

# Factorization in Deep Neural Networks



**IMT Atlantique**  
Bretagne-Pays de la Loire  
École Mines-Télécom

## Sessions

- 1 Deep Learning and Transfer Learning,
- 2 Quantization,
- 3 Pruning,
- 4 Data Augmentation
- 5 Factorization,
- 6 Distillation,
- 7 Embedded Software and Hardware for DL,
- 8 Presentations for challenge.

## Sessions

- 1 Deep Learning and Transfer Learning,
- 2 Quantization,
- 3 Pruning,
- 4 Data Augmentation
- 5 **Factorization**,
- 6 Distillation,
- 7 Embedded Software and Hardware for DL,
- 8 Presentations for challenge.

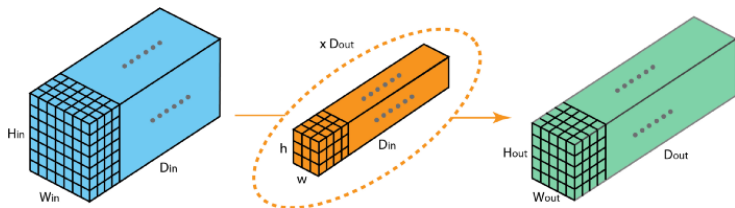
# Factorization of Convolutional Networks

## Why?

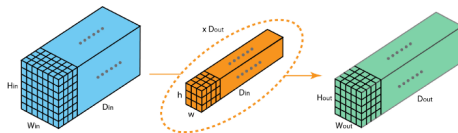
- 1 Reduce memory footprint
- 2 Reduce number of operations

## How?

Modifying (decomposing, simplifying) convolutional filters structure



# General principle



## Complexity of 2D Convolutions

$$N_{ops} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out}$$

with kernel size  $(h, w)$ ,  $D_{in}$  the number of input feature maps,  $D_{out}$  the number of output feature maps of height  $H_{out}$  and width  $W_{out}$ .

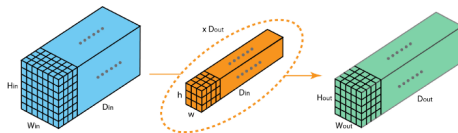
To reduce the number of parameters, we can :

- Reduce the size of kernels
- Reduce the number of feature maps

Different strategies :

- Decompose kernels (Spatial separable convolutions)
- Depthwise Separable Convolutions
- Grouped Convolutions

# General principle



## Complexity of 2D Convolutions

$$N_{ops} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out}$$

with kernel size  $(h, w)$ ,  $D_{in}$  the number of input feature maps,  $D_{out}$  the number of output feature maps of height  $H_{out}$  and width  $W_{out}$ .

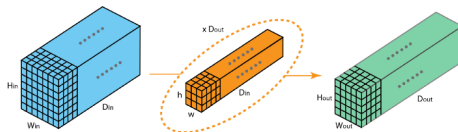
To reduce the number of parameters, we can :

- Reduce the size of kernels
- Reduce the number of feature maps

Different strategies :

- Decompose kernels (Spatial separable convolutions)
- Depthwise Separable Convolutions
- Grouped Convolutions

# General principle



## Complexity of 2D Convolutions

$$N_{ops} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out}$$

with kernel size  $(h, w)$ ,  $D_{in}$  the number of input feature maps,  $D_{out}$  the number of output feature maps of height  $H_{out}$  and width  $W_{out}$ .

To reduce the number of parameters, we can :

- Reduce the size of kernels
- Reduce the number of feature maps

Different strategies :

- Decompose kernels (Spatial separable convolutions)
- Depthwise Separable Convolutions
- Grouped Convolutions

# Decompose Kernels

## Spatially separable convolutions

To simplify, assuming  $D_{in} = D_{out}$ , decompose  $(h, w)$  kernel by  $(h, 1)$  and  $(1, w)$ :

$$N_{ops} = h \cdot 1 \cdot D_{in}^2 + 1 \cdot w \cdot D_{in}^2 = (h + w) \cdot D_{in}^2$$

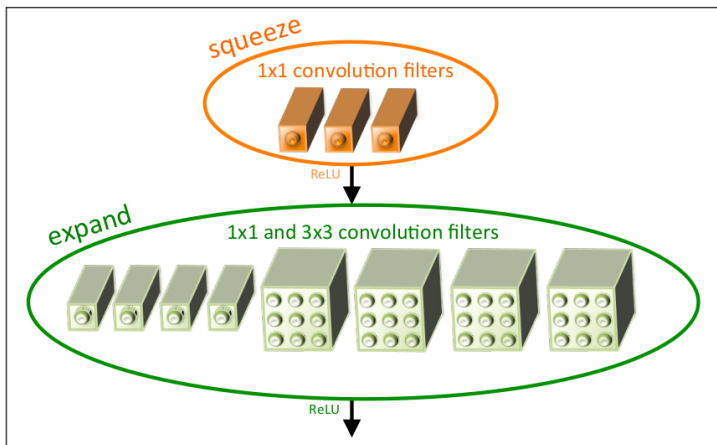
with kernel size  $(h, w)$ ,  $D_{in}$  input and out number of feature maps.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$



# Example: SqueezeNet

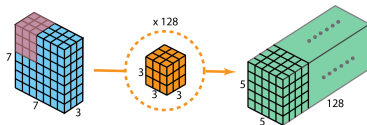
## Introducing the Fire Module



landola et al. 2016, <https://arxiv.org/abs/1602.07360>

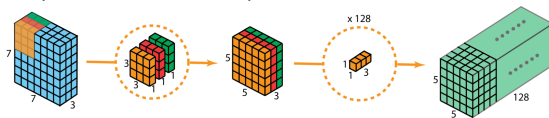
# Depthwise separable convolutions

Instead of learning parameters that recombine all input feature maps to compute each output feature map:



$$N_{mul}^N = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out} = 5 \cdot 5 \cdot 3 \cdot 3 \cdot 3 \cdot 128 = 86400$$

One can separate the operations into two steps:



$$N_{mul}^D = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot 1 + H_{out} \cdot W_{out} \cdot 1 \cdot 1 \cdot D_{in} \cdot D_{out}$$

$$N_{mul}^D = 5 \cdot 5 \cdot 3 \cdot 3 \cdot 3 \cdot 1 + 5 \cdot 5 \cdot 1 \cdot 1 \cdot 3 \cdot 128 = 10275$$

$$N_{mul}^D = \left( \frac{1}{D_{out}} + \frac{1}{h^2} \right) \cdot N_{mul}^N, h = w$$

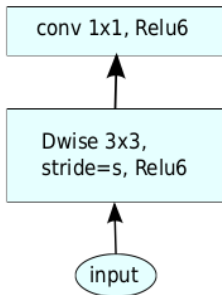
<https://towardsdatascience.com/>

a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215 ▶

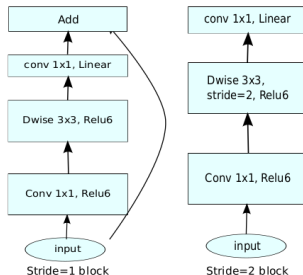
Navigation icons: back, forward, search, etc.

# Example: MobileNet

## MobileNetV1



## MobileNetV2



<https://arxiv.org/abs/1704.04861>, <https://arxiv.org/abs/1801.04381>

# Example: MobileNet

Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

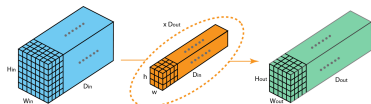
Table 10. MobileNet for Stanford Dogs

Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

<https://arxiv.org/abs/1704.04861>, <https://arxiv.org/abs/1801.04381>

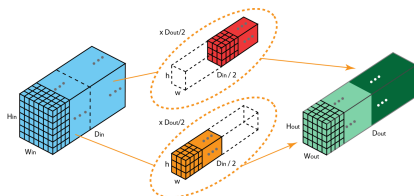
# Grouped Convolutions

Instead of learning parameters that recombine all input feature maps to compute each output feature map:



$$N_{mul}^N = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out}$$

One can divide the kernels into multiple groups:



$$N_{mul}^G = H_{out} \cdot W_{out} \cdot h \cdot w \cdot \frac{D_{in}}{2} \cdot \frac{D_{out}}{2} + H_{out} \cdot W_{out} \cdot h \cdot w \cdot \frac{D_{in}}{2} \cdot \frac{D_{out}}{2}$$
$$N_{mul}^G = \frac{N_{mul}^N}{2}$$

# Examples

## AlexNet filters



## ResNeXt block

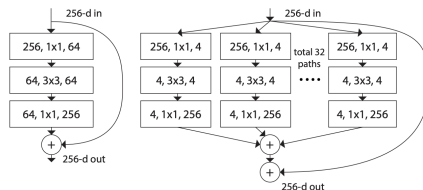


Figure 1. **Left:** A block of ResNet [14]. **Right:** A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

<https://arxiv.org/abs/1704.04861>, <https://arxiv.org/abs/1801.04381>, <https://arxiv.org/abs/1611.05431>

# Combining Factorization with other Techniques: Attention based Pruning

## Introducing Shift Attention Layer

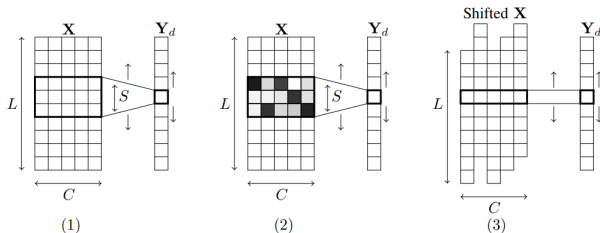


Figure 1: Overview of the proposed method: we depict here the computation for a single output feature map  $d$ , considering a 1d convolution and its associated shift version. Panel (1) represents a standard convolutional operation: the weight filter  $W_{d,:}$ , containing  $SC$  weights is moved along the spatial dimension ( $L$ ) of the input to produce each output in  $Y_d$ . In panel (2), we depict the attention tensor  $A$  on top of the weight filter: the darker the cell, the most important the corresponding weight has been identified to be. At the end of the training process,  $A$  should contain only binary values with a single 1 per slice  $A_{d,c,:}$ . In panel (3), we depict the corresponding obtained shift layer: for each slice along the input feature maps ( $C$ ), the cell with the highest attention is kept and the others are disregarded. As a consequence, the initial convolution with a kernel size  $S$  has been replaced by a convolution with a kernel size 1 on a shifted version of the input  $X$ . As such, the resulting operation in panel (3) is exactly the same as the shift layer introduced in Wu et al. [2017], but here the shifts have been trained instead of being arbitrarily predetermined.

# Combining Factorization with other Techniques: Attention based Pruning

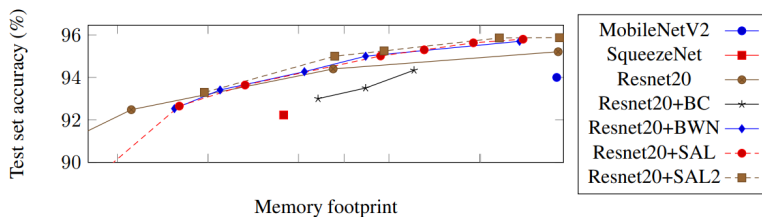
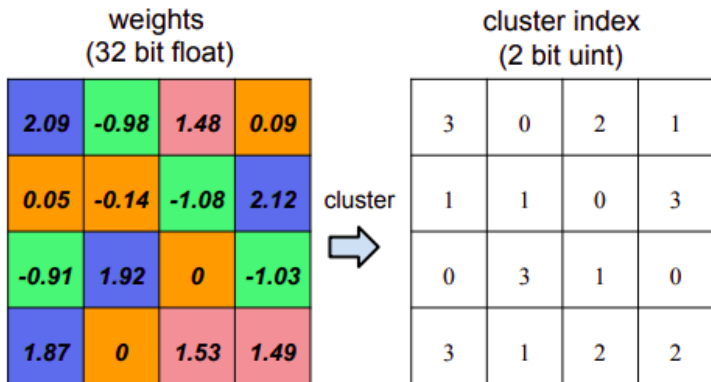


Figure 7: Evolution of accuracy when applying compression methods on different DNN architectures trained on CIFAR10.

Hacene et al. 2019, <https://arxiv.org/abs/1905.12300>



# Combining Factorization with other Techniques: using clustering to share kernel weights



from <https://arxiv.org/abs/1510.00149>

## Factorizing a CNNs using Pytorch

- Read carefully the documentation of **conv2d** (<https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html>) and play with the parameters *in channels*, *out channels* and *groups* to implement factorised convolutions
- Have a look at the MobileNet implementation for CIFAR10 (<https://github.com/kuangliu/pytorch-cifar/blob/master/models/mobilenet.py>)

## Work for your Long Project

- If you haven't done it yet, familiarise yourself with the `micronet-resources` folder (`profile.py` and challenge rules)
- Combine/test different strategies to improve your MicroNet score!



## 1 Overview of unsupervised learning

- Clustering
- Decomposition using Sparse Dictionary Learning

# Unsupervised learning

## Goal

Discover patterns/structure in  $X$ ,

## Unsupervised learning

- Unsupervised = no expert, no labels,
- Two main approaches:
  - Clustering = find a partition of  $X$  in  $K$  subsets,
  - Decomposition using  $K$  vectors.
- Applications :
  - Quantization,
  - Visualization...



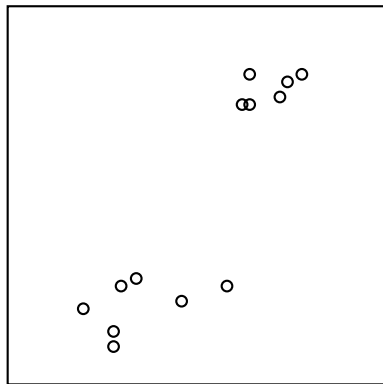
# Unsupervised learning

## Goal

Discover patterns/structure in  $X$ ,

## Unsupervised learning

- Unsupervised = no expert, no labels,
- Two main approaches:
  - Clustering = find a partition of  $X$  in  $K$  subsets,
  - Decomposition using  $K$  vectors.
- Applications :
  - Quantization,
  - Visualization...



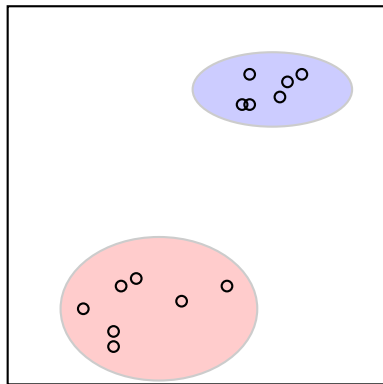
# Unsupervised learning

## Goal

Discover patterns/structure in  $X$ ,

## Unsupervised learning

- Unsupervised = no expert, no labels,
- Two main approaches:
  - Clustering = find a partition of  $X$  in  $K$  subsets,
  - Decomposition using  $K$  vectors.
- Applications :
  - Quantization,
  - Visualization...



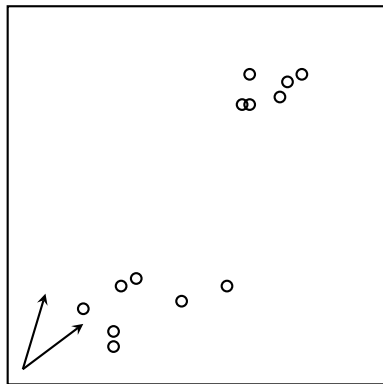
# Unsupervised learning

## Goal

Discover patterns/structure in  $X$ ,

## Unsupervised learning

- Unsupervised = no expert, no labels,
- Two main approaches:
  - Clustering = find a partition of  $X$  in  $K$  subsets,
  - Decomposition using  $K$  vectors.
- Applications :
  - Quantization,
  - Visualization...





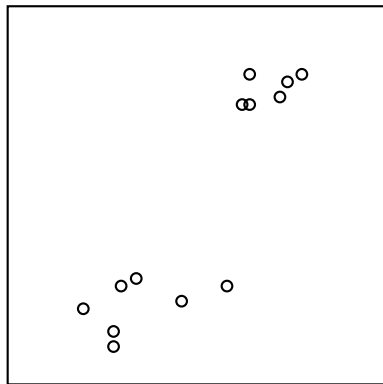
# Unsupervised learning

## Goal

Discover patterns/structure in  $X$ ,

## Unsupervised learning

- Unsupervised = no expert, no labels,
- Two main approaches:
  - Clustering = find a partition of  $X$  in  $K$  subsets,
  - Decomposition using  $K$  vectors.
- Applications :
  - Quantization,
  - Visualization...



# Example: clustering using $L_2$ norm

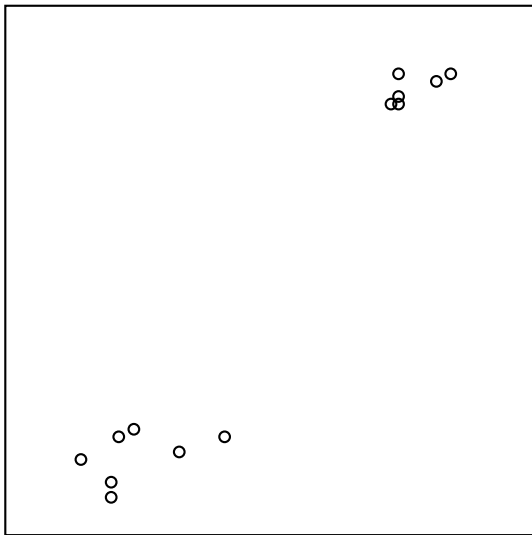
An example to perform clustering is to rely on distances to centroids. We define  $K$  cluster centroids  $\Omega_k, \forall k \in [1..K]$

## Definitions

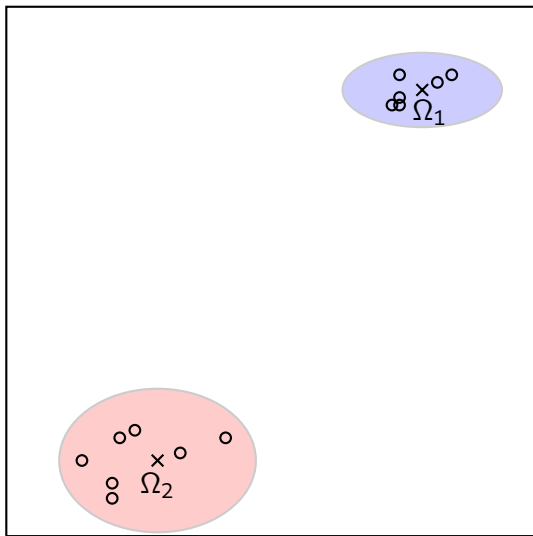
We denote  $q : \mathbb{R}^d \rightarrow [1..K]$  a function that associates a vector  $\mathbf{x}$  with the index of (one of) its closest centroid  $q(\mathbf{x})$ . Formally:

- $\forall k \in [1..K], \Omega_k \in \mathbb{R}^d$
- $\forall \mathbf{x} \in X, \forall j \in [1..K], \|\mathbf{x} - \Omega_{q(\mathbf{x})}\|_2 \leq \|\mathbf{x} - \Omega_j\|_2$
- Error  $E(q) \triangleq \sum_{\mathbf{x} \in X} \|\mathbf{x} - \Omega_{q(\mathbf{x})}\|_2$
- $X = \bigcup_k \underbrace{\{\mathbf{x} \in X, q(\mathbf{x}) = k\}}_{\text{cluster } k}$

# Example: clustering using $L_2$ norm



# Example: clustering using $L_2$ norm



# Clustering using $L_2$ norm

## Quantizing MNIST

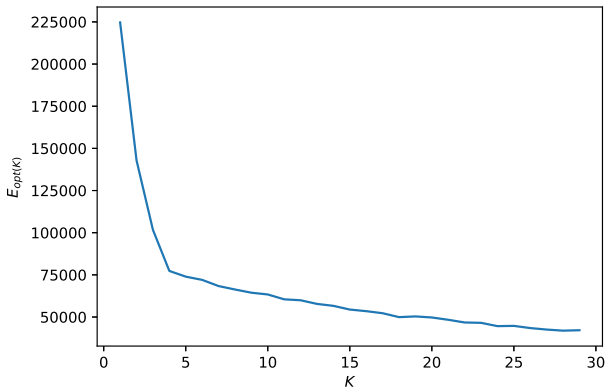
- Replace  $\mathbf{x}$  by  $\Omega_{k(\mathbf{x})}$
- Compression factor  $\kappa = 1 - K/N$



# Clustering using $L_2$ norm

## Choosing K

- Finding a compromise between error and compression,
- Simple practical method : "elbow".



# Sparse Dictionary Learning

## Definitions

Dictionary learning solves the following matrix factorization problem:

- The set  $X$  is considered as a matrix  $X \in \mathcal{M}_{N \times d}(\mathbb{R})$ ,
- We consider decompositions using a dictionary  $V \in \mathcal{M}_{K \times d}(\mathbb{R})$  and a code  $U \in \mathcal{M}_{N \times K}(\mathbb{R})$ , with the lines of  $V$  being with norm 1,
- Error  $E(U, V) \triangleq \|X - UV\|_2 + \alpha \|U\|_1$
- Training: find  $U^*, V^*$  that minimizes  $E(U^*, V^*)$
- $\alpha$  is a sparsity control parameter that enforces codes with soft ( $\ell_1$ ) sparsity

