CNN

August 11, 2025 12:16 PM

Convolution Neural Networks

- Grid like data
  e.g. 2D images, 1D time series

- Instead of fully connected layers (to all neurons)
  Focus on small regions → Features
  Using filters (sliding over the data)

- Helps focus on immediate neighbors instead of noise from all the features

- Same filter across all input so kind of same weight

- Creating automatic features (like maybe eyes, nose etc. in an image)

CNN

e.g. A sample input → $I = \begin{bmatrix} 10 & 11 & 3 & 8 \\ 2 & 15 & 12 & 19 \\ 7 & 1 & 18 & 11 \\ 13 & 4 & 3 & 14 \end{bmatrix}$ $I = H \times W \times C$

Each Value corresponds to a pixel (in case of an image)

$H \times W \times C$

$4 \times 4 \times 1$

Height Width channels (e.g. R,G,B)

0 Usually we do a bit of preprocessing

- Normalize the matrix - so values b/w 0 & 1 - So weights are not too funky & we don't run into problems training
- Add a padding - Adding empty rows/columns to make sure the output doesn't shrink too much
  - Usually we add it, if we want our 'features' down the line to be of specific size
  Some networks add padding in every layer to preserve resolution

1.1 Convolution

For simplicity we will walk through our sample without normalization OR padding for now

What's Happening

$$I_{H \times W \times C} \xrightarrow[\text{with } K_{(f_h, f_w \times C)} \text{ filters}]{\text{Convolution Layer}} S_{H_{out}, W_{out}, C}$$

Mathematically

$$\underbrace{S}_{\text{Output of Convolution layer}}\underbrace{(i,j)}_{\text{element on } i^{th} \text{ row } \& j^{th} \text{ column}} = \overbrace{\sum_{m=0}^{f_h-1} \sum_{n=0}^{f_w-1} \sum_{c=0}^{C-1}}^{\text{Nested loops for each position in filter (every row, col, channel)}} \underbrace{I}_{\text{Input Matrix } H \times W \times C \ (i,j,c)}(i\underbrace{s}_{\text{Stride (Sliding window)}}+m, js+n, c) \overbrace{\cdot}^{\text{Multiplication}} \underbrace{K}_{\text{Filter } (f_h \cdot f_w, C)}(m,n,c)$$

{ (m),(n),(c) under row, col, channel }

output of Convolution layer   element on i^{th} row & j^{th} column   Input Matrix $H \times W \times C$ $(i),(j),(c)$ {index → row, col., Channel}   (sliding window) $(f_h \cdot f_w, C)$ $(m),(n),(c)$ {index row, col, channel}

Inside the layer

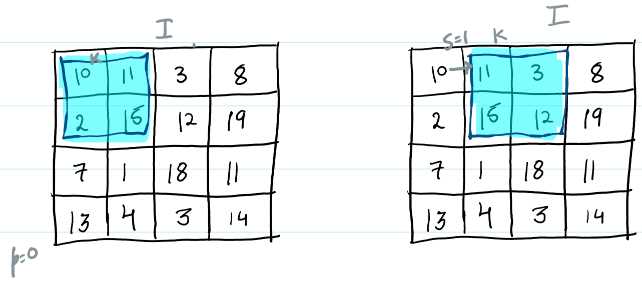
Let $K = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ $2 \times 2 \times 1$ $f_w * f_h \times c$

$s = 1$ Stride (Sliding window)

1 means window moves by 1 index

$p = 0$ Nopadding (Downsampling)

How it works



10	11	3	8
2	15	12	19
7	1	18	11
13	4	3	14

10	11	3	8
2	15	12	19
7	1	18	11
13	4	3	14

- - - - -

9 times

$$H_{out} = \frac{H - f_h + 2p}{s} + 1 = \frac{4-2}{1} + 1 = 3$$

$$W_{out} = \frac{W - f_h + 2p}{s} + 1 = \frac{4-2}{1} + 1 = 3$$

$$S(0,0) = \begin{bmatrix} 10 & 11 \\ 2 & 15 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = 10 - 15 \Rightarrow -5$$

$$= \begin{bmatrix} I(0,0) & I(0,1) \\ I(1,0) & I(1,1) \end{bmatrix} \begin{bmatrix} K(0,0) & K(0,1) \\ K(1,0) & K(1,1) \end{bmatrix}$$

$$S(0,1) = \begin{bmatrix} 11 & 3 \\ 15 & 12 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = 11 - 12 \Rightarrow -1$$

$$= \begin{bmatrix} I(0,1) & I(0,2) \\ I(1,1) & I(1,2) \end{bmatrix} \begin{bmatrix} K(0,0) & K(0,1) \\ K(1,0) & K(1,1) \end{bmatrix}$$

$$S(i,j) = \sum_m \sum_n I(i+m, j+n) \; K(m,n)$$

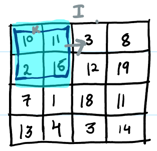
Overall

$$I_{H \times W \times C} \xrightarrow[\text{with } K_{(f_h, f_w \times C)} \text{ filters}]{\text{Convolution Layer}} S_{H_{out}, W_{out}, C}$$

10	11	3	8
2	15	12	19
7	1	18	11
13	4	3	14

$I_{(4 \times 4 \times 1)}$

$\longrightarrow$



$K(2,2,1), p=0, s=1$

-5	-1	-16
1	-3	1
3	-2	4

$S(3,3,1)$

(1.2) Activation

This is pretty standard (happens in other NN)

Here we applied on the convolution output (S) above

There are various functions that can be used

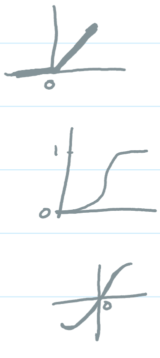
Here we'll use ReLU, one of the popular activation function for introducing some sparsity

e.g

ReLU $f(x) = \max(0, x)$

Sigmoid $f(x) = \dfrac{1}{1+e^{-x}}$

tanh $f(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$

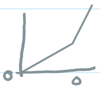


Now we'll use ReLU, one of the popular activation function for [illegible]

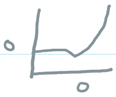
tanh $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Softmax $f(x) = \frac{e^x}{\sum_j e^{x_j}}$

LeakyReLU $f(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$

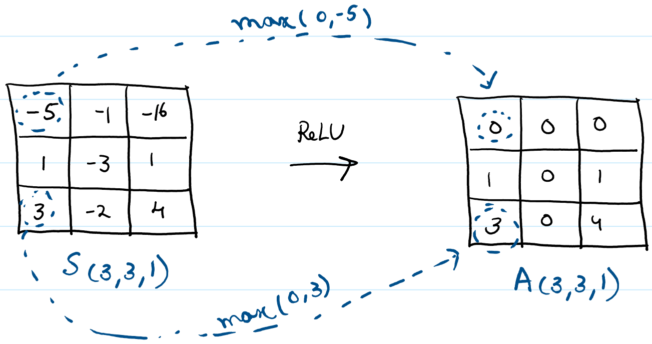


Swish $f(x) = \frac{x}{1 + e^{-x}}$



$$\underbrace{A(i,j)}_{\substack{\text{The output element} \\ \text{in } i^{th} \text{ row} \\ \text{\& } j^{th} \text{ column}}} = \underbrace{ReLU}_{\substack{\max(0,x) \\ \text{Only keep +ve} \\ \text{values} \\ \text{Helps with potential gradient} \\ \text{optimizations downstream}}}(\underbrace{S(i,j)}_{\substack{\text{Convolution} \\ \text{output}}})$$

$$S_{H_{out} W_{out} C} \xrightarrow[\text{funct}]{\text{Activation}} A_{H_{out} W_{out} C}$$



$$S(3,3,1) = \begin{bmatrix} -5 & -1 & -16 \\ 1 & -3 & 1 \\ 3 & -2 & 4 \end{bmatrix} \xrightarrow{ReLU} A(3,3,1) = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 3 & 0 & 4 \end{bmatrix}$$

(2) Pooling

Point is to reduce the size further (to control overfitting & improve computation)

Coalescing different features - a sliding window over the output of convolution + Activation

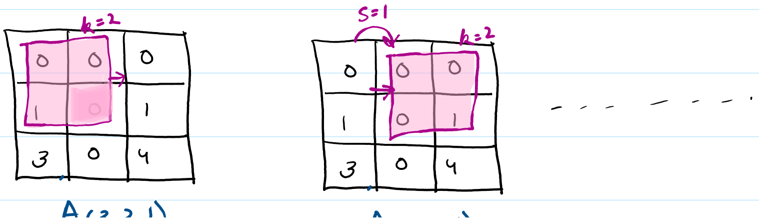
e.g Max pooling - max of the window (keep only the strongest feature)

Avg. pooling - avg. of the window (combines features into a smaller more smoother matrix)

In this case, we'll use max pooling

$$\underbrace{P(i,j)}_{\substack{\text{Output element} \\ \text{in } i^{th} \text{ row \&} \\ j^{th} \text{ column}}} = \max_{\underbrace{0<m,n<k}_{\substack{\text{Pooling} \\ \text{window}}}} \underbrace{A}_{\substack{\text{Activation} \\ \text{output}}}(i.\underbrace{s}_{\text{stride}}+m, j.s+m)$$

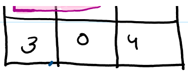
e.g $\underbrace{k=2}_{\text{Pooling window}}$ $\underbrace{s=1}_{\text{stride}}$ Usually no padding



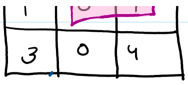
4 times

$$H_{out} = \frac{H - k_h + 2p}{s} + 1 = \frac{3-2+0}{1} + 1 = 2$$

$$W_{out} = W - k_w + 2p + 1 = \frac{3-2+0}{1} + 1 = 2$$



A(3,3,1)



A(3,3,1)

$$H_{out} = \frac{H - k_h + 2p}{s} + 1 = \frac{3-2+0}{1} + 1 = 2$$

$$W_{out} = \frac{W - k_w + 2p}{s} + 1 = \frac{3-2+0}{1} + 1 = 2$$

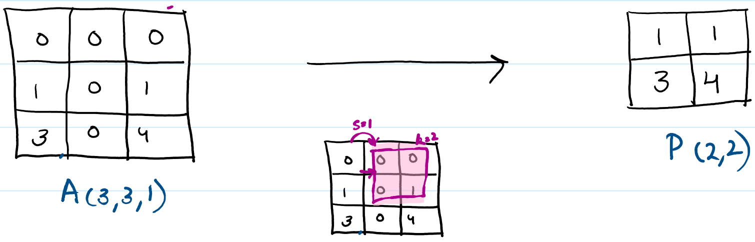
$P_{2\times2}$

$$P(0,0) = \max\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}_{2\times2} = 1 \qquad P(0,1) = \max\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}_{2\times2} = 1$$

$$P(0,0) = \max\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \qquad P(0,1) = \max\begin{pmatrix} A_{01} & A_{02} \\ A_{11} & A_{12} \end{pmatrix}$$

$$P(i,j) = \max_{0<m,n<k} A(i+m, j+n)$$

$$A(H_{out}, W_{out}, C) \xrightarrow{\text{Pooling layer}} P(H_{out}', W_{out}', C)$$



A(3,3,1)

k=2 , s=1

P(2,2)

③ Flatten

Just as it sounds

Takes this condensed feature matrix (after convolution, activation, pooling) & puts into 1d vector

$$P_{(H,W,C)} \xrightarrow{\text{Flattening layer}} F(1, H.W.C)$$

$$\begin{bmatrix} 1 & 1 \\ 3 & 4 \end{bmatrix} \longrightarrow [1, 1, 3, 4]$$

④ Fully Connected

Again a standard layer (last layer) followed by an activation function for final output

4 Fully Connected

Again a standard layer (last layer) followed by an activation function for final output

e.g a softmax function

$$y = \underbrace{w}_{\text{Learnt params}} x + \underbrace{b}$$

$$\Rightarrow \text{Final Output} = \underbrace{\sigma}(\overbrace{wx+b})$$

$$y_j = \sigma\left(\sum_{i}^{N} w_{ij}\, x_i + b_j\right)$$

In our case

$$\text{Final output} = \sigma\left(W \begin{bmatrix} 1 \\ 1 \\ 3 \\ 4 \end{bmatrix} + b\right)$$

If our fully connected layer has lets say 2 neurons, w will be 4×2

& b " " 2×1

e.g. for a binary classification

here output of each neuron describes the probab. of that class

$$y = \sigma\left(\underset{2\times 4}{\begin{bmatrix} w_{01} & w_{11} \\ \vdots & \vdots \\ w_{03} & w_{13} \end{bmatrix}} \underset{4\times 1}{\begin{bmatrix} 1 \\ 1 \\ 3 \\ 4 \end{bmatrix}} + \underset{2\times 1}{[\,b_1 \;\; b_2\,]}\right)$$

$$= \sigma\left(\underset{2\times 1}{\begin{bmatrix} w_1 x_1 + b_1 \\ w_2 x_2 + b_2 \end{bmatrix}}\right) = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \begin{matrix} \rightarrow \text{Neuron1} \\ \rightarrow \text{Neoron2} \end{matrix}$$