5. Neural Network and CNN

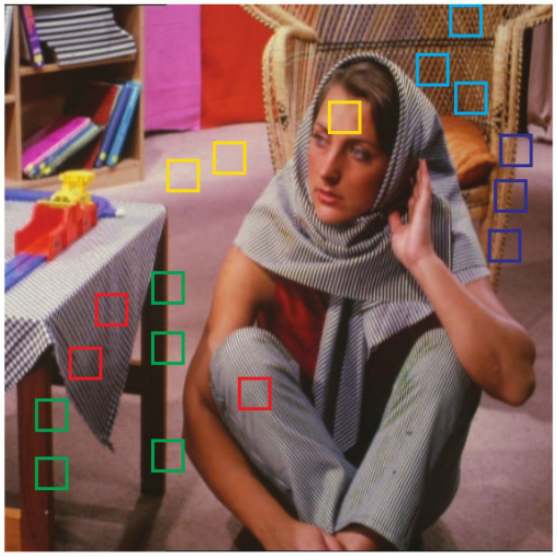
Need of priors

The idea here is to create models that are flexible and general enough to handle a wide range of tasks. However, to improve their performance on specific problems, it helps to use "priors" — assumptions or knowledge about the data, like its structure or patterns. These priors make the model better suited to the task without being overly tied to it.

Structure as a Strong Priors

Key insight: Data often carries structural priors in terms of repeating patterns, compositionality, locality..

Self-Similarity



Data tend to be *self-similar* across the domain.

Translation Invariance

Translations do not change the image content.

Define the translation operator *T* along vector v as:

$$\mathcal{T}_v f(x) = f(x - v)$$

Therefore, it is desiderable to enforce translation invariance:

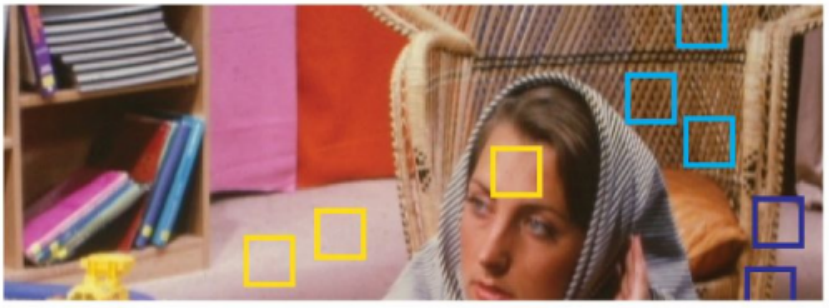
$$y(\mathcal{T}_v f) = y(f) \quad \forall f, \mathcal{T}_v$$

where y is a classification functional.

Hierarchy and Compositionality

Images also tend to have a hierchical structure that has to be compositional.

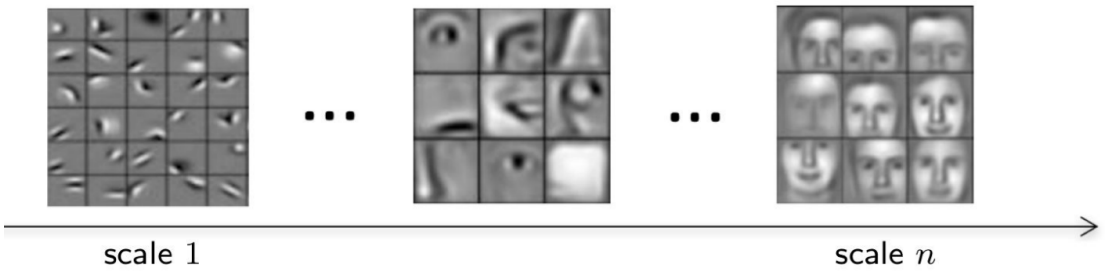
Translation invariance is desiderable across multiple scales:



We expect local features to be invariant to their location in the image:

$$z(\mathcal{T}_v p) = z(p) \quad \forall p, \mathcal{T}_v$$

where p are image patches of variable size



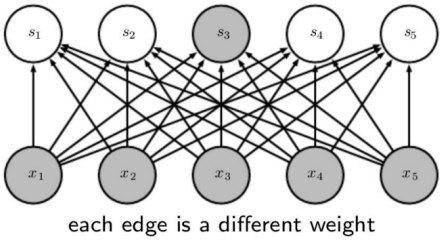
CNN is designed to process and analyze grid-like data (image, video..) Is used in computer video task

Convolutional Neural Network

Data is often composed of hierarchical, local, shift-invariant patterns → CNNs directly exploit this fact as a prior

The key innovation is the use of convolutional layers that automatically detect features

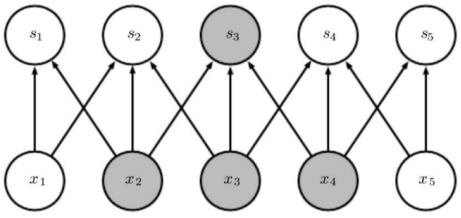
Sparse Interaction



each edge is a different weight

Fully connected layer → every neuron was fully dependent of other neurons

Since I know that my data has a hierarchical and compositional nature, I can create specific types of connections and leverage different kinds of relationships.



Convolutional layer → every neuron depends only on a window of 3 elements (neighbour) in the previous layer

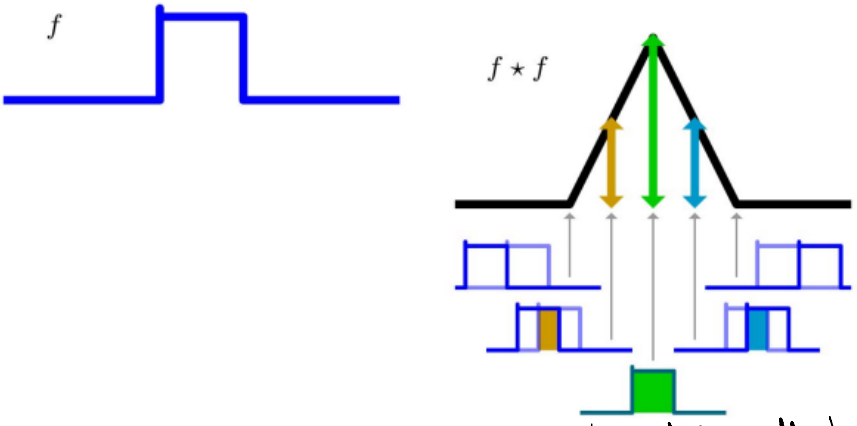
In convolutional layers (used in neural networks like CNNs), "weight sharing" is a key principle. Instead of assigning separate weights for every connection, the same set of weights is reused across the input. This drastically reduces the number of parameters, making the model more efficient and better at recognizing patterns (like edges or textures) regardless of their position in the input.

Convolution

Combine two functions to produce another function

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(t)g(x-t)dt$$

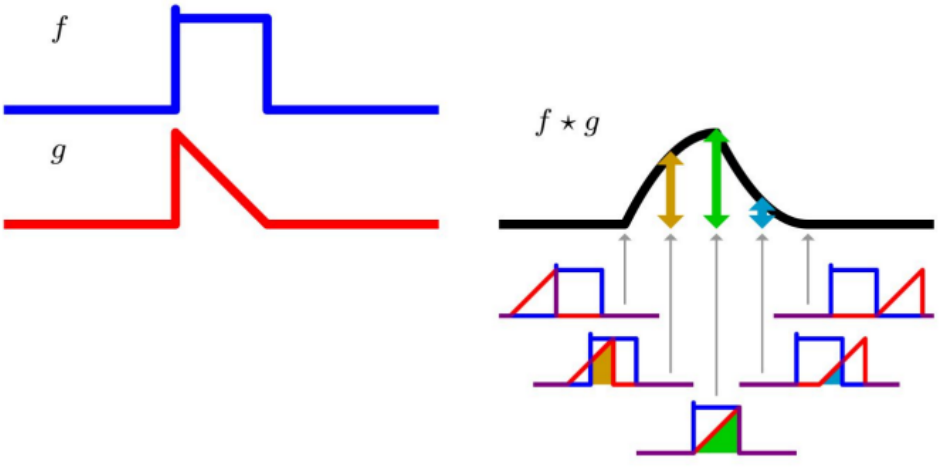


Convolution is applied to images using a small matrix called filter or kernel

The filter slides over the image and perform an element-wise multiplication followed by summing the results

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$\underbrace{(f \star g)(x)}_{\text{feature map}} = \int_{-\pi}^{\pi} f(t)\, \underbrace{g(x-t)}_{\text{kernel}}\, dt$$



Convolution: Properties

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$\underbrace{(f \star g)(x)}_{\text{feature map}} = \int_{-\pi}^{\pi} f(t)\, \underbrace{g(x-t)}_{\text{kernel}}\, dt$$

Convolution is commutative:

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(t) g(x-t) dt \overset{\substack{z:=x-t \\ \downarrow}}{=} \int_{-\pi}^{\pi} f(x-z) g(z) dz = (g \star f)(x)$$

Further, convolution is shift-equivariant:

$$f(x - x_0) \star g(x) = (f \star g)(x - x_0)$$

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f \star g)(x) = \int_{-\pi}^{\pi} f(t)\, \underbrace{g(x-t)}_{\text{kernel}}\, dt$$

It is easy to show that $\mathcal{G}$ is linear:

$$\begin{aligned} \mathcal{G}(\alpha f(x)) &= \alpha \int_{-\pi}^{\pi} f(t) g(x-t) dt = \alpha\, \mathcal{G}f(x) \\ \mathcal{G}(f+h)(x) &= \int_{-\pi}^{\pi} f(t) g(x-t) dt + \int_{-\pi}^{\pi} h(t) g(x-t) dt \\ &= \mathcal{G}f(x) + \mathcal{G}h(x) \end{aligned}$$

Translation equivariance can then be phrased as:

$$\mathcal{G}(\mathcal{T}f) = \mathcal{T}(\mathcal{G}f)$$

i.e., the convolution and translation operators commute.

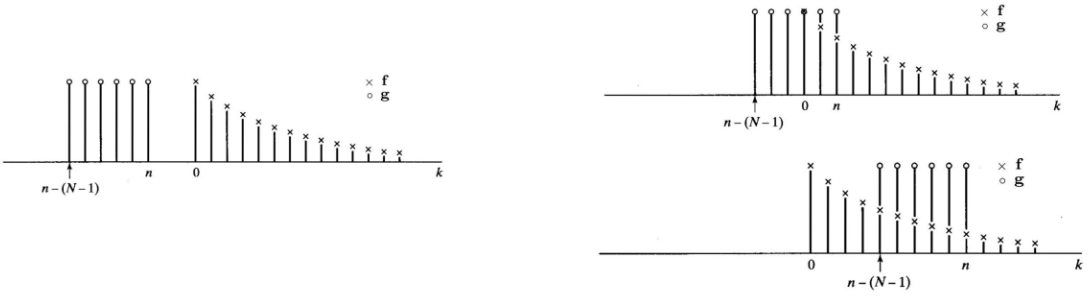
Discrete Convolution

We can use the logic of convolution also for discrete function (classification)

(images are discrete) → represent as a grid of pixels

In the discrete setting, we deal with vectors $\mathbf{f}, \mathbf{g}$.
We define the convolution sum:

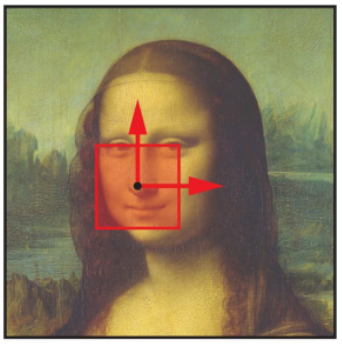
$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$



On 2D domains (e.g. RGB images $f : \mathbb{R}^2 \to \mathbb{R}^3$), for each channel:

$$(\mathbf{f} \star \mathbf{g})[m, n] = \sum_{k} \sum_{\ell} \mathbf{f}[k, \ell]\, \mathbf{g}[m-k, n-\ell]$$

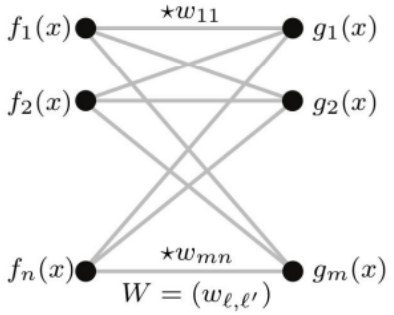
We get the classical interpretation in terms of a moving window:



Where are the parameters? The window (filter) contains numbers that are the parameters of our network that we need to optimize for.

Convolutional Neural Network

Main idea: Compose equivalent layers implemented via convolution.



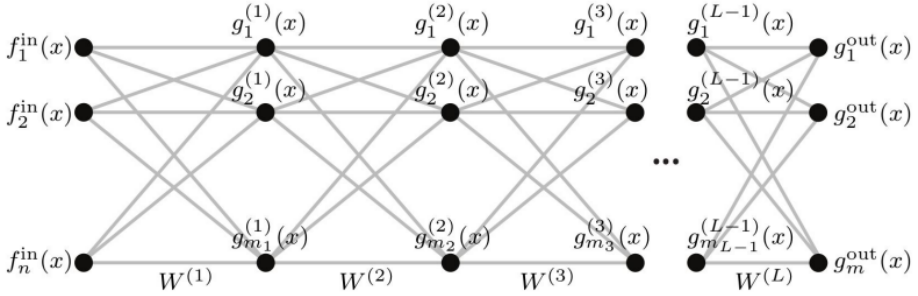
Single convolutional layer

We no longer use fully connected layers. Instead of having a large matrix **W** filled with weights, we now have a smaller matrix **w**, which represents the values within our filters.

Conv. layer $g_\ell(x) = \sigma\left(\sum_{\ell'=1}^{n} (f_{\ell'} \star w_{\ell,\ell'})(x)\right)$ $\ell = 1, \ldots, m$, $\ell' = 1, \ldots, n$

Activation, e.g. $\sigma(x) = \max\{x, 0\}$ rectified linear unit (ReLU)

Parameters filters W



CNN consisting of L convolutional layers

Conv. layer $g_\ell^{(k)}(x) = \sigma\left(\sum_{\ell'=1}^{m_{k-1}} (g_{\ell'}^{(k-1)} \star w_{\ell,\ell'}^{(k)})(x)\right)$ $\ell = 1, \ldots, m_k$, $\ell' = 1, \ldots, m_{k-1}$

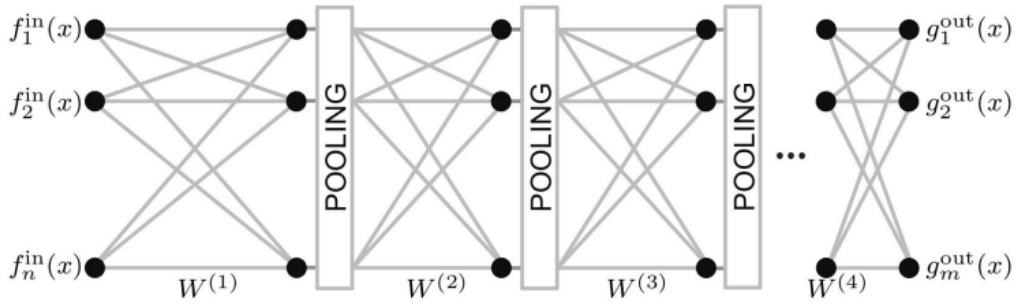
Activation, e.g. $\sigma(x) = \max\{x, 0\}$ rectified linear unit (ReLU)

Parameters filters of all layers $W^{(1)}, \ldots, W^{(L)}$

POOLING

Im CNN is a downsampling technique used to reduce the spatial dimension (height/width) of feature map while retaining the most important information

It helps to reduce the computational load, control overfitting and make the model more invariant to small translation, rotation or distortion in the input image



CNN consisting of L convolutional layers interleaved with pooling

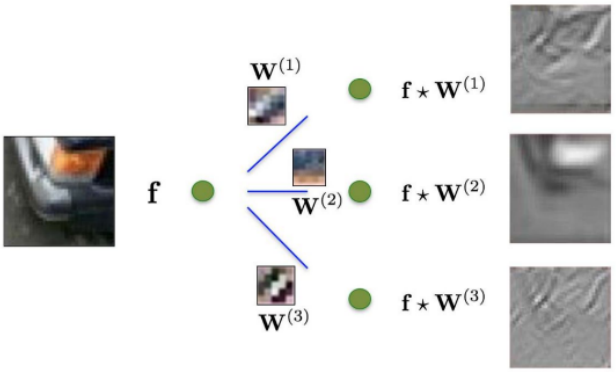
We not only have convolutional layer but we have also intermediate layers (pooling) that reduce the number of parameters thatr we needed because rescale the dimensionality of the input. We need it because we have to do a sort of zoom out effect that we were looking at then we were when you observe an image we have composionality

Conv. layer	$g_\ell^{(k)}(x) = \sigma\left(\sum_{\ell'=1}^{m_{k-1}} (g_{\ell'}^{(k-1)} \star w_{\ell,\ell'}^{(k)})(x)\right)$ $\ell = 1,\dots,m_k$, $\ell' = 1,\dots,m_{k-1}$
Activation, e.g.	$\sigma(x) = \max\{x, 0\}$ rectified linear unit (ReLU)
Parameters	filters of all layers $W^{(1)},\dots,W^{(L)}$
Pooling	$g_\ell^{(k)}(x) = \|g_\ell^{(k-1)}(x') : x' \in \mathcal{N}(x)\|_n$ $n = 1, 2,$ or ∞

Local Filters → refers to a restricted matrix that focuses on a local region

↗ The output of a CNN remains relatively stable

Shift-invariance is implemented via convolutional operators.

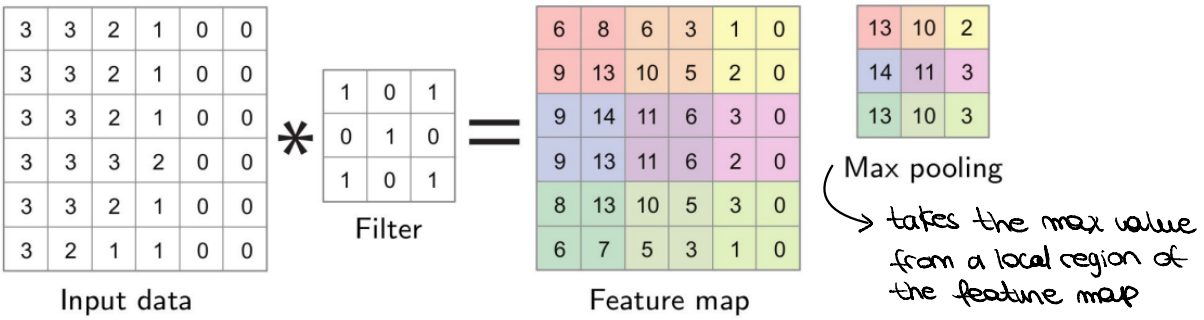


- $O(1)$ parameters per filter; huge gain compared to the MLP.
- Filter weights are applied across the entire image ⇒ weight sharing.

These are the filters that the network found out and needs to be optimize. We see these effects of highlighted react to these kind of filters.

Pooling

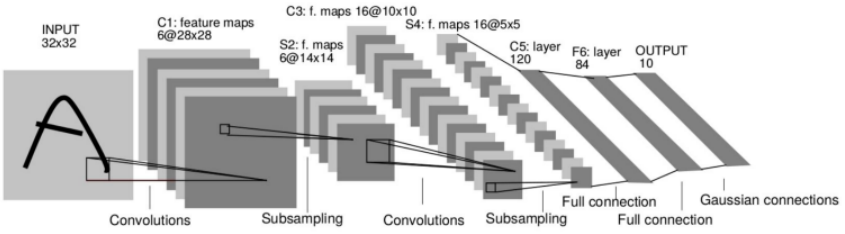
At deep layers, filters interact with larger portions of the input.



This allows to capture complicated non-local interactions via simple building blocks that only describe sparse interactions.

I move the filter over the input. With max pooling we are taking the maximum. The effect is reducing the feature map.

Key Properties of CNN



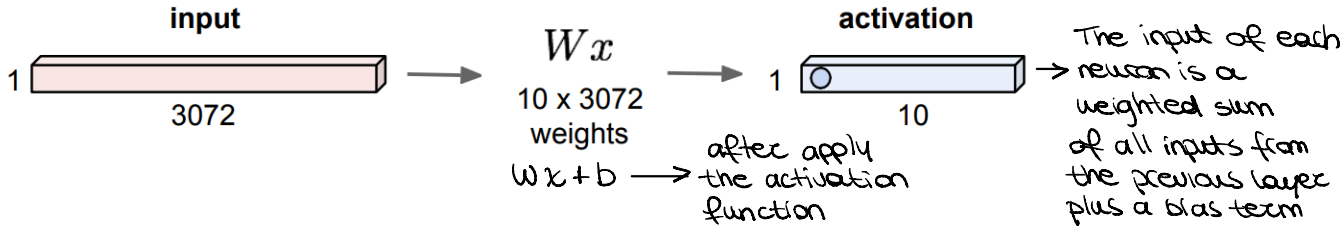
- Convolutional filters **(Translation equivariance)**
- Multiple layers **(Compositionality)**
- Filters localized in space **(Locality)**
- Weight sharing **(Self-similarity)** use same weights for all location in an input feature map
- $\mathcal{O}(1)$ parameters per filter (independent of input image size n)

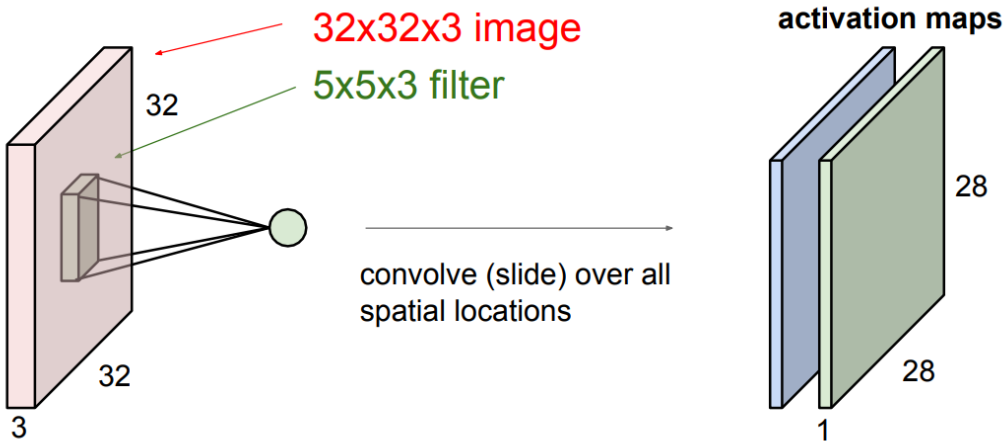
FULLY CONNECTED LAYER

Every neuron in a fully connected layer is connected to every neuron in the previous layer

32x32x3 image -> stretch to 3072 x 1

It applies a linear trasformation followed by a non linear activation function to compute its output

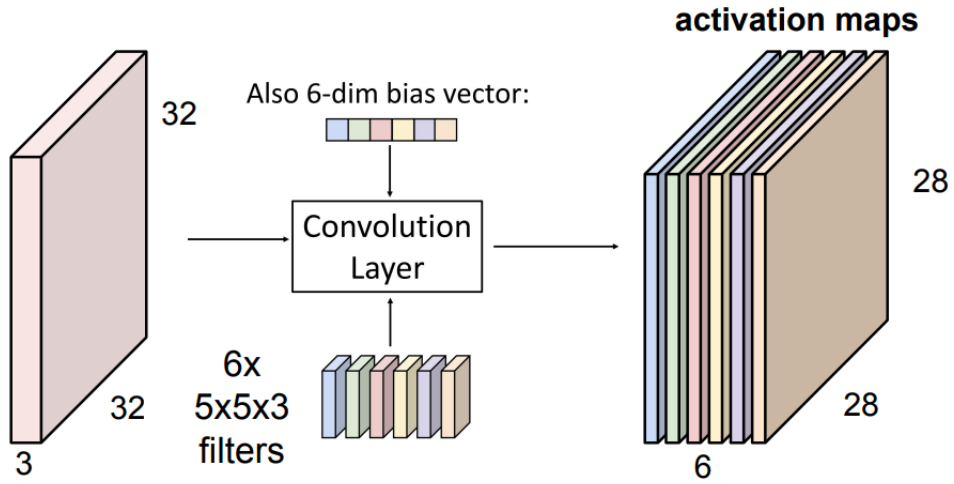




Fully connected layers take features extracted from earlier layers and combine them to make prediction

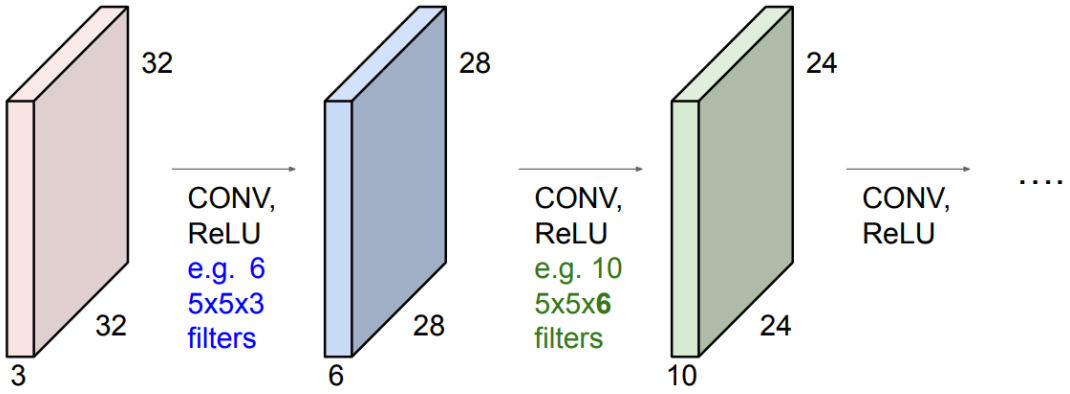
The content of the filter is multiplied with the content fo the image.

In the output everytime you get one number.



We stack these up tp get a new image of size 28×28×6

Fully connected layer transform high dimensional feature map into a lower dimensional representation
The last layer is usually a fully connected layer that produces the final output

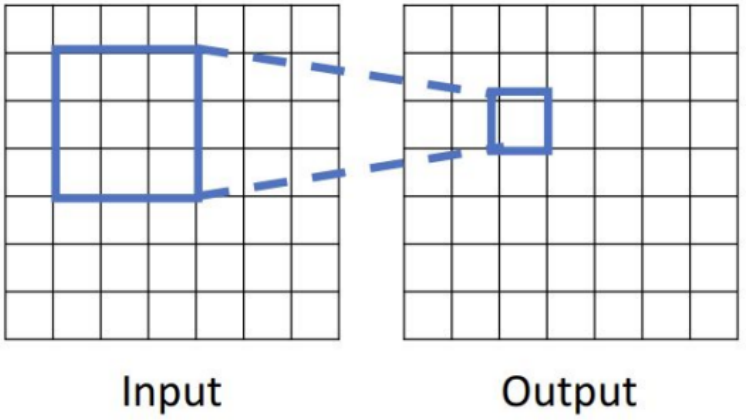


In the previous slide the output becomes the input of the following layers.
Progressively his dimensionality is reduced

Receptive Fields

Every element in th eoutput depends on the area of your kernel. For convolution with kernel size K, each element in the output depends on a K x K receptive filed in the input

Refers to the region of the input image that influences a particular neuron im a feature map

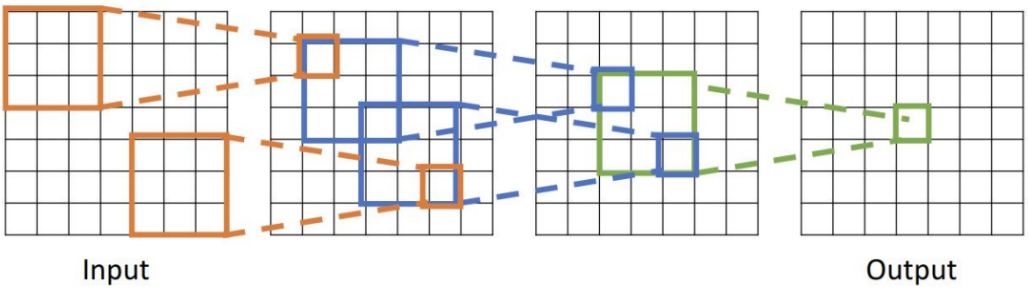


Each neuron im CNN layer extracts informations from its receptive field in the input

Each successive convolution adds K-1 to the receptive filed size. With L layers the receptive filed size is 1 + L * (K-1)

Im lower layers the receptive field is small, so neurons detect local patterns

Im deeper layers the receptive field becomes larger, allowing neurons to detect global pattern or high level features



lower layers only see small portion of the image due to the size of receptive fields

Problem → For large images we need many layers for each output to see the whole image

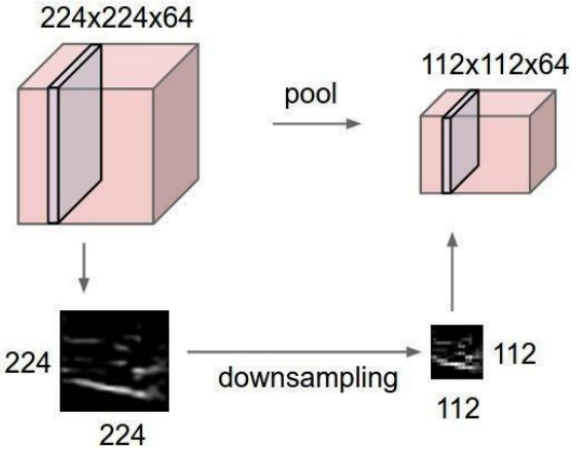
Solution → Downsample inside the network

Reduce the spatial size of the image as it passes through network

Pooling layers

Makes the representations smaller and more manageable.

Operates over each activation map (reduce the dimensionality) independently



Hyperparameters:
Kernel Size
Stride
Pooling function

Pooling layer don't have parameters that you have to learn, but just hyperparameters.