Convolutional Neural Networks (CNN) - A Beginner's Guide

1 Introduction

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing structured grid-like data such as images. They consist of convolutional layers, pooling layers, and fully connected layers.

2 Architecture of CNN

A CNN typically consists of the following layers:

2.1 Convolutional Layer

- Applies filters to extract features from an image. - A filter (also called a kernel) slides over the image, performing element-wise multiplication and summation. - Mathematically, convolution operation is:

$$(I * K)(x, y) = \sum_{i=0}^{m} \sum_{j=0}^{n} I(x+i, y+j)K(i, j) \tag{1}$$

where I is the input image and K is the filter/kernel.

2.2 Activation Function

- Introduces non-linearity, enabling the CNN to learn complex patterns. - Common activation functions: - **ReLU (Rectified Linear Unit)**: $f(x) = \max(0, x)$ - **Sigmoid**: $\sigma(x) = \frac{1}{1+e^{-x}}$ - **Tanh**: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

2.3 Pooling Layer

- Reduces the spatial dimensions of feature maps. - Common pooling techniques: - **Max Pooling**: Takes the maximum value in a window. - **Average Pooling**: Takes the average value in a window.

2.4 Fully Connected Layer

- Flattens the feature maps and passes them to a fully connected network. - The output is passed through a softmax function to obtain class probabilities:

$$P(y_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2}$$

where z are the logits from the last fully connected layer.

3 Implementation of CNN in Python

Below is a simple implementation of a CNN from scratch.

3.1 Convolutional Layer

```
import numpy as np
class ConvLayer:
    def __init__(self, num_filters, filter_size):
        self.num_filters = num_filters
        self.filter_size = filter_size
        self.filters = np.random.randn(num_filters, filter_size, filter_size)
    def iterate_regions(self, image):
        h, w = image.shape
        for i in range(h - self.filter_size + 1):
            for j in range(w - self.filter_size + 1):
                region = image[i:i+self.filter_size, j:j+self.filter_size]
                yield i, j, region
    def forward(self, input):
        self.last_input = input
        h, w = input.shape
        output = np.zeros((h - self.filter_size + 1, w - self.filter_size + 1, self.num_filt
        for i, j, region in self.iterate_regions(input):
            output[i, j] = np.sum(region * self.filters, axis=(1,2))
        return output
```

3.2 Pooling Layer

```
class MaxPoolLayer:
    def __init__(self, pool_size):
        self.pool_size = pool_size
    def iterate_regions(self, input):
        h, w, num_filters = input.shape
        for i in range(0, h, self.pool_size):
            for j in range(0, w, self.pool_size):
                region = input[i:i+self.pool_size, j:j+self.pool_size]
```

```
            yield i, j, region
    def forward(self, input):
        self.last_input = input
        h, w, num_filters = input.shape
        output = np.zeros((h // self.pool_size, w // self.pool_size, num_filters))
        for i, j, region in self.iterate_regions(input):
            output[i//self.pool_size, j//self.pool_size] = np.amax(region, axis=(0, 1))
        return output
```

4 Key Takeaways

- CNNs use convolutional layers to extract spatial features. - Pooling layers help reduce computation by downsampling. - Fully connected layers map extracted features to output classes. - Softmax is used in the final layer for classification. - Training involves backpropagation and optimization using gradient descent.

5 Important Equations

1. **Convolution Operation:**

$$(I * K)(x, y) = \sum_{i=0}^{m} \sum_{j=0}^{n} I(x+i, y+j) K(i, j) \tag{3}$$

2. **Activation Function (ReLU):**

$$f(x) = \max(0, x) \tag{4}$$

3. **Softmax Function:**

$$P(y_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{5}$$

4. **Cross-Entropy Loss:**

$$L = -\sum_i y_i \log(\hat{y}_i) \tag{6}$$