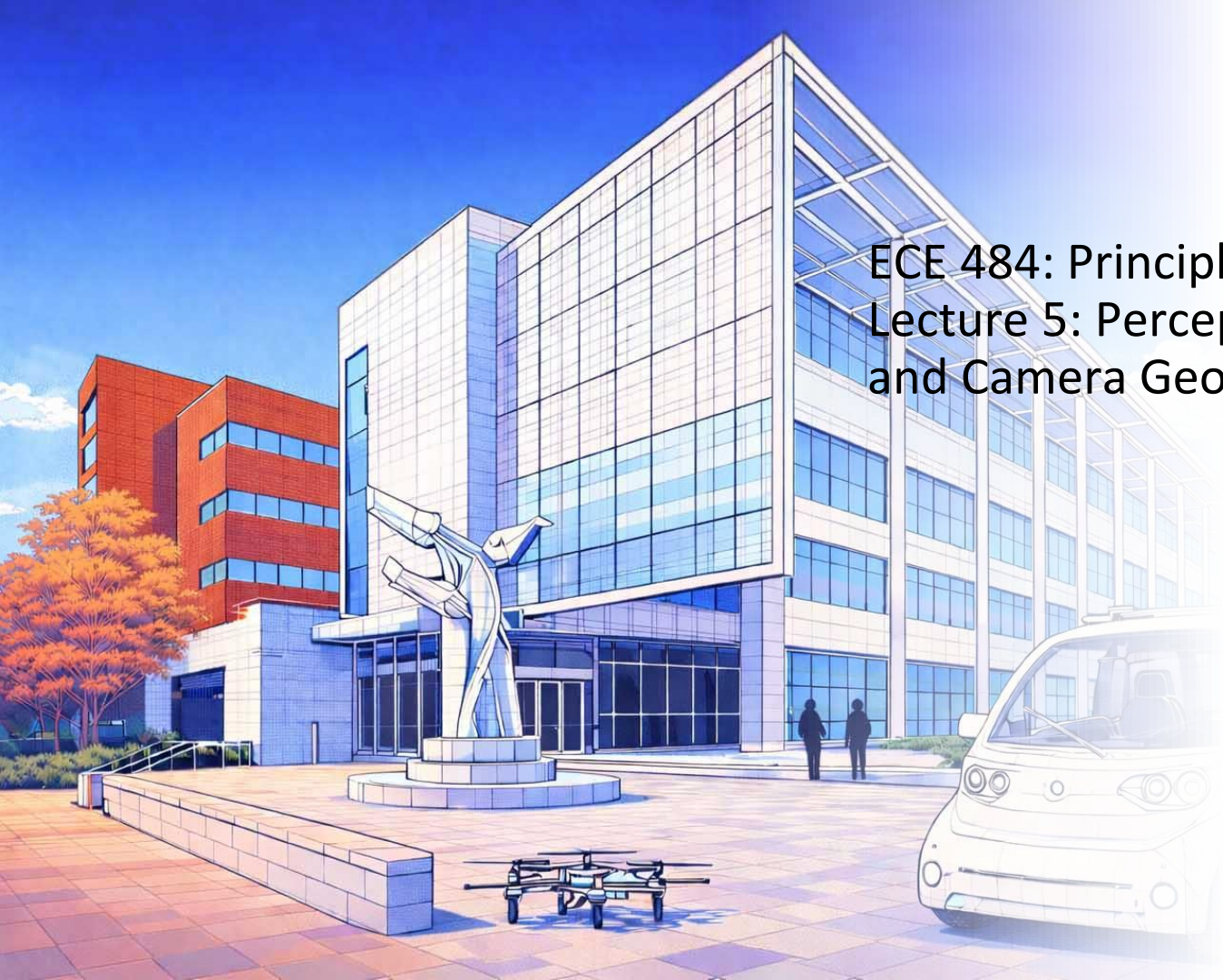


ECE 484: Principles of Safe Autonomy
Lecture 5: Perception: NN Architectures and Camera Geometry

Prof. Huan Zhang

Fall 2026

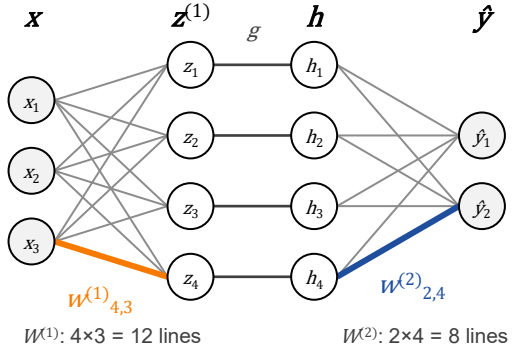
Slides adapted in part from materials by Prof. Sayan Mitra.

Outline

- Review: forward pass and backpropagation (with exercises)
- Neural network architectures: convolution, pooling, normalization, dropout
- Cameras: from the 3-D world to the image — coordinate frames and rigid transformations
- Next lecture: perspective projection, intrinsic parameters, calibration

Review: the two-layer network, its loss, and the gradients we need

2-layer network, d = 3, m = 4, k = 2; biases not drawn



$W^{(1)}$: 4×3 = 12 lines $W^{(2)}$: 2×4 = 8 lines

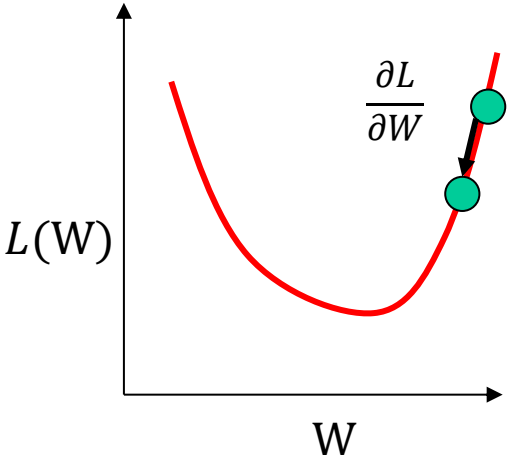
Forward, one example: $\boldsymbol{z}^{(1)} = W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)},\ \boldsymbol{h} = g(\boldsymbol{z}^{(1)}),\ \hat{\boldsymbol{y}} = W^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}$

Shapes (d inputs, m hidden units, k outputs; figure: $d = 3, m = 4, k = 2$): $\boldsymbol{x} \in \mathbb{R}^d$, $W^{(1)} \in \mathbb{R}^{m\times d}$, $\boldsymbol{b}^{(1)}, \boldsymbol{z}^{(1)}, \boldsymbol{h} \in \mathbb{R}^m$, $W^{(2)} \in \mathbb{R}^{k\times m}$, $\boldsymbol{b}^{(2)}, \hat{\boldsymbol{y}}, \boldsymbol{y} \in \mathbb{R}^k$

Loss (regression, one example): $L = \frac{1}{2}\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2 = \frac{1}{2}\sum_{j=1}^{k}(\hat{y}_j - y_j)^2$ ($\hat{y}_j, y_j$: j-th output; for a batch, average over the examples)

Training = gradient descent on L: $W^{(\ell)} \leftarrow W^{(\ell)} - \eta\,\frac{\partial L}{\partial W^{(\ell)}},\ \boldsymbol{b}^{(\ell)} \leftarrow \boldsymbol{b}^{(\ell)} - \eta\,\frac{\partial L}{\partial \boldsymbol{b}^{(\ell)}}$, $\ell = 1,2$

So we need $\frac{\partial L}{\partial W^{(2)}}, \frac{\partial L}{\partial \boldsymbol{b}^{(2)}}, \frac{\partial L}{\partial W^{(1)}}, \frac{\partial L}{\partial \boldsymbol{b}^{(1)}}$ — every gradient has the shape of its parameter, and all come from the **chain rule** (backpropagation)



Q: How many parameters (weights + biases) does the network in the figure have?

$W^{(1)}$: $4 \times 3 = 12$, $\boldsymbol{b}^{(1)}$: 4, $W^{(2)}$: $2 \times 4 = 8$, $\boldsymbol{b}^{(2)}$: 2 → 26 parameters. Gradient descent updates all 26 at once; the gradient of L has 26 entries.

Q ($k = 2$): $L = \frac{1}{2}\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2 = \frac{1}{2}[(\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2]$, solve $\frac{\partial L}{\partial \hat{\boldsymbol{y}}}$

Notation: $\hat{\boldsymbol{y}}, \boldsymbol{y} \in \mathbb{R}^2$ are vectors (bold); $\hat{y}_1, \hat{y}_2$ and y_1, y_2 are their entries (the two outputs of the network, not two examples).

$\frac{\partial L}{\partial \hat{\boldsymbol{y}}}$ has one entry per output, shape 1×2: $\left(\frac{\partial L}{\partial \hat{y}_1}, \frac{\partial L}{\partial \hat{y}_2}\right)$

$\frac{\partial L}{\partial \hat{y}_1} = \hat{y}_1 - y_1, \quad \frac{\partial L}{\partial \hat{y}_2} = \hat{y}_2 - y_2$ (the $\frac{1}{2}$ cancels the 2 from differentiating the square)

So $\frac{\partial L}{\partial \hat{\boldsymbol{y}}} = [\hat{y}_1 - y_1, \hat{y}_2 - y_2] = (\hat{\boldsymbol{y}} - \boldsymbol{y})^T$ — the prediction error

Exercise: $\hat{\boldsymbol{y}} = W^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}$, $W^{(2)} \in \mathbb{R}^{2\times 4}$ $(m = 4)$, solve $\frac{\partial \hat{\boldsymbol{y}}}{\partial W^{(2)}}$

$$\hat{\boldsymbol{y}} = \begin{bmatrix}\hat{y}_1\\ \hat{y}_2\end{bmatrix} = \begin{bmatrix} w_{11}^{(2)}h_1 + w_{12}^{(2)}h_2 + w_{13}^{(2)}h_3 + w_{14}^{(2)}h_4 + b_1^{(2)} \\ w_{21}^{(2)}h_1 + w_{22}^{(2)}h_2 + w_{23}^{(2)}h_3 + w_{24}^{(2)}h_4 + b_2^{(2)} \end{bmatrix}$$

($w_{ij}^{(2)}$: entry (i,j) of $W^{(2)}$, the weight from h_j to $\hat{y}_i$)

Number of elements in $\frac{\partial \hat{\boldsymbol{y}}}{\partial W^{(2)}}$: $2 \times 2 \times 4 = 16$ (two outputs, 2×4 elements in $W^{(2)}$)

For simplicity, we can "vectorize" $W^{(2)}$ into a vector with 8 elements (row by row). Then we calculate the 2×8 Jacobian matrix:

$$\frac{\partial \hat{\boldsymbol{y}}}{\partial \mathrm{vec}(W^{(2)})} = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial w_{11}^{(2)}} & \frac{\partial \hat{y}_1}{\partial w_{12}^{(2)}} & \frac{\partial \hat{y}_1}{\partial w_{13}^{(2)}} & \frac{\partial \hat{y}_1}{\partial w_{14}^{(2)}} & \frac{\partial \hat{y}_1}{\partial w_{21}^{(2)}} & \frac{\partial \hat{y}_1}{\partial w_{22}^{(2)}} & \frac{\partial \hat{y}_1}{\partial w_{23}^{(2)}} & \frac{\partial \hat{y}_1}{\partial w_{24}^{(2)}} \\ \frac{\partial \hat{y}_2}{\partial w_{11}^{(2)}} & \frac{\partial \hat{y}_2}{\partial w_{12}^{(2)}} & \frac{\partial \hat{y}_2}{\partial w_{13}^{(2)}} & \frac{\partial \hat{y}_2}{\partial w_{14}^{(2)}} & \frac{\partial \hat{y}_2}{\partial w_{21}^{(2)}} & \frac{\partial \hat{y}_2}{\partial w_{22}^{(2)}} & \frac{\partial \hat{y}_2}{\partial w_{23}^{(2)}} & \frac{\partial \hat{y}_2}{\partial w_{24}^{(2)}} \end{bmatrix}$$

$$= \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_1 & h_2 & h_3 & h_4 \end{bmatrix}$$

(row i: $\frac{\partial \hat{y}_i}{\partial w_{ij}^{(2)}} = h_j$, every other entry is 0)

Q: $\frac{\partial \hat{y}_1}{\partial w_{13}^{(2)}} = ?$ and $\frac{\partial \hat{y}_2}{\partial w_{13}^{(2)}} = ?$ (then fill in the whole 2×8 matrix)

h_3 and 0: $\hat{y}_1$ uses only row 1 of $W^{(2)}$, $\hat{y}_2$ only row 2 — half of the Jacobian is zero

Abusing the notation a little bit for simplicity — we can call this $\frac{\partial \hat{\boldsymbol{y}}}{\partial W^{(2)}}$

Backprop review: the output-layer weight gradient

Shape of a Jacobian: a function with m inputs and k outputs ($\boldsymbol{f}: \mathbb{R}^m \to \mathbb{R}^k$) has a $k \times m$ Jacobian $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$: one row per output, one column per input ($k \cdot m$ entries).

Here: L has 2 inputs ($\hat{y}_1, \hat{y}_2$), 1 output → 1×2; $\hat{\boldsymbol{y}}$ has 8 inputs (the entries of $W^{(2)}$), 2 outputs → 2×8; product $(1 \times 2)(2 \times 8) = 1 \times 8$.

Shape 1x8

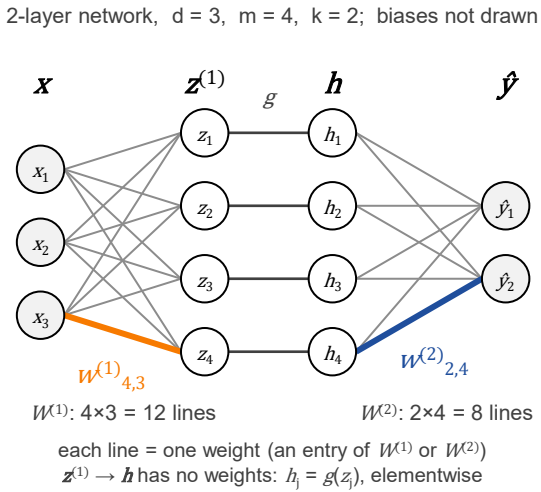
$$\text{Finally, } \frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial W^{(2)}} = [\hat{y}_1 - y_1, \hat{y}_2 - y_2] \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_1 & h_2 & h_3 & h_4 \end{bmatrix}$$

Shape 1x2

Shape 2x8

Backprop review: the first-layer chain (exercises, answers on click)

$$\boldsymbol{z}^{(1)} = W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$$
$$\boldsymbol{h} = \mathrm{ReLU}(\boldsymbol{z}^{(1)})$$
$$\hat{\boldsymbol{y}} = W^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}$$



Now how about $\frac{\partial L}{\partial W^{(1)}}$? How many elements in this gradient?

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}^{(1)}}\frac{\partial \boldsymbol{z}^{(1)}}{\partial W^{(1)}}$$

$\boldsymbol{x} \in \mathbb{R}^3$, $W^{(1)} \in \mathbb{R}^{4\times 3}$, $\boldsymbol{b}^{(1)} \in \mathbb{R}^4$
$\boldsymbol{z}^{(1)} \in \mathbb{R}^4$, $\boldsymbol{h} = \mathrm{ReLU}(\boldsymbol{z}^{(1)}) \in \mathbb{R}^4$
$W^{(2)} \in \mathbb{R}^{2\times 4}$, $\boldsymbol{b}^{(2)} \in \mathbb{R}^2$, $\hat{\boldsymbol{y}}, \boldsymbol{y} \in \mathbb{R}^2$, $L \in \mathbb{R}$

Exercises:

1. What is the shape of each Jacobian matrix? 1x2, 2x4, 4x4, 4x(4x3)
2. What is $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}^{(1)}}$?

A Diagonal matrix with 0 or 1 on its diagonal, depending on the value of $\boldsymbol{z}^{(1)}$ during forward propagation

Backprop review: $\frac{\partial L}{\partial W^{(1)}}$ in closed form (1 of 2) — the four Jacobians

Same network ($d = 3$, $m = 4$, $k = 2$), one example, $L = \frac{1}{2}\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2$, $\boldsymbol{h} = \mathrm{ReLU}(\boldsymbol{z}^{(1)})$. Chain rule:

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}^{(1)}}\frac{\partial \boldsymbol{z}^{(1)}}{\partial W^{(1)}}$$ — write down each factor with its shape:

1. $\frac{\partial L}{\partial \hat{\boldsymbol{y}}} = [\hat{y}_1 - y_1, \hat{y}_2 - y_2]$ (shape 1×2)

2. $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{h}} = W^{(2)} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} & w_{14}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} & w_{24}^{(2)} \end{bmatrix}$ (shape 2×4), because $\hat{y}_i = \sum_{j=1}^{4} w_{ij}^{(2)} h_j + b_i^{(2)}$ gives $\frac{\partial \hat{y}_i}{\partial h_j} = w_{ij}^{(2)}$

3. $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}^{(1)}} = \mathrm{diag}(s_1, s_2, s_3, s_4)$ (4×4), $s_j = 1$ if $z_j^{(1)} > 0$ else 0 — $h_j = \mathrm{ReLU}(z_j^{(1)})$ depends on $z_j^{(1)}$ only

4. $\frac{\partial \boldsymbol{z}^{(1)}}{\partial \mathrm{vec}(W^{(1)})}$: $z_j^{(1)} = w_{j1}^{(1)} x_1 + w_{j2}^{(1)} x_2 + w_{j3}^{(1)} x_3 + b_j^{(1)}$, so $\frac{\partial z_j^{(1)}}{\partial w_{jn}^{(1)}} = x_n$, and row j of $W^{(1)}$ feeds no other $z_j^{(1)}$:

$$\frac{\partial \boldsymbol{z}^{(1)}}{\partial \mathrm{vec}(W^{(1)})} = \begin{bmatrix} x_1 & x_2 & x_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & x_2 & x_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_1 & x_2 & x_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_1 & x_2 & x_3 \end{bmatrix}$$ $(4 \times 12$; $W^{(1)}$ vectorized row by row)

Shape check: $(1 \times 2)(2 \times 4)(4 \times 4)(4 \times 12) = 1 \times 12$ → reshaped to 4×3, the shape of $W^{(1)}$: 12 numbers, one per weight.

Backprop review: $\frac{\partial L}{\partial W^{(1)}}$ in closed form (2 of 2) — multiply, then reshape

Step 1 — through the output layer $(1 \times 2)(2 \times 4) = 1 \times 4$ — each output error, weighted by the outgoing weights of h_j:

$$\frac{\partial L}{\partial \boldsymbol{h}} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{h}} = [\hat{y}_1 - y_1, \hat{y}_2 - y_2]\, W^{(2)} = [u_1, u_2, u_3, u_4], \quad u_j = (\hat{y}_1 - y_1)\, w_{1j}^{(2)} + (\hat{y}_2 - y_2)\, w_{2j}^{(2)}$$

Step 2 — through the ReLU $(1 \times 4)(4 \times 4) = 1 \times 4$ — an inactive unit ($s_j = 0$) passes no gradient:

$$\frac{\partial L}{\partial \boldsymbol{z}^{(1)}} = [u_1, u_2, u_3, u_4]\, \mathrm{diag}(s_1, s_2, s_3, s_4) = [s_1 u_1, s_2 u_2, s_3 u_3, s_4 u_4] =: [\delta_1, \delta_2, \delta_3, \delta_4]$$

Step 3 — through the first layer $(1 \times 4)(4 \times 12) = 1 \times 12$:

$$\frac{\partial L}{\partial \mathrm{vec}(W^{(1)})} = [\delta_1, \delta_2, \delta_3, \delta_4] \cdot (4 \times 12 \text{ matrix on the previous slide})$$
$$= [\delta_1 x_1, \delta_1 x_2, \delta_1 x_3, \delta_2 x_1, \delta_2 x_2, \delta_2 x_3, \delta_3 x_1, \delta_3 x_2, \delta_3 x_3, \delta_4 x_1, \delta_4 x_2, \delta_4 x_3]$$

Step 4 — reshape row by row into the shape of $W^{(1)}$ (4×3):

$$\frac{\partial L}{\partial W^{(1)}} = \begin{bmatrix} \delta_1 x_1 & \delta_1 x_2 & \delta_1 x_3 \\ \delta_2 x_1 & \delta_2 x_2 & \delta_2 x_3 \\ \delta_3 x_1 & \delta_3 x_2 & \delta_3 x_3 \\ \delta_4 x_1 & \delta_4 x_2 & \delta_4 x_3 \end{bmatrix} = \boldsymbol{\delta}^{(1)}\, \boldsymbol{x}^T \quad \text{with} \quad \boldsymbol{\delta}^{(1)} = [\delta_1, \delta_2, \delta_3, \delta_4]^T$$

i.e. $\frac{\partial L}{\partial w_{jn}^{(1)}} = \delta_j\, x_n = s_j[(\hat{y}_1 - y_1) w_{1j}^{(2)} + (\hat{y}_2 - y_2) w_{2j}^{(2)}]\, x_n$

Closed form: $\frac{\partial L}{\partial W^{(1)}} = ((W^{(2)})^T(\hat{\boldsymbol{y}} - \boldsymbol{y}) \odot 1[\boldsymbol{z}^{(1)} > 0])\, \boldsymbol{x}^T$; likewise $\frac{\partial L}{\partial \boldsymbol{b}^{(1)}} = \boldsymbol{\delta}^{(1)}$ (since $\frac{\partial z_j^{(1)}}{\partial b_j^{(1)}} = 1$).

Backprop in general: the compact equations

- $\boldsymbol{\delta}^{(2)} := \nabla_{\hat{\boldsymbol{y}}} L = \hat{\boldsymbol{y}} - \boldsymbol{y}$
- $\frac{\partial L}{\partial W^{(2)}} = \boldsymbol{\delta}^{(2)} \boldsymbol{h}^T$ ($k \times m$ outer product), $\frac{\partial L}{\partial \boldsymbol{b}^{(2)}} = \boldsymbol{\delta}^{(2)}$
- $\nabla_{\boldsymbol{h}} L = \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{h}}\right)^T \boldsymbol{\delta}^{(2)} = {W^{(2)}}^T \boldsymbol{\delta}^{(2)}$ (Jacobian of the linear output layer is $W^{(2)}$)
- $\boldsymbol{\delta}^{(1)} := \nabla_{\boldsymbol{z}^{(1)}} L = \mathrm{diag}\left(g'(\boldsymbol{z}^{(1)})\right) \nabla_{\boldsymbol{h}} L = \left({W^{(2)}}^T \boldsymbol{\delta}^{(2)}\right) \odot g'(\boldsymbol{z}^{(1)})$
- $\frac{\partial L}{\partial W^{(1)}} = \boldsymbol{\delta}^{(1)} \boldsymbol{x}^T$ ($m \times d$), $\frac{\partial L}{\partial \boldsymbol{b}^{(1)}} = \boldsymbol{\delta}^{(1)}$
- Cost: one matrix–vector product per layer, backwards — the same order as the forward pass; $\boldsymbol{\delta}^{(2)}$ is computed once and reused. For a batch, average the per-example gradients.

$$\boldsymbol{z}^{(1)} = W^{(1)} \boldsymbol{x} + \boldsymbol{b}^{(1)}$$
$$\boldsymbol{h} = g(\boldsymbol{z}^{(1)})$$
$$\hat{\boldsymbol{y}} = W^{(2)} \boldsymbol{h} + \boldsymbol{b}^{(2)}$$
$$L = \frac{1}{2} \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2 \quad \text{(one sample)}$$

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}^{(1)}} = \mathrm{diag}\left(g'(\boldsymbol{z}^{(1)})\right)$$
diagonal Jacobian of the elementwise activation — the reason $\odot$ appears

Q: Where does the activation's derivative enter? What happens to the gradient of row j of $W^{(1)}$ if unit j is inactive ($z_j^{(1)} < 0$, ReLU)?

In $\boldsymbol{\delta}^{(1)} = (W^{(2)T} \boldsymbol{\delta}^{(2)}) \odot g'(\boldsymbol{z}^{(1)})$: ReLU$'(z_j^{(1)}) = 0$, so $\delta_j^{(1)} = 0$ — row j of $\boldsymbol{\delta}^{(1)} \boldsymbol{x}^T$ and $\partial L / \partial b_j^{(1)}$ are zero: the unit learns nothing on this example.

Q: backprop on a tiny network — compute the numbers

- Network: $x = 1$, two sigmoid hidden units, one linear output, target $y = 0$: $W^{(1)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $\boldsymbol{b}^{(1)} = 0$, $W^{(2)} = [1 \quad 1]$, $b^{(2)} = 0$
- **Forward:** $\boldsymbol{z}^{(1)} = (1, -2)^T$, $\boldsymbol{h} = \sigma(\boldsymbol{z}^{(1)}) = (0.7311, 0.1192)^T$, $\hat{y} = 0.8503$

$$L = \frac{1}{2}\hat{y}^2 = 0.3615$$

- **Backward:** $\delta^{(2)} = \hat{y} - y = 0.8503$
  - $\frac{\partial L}{\partial W^{(2)}} = \delta^{(2)}\boldsymbol{h}^T = (0.6216, 0.1014)$, $\frac{\partial L}{\partial b^{(2)}} = 0.8503$
  - $\nabla_{\boldsymbol{h}} L = {W^{(2)}}^T \delta^{(2)} = (0.8503, 0.8503)^T$, $\sigma'(\boldsymbol{z}^{(1)}) = \boldsymbol{h} \odot (1 - \boldsymbol{h}) = (0.1966, 0.1050)^T$
  - $\boldsymbol{\delta}^{(1)} = \nabla_{\boldsymbol{h}} L \odot \sigma'(\boldsymbol{z}^{(1)}) = (0.1672, 0.0893)^T$
    $\frac{\partial L}{\partial W^{(1)}} = \boldsymbol{\delta}^{(1)} x = (0.1672, 0.0893)^T$, $\frac{\partial L}{\partial \boldsymbol{b}^{(1)}} = \boldsymbol{\delta}^{(1)}$

One gradient step, η = 1: $W^{(2)} \leftarrow (0.378, 0.899)$
$b^{(2)} \leftarrow -0.850$
$W^{(1)} \leftarrow (0.833, -2.089)^T$
$\boldsymbol{b}^{(1)} \leftarrow (-0.167, -0.089)^T$

New forward pass: $\hat{y} = -0.509$, $L = 0.1295 < 0.3615$ — the loss went down (η = 1 overshot the target 0; a smaller η would not).

Check: finite difference $[L(W_1^{(1)} + 10^{-6}) - L]/10^{-6} = 0.16717$ = the backprop value — this is how a hand-written backward pass is debugged.

Notice: $|\delta_j^{(1)}| = \sigma'(z_j)\,|\nabla_{h_j} L|$ with $\sigma' = 0.20, 0.10$: one sigmoid layer already shrank the error signal 5–10× (vanishing gradients, next slides).

Numbers: figure_sources/Spring26_Lecture4_perceptionNN/backprop_worked_example.py

NN architectures

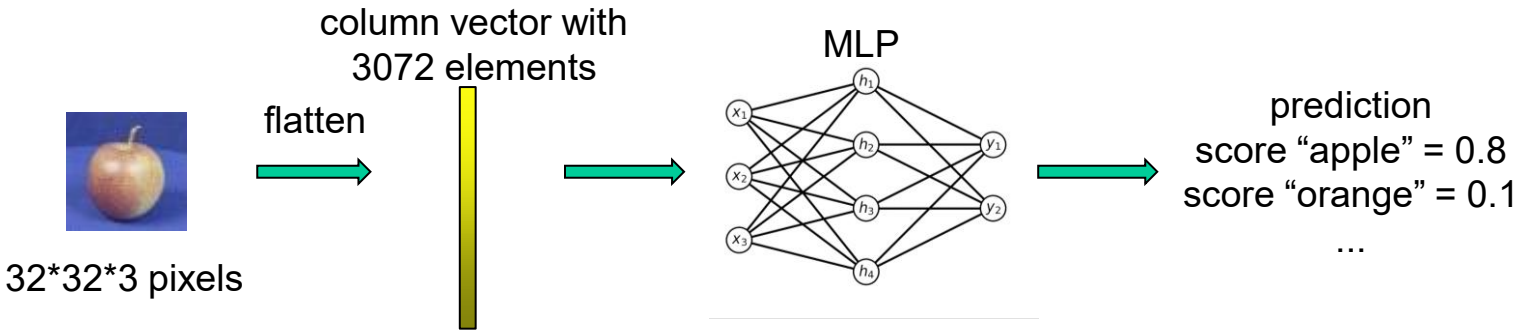
So far every layer has been a plain weight matrix W:

- $\boldsymbol{z}^{(1)} = W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}, \quad \boldsymbol{h} = g(\boldsymbol{z}^{(1)}), \quad \hat{\boldsymbol{y}} = W^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}$

$W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$ is a **fully-connected (FC) layer**; a stack of FC layers with a nonlinear activation between them is a **multilayer perceptron (MLP)**

Q: An MLP on a $32 \times 32 \times 3$ image with 1000 hidden units: how many parameters in the first FC layer?

3072×1000 weights $+1000$ biases ≈ 3.07 million — for a thumbnail-sized image; a 1024×1024 image would need 3.1 billion.
Argument 1 for convolution (argument 2: next slide).



NN architectures: beyond fully-connected layers

Fully connected layer $W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$ can be replaced by other linear or nonlinear operators.

For images, flattening into a 1-D vector and feeding it to an FC layer ignores the spatial structure: the layer has no notion of which entries are neighbouring pixels (vertically adjacent pixels end up a whole image row apart).

Layers that are often used in NNs for perception:

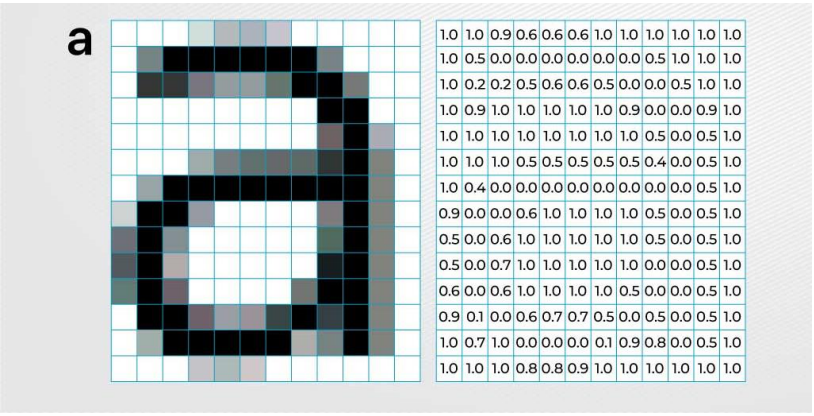
1. Convolutional layers
2. Pooling layers (maxpool, avgpool)
3. Transposed convolution layers
4. Normalization layers
5. Dropout

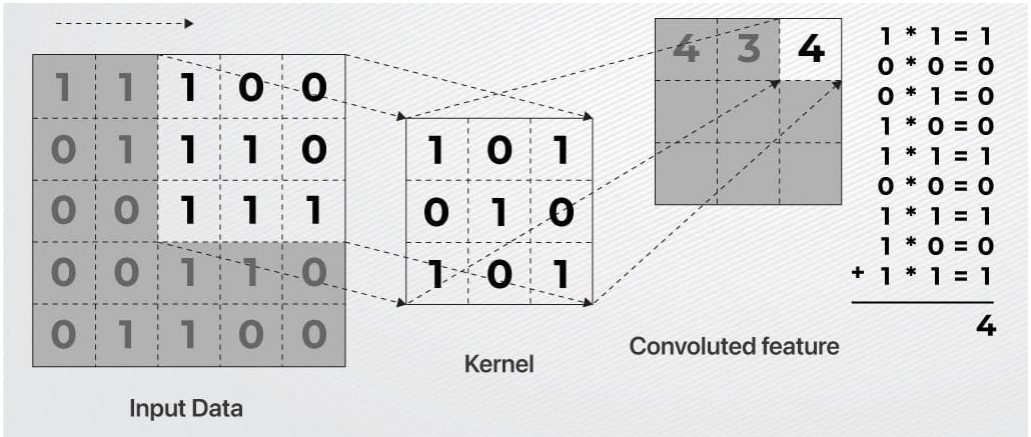
NN architectures: convolutional layers

The convolution operator instead operates on 1D, 2D or 3D inputs directly, combining input features that are spatially adjacent

"Kernel" or "filter" *K* replaces the weights *W*

Typically the spatial dimension shrinks after convolution (unless the input is padded)



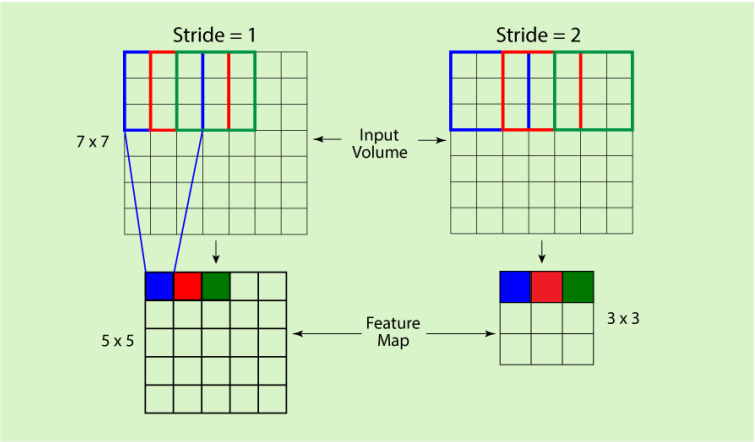


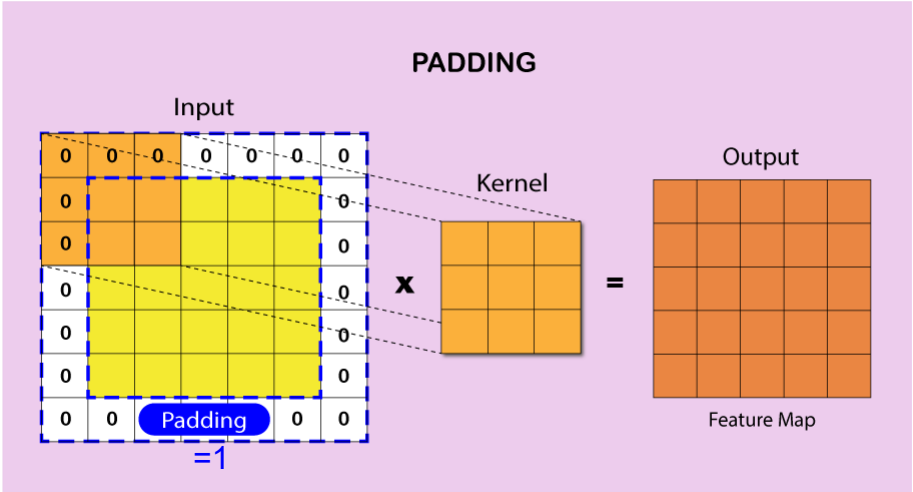
Note: what deep-learning libraries call convolution is cross-correlation — the kernel is not flipped. The distinction does not matter when K is learned.

$\boldsymbol{x}$ (5×5) K (3×3) $\boldsymbol{x} * K$ (3×3)

https://www.analytixlabs.co.in/blog/convolutional-neural-network/

Convolutional layers with stride and padding





Stride: how far the kernel moves between successive positions (stride 1: every position, stride 2: every other)

Q: 7×7 input, 3×3 kernel, stride 2, no padding: output size?

Padding: how many extra rows/columns of zeros are added on all sides of the input

3×3: $\left\lfloor \frac{7-3}{2} \right\rfloor + 1 = 3$. In general $H_{out} = \left\lfloor \frac{H+2p-k}{s} \right\rfloor + 1$ (input H, kernel k, padding p, stride s; with dilation d replace k by $d(k-1)+1$)

https://developersbreach.com/convolution-neural-network-deep-learning/

Convolutional layers with dilation

Dilation d: insert $d-1$ zeros between the kernel taps — a $k \times k$ kernel then covers a $(d(k-1)+1) \times (d(k-1)+1)$ window with the same k^2 weights ($d=1$: ordinary convolution).

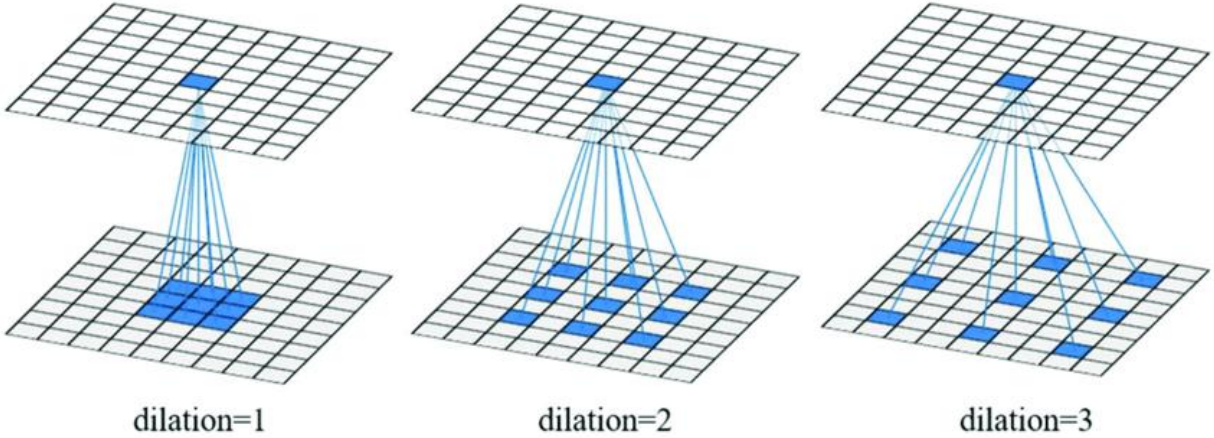


Image from https://www.mdpi.com/2072-4292/11/19/2220

Convolutional layers with multiple input channels and output channels

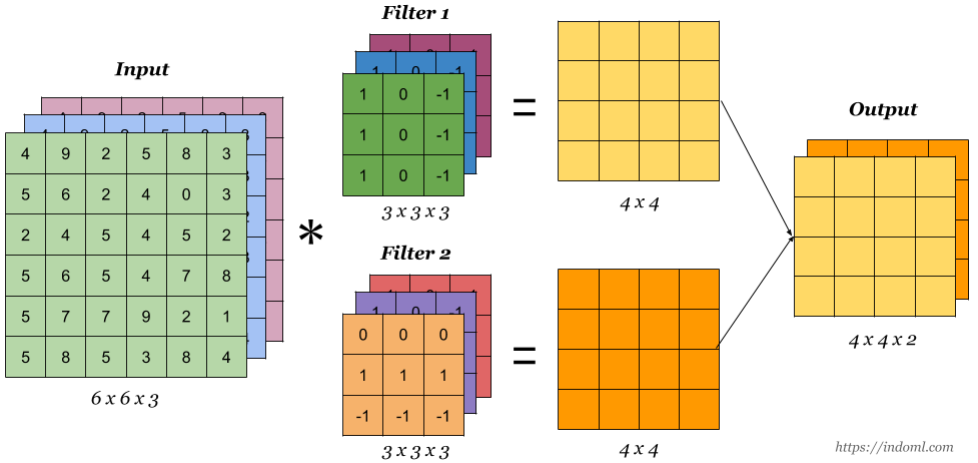
Input dimension: 6×6×3

Kernel (filter) size: 3×3×3 — the depth must equal the 3 input channels

Output channels: 2 — one filter per output channel

Stride: 1

Padding: 0

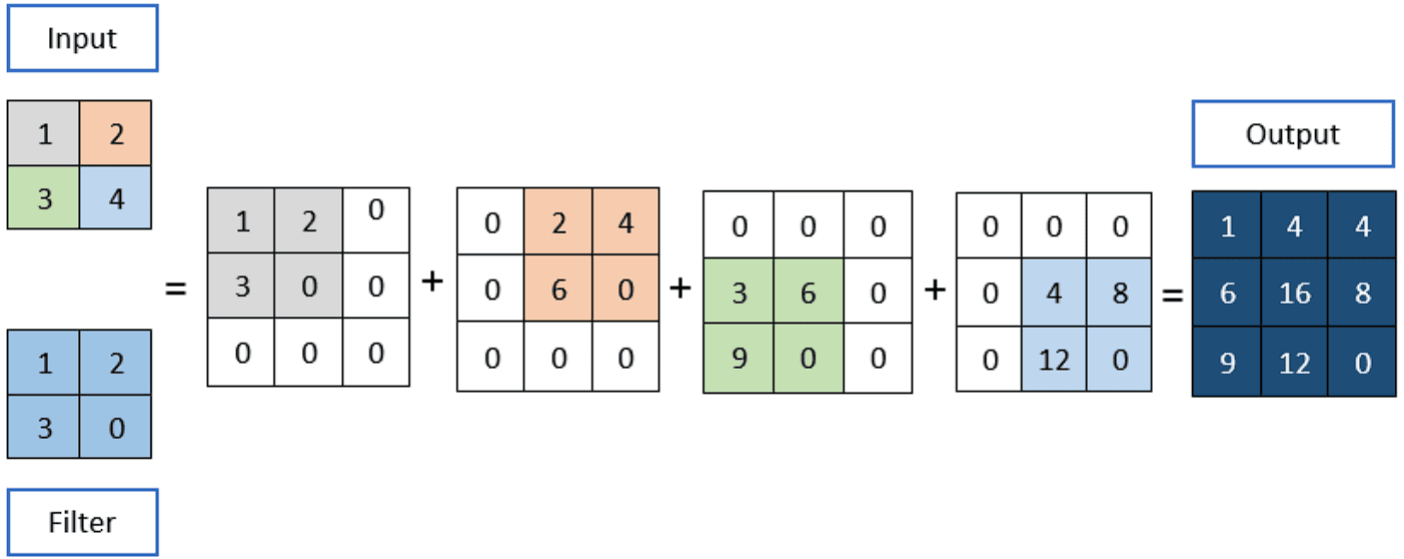


Output $4 \times 4 \times 2$: each $3 \times 3 \times 3$ filter fits in $(6-3)/1+1=4$ positions per axis, one map per filter.
Parameters $2 \times (3 \cdot 3 \cdot 3 + 1) = 56$ — vs. 3488 for a fully-connected layer from 108 inputs to 32 outputs.

Q: For this layer: output size? number of parameters? (HW: derive the general output size from input size, kernel size, stride, padding, dilation)

Transposed convolution

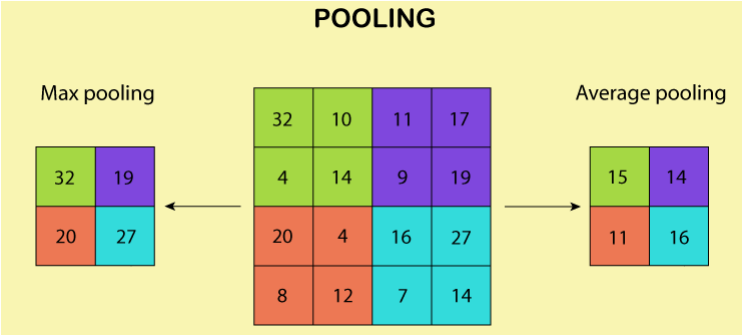
- Transposed convolution can increase the spatial dimension (2x2 -> 3x3 in this example)
- Useful for upsampling



https://viso.ai/deep-learning/convolution-operations/

Pooling layers

Take max of all numbers in the 2x2 square of the same color



Take the average of all numbers in the 2x2 square of the same color

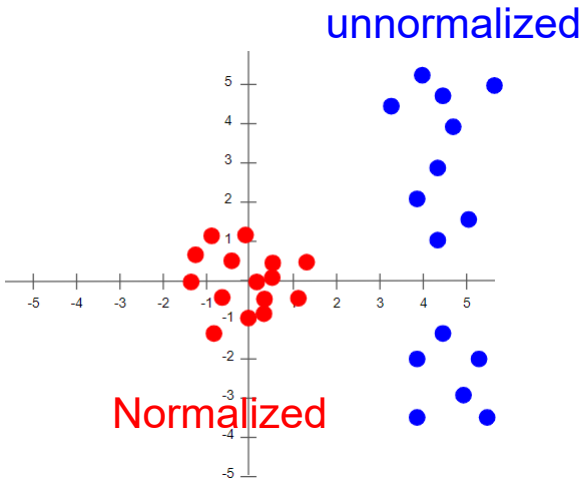
- Pooling hyper-parameters: kernel size, stride, padding (typically, kernel size = stride)
- Pooling layer has no parameters to learn (it is a fixed operation)
- Average pooling is a special case of convolution (why?)

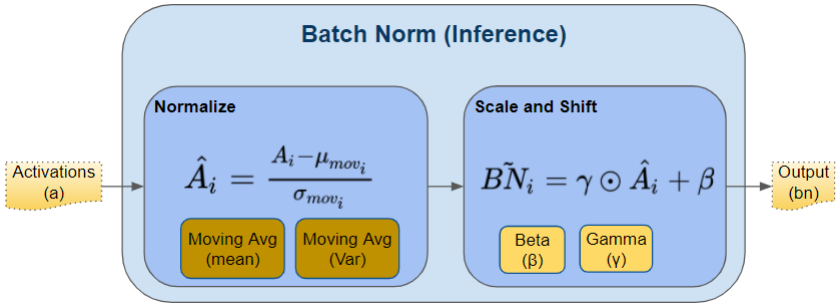
A: yes — average pooling with a $k \times k$ window and stride k is a convolution with the constant kernel $1/k^2$ (per channel) and stride k. Max pooling is not linear, so it is not a convolution: its gradient flows only to the entry that attained the maximum.

https://developersbreach.com/convolution-neural-network-deep-learning/

Batch normalization

- Simple idea: make the features entering a layer have zero mean and unit variance — so normalize them





mean and variance: the current batch's statistics during training, running averages at test time

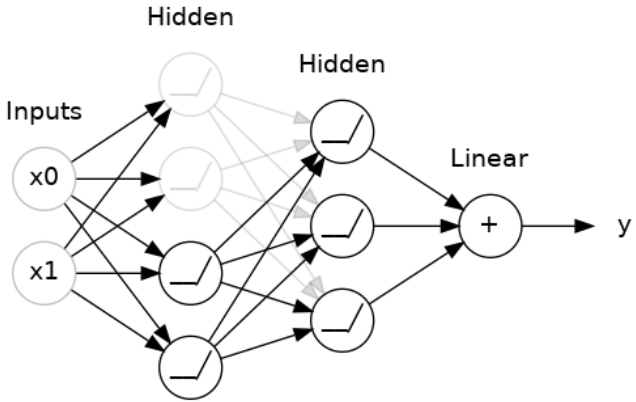
then a learnable scale γ and shift β (can be switched off in code)

https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739/

Read about different types of normalization layers: https://arxiv.org/pdf/1803.08494

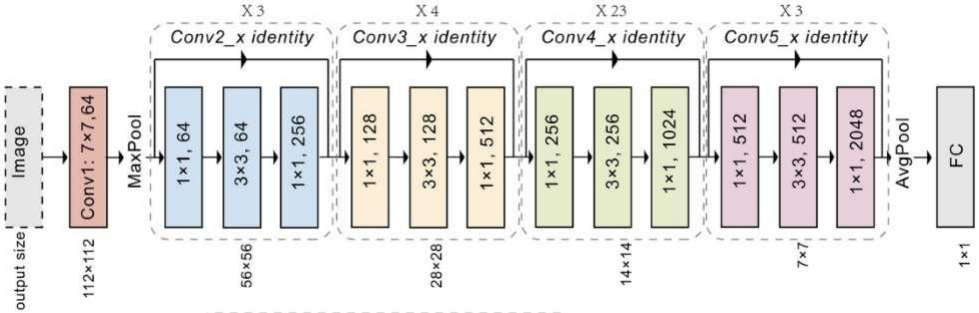
Dropout

- During training, drop each unit (with all of its connections) independently with probability p
- Avoid learning spurious patterns in training data, and tend to obtain a more robust and generalizable NN
- During testing nothing is dropped — with inverted dropout (activations divided by 1−p during training) the layer is then exactly the identity



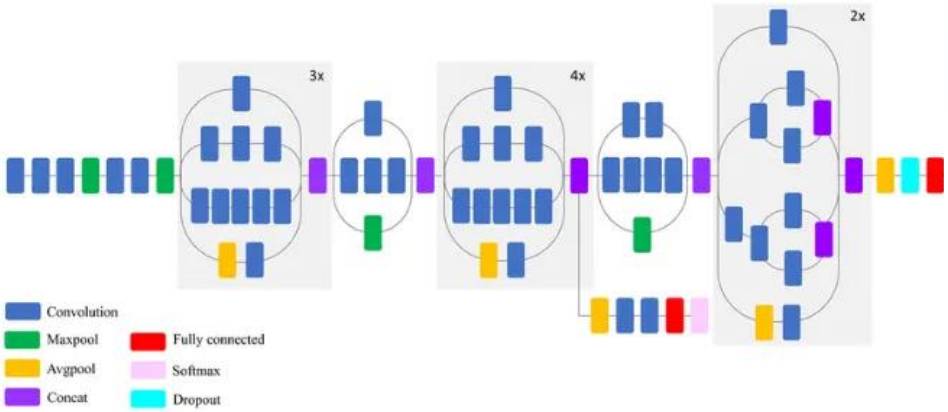
https://www.kaggle.com/code/ryanholbrook/dropout-and-batch-normalization

More general NN architectures

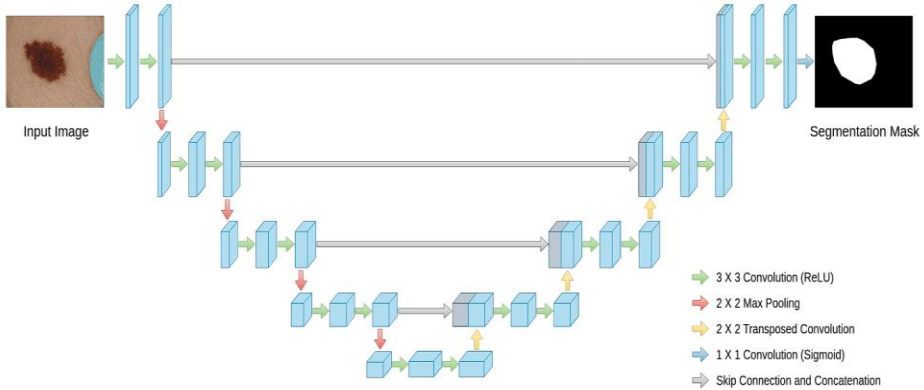


These networks output labels or per-pixel maps. To act, the vehicle also needs geometry: where is that object in 3-D? → Part 3: cameras.

ResNet (residual network)



Inception-v3



U-Net

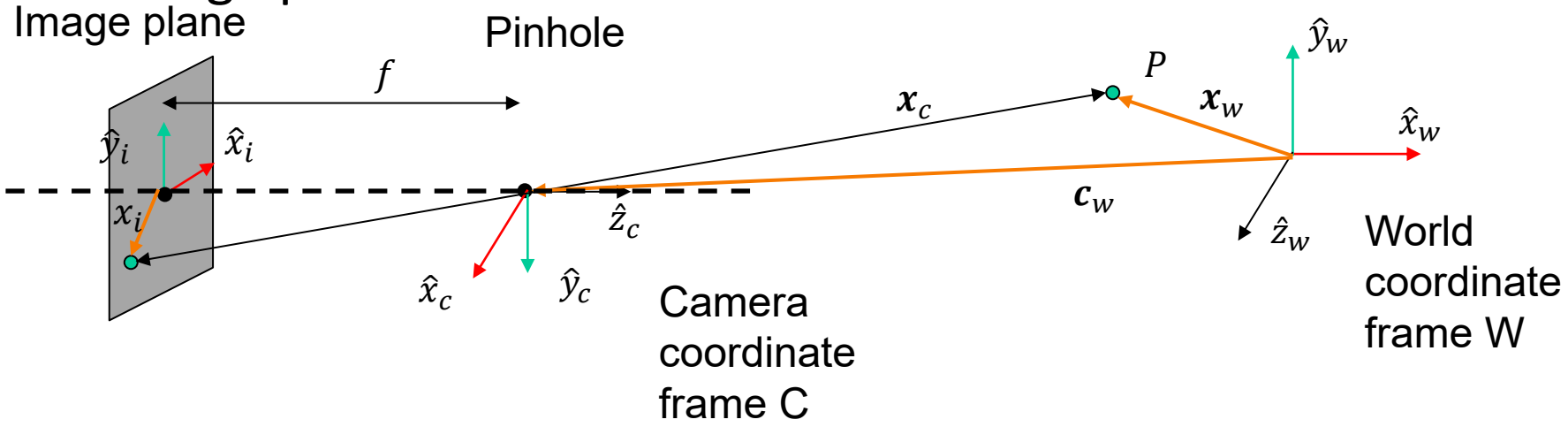
Part 3 — 3D vision: what we want from a camera

- Reconstruct the 3-D structure of the scene from images
- Reconstruct the 6-DoF pose (3 rotation + 3 translation) of the camera from images
- Calibrate the camera: find its internal parameters

- Input: an image, with points located in pixels
- Output: positions in the world frame (metres), the camera pose, the intrinsic parameters

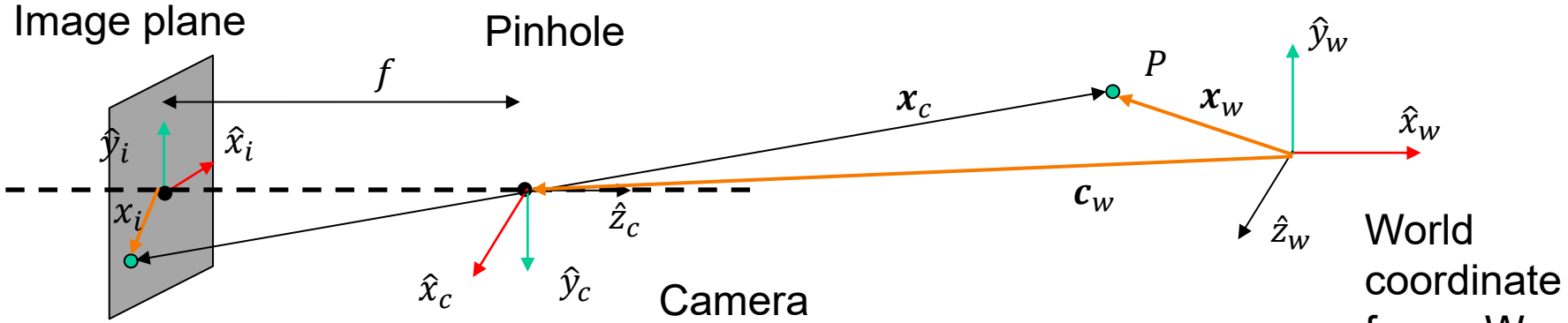
- Today: coordinate frames and the world → camera transformation (extrinsic parameters)
- Next lecture: camera → pixels (perspective projection, intrinsics), then calibration

The forward imaging model: from a 3-D point to a 2-D image point



Slide ideas from First Principles of Computer Vision by Shree Nayar

The forward imaging model has two stages



Q: Which of the two stages can be undone (inverted) from its output alone?

3D→3D (rigid motion): yes — rotation and translation are invertible. 3D→2D (projection): no — every point on the ray through the pinhole lands on the same image point, so the depth z_c is lost (this is why stereo exists).

$$\boldsymbol{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{3D-2D} \quad \boldsymbol{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \quad \text{3D-3D} \quad \boldsymbol{x}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$

Slide ideas from First Principles of Computer Vision by Shree Nayar

Coordinate frames: the same point, different numbers

A **frame** A = an origin O_A + three orthonormal, right-handed axes $\hat{x}_A, \hat{y}_A, \hat{z}_A$

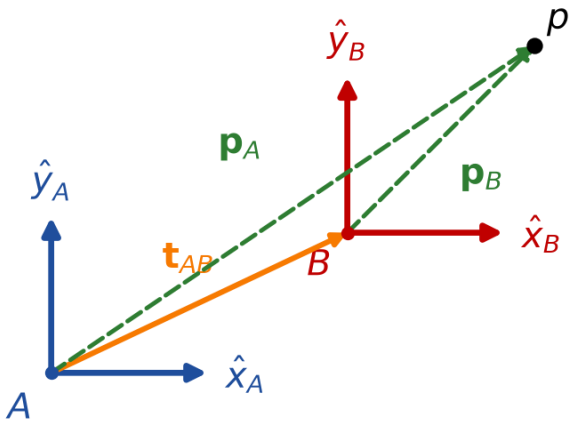
A point p has coordinates $\boldsymbol{p}_A = (x_A, y_A, z_A)^T$ in frame A: $p = O_A + x_A\hat{x}_A + y_A\hat{y}_A + z_A\hat{z}_A$

The same point has different coordinates $\boldsymbol{p}_B$ in another frame B — the vehicle, the camera, the map all have their own frame

Translation only (axes of A and B parallel): let $\boldsymbol{t}_{AB}$ = position of O_B expressed in A. Then

- $\boldsymbol{p}_A = \boldsymbol{p}_B + \boldsymbol{t}_{AB}$, inverse $\boldsymbol{p}_B = \boldsymbol{p}_A - \boldsymbol{t}_{AB}$ $(= \boldsymbol{p}_A + \boldsymbol{t}_{BA}$, with $\boldsymbol{t}_{BA} = -\boldsymbol{t}_{AB})$

Subscript convention: $\boldsymbol{p}_A$ = "coordinates in A"; $\boldsymbol{t}_{AB}, R_{AB}, T_{AB}$ = "takes B-coordinates to A-coordinates"



Q: O_B is at $\boldsymbol{t}_{AB} = (2,1,0)^T$ in A; a point has $\boldsymbol{p}_B = (1,1,3)^T$. What is $\boldsymbol{p}_A$?

$\boldsymbol{p}_A = (1,1,3)^T + (2,1,0)^T = (3,2,3)^T$ — and $\boldsymbol{p}_B = \boldsymbol{p}_A - \boldsymbol{t}_{AB}$ takes you back.

Rotation in 2-D: build the matrix from the axes of the rotated frame

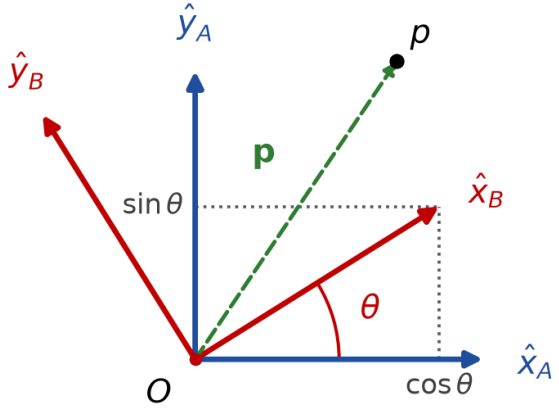
Frame B has the same origin as A, rotated counter-clockwise by θ

Axes of B written in A-coordinates: $\hat{x}_B = \begin{bmatrix}\cos\theta \\ \sin\theta\end{bmatrix}$, $\hat{y}_B = \begin{bmatrix}-\sin\theta \\ \cos\theta\end{bmatrix}$

A point with B-coordinates $\boldsymbol{p}_B = (x_B, y_B)^T$ is $p = O + x_B\hat{x}_B + y_B\hat{y}_B$, so in A-coordinates

- $\boldsymbol{p}_A = x_B\,\hat{x}_B + y_B\,\hat{y}_B = [\hat{x}_B \quad \hat{y}_B]\boldsymbol{p}_B = R_{AB}(\theta)\,\boldsymbol{p}_B$
- $R_{AB}(\theta) = [\hat{x}_B \quad \hat{y}_B] = \begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}$

Rotation matrix R_{AB}: its **columns are the axes of** B **expressed in** A; it maps B-coordinates to A-coordinates



Q: What is R_{BA}, the matrix that takes A-coordinates to B-coordinates?

Rotate back by $-\theta$: $R_{BA} = R_{AB}(-\theta) = \begin{bmatrix}\cos\theta & \sin\theta \\ -\sin\theta & \cos\theta\end{bmatrix} = R_{AB}(\theta)^T$ — the inverse of a rotation is its transpose (next slide: why, in any dimension).

Rotation matrices: what makes a matrix a rotation

Definition: $R \in \mathrm{SO}(3) = \{R \in \mathbb{R}^{3\times 3} \mid R^T R = I,\ \det R = +1\}$ (the "special orthogonal" matrices)

- $R^T R = I$: the columns (the axes) are unit length and mutually orthogonal — equivalently the rows are
- $\det R = +1$: right-handed (orientation preserved); $\det R = -1$ would be a reflection, not a rigid motion

Consequences: $R^{-1} = R^T$ (free inverse); $\|R\boldsymbol{p}\| = \|\boldsymbol{p}\|$ and $(R\boldsymbol{p}) \cdot (R\boldsymbol{q}) = \boldsymbol{p} \cdot \boldsymbol{q}$ — lengths and angles are preserved

Degrees of freedom: 9 entries − 6 constraints (3 unit lengths, 3 orthogonalities) $= 3$ — a rotation in 3-D is 3 numbers (e.g. yaw, pitch, roll)

The same holds in 2-D with $\mathrm{SO}(2)$: 4 entries, 3 constraints, 1 angle

Q: Is $\mathrm{diag}(1,1,-1)$ a rotation matrix? Is $\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$?

No: $R^T R = I$ holds but $\det R = -1$ — it mirrors the z axis and turns a right-handed frame into a left-handed one; no rigid motion does that. Yes: orthonormal columns and $\det R = 0 \cdot 0 - (-1) \cdot 1 = 1$; it is $R_z(90°)$, the yaw by a quarter turn.

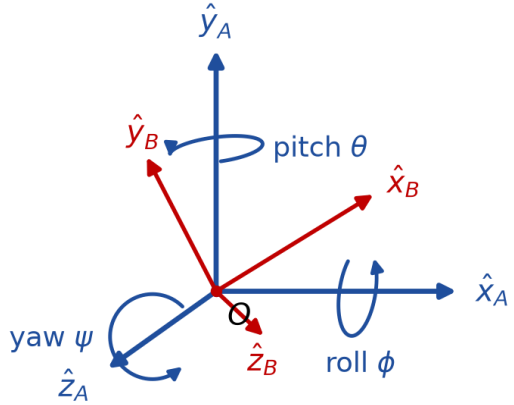
Rotation in 3-D: yaw, pitch and roll

$R_{AB} = [\hat{x}_B \quad \hat{y}_B \quad \hat{z}_B] = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$: column j = axis j of B in A-coordinates

$\boldsymbol{p}_A = R_{AB}\,\boldsymbol{p}_B$; row i = axis i of A in B-coordinates (since $R_{BA} = R_{AB}^T$)

Any rotation = **yaw** ψ about $\hat{z}$, **pitch** θ about $\hat{y}$, **roll** ϕ about $\hat{x}$ (figure) — each a 2-D rotation in one coordinate plane:

- $R = R_z(\psi)\,R_y(\theta)\,R_x(\phi)$ (acting on $\boldsymbol{p}_B$ right to left: roll, then pitch, then yaw)



$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \quad R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_z(\psi)\,R_y(\theta)\,R_x(\phi) = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

Checks: $\theta = \psi = 0$ leaves $R = R_x(\phi)$; each factor has $\det = +1$, hence $\det R = +1$; three angles for the 3 degrees of freedom of $\mathrm{SO}(3)$.
Order matters: $R_x(\phi)\,R_z(\psi) \neq R_z(\psi)\,R_x(\phi)$.

Rotation + translation: the rigid transform

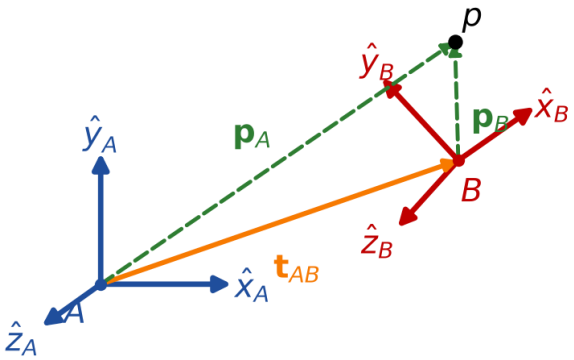
Frame B is rotated by R_{AB} and its origin sits at $\boldsymbol{t}_{AB}$ (in A-coordinates).

Then

- $\boldsymbol{p}_A = R_{AB}\,\boldsymbol{p}_B + \boldsymbol{t}_{AB}$ — first rotate B-coordinates into the axes of A, then add the origin offset

Inverse: $\boldsymbol{p}_B = R_{AB}^T(\boldsymbol{p}_A - \boldsymbol{t}_{AB})$, i.e. $R_{BA} = R_{AB}^T$ and $\boldsymbol{t}_{BA} = -R_{AB}^T\,\boldsymbol{t}_{AB}$

A rigid transform preserves distances and angles (6 degrees of freedom: 3 rotation + 3 translation) — exactly what "moving a camera" can do



Q: Is $\boldsymbol{t}_{BA} = -\boldsymbol{t}_{AB}$?

Only if $R_{AB} = I$. In general $\boldsymbol{t}_{BA} = -R_{AB}^T\boldsymbol{t}_{AB}$: it is the same displacement (from O_B back to O_A) but measured along the axes of B, so it must be rotated into the coordinates of B. Keep this in mind for the camera: its translation vector $\boldsymbol{t}$ is not minus its position.

Homogeneous coordinates: one 4×4 matrix

Q: Can $\boldsymbol{p}_A = R_{AB}\,\boldsymbol{p}_B + \boldsymbol{t}_{AB}$ be written as a single matrix product $M\boldsymbol{p}_B$?

No. A linear map sends 0 to 0, but $\boldsymbol{p}_B = 0$ (the origin of B) must map to $\boldsymbol{t}_{AB} \neq 0$. The map is **affine**, not linear.

- The fix: append a constant 1 — the **homogeneous coordinates** $\tilde{\boldsymbol{p}} = \begin{bmatrix}\boldsymbol{p}\\ 1\end{bmatrix} \in \mathbb{R}^4$. Rotation and translation become one 4×4 matrix:
  - $\tilde{\boldsymbol{p}}_A = T_{AB}\,\tilde{\boldsymbol{p}}_B, \quad T_{AB} = \begin{bmatrix}R_{AB} & \boldsymbol{t}_{AB}\\ 0^T & 1\end{bmatrix} \in \mathrm{SE}(3)$
- Check the last row: $0 \cdot x_B + 0 \cdot y_B + 0 \cdot z_B + 1 \cdot 1 = 1$; rows 1–3 give $R_{AB}\boldsymbol{p}_B + \boldsymbol{t}_{AB}$
- Inverse and chaining are now matrix algebra: $T_{BA} = T_{AB}^{-1} = \begin{bmatrix}R_{AB}^T & -R_{AB}^T\boldsymbol{t}_{AB}\\ 0^T & 1\end{bmatrix}, \quad T_{AC} = T_{AB}\,T_{BC}$
- Example, map → vehicle → camera: $\tilde{\boldsymbol{p}}_{map} = T_{map,veh}\,T_{veh,cam}\,\tilde{\boldsymbol{p}}_{cam}$ — the same trick as the bias column of a neural-network layer