

Convolutional Neural Networks

Session outline

CNN origins

- A parsimonious technique for image recognition

General Principles

- Convolution operator

Main parameters

- Padding, Stride, Dilatation

Pooling layers

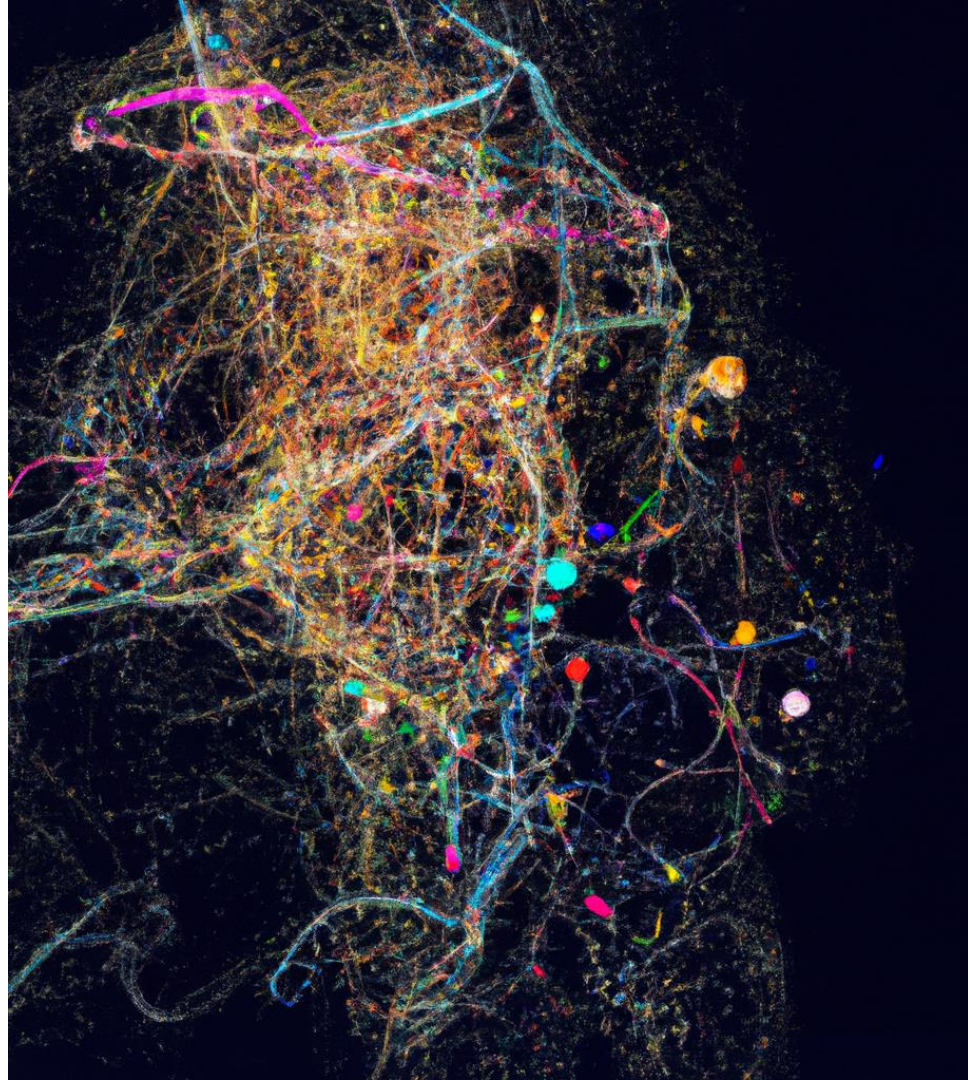
- Average and Max Pooling

CNN architectures

- Common approaches

CNN origins

A parsimonious technique for
image recognition



The first CNNs

CNN origins

- Kunihiro Fukushima developed the first CNN in 1979 with the creation of Neocognitron a hierarchical, multilayered design to recognize visual patterns. It was trained with a reinforcement strategy of recurring activation in multiple layers.
- A decade later, in 1989, after Rumelhart et al. had shown that backpropagation could be applied to neural networks successfully, Yann LeCun demonstrated the practical use of backpropagation for reading handwritten digits using convolutional neural networks at Bell Labs.
- Interestingly, this last discovery happened during the second AI winter, that occurred between the late 1980s and early 1990s which again impacted research on neural networks and deep learning. The origins of this second AI winter were again the failure of AI systems to live up to high expectations.
- Other contributing factors included the high cost of developing AI systems and limited computing power at the time.

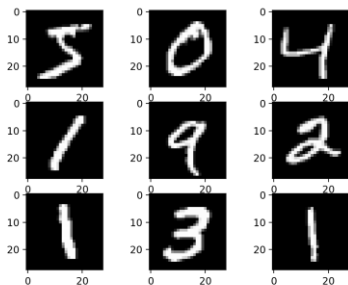
FCNN* dimensional curse

CNN origins

An example with a single hidden layer containing **1,000** neurons

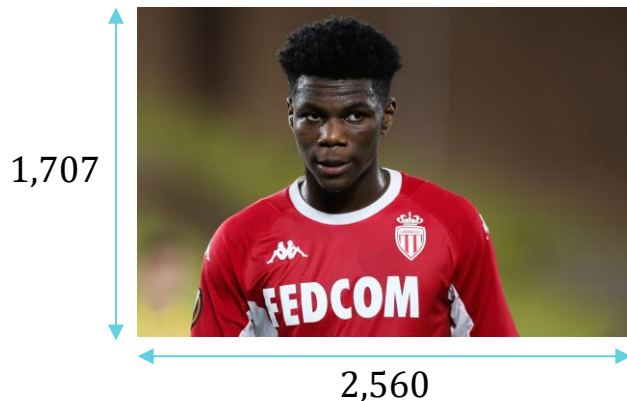
MNIST (greyscale)

- $28 \times 28 = 784$ pixels
- $784 \times 1,000 \approx 8 \times 10^5$ parameters to be estimated



High quality images (RGB)

- $2,560 \times 1,707 \times 3 \approx 13 \times 10^6$ pixels
- $13 \times 10^6 \times 1,000 \approx 13 \times 10^9$ parameters to be estimated



*FCNN = Fully Connected Neural Network

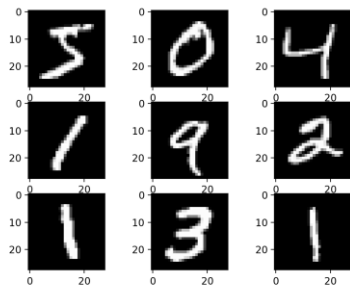
FCNN* dimensional curse

CNN origins

Parameters encoded as float32

MNIST (greyscale)

- $8 \times 10^5 \times 32\text{bits} = 3,2 \text{ Mo}$
- Memory capacity ok



High quality images (RGB)

- $13 \times 10^9 \times 32\text{bits} = 52 \text{ Go}$
- Memory capacity exceeded!!!



*FCNN = Fully Connected Neural Network

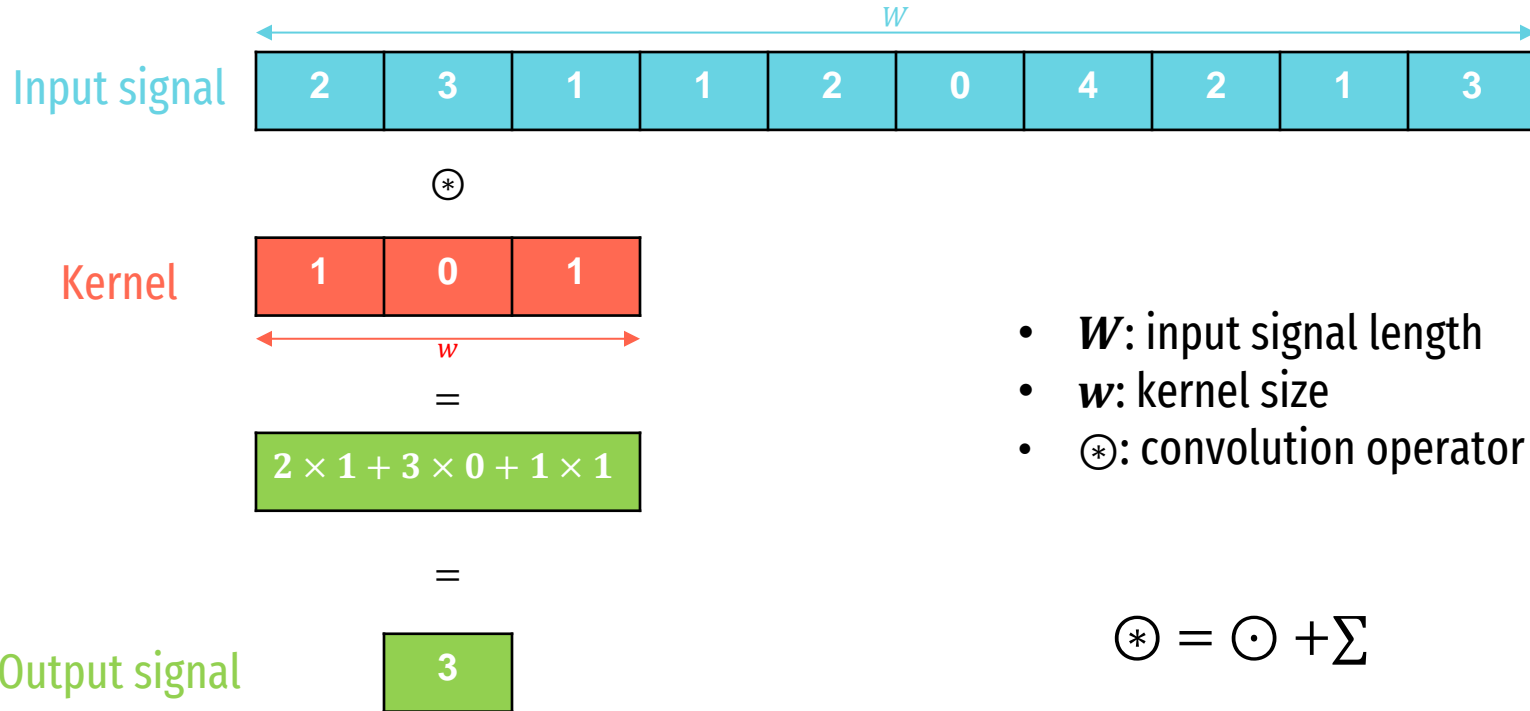
General Principles

Convolution operator



1D Convolutions

General Principles



1D Convolutions

General Principles



Kernel



=



=

Output signal

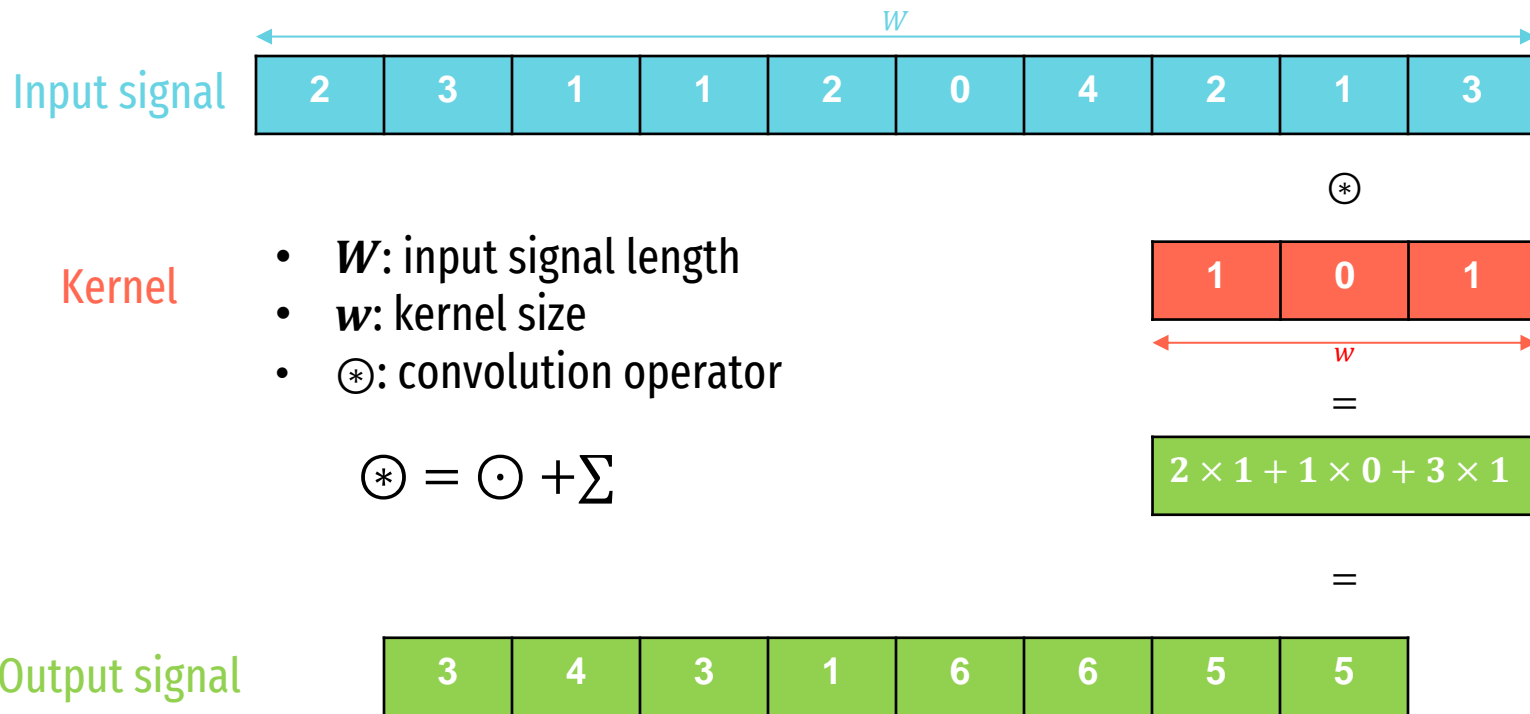


- W : input signal length
- w : kernel size
- \otimes : convolution operator

$$\otimes = \odot + \Sigma$$

1D Convolutions

General Principles



1D Convolutions

General Principles



Kernel

- $w + 1$ parameters to be estimated (including a bias)

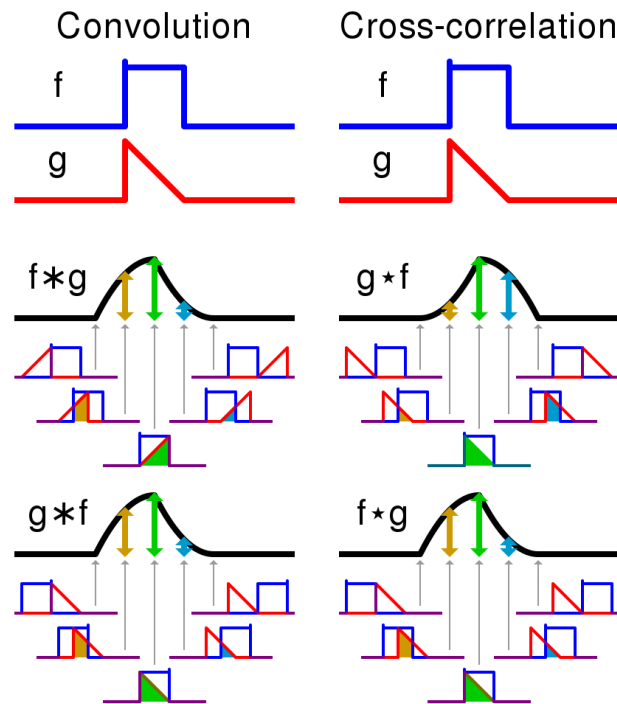


Output signal



Cross-correlation is the new convolution

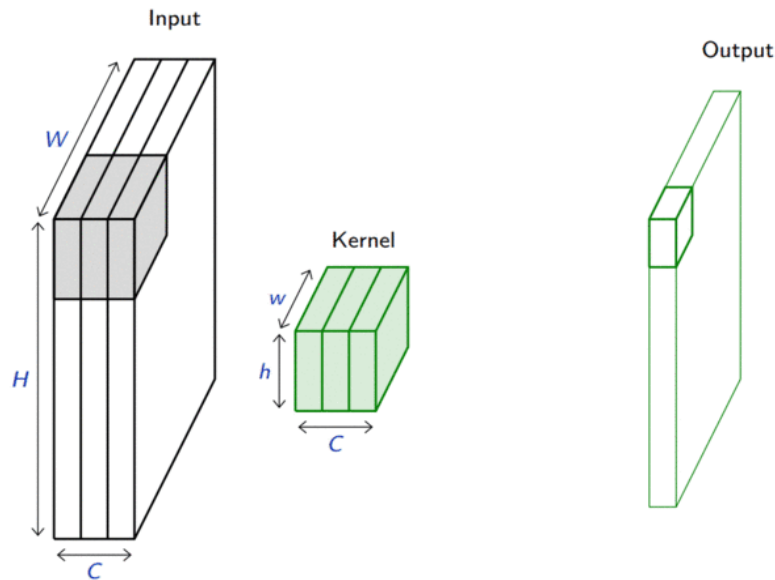
- Despite its name, a convolution in deep learning is what is called a cross-correlation in mathematics
- These two operations are very similar with the only difference that the second function is reversed in the convolution while it is kept the same in the cross-correlation
- This definition of the cross-correlation keeps it from being commutative as it is the case for the convolutions



2D Convolutions (for RGB images)

General Principles

- A 2D convolution usually takes as input a 3D tensor, called the input feature map
- A kernel slides across the input feature map, along its height and width
- At each location, we compute the Hadamard product between the kernel and the input
- We sum all elements obtained from the Hadamard product to get the output



W : input width **H** : input height **C** : number of input channels (RGB)
 w : kernel width **h** : kernel height

2D Convolutions (for RGB images)

General Principles

- Let us define $x \in \mathbb{R}^{W \times H \times C}$ the input tensor (input feature map) and $u \in \mathbb{R}^{w \times h \times C}$ a kernel
- The convolution final output is called the output feature map and can be written as follows:

$$o_{j,i} = b_{j,i} + \sum_{c=0}^{C-1} (x_c \odot u_c)[j, i] = b_{j,i} + \sum_{c=0}^{C-1} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} x_{c,n+j,m+i} u_{c,n,m}$$

Remarks

- The parameters to be estimated with backpropagation are b and u
- The size of the output feature map is $(H - h + 1) \times (W - w + 1)$
- We can repeat D convolutions to obtain D output feature maps
- In our previous example for 1D convolutions we had $H = h = C = 1$
- It is quite rare to find more than 1 channel for 1D input signals

W : input width H : input height C : number of input channels (RGB)
 w : kernel width h : kernel height $o_{j,i}$: output feature map coordinate

Main parameters

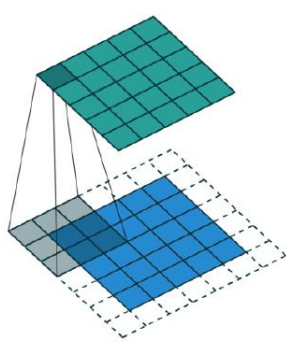
Padding, Stride, Dilatation



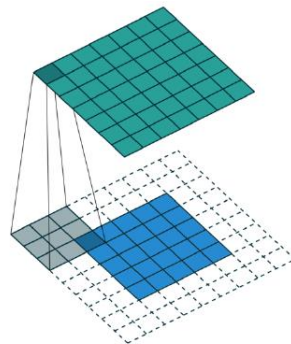
Padding

Main parameters

- We've seen that by default the output feature map size is $(H - h + 1) \times (W - w + 1)$
- Consequently, the output image will be smaller by definition (*padding = 'valid'*)
- To get an output feature map of a different size than the one by default we can use different parameters and in particular the padding parameter
- Padding essentially lets the kernel go partly outside the image border such that we can create an output feature map of varying size



Padding = 'same'

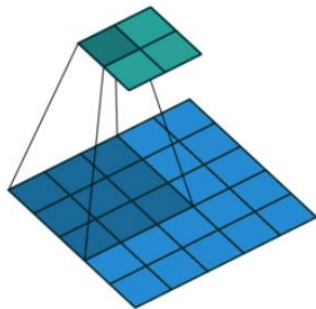


Padding = (2,2)

Stride

Main parameters

- It is perfectly possible to consider sliding along the kernel by more than 1 square at every step
- For instance, to have a non-overlapping convolution in the width direction we'd have to take a stride of the same value as the kernel width
- Therefore, the more stride we add the more we decrease the size of the output feature map
- Like for padding, you can specify an integer or a tuple for the stride (e.g., 2 or (2,1))

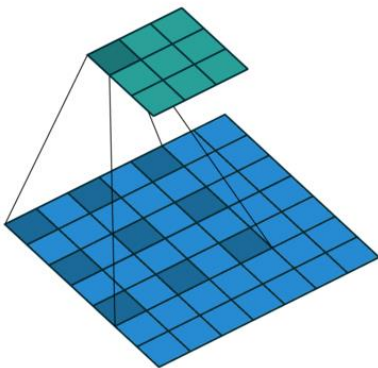


Stride = 2 = (2,2)

Dilatation

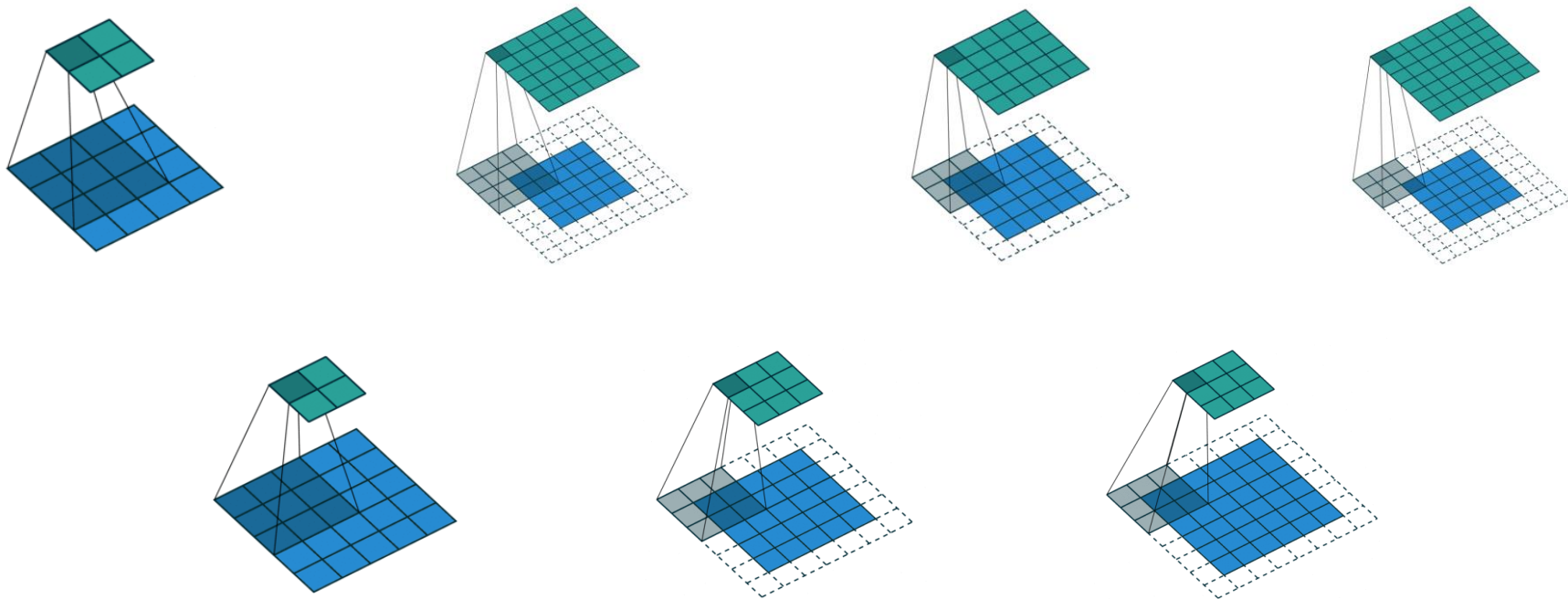
Main parameters

- Dilatation controls the spacing between the values in the kernel used in the layer
- Increasing the dilation allows the filter to "skip" over more values in the input
- This increases the field of view of the filter and allows it to capture larger patterns in the input
- This can be useful in image segmentation tasks where spatial relationships between objects need to be preserved



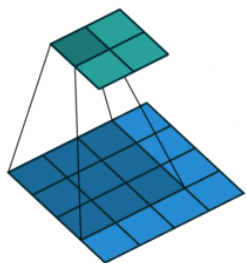
Different parameters, different convolutions

Main parameters

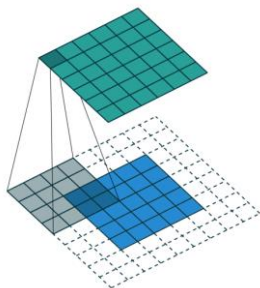


Different parameters, different convolutions

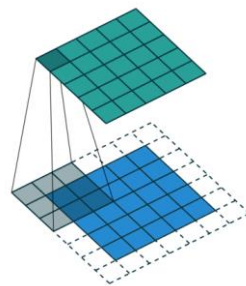
Main parameters



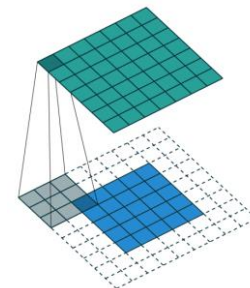
*kernel size = 3
padding = 0 = 'valid'
stride = 1*



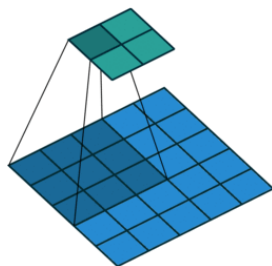
*kernel size = 4
padding = 2
stride = 1*



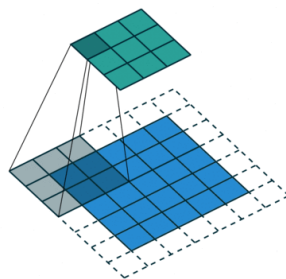
*kernel size = 3
padding = 1 = 'same'
stride = 1*



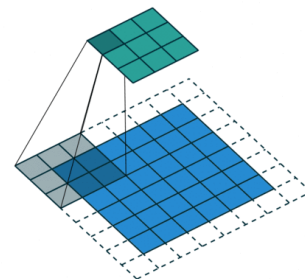
*kernel size = 3
padding = 2
stride = 1*



*kernel size = 3
padding = 'valid'
stride = 2*



*kernel size = 3
padding = 1
stride = 2*



*kernel size = 3
padding = 1
stride = 2*

A word of parsimony

Main parameters

- **FCNN** with an input size $W \times H \times C$ and a single hidden layer of m neurons

$m(W \times H \times C + 1)$ parameters to be estimated

- **CNN** with input size $W \times H \times C$, a single convolutional layer of depth m with kernel size (k_1, k_2)

$m(Ck_1k_2 + 1)$ parameters to be estimated

Remark

- The number of parameters to be estimated for a CNN is independent of the width and height of the images

Pooling layers

Max and average pooling



General Principle

- Pooling is a technique used in convolutional neural networks to reduce the spatial dimensions of the feature maps while retaining important information
- It is applied after the convolutional layers to reduce the number of parameters in the network and to prevent overfitting
- Commonly used pooling operations include max pooling and average pooling
- Max pooling selects the maximum value from each patch of the feature map, while average pooling takes the average of all values in the patch
- Pooling layers typically have a fixed kernel size and stride, and no weights are learned during training

Main pooling operators

Let us consider a pooling size of $h \times w$ for a 3D tensor $x \in \mathbb{R}^{C \times (rh) \times (sw)}$

- **Average Pooling**

$$o_{c,j,i} = \frac{1}{hw} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} x_{c,rj+n,si+m}$$

- **Max Pooling**

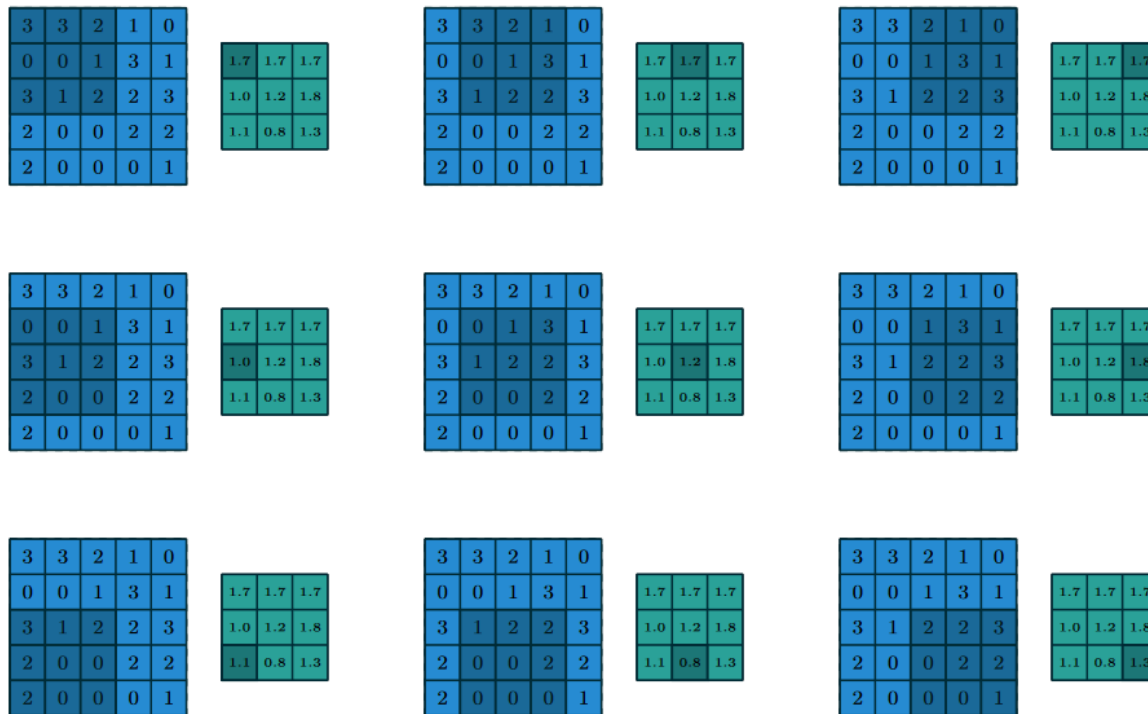
$$o_{c,j,i} = \max_{n < h, m < w} x_{c,rj+n,si+m}$$

Remark

- The output tensor is of size $C \times r \times s$

Average Pooling example

Pooling layers



Max Pooling example

Pooling layers

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

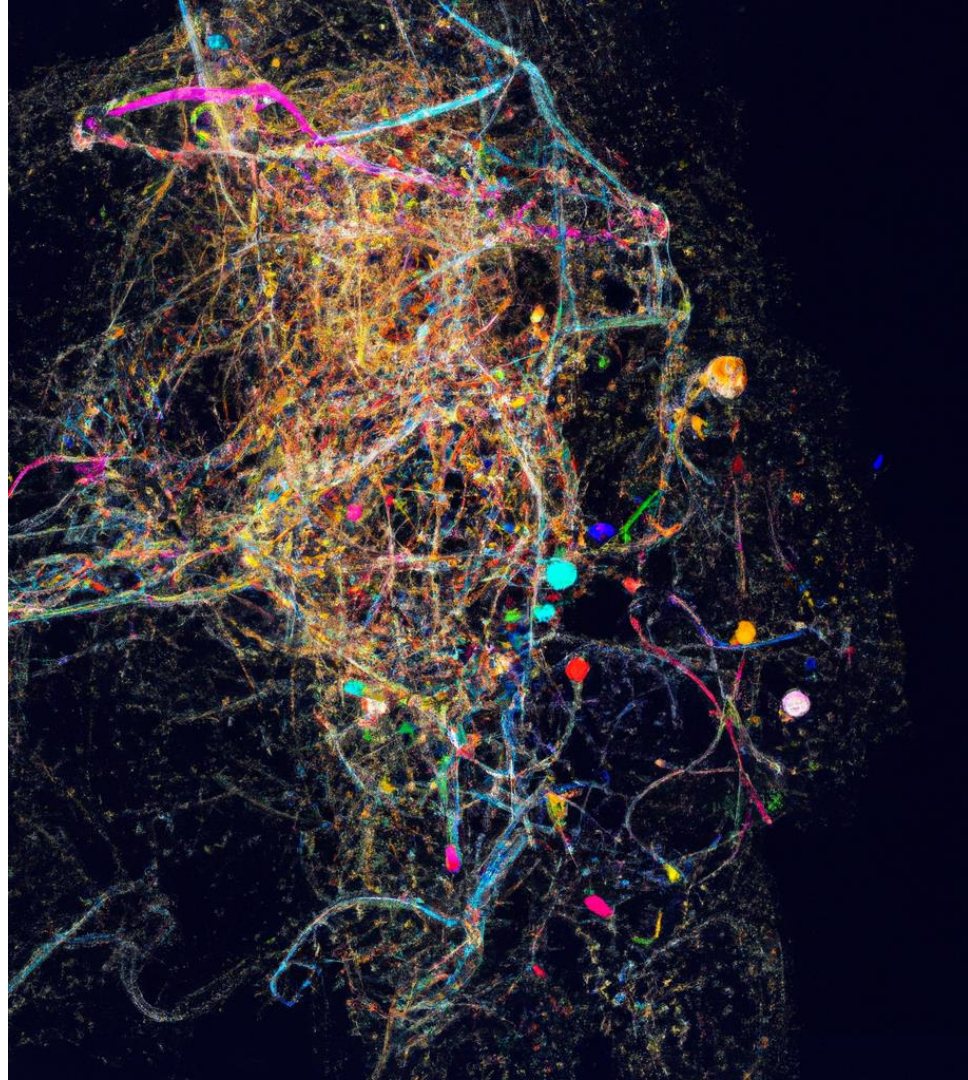
3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

CNN architectures

Common approaches



A typical CNN architecture

Common approaches

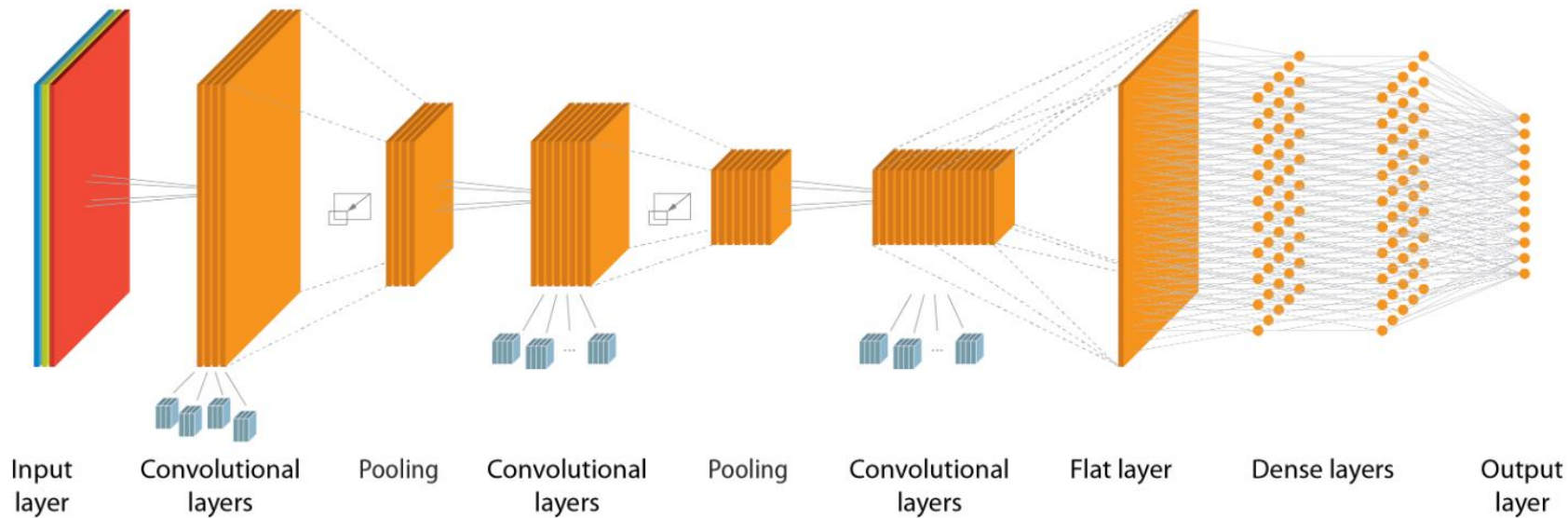
A typical **CNN** architecture consists of multiple layers, including:

- **Input layer:** where the raw image data is fed into the network.
- **Convolutional layers:** where the image is broken down into features through convolutions
- **Pooling layers:** where we reduce the image spatial dimensions and network parameters
- **Fully connected layers:** where the features are processed for classification or regression
- **Output layer:** where the final outputs are produced.

Between each layer, non-linear activation functions like ReLU are applied to introduce non-linearity into the network.

A typical CNN architecture

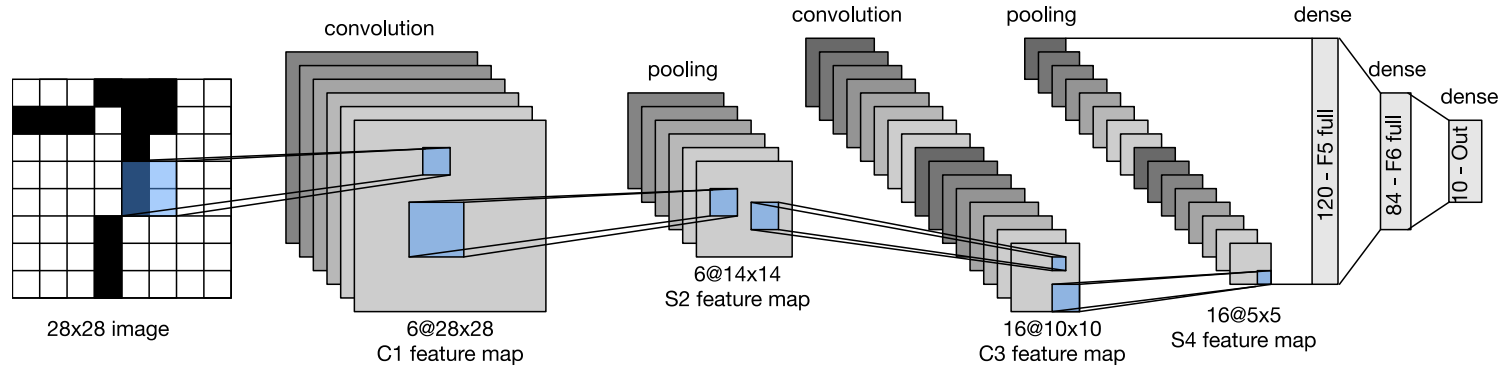
Common approaches



- We could add as many **Convolutional Layers** as we'd want
- **Pooling Layers** are optional

LeNet

Common approaches

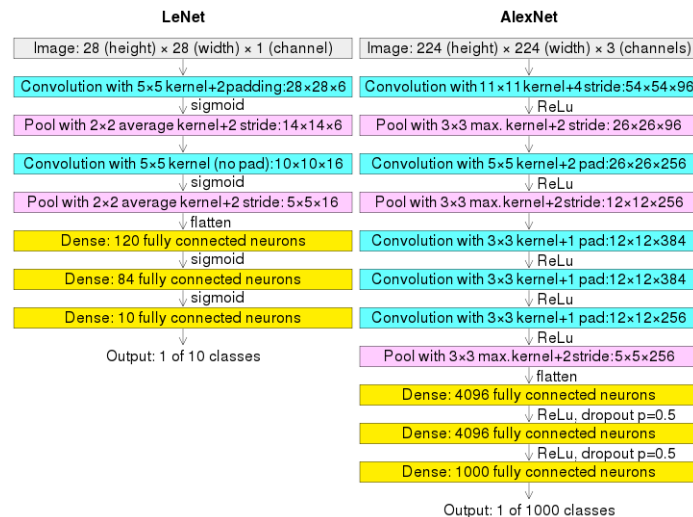


- **Sigmoid** activation functions in hidden layers
- **Softmax** output with a negative log-likelihood loss

AlexNet

Common approaches

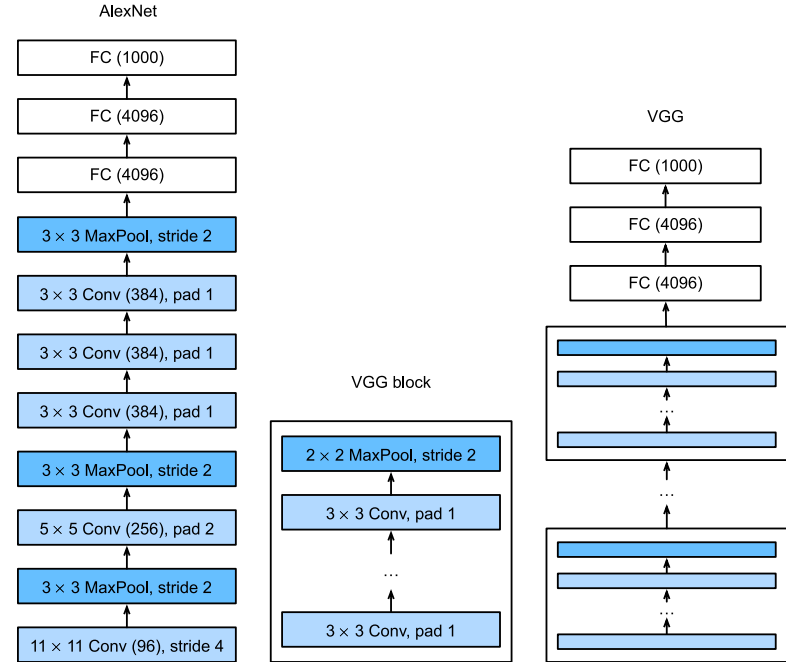
- The first **modern CNN** (Krizhevsky et al., 2012), named **AlexNet** after one of its inventors, **Alex Krizhevsky**, is largely an evolutionary improvement over **LeNet**. It achieved excellent performance in the 2012 ImageNet challenge.
- **LeNet** has **7** hidden layers while **AlexNet** has **11** hidden layers
- **AlexNet** has traded sigmoid activation functions for **ReLU** which brought significant improvement
- **AlexNet** added **dropout** in the dense layers of the network



VGG

- The VGG network was first proposed by the **Visual Geometry Group** at Oxford University
- Simonyan and Zisserman (2014) proposed using multiple convolutions in between downsampling via max-pooling in the form of a block
- They conducted a detailed analysis and found that deep and narrow networks significantly outperform shallow counterparts

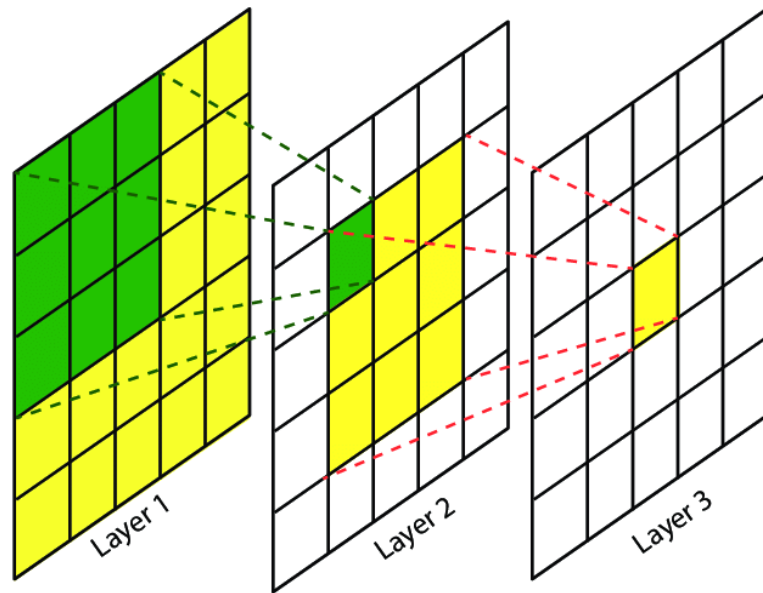
Common approaches



Effective receptive field

Common approaches

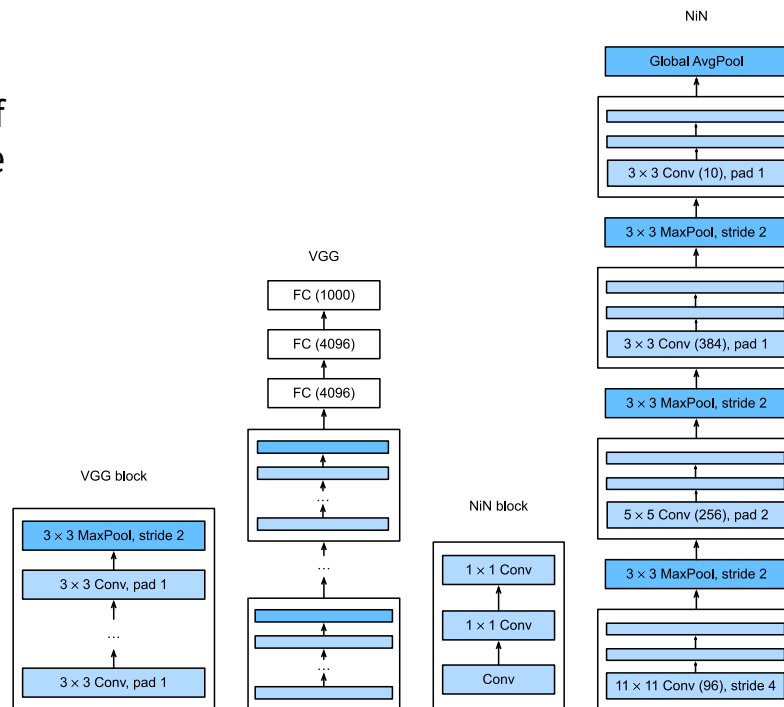
- The effective receptive field is the area of the image that affects a unit through previous layers and increases with depth.
- Multiple small kernels can achieve the same effective receptive field as a single large kernel with fewer parameters
- For instance, two consecutive 3 by 3 kernels are equivalent to a single 5 by 5 kernel with fewer parameters



Network in Network

- The inputs and outputs of convolutional layers consist of four-dimensional tensors with axes corresponding to the example, channel, height, and width
- The inputs and outputs of fully connected layers are typically two-dimensional tensors corresponding to the example and feature.
- NiN applies a convolution that can be thought as a fully connected layer acting independently on each pixel location
- This can be seen as a structural difference between VGG and NiN in the blocks, where NiN uses initial convolution followed by other convolutions whereas VGG retains convolutions and NiN no longer requires a giant fully connected layer at the end

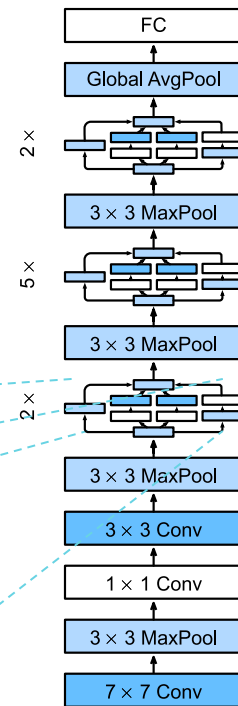
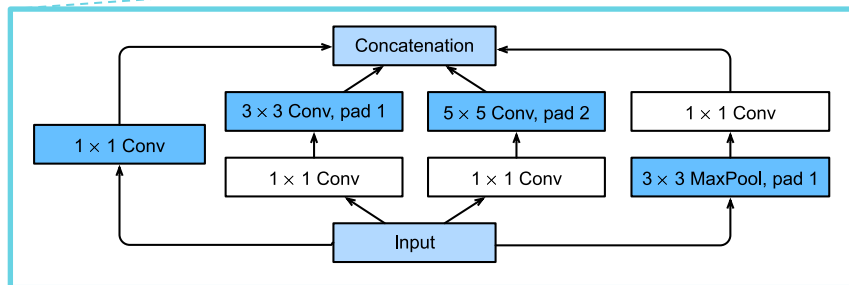
Common approaches



GoogLeNet – Inception Blocks

Common approaches

- The **Inception block** is the basic building block in GoogLeNet, named after the "we need to go deeper" meme from the movie Inception
- GoogLeNet won the ImageNet Challenge in 2014 (Szegedy et al., 2015)
- The network kernels explore the image with different sizes, allowing efficient recognition of details at different extents.
- The network allocates different parameter numbers for the kernels



ResNet and DenseNet

Common approaches

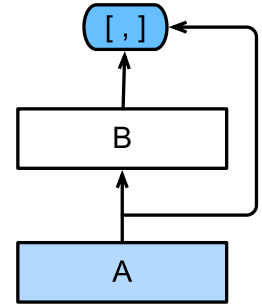
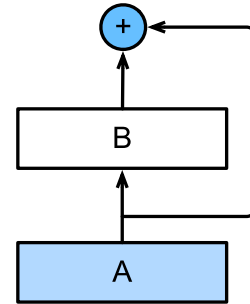
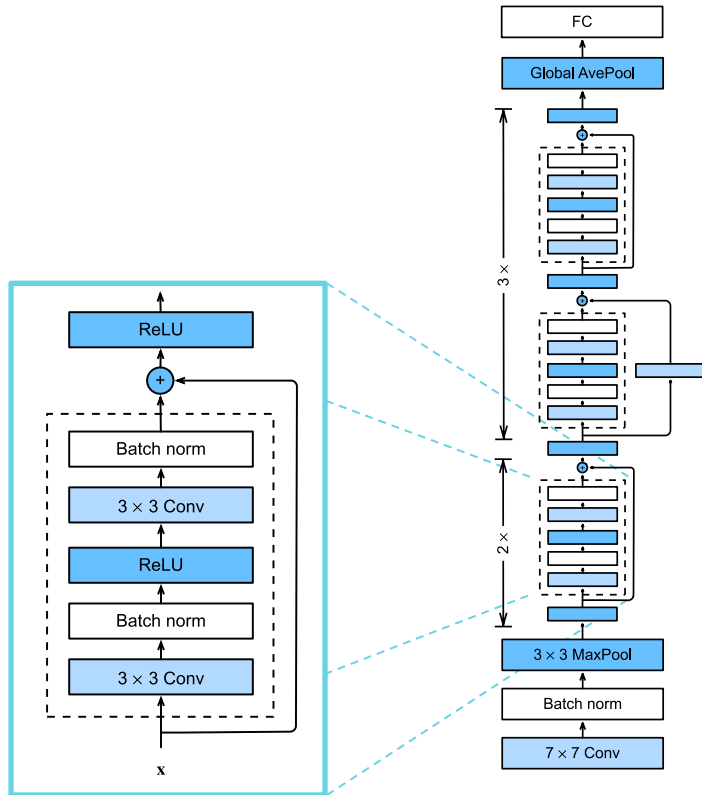
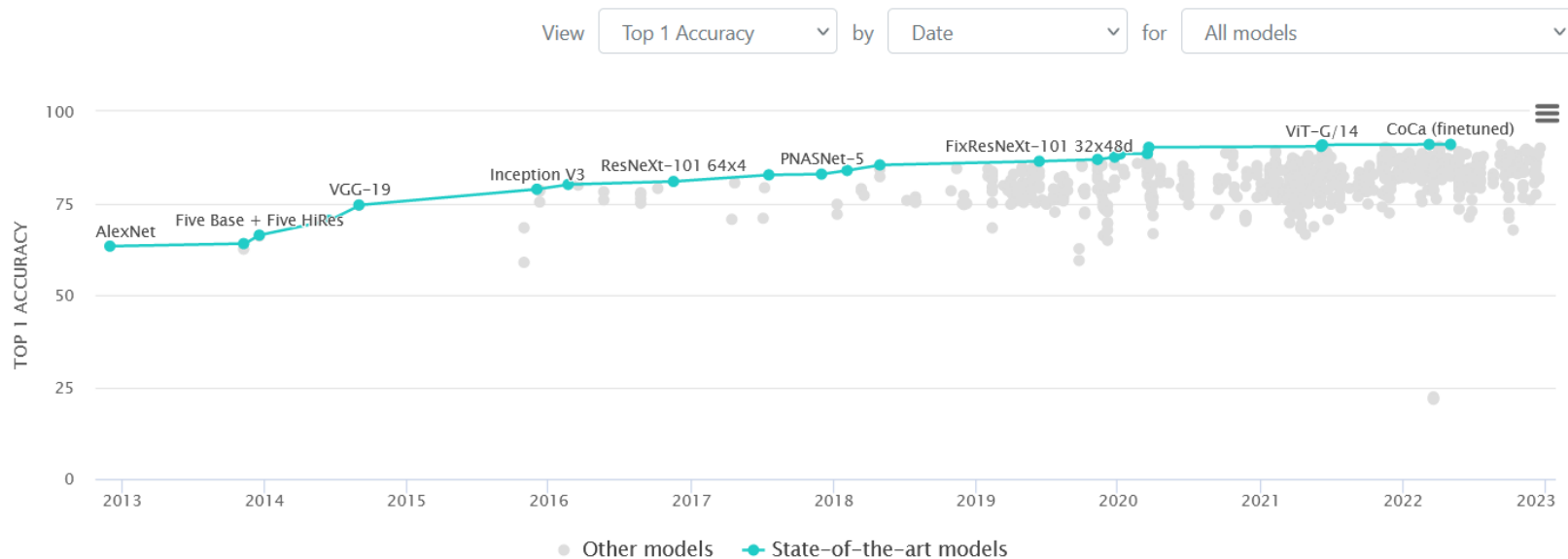


Image Classification on ImageNet

Common approaches

Leaderboard

Dataset



ENBIS 2025



“Explainable” Deep Learning in a decentralized grid

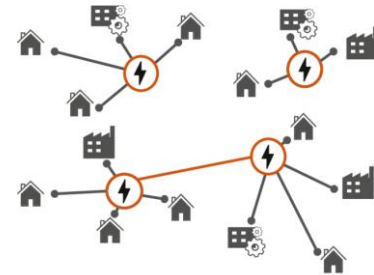
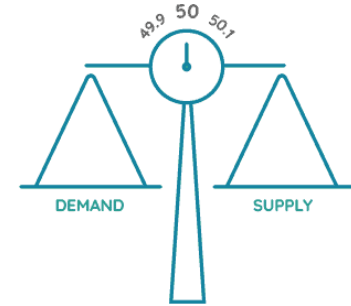
Yvenn Amara-Ouali

17th September 2025



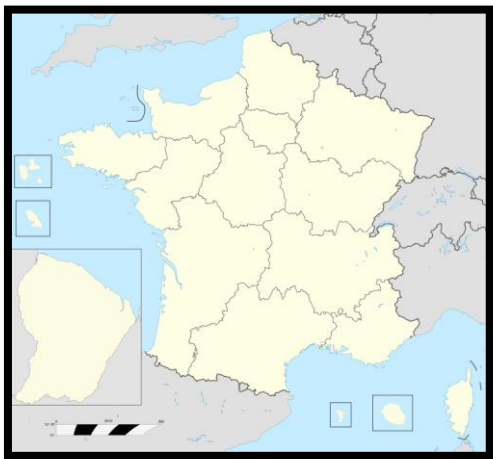
Motivations

- **Electricity demand must be forecasted accurately** to maintain balance between supply and demand, as electricity cannot be stored efficiently and must match real-time consumption.
- **New challenges arise** from decentralized networks, economic shifts like Covid, and non-stationary consumption patterns, increasing forecast complexity.
- **Our work focuses on multi-scale forecasting**, leveraging granular consumption and geolocalized data to reduce uncertainty across different levels.



The aim of the following projects is therefore to focus on methods that can propose a forecast at **different levels of granularity**

What do we mean by « Hierarchical Data »?



Spatial

Cities → Regions → Nation

Temporal

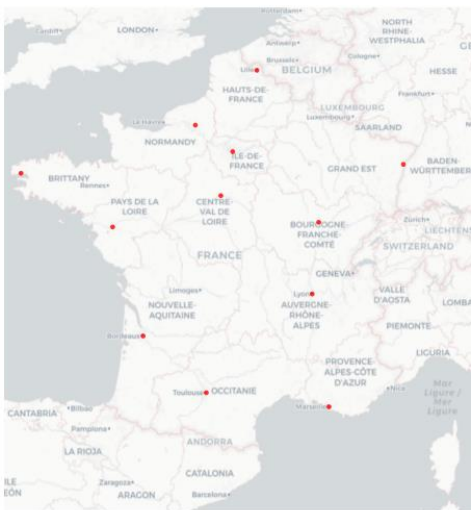
Days → Months → Year(s)

Functional

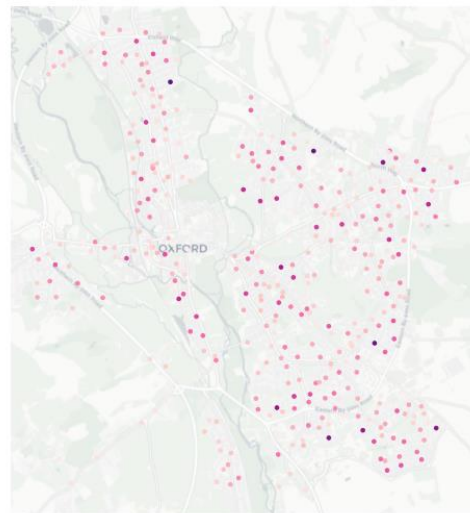
Total Load →
Industrial vs Residential

Using Graph Neural Networks and Attention

- There is a **lack of models leveraging spatial dependencies** between regions/meters, despite their importance for capturing correlated demand patterns.
- **Attention-based Graph Neural Networks (GNNs)** offer both predictive accuracy and some degree of interpretability, which remains scarce in the load forecasting literature.



France: regional aggregated load



Oxford, UK: household-level consumption

Notations and Metric

Notations

$\tilde{\mathbf{X}} \in \mathbb{R}^{n \times d \times T}$: node features (n nodes, d features, T time steps)

$\mathbf{y} \in \mathbb{R}^{n \times T}$: target values

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$: graph with vertices \mathcal{V} and edges \mathcal{E}

ϕ_{θ} : GNN with parameters θ

w, h : input and output window sizes

\mathcal{T} : set of timesteps to be covered

Training Loss Function

$$\min_{\theta} \sum_{t \in \mathcal{T}} \|\phi_{\theta}(\mathcal{G}, \tilde{\mathbf{X}}_t \mathbf{y}_{t-1:t-w}) - \mathbf{y}_{t:t+h}\|^2$$

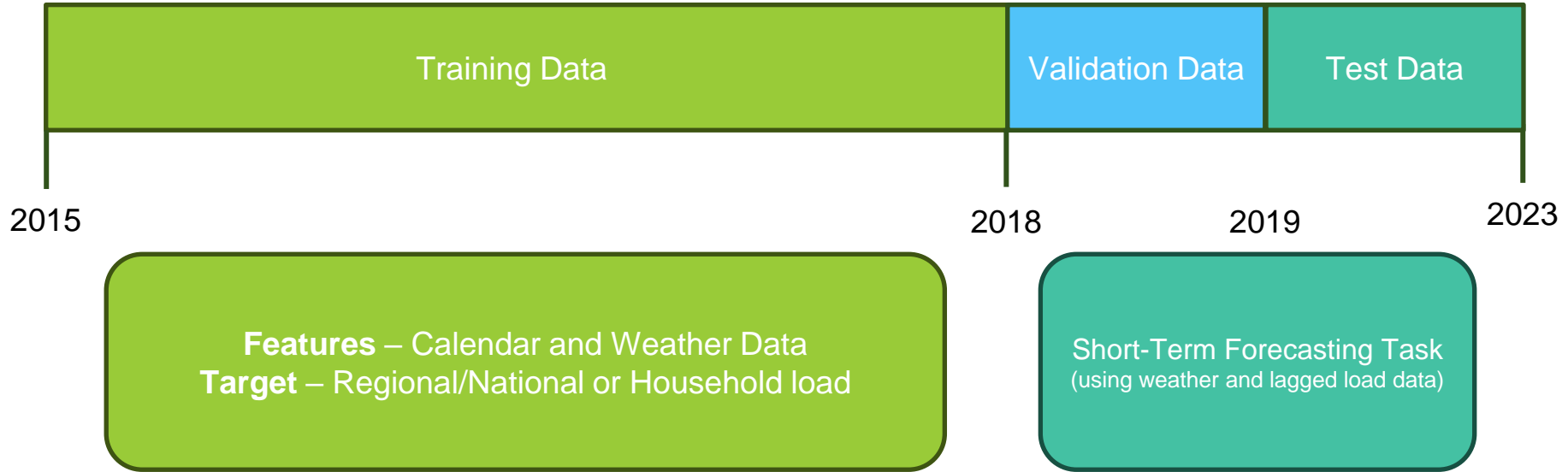
$$\|\hat{\mathbf{y}}_{t:t+h} - \mathbf{y}_{t:t+h}\|^2 = \sum_{i \in \mathcal{V}} \sum_{k=0}^h (\hat{y}_{i,t+k} - y_{i,t+k})^2$$

GNN updates with or without attention

We denote by N_v the neighborhood of a node v , by $\mathbf{h}_v^{(\ell)}$ the representation vector of node v at layer ℓ , by A the adjacency matrix, and by \mathbf{W} and \mathbf{b} the learned weight matrices and bias vectors, respectively.

Model	Example	Update rule
General	—	$\mathbf{h}_v^{(\ell+1)} = \text{UPDATE}^{(\ell)}\left(\mathbf{h}_v^{(\ell)}, \text{AGGREGATE}^{(\ell+1)}\{\mathbf{h}_u^{(\ell)} \mid u \in \mathcal{N}_v\}\right)$
Spatial	GCN, SAGE	$\mathbf{h}_v^{(\ell+1)} = \sigma\left(\sum_{u \in \mathcal{N}_v} c_{uv} \mathbf{W}^{(\ell)} \mathbf{h}_u^{(\ell)} + \mathbf{b}\right)$
Attentional	GAT, GATv2, Transformer	$\mathbf{h}_v^{(\ell+1)} = \sigma\left(\sum_{u \in \mathcal{N}_v} \alpha_{uv}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_u^{(\ell)} + \mathbf{b}\right)$
Spectral	TAG, Cheby	$\mathbf{h}_v^{(\ell+1)} = \sigma\left(\sum_{k=0}^K P_k(\mathbf{A}) \mathbf{W}_k \mathbf{h}_v^{(\ell)} + \mathbf{b}\right)$
Propagation	APPNP	$\mathbf{h}_v^{(\ell+1)} = \sigma\left((1 - \alpha) \sum_{u \in \mathcal{N}_v} \hat{A}_{uv} \mathbf{h}_u^{(\ell)} + \alpha \mathbf{h}_v^{(0)} + \mathbf{b}\right)$

Experimental setting



```
date,pred_France,pred_Auvergne-Rhône-Alpes,[...],pred_Île-de-France,pred_Montpellier Méditerranée  
Métropole,[...],pred_Toulouse Métropole  
2022-01-01 00:00:00+01:00,69135.70,9321.05,1443.35,1078.94,1676.82  
2022-01-01 00:30:00+01:00,67406.70,9114.47,2780.58,3579.17,2850.35  
2022-01-01 01:00:00+01:00,64773.11,8833.41,2721.82,3328.47,2739.76  
2022-01-01 01:30:00+01:00,64469.11,8828.70,2755.88,5793.47,6824.76
```

Results for GNN on French and UK datasets

Model	French Dataset				UK Dataset			
	MAPE (%)		RMSE (MW)		MAPE (%)		RMSE (MW)	
	Unif.	Agg.	Unif.	Agg.	Unif.	Agg.	Unif.	Agg.
GCN	1.21	<u>1.09</u>	895	839	8.63	8.51	15.09	14.91
SAGE	<u>1.12</u>	1.14	873	898	8.19	<u>8.07</u>	14.87	14.85
GAT	1.11	1.08	883	<u>871</u>	<u>7.79</u>	8.29	<u>14.13</u>	<u>14.72</u>
GATv2	1.14	1.14	888	902	8.78	8.94	16.44	16.40
Transformer	1.14	1.15	883	910	8.36	8.48	15.32	15.53
TAG	1.14	1.14	884	881	9.23	9.41	15.79	16.19
Cheby	1.16	1.15	<u>878</u>	876	8.60	8.93	17.23	17.37
APPNP	1.22	1.20	944	943	6.54	7.60	12.61	14.13

France

- GAT achieves best MAPE
- SAGE achieves best RMSE
- GCN remain competitive

UK

- APPNP performs best
- GAT produces the 2nd best performance