INF721 - Deep Learning
L10: Convolutional Neural Network

Prof. Lucas N. Ferreira
Universidade Federal de Viçosa

2024/2

1 Introduction

This lecture introduces Convolutional Neural Networks (CNNs), a class of deep learning models particularly effective for processing grid-like data such as images. CNNs address the limitations of traditional fully connected neural networks when dealing with high-dimensional inputs like images.

2 Motivation: Parameter Explosion MLPs

When processing images with Multi-Layer Perceptrons (MLPs), we encounter a significant problem known as parameter explosion. Consider an image with dimensions:

- Height (h) = 1000 pixels
- Width (w) = 1000 pixels
- Channels = 3 (for RGB color)

To process this image with an MLP, we need to flatten it into a feature vector:

$$d = h \times w \times 3 = 1000 \times 1000 \times 3 = 3,000,000$$

If we create a hidden layer with 1000 neurons, the weight matrix for this layer would have dimensions:

$$W^{[1]} = (n^{[1]}, n^{[0]}) = (1000, 3,000,000)$$

This results in 3 billion parameters for just one layer! This parameter explosion leads to two main problems:

1. **Computational Resources**: Enormous memory requirements and computational power needed.

2. **Overfitting**: With so many parameters, the network is prone to memorizing the training data rather than learning generalizable features, unless an extremely large dataset is available.

3 Introduction to Convolutions

To address the parameter explosion problem, we introduce convolutions. Convolutions allow us to process large images with a constant number of parameters, regardless of the input image size. To understand convolutions operations, we first need to introduce the concept of filters.

3.1 Filters (Kernels)

A filter, also known as a kernel, is a small matrix of weights used to transform an image. It's typically a 3x3 matrix, though other sizes are possible:

$$K = \begin{bmatrix} 12.0 & 12.0 & 17.0 \\ 10.0 & 17.0 & 19.0 \\ 9.0 & 6.0 & 14.0 \end{bmatrix}$$

3.2 Convolution

A convolution operation involves sliding a filter over an image and computing the weighted sum of the pixel values covered by the filter at each position. This operation transforms the original image I based on the filter's weights K. Formally, it can be defined as:

$$A_{ij} = \sum_m \sum_n I_{i-m,j-n} \cdot K_{mn}$$

where:

- A_{ij} is the output pixel at position (i, j)
- $I_{i+m,j+n}$ is the input pixel at position $(i + m, j + n)$
- K_{mn} is the filter weight at position (m, n)

The figure below shows an example of convolution operation:

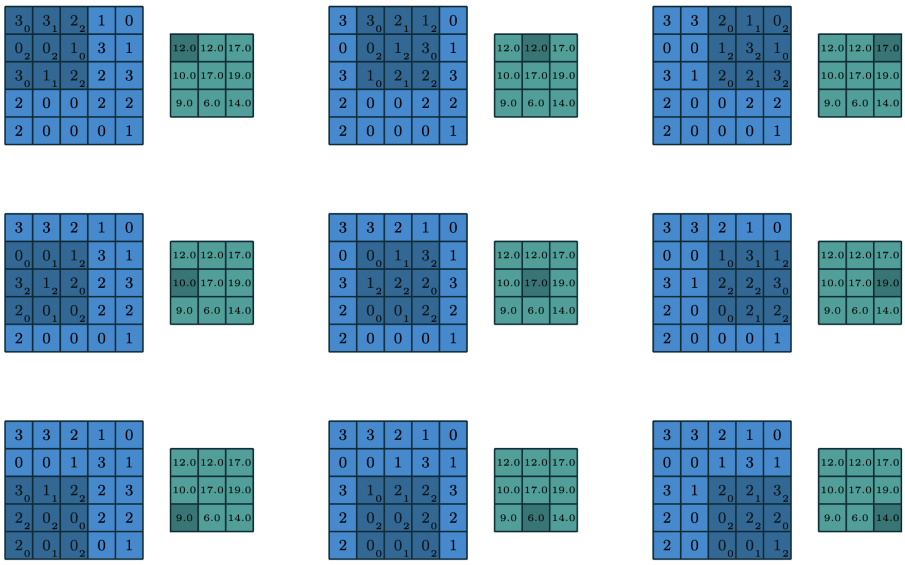

Figure 1: Example of a convolution operation

4 Edge Detection with Convolutions

Convolutions with filters can be used to extract specific features from an image. One example which is particularly importante for computer vision is edge detection. By designing specific filters, we can highlight vertical or horizontal edges in an image.

4.1 Vertical Edge Detection

$$\text{Vertical Edge Filter} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

4.2 Horizontal Edge Detection

$$\text{Horizontal Edge Filter} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

These filters can detect abrupt changes in pixel intensity, which often correspond to edges in the image. Thus, the resulting feature maps can be used as input to classic image classification algorithms, such as Support Vector Machines (SVM), for example.

5 Learning Filters with CNNs

While we can manually design filters for specific tasks like edge detection, the power of Convolutional Neural Networks lies in their ability to learn optimal filters directly from data. In a CNN, the weights of the network are organized into convolutional filters. During training, these filters are learned through backpropagation and gradient descent, allowing the network to automatically discover useful features for the given task.

6 Properties of Convolutions

6.1 Size Reduction

One important property of convolutions is that they reduce the size of the input. For an input image of size $h \times w$ and a filter of size $f \times f$, the output size is:

$$(h - f + 1) \times (w - f + 1)$$

This reduction in size can be problematic if we apply multiple convolutions, as the image might become too small.

6.2 Padding

To address the size reduction issue and give equal importance to edge pixels, we introduce padding. Padding involves adding extra pixels (usually zeroes) around the border of the input image. With padding p, the output size becomes:

$$(h + 2p - f + 1) \times (w + 2p - f + 1)$$

To preserve the input size after convolution, we can calculate the required padding as:

$$p = \frac{f - 1}{2}$$

6.3 Strided Convolutions

Another variation on the basic convolution is the strided convolution. Instead of moving the filter one pixel at a time, we can move it by a larger step, called the stride. For a stride s, the output size becomes:

$$\left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right)$$

Strided convolutions can be used to further reduce the spatial dimensions of the feature maps.

7 Convolutions Over Volumes

When dealing with color images or multi-channel feature maps, we need to extend our concept of convolutions to work over volumes.

7.1 RGB Images

For an RGB image with 3 channels, our filter must also have 3 channels. The convolution operation is performed independently on each channel and then summed to produce a single output channel. For example, consider a 5x5x3 image I and a 3x3x3 filter F with the following values:

$$I_1 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix}, I_2 = \begin{bmatrix} 2 & 4 & 6 & 8 & 10 \\ 12 & 14 & 16 & 18 & 20 \\ 22 & 24 & 26 & 28 & 30 \\ 32 & 34 & 36 & 38 & 40 \\ 42 & 44 & 46 & 48 & 50 \end{bmatrix}, I_3 = \begin{bmatrix} 3 & 6 & 9 & 12 & 15 \\ 18 & 21 & 24 & 27 & 30 \\ 33 & 36 & 39 & 42 & 45 \\ 48 & 51 & 54 & 57 & 60 \\ 63 & 66 & 69 & 72 & 75 \end{bmatrix}$$

$$F_1 = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, F_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}, F_3 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

The convolution operation at position (x, y) is computed as:

$$(I * F)_{x,y} = \sum_{i=1}^{3} \sum_{m=0}^{2} \sum_{n=0}^{2} I_i(x+m, y+n) \cdot F_i(m, n)$$

Let's compute the convolution at position (1,1):

$$\begin{aligned}
(I * F)_{1,1} &= (1 \cdot 1 + 2 \cdot 0 + 3 \cdot -1 + 6 \cdot 1 + 7 \cdot 0 + 8 \cdot -1 + 11 \cdot 1 + 12 \cdot 0 + 13 \cdot -1) \\
&\quad + (2 \cdot 0 + 4 \cdot 1 + 6 \cdot 0 + 12 \cdot 0 + 14 \cdot 1 + 16 \cdot 0 + 22 \cdot 0 + 24 \cdot 1 + 26 \cdot 0) \\
&\quad + (3 \cdot -1 + 6 \cdot 0 + 9 \cdot 1 + 18 \cdot -1 + 21 \cdot 0 + 24 \cdot 1 + 33 \cdot -1 + 36 \cdot 0 + 39 \cdot 1) \\
&= (1 - 3 + 6 - 8 + 11 - 13) + (4 + 14 + 24) + (-3 + 9 - 18 + 24 - 33 + 39) \\
&= -6 + 42 + 18 \\
&= 54
\end{aligned}$$

This operation results in a single value, 54, which is the sum of the element-wise multiplications of the 3x3x3 subvolume of the image centered at $I(1, 1)$ with the 3x3x3 filter. The output of the convolution will be a 3x3 matrix (assuming stride 1 and no padding), where each element is computed similarly:

$$\text{Feature Map} = \begin{bmatrix} 54 & 60 & 66 \\ 114 & 120 & 126 \\ 174 & 180 & 186 \end{bmatrix}$$

7.2 Multiple Filters

In practice, a convolutional layer typically applies multiple filters to its input. Each filter produces its own output channel, and these are stacked to form the output volume. For example, if we apply 2 filters to an RGB image, the output will have 2 channels, each representing a different learned feature.

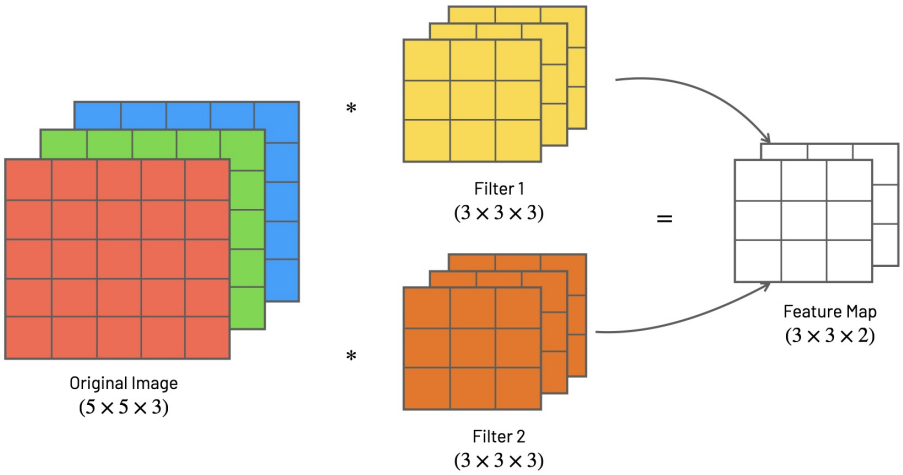

Figure 2: Example of applying 2 filters to an RGB image

8 Convolutional Layer

A convolutional layer in a neural network consists of:

1. Convolution operation with filters (analogous to linear combination in fully connected layers)
2. Bias addition
3. Activation function (typically ReLU)

Mathematically, for layer l:

$$Z^{[l]} = W^{[l]} * A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g(Z^{[l]})$$

where $*$ denotes the convolution operation, $W^{[l]}$ are the filters, $b^{[l]}$ is the bias, and g is the activation function.

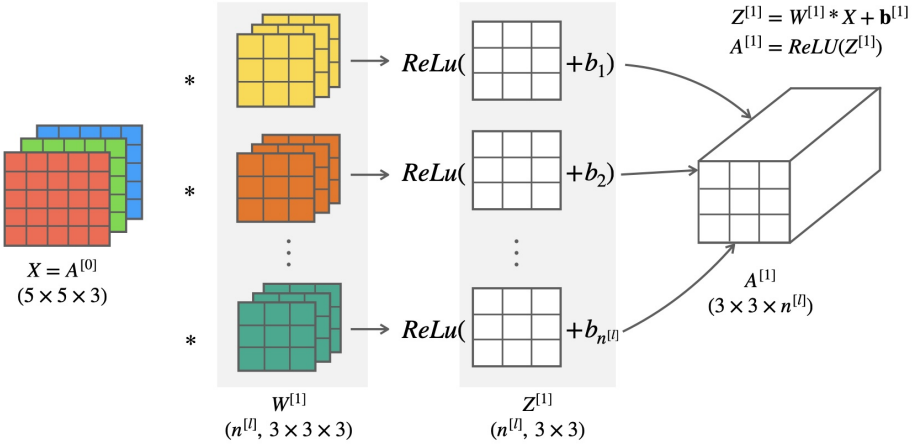

Figure 3: Example of a convolutional layer

9 Convolutional Neural Network Architecture

A typical CNN architecture for image classification might look like this:

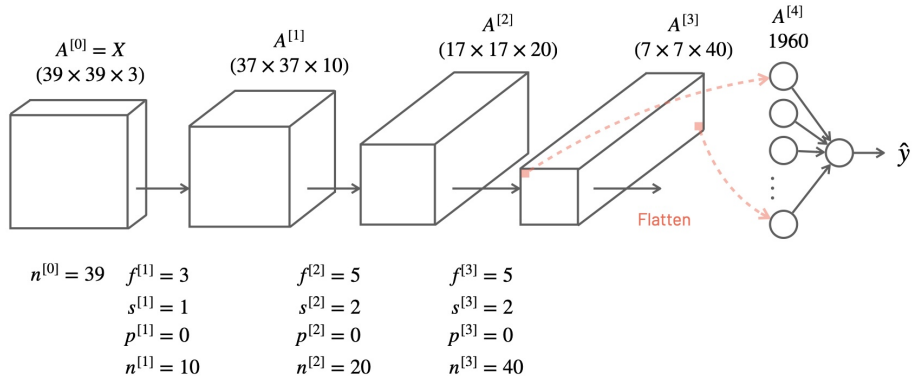

Figure 4: Example CNN Architecture

Key components:

- Input layer: The raw image (e.g., 39x39x3)
- Convolutional layers: Apply filters to extract features
- Flatten layer: Convert the final feature maps to a 1D vector
- Fully connected layers: Make the final classification decision

Each convolutional layer is characterized by:

- $f^{[l]}$: Size of filters
- $s^{[l]}$: Stride size
- $p^{[l]}$: Padding size
- $n^{[l]}$: Number of filters

10 Parameter Efficiency of CNNs

One of the key advantages of CNNs over fully connected networks is their parameter efficiency. Let's consider a convolutional layer with 10 filters, each of size 3x3x3.

Number of parameters:

- Weights: $10 \times (3 \times 3 \times 3) = 270$
- Biases: 10
- Total: 280 parameters

Compare this to the billions of parameters we had with a fully connected layer processing the same image. Moreover, this number of parameters is independent of the input image size, allowing CNNs to process large images efficiently.

11 Conclusion

Convolutional Neural Networks represent a powerful and efficient approach to processing image data. By leveraging the properties of convolutions, CNNs can automatically learn hierarchical features from images while maintaining a manageable number of parameters. This makes them particularly well-suited for a wide range of computer vision tasks, from image classification to object detection and segmentation.

In the next lecture, we will explore some famous CNN architectures that have pushed the boundaries of computer vision, including LeNet-5, AlexNet, VGG, ResNet, and Inception.

Exercises

1. What is the main advantage of using Convolutional Neural Networks over traditional Multi-Layer Perceptrons for image processing?

   (a) CNNs can process larger images
   (b) CNNs are faster to train
   (c) CNNs use fewer parameters, reducing overfitting and computational resources
   (d) CNNs can only be used for black and white images

2. Given an input image of size 28x28 and a convolutional filter of size 3x3 with stride 1 and no padding, what will be the size of the output feature map?

   (a) 28x28
   (b) 26x26
   (c) 30x30
   (d) 25x25

3. In a convolutional layer with 16 filters, each of size 5x5x3, how many total parameters are there (including biases)?

   (a) 1200
   (b) 1201
   (c) 1216
   (d) 1225

4. What is the purpose of padding in convolutional operations?

   (a) To increase the number of filters
   (b) To reduce the size of the output feature map
   (c) To preserve the input size and give equal importance to edge pixels
   (d) To increase the stride of the convolution

5. Given the following 5x5 input image and 3x3 filter, what is the resulting 3x3 feature map after convolution (assuming stride 1 and no padding)?

   Input Image:

$$\begin{bmatrix} 1 & 2 & 0 & 1 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 2 & 0 & 0 & 2 & 1 \\ 1 & 2 & 1 & 0 & 2 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

   Filter:

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

a) $\begin{bmatrix} 1 & -2 & 1 \\ 2 & -2 & 3 \\ 0 & -1 & 1 \end{bmatrix}$

b) $\begin{bmatrix} 2 & -1 & 2 \\ 3 & -3 & 4 \\ 1 & -2 & 2 \end{bmatrix}$

c) $\begin{bmatrix} 3 & -2 & 3 \\ 4 & -4 & 5 \\ 2 & -3 & 3 \end{bmatrix}$

d) $\begin{bmatrix} 4 & -3 & 4 \\ 5 & -5 & 6 \\ 3 & -4 & 4 \end{bmatrix}$