What is a Convolution

- Operation that involves adding & multiplying
- Input image $*$ Filter (kernel) = Output Image

  $\uparrow$ convolve operation

- useful for performing edge detection

$$G_x = \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix} \qquad G_y = \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

- convolution is like image modifier
  - └ feature transformation

- Different filters will result in different outputs
  - └ Gaussian filter = Blurs image
  - Edge detection filter = Edge detection

Convolution Operation

$$\begin{matrix} 0 & 10 & 10 & 0 \\ 20 & 30 & 30 & 20 \\ 10 & 20 & 20 & 10 \\ 0 & 5 & 5 & 0 \end{matrix} \quad * \quad \begin{matrix} 1 & 0 \\ 0 & 2 \end{matrix}$$

1) $(0 \times 1) + (10 \times 0) + (20 \times 0) + (30 \times 2) = 60$

2) $(10 \times 1) + (10 \times 0) + (30 \times 0) + (30 \times 2) = 70$

3) $(10 \times 1) + (0 \times 0) + (30 \times 0) + (20 \times 2) = 50$

Output matrix

$$\begin{matrix} 60 & 70 & 50 \\ 60 & 70 & 50 \\ 20 & 30 & 20 \end{matrix}$$

* convolution is commutative

$A * W = W * A$

Convolution output

if input length = N   then output length $= (N - K) + 1$

kernel length = K

* Images are typically not square ← some neural nets use square for convenience.

* Filters are almost always square

Convolution Equation

$$(A * W)_{ij} = \sum_{i'=0}^{K-1} \sum_{j'=0}^{K-1} \underbrace{A(i+i', j+j')}_{\text{Input}} \underbrace{W(i', j')}_{\text{filter}}$$

i': kernel row, j': kernel col

* Filters in CNNs are learned using gradient descent

└ 1. Filters assigned with random values
2. Input convolve filters to produce feature maps
3. Loss calculated & compared with labels
4. Backprop to calculate gradient
5. Filters updated using gradient descent

Padding

- Pad input array with 0s to make convolved output same size as input

"valid" : output $= N - K + 1$ (filter can only touch valid input)

"same" : output $= N$ (output same size as input, done with padding)

"full" : output $= N + K - 1$ (ensure non-zero output) (Not typically used)

Dot Product & correlation

Angle b/w a & b

- $a \cdot b = \sum_{i=1}^{N} a_i b_i = |a||b| \cos \theta_{ab}$
- Also called cosine similarity / cosine distance
- $\cos(0) = 1$ ↗ same dirn

  $\cos(90) = 0$ orthogonal

  $\cos(180) = -1$ ⤢ opp dirn.

$$\cos \theta_{ab} = \frac{a \cdot b}{|a|\,|b|}$$: how similar are a & b.

max = 1

min = −1

0 : orthogonal

magnitude of vector $|a| = \sqrt{\sum_{i=1}^{N} a_i^2}$

$$\text{Pearson correlation} = \frac{\sum_{i=1}^{N} (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{N} (a_i - \bar{a})^2} \sqrt{\sum_{i=1}^{N} (b_i - \bar{b})^2}}$$

← very similar to dot product, just subtract mean.

Dot product is like a correlation measure of 2 vectors.

└ related to filter in CNN finding correlated features

Matrix operation & convolution

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \quad W = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix} \quad a * W = \begin{bmatrix} a_1 W_1 + a_2 W_2 \\ a_2 W_1 + a_3 W_2 \\ a_3 W_1 + a_4 W_2 \end{bmatrix} = \begin{pmatrix} W_1 & W_2 & 0 & 0 \\ 0 & W_1 & W_2 & 0 \\ 0 & 0 & W_1 & W_2 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

↑ Repeating filters simulate convolutions

(Takes up space)

Consider opposite case:

- using convolutions to replace matrix multiplications.
- main idea behind parameter / weight sharing

Activation $= W^T x$

└ instead of full weight matrix, use same 2 weights over.

Translation invariance

- convolution operation recognize patterns / features regardless of location within input space
- filter slides over input data, ensuring same filter applied across entire input

CNN Input vs output

- For colour Images, shape = H × W × 3
- Kernel size = K × K
- output = H × W for same size operation.

* Discrepancy : Input = 3D, output = 2D. Need same shape to stack convolution layers.

Soln : For each feature output, stack it.

eg: Nose filter = H × W
Eye filter = H × W
} stacked = H × W × 2

∴ output = H × W × No. of filters

CNN in Deep Neural Network

$B = A * w$, $A = H \times W \times C_1$ ← Colour, typically 3

filter $= C_1 \times K \times K \times C_2$ ← No. of features per convolution layer.

$H \times W \times C_2$, Assuming same shape convolution

No. of features in final convolution layer.

* 3^{rd} axis no longer colour but known as feature maps. or channels

Conv layer: $\sigma(\underbrace{W}_{H \times W \times C} * x + \underset{\text{size } C}{b})$: Different shape but rules of broadcasting in Numpy allow such operation

Dense layer: $\sigma(\underbrace{W^T \cdot x}_{\text{size } M} + \underset{\text{size } M}{b})$

Cost savings (conv vs matrix multiplication)

Convolution:

Input : $32 \times 32 \times 3$

Kernel: $3 \times 5 \times 5 \times 64$ = 4800 params.

output: $28 \times 28 \times 64$

$(32 - 5 + 1)$

Matrix Multiplication: (flatten)

Input = $32 \times 32 \times 3$ = 3072

output: $28 \times 28 \times 64 = 50176$

Weight: 3072×50176 = 154 000 000 params.

↳ 32000 x more than conv operation.

Suffers from translation variance.

Need to learn every possible location