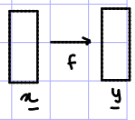
9. CONVOLUTIONAL NEURAL NETWORK

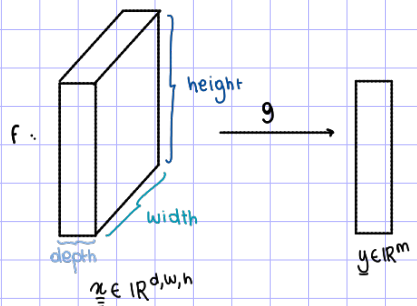
9.1 Increase dimensional input of FFNN

In feed forward neural network we can map $\mathbb{R}^n \longrightarrow \mathbb{R}^m$ with the non-linear function $f: \mathbb{R}^n \to \mathbb{R}^m$, $x \mapsto y = f(x)$



How we can generalite this kind of NN so that we have an input of bigger dimension, i.e. 3 dimensional input?

Convolutional neural network CNN. This NN is tipically used to process image



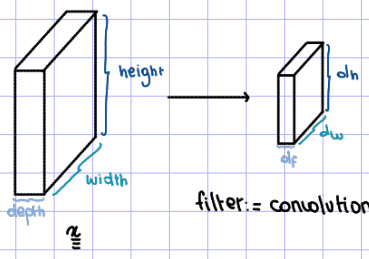
We want to add a new function so that the 3d input can transform, via g, in a long array so that can be the input of the FFNN to find the output (the result)

Problem. How to pass from a 3dimensional array to 1dimensional?

Solutions. Ingredients of g ⟶
- a. Convolutional layer
- b. Pooling layer
- c. Normalitation layer

NB Convolutional NN extend the idea of FFNN but mantaining the structure of the original data

9.2 Convolutional layer ⟶ is a filter



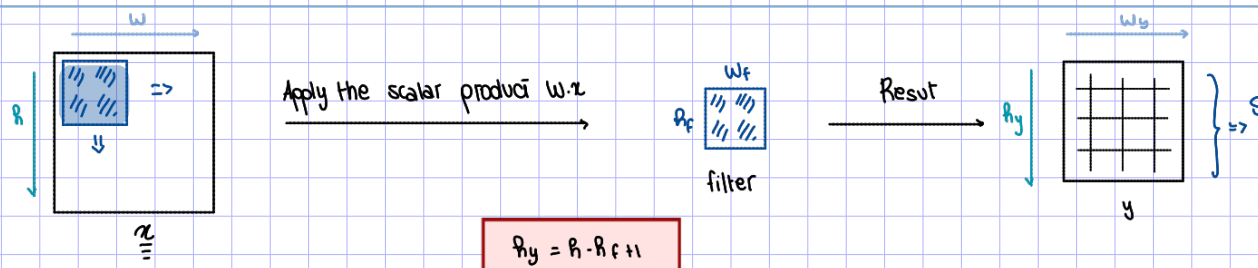
filter:= convolutional layer

Consider $d_f = d$, $\underline{\underline{W}}$:= the weight $\in \mathbb{R}^{d_f \times w_f \times h_f}$ and we want to build a filter so that

$$\underline{y} = \underline{\underline{W}} \cdot \underline{x} + w_0$$

, where w_0 is the bias and W the associated matrix

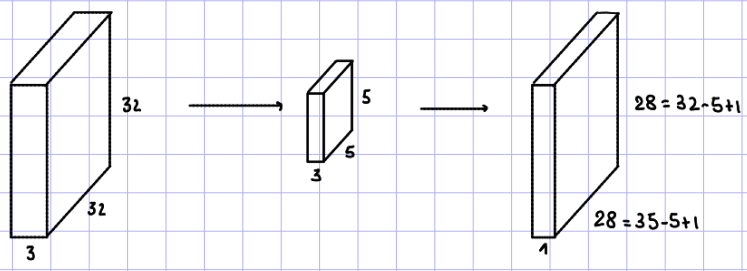
The idea is that the filter can move left-right & up-down on the image
↳ We shift the convolutional filter around $\underline{x}$ doing convolution $y = Wx + w_0$



In which, the final dimension are

$$h_y = h - h_f + 1$$
$$w_y = w - w_f + 1$$
$$d_y = 1$$

Example.

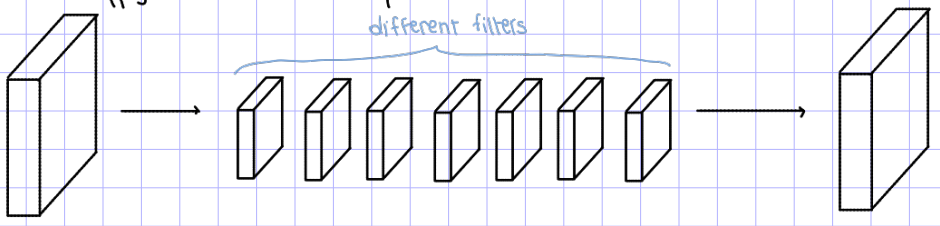


NB In general, we can use more than one convolutional filter, obtaining different filters inputs

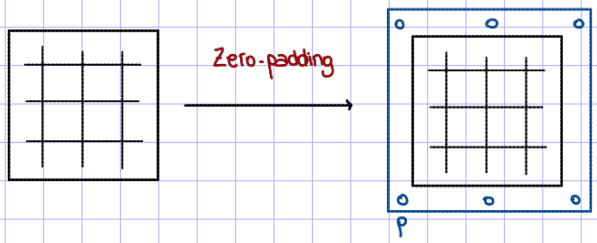
Observation. After the convolutional layer you have reduced the dimension of the image, this caused some issue.

↓ Let's now try to solve them

a. **Channel dimension.** The depht is $d_y = 1$. How we can deal with this?
We can apply different filter in parallel in which I can choose the dimension d called channel to use for all the layer

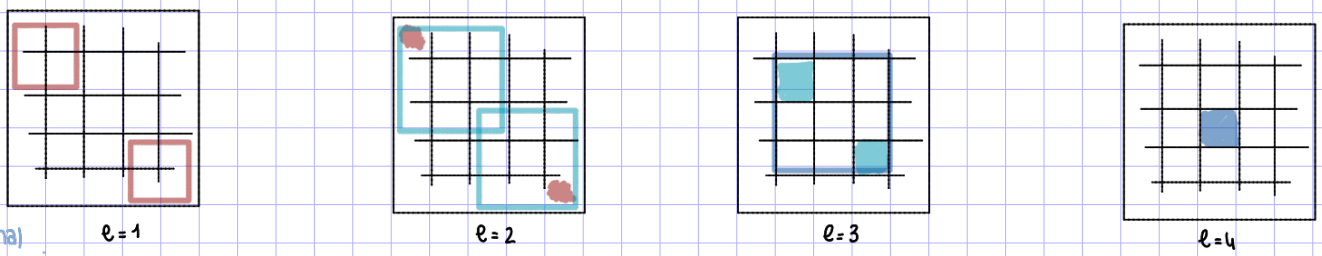


b. **Padding.** If we apply K convolution one after the other it happens that the dimension of $\underline{x}$ goes to zero rapidly (Shrink)
How we can deal with this? We generalite the convolutional layer with the operation of padding, to avoid constraints on the number of layer in the NN



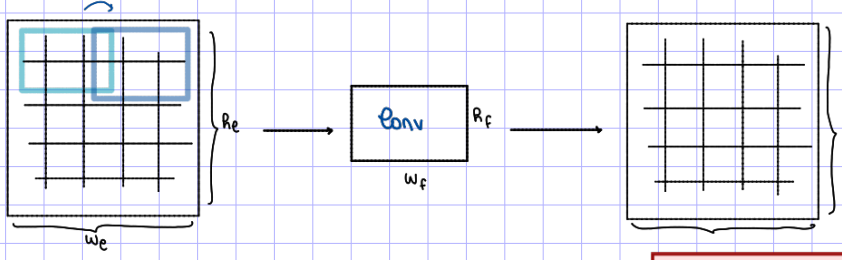
$$W_y = W - W_f + 1 + 2p$$
$$R_y = R - R_f + 1 + 2p$$

c. **Receptive fields.** With convolutional layer the next data is influenced by a particular region of the previous data and by a particular region of the input
Let's now imagine to apply the filter to my input.
How many layer we need to see all the information of the input?



Convolutional layer:
In this simple example, we need 4 layer, think about bigger one!

↓ We add the last parameter to move it "faster" in the image: **Stride**



$$R_{e+1} = \frac{(R_e - R_f + 2p)}{s} + 1$$

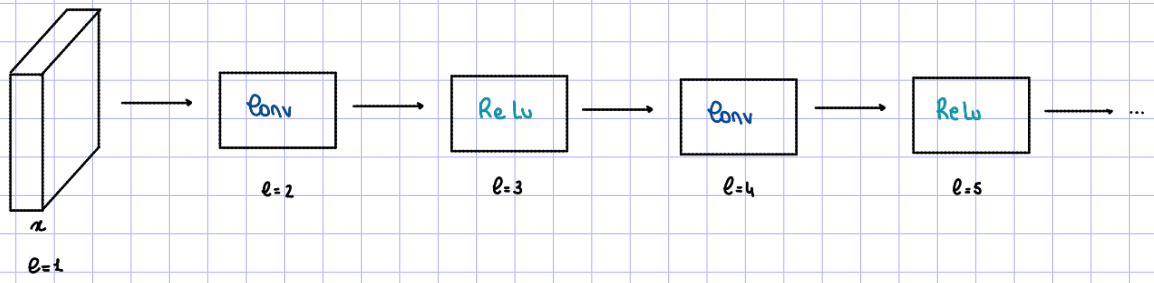
$$W_{e+1} = \frac{(W_e - W_f + 2p)}{s} + 1$$

Example. Consider
Conv of W_f h_f
- $\underline{x}^e$: $3 \times 32 \times 32$ (d W R) is a convolutional filter where W: $10 \times 3 \times 5 \times 5$ with $s=1$, $p=2$ ⟶ $W = 760$ unknows to minimize
- $\underline{x}^{e+1}$: $10 \times 32 \times 32$

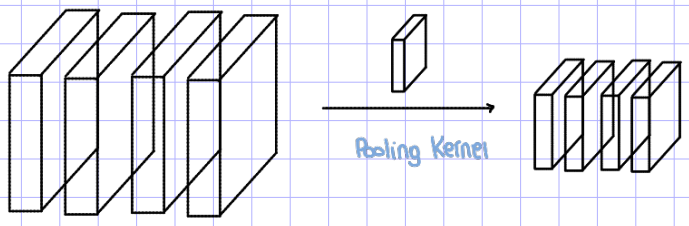
The number of operations to do are $\simeq$ 1 milion to perform on each convolutional layer ⟶ done in parallel via GPU

Definitions. • **Learnable parameters**: weights inside the filter for each channel w_{ij} ⟶ learn by the minimization problem
• **Hyperparameters**: not learnable parameters ⟶ fixed by the user before the learning phase, i.e. padding, stride

Till now, we have NOT used a non-linearity ...
... Hence, we add the activation function, i.e. ReLu to add non-linearity



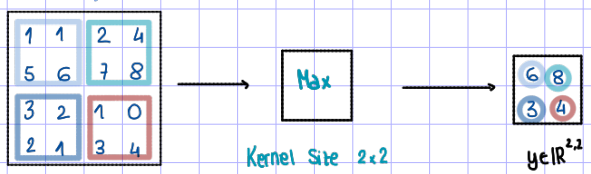
9.3 Pooling layers ⟶ the idea is to STRINCK the input



The **pooling Kernels** are just function i.e.
- Avg pool $K=2, S=2$
- Max pool $K=2, S=2$
- AlexNet pool $K=3, S=2$

↓
are used to downsize the sample without any learnable parameters

Let's make an example to better understand
Max pooling



$x \in \mathbb{R}^{4,4}$
Stride = 2
$y \in \mathbb{R}^{2,2}$

The pooling, here, just select one element, i.e. the maximum
↓
we apply non linear function use just to reduce the dimension
NB It's NOT mandatory that the pooling layer is NON-linear

↓

With 9.2 I enlarge the numer of channel, mantaining w_f and b_f applying convolutional layer, activation σ
After with 9.3 I modify the w_f and b_f in which I enlarge the number of channel in output
Till we can transform the dimension in an array so that we can apply FFNN

9.4 Normalitation ⟶ Important to find the right solution, used especially to improve the speed of the computation
Consider to minimite $J(z,y) = \|z-\hat{z}\|^2 + \|y-\hat{y}\|^2$ ⟶ if y is bigger, you focus on it to improve
But we do not want to focus only in one variable ⇒ we normalite so that z & y has the same order of magnitude

$$\hat{z} = \frac{z - E(z)}{\sqrt{Var(z)}}$$

so that we have $E(z)=0$ & $Var(z)=1$

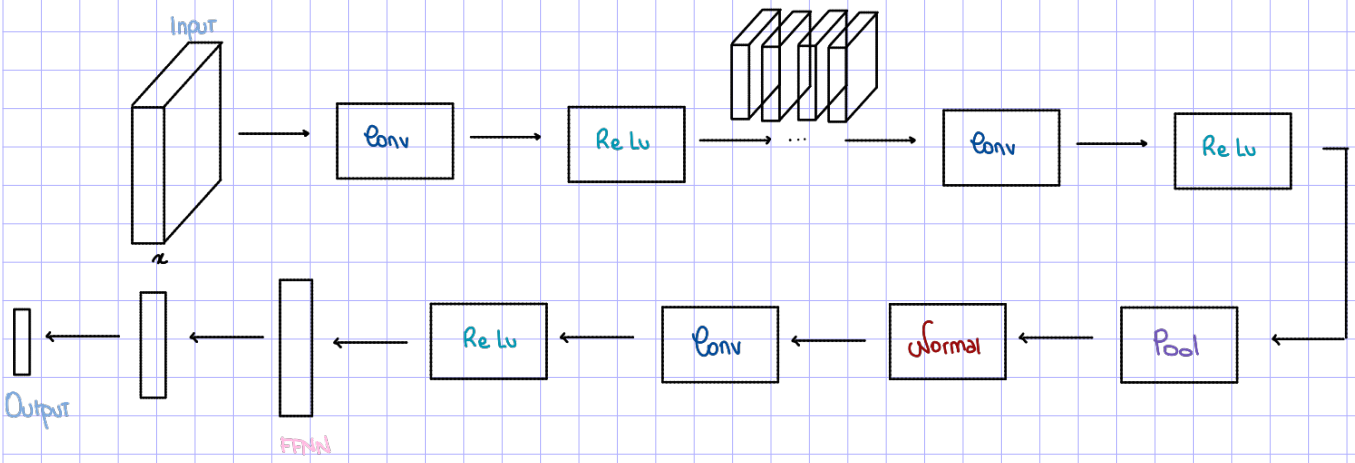
$E(z) = \frac{1}{N}\sum_i^N z$ $\quad$ $Var(z) = \frac{1}{N-1}\sum_i^N (z-E(z))^2$

Important: the mean & variance are computed only offline^{-1} during the training phase, in the validation phase they're fixed
The final oupur, has to belong to the originale scale, hence

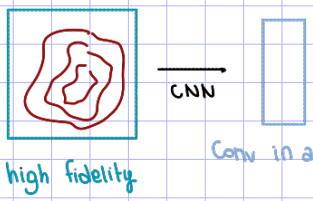
$$y = \gamma\hat{z} + \beta$$ with γ, β learnable parameters learn in the training phase

↓
capacity of the NN to get back z

9.5 Structure of a CNN



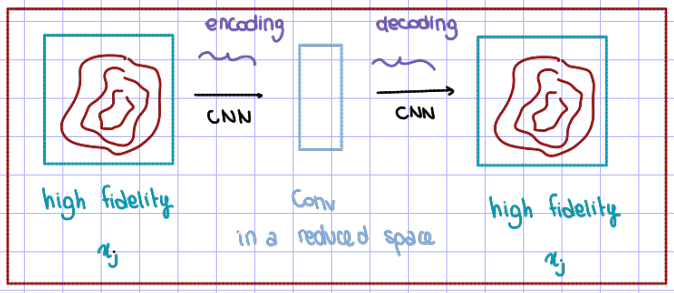
During this process, I do a sort of **convolution of the volume** during all this step
we start from the dimension of x_i and at the end we reach a smaller dimension
⟶ we can exploit this idea in PDE to reduce the image of the domain of PDE, i.e.



This process is called **encoding**, but we can see also the opposite one

↓

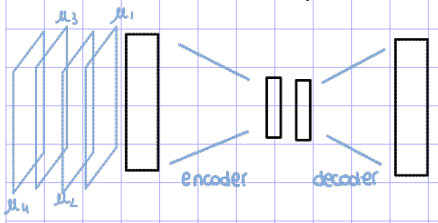
How we can encode the data in a supervised way?
↳ We add another convolutional neural network to enlarg again the dimension



⟶ This is an **AUTOENCODER NEURAL NETWORK**
After that we can use the decoders to build the final solution of the PDE

↓

The idea is that you want **exact** to reproduce the image. The scheme that you have in mind is this one



1. Reduce the space ⟶ not-linear step — encoder
2. Enlarg again the space — decoder

What happen if I want to work with parametric stuff? We want to exploit the idea of NN, but for solving our purpose. Hence we introduce another NN that takes the parameters μ and give in result the reduce space

- Offline you trane the net for each different parameter μ and build the encoder
- Online you exploit the decoder phase to obtain the result in the huge initial space

↓

The idea is to move from pixel to **graph neural network** to increase the accuracy of the encoder
Hence in the μ_i layer there are all the nodes and edge of the graph

NB We do not have any kind of idea of the error that we're doing during this step