Backpropagation in CNNs (Convolutional Neural Networks)

Backpropagation is the key algorithm used for training neural networks, including CNNs. It involves the process of adjusting the weights and biases in the network by propagating the error (or loss) backward from the output layer to the input layer, using the chain rule of calculus to update the parameters in a way that minimizes the loss function. Here's how backpropagation works in CNNs:

Steps of Backpropagation in CNN:

1. **Forward Pass**:
    a. The input image (or feature map) is passed through the network, where convolutional layers, activation functions, pooling layers, and fully connected layers (if applicable) perform operations to produce the output (predicted result).
    b. The final output is compared with the true label (in supervised learning) to compute the **loss** using a loss function (e.g., cross-entropy for classification tasks, mean squared error for regression tasks).
2. **Loss Calculation**:
    a. The **loss** (or error) is calculated as the difference between the predicted output and the true label. This loss measures how well or poorly the network's prediction matches the actual target.
3. **Backpropagation (Error Propagation)**:
    a. The error is propagated backward through the network to update the weights and biases in each layer.
    b. **Gradient Calculation**: For each layer in the CNN (starting from the output layer and moving backward), the gradient of the loss with respect to each weight and bias is calculated using the chain rule. This involves:
        i. **Convolutional Layer Gradients**: The gradient of the loss with respect to the weights in the convolutional layer is computed by considering the effect of the weights on the output feature map and how they affect the final loss.
        ii. **Activation Function Gradients**: The gradient of the loss with respect to the activations of each neuron is calculated by applying the derivative of the activation function (e.g., ReLU, sigmoid, tanh).

iii. **Pooling Layer Gradients**: The gradients for pooling layers (such as max pooling or average pooling) are computed by tracing back how the pooling operation affects the activation.

4. **Weight Update**:
   a. Once the gradients are computed for each layer, the weights and biases of the convolutional layers are updated using an optimization algorithm, most commonly **Stochastic Gradient Descent (SGD)** or its variants (e.g., Adam, RMSProp).
   b. The update rule for weights typically follows this formula:

$$w = w - \eta \cdot \frac{\partial \text{Loss}}{\partial w}$$

   where:

   - w is the weight,
   - η is the learning rate
   - $\frac{\partial \text{Loss}}{\partial w}$ is the gradient of the loss with respect to the weight.
   -

5. **Repeat**:
   a. The forward pass and backpropagation process is repeated over multiple iterations (or epochs), with the weights being adjusted after each batch or dataset pass, until the network converges and the loss is minimized.

Backpropagation Through Layers in CNNs:

- **Convolutional Layers**: The gradient of the loss with respect to the convolutional filters (kernels) is calculated by applying the chain rule to the convolution operation. This involves computing the partial derivative of the loss with respect to each filter's weights.
- **Activation Functions**: The gradients of the activation function (such as ReLU, sigmoid, or tanh) are computed and used to adjust the weights accordingly.
- **Pooling Layers**: In max pooling layers, the gradient is backpropagated only to the neuron that was selected in the pooling operation. For average pooling, the gradient is distributed equally to all neurons involved in the pooling.