MATHEMATICS
FOR MACHINE LEARNING

Nagoya University, Fall 2023

Lecture 7

Neural Networks IV:
Finishing Backpropagation & Convolutional Neural Networks

This week Tutorial: Wednesday 22nd Nov. 5th period

https://www.henrikbachmann.com/mml2023.html

Semester project

Objective: Choose a project topic related to Nagoya or Japan more broadly that can be addressed using machine learning algorithms. Your task is to develop a machine learning model to solve a specific problem or provide insights into an aspect of life, business, environment, culture, etc., in Nagoya/Japan.

Group Size: 1-3 members

Code: Preferably a Google Colab notebook. Exceptions are possible; please provide full documentation for any different technology or package used. If you plan not to submit a Google Colab, please contact us in advance.

Documentation: 5-10 slides as if you were going to present the project.
Your slides should cover (for example):

- Problem Statement
- Data Collection
- Data Exploration and Visualization
- Model Building and Evaluation
- Conclusion

Semester project

- The notebook should contain outputs (we might not run/train)
- Kaggle.com for possible datasets
- You do not need to cure cancer
- A “bad result” is also a result
- Be able to answer questions about your project in person
- We are not strict about the “Japan & Nagoya” connection

Neural Network: forward pass

Definition 4.4. *Let $N = (L^{[1]}, \ldots, L^{[r]})$ be a r-layer neural network of input size n and output size m. We want to view it as a function $N : \mathbb{R}^n \to \mathbb{R}^m$ by defining $N(x)$ for an* **input** *$x \in \mathbb{R}^n$ by the output of its last layer $N(x) = a^{[r]}$. Here we define for $i = 1, \ldots, r$ the following:*

(i) The **linear part** *of the layer $L^{[i]}$ is defined by*

$$z^{[i]} = W^{[i]} a^{[i-1]} + b^{[i]},$$

where $a^{[i-1]} \in \mathbb{R}^{m_{i-1}}$ is the output from the previous layer. In the case $i = 1$ we set $a^{[0]} = x$.

(ii) The **output** *of the layer $L^{[i]}$ is defined by applying the activation function to the linear part, i.e.*

$$a^{[i]} = \sigma^{[i]}(z^{[i]}) \in \mathbb{R}^{m_i}$$

$$\begin{tikzcd}
a^{[0]} = x \arrow[r] & \ldots \arrow[rr] & & z^{[r-1]} \arrow[rr, "\sigma^{[r-1]}"] & & a^{[r-1]} \arrow[rr] & & z^{[r]} \arrow[rr, "\sigma^{[r]}"] & & a^{[r]} = N(x) \\
& & W^{[r-1]} \arrow[ur] & & b^{[r-1]} \arrow[ul] & & W^{[r]} \arrow[ur] & & b^{[r]} \arrow[ul] &
\end{tikzcd}$$

Backpropagation

Suppose we have a training set $\mathcal{T} = \big((x^{(1)}, y^{(1)}), \ldots, (x^{(t)}, y^{(t)})\big)$

For a neural network N, we will think of the collection of all weight matrices and biases as a vector $\theta \in \mathbb{R}^d$, called the **parameters**. For example, θ_1 could be the top left entry of the first weight matrix. The output of a neural network depends on the current choice of θ and therefore we write N_θ.

For example, we could use the sum of squares as a cost function

$$J(\theta) = \frac{1}{2} \sum_{j=1}^{t} \| N_\theta(x^{(j)}) - y^{(j)} \|^2 \, .$$

For gradient descent, we need to calculate the gradient of J.

Backpropagation: A simple example in one dimensions

$$N : \mathbb{R} \to \mathbb{R} \qquad \mathcal{T} = ((x, y))$$

$$x = a^{[0]} \longrightarrow a^{[1]} \longrightarrow a^{[2]} = N_\theta(x) \qquad r = 2$$

$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}, \quad a^{[1]} = \sigma^{[1]}(z^{[1]}) \quad W^{[1]}, b^{[1]} \in \mathbb{R}$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}, \quad a^{[2]} = \sigma^{[2]}(z^{[2]}) \quad W^{[2]}, b^{[2]} \in \mathbb{R}$$

$$\Theta = \begin{pmatrix} W^{[1]} \\ b^{[1]} \\ W^{[2]} \\ b^{[2]} \end{pmatrix} \qquad J(\Theta) = \frac{1}{2}\left(N_\theta(x) - y\right)^2$$

$$= \frac{1}{2}\left(a^{[2]} - y\right)^2.$$

Want: $\nabla J = \begin{pmatrix} \frac{\partial J}{\partial W^{[1]}} \\ \frac{\partial J}{\partial b^{[1]}} \\ \frac{\partial J}{\partial W^{[2]}} \\ \frac{\partial J}{\partial b^{[2]}} \end{pmatrix}$

Want: $\nabla J = \begin{pmatrix} \frac{\partial J}{\partial W^{[1]}} \\ \frac{\partial J}{\partial b^{[1]}} \\ \frac{\partial J}{\partial W^{[2]}} \\ \frac{\partial J}{\partial b^{[2]}} \end{pmatrix}$ $\quad J(\theta) = \frac{1}{2}(N_\theta(x) - y)^2$

$$= \frac{1}{2}(a^{[2]} - y)^2$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}, \quad a^{[2]} = \sigma^{(2)}(z^{[2]})$$

$$\frac{\partial J}{\partial b^{[2]}} = \boxed{\frac{\partial J}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}}} \frac{\partial z^{[2]}}{\partial b^{[2]}}$$

$$\frac{\partial J}{\partial z^{[2]}} = \delta^{[2]} \quad (a^{[2]} - y) \cdot \sigma^{(2)'}(z^{[2]}) \cdot 1$$

$$\frac{\partial J}{\partial W^{[2]}} = \boxed{\frac{\partial J}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}}} \underbrace{\frac{\partial z^{[2]}}{\partial W^{[2]}}}_{a^{[1]}}$$

Recall chain rule

$$y = f(g(x))$$

Leibniz' notation

$$\frac{\partial y}{\partial x} = \underbrace{\frac{\partial y}{\partial g}}_{f'(g)} \underbrace{\frac{\partial g}{\partial x}}_{g'(x)}$$

$$\frac{\partial J}{\partial b^{[1]}} = \underbrace{\underbrace{\frac{\partial J}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}}}_{\delta^{[2]}} \underbrace{\frac{\partial z^{[2]}}{\partial a^{[1]}}}_{W^{[2]}} \underbrace{\frac{\partial a^{[1]}}{\partial z^{[1]}}}_{\sigma^{[1]}(z^{[1]})}}_{\delta^{[1]}} \underbrace{\frac{\partial z^{[1]}}{\partial b^{[1]}}}_{1}$$

$$\frac{\partial J}{\partial W^{[1]}} = \delta^{[1]} \cdot \underbrace{\frac{\partial z^{[1]}}{\partial W^{[1]}}}_{a^{0}}$$

Backpropagation: General idea

Suppose we have a training set $\mathcal{T} = ((x^{(1)}, y^{(1)}), \dots, (x^{(t)}, y^{(t)}))$

Want $\dfrac{\partial J}{\partial W^{[i]}}, \quad \dfrac{\partial J}{\partial b^{[i]}}, \quad \delta^{i} := \dfrac{\partial J}{\partial z^{[i]}} = \begin{pmatrix} \frac{\partial J}{\partial z^{[i]}_1} \\ \frac{\partial J}{\partial z^{[i]}_2} \\ \vdots \\ \frac{\partial J}{\partial z^{[i]}_{m_i}} \end{pmatrix}$

$$\frac{\partial J}{\partial W^{[i]}} = \begin{array}{c} \xleftarrow{\quad m_{i-1} \quad} \\ \begin{pmatrix} \frac{\partial J}{\partial W^{[i]}_{1,1}} & & \\ & \ddots & \\ & & \end{pmatrix} \end{array} \updownarrow m_i \qquad \frac{\partial J}{\partial b^{[i]}} = \begin{pmatrix} \frac{\partial J}{\partial b^{[i]}_1} \\ \vdots \\ \frac{\partial J}{\partial b^{[i]}_{m_i}} \end{pmatrix}$$

m_i : size of layer i

$$\begin{tikzcd}
 & & & & & & & y^{(j)} \arrow[d] \\
x^{(j)} \arrow[r] & \cdots \arrow[r] & z^{[r-1]} \arrow[r, "\sigma^{[r-1]}"] & a^{[r-1]} \arrow[rr] & & z^{[r]} \arrow[r, "\sigma^{[r]}"] & a^{[r]} \arrow[r] & J \\
 & W^{[r-1]} \arrow[ur] & b^{[r-1]} \arrow[u] & & W^{[r]} \arrow[ur] & b^{[r]} \arrow[u] & &
\end{tikzcd}$$

$W^{[i]} \in \mathbb{R}^{m_i \times m_{i-1}}$

$z^{[r]} = W^{[r]} a^{[r-1]} + b^{[r]}$

General chain rule

$$g_j = g_j(\theta_1, \ldots, \theta_p)$$

$$J = J(g_1, \ldots, g_m) \in \mathbb{R}$$

$$\frac{\partial J}{\partial \theta_i} = \sum_{j=1}^{m} \frac{\partial J}{\partial g_j} \frac{\partial g_j}{\partial \theta_i}$$

General Backpropagation

Want

$$\delta^{[i]} = \frac{\partial J}{\partial z^{[i]}} = \begin{pmatrix} \frac{\partial J}{\partial z_1^{[i]}} \\ \vdots \\ \frac{\partial J}{\partial z_{m_i}^{[i]}} \end{pmatrix} \qquad 1 \le k \le m_i$$

$$\frac{\partial J}{\partial z_k^{[i]}} = \frac{\partial J}{\partial a_k^{[i]}} \cdot \frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} \qquad \sigma^{[i]\prime}(z_k^{[i]})$$

$$= \sum_{r=1}^{m_{i+1}} \boxed{\frac{\partial J}{\partial z_r^{[i+1]}}} \boxed{\frac{\partial z_r^{[i+1]}}{\partial a_k^{[i]}}} \cdot \sigma^{[i]\prime}(z_k^{[i]})$$

$\delta_r^{[i+1]}$ $\quad W_{r,k}^{(i+1)}$ $\qquad z^{(i+1)} = W^{(i+1)} a^{(i)} + b^{(i+1)}$

$$r \longrightarrow \overbrace{\begin{pmatrix} \ddots & & \\ & W_{r,k}^{(i+1)} & \\ & & \ddots \end{pmatrix}}^{W^{(i+1)}} \begin{pmatrix} a_1^{(i)} \\ \vdots \\ a_k^{(i)} \\ \vdots \\ a_{m_i}^{(i)} \end{pmatrix}$$

$$= \underbrace{\sum_{r=1}^{m_{i+1}} \delta_r^{(i+1)} \cdot W_{r,k}^{(i+1)}}_{\left(W^{(i+1)T} \delta^{[i+1]}\right)_k} \cdot \underbrace{\sigma^{(i)'}\left(z_k^{(i)}\right)}_{\sigma^{(i)'}\left(z_k^{(i)}\right)}$$

$$\delta^{(i)} = \frac{\partial J}{\partial z^{(i)}} = W^{(i+1)T} \delta^{(i+1)} \odot \sigma^{(i)'}\left(z^{(i)}\right)$$

component-wise multiplication

Backpropagation

Compute and store the values of $a^{[k]}$'s and $z^{[k]}$'s for $k = 1, \ldots, r$, and J.
. $\triangleright$ This is often called the "forward pass"
.
for $k = r$ to 1 **do** $\triangleright$ This is often called the "backward pass"
    if $k = r$ **then**
        compute $\delta^{[r]} \triangleq \frac{\partial J}{\partial z^{[r]}}$
    else
        compute

$$\delta^{[k]} \triangleq \frac{\partial J}{\partial z^{[k]}} = \left(W^{[k+1]^\top} \delta^{[k+1]}\right) \odot \mathrm{ReLU}'(z^{[k]})$$

    Compute

$$\frac{\partial J}{\partial W^{[k]}} = \delta^{[k]} a^{[k-1]^\top}$$
$$\frac{\partial J}{\partial b^{[k]}} = \delta^{[k]}$$

See Lecture 6 notebook

Neural Networks so far

The fully connected Neural Networks we discussed so far come with some problems:

1. **High Parameter Count**: Excessive parameters lead to computational and memory inefficiency.

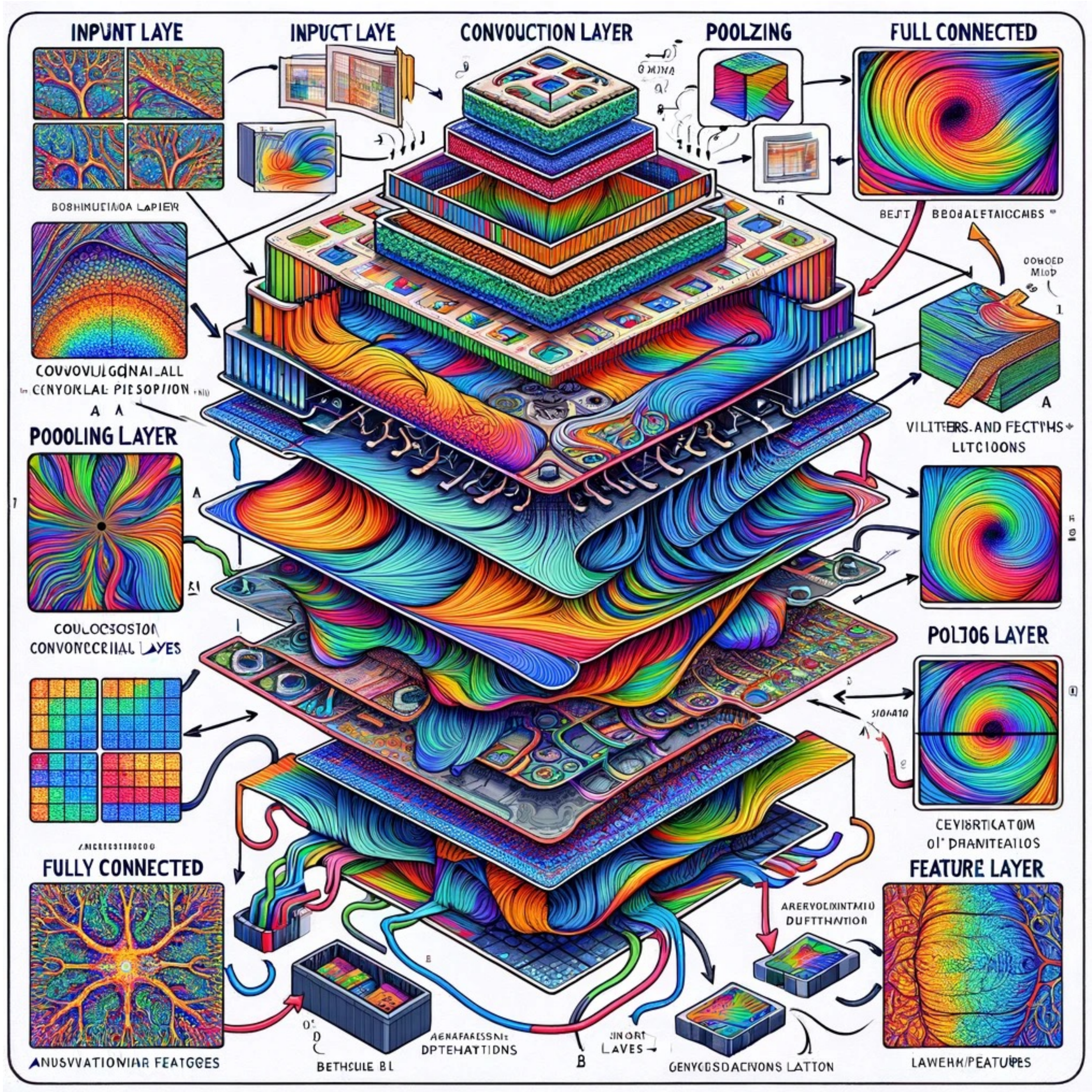
2. **Overfitting Risk**: Prone to overfitting due to a large number of parameters.

3. **Spatial Inefficiency**: Inability to efficiently process spatial data and structure in images.

4. **No Translation Invariance**: Lacks inherent ability to recognize shifted or translated features.

5. **Unsuitable for (large) Images**: Increasing image size leads to a disproportionate increase in parameters.

Convolutional Neural Networks (according to ChatGPT)

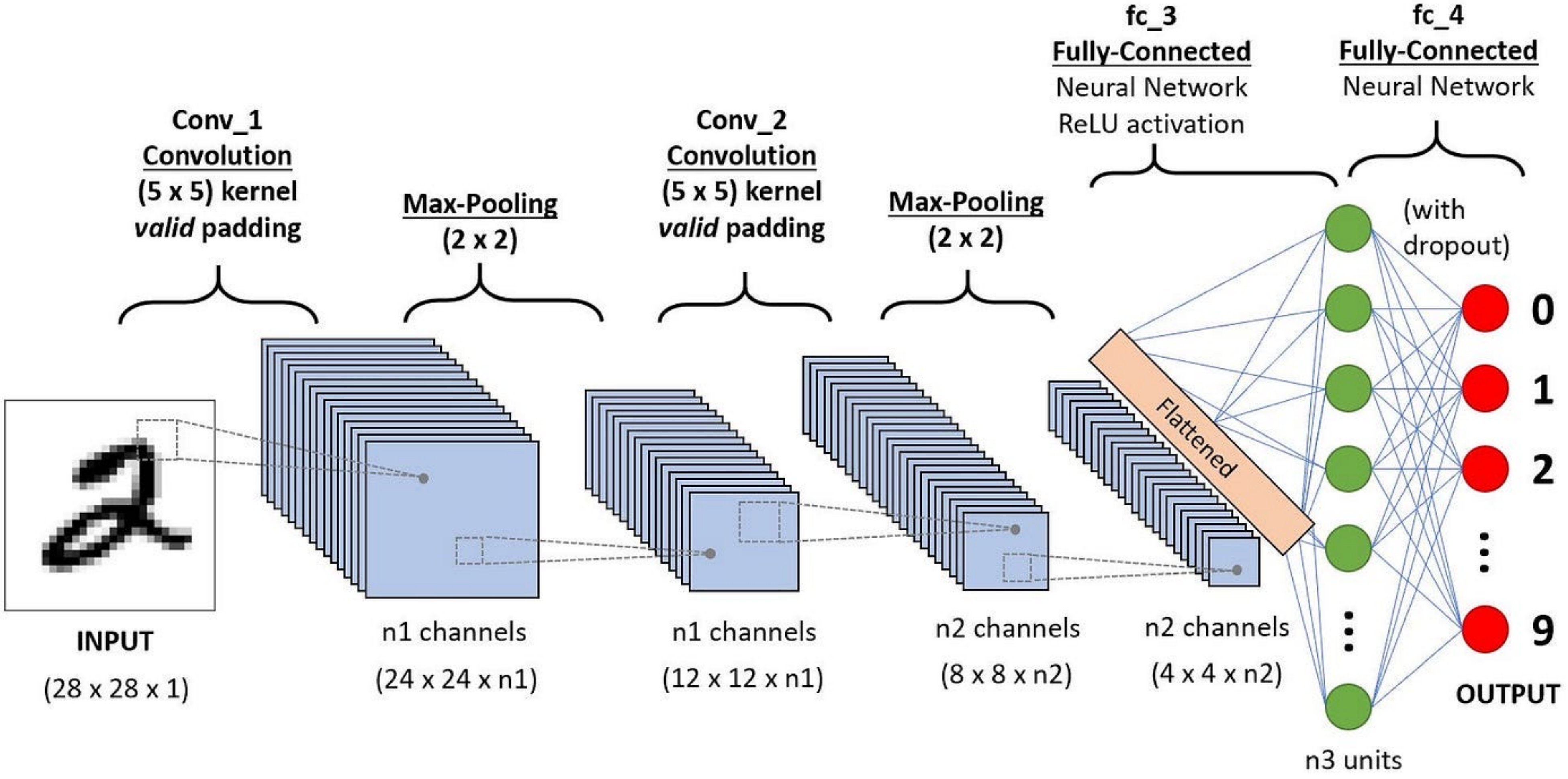


Convolutional Neural Networks (more Japanese)

What is convolution?

$$\underbrace{\begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 2 & 1 \\ \hline 0 & 1 & 0 & 2 \\ \hline \end{array}}_{\text{Picture}} \circledast \underbrace{\begin{array}{|c|c|}\hline 0 & 1 \\ \hline -1 & 0 \\ \hline \end{array}}_{\text{Filter / Kernel}} = \begin{pmatrix} \overbrace{1\cdot 0 + 2\cdot 1 + 0\cdot(-1) + 0\cdot 0}^{2} & 4 \\ 0 & 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 2 & 4 \\ 0 & 1 \end{pmatrix}}_{\text{Feature map}}$$

Convolutional Neural Networks



Types of layers

1.Dense (Fully Connected) Layer: Every neuron in this layer is connected to all neurons in the previous layer, commonly used for classification or regression tasks. (the only thing we considered so far)

2.Convolutional Layer: Applies a set of learnable filters to extract spatial features from data like images.

3.Pooling Layer: Reduces the spatial dimensions (width, height) of the input volume, commonly used for downsampling. Examples include Max Pooling and Average Pooling. (e.g. 2x2-MaxPooling)

4.Dropout Layer: Randomly sets a fraction of the input units to 0 during training, helping prevent overfitting.

CNN for MNIST

https://www.kaggle.com/code/amyjang/tensorflow-mnist-cnn-tutorial