Convolutional Neural Networks
Earliest “deep” architecture
Neocognitron

(Fukushima 1974-1982)
**Goal:** Given an image, we want to identify what class that image belongs to.
Pipeline:

Input

Convolutional Neural Network (CNN)

Output

A Monitor
Convolutional Neural Nets (CNNs) in a nutshell:

- A typical CNN takes a raw RGB image as an input.
- It then applies a series of non-linear operations on top of each other.
- These include convolution, sigmoid, matrix multiplication, and pooling (subsampling) operations.
- The output of a CNN is a highly non-linear function of the raw RGB image pixels.
How the key operations are encoded in standard CNNs:

• Convolutional Layers: 2D Convolution
• Fully Connected Layers: Matrix Multiplication
• Sigmoid Layers: Sigmoid function
• Pooling Layers: Subsampling
Convolutional Neural Networks
2D convolution:

\[ h = f \otimes g \]

- \( f \) - the values in a 2D grid that we want to convolve
- \( g \) - convolutional weights of size MxN

\[ h_{ij} = \sum_{m=0}^{M} \sum_{n=0}^{N} f(i - m, j - n) g(m,n) \]

A sliding window operation across the entire grid \( f \).
\[ f = \begin{bmatrix} \begin{array}{ccc} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{array} \end{bmatrix} \quad g_1 = \begin{bmatrix} \begin{array}{ccc} 0.107 & 0.113 & 0.107 \\ 0.113 & 0.119 & 0.113 \\ 0.107 & 0.113 & 0.107 \end{array} \end{bmatrix} \quad g_2 = \begin{bmatrix} \begin{array}{ccc} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{array} \end{bmatrix} \]

\[ f \otimes g_1 \]

\[ f \otimes g_2 \]

\[ f \otimes g_3 \]

Unchanged Image

Blurred Image

Vertical Edges
CNNs aim to learn convolutional weights directly from the data.
Early layers learn to detect low level structures such as oriented edges, colors and corners.
Convolutional Neural Network (CNN)

Deep layers learn to detect high-level object structures and their parts.
A Closer Look inside the Convolutional Layer

A Chair Filter

A Person Filter

A Table Filter

A Cupboard Filter

Input Image
Convolutional Neural Networks
Fully Connected Layers:

- \( z_i^{(l)} \) - the output unit \( i \) in layer \( l \)
- \( W_{ij}^{(l)} \) - the weight connection between unit \( j \) in layer \( l \) and unit \( i \) in layer \( l + 1 \)

\[
\begin{align*}
  z_1^{(1)} &= W_{11}^{(0)} x_1 + W_{12}^{(0)} x_2 + W_{13}^{(0)} x_3 \\
  z_2^{(1)} &= W_{21}^{(0)} x_1 + W_{22}^{(0)} x_2 + W_{23}^{(0)} x_3 \\
  z_3^{(1)} &= W_{31}^{(0)} x_1 + W_{32}^{(0)} x_2 + W_{33}^{(0)} x_3
\end{align*}
\]
**Fully Connected Layers:**

Hidden Layer Connections

Input Data

Layer 0

Layer 1

\[ z_1^{(1)} \]

\[ z_2^{(1)} \]

\[ z_3^{(1)} \]

\[ z_i^{(l)} \] - the output unit \( i \) in layer \( l \)

\[ W_{ij}^{(l)} \] - the weight connection between unit \( j \) in layer \( l \) and unit \( i \) in layer \( l + 1 \)

\[ z^{(1)} = W^{(0)}x \]

\[ \backslash \]

matrix multiplication
Convolutional Neural Networks
https://medium.com/@yu4u/why-mobilenet-and-its-variants-e-g-shufflenet-are-fast-1c7048b9618d
Convolutional Neural Networks
Hierarchical Pooling
Max Pooling Layer:

- Sliding window is applied on a grid of values.
- The maximum is computed using the values in the current window.

```
1 2 3
4 5 6
7 8 9
```
Max Pooling Layer:

• Sliding window is applied on a grid of values.
• The maximum is computed using the values in the current window.

```
1 2 3
4 5 6
7 8 9
```

\[ \rightarrow \begin{array}{c}
5
\end{array} \]
Max Pooling Layer:

- Sliding window is applied on a grid of values.
- The maximum is computed using the values in the current window.
Max Pooling Layer:

- Sliding window is applied on a grid of values.
- The maximum is computed using the values in the current window.

```
  1 2 3
 4 5 6
 7 8 9
```

```
  5 6
 8 9
```
Sigmoid Layer:

- Applies a sigmoid function on an input

\[ a^{(l)} = f(z^{(l)}) = \frac{1}{1 + \exp(-z^{(l)})} \]
Convolutional Networks

Let us now consider a CNN with a specific architecture:

- 2 convolutional layers.
- 2 pooling layers.
- 2 fully connected layers.
- 3 sigmoid layers.
Notation:

- convolutional layer output
- pooling layer
- fully connected layer output
- sigmoid function $f$
- softmax function

Forward Pass:

$x$
Notation:

- convolutional layer output
- pooling layer
- fully connected layer output
- sigmoid function $f$
- softmax function

Forward Pass:

$\mathbf{x} \rightarrow z^{(1)}$
Notation:
- convolutional layer output
- pooling layer
- fully connected layer output
- sigmoid function $f$
- softmax function

Forward Pass:
$x \rightarrow z^{(1)} \rightarrow \alpha^{(1)}$
Notation:
- convolutional layer output
- pooling layer
- fully connected layer output
- sigmoid function \( f \)
- softmax function

Forward Pass:
\( x \)
Notation:

- convolutional layer output
- fully connected layer output
- pooling layer
- sigmoid function $f$
- softmax function

Forward Pass:

$x$ $\rightarrow$ $z^{(1)}$ $\rightarrow$ $\alpha^{(1)}$ $\rightarrow$ $z^{(2)}$
Notation:

- convolutional layer output
- pooling layer
- fully connected layer output
- sigmoid function $f$
- softmax function

Forward Pass:

$x \rightarrow z^{(1)} \rightarrow a^{(1)} \rightarrow z^{(2)} \rightarrow a^{(2)}$
Notation:

- convolutional layer output
- pooling layer
- fully connected layer output
- sigmoid function \( f \)
- softmax function

Forward Pass:

\[
\mathbf{x} \rightarrow \mathbf{z}^{(1)} \rightarrow \mathbf{a}^{(1)} \rightarrow \mathbf{z}^{(2)} \rightarrow \mathbf{a}^{(2)}
\]
Notation:
- □ - convolutional layer output
- ▼ - fully connected layer output
- • - pooling layer
- ✓ - sigmoid function $f$
- ✓ - softmax function

Forward Pass:
\[ x \rightarrow z^{(1)} \rightarrow a^{(1)} \rightarrow z^{(2)} \rightarrow a^{(2)} \rightarrow z^{(3)} \]
Notation:

- □ - convolutional layer output
- - pooling layer
- - fully connected layer output
- - sigmoid function $f$
- - softmax function

Forward Pass:

$\mathbf{x}$ $\rightarrow$ $z^{(1)}$ $\rightarrow$ $a^{(1)}$ $\rightarrow$ $z^{(2)}$ $\rightarrow$ $a^{(2)}$ $\rightarrow$ $z^{(3)}$ $\rightarrow$ $\mathbf{a}^{(3)}$
Convolutional Networks

Notation:
- convolutional layer output
- fully connected layer output
- pooling layer
- sigmoid function $f$
- softmax function

Forward Pass:

$\mathbf{x}$ $\rightarrow$ $z^{(1)}$ $\rightarrow$ $\mathbf{a}^{(1)}$ $\rightarrow$ $z^{(2)}$ $\rightarrow$ $\mathbf{a}^{(2)}$ $\rightarrow$ $z^{(3)}$ $\rightarrow$ $\mathbf{a}^{(3)}$ $\rightarrow$ $z^{(4)}$
**Notation:**

- □ - convolutional layer output
- | - fully connected layer output
- | - pooling layer
- | - sigmoid function \( f \)
- | - softmax function

**Forward Pass:**

\[ x \rightarrow z^{(1)} \rightarrow a^{(1)} \rightarrow z^{(2)} \rightarrow a^{(2)} \rightarrow z^{(3)} \rightarrow a^{(3)} \rightarrow z^{(4)} \rightarrow \hat{y} \]
Notation:
- convolutional layer output
- fully connected layer output
- pooling layer
- sigmoid function \( f \)
- softmax function

Forward Pass:

\[ x \rightarrow z^{(1)} \rightarrow a^{(1)} \rightarrow z^{(2)} \rightarrow a^{(2)} \rightarrow z^{(3)} \rightarrow a^{(3)} \rightarrow z^{(4)} \hat{y} \]

Final Predictions
Notation:
- convolutional layer output
- fully connected layer output
- pooling layer
- sigmoid function $f$
- softmax function

Forward Pass:
Convolutional layer parameters in layers 1 and 2
Notation:

- convolutional layer output
- fully connected layer output
- pooling layer
- sigmoid function $f$
- softmax function

Forward Pass:

$g^{(1)} \rightarrow z^{(1)} \rightarrow a^{(1)} \rightarrow g^{(2)} \rightarrow z^{(2)} \rightarrow a^{(2)} \rightarrow z^{(3)} \rightarrow a^{(3)} \rightarrow z^{(4)} \rightarrow \hat{y}$

Fully connected layer parameters in the fully connected layers 1 and 2
Notation:
- □ - convolutional layer output
- - fully connected layer output
- - pooling layer
- - sigmoid function \( f \)
- - softmax function

Forward Pass:

\[ x \rightarrow g^{(1)} \rightarrow z^{(1)} \rightarrow a^{(1)} \rightarrow g^{(2)} \rightarrow z^{(2)} \rightarrow a^{(2)} \rightarrow W^{(1)} \rightarrow W^{(2)} \rightarrow z^{(4)} \hat{y} \]
Notation:
- convolutional layer output
- fully connected layer output
- pooling layer
- sigmoid function $f$
- softmax function

Forward Pass:

$$a^{(1)} = pool(f(g^{(1)} \ast x))$$
Notation:
- convolutional layer output
- fully connected layer output
- pooling layer
- sigmoid function $f$
- softmax function

Forward Pass:

1. $a^{(1)} = \text{pool}(f(g^{(1)} \ast x))$
2. $a^{(2)} = \text{pool}(f(g^{(2)} \ast a^{(1)}))$
**Notation:**
- convolutional layer output
- fully connected layer output
- pooling layer
- sigmoid function $f$
- softmax function

**Forward Pass:**

1. $a^{(1)} = \text{pool}(f(g^{(1)} \ast x))$
2. $a^{(2)} = \text{pool}(f(g^{(2)} \ast a^{(1)}))$
3. $a^{(3)} = f(W^{(1)}a^{(2)})$
Notation:
- convolutional layer output
- fully connected layer output
- pooling layer
- sigmoid function $f$
- softmax function

Forward Pass:

1. $a^{(1)} = \text{pool}(f(g^{(1)} \ast x))$
2. $a^{(2)} = \text{pool}(f(g^{(2)} \ast a^{(1)}))$
3. $a^{(3)} = f(W^{(1)} a^{(2)})$
4. $\hat{y} = \text{softmax}(W^{(2)} a^{(3)})$
Forward Pass:

Key Question: How to learn the parameters from the data?
Backpropagation

for

Convolutional Neural Networks
How to learn the parameters of a CNN?

- Assume that we are given a labeled training dataset
  \[ \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\} \]

- We want to adjust the parameters of a CNN such that CNN’s predictions would be as close to true labels as possible.

- This is difficult to do because the learning objective is highly non-linear.
Gradient descent:

- Iteratively minimizes the objective function.
- The function needs to be differentiable.

\[ \theta = \theta - \alpha \frac{\partial L(\theta)}{\partial \theta} \]
Gradient descent:

- Iteratively minimizes the objective function.
- The function needs to be differentiable.

\[ \theta = \theta - \alpha \frac{\partial L(\theta)}{\partial \theta} \]
Gradient descent:

- Iteratively minimizes the objective function.
- The function needs to be differentiable.

\[ \theta = \theta - \alpha \frac{\partial L(\theta)}{\partial \theta} \]
Gradient descent:
• Iteratively minimizes the objective function.
• The function needs to be differentiable.

\[
\theta = \theta - \alpha \frac{\partial L(\theta)}{\partial \theta}
\]
1. Compute the gradients of the overall loss w.r.t. to our predictions and propagate it back:

\[ \frac{\partial L}{\partial \hat{y}} \]
2. Compute the gradients of the overall loss and propagate it back:

\[ \frac{\partial L}{\partial z^{(4)}} \]
3. Compute the gradients to adjust the weights:
\[
\frac{\partial L}{\partial W^{(2)}}
\]
4. Backpropagate the gradients to previous layers: \( \frac{\partial L}{\partial z^{(3)}} \)
5. Compute the gradients to adjust the weights:

\[
\frac{\partial L}{\partial W^{(1)}}
\]
6. Backpropagate the gradients to previous layers:

\[ \frac{\partial L}{\partial z^{(2)}} \]
7. Compute the gradients to adjust the weights:

\[ \frac{\partial L}{\partial g^{(2)}} \]
8. Backpropagate the gradients to previous layers:
9. Compute the gradients to adjust the weights:

\[
\frac{\partial L}{\partial g^{(1)}}
\]
An Example of Backpropagation Convolutional Neural Networks
Assume that we have K=5 object classes:

- **Class 1:** Penguin
- **Class 2:** Building
- **Class 3:** Chair
- **Class 4:** Person
- **Class 5:** Bird

\[
\hat{y} = \begin{bmatrix} 0.5 & 0 & 0.1 & 0.2 & 0.1 \end{bmatrix}
\]

\[
y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}
\]
where

\[ L = - \sum_{i=1}^{K} y_i \log (\hat{y}_i) \]

where \( \hat{y}_i = \frac{\exp(z_i^{(4)})}{\sum_{j=1}^{K} \exp(z_j^{(4)})} \)
\[ L = - \sum_{i=1}^{K} y_i \log (\hat{y}_i) \text{ where } \hat{y}_i = \frac{\exp(z_i^{(4)})}{\sum_{j=1}^{K} \exp(z_j^{(4)})} \]

\[
\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(4)}}
\]
\[ L = -\sum_{i=1}^{K} y_i \log (\hat{y}_i) \quad \text{where} \quad \hat{y}_i = \frac{\exp(z^{(4)}_i)}{\sum_{j=1}^{K} \exp(z^{(4)}_j)} \]

\[ \frac{\partial L}{\partial z^{(4)}_i} = \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z^{(4)}_i} \]

\[ \frac{\partial L}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i} \]
\[ L = - \sum_{i=1}^{K} y_i \log (\hat{y}_i) \quad \text{where} \quad \hat{y}_i = \frac{\exp (z_i^{(4)})}{\sum_{j=1}^{K} \exp (z_j^{(4)})} \]

\[
\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_i^{(4)}} \\
\frac{\partial L}{\partial \hat{y}_i} = -y_i/\hat{y}_i \\
\frac{\partial \hat{y}_i}{\partial z_i^{(4)}} = \begin{cases} 
\hat{y}_i(1 - \hat{y}_i), & \text{if } i = j \\
-\hat{y}_i \hat{y}_j, & \text{if } i \neq j 
\end{cases}
\]
\[
\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(4)}}
\]
\[
\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i^{(4)}} + \sum_{i \neq j} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_i^{(4)}}
\]
\[
\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(4)}}
\]

\[
= \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i^{(4)}} + \sum_{i \neq j} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_i^{(4)}}
\]

\[
= \hat{y}_i - y_i
\]
Assume that we have $K=5$ object classes:

- **Class 1:** Penguin
- **Class 2:** Building
- **Class 3:** Chair
- **Class 4:** Person
- **Class 5:** Bird

\[
\hat{y} = \begin{bmatrix} 0.5 & 0 & 0.1 & 0.2 & 0.1 \end{bmatrix},
\]
\[
y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}.
\]
Assume that we have $K=5$ object classes:

| Class 1: Penguin | Class 2: Building | Class 3: Chair | Class 4: Person | Class 5: Bird |

$\hat{y} = \begin{bmatrix} 0.5 & 0 & 0.1 & 0.2 & 0.1 \end{bmatrix}$

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

$$\frac{\partial L}{\partial z^{(4)}} = \begin{bmatrix} -0.5 & 0 & 0.1 & 0.2 & 0.1 \end{bmatrix}$$
Assume that we have $K=5$ object classes:

- **Class 1**: Penguin
- **Class 2**: Building
- **Class 3**: Chair
- **Class 4**: Person
- **Class 5**: Bird

Let's denote the true label as $y$ and the predicted label as $\hat{y}$.

- $y = [1, 0, 0, 0, 0]$ (indicating the true class is Penguin)
- $\hat{y} = [0.5, 0, 0.1, 0.2, 0.1]$ (indicating the predicted class probabilities)

The gradient of the loss with respect to the $4$th element of the predicted label is given by:

$$\frac{\partial L}{\partial z^{(4)}} = [-0.5, 0, 1, 0.2, 0.1]$$

Increasing the score corresponding to the true class decreases the loss.
Assume that we have $K=5$ object classes:

| Class 1: Penguin | Class 2: Building | Class 3: Chair | Class 4: Person | Class 5: Bird |

$$\begin{align*}
\hat{y} & = \begin{bmatrix} 0.5 & 0 & 0.1 & 0.2 & 0.1 \end{bmatrix} \\
y & = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}
\end{align*}$$

$$\frac{\partial L}{\partial z^{(4)}} = \begin{bmatrix} -0.5 & 0 & 0.1 & 0.2 & 0.1 \end{bmatrix}$$

Decreasing the score of other classes also decreases the loss.
Adjusting the weights:
Adjusting the weights:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial W^{(2)}}
\]
Adjusting the weights:

Need to compute the following gradient

\[
\frac{\partial L}{\partial W(2)} = \frac{\partial L}{\partial z(4)} \frac{\partial z(4)}{\partial W(2)}
\]
Adjusting the weights:

Need to compute the following gradient

\[
\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial W^{(2)}}
\]

\[
\frac{\partial L}{\partial z^{(4)}}
\]

was already computed in the previous step
Adjusting the weights:

Need to compute the following gradient

$$\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial W^{(2)}}$$
Adjusting the weights:

Need to compute the following gradient

\[ \frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial W^{(2)}} \]

\[ z^{(4)}_i = \sum_{k=1}^{N} W^{(2)}_{ik} f(z^{(3)}_k) \]

where \( f(z^{(3)}) = a^{(3)} \)
Adjusting the weights:

Need to compute the following gradient

\[ \frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial W^{(2)}} \]

\[ z_i^{(4)} = \sum_{k=1}^{N} W_{ik}^{(2)} f(z_k^{(3)}) \quad \text{where} \quad f(z^{(3)}) = a^{(3)} \]

\[ \frac{\partial z_i^{(4)}}{\partial W_{ij}^{(2)}} = f(z_j^{(3)}) \]
Adjusting the weights:

Need to compute the following gradient

\[
\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial W^{(2)}}
\]

Update rule:

\[
W_{ij}^{(2)} = W_{ij}^{(2)} - \alpha \frac{\partial L}{\partial W_{ij}^{(2)}}
\]
Backpropagating the gradients:
Backpropagating the gradients:

Need to compute the following gradient:

$$\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial f(z^{(3)})} \frac{\partial f(z^{(3)})}{\partial z^{(3)}}$$
Backpropagating the gradients:

Need to compute the following gradient:

$$\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial f(z^{(3)})} \frac{\partial f(z^{(3)})}{\partial z^{(3)}}$$
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial f(z^{(3)})} \frac{\partial f(z^{(3)})}{\partial z^{(3)}}
\]

\[
\frac{\partial L}{\partial z^{(4)}}
\]

was already computed in the previous step.
Backpropagating the gradients:

Need to compute the following gradient:

$$\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial f(z^{(3)})} \frac{\partial f(z^{(3)})}{\partial z^{(3)}}$$
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \cdot \frac{\partial z^{(4)}}{\partial f(z^{(3)})} \cdot \frac{\partial f(z^{(3)})}{\partial z^{(3)}}
\]

\[
z^{(4)}_i = \sum_{k=1}^{N} W^{(2)}_{ik} f(z^{(3)}_k) \quad \text{where} \quad f(z^{(3)}) = a^{(3)}
\]
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial f(z^{(3)})} \frac{\partial f(z^{(3)})}{\partial z^{(3)}}
\]

\[
z_i^{(4)} = \sum_{k=1}^{N} W_{ik}^{(2)} f(z_k^{(3)}) \quad \text{where} \quad f(z^{(3)}) = a^{(3)}
\]

\[
\frac{\partial z_i^{(4)}}{\partial f(z_i^{(3)})} = W_{ij}^{(2)}
\]
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial f(z^{(3)})} \frac{\partial f(z^{(3)})}{\partial z^{(3)}}
\]
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial f(z^{(3)})} \frac{\partial f(z^{(3)})}{\partial z^{(3)}}
\]

\[
f(z^{(3)}) = \frac{1}{1 + \exp(-z^{(3)})}
\]
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial f(z^{(3)})} \frac{\partial f(z^{(3)})}{\partial z^{(3)}}
\]

\[
f(z^{(3)}) = \frac{1}{1 + \exp(-z^{(3)})}
\]

\[
\frac{\partial f(z^{(3)})}{\partial z^{(3)}} = f(z^{(3)})(1 - f(z^{(3)}))
\]
Adjusting the weights:
Adjusting the weights:

Need to compute the following gradient
\[
\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial W^{(1)}}
\]

Update rule:
\[
W_{ij}^{(1)} = W_{ij}^{(1)} - \alpha \frac{\partial L}{\partial W_{ij}^{(1)}}
\]
Backpropagating the gradients:
Backpropagating the gradients:

Need to compute the following gradient:

$$\frac{\partial L}{\partial z^{(2)}} = \frac{\partial L}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial f(z^{(2)})} \frac{\partial f(z^{(2)})}{\partial z^{(2)}}$$
Adjusting the weights:
Adjusting the weights:

Need to compute the following gradient

\[
\frac{\partial L}{\partial g^{(2)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial g^{(2)}}
\]
Adjusting the weights:

Need to compute the following gradient

\[
\frac{\partial L}{\partial g^{(2)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial g^{(2)}}
\]
Adjusting the weights:

Need to compute the following gradient

$$\frac{\partial L}{\partial g^{(2)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial g^{(2)}}$$

was already computed in the previous step.
Adjusting the weights:

Need to compute the following gradient

\[
\frac{\partial L}{\partial g^{(2)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial g^{(2)}}
\]
Adjusting the weights:

Need to compute the following gradient

\[
\frac{\partial L}{\partial g^{(2)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial g^{(2)}}
\]

\[
z_{i,j}^{(2)} = \sum_{u=0}^{M} \sum_{v=0}^{N} g_{uv}^{(2)} a_{(i-u)(j-v)}^{(1)} \quad \text{where} \quad f(z^{(1)}) = a^{(1)}
\]
Adjusting the weights:

Need to compute the following gradient

\[ \frac{\partial L}{\partial g^{(2)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial g^{(2)}} \]

\[ z_{ij}^{(2)} = \sum_{u=0}^{M} \sum_{v=0}^{N} g_{uv}^{(2)} a_{(i-u)(j-v)}^{(1)} \text{ where } f(z^{(1)}) = a^{(1)} \]

\[ \frac{\partial z_{ij}^{(2)}}{\partial g_{mn}^{(2)}} = a_{(i-m)(j-n)}^{(1)} \]
Adjusting the weights:

Need to compute the following gradient

\[
\frac{\partial L}{\partial g^{(2)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial g^{(2)}}
\]

Update rule: 

\[
g_{mn}^{(2)} = g_{mn}^{(2)} - \alpha \frac{\partial L}{\partial g_{mn}^{(2)}}
\]
Backpropagating the gradients:
Backpropagating the gradients:

Need to compute the following gradient:

$$\frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial f(z^{(1)})} \frac{\partial f(z^{(1)})}{\partial z^{(1)}}$$
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(1)}} = \left[ \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial f(z^{(1)})} \frac{\partial f(z^{(1)})}{\partial z^{(1)}} \right]
\]
Backpropagating the gradients:

Need to compute the following gradient:

$$\frac{\partial L}{\partial z^{(1)}} = \left[ \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial f(z^{(1)})} \frac{\partial f(z^{(1)})}{\partial z^{(1)}} \right]$$

was already computed in the previous step
Backpropagating the gradients:

Need to compute the following gradient:

$$\frac{\partial L}{\partial z(1)} = \frac{\partial L}{\partial z(2)} \frac{\partial z(2)}{\partial f(z(1))} \frac{\partial f(z(1))}{\partial z(1)}$$
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial f(z^{(1)})} \frac{\partial f(z^{(1)})}{\partial z^{(1)}}
\]

\[
z_{i,j}^{(2)} = \sum_{u=0}^{M} \sum_{v=0}^{N} g_{uv}^{(2)} a_{(i-u)(j-v)}^{(1)} \quad \text{where} \quad f(z^{(1)}) = a^{(1)}
\]
Backpropagating the gradients:

Need to compute the following gradient:

\[ \frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial f(z^{(1)})} \frac{\partial f(z^{(1)})}{\partial z^{(1)}} \]

\[ z_{i,j}^{(2)} = \sum_{u=0}^{M} \sum_{v=0}^{N} g_{uv}^{(2)} a_{(i-u)(j-v)}^{(1)} \quad \text{where} \quad f(z^{(1)}) = a^{(1)} \]

\[ \frac{\partial z_{i,j}^{(2)}}{\partial a_{(i-m)(j-n)}^{(1)}} = g_{mn}^{(2)} \]
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial z^{(1)}} \frac{\partial f(z^{(1)})}{\partial z^{(1)}}
\]
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial f(z^{(1)})} \frac{\partial f(z^{(1)})}{\partial z^{(1)}}
\]

\[
f(z^{(1)}) = \frac{1}{1 + \exp(-z^{(1)})}
\]
Backpropagating the gradients:

Need to compute the following gradient:

\[
\frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial z^{(1)}} \frac{\partial f(z^{(1)})}{\partial z^{(1)}}
\]

\[
f(z^{(1)}) = \frac{1}{1 + \exp(-z^{(1)})}
\]

\[
\frac{\partial f(z^{(1)})}{\partial z^{(1)}} = f(z^{(1)})(1 - f(z^{(1)}))
\]
Adjusting the weights:
**Adjusting the weights:**

Need to compute the following gradient
\[
\frac{\partial L}{\partial g^{(1)}} = \frac{\partial L}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial g^{(1)}}
\]

Update rule:
\[
g^{(1)}_{mn} = g^{(1)}_{mn} - \alpha \frac{\partial L}{\partial g^{(1)}_{mn}}
\]
Visual illustration

Backpropagation

Convolutional Neural Networks
Fully Connected Layers:

Forward:

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]
Fully Connected Layers:

Forward:

\[ W^{(l)} f(z^{(l)}) z^{(l+1)} \]
Activation unit of interest

The weight that is used in conjunction with the activation unit of interest

Forward:

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]

The output
Fully Connected Layers:

Forward:

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]
Fully Connected Layers:

Forward:

\[ W^{(l)} f(z^{(l)}) \]
Fully Connected Layers:

Forward:

\[ W^{(l)} f(z^{(l)}) z^{(l+1)} \]
Backpropagation

Fully Connected Layers:

Forward:

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]
Fully Connected Layers:

Forward:

\[
W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}
\]
**Fully Connected Layers:**

**Forward:**

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]
Fully Connected Layers:

Forward:

\[ W^{(l)} \cdot f(z^{(l)}) = z^{(l+1)} \]
**Fully Connected Layers:**

**Forward:**

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]

**Backward:**

\[ \frac{\partial L}{\partial z^{(l+1)}} \]
**Fully Connected Layers:**

**Forward:**

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]

**Backward:**

A measure how much an activation unit contributed to the loss

\[ \frac{\partial L}{\partial z^{(l+1)}} = \frac{\partial L}{\partial f(z^{(l)})} \]
**Fully Connected Layers:**

**Forward:**

\[ W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)} \]

**Backward:**

A measure how much an activation unit contributed to the loss

\[ (W^{(l)})^T \quad \frac{\partial L}{\partial z^{(l+1)}} \quad \frac{\partial L}{\partial f(z^{(l)})} \]
Fully Connected Layers:

Forward:

\[
W^{(l)} f(z^{(l)}) = z^{(l+1)}
\]

Backward:

\[
(W^{(l)})^T \frac{\partial L}{\partial z^{(l+1)}} = \frac{\partial L}{\partial f(z^{(l)})}
\]
Fully Connected Layers:

Forward:

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]

Backward:

\[ (W^{(l)})^T \frac{\partial L}{\partial z^{(l+1)}} \frac{\partial L}{\partial f(z^{(l)})} \]
**Fully Connected Layers:**

**Forward:**

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]

**Backward:**

\[ (W^{(l)})^T \frac{\partial L}{\partial z^{(l+1)}} = \frac{\partial L}{\partial f(z^{(l)})} \]
**Fully Connected Layers:**

**Forward:**

\[
W^{(l)} f(z^{(l)}) = z^{(l+1)}
\]

**Backward:**

\[
(W^{(l)})^T \frac{\partial L}{\partial z^{(l+1)}} = \frac{\partial L}{\partial f(z^{(l)})}
\]
Fully Connected Layers:

Forward:

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]

Backward:

\[ (W^{(l)})^T \frac{\partial L}{\partial z^{(l+1)}} = \frac{\partial L}{\partial f(z^{(l)})} \]
Fully Connected Layers:

Forward:

\[ W^{(l)} f(z^{(l)}) = z^{(l+1)} \]

Backward:

\[ (W^{(l)})^T \frac{\partial L}{\partial z^{(l+1)}} = \frac{\partial L}{\partial f(z^{(l)})} \]
Summary for fully connected layers

Backpropagation
Convolutional Neural Networks
Summary:

1. Let $\frac{\partial L}{\partial z_i^{(n)}} = \hat{y}_i - y_i$, where $n$ denotes the number of layers in the network.
Summary:

1. Let \( \frac{\partial L}{\partial z_i^{(n)}} = \hat{y}_i - y_i \), where \( n \) denotes the number of layers in the network.

2. For each fully connected layer \( \ell \):
   
   • For each node \( i \) in layer \( \ell \) set:

   \[
   \frac{\partial L}{\partial z_i^{(\ell)}} = \left( \sum_{j=1}^{s^{\ell+1}} W_{ji}^{(\ell)} \frac{\partial L}{\partial z_j^{(\ell+1)}} \right) \frac{\partial f(z_i^{(\ell)})}{\partial z_i^{(\ell)}}
   \]
Summary:

1. Let \( \frac{\partial L}{\partial z_i^{(n)}} = \hat{y}_i - y_i \), where \( n \) denotes the number of layers in the network.

2. For each fully connected layer \( l \) :
   
   - For each node \( i \) in layer \( l \) set:
     \[
     \frac{\partial L}{\partial z_i^{(l)}} = \left( \sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \frac{\partial L}{\partial z_j^{(l+1)}} \right) \frac{\partial f(z_i^{(l)})}{\partial z_i^{(l)}}
     \]
   
   - Compute partial derivatives:
     \[
     \frac{\partial L}{\partial W_{ij}^{(l)}} = f(z_j^{(l)}) \frac{\partial L}{\partial z_i^{(l+1)}}
     \]
Summary:

1. Let $\frac{\partial L}{\partial z_i^{(n)}} = \hat{y}_i - y_i$, where n denotes the number of layers in the network.

2. For each fully connected layer $l$:
   
   - For each node $i$ in layer $l$ set:
     \[
     \frac{\partial L}{\partial z_i^{(l)}} = \left( \sum_{j=1}^{s_i^{l+1}} W_{ji}^{(l)} \frac{\partial L}{\partial z_j^{(l+1)}} \right) \frac{\partial f(z_i^{(l)})}{\partial z_i^{(l)}}
     \]
   
   - Compute partial derivatives: $\frac{\partial L}{\partial W_{ij}^{(l)}} = f(z_j^{(l)}) \frac{\partial L}{\partial z_i^{(l+1)}}$

   - Update the parameters: $W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial L}{\partial W_{ij}^{(l)}}$
Visual illustration

Backpropagation

Convolutional Neural Networks
Convolutional Layers:

Forward:

\[ a^{(l)} \otimes g^{(l)} = z^{(l+1)} \]
Convolutional Layers:

Forward:

\[ a^{(l)} \otimes g^{(l)} = z^{(l+1)} \]

Backward:

A measure how much an activation unit contributed to the loss

\[ \frac{\partial L}{\partial z^{(l+1)}} \quad \frac{\partial L}{\partial f(z^{(l)})} \]
Convolutional Layers:

**Forward:**

\[ a^{(l)} \otimes g^{(l)} = z^{(l+1)} \]

**Backward:**

\[ \text{sum} \left( g^{(l)} \circ \frac{\partial L}{\partial z^{(l+1)}} \right) = \frac{\partial L}{\partial f(z^{(l)})} \]
Summary:

1. Let \( \frac{\partial L}{\partial z^{(c)}} \), where \( c \) denotes the index of a first fully connected layer.

2. For each convolutional layer \( l \):
   - For each node \( i_{lj} \) in layer \( l \) set
     \[
     \frac{\partial L}{\partial z_{ij}^{(l)}} = \left( \sum_{m=0}^{M} \sum_{n=0}^{N} g_{mn}^{(l)} \frac{\partial L}{\partial z_{(i+m)(j+n)}^{(l+1)}} \right) \frac{\partial f(z_{ij}^{(l)})}{\partial z_{ij}^{(l)}}
     \]
Summary:

1. Let \( \frac{\partial L}{\partial z^{(c)}} \), where \( c \) denotes the index of a first fully connected layer.

2. For each convolutional layer \( l \):
   
   • For each node \( ij \) in layer \( l \) set
     \[
     \frac{\partial L}{\partial z_{ij}^{(l)}} = \left( \sum_{m=0}^{M} \sum_{n=0}^{N} g_{mn}^{(l)} \frac{\partial L}{\partial z_{(i+m)(j+n)}^{(l+1)}} \right) \frac{\partial f(z_{ij}^{(l)})}{\partial z_{ij}^{(l)}}
     \]
   
   • Compute partial derivatives:
     \[
     \frac{\partial L}{\partial g_{ij}^{(l)}} = \sum_{y=0}^{H} \sum_{x=0}^{W} \frac{\partial L}{\partial z_{yx}^{(l+1)}} f(z_{(y-i)(x-j)}^{(l)})
     \]
Summary:

1. Let $\frac{\partial L}{\partial z^{(c)}}$, where $c$ denotes the index of a first fully connected layer.

2. For each convolutional layer $l$:
   - For each node $ij$ in layer $l$ set
     
     $$
     \frac{\partial L}{\partial z_{ij}^{(l)}} = \left( \sum_{m=0}^{M} \sum_{n=0}^{N} g_{mn}^{(l)} \frac{\partial L}{\partial z^{(l+1)}_{(i+m)(j+n)}} \right) \frac{\partial f(z_{ij}^{(l)})}{\partial z_{ij}^{(l)}}
     $$
   
   - Compute partial derivatives:
     
     $$
     \frac{\partial L}{\partial g_{ij}^{(l)}} = \sum_{y=0}^{H} \sum_{x=0}^{W} \frac{\partial L}{\partial z_{yx}^{(l+1)}} f(z_{(y-i)(x-j)}^{(l)})
     $$
   
   - Update the parameters: $g_{ij}^{(l)} = g_{ij}^{(l)} - \alpha \frac{\partial L}{\partial g_{ij}^{(l)}}$
Gradient in pooling layers:

- There is no learning done in the pooling layers.
- The error that is backpropagated to the pooling layer, is sent back from the node where it came from.
Gradient in pooling layers:

- There is no learning done in the pooling layers
- The error that is backpropagated to the pooling layer, is sent back from to the node where it came from.
Gradient in pooling layers:

- There is no learning done in the pooling layers.
- The error that is backpropagated to the pooling layer, is sent back from to the node where it came from.
A Closer Look inside the Convolutional Layer

- A Chair Filter
- A Person Filter
- A Table Filter
- A Cupboard Filter

Input Image
A Closer Look inside the Convolutional Layer:
A Closer Look inside the Back Propagation Convolutional Layer:

\[
\frac{\partial L}{\partial z_{ij}^{(l)}} = \left( \sum_{m=0}^{M} \sum_{n=0}^{N} g_{mn}^{(l)} \frac{\partial L}{\partial z_{(i+m)(j+n)}^{(l+1)}} \right) \frac{\partial f(z_{ij}^{(l)})}{\partial z_{ij}^{(l)}}
\]
Adjusting the weights:
Training Batch

\[
\frac{\partial L}{\partial g_{ij}}^{(t)} = \sum_{y=0}^{H} \sum_{x=0}^{W} \frac{\partial L}{\partial z_{yx}^{(l+1)}} f(z_{(y-i)(x-j)}^{(l)})
\]

Average the Gradient Across the Batch

Parameter Update

\[
g^{(1)} = g^{(1)} - \frac{\partial L}{\partial g^{(1)}}
\]

new \( g^{(1)} \)
old \( g^{(1)} \)
\[
\frac{\partial L}{\partial g^{(1)}}
\]

(performed in a sliding window fashion)

- elementwise multiplication
Training Batch

\[
\frac{\partial L}{\partial g_{ij}^{(i)}} = \sum_{y=0}^{H} \sum_{x=0}^{W} \frac{\partial L}{\partial z_{yx}^{(i+1)}} f(z_{y-i}^{(i)}(x-j))
\]

\[
\text{sum (} x \text{)} = \sum \frac{\partial L}{\partial g_{ij}^{(i)}}
\]

(average the gradient across the batch)

\[
g^{(1)} = g^{(1)} - \frac{\partial L}{\partial g^{(1)}}
\]

(new \( g^{(1)} \) - old \( g^{(1)} \))

- elementwise multiplication
\[
\frac{\partial L}{\partial g_{ij}^{(l)}} = \sum_{y=0}^{H} \sum_{x=0}^{W} \frac{\partial L}{\partial z_{yx}^{(l+1)}} f'(z_{yx}^{(l+1)})
\]

- Training Batch

\[
\begin{align*}
\sum (4) &= \text{elementwise multiplication} \\
\sum (1) &= \text{elementwise multiplication} \\
\sum (0) &= \text{elementwise multiplication} \\
\sum (8) &= \text{elementwise multiplication} \\
\sum (1) &= \text{elementwise multiplication}
\end{align*}
\]

(Performed in a sliding window fashion)

\[
\frac{\partial L}{\partial g^{(1)}} = \text{Average the Gradient Across the Batch} \\
\]

Parameter Update

\[
\begin{align*}
g^{(1)} &= g^{(1)} - \frac{\partial L}{\partial g^{(1)}} \\
\text{new } g^{(1)} &= \text{old } g^{(1)} - \frac{\partial L}{\partial g^{(1)}}
\end{align*}
\]
Adjusting the weights:
\[
\frac{\partial L}{\partial g^{(i)}} = \sum_{y=0}^{H} \sum_{x=0}^{W} \frac{\partial L}{\partial z_{yx}^{(l+1)}} f(z_{(y-\Delta y) (x-\Delta x)}^{(l)})
\]

Average the Gradient Across the Batch

Parameter Update

\[
g^{(2)} = g^{(2)} - \frac{\partial L}{\partial g^{(2)}}
\]

(new \(g^{(2)}\) - old \(g^{(2)}\))

(performing in a sliding window fashion)

\(\odot\) - elementwise multiplication
Training Batch

\[
\frac{\partial L}{\partial g_{ij}^{(l)}} = \sum_{y=0}^{H} \sum_{x=0}^{W} \frac{\partial L}{\partial z_{yx}^{(l+1)}} f(z_{(y-i)(x-j)}^{(l)})
\]

Average the Gradient Across the Batch

Parameter Update

\[
g^{(2)}(\text{new}) = g^{(2)}(\text{old}) - \frac{\partial L}{\partial g^{(2)}}
\]

- elementwise multiplication
Training Batch

\[
\frac{\partial L}{\partial g_{ij}^{(l)}} = \sum_{y=0}^{H} \sum_{x=0}^{W} \frac{\partial L}{\partial z_{y,x}^{(l+1)}} f(z_{y,x}^{(l)}(x-j))
\]

- Elementwise multiplication

Average the Gradient Across the Batch

Parameter Update

\[
g^{(2)} = g^{(2)} - \frac{\partial L}{\partial g^{(2)}}
\]

new \( g^{(2)} \)  old \( g^{(2)} \)  \( \frac{\partial L}{\partial g^{(2)}} \)

(Performed in a sliding window fashion)