Understanding Convolution in CNNs: How Filters Learn Features

1. Introduction: The "Eyes" of the Network

Convolutional Neural Networks (CNNs) revolutionized computer vision by providing a robust way for machines to interpret images. Unlike a traditional Multilayer Perceptron (MLP) that treats an image as a single, long vector of pixels, a CNN preserves the spatial structure (height and width). The fundamental building block enabling this is the **Convolutional Layer**.

The heart of the convolutional layer is the **filter** (or **kernel**). This tutorial's objective is to demystify this critical component: to teach you what a convolution operation actually does, and how the filters, which initially contain random numbers, evolve into powerful, specialized **feature detectors** during training. Understanding filter mechanics is essential for anyone seeking to build, debug, or improve image recognition systems.

By the end of this tutorial, you will be able to:

- Intuitively explain the convolution operation as a process of feature extraction.
- Understand how filters learn to detect primitive features like edges and gradients.
- Analyze the impact of hyperparameters like stride and padding on feature maps.
- Implement code to extract and visualize the learned filters from a trained CNN.

All visuals in this tutorial use **grayscale or colour-blind–friendly palettes** to ensure accessibility

2. The Convolution Operation: A Sliding Feature Detector

A convolution is a simple, but powerful, mathematical operation that forms the basis of all CNNs. It is essentially a process where a small matrix of weights, the **filter (K)**, is systematically slid across the input image (I).

2.1 The Mechanism

The filter, typically 3×3 or 5×5 pixels, moves across the entire image. At each stop, the filter performs an element-wise multiplication with the corresponding patch of the input image, and all results are summed up to produce a single number in the output array, known as the **feature map** or **activation map**.

$$Output(i,j) = (I * K)(i,j) = \sum_m \quad \sum_n I\,(i-m, j-n)K(m,n)$$

This single number represents **how strongly the feature encoded in the filter is present at that specific location** in the image. If the filter matches the underlying feature (e.g., a vertical edge), the resulting sum (the activation) will be high.

2.2 Controlling the Output: Hyperparameters

Three hyperparameters critically control the size of the output feature map and the filter's movement:

- **Stride (S):** The number of pixels the filter shifts at each step. A stride of 1 means the filter moves one pixel at a time; a stride of 2 means it skips pixels, effectively downsampling the image.
- **Padding (P):** The number of zero pixels added around the border of the input image. Padding is often used to ensure the output feature map has the same spatial dimensions as the input, preventing information loss at the edges.
- **Kernel Size (K):** The dimensions of the filter (e.g., 3×3). Smaller kernels tend to be more efficient and lead to better results when stacked.

3. Filters as Learned Feature Detectors

The true power of the convolutional layer is that the values within the filter matrix are not fixed; they are the **learnable parameters** of the network. During backpropagation, the network adjusts these values to minimize the error, causing the filters to specialize.

3.1 Hierarchical Feature Learning

CNNs learn features hierarchically:

1. **Early Layers (The Focus of Our Study):** Filters in the first convolutional layer typically learn simple, generic, low-level features that are universal across all image types, such as **vertical edges, horizontal edges, diagonal lines, and color blobs**.
2. **Deeper Layers:** Filters combine the primitive features from the previous layer to detect complex patterns, such as corners, textures, and curves.
3. **Final Layers:** Deepest filters detect highly abstract, object-specific features like "an eye," "a wheel," or "a texture of fur."

3.2 Visualizing Learned Weights

By extracting the weight matrix from the first convolutional layer after training, we can visualize the filters themselves. Each filter, when plotted, reveals the pattern it has learned to look for. For instance, a filter that learned to detect vertical edges will look like a vertical line, with high weights on one side (the line) and low/negative weights on the other (the background).

This visualization provides a crucial layer of interpretability for otherwise black-box models, satisfying the need to understand how the system works internally.

4. Practical Demonstration (PyTorch)

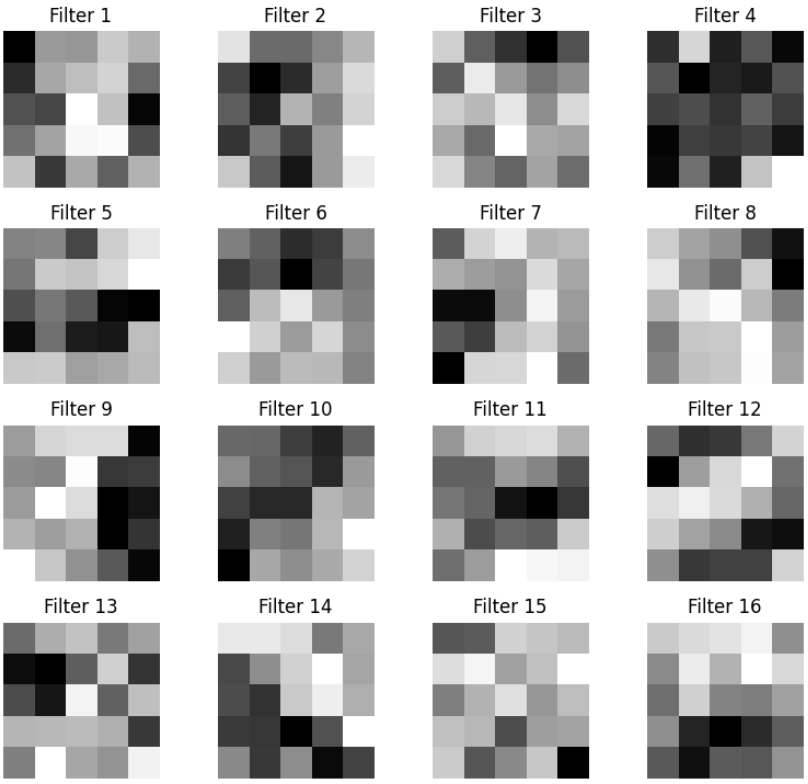
In the accompanying code notebook, we will train a small CNN on the **MNIST dataset** (grayscale images of handwritten digits) and then perform two key visualizations to understand feature learning.

Both visualizations include alt-text and captions for accessibility.

Demonstration Steps:

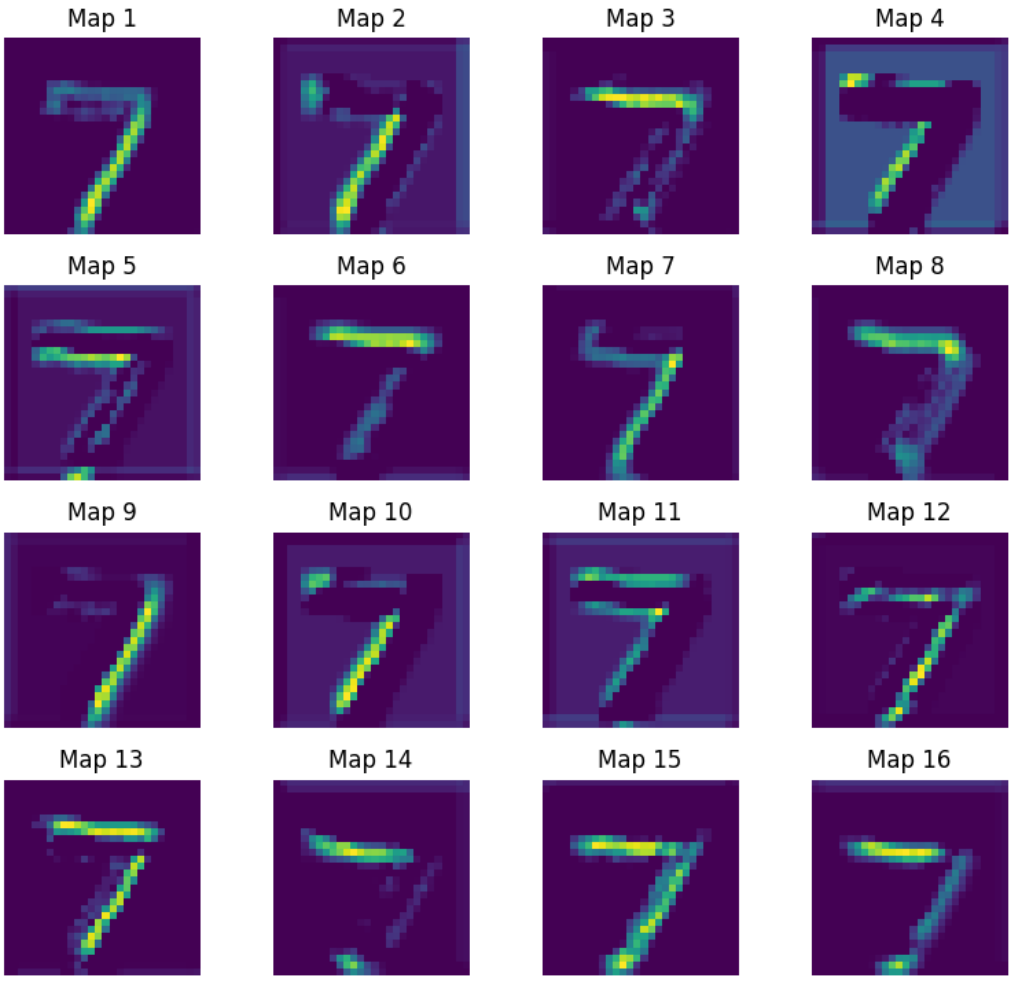
1. **Model Training:** Train a simple CNN (two convolutional layers followed by a classifier) on MNIST.
2. **Filter Extraction:** Access the weight tensor of the first convolutional layer (conv1).
3. **Figure 1: Visualization of Learned Filters:** Plot each of the learned filters (e.g., 10 or 32 filters) as a small image. We expect to see various patterns resembling edge and gradient detectors.

Figure 1: Learned Filters of the First Convolutional Layer (Conv1)



4. **Figure 2: Feature Map Activation:** Pass a sample input image (e.g., the digit '7') through the first convolutional layer and plot the resulting **feature maps**. This shows which features were *activated* for a specific input.

Figure 2: Feature Maps for Sample Digit (Output of Conv1)



These visualizations will empirically prove that the convolutional layer successfully learns meaningful, interpretable visual features.

---

5. Conclusion

The convolutional layer is the engine of the CNN, and the **filter** is its most critical component. This tutorial demonstrated that convolution is more than just a mathematical operation—it is a powerful mechanism for **local feature extraction**. The process of learning transforms random weight matrices into specialized detectors for edges and textures, which are then combined hierarchically to recognize complex objects. By visualizing these learned filters, we gain essential insights into how CNNs "see" the world, transforming the model from a black box into a transparent, feature-learning system.

GitHub Repository Link:
https://github.com/AnuragNalla/Convolution_in_CNNs_Filter_Visualization?tab=readme-ov-file

6. References

- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324. (Foundational paper on CNNs)
- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *ICLR.* (Discusses the benefits of small kernel sizes).
- Zeiler & Fergus (2014). *Visualizing and understanding convolutional networks.*
- Olah et al. (2017). *Feature visualization* (Distill.pub).
- Dumoulin & Visin (2016). *A guide to convolution arithmetic*