Mathematical Overview of My Convolutional Neural Network (CNN)

Roshan Virdee

July 2025

1 Introduction

This document will be an overview of the CNN that I created for my MNIST-Image Classifier. The CNN contains three main sections, the convolutional layer, the max-pooling layer, and the fully connected network layer. In the program, only one of each layer is used for forward/back propagation.

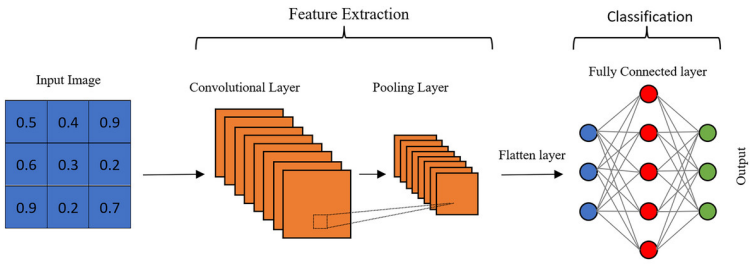

Figure 1: CNN Architecture

Figure 1 shows the general architecture of how CNN forward propagation works. A filter is applied to the image during the convolutional layer, for feature extraction. The feature maps are then reduced in size by applying the max-pooling filter. The reduced feature maps are then flattened and processed through the fully connected network. Then backpropagation will take place performing gradient descent to improve the network to increase the accuracy when it comes to image classification.

2 Forward Propagation

2.1 Convolutional Layer

In the convolutional layer, a single 6x6 filter is applied to the 28x28 MNIST image with a stride of 1.

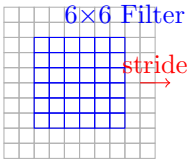

Figure 2: 6x6 Filter Sliding over 28x28 Image(Partial View)

$$x_{pool} = Cross\_Correlate(x_{conv}, f) \tag{1}$$

When the filter slides across the image, it will perform element-wise multiplication from the filter and the 6x6 patch, which is known as cross-correlation. The multiplied values are then summed and the summed number becomes one pixel in the output feature map. After applying the filter to the image, it will produce a 23x23 feature map.

2.2 Max-Pooling Layer

In the max-pooling layer, a 2x2 pooling filter is applied to the 23x23 feature map, with a stride of 2. This is performed to reduce the spatial dimensions of the feature map.

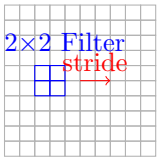

Figure 3: 2x2 Filter Sliding Over 23×23 Image (Partial View)

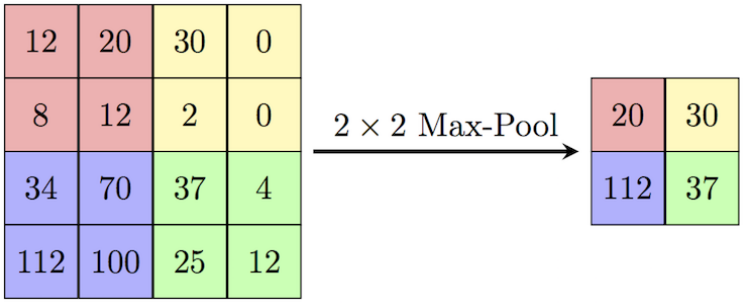

Figure 4: Max-Pooling Sample

Figure 4 shows how a 2x2 filter in max-pooling is used to decrease the spatial dimensions of an input. This will then take the 23x23 feature map to become an 11x11 feature map.

2.3 Fully Connected Layer

Once the pooling has been applied to the feature maps, they are then flattened. In our case, we only use one filter map so that the flattened output is a 121 value array. The input values are then fed into a fully connected network to go from the 121 values to 10. A one-hot prediction is then used to find which number has the highest probability, for the image.

To correctly perform the forward propagation for the fully connected network we apply a linear transformation. Once this has been calculated, we then apply a softmax function (with a shift for exponential stability) as the activation function. The cross-entropy loss function is then calculated to compare the predictions to the targets.

$$z = Wx + b \tag{2}$$

$$\hat{y} = \frac{e^{z_i - max_j z_j}}{\sum_{k=1}^{N} e^{z_k - max_j z_j}} \tag{3}$$

$$L = -\sum_{i=1}^{C} y_i log(\hat{y}_i), \qquad where \;\; \hat{y}_i = min(max(\hat{y}_i, \epsilon), 1 - \epsilon) \tag{4}$$

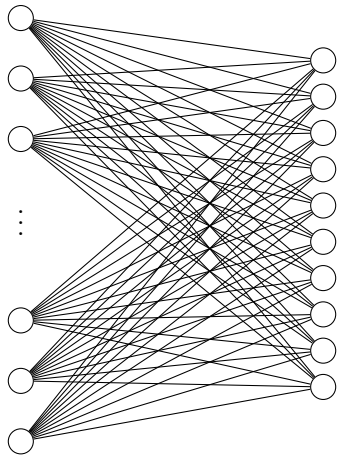

Figure 5: Partial visualization of the fully connected layer from 121 input features to 10 output classes.

The activation function is necessary to introduce nonlinearity, enabling the network to learn complex patterns in the data. The loss function shows the single sample cross-entropy loss function as we do not need to implement the multi-sample one for our program.

The array of 10 values is then turned into one hot encoded array, where the number with the highest probability has the value 1 and the rest 0. The index with value 1 then corresponds to the predicted number of the image.

$$\begin{bmatrix} 0.01 \\ 0.03 \\ 0.02 \\ 0.10 \\ 0.05 \\ 0.05 \\ 0.01 \\ 0.70 \\ 0.01 \\ 0.02 \end{bmatrix} \xrightarrow{\text{one-hot}} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{prediction}} 7$$

3 Backpropagation

Backpropagation is explained in reverse order of forward propagation, matching the way it is executed in the program, which makes it more intuitive to understand.

3.1 Fully Connected

To perform gradient descent, we will use the chain rule to calculate the derivatives needed. Gradient descent will help the network improve its accuracy by calculating the difference of the loss function with respect to the weights and biases. Once the derivatives have been calculated, we will be able to calculate the new weights and biases by applying the learning rate to the calculated derivatives.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w} \tag{5}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b} \tag{6}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial x} \tag{7}$$

3.1.1 Separate Derivatives

The separate derivatives used in the chain rule have been calculated to make it easier to follow.

$$\frac{\partial L}{\partial \hat{y}} = \sum_{i=1}^{C} -\frac{y_i}{\hat{y}_i} \tag{8}$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}_i(\delta_{i,k} - \hat{y}_k), \qquad \text{where } \delta_{i_k} = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases}$$

$$\frac{\partial z}{\partial w} = x^T \tag{9}$$

$$\frac{\partial z}{\partial b} = 1 \tag{10}$$

$$\frac{\partial z}{\partial x_{full}} = w^T \tag{11}$$

3.1.2 Calculating $\frac{\partial L}{\partial z}$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \tag{12}$$

$$= \sum_{i=1}^{C} -\frac{y_i}{\hat{y}_i} \cdot \hat{y}_i(\delta_{i,k} - \hat{y}_k) \tag{13}$$

$$= -\sum_{i=1}^{C} y_i \cdot (\delta_{i_k} - \hat{y}_k) \tag{14}$$

$$= -\sum_{i=1}^{C} y_i \delta_{i,k} + \sum_{i=1}^{C} y_i \hat{y}_k \tag{15}$$

$$= -y_k + \hat{y}_k \text{(Notes show simplification)} \tag{16}$$

$$= \hat{y}_k - y_k \tag{17}$$

Notes:

$$\sum_{i=1}^{C} y_i \delta_{i,k} = y_k \tag{18}$$

Since y is one hot, the summation shown in (18) will only equal the position $i = k$ as the rest of the positions will have a value of 0.

$$\sum_{i=1}^{C} y_i \hat{y}_k = \hat{y}_k \sum_{i=1}^{C} y_i = \hat{y}_k \tag{19}$$

The summation shown in (19) $\sum_{i=1}^{C} y_i = 1$ as y is one hot, it will have a single value with 1 and the rest 0.

3.1.3 Calculating $\frac{\partial L}{\partial w}$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w} \tag{20}$$

$$= (\hat{y}_k - y_k) \cdot x^T \tag{21}$$

3.1.4 Calculating $\frac{\partial L}{\partial b}$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial b} \tag{22}$$

$$= \frac{\partial L}{\partial z} \cdot 1 \tag{23}$$

$$= \hat{y}_k - y_k \tag{24}$$

3.2 Calculating $\frac{\partial L}{\partial x}$

This will not be used in gradient descent but will be passed into max-pooling backpropagation.

$$\frac{\partial L}{\partial x_{full}} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial x_{full}} \tag{25}$$

$$= w^T \cdot (\hat{y}_k - y_k) \tag{26}$$

3.2.1 Gradient Descent

Now that $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$ have been calculating the new weights and biases can be calculated.

$$w_{new} = w_{old} - \gamma \frac{\partial L}{\partial w} \tag{27}$$

$$b_{new} = b_{old} - \gamma \frac{\partial L}{\partial b} \tag{28}$$

3.3 Max-Pooling Layer

The max-pooling layer will perform gradient routing before passing to the convolutional layer. It does this by multiplying a binary mask by each pixel scalar of $\frac{\partial L}{\partial x_{full}}$. This ensures that the gradient is correctly backpropagated only to the neuron that was selected during the max operation in the forward pass.

3.3.1 Example

$$Channel\_Number = 1 \tag{29}$$

$$Patch\_Size = (2\text{x}2) \tag{30}$$

$$Pixel(i, j) = (0, 0) \tag{31}$$

$$\frac{\partial L}{\partial x_{full}}^{0,0} = 0.8 \tag{32}$$

$$Patch = \begin{bmatrix} 1 & 5 \\ 2 & 3 \end{bmatrix}$$

$$Mask = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$\frac{\partial L}{\partial x_{pool}} = 0.8 \cdot \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0.8 \\ 0 & 0 \end{bmatrix}$$

This is repeated throughout the rest of the input to ensure that only the maximal features selected in the forward pass have the gradients passed.

3.4 Convolutional Layer

To perform gradient descent, we will need to calculate the derivative of the loss function with respect to the filters and input. As the forward propagation of the convolutional layer applies cross-correlation, the backpropagation will also apply cross-correlation and full cross-correlation (convolution) to obtain the derivatives.

$$\frac{\partial L}{\partial f} = \text{Cross\_Correlate}(\frac{\partial x_{pool}}{\partial f}, \frac{\partial L}{\partial x_{pool}}) \tag{33}$$

$$\frac{\partial L}{\partial x_{conv}} = \text{Full\_Cross\_Correlate}(\frac{\partial L}{\partial x_{pool}}, \frac{\partial x_{pool}}{\partial x_{conv}}) \tag{34}$$

3.4.1 Seperate Derivatives

$$\frac{\partial x_{pool}}{\partial f} = x_{conv} \tag{35}$$

$$\frac{\partial x_{pool}}{\partial x} = \text{f} \tag{36}$$

3.4.2 Calculating $\frac{\partial L}{\partial f}$

$$\frac{\partial L}{\partial f} = CrossCorrelate(x_{conv}, \frac{\partial L}{\partial x_{pool}}) \tag{37}$$

3.4.3 Calculating $\frac{\partial L}{\partial x_{conv}}$

This derivative is not needed in the training/use of the CNN, but it is included, so all derivatives are calculated.

$$\frac{\partial L}{\partial x_{conv}} = \text{Full\_Cross\_Correlation}(\frac{\partial L}{\partial x_{pool}}, f) \tag{38}$$

3.4.4 Gradient Descent

$$f_{new} = f_{old} - \gamma \frac{\partial L}{\partial f} \tag{39}$$