task for finding the dimension of the feature map after applying padding.

Code_implementation: [🔗 keras_padding_implementation.ipynb](#)

Stride in Convolutional Neural Networks (CNNs)

Stride in CNNs refers to the number of pixels by which the filter (kernel) moves across the input image during the convolution operation. It is an essential hyperparameter that controls how much the feature map is reduced and how the convolution process progresses over the input.

Why is Stride Important?

1. Controls Feature Map Size:

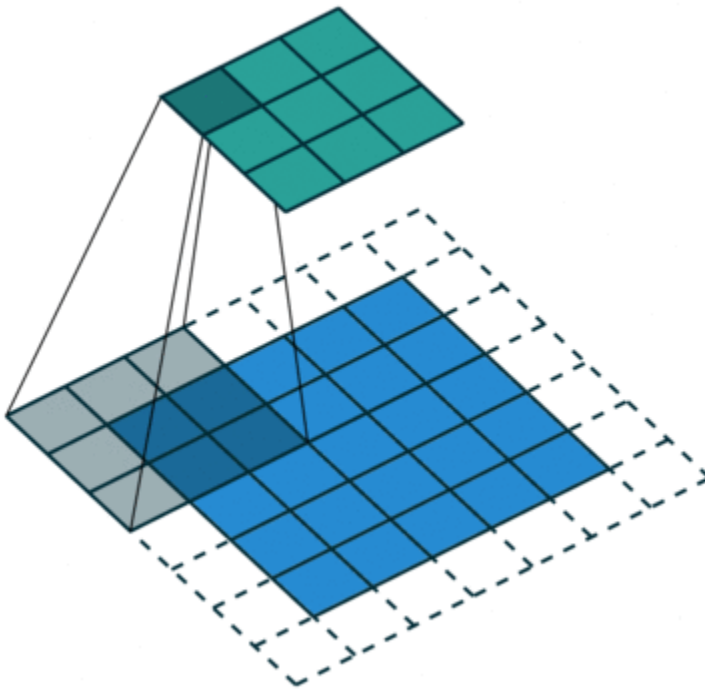
- Larger strides reduce the size of the output feature map, which decreases computational complexity.
- Smaller strides generate larger feature maps, capturing more details.

2. Feature Extraction:

- By skipping pixels (larger stride), the network focuses on more prominent features.
- Smaller strides allow capturing finer details.

How Stride Works

1. The filter is placed at the top-left corner of the input image.
2. It moves horizontally and vertically by the stride value.
3. At each position, convolution is performed (element-wise multiplication and summation).
4. The output of the operation is placed in the corresponding position of the feature map.



Effect of Stride on Output Size

The size of the output feature map is determined using the formula:

$$\text{Output Size} = \left\lfloor \frac{\text{Input Size} - \text{Filter Size} + 2 \times \text{Padding}}{\text{Stride}} \right\rfloor + 1$$

- **Input Size:** Dimensions of the input image (e.g., height and width).
- **Filter Size:** Dimensions of the kernel (e.g., 3x3).
- **Padding:** Pixels added around the input (if any).
- **Stride:** Number of pixels the filter moves in each step.

Impact of Stride on CNNs

1. Smaller Strides:

- More overlapping regions between filter applications.
- Larger feature maps with more detailed features.
- Higher computational cost.

2. Larger Strides:

- Less overlapping and faster processing.
- Smaller feature maps, focusing on high-level features.
- Potential loss of finer details.

Code: [🔗 keras_padding_stride_implementation.ipynb](#)

Pooling in Convolutional Neural Networks (CNNs)

Why do we need pooling operations?

1. Memory issue.
2. Translation Variance.

Pooling is a crucial operation in CNNs used to reduce the spatial dimensions of feature maps while preserving their most important information. It helps in down-sampling the input, making the network computationally efficient, less prone to overfitting, and capable of detecting features in a more generalized manner.

1. What is Pooling?

Pooling is a mathematical operation that replaces a region of the feature map with a summary statistic (e.g., maximum or average value). It reduces the spatial size of the feature map while retaining the critical features, thereby helping the network focus on prominent patterns.

2. Types of Pooling

a. Max Pooling

- Selects the maximum value from a pooling window.
- Highlights the strongest activation in the region, often corresponding to the most prominent feature.

b. Average Pooling

- Computes the average of all values within the pooling window.
- Retains a smoother representation of the feature map.

c. Global Pooling

- Reduces the entire feature map to a single value by applying max or average pooling over the entire map.
 - Often used at the end of a CNN to replace fully connected layers.
-

3. Key Parameters in Pooling

a. Pooling Window (Kernel)

- The size of the region to apply pooling (e.g., 2×2 , 3×3).

b. Stride

- The step size to move the pooling window across the feature map.
- A stride of 2 skips every alternate position, reducing the size of the output.

c. Padding

- Determines whether the feature map is padded before pooling.
 - Helps control the output size.
-

4. How Pooling Works

Step-by-Step Process

1. **Choose a Pooling Window:** Define the size (e.g., 2×2) of the region to summarize.
 2. **Apply Pooling:** Slide the pooling window over the feature map, summarizing the values in each region.
 3. **Move the Window:** Shift the window by the stride and repeat the pooling operation.
 4. **Output the Pooled Map:** The result is a down-sampled feature map that retains the most critical information.
-

6. Benefits of Pooling

a. Dimensionality Reduction

- Reduces the size of the feature maps, making computations faster and requiring less memory.

b. Translation Invariance

- Ensures that small shifts or distortions in the input do not affect feature detection.

c. Focus on Dominant Features

- Helps the network concentrate on the most critical patterns while ignoring less relevant details.


d. Prevents Overfitting

- Simplifies the model by reducing the number of parameters and activations, which can reduce overfitting.
-

7. Applications of Pooling

- **Image Classification:** Generalizes feature detection by reducing the spatial size of images.
 - **Object Detection:** Enhances the ability to detect objects in images, regardless of their position.
 - **Speech Recognition:** Down-samples feature maps of audio signals to improve computational efficiency.
-

Demo: <https://deeplizard.com/resource/pavq7noze3>

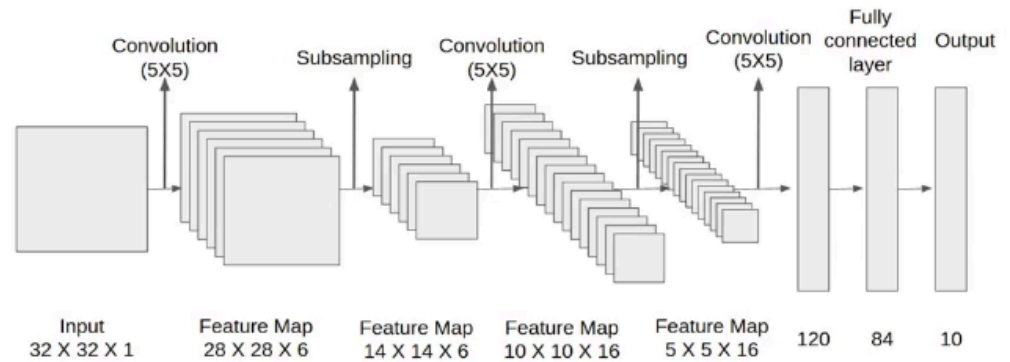
Code:  `keras_pooling_implementation.ipynb`

CNN Architectures

1. LeNet

In 1998, Lecun et al. introduced a pioneering CNN called LeNet-5.¹ The LeNet-5 architecture is straightforward, and the components are not new to you (they were new back in 1998); you learned about convolutional, pooling, and fully connected layers. The architecture is composed of five weight layers, and hence the name LeNet-5: three convolutional layers and two fully connected layers.

Here is the final architecture of the Lenet-5 model.



Architecture Details

Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

Let's understand the architecture in more detail. The first layer is the input layer with feature map size 32X32X1. Then we have the first convolution layer with 6 filters of size 5X5 and stride is 1. The activation function used at this layer is tanh. The output feature map is 28X28X6. Next, we have an average pooling layer with filter size 2X2 and stride 2. The resulting feature map is 14X14X6. Since the pooling layer doesn't affect the number of channels.

After this comes the second convolution layer with 16 filters of 5X5 and stride 1. Also, the activation function is tanh. Now the output size is 10X10X16. Again comes the other average pooling layer of 2X2 with stride 2. As a result, the size of the feature map reduced to 5X5X16. The final pooling layer has 120 filters of 5X5 with stride 1 and activation function tanh. Now the output size is 120. The next is a fully connected layer with 84 neurons that result in the output to 84 values and the activation function used here is again tanh. The last layer is the output layer with 10 neurons and Softmax function. The Softmax gives the probability that a data point belongs to a particular class. The highest value is then predicted. This is the entire architecture of the Lenet-5 model. The number of trainable parameters of this architecture is around sixty thousand.

Code: [🔗 LeNet.ipynb](#)