INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Machine Learning 2 – 5A
Convolutional Neural Networks

Team: E. Chanthery, M.-V. Le Lann, P. Leleux, M. Siala

Université Fédérale Toulouse Midi-Pyrénées

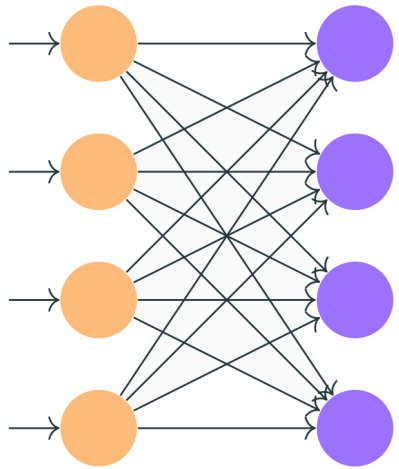
Image classification with embedded AI
Convolutional Neural Networks

Example : image classification of ImageNet.
Images of size $224 \times 224 \times 3$, in 1 000 classes.

```
model = keras.Sequential()
model.add(layers.Dense(1000, activation='relu', input_shape=(224*224*3,)))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 1000)              150529000

=================================================================
Total params: 150,529,000
Trainable params: 150,529,000
Non-trainable params: 0
_________________________________________________________________
```



$224 \times 224 \times 3$ pixels | $224 \times 224 \times 3$ neurones | 1000 neurones

- Spacial information is lost during *flattening* (input is a vector).
- A classical perceptron requires $(224 \times 224 \times 3 + 1) \times 1\,000 + 1\,000$ parameters, i.e. $> 1.5 \cdot 10^8$ parameters !

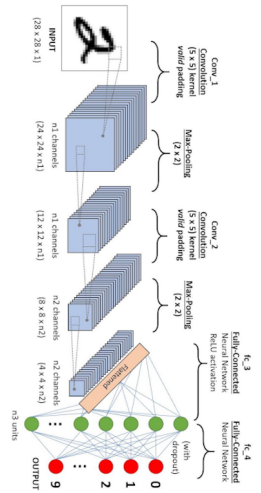
Image classification with embedded AI

Example : image classification of ImageNet.
Images of size $224 \times 224 \times 3$, in 1 000 classes.

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 220, 220, 8)	608
batch_normalization_6 (BatchNormalization)	(None, 220, 220, 8)	32
max_pooling2d_6 (MaxPooling2D)	(None, 110, 110, 8)	0
conv2d_13 (Conv2D)	(None, 106, 106, 16)	3216
batch_normalization_7 (BatchNormalization)	(None, 106, 106, 16)	64
max_pooling2d_7 (MaxPooling2D)	(None, 53, 53, 16)	0
flatten_9 (Flatten)	(None, 44944)	0
dense_17 (Dense)	(None, 32)	1438240
dense_18 (Dense)	(None, 2)	66

Total params: 1442226 (5.50 MB)
Trainable params: 1442178 (5.50 MB)
Non-trainable params: 48 (192.00 Byte)

$224 \times 224 \times 3$ pixels



4 Conv layers + 34 neurons

- Use of spacial information to (auto) extract features inside the image.
- A convolutional neural networks requires
  $(3\times5\times5+1+4)\times8+(8\times5\times5+1+4)\times16+(53\times53\times16+1)\times32+(32+1)\times2$
  parameters, i.e. $1.4 \cdot 10^6$ parameters (mostly in the dense layer)

Image classification with embedded AI
Computational/Memory complexity

Network ← neural network with initial weights
while not converged **do**
    BACKPROP-ITER(E, Network)

Problem :

- slow, requires the derivatives
- gradient computation is costly and increases with
  - number of weight $|w|$
  - number of examples $|E|$

  $\Longrightarrow \quad O(|w| \times |E|)$

Solution : *(Stochastic/mini-batch gradient descent)* :
select a small subset of example on which to propagate the error

Network ← neural network with initial weights
while not converged **do**
    MiniBatch ← *sample*(E, k)
    BACKPROP-ITER(*MiniBatch*, Network)

Image classification with embedded AI
Computational/Memory complexity

Complexity :

- NN : $\mathcal{O}(|w| \times |E|)$ with $|E|$ number of examples and $|w|$ number of weights
- CNN :
  - conv. layers :$\mathcal{O}(|E| \times (px \times f^2 \times k))$, with $|E|/px$ number/size of images, f/k size/number of filters in the layer,
  - dense layer :$\mathcal{O}(|E| \times |\tilde{w}|))$, with $|\tilde{w}| \ll |w|$ weights.

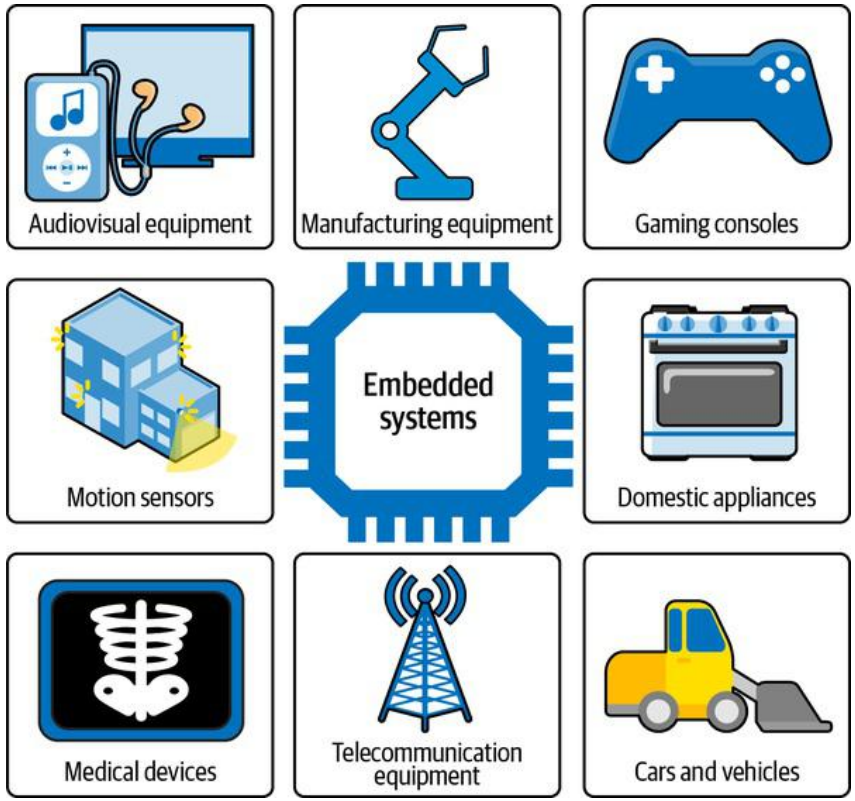
For a same application, we have generally that $Complexity(NN) > Complexity(CNN)$ because :

- the dense layers work on reduced (but enriched) data,
- each image is (kind of) used for several gradient updates of filter coefficients (several applications of the convolution).

Image classification with embedded AI
Frugal AI / Tiny ML

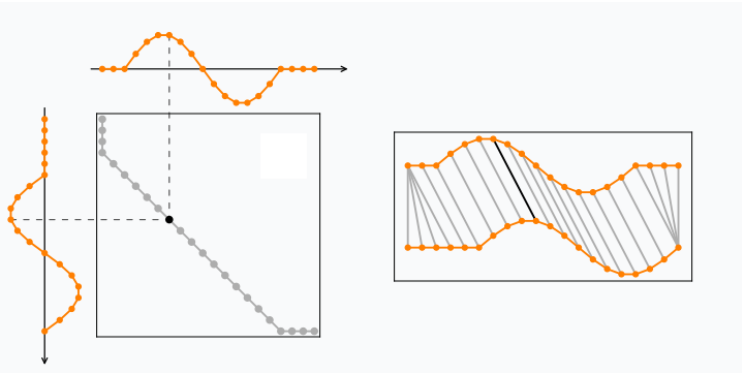
Constraints

- Low resources : memory, computations, instructions, etc.
- digital signal processing,
- data transmission.



Question : what can we do to decrease the complexity of CNN to make embedding on device possible ?

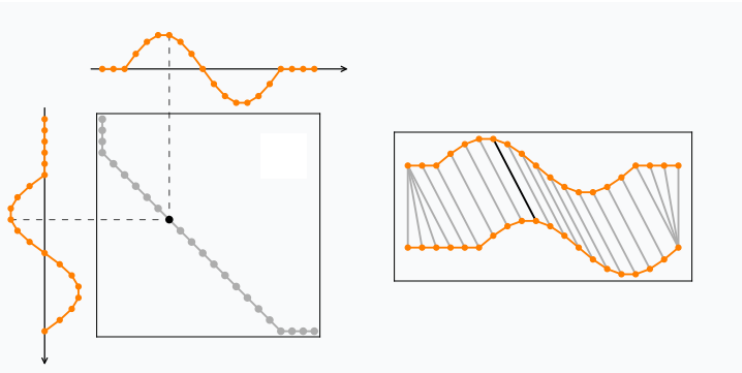
Image classification with embedded AI
Frugal AI example : DTW + k-NN



Complexity :

- DTW similarity matrix : $\mathcal{O}(|E| * n^2)$ with n dimension of time series

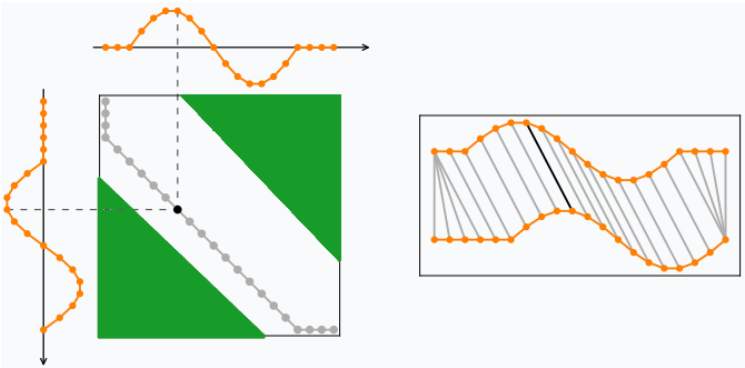
Image classification with embedded AI
Frugal AI example : DTW + k-NN



Complexity :

- DTW similarity matrix : $\mathcal{O}(|E| * n^2)$ with n dimension of time series
  $\Longrightarrow$ decrease size of times series ? E.g. decrease frequency but...

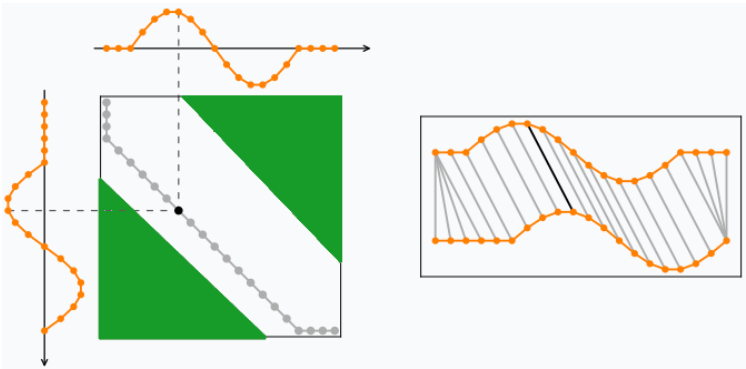
Image classification with embedded AI
Frugal AI example : DTW + k-NN



Complexity :

- DTW similarity matrix : $\mathcal{O}(|E| * n^2)$ with n dimension of time series
  $\Longrightarrow$ decrease size of times series ? E.g. decrease frequency but...
  $\Longrightarrow$ with global constraint $\mathcal{O}(|E| * B * n)$ with B the constraint band size

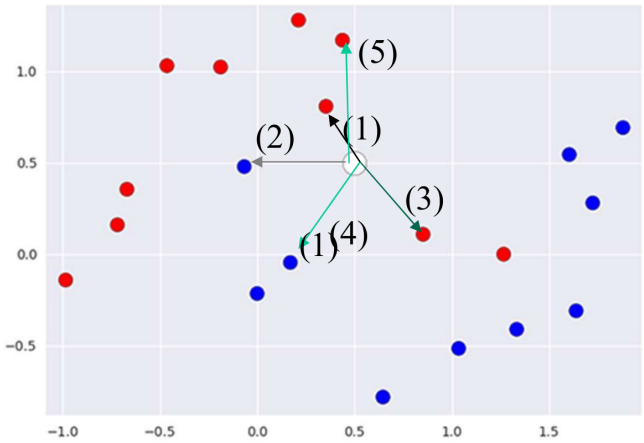
Image classification with embedded AI
Frugal AI example : DTW + k-NN



Complexity :

- DTW similarity matrix : $\mathcal{O}(|E| * n^2)$ with n dimension of time series
  $\Longrightarrow$ decrease size of times series ? E.g. decrease frequency but...
  $\Longrightarrow$ with global constraint $\mathcal{O}(|E| * B * n)$ with B the constraint band size
- k-NN : $\mathcal{O}(|E|log(|E|)$ (sort distances)

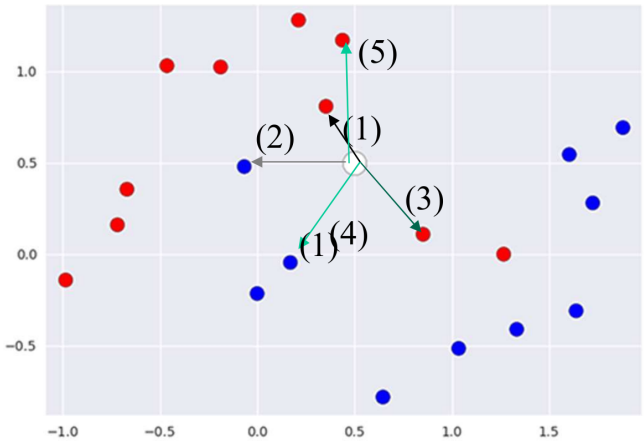
Image classification with embedded AI
Frugal AI example : DTW + k-NN

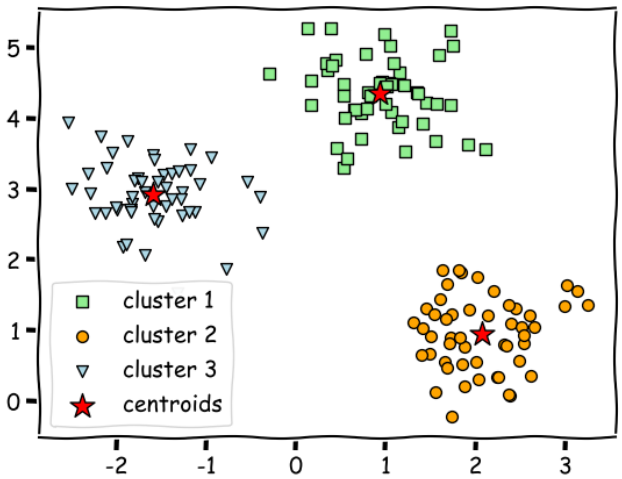


Complexity :

- DTW similarity matrix : $\mathcal{O}(|E| * n^2)$ with n dimension of time series
  $\Longrightarrow$ decrease size of times series ? E.g. decrease frequency but...
  $\Longrightarrow$ with global constraint $\mathcal{O}(|E| * B * n)$ with B the constraint band size
- k-NN : $\mathcal{O}(|E|log(|E|)$ (sort distances)
  $\Longrightarrow$ select best/mean profile(s) from training set for each class !

Image classification with embedded AI
Frugal AI example : DTW + k-NN





Complexity :

- DTW similarity matrix : $\mathcal{O}(|E| * n^2)$ with n dimension of time series
  $\Longrightarrow$ decrease size of times series ? E.g. decrease frequency but...
  $\Longrightarrow$ with global constraint $\mathcal{O}(|E| * B * n)$ with B the constraint band size
- k-NN : $\mathcal{O}(|E|log(|E|)$ (sort distances)
  $\Longrightarrow$ select best/mean profile(s) from training set for each class !

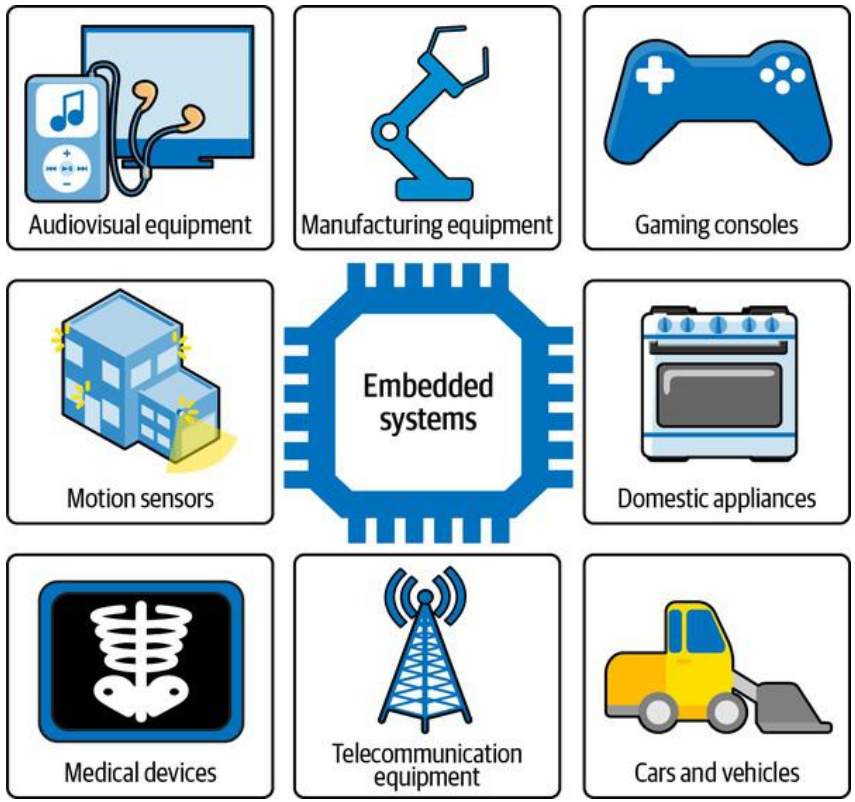
Image classification with embedded AI
Optimization of CNN

Constraints

- Low resources : memory, computations, instructions, etc.
- digital signal processing,
- data transmission.

Complexity of CNN

- Number of images : $|E|$,
- Size of images : px,
- Number of weights : $|w|$,
- Size/number of filters : (f, k),
- Network architecture.



Question : what can we do to decrease the complexity of CNN to make embedding on device possible ?

Section 1

Optimization of CNN

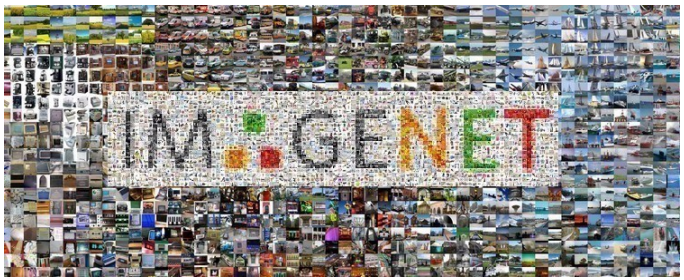
Image classification with embedded AI
Frugal AI example : DTW + k-NN

Complexity :

- conv. layers :$\mathcal{O}(|E| \times (px \times f^2 \times k))$, with $|E|/px$ number/size of images, f/k size/number of filters in the layer,
- dense layer :$\mathcal{O}(|E| \times |\tilde{w}|)$, with $|\tilde{w}|$ weights.

Optimization/Compression ?

Image classification with embedded AI
Frugal AI example : DTW + k-NN



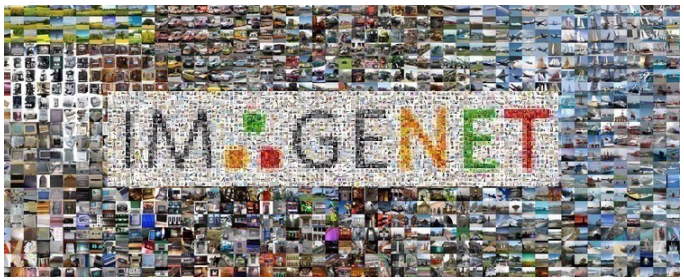
Complexity :

- conv. layers :$\mathcal{O}(|E| \times (\cancel{px} \times f^2 \times k))$, with $|E|/px$ number/size of images, f/k size/number of filters in the layer,
- dense layer :$\mathcal{O}(|E| \times |\tilde{w}|)$, with $|\tilde{w}|$ weights.

Optimization/Compression ?

- Dimension reduction :
  - decrease px : dimension reduction

Image classification with embedded AI
Frugal AI example : DTW + k-NN



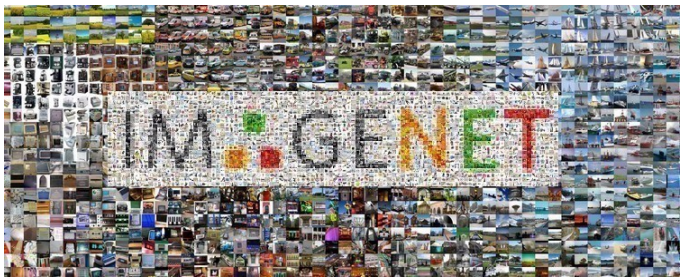
Complexity :

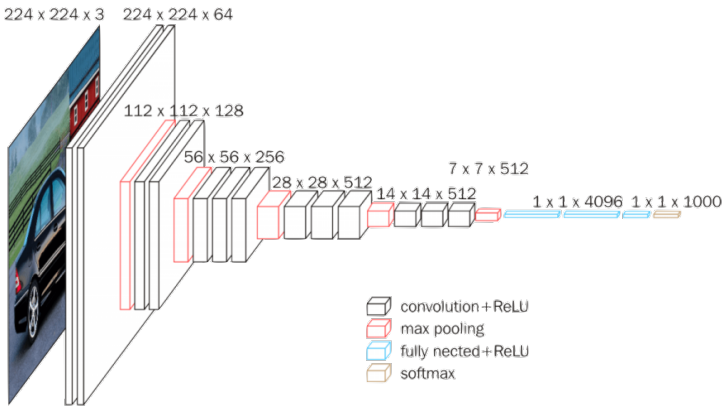
- conv. layers :$\mathcal{O}(\cancel{|E|} \times (\cancel{px} \times f^2 \times k))$, with $|E|/px$ number/size of images, f/k size/number of filters in the layer,
- dense layer :$\mathcal{O}(\cancel{|E|} \times |\tilde{w}|))$, with $|\tilde{w}|$ weights.

Optimization/Compression ?

- Dimension reduction :
  - decrease px : dimension reduction
  - decrease $|E|$: **stocchastic** gradient descent

Image classification with embedded AI
Frugal AI example : DTW + k-NN





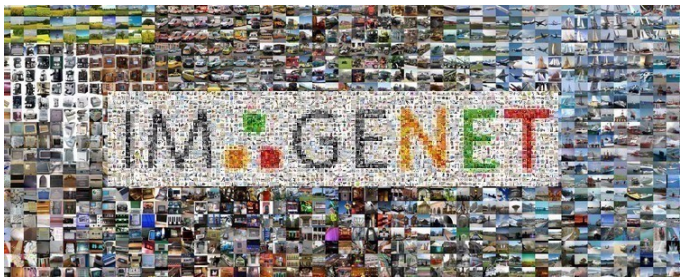
Complexity :

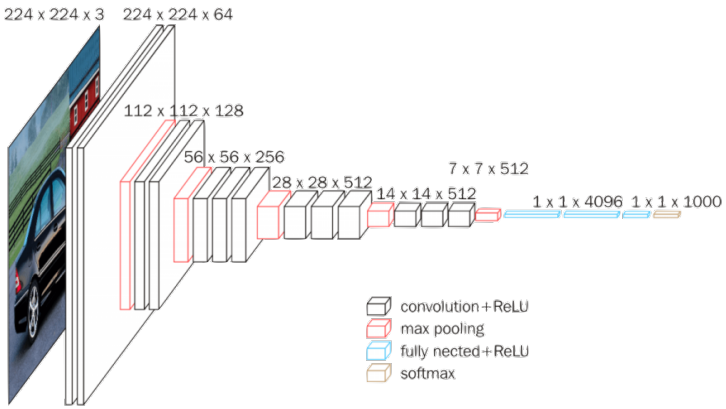
- conv. layers :$\mathcal{O}(\cancel{|E|} \times (\cancel{px} \times \cancel{f}^2 \times \cancel{k}))$, with $|E|/px$ number/size of images, f/k size/number of filters in the layer,
- dense layer :$\mathcal{O}(\cancel{|E|} \times \cancel{|\tilde{w}|})$, with $|\tilde{w}|$ weights.

Optimization/Compression ?

- Dimension reduction :
  - decrease px : dimension reduction
  - decrease $|E|$: **stocchastic** gradient descent
- then decrease... f, k, $\tilde{w}$?

Image classification with embedded AI
Frugal AI example : DTW + k-NN





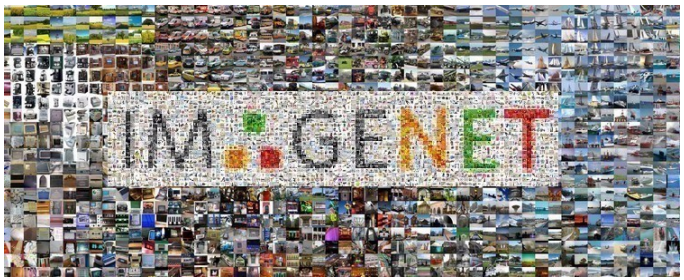
Complexity :

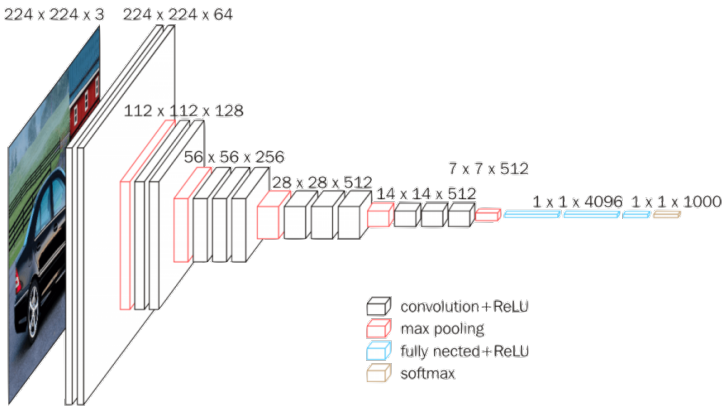
- conv. layers : $\cancel{\mathcal{O}}(\cancel{|E|} \times (\cancel{px} \times \cancel{f^2} \times \cancel{k}))$, with $|E|/px$ number/size of images, f/k size/number of filters in the layer,
- dense layer : $\cancel{\mathcal{O}}(\cancel{|E|} \times \cancel{|\tilde{w}|})$, with $|\tilde{w}|$ weights.

Optimization/Compression ?

- Dimension reduction :
  - decrease px : dimension reduction
  - decrease $|E|$: **stocchastic** gradient descent
- then decrease... f, k, $\tilde{w}$? $\mathcal{O}$ (flops/MB) ?

Image classification with embedded AI
Frugal AI example : DTW + k-NN





Complexity :

- conv. layers : $\mathcal{O}(|E| \times (px \times f^2 \times k))$, with $|E|/px$ number/size of images, f/k size/number of filters in the layer,
- dense layer : $\mathcal{O}(|E| \times |\tilde{w}|)$, with $|\tilde{w}|$ weights.

Optimization/Compression ?

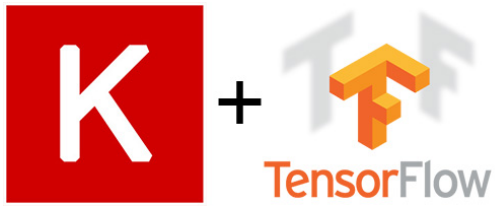
- Dimension reduction :
  - decrease px : dimension reduction
  - decrease $|E|$: **stocchastic** gradient descent
- then decrease... f, k, $\tilde{w}$? $\mathcal{O}$ (flops/MB) ?
- Today's lecture (and part of next) : techniques for CNN optimization/compression *(and NN)*

CNN optimisation
Parenthesis : libraries

Do you know the difference between Tensorflow, Keras, scikit-learn, Pytorch ?


CNN optimisation
Parenthesis : libraries

Do you know the difference between Tensorflow, Keras, scikit-learn, Pytorch ?

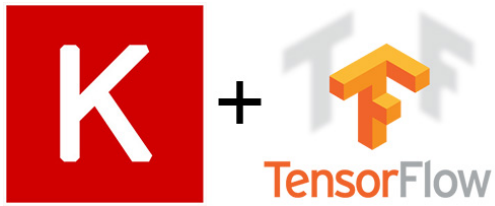


All are Python libraries for machine learning !

- *scikit-learn* : basic methods, VERY user friendly !
- **Tensorflow** : advanced, CPU+GPU, deep learning
- **Keras** : high level layer over Tensorflow, user friendly !
- Pytorch : advanced, CPU+GPU, deep learning (complicated...)

CNN optimisation
Parenthesis : libraries

Do you know the difference between Tensorflow, Keras, scikit-learn, Pytorch ?

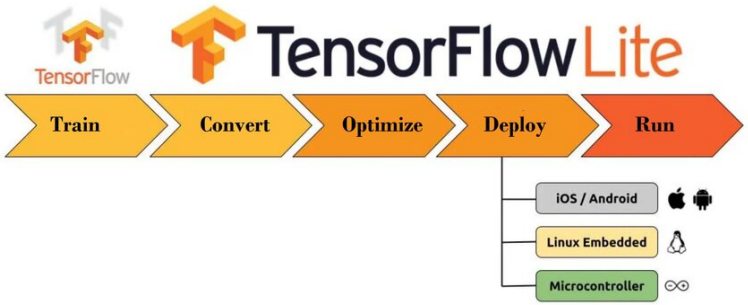


All are Python libraries for machine learning !

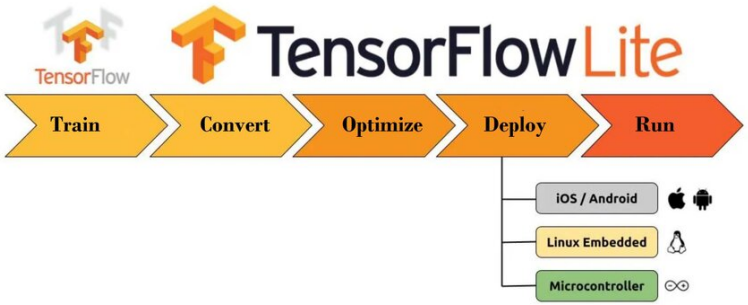
- *scikit-learn* : basic methods, VERY user friendly ! [Lab 1]
- **Tensorflow** : advanced, CPU+GPU, deep learning [Lab 2]
- **Keras** : high level layer over Tensorflow, user friendly ! [Lab 2]
- Pytorch : advanced, CPU+GPU, deep learning (complicated...)

What will we use in labs ?
And what about embedded AI ?

CNN optimization
Tensorflow Lite



CNN optimization
Tensorflow Lite

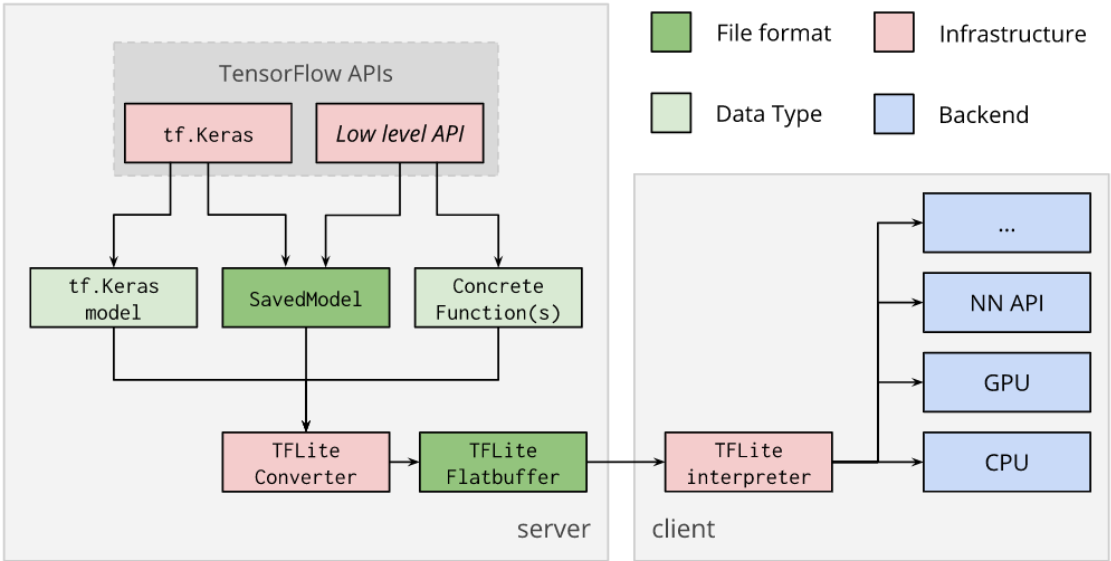


2 steps ($\frac{1}{2}$) :

1. TF : better model

(1.5. TF : compress)

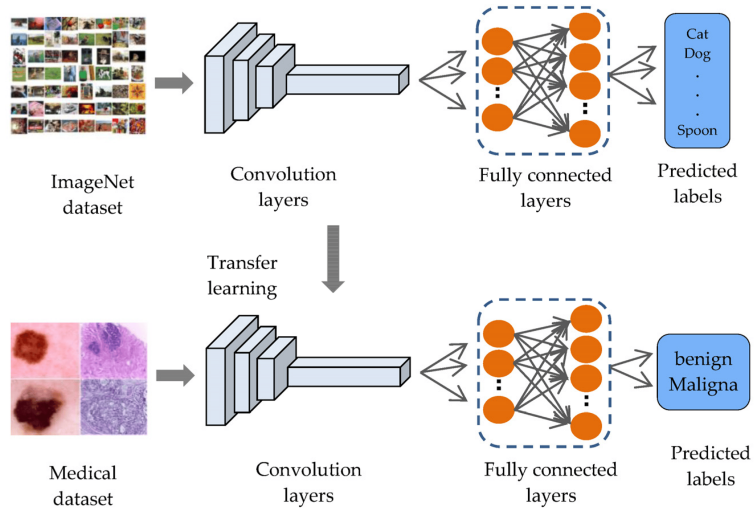
2. TFLite :
   convert $\supset$ optimize



CNN optimization
1) Tensorflow : better model

Classical steps combined with validation approach (e.g. k-folds) :

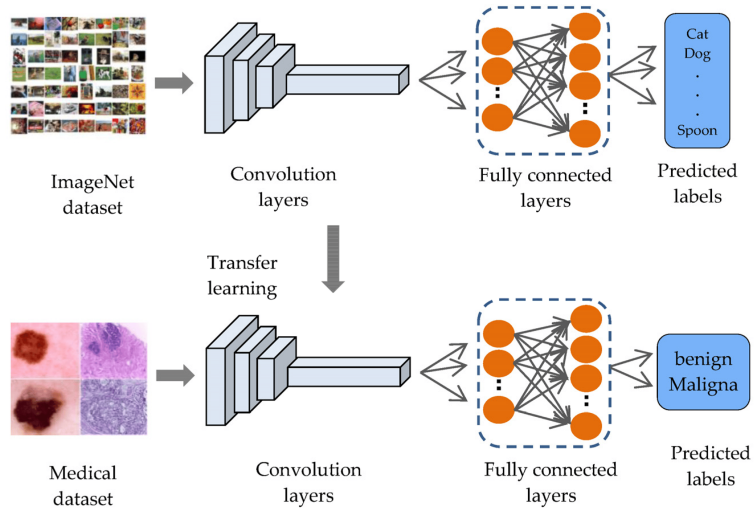
1. Data preprocessing
2. Finding hyperparameters ?



CNN optimization
1) Tensorflow : better model

Classical steps combined with validation approach (e.g. k-folds) :

1. Data preprocessing
2. Finding hyperparameters ?
   - Structure : depth/width/resolution, convolutions, ···
   - Algorithm : activation, update algorithm, ···
   - Stopping criteria

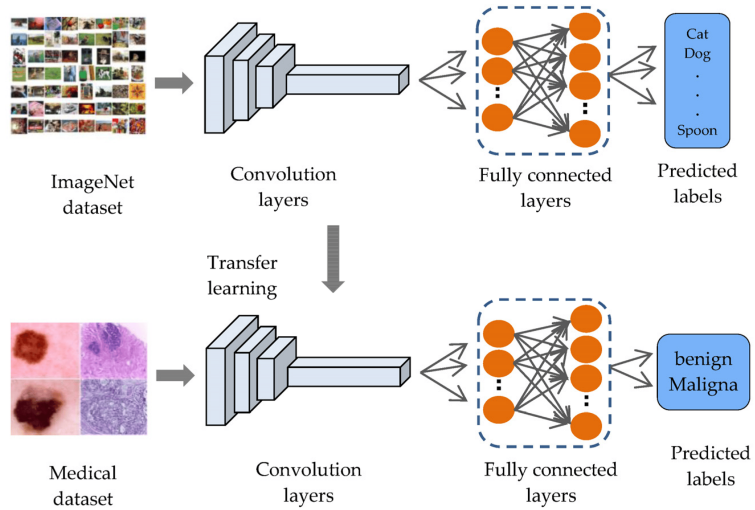


CNN optimization
1) Tensorflow : better model

Classical steps combined with validation approach (e.g. k-folds) :

1. Data preprocessing
2. Finding hyperparameters ? (Not the subject today, Lab1)
   - Structure : depth/width/resolution, convolutions, ···
   - Algorithm : activation, update algorithm, ···
   - Stopping criteria

2 Transfer learning ?

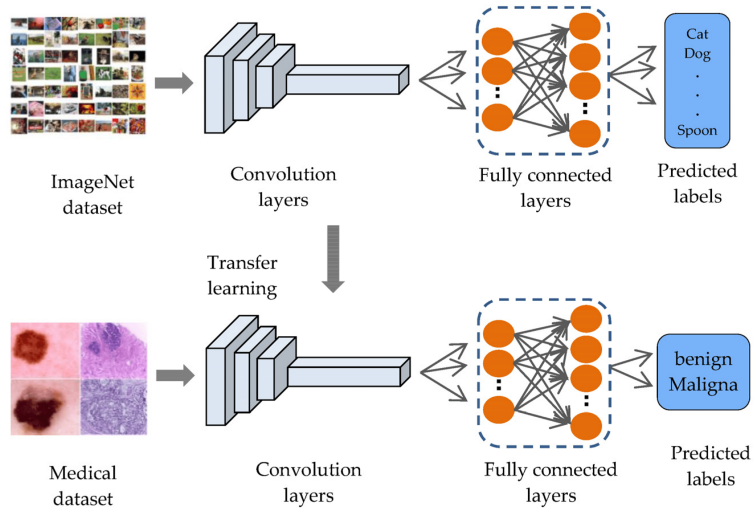


CNN optimization
1) Tensorflow : better model

Classical steps combined with validation approach (e.g. k-folds) :

1. Data preprocessing
2. Finding hyperparameters ?
   - Structure : depth/width/resolution, convolutions, ···
   - Algorithm : activation, update algorithm, ···
   - Stopping criteria

2 Transfer learning ?
- Load weights and structure (efficient CNN)
- Reuse conv. + train dense
- (Opt. : retrain last conv.)

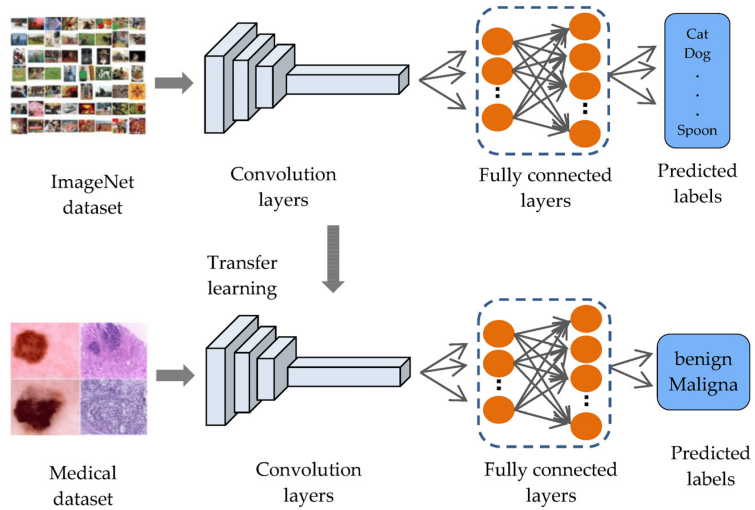


CNN optimization
1) Tensorflow : better model

Classical steps combined with validation approach (e.g. k-folds) :

1. Data preprocessing
2. Finding hyperparameters ?
   - Structure : depth/width/resolution, convolutions, ···
   - Algorithm : activation, update algorithm, ···
   - Stopping criteria

2 Transfer learning ?
- Load weights and structure (efficient CNN)
- Reuse conv. + train dense
- (Opt. : retrain last conv.)



$\Longrightarrow$ Now, we consider that we know how to have a model performing well and want to optimize its computation.

CNN optimization
2) compression/optimization

Several mechanisms to decrease the computational and memory load of a model :

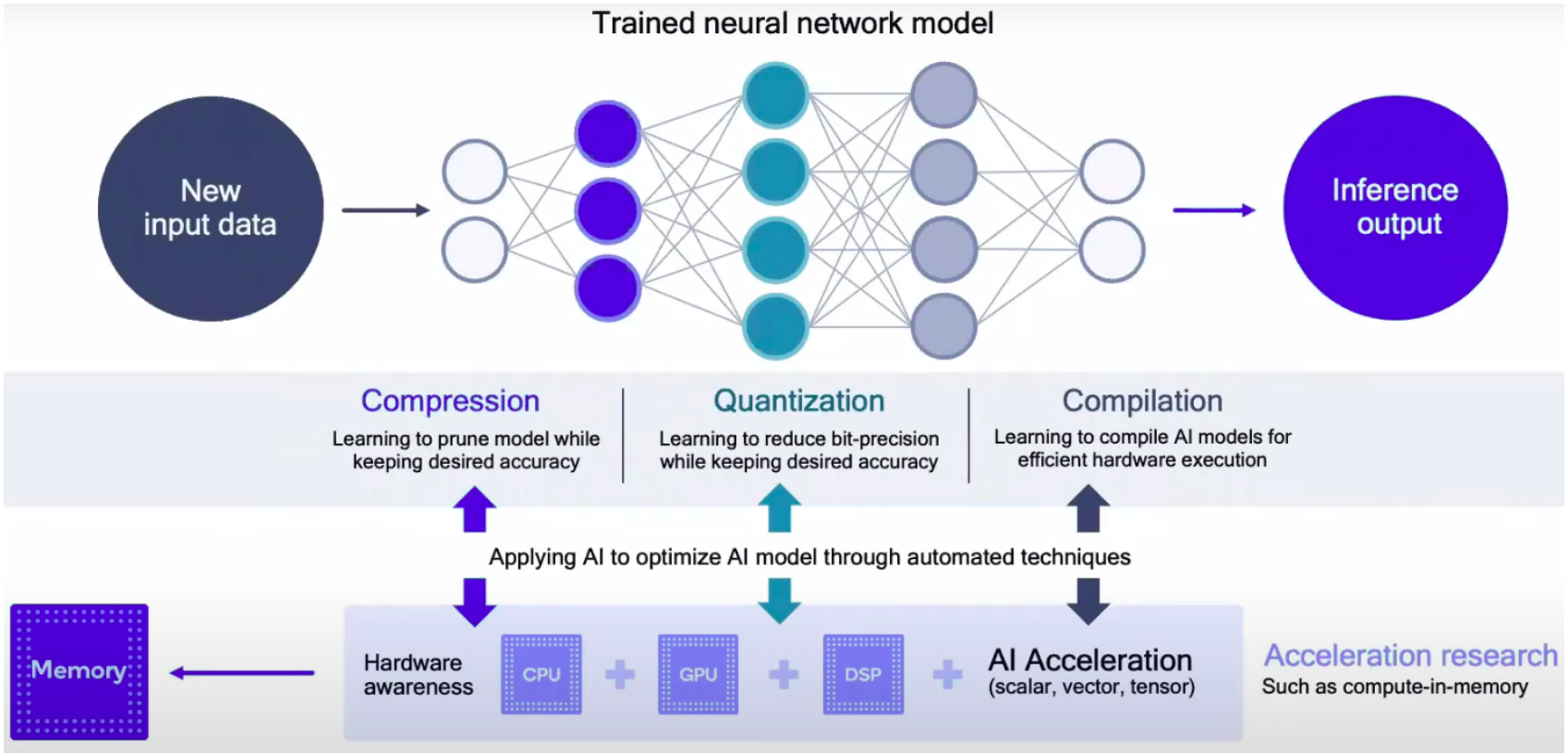
1. *(TF)* Sparse Representations
2. *(TF)* Knowledge Distillation
3. *(TF)* Efficient Architectures (performance vs. resources)
   - MobileNet, EfficientNet, etc.
   - Other improvements : weight sharing, layer factorization, low-rank Approximations
4. *(TF)* Model pruning
5. *(TF/TFLite)* Quantization
6. *(TFLite)* Batch Normalization Fusion (conversion)
7. *(TFLite)* Compiler/Hardware-aware optimizations (conversion)

CNN optimization
2) compression/optimization

Several mechanisms to decrease the computational and memory load of a model :

1. *(TF)* Sparse Representations
2. *(TF)* Knowledge Distillation
3. *(TF)* Efficient Architectures (performance vs. resources)
   - MobileNet, EfficientNet, etc.
   - Other improvements : weight sharing, layer factorization, low-rank Approximations
4. *(TF)* Model pruning
5. *(TF/TFLite)* Quantization
6. *(TFLite)* Batch Normalization Fusion (conversion)
7. *(TFLite)* Compiler/Hardware-aware optimizations (conversion)

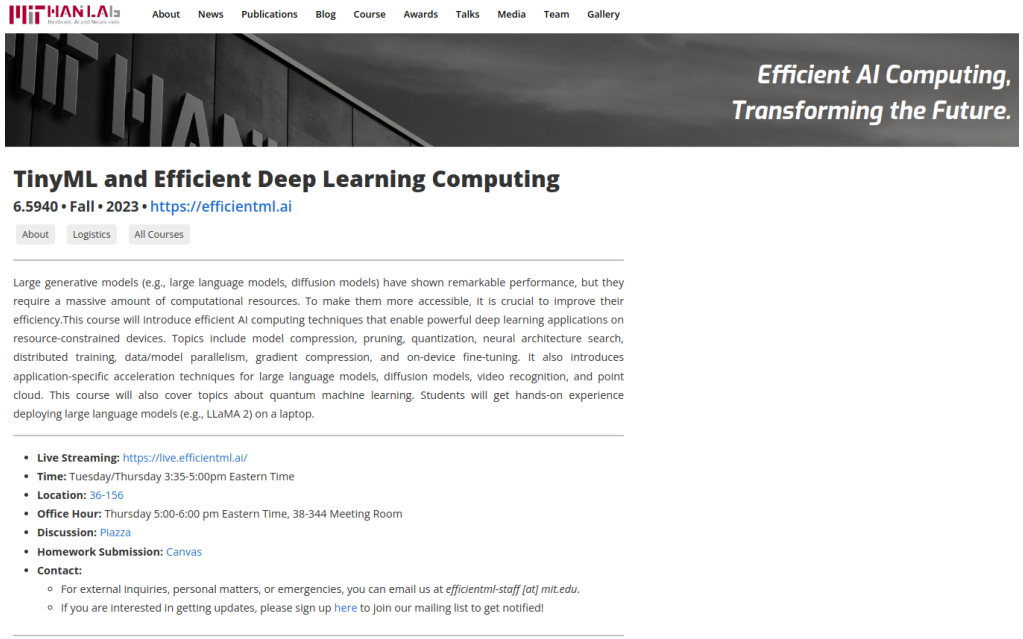
CNN optimization



$\Longrightarrow$ In the following, and in Lab2 !

CNN optimization
And a LOT more...

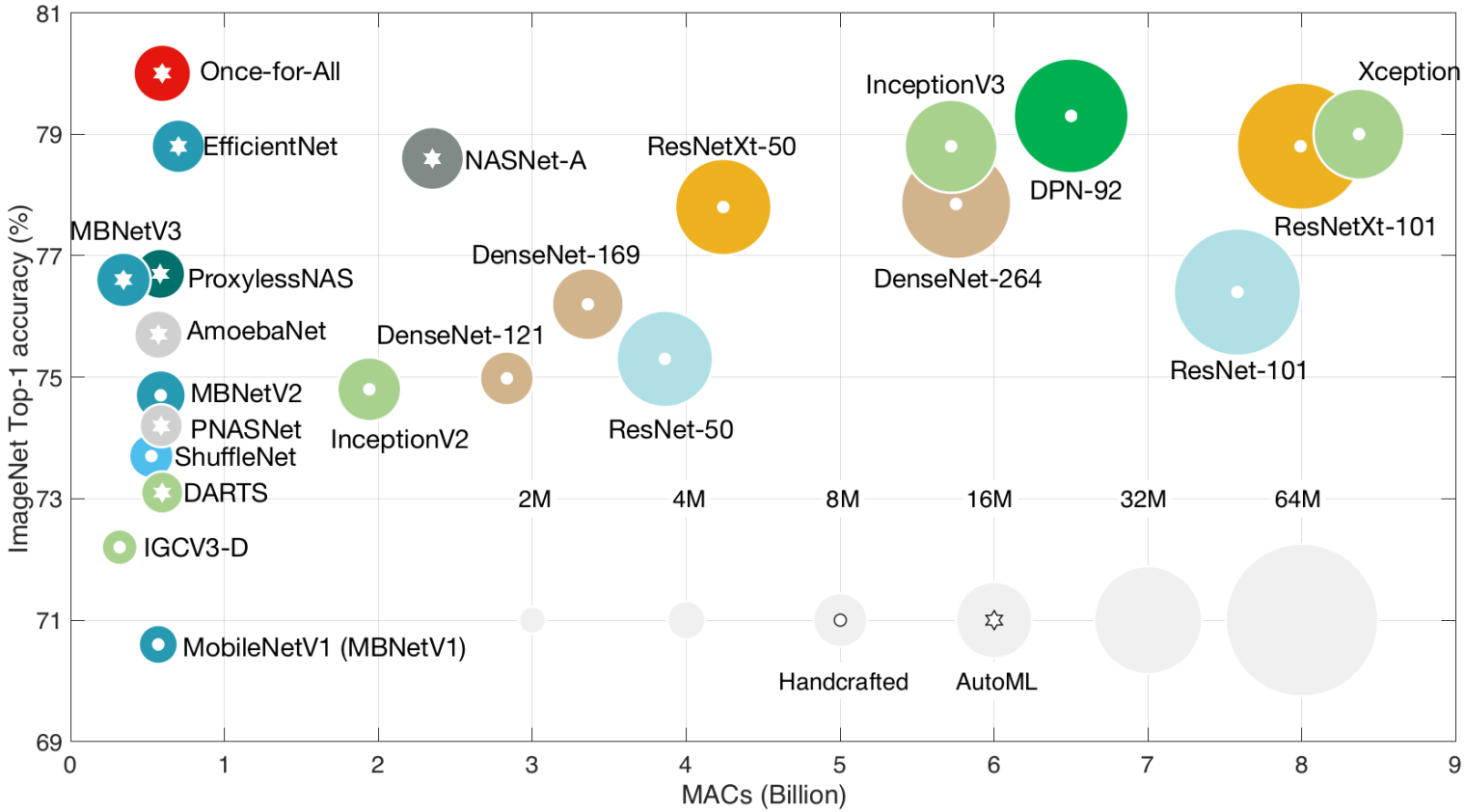
MIT Han Lab course (Fall 2023) – TinyML and Efficient Deep Learning Computing :
https://hanlab.mit.edu/courses/2023-fall-65940



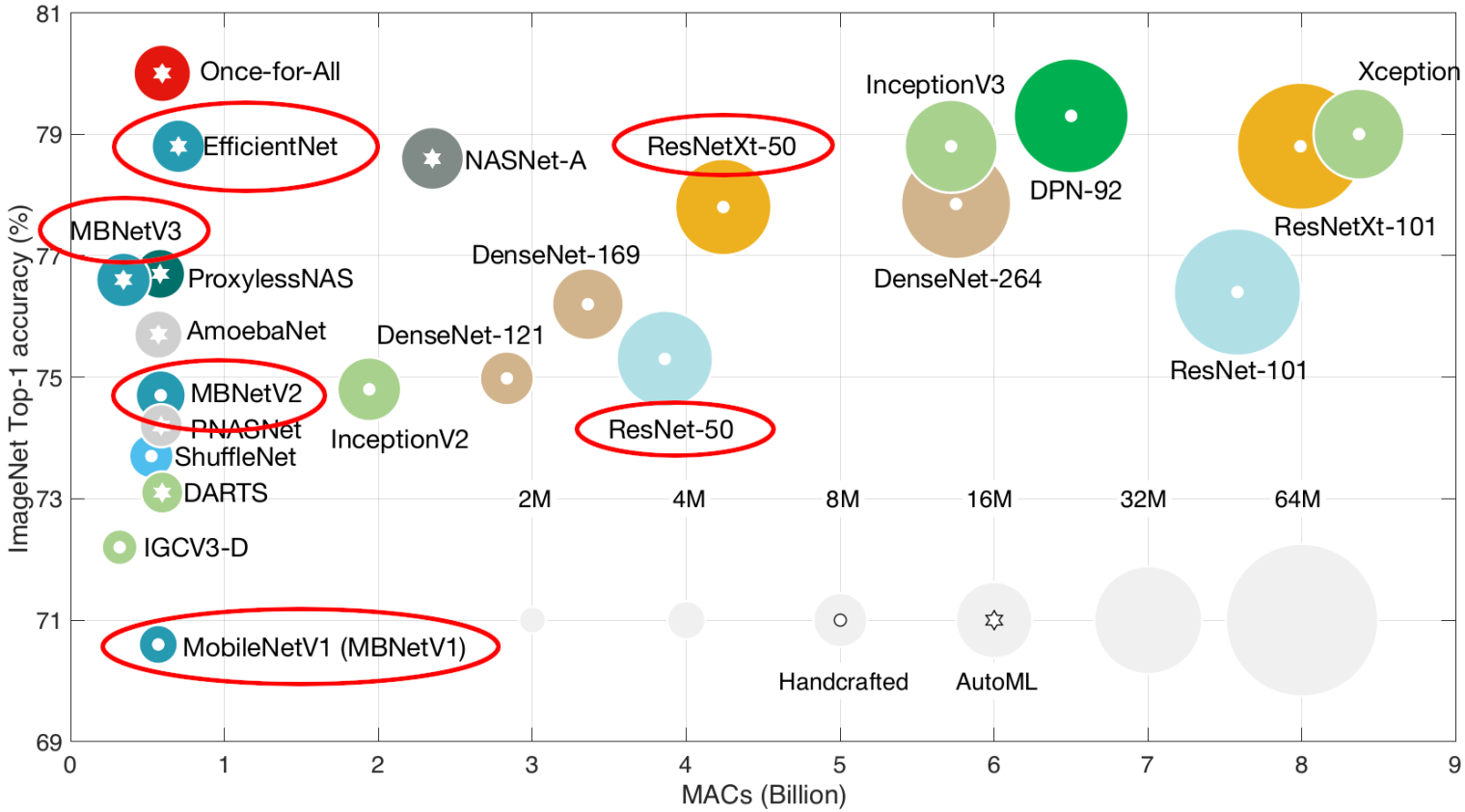
Section 2

Efficient CNN architectures

Efficient CNN architectures (transfer learning !)
A tradeoff : Accuracy vs. Cost



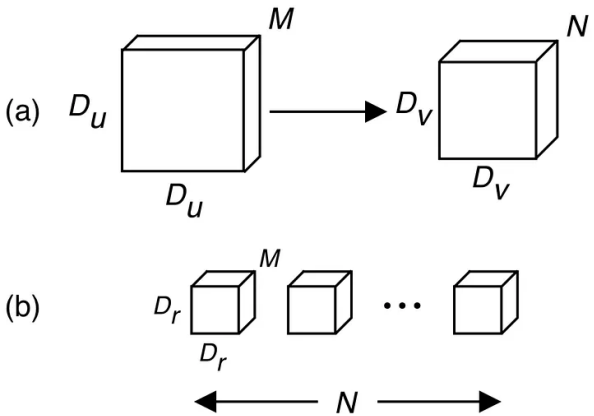
Efficient CNN architectures (transfer learning !)
A tradeoff : Accuracy vs. Cost

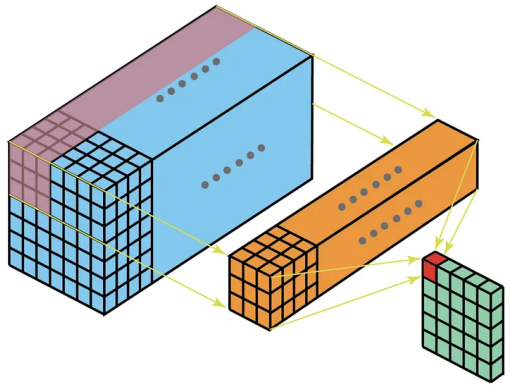


$\Longrightarrow$ Tons of optimization mechanisms through the recent history (2015-) :
2 aspects : improved convolution layers + Neural Architecture Search (NAS)

Improved convolution layers
Classical convolution

Given an input : $D_u \times D_u \times M$ (e.g. height, width, channels).



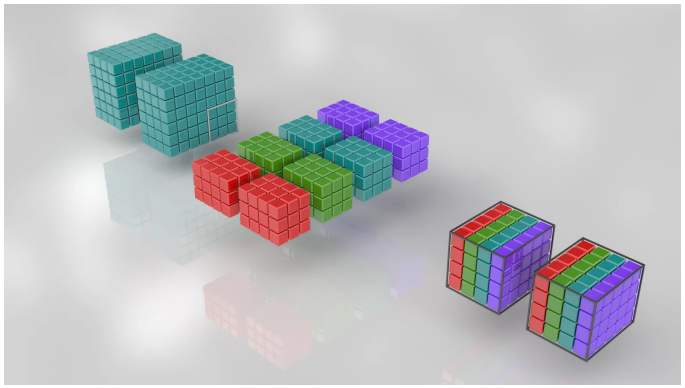


Computational cost :

- Classical convolution : $(O)(D_r^2 M D_v^2 N)$
-
-

Improved convolution layers
ResNeXt : grouped convolution

Given an input : $D_u \times D_u \times M$ (e.g. height, width, channels).



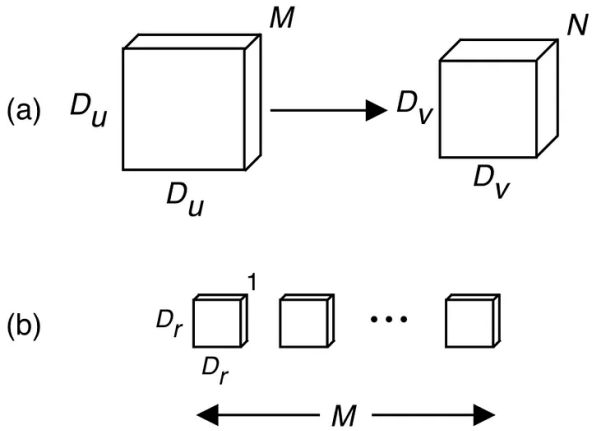
Computational cost :

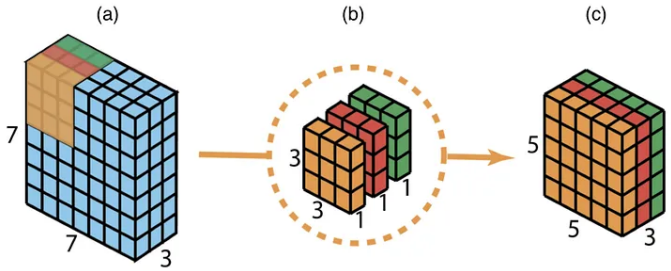
- Classical convolution : $(O)(D_r^2 M D_v^2 N)$
- Grouped convolution : $(O)(D_r^2 M D_v^2 N/g)$
-

Xie et al., *"Aggregated residual transformations for deep neural networks"*, 2017.

Improved convolution layers
Depth convolution

Given an input : $D_u \times D_u \times M$ (e.g. height, width, channels).



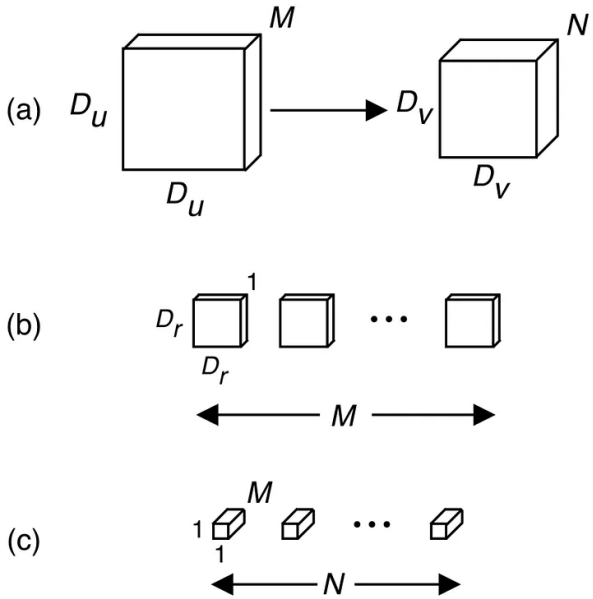


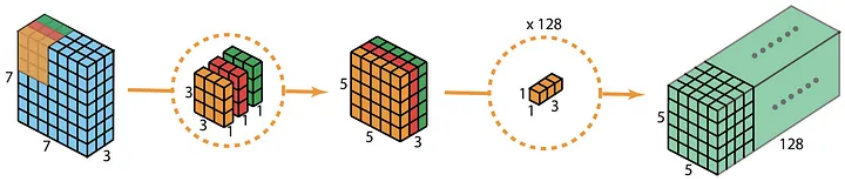
Computational cost :

- Classical convolution : $(O)(D_r^2 M D_v^2 N)$
- Grouped convolution : $(O)(D_r^2 M D_v^2 N/g)$
-

Improved convolution layers
MobileNet : depthwise-separable block

Given an input : $D_u \times D_u \times M$ (e.g. height, width, channels).





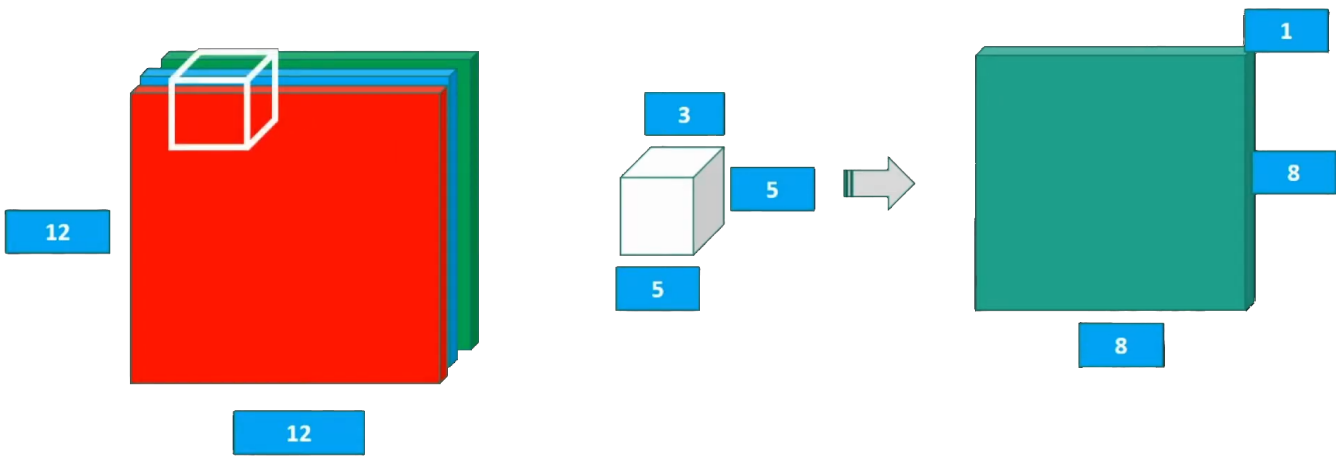
Computational cost :

- Classical convolution : $(O)(D_r^2 M D_v^2 N)$
- Grouped convolution : $(O)(D_r^2 M D_v^2 N/g)$
- Depthwise-separable convolution : $(O)(D_r^2 M D_v^2 + M D_v^2 N)$

Howard et al., *"Mobilenets: Efficient convolutional neural networks for mobile vision applications"*, 2017

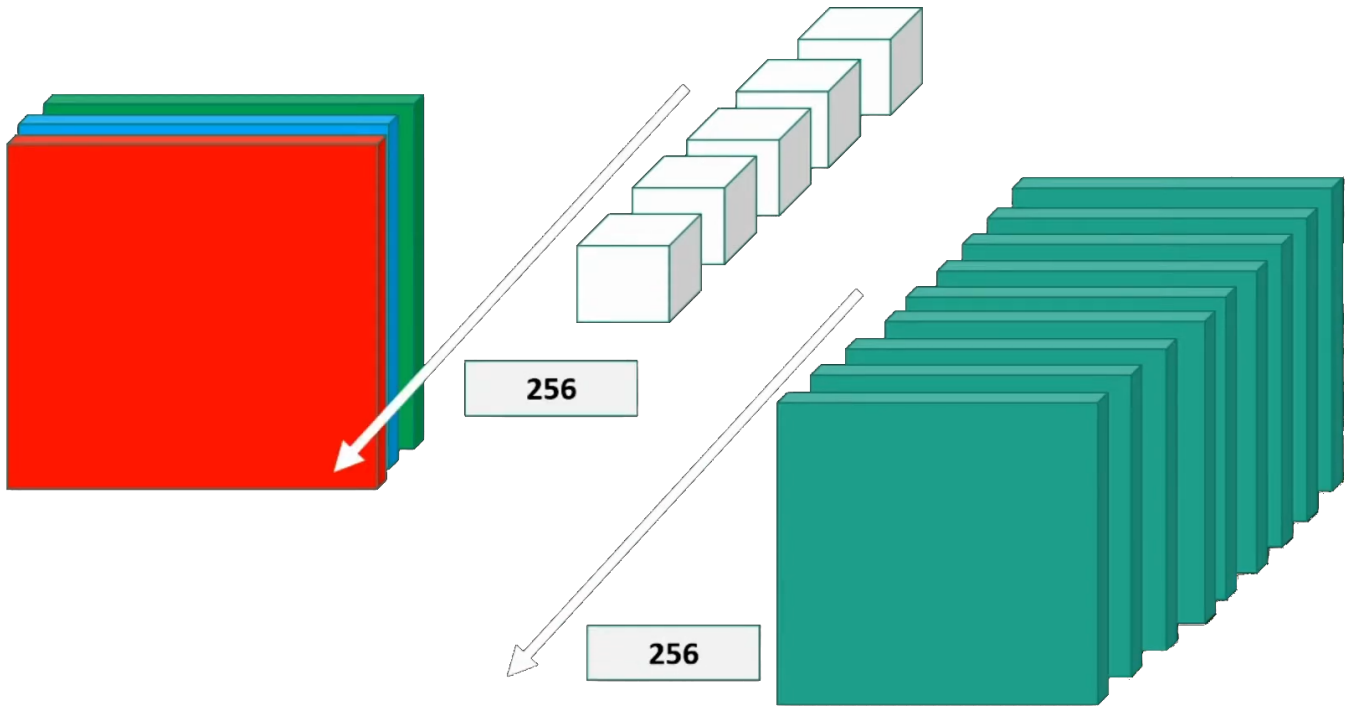
Improved convolution layers
Example : depthwise-separable convolution

$$D_r^2 M D_v^2 = 5^2 \times 3 \times 8^2$$

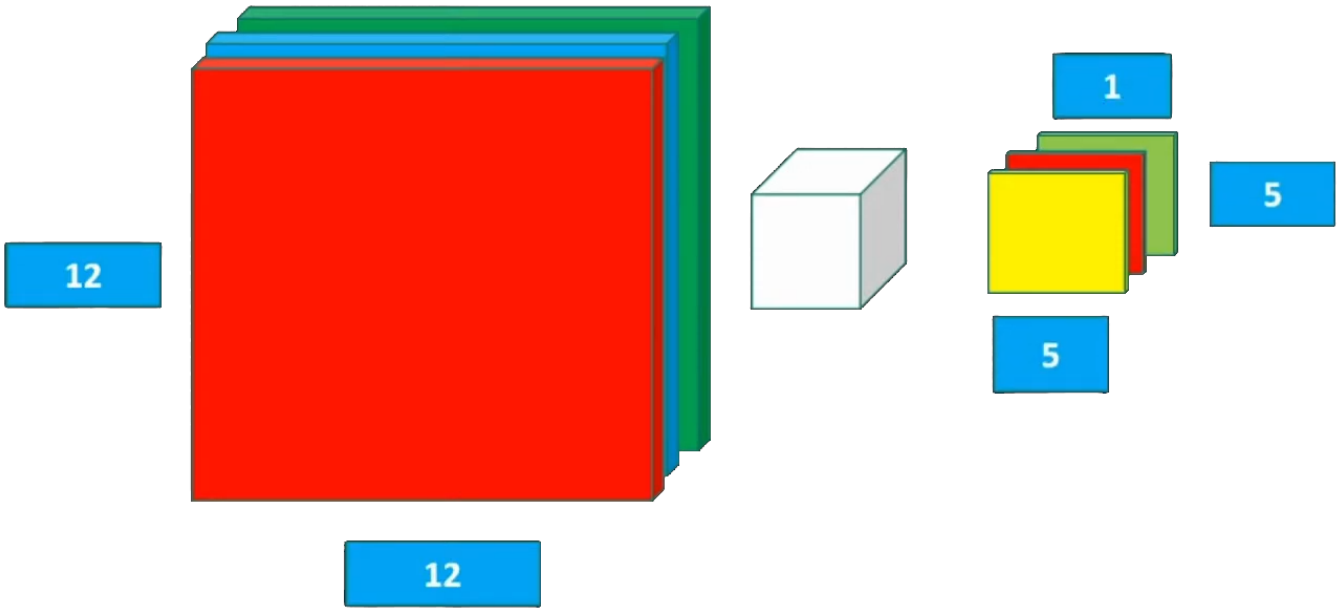


Improved convolution layers
Example : depthwise-separable convolution

$$D_r^2 M D_v^2 N = 5^2 \times 3 \times 8^2 \times 256 = 1.2 \cdot 10^6 ops$$

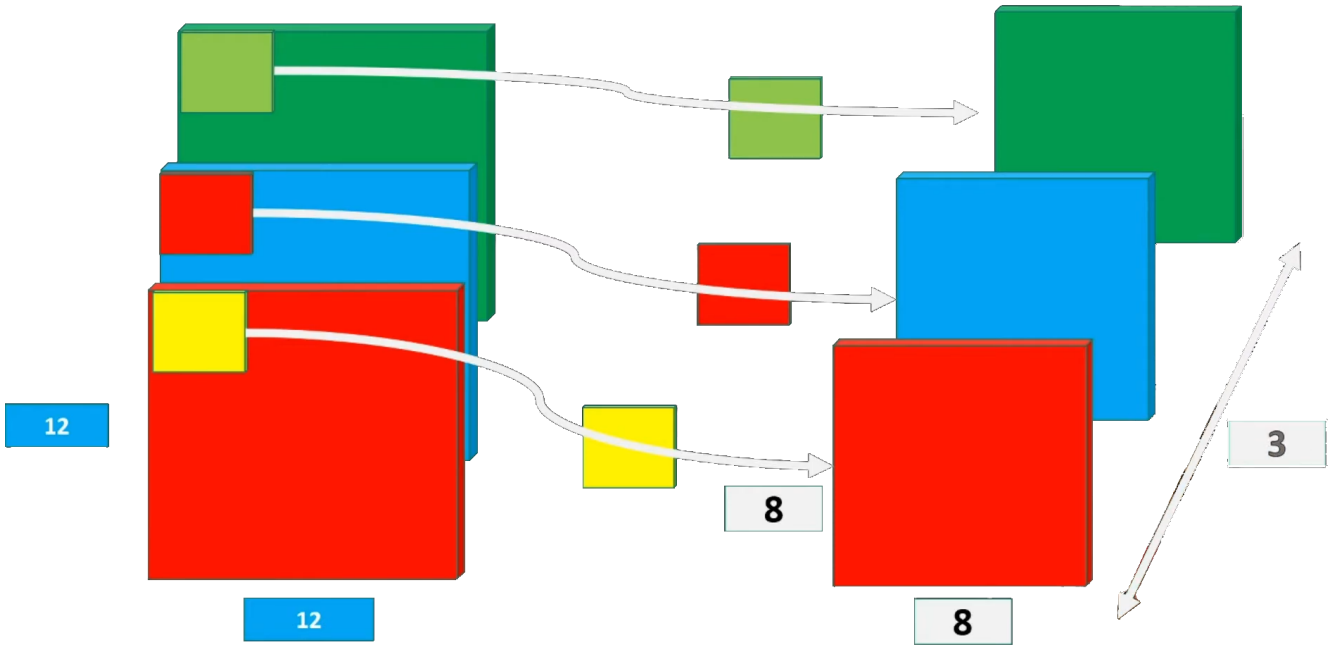


Improved convolution layers



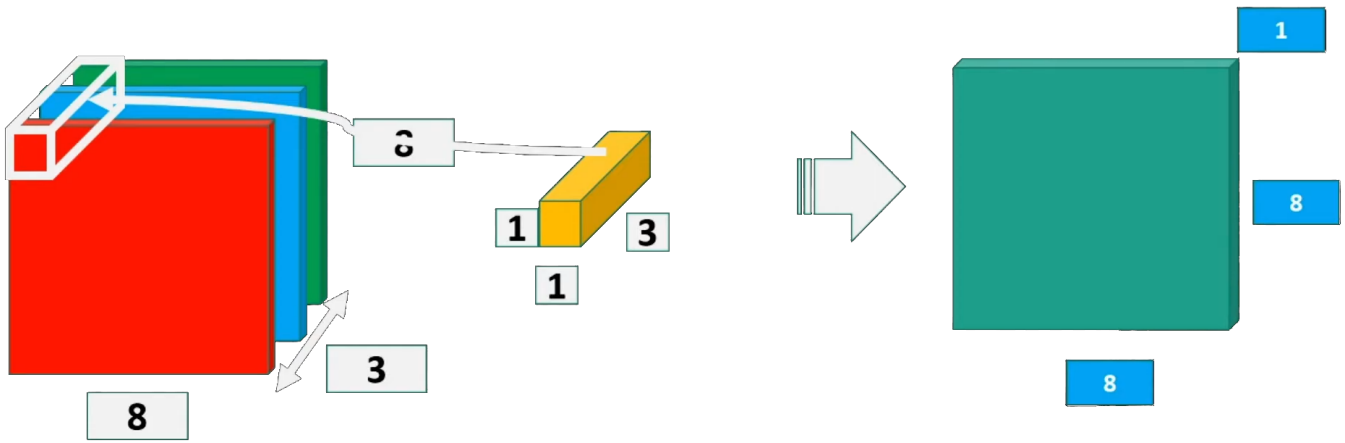
Improved convolution layers

$$D_r^2 M D_v^2 = 5^2 \times 3 \times 8^2$$



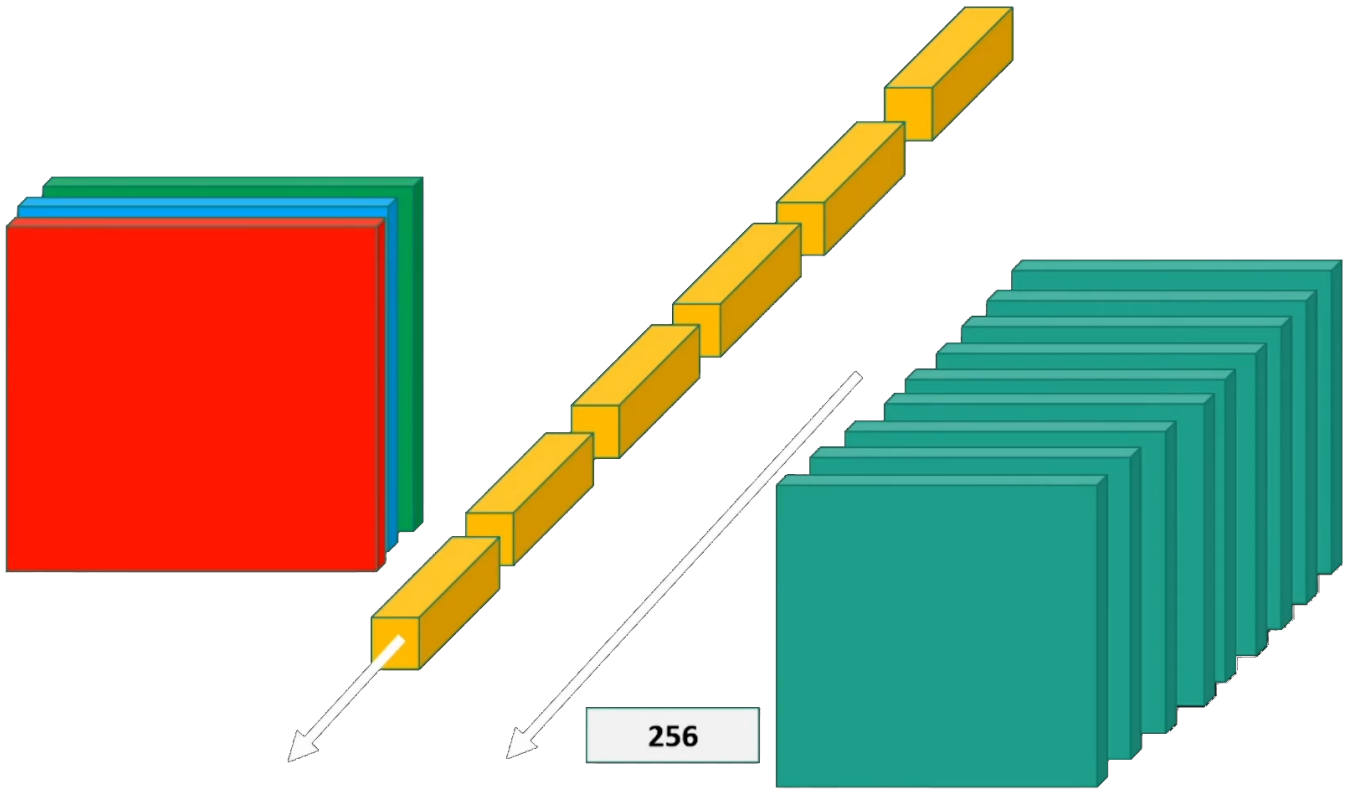
Improved convolution layers

$$D_r^2 MD_v^2 + MD_v^2 = 5^2 \times 3 \times 8^2 + 3 \times 8^2$$



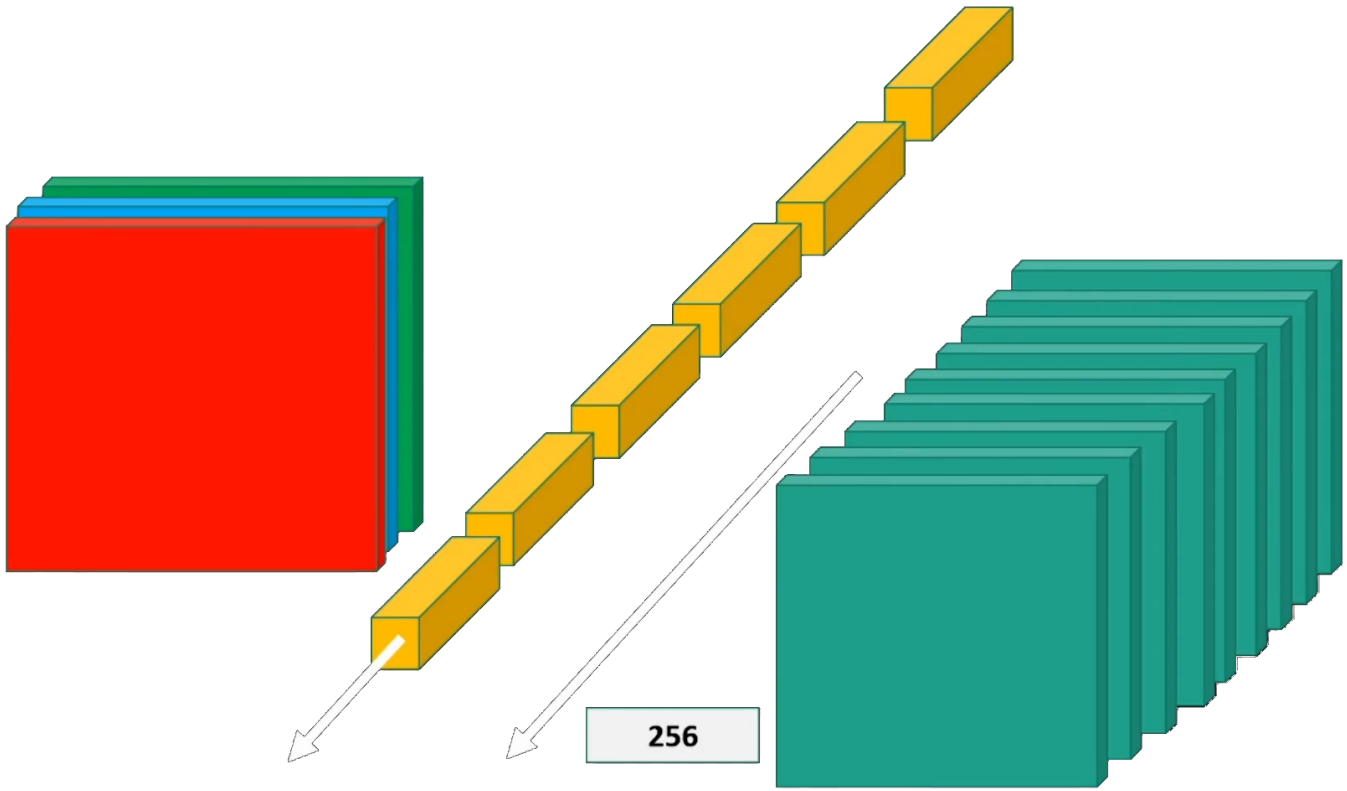
Improved convolution layers

$$D_r^2 M D_v^2 + M D_v^2 N = 5^2 \times 3 \times 8^2 + 3 \times 8^2 \times 256 = 5.4 \cdot 10^4 ops$$



Improved convolution layers

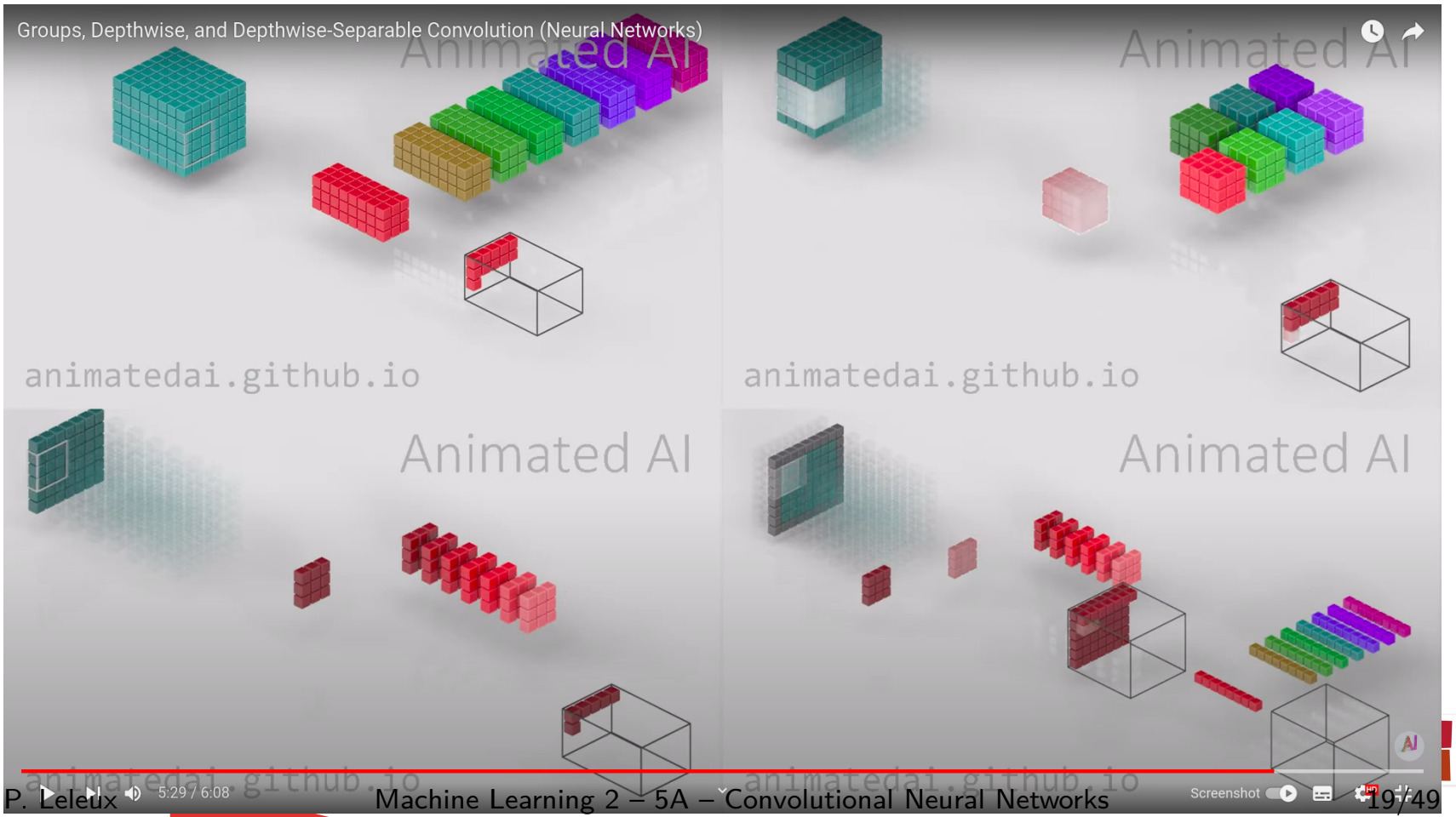
Summary : Classical conv. : $1.2 \cdot 10^6$ vs. $5.4 \cdot 10^4$: Depth-separable conv.



Improved convolution layers
classical vs. grouped vs. depthwise-separable convolution

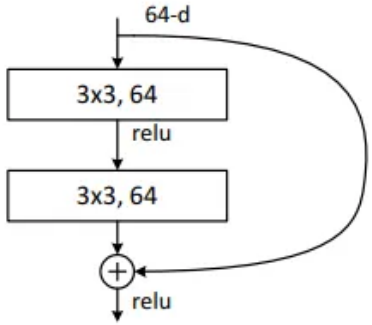
See the animation on YouTube :

https://www.youtube.com/watch?v=vVaRhZXovbw

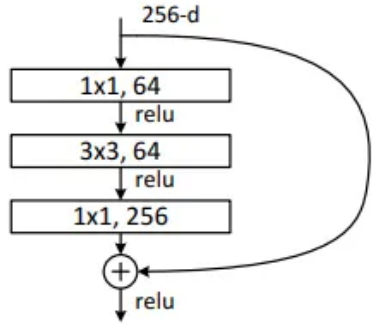


Improved convolution layers
ResNet50 : residual and bottleneck block

Residual block



Bottleneck block



Shortcuts between layers

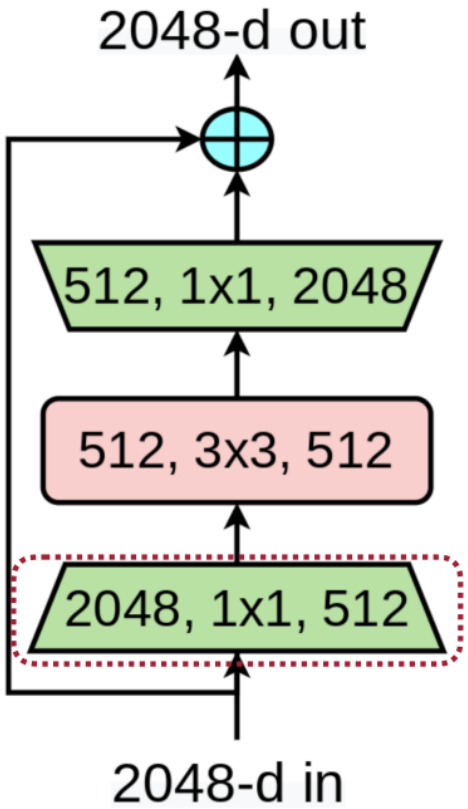
- More layers $\neq$ Higher accuracy,
- Vanishing Gradient Problem,
- Identity Mapping by Shortcuts : $\text{Layer}_{i+1} \geq \text{Layer}_i$,
- Encouraging Feature Reuse.

Reducing Computational Complexity

- Same advantages as residual block,
- Facilitating Deeper Networks.

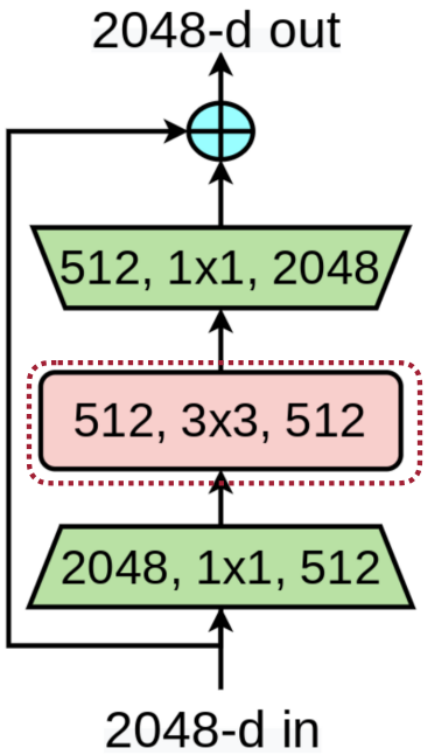
He et al., *"Deep residual learning for image recognition"*, 2016.

Improved convolution layers
Example : bottleneck block



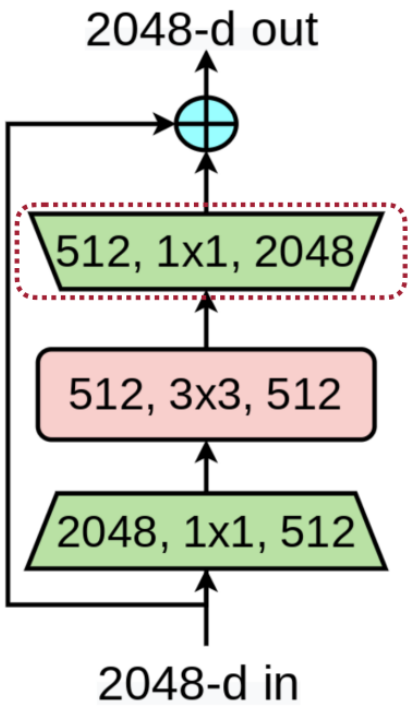
Reduce the number of channels by 4x
via 1x1 convolution

Improved convolution layers
Example : bottleneck block



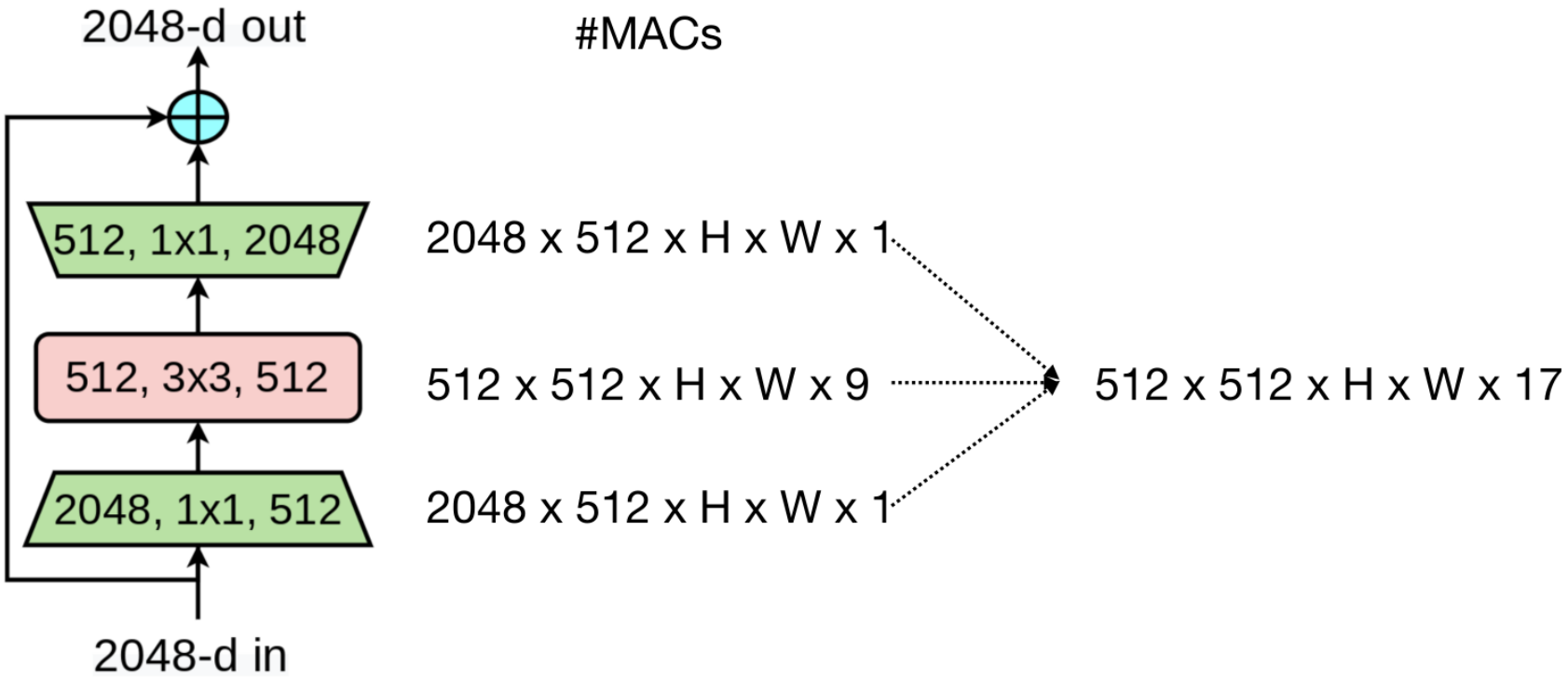
Feed reduced feature map to 3x3 convolution

Improved convolution layers
Example : bottleneck block

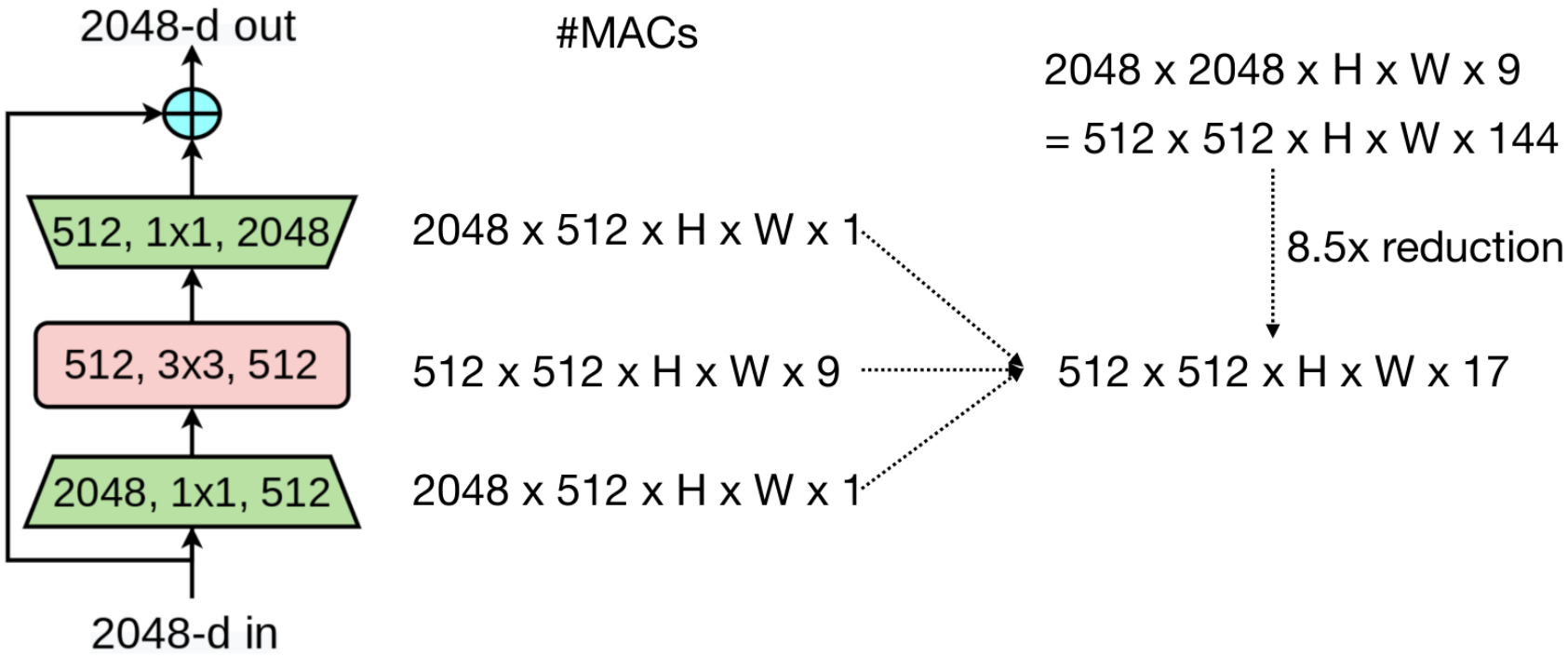


Expand the number of channels via 1x1 convolution

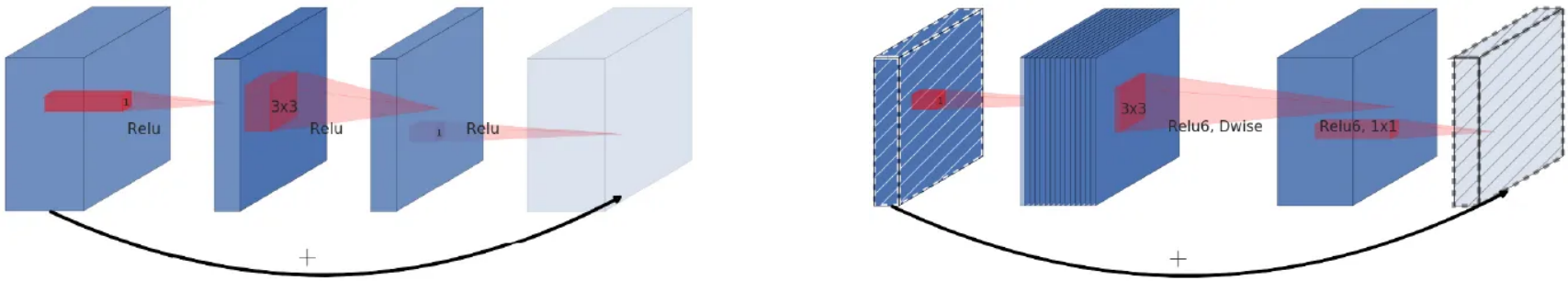
Improved convolution layers
Example : bottleneck block



Improved convolution layers
Example : bottleneck block



Improved convolution layers
MobileNetV2 : Mobile Inverted Bottleneck Conv (MBConv)



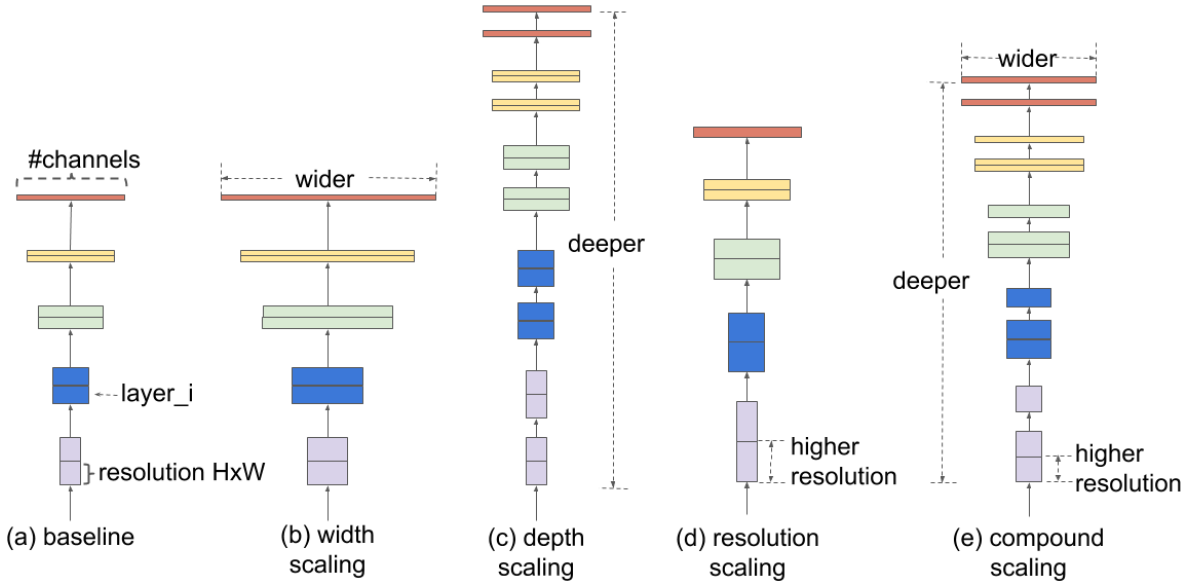
Efficient Processing of Features

- Improved feature extraction : more complex info,
- Depthwise Separable Convolutions,
- Reduced Parameter Count,
- Advantages of shortcut.

Sandler et al., *"Mobilenetv2: Inverted residuals and linear bottlenecks"*, 2018.

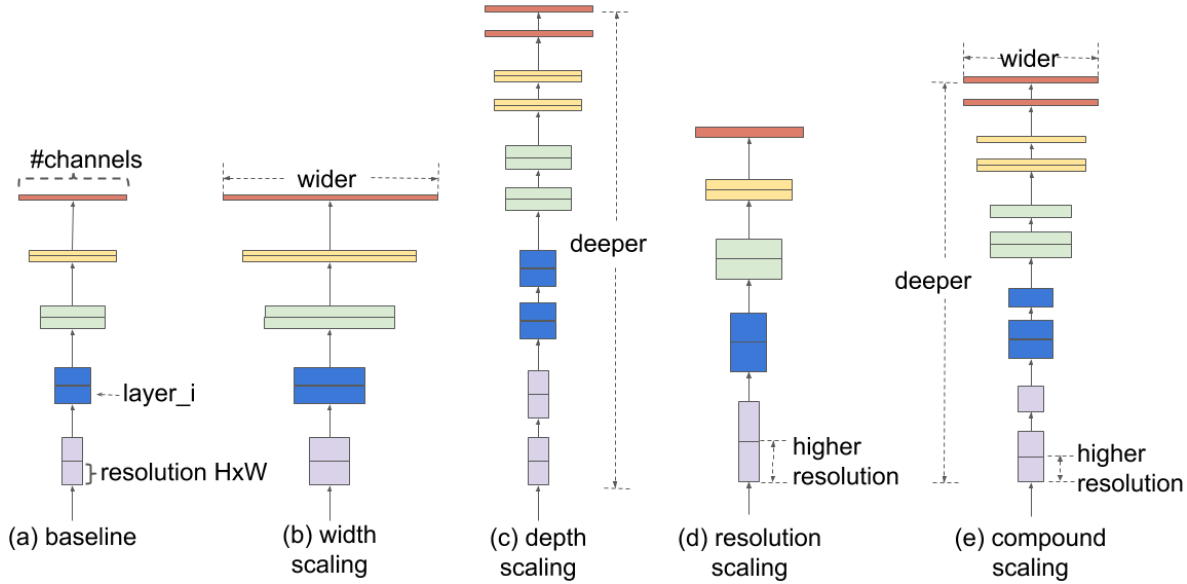
Neural Architecture Search (NAS)
EfficientNet : compound scaling

How to choose the structure of a CNN model ? **Manual exploration ?**



Neural Architecture Search (NAS)
EfficientNet : compound scaling

How to choose the structure of a CNN model ?

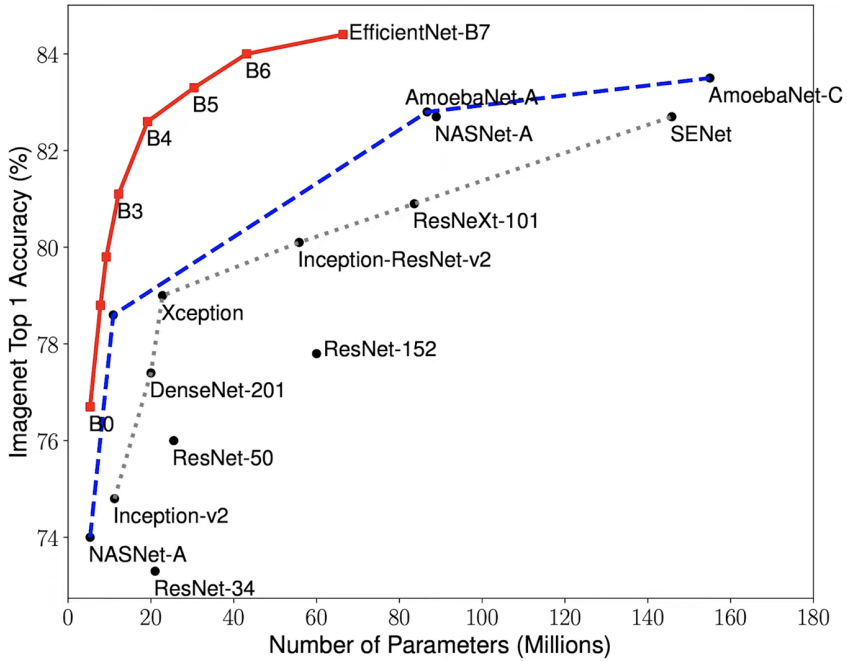


- Balance dimensions of width/depth/resolution by scaling with a constant ratio,
- 2^ϕ more computational resources

$$\begin{cases} \text{depth}: & d = \alpha^\phi \\ \text{width}: & w = \beta^\phi \\ \text{resol.}: & r = \gamma^\phi \\ \text{s.t.} & \alpha \cdot \beta^2 \cdot \gamma \approx 2 \\ & \alpha, \beta, \gamma \geq 1 \end{cases}$$

Neural Architecture Search (NAS)
EfficientNet : compound scaling

How to choose the structure of a CNN model ?



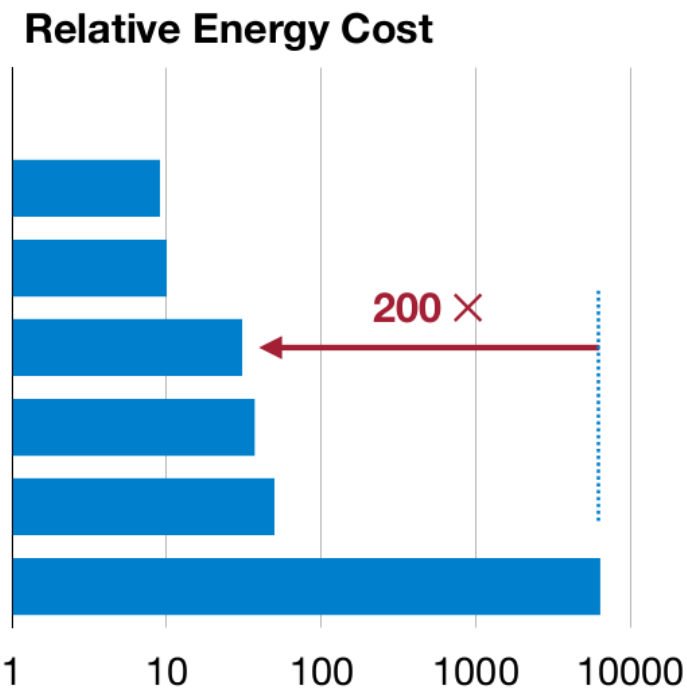
How to get α, β, γ (i.e. first network)

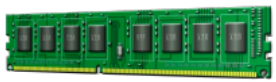
- Fix $\phi = 1$ + GridSearch with $\alpha \cdot \beta^2 \cdot \gamma = 2$,
  $\Longrightarrow$ EfficientNet-B0 : $\alpha = 1.2$, $\beta = 1.1$, $\gamma = 1.15$,
- Then, increase ϕ : EfficientNet-B1/B7.

Limitations of efficient models
Need of compression : memory

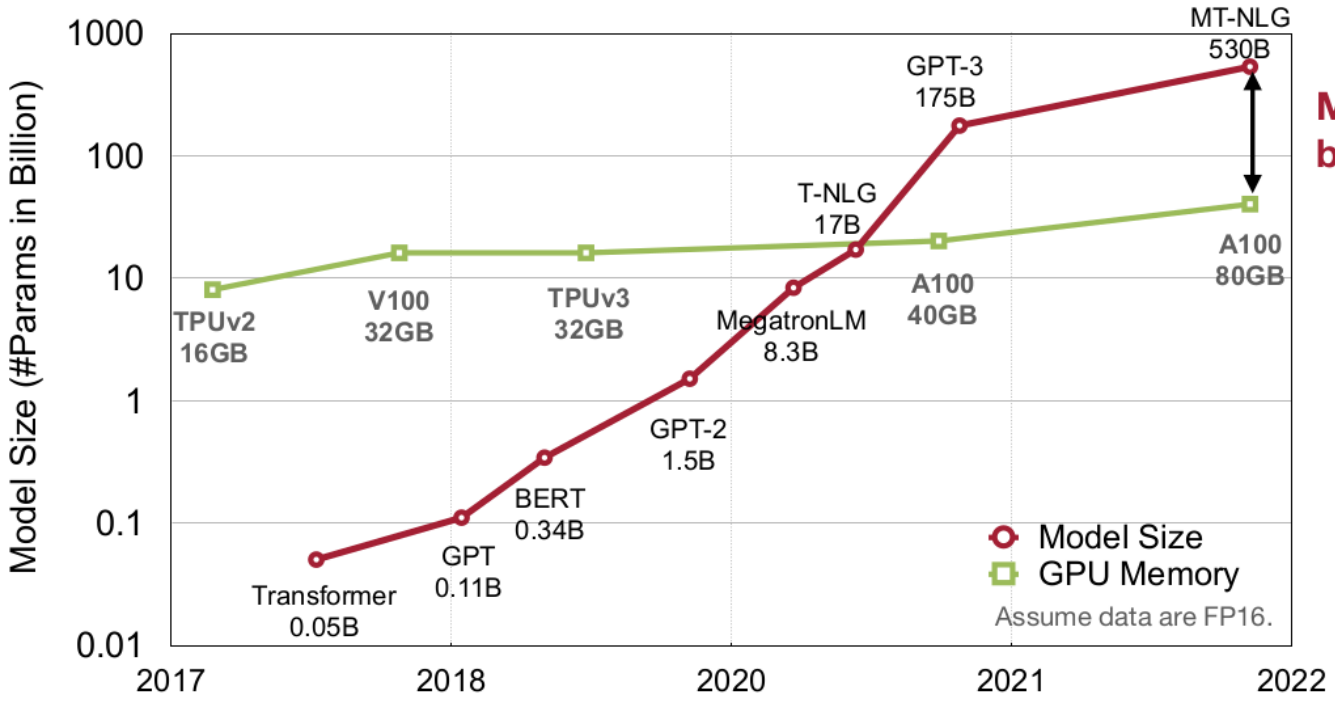
Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	640

Rough Energy Cost For Various Operations in 45nm 0.9V



1  = 200 ×+

Limitations of efficient models
Need of compression : memory and computation

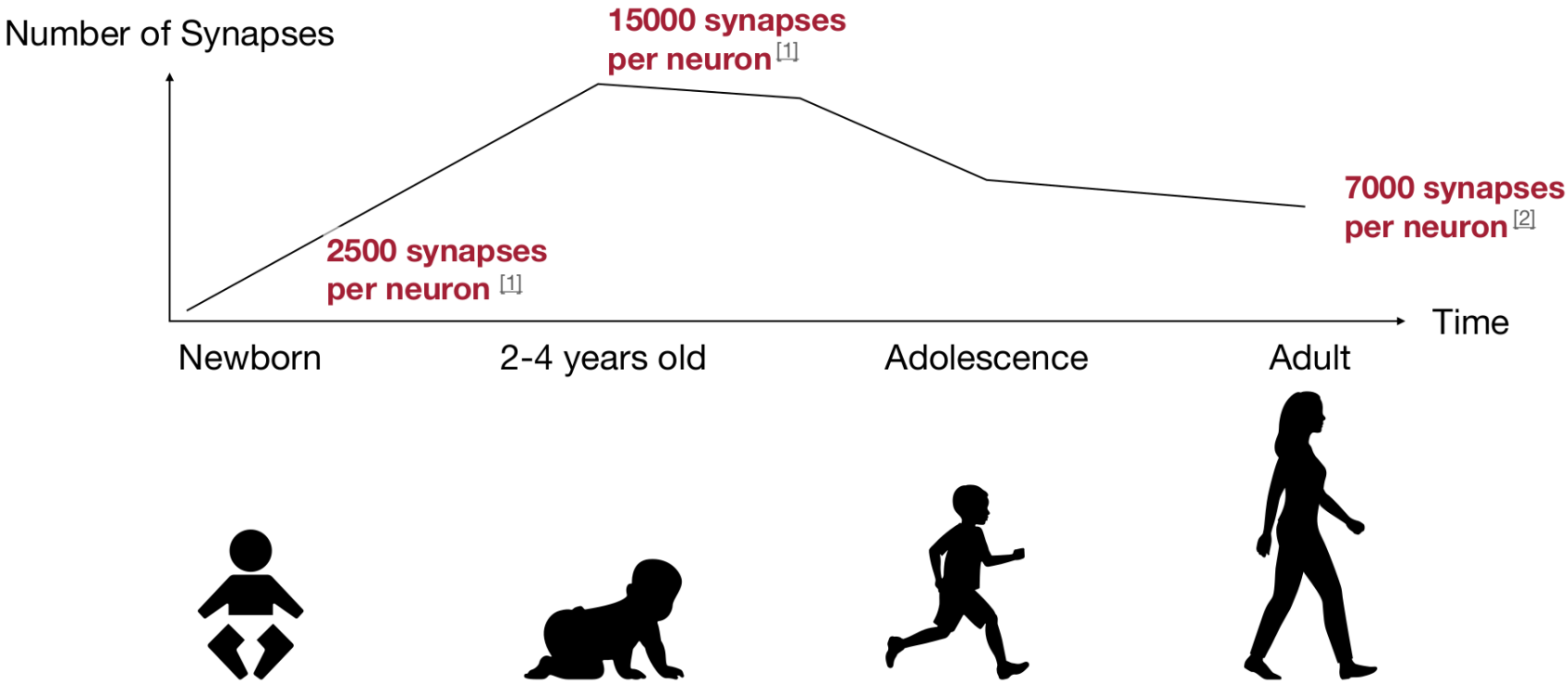


Model compression bridges the gap.

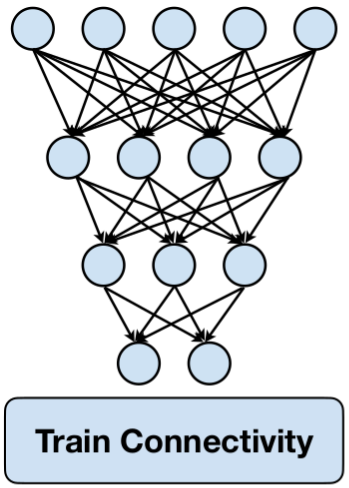
Section 3

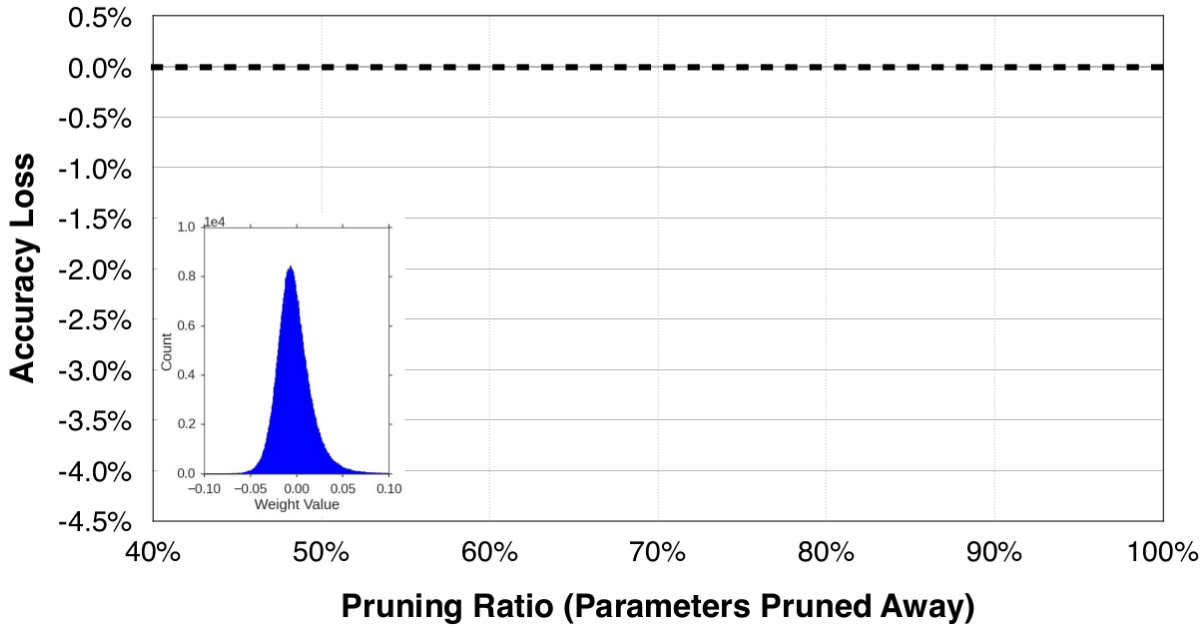
Network pruning

Network pruning
Pruning Happens in Human Brain

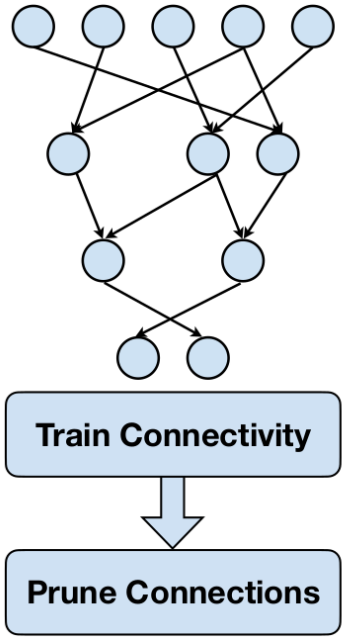


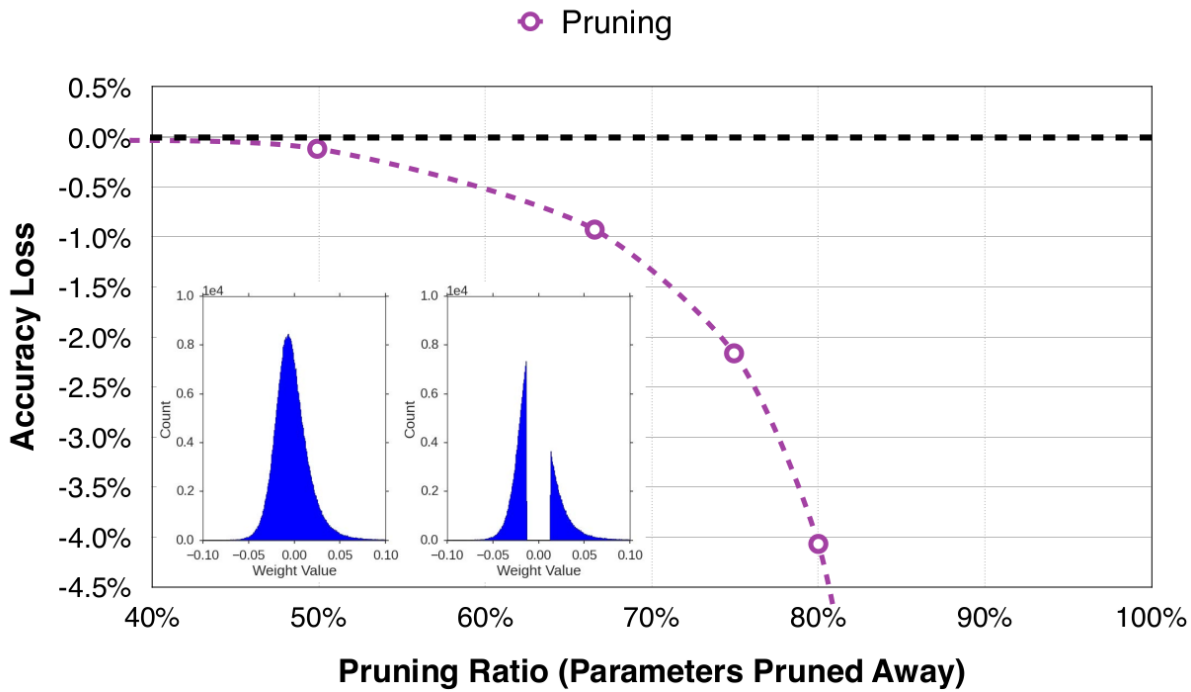
Network pruning
Principle : compress NN by removing synapses and neurons



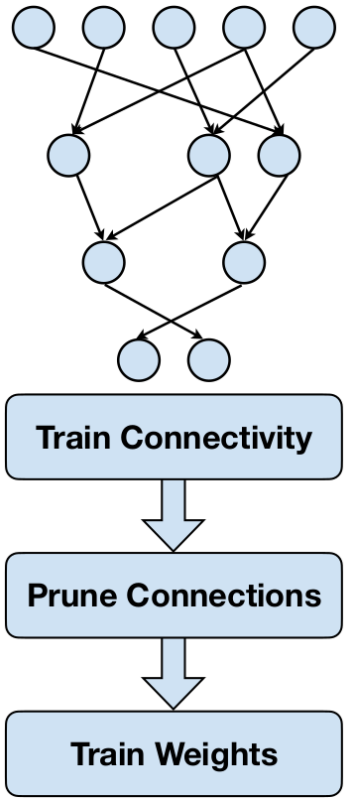


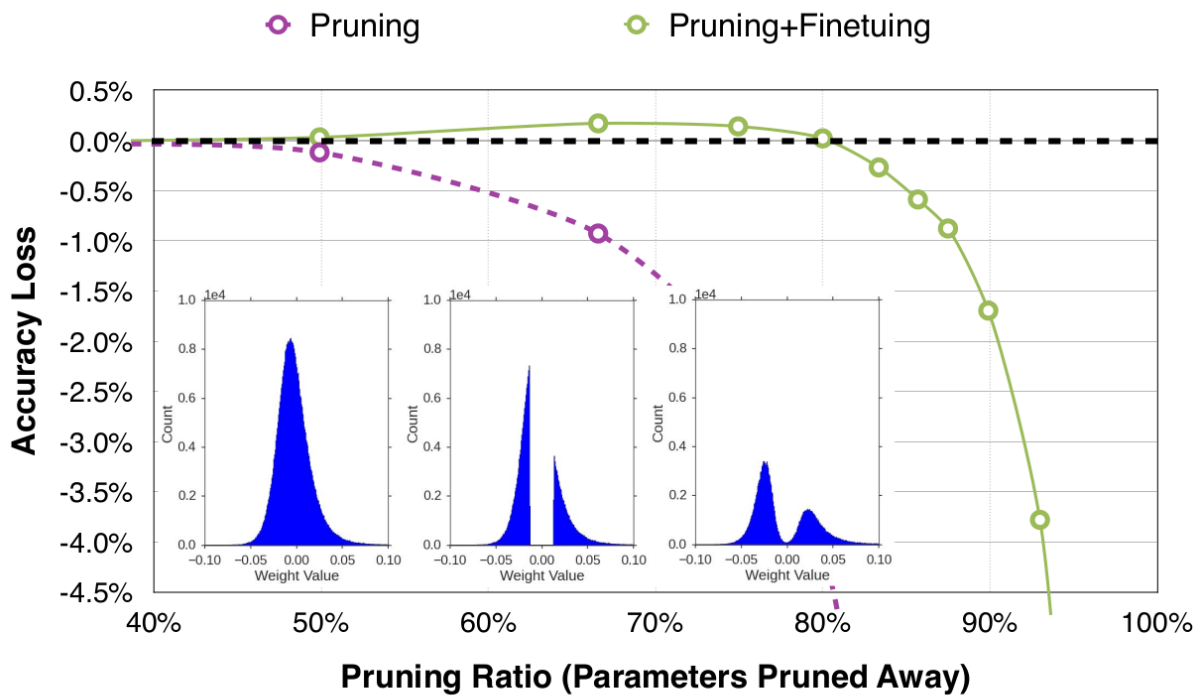
Network pruning
Principle : compress NN by removing synapses and neurons



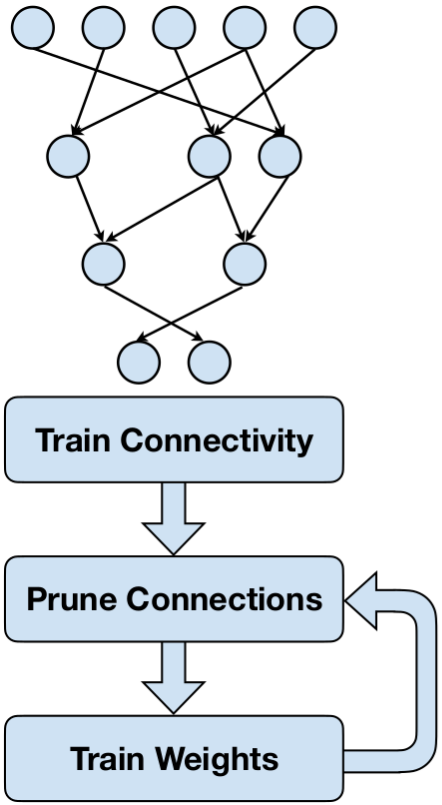


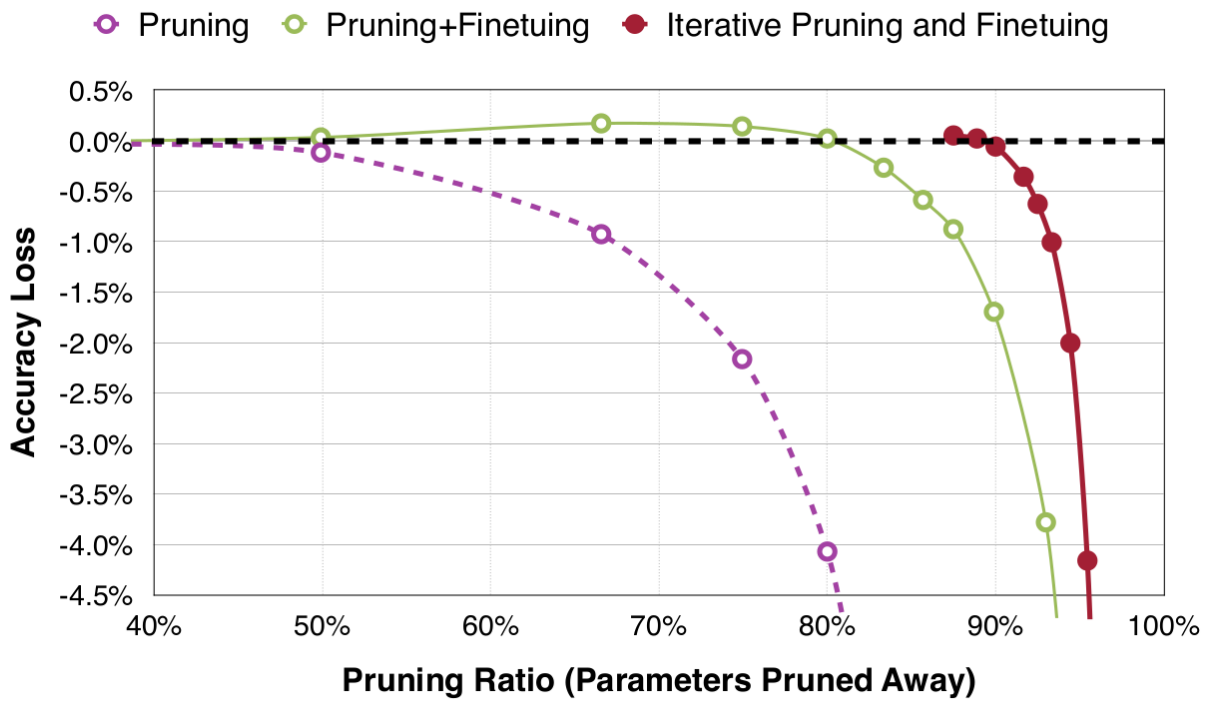
Network pruning
Principle : compress NN by removing synapses and neurons





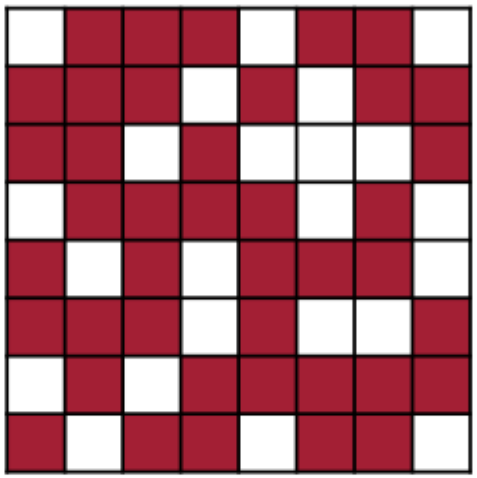
Network pruning
Principle : compress NN by removing synapses and neurons



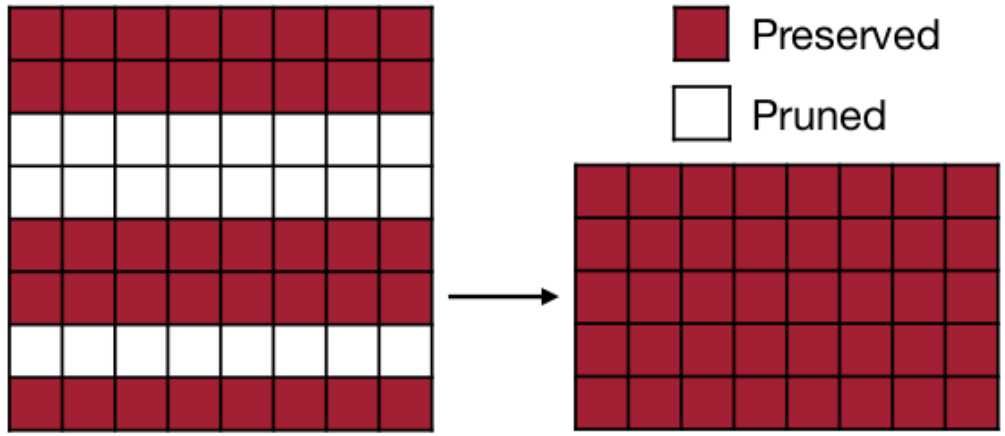


Network pruning
Granularity of pruning : structured to unstructured

Pruning weights (NN/CNN)



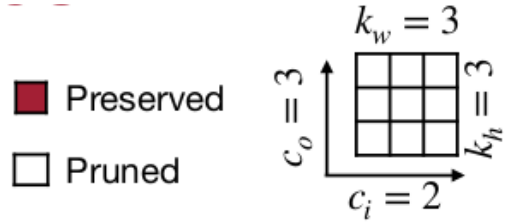
Fine-grained/Unstructured

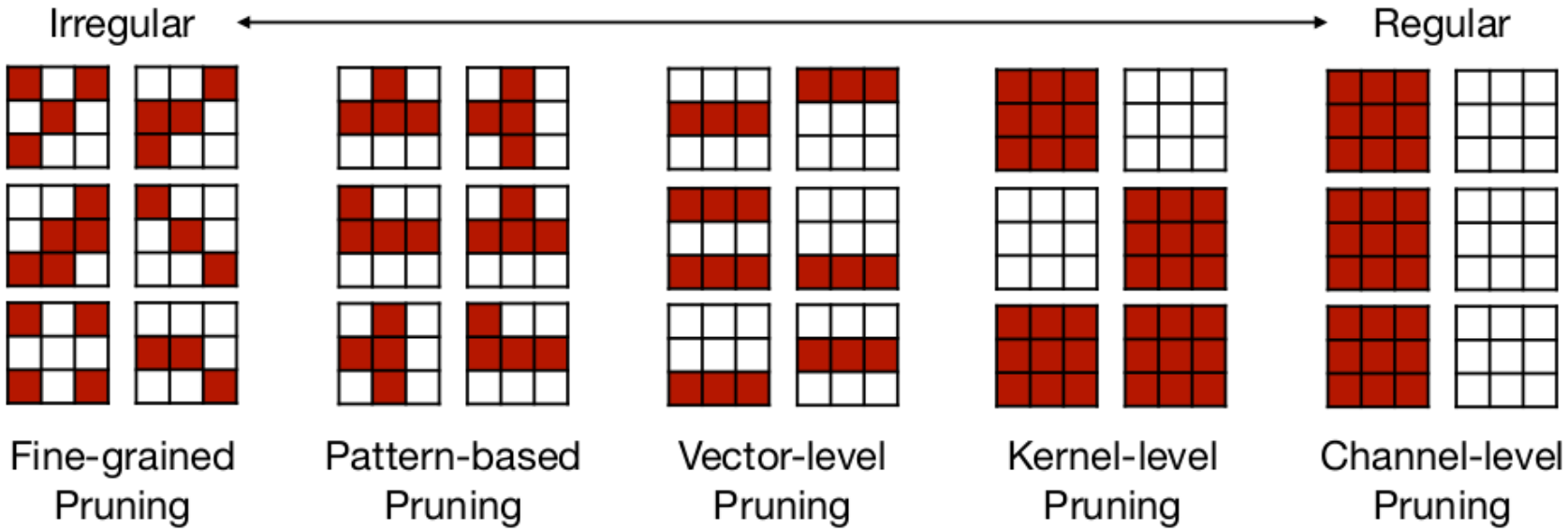


Coarse-grained/Structured

Network pruning
Granularity of pruning : structured to unstructured

Pruning kernels (CNN)

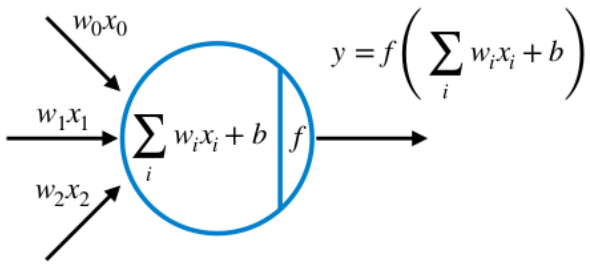




Network pruning
Selection of Synapses to Prune

When removing parameters from a NN model :

- the **less important** the parameters being removed are
- the better the performance of the pruned neural network is.



Example :
$f(\cdot) = ReLU(\cdot),\ W = [10, -8, 0.1]$
$\Longrightarrow y = ReLU(10x_0 - 8x_1 + 0.1x_2)$

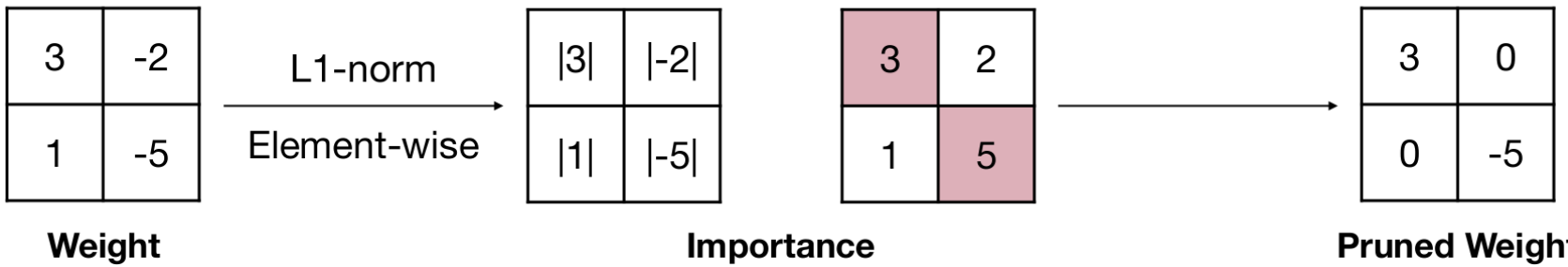
Which weight should be removed ?

Network pruning
Selection of Synapses to Prune : A heuristic criterion

Magnitude-based pruning considers that weights with larger absolute values are more important than other weights :

- For element-wise pruning, $Importance = |W|$
- For row-wise pruning, the norm ($L1$, $L2$, Lp) magnitude can be defined as $Importance = \|W^{(S)}\| = \sum_{i \in S} |w_i|$, where $W^{(S)}$ is the structural set S of parameters W

Example

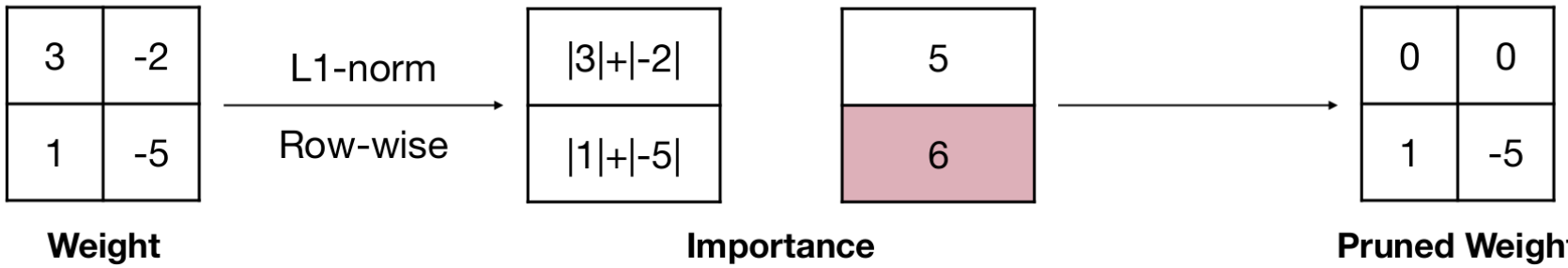


Network pruning
Selection of Synapses to Prune : A heuristic criterion

Magnitude-based pruning considers that weights with larger absolute values are more important than other weights :

- For element-wise pruning, $Importance = |W|$
- For row-wise pruning, the norm ($L1$, $L2$, Lp) magnitude can be defined as $Importance = \|W^{(S)}\| = \sum_{i \in S} |w_i|$, where $W^{(S)}$ is the structural set S of parameters W

Example

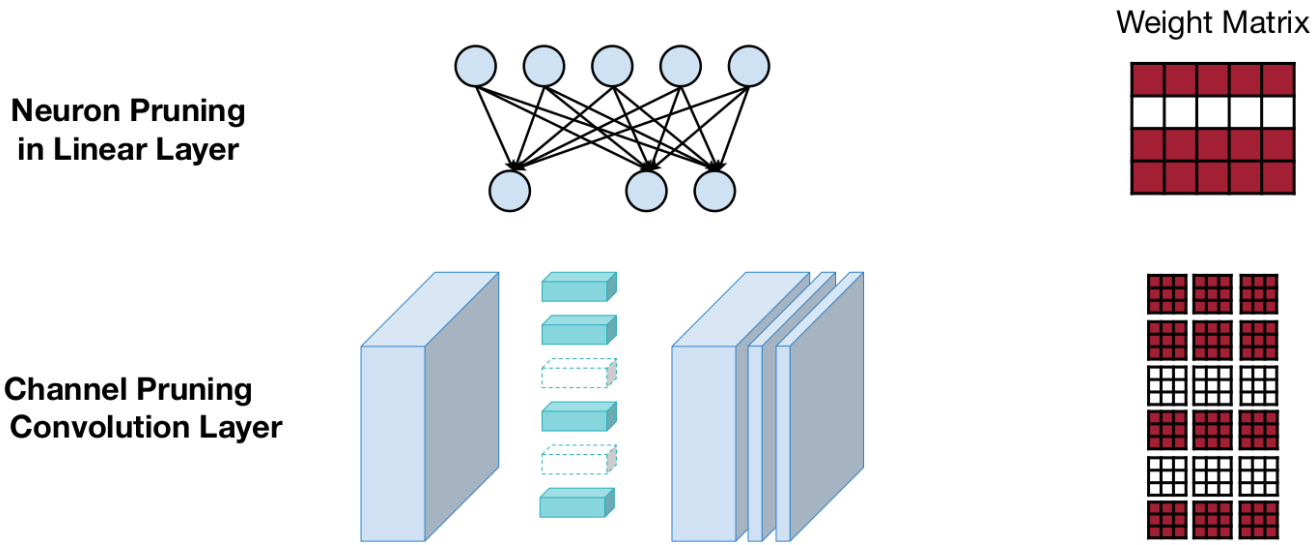


Network pruning
Selection of Neurons/Kernels to Prune

When removing neurons from a neural network model :

- the less useful the neurons being removed are,
- the better the performance of pruned neural network is.

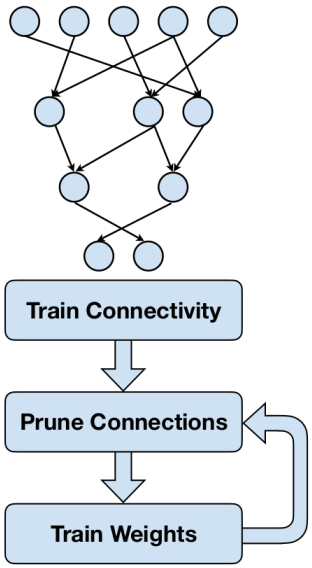
Neuron pruning is coarse-grained weight pruning.

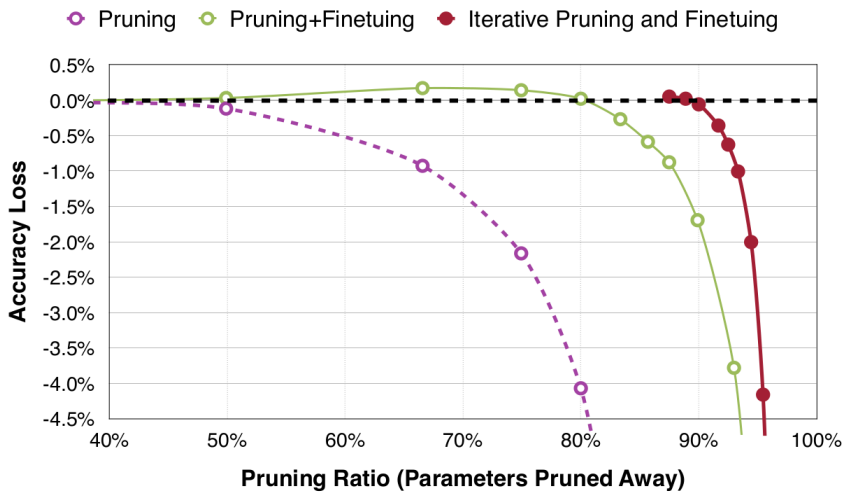


Pruning of neurons can be performed e.g. based on the values of the activation functions.

Network pruning
Finetuning Pruned Neural Networks

- After pruning, the model may decrease, especially for larger pruning ratio.
- Fine-tuning the pruned neural networks : iterative process
  - Help recover the accuracy,
  - Learning rate of fine-tuning usually 1/100 or 1/10 of the original learning rate
  - Boost pruning ratio from 5× to 9× on AlexNet compared to single-step aggressive pruning.





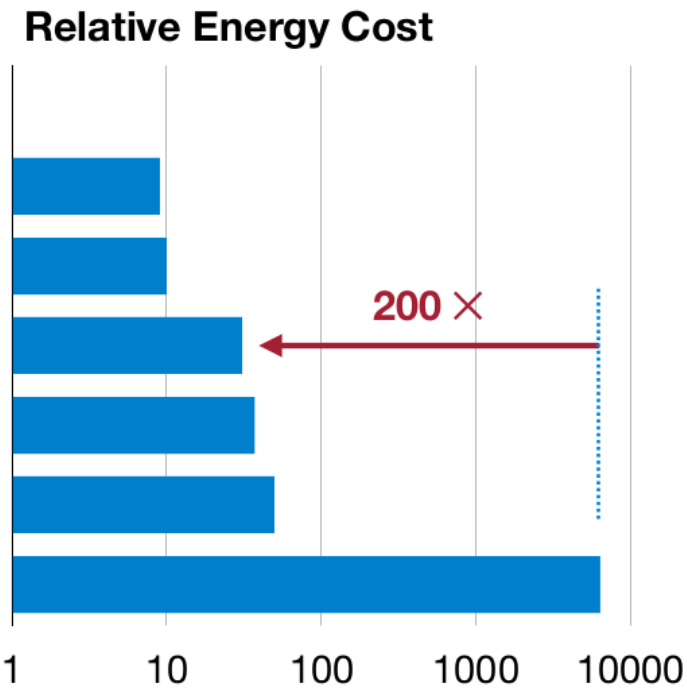
Section 4

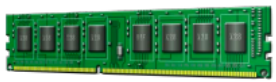
Quantization

Quantization
Motivation : less bit-width = less energy

Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	640

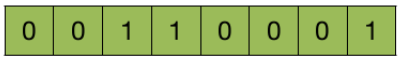
Rough Energy Cost For Various Operations in 45nm 0.9V



1  = 200 × +

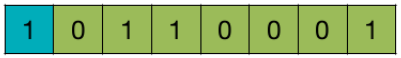
Quantization
Data representation

Integers

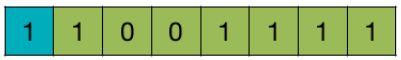


$$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 49$$

Sign Bit



$$-\ 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -49$$



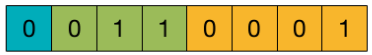
$$-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -49$$

Fixed-point numbers

Integer . Fraction

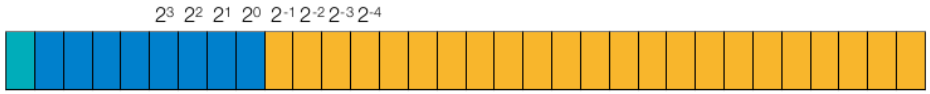
"Decimal" Point

$$-2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = 3.0625$$



$$(\ -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0\ ) \times 2^{-4}$$

Quantization
Data representation : floating-point numbers

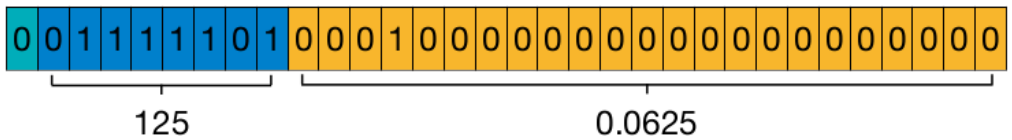


Sign 8 bit Exponent **23 bit Fraction**

$$(-1)^{sign} \times (1 + Fraction) \times 2^{Exponent-127} \longleftarrow Exp.Bias = 2^{8-1} - 1$$

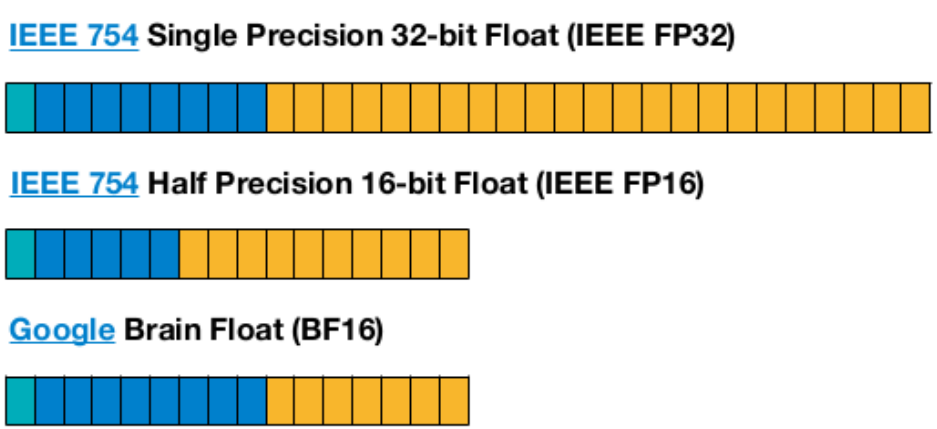
significant / mantissa

$$0.265625 = 1.0625 \times 2^{-2} = (1 + 0.0625) \times 2^{125-127}$$



Quantization
Data representation : smaller precisions

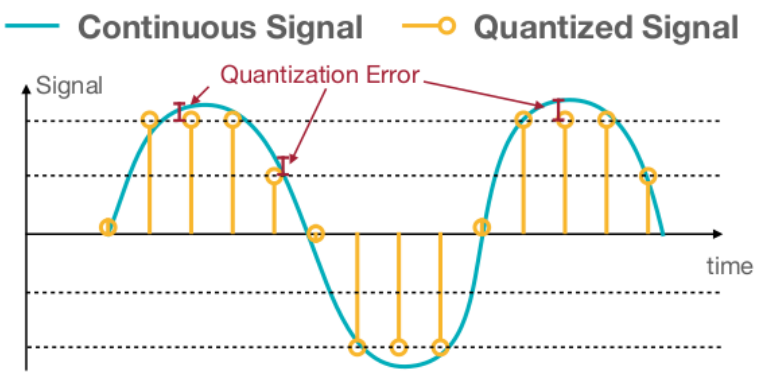
- Exponent width=range
- Fraction width=precision



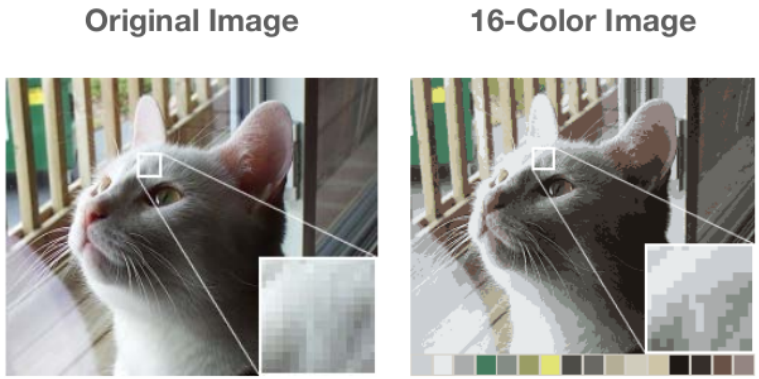
Exponent (bits)	Fraction (bits)	Total (bits)
8	23	32
5	10	16
8	7	16

Quantization
Principle : example

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.

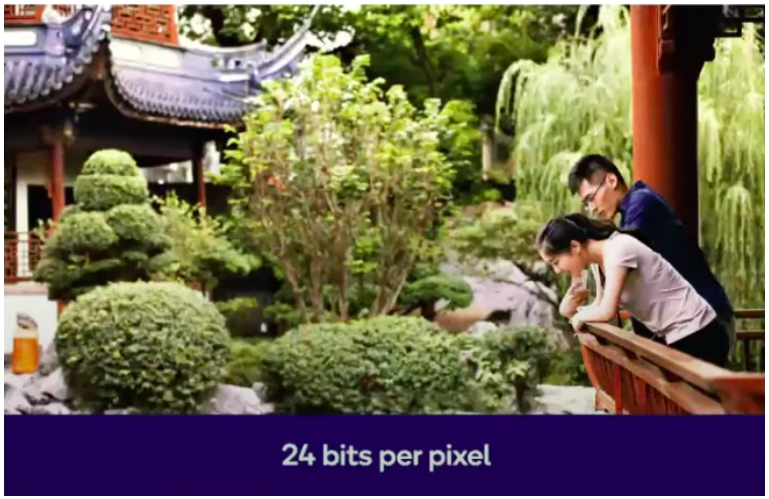


The difference between an input value and its quantized value is referred to as quantization error.



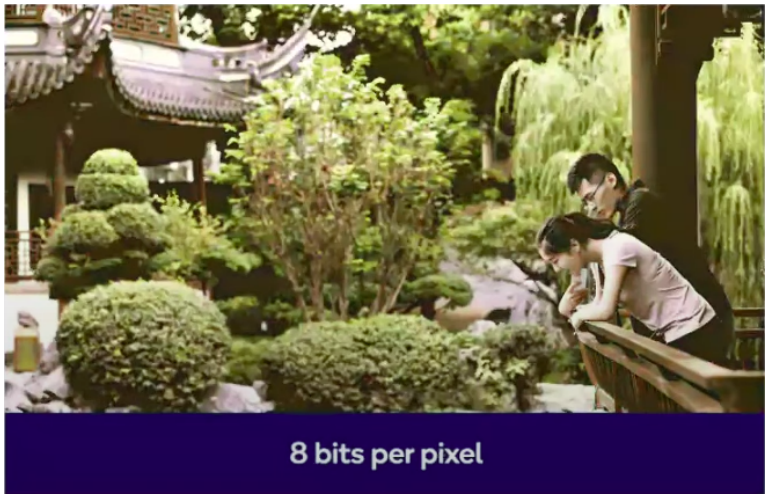
Quantization
Principle : example

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.



Quantization
Principle : example

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.

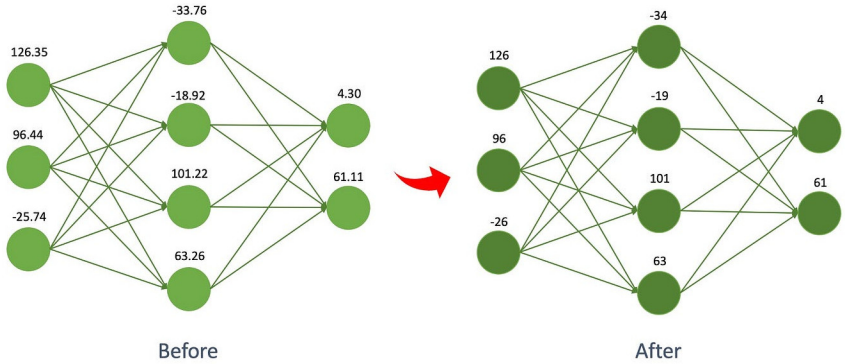


Quantization
Principle : example

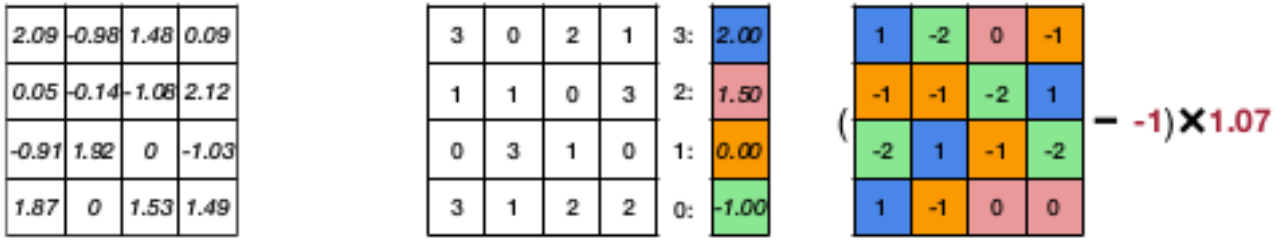
Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.

Quantization
Principle : example

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.



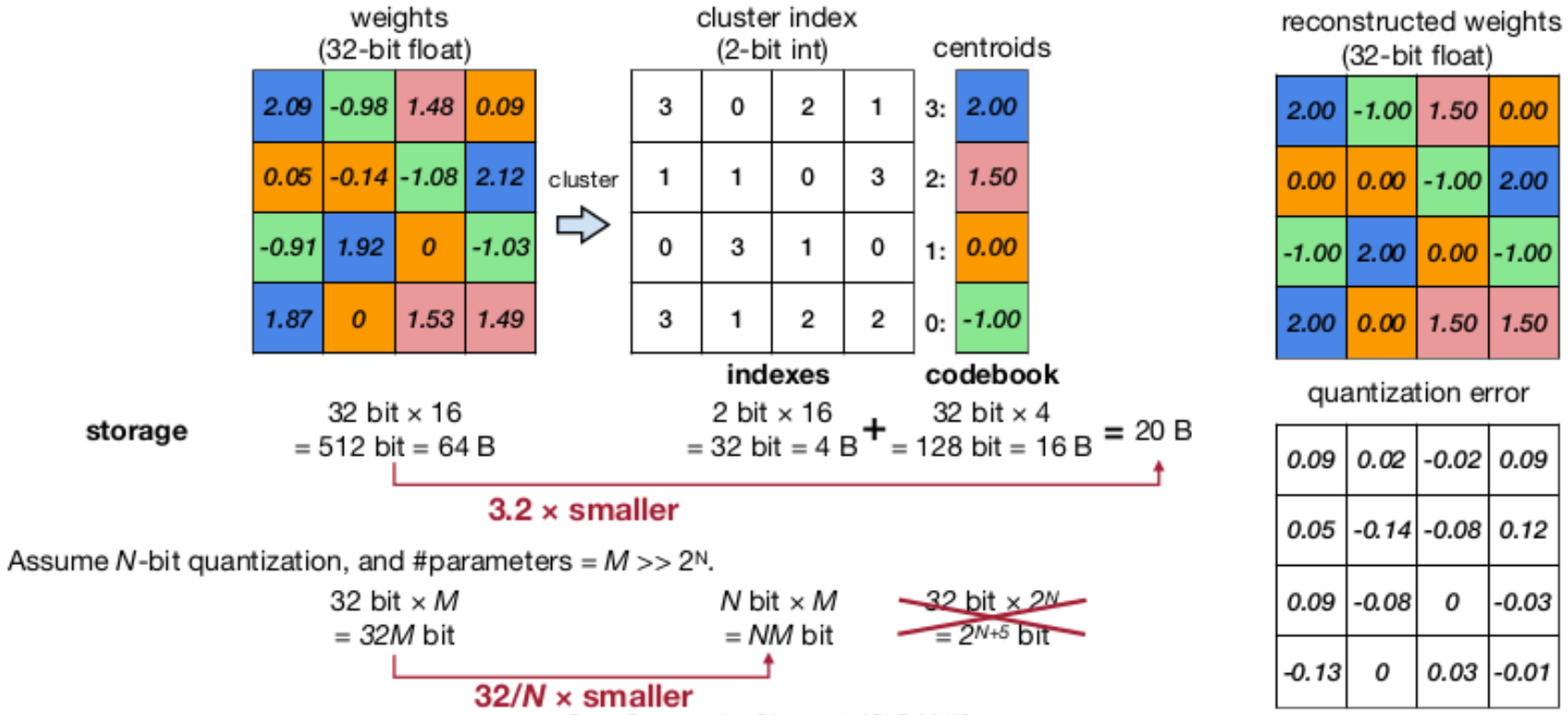
Quantization
Methods



	K-Means-based Quantization	Linear Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic

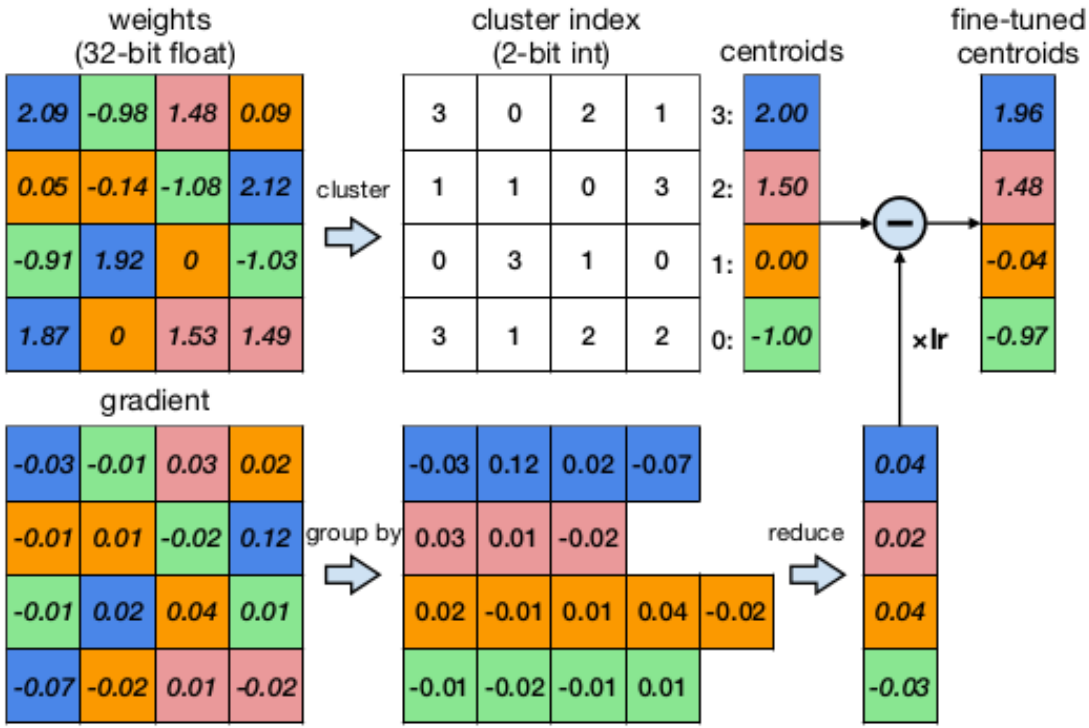
Quantization
K-means-based quantization

Quantize weights



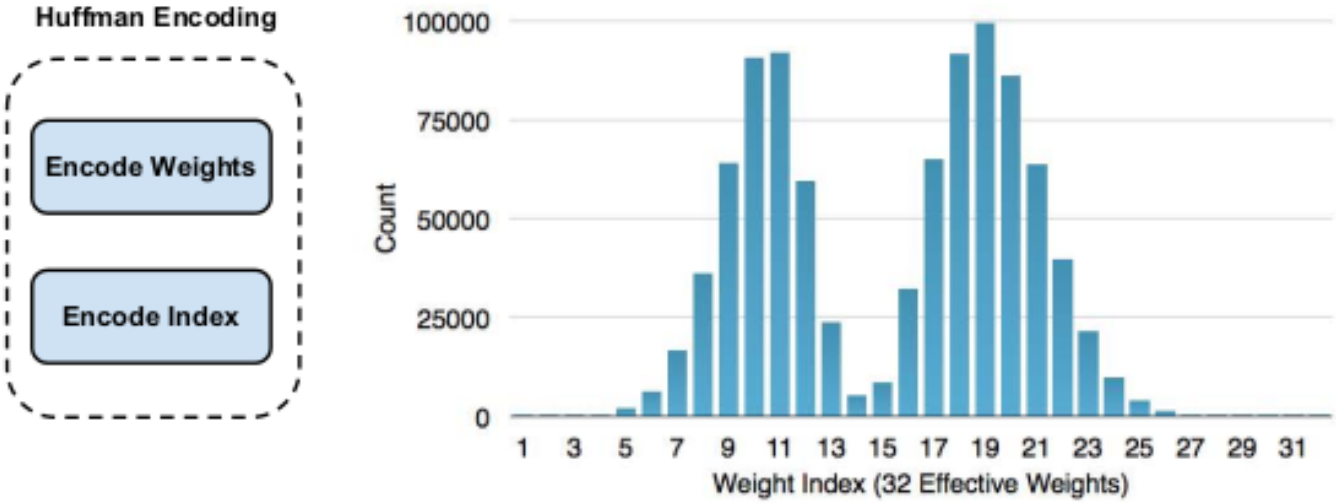
Quantization
K-means-based quantization

Quantize weights + fine-tuning



Quantization
K-means-based quantization

Quantize weights + fine-tuning + Huffman encoding

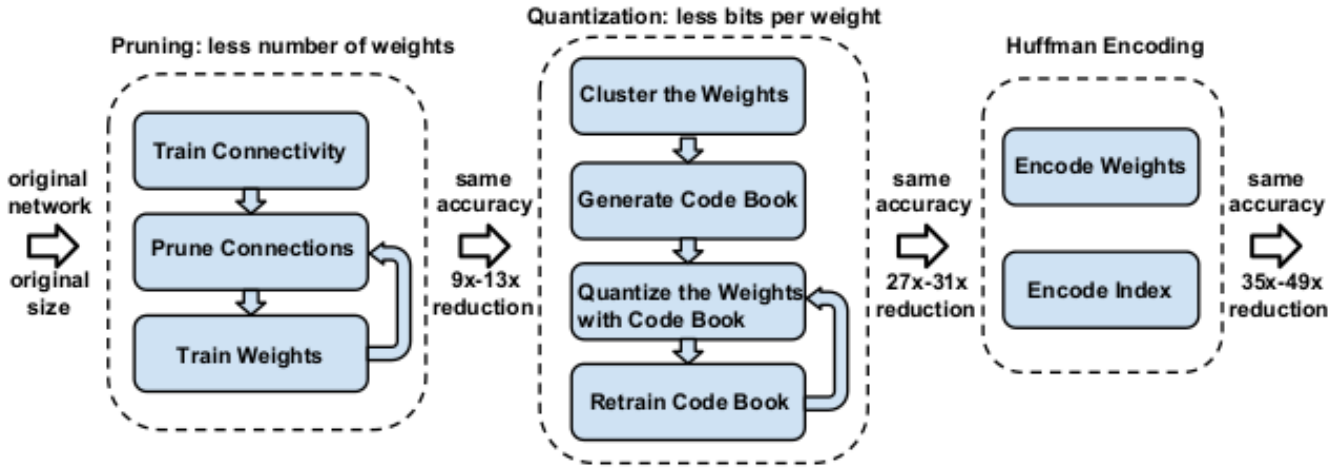


- In-frequent weights: use more bits to represent
- Frequent weights: use less bits to represent

- In-frequent weights : use more bits to represent,
- Frequent weights : use less bits to represent.

Quantization
K-means-based quantization

$\Longrightarrow$ **Deep compression !**



- In-frequent weights : use more bits to represent,
- Frequent weights : use less bits to represent.

Han et al., *"Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding"*, 2015.

Quantization
K-means-based quantization

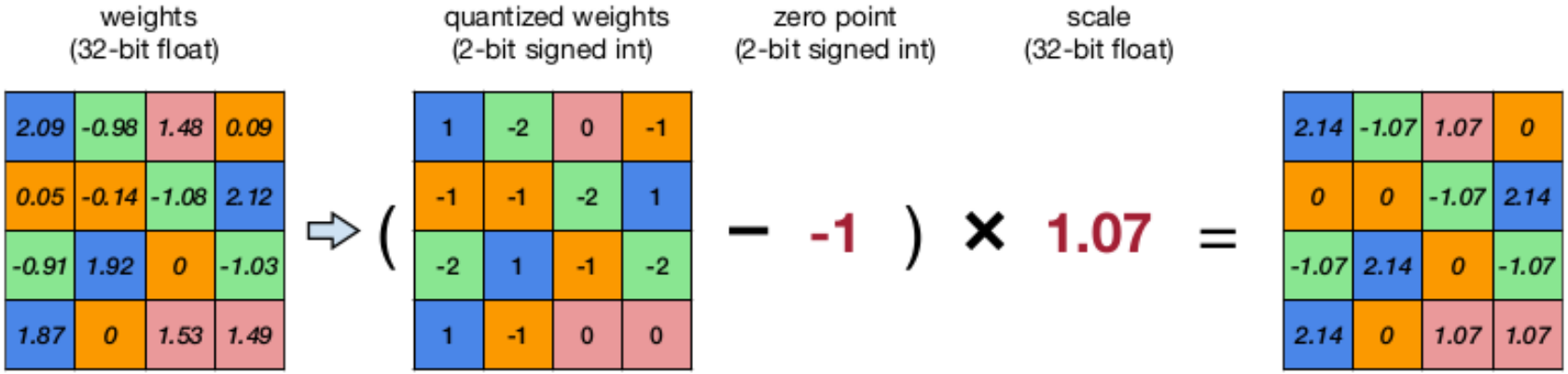
$\Longrightarrow$ **Deep compression !**

Network	Original Size	Compressed Size	Compression Ratio	Original Accuracy	Compressed Accuracy
LeNet-300	1070KB	27KB	**40x**	98.36%	98.42%
LeNet-5	1720KB	44KB	**39x**	99.20%	99.26%
AlexNet	240MB	6.9MB	**35x**	80.27%	80.30%
VGGNet	550MB	11.3MB	**49x**	88.68%	89.09%
GoogleNet	28MB	2.8MB	**10x**	88.90%	88.92%
ResNet-18	44.6MB	4.0MB	**11x**	89.24%	89.28%

- Weights decompressed using a lookup table during runtime inference,
- **Computations and memory access are still floating-point**,
- Only saves storage cost of a NN.

Quantization
Linear quantization

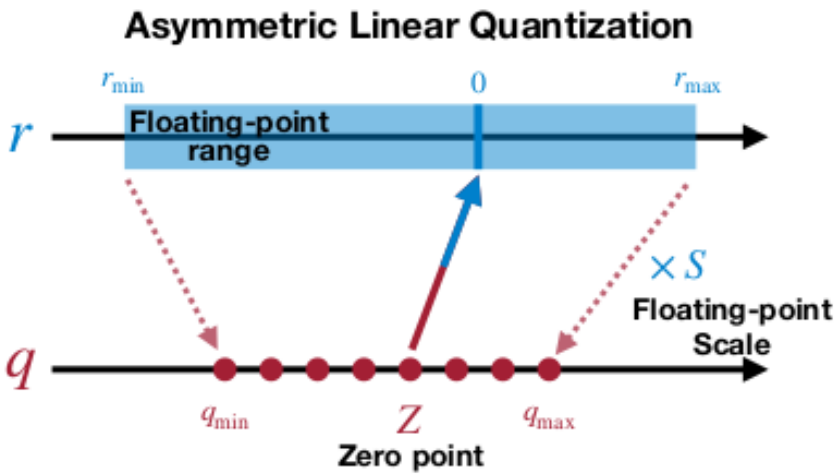
Linear Quantization is an affine mapping of integers to real numbers :
$r = S(q - Z)$



Binary	Decimal
01	1
00	0
11	-1
10	-2

Quantization
Linear quantization

Linear Quantization is an affine mapping of integers to real numbers :
$r = S(q - Z)$



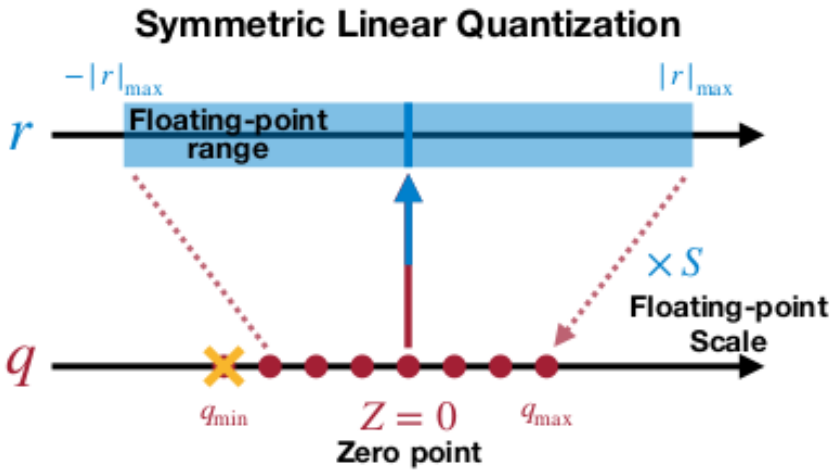
2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}} = \frac{2.12 - (-1.08)}{1 - (-2)} = 1.07$$

$$Z = q_{min} - \frac{r_{min}}{S} = round(-2 - \frac{-1.08}{1.07}) = -1$$

Quantization
Linear quantization

Linear Quantization is an affine mapping of integers to real numbers :
$r = S(q - Z)$



2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

$$S = \frac{|r_{max}|}{q_{max}} = \frac{2.12}{1} = 2.12 \qquad Z = 0$$

Quantization
Post-training quantization

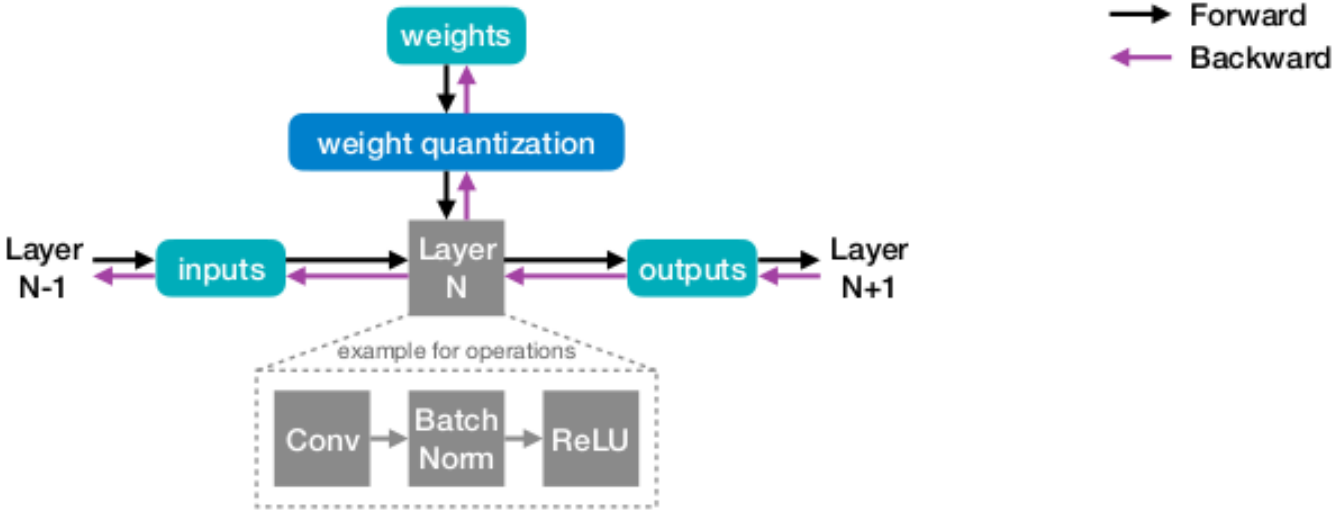
	K-Means-based Quantization	Linear Quantization
Storage	Integer Weights; Floating-Point Codebook	Integer Weights
Computation	Floating-Point Arithmetic	Integer Arithmetic

- Zero Point
  - Asymmetric
  - Symmetric
- Scaling Granularity
  - Per-Tensor
  - Per-Channel
  - Group Quantization
- Range Clipping
  - Exponential Moving Average
  - Minimizing KL-Divergence
  - Minimizing Mean-Square Error
- Rounding
  - Round-to-Nearest
  - AdaRound

Quantization
Quantization-aware training

Train the model taking quantization into consideration

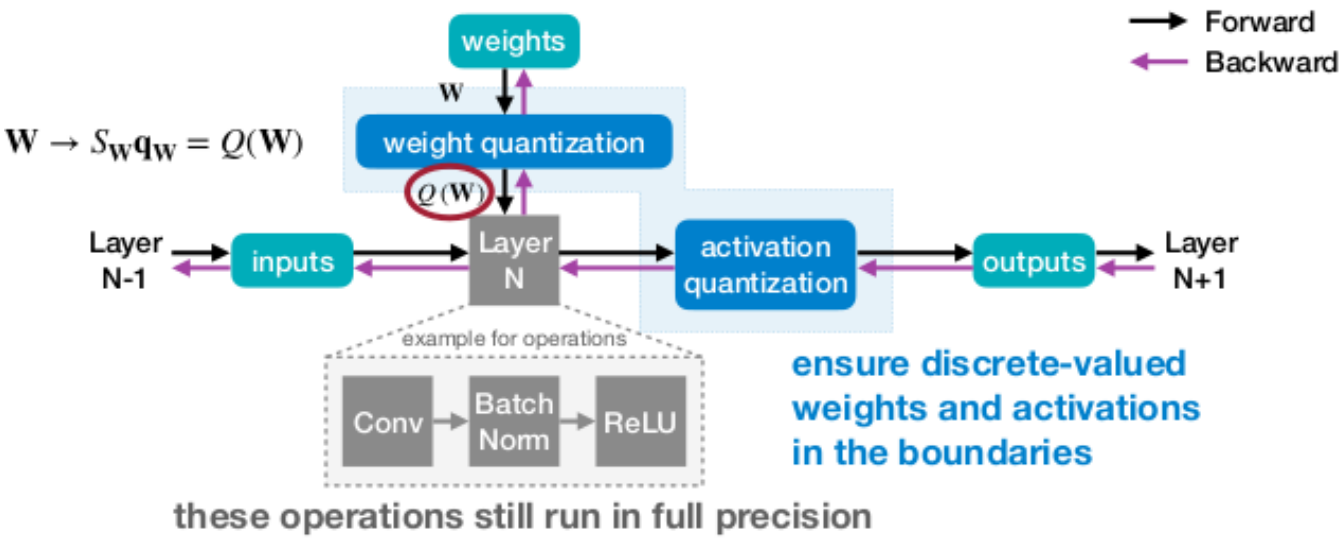
- A full precision copy of the weights W is maintained throughout the training.
- The small gradients are accumulated without loss of precision.
- Once the model is trained, only the quantized weights are used for inference.



Quantization
Quantization-aware training

Train the model taking quantization into consideration

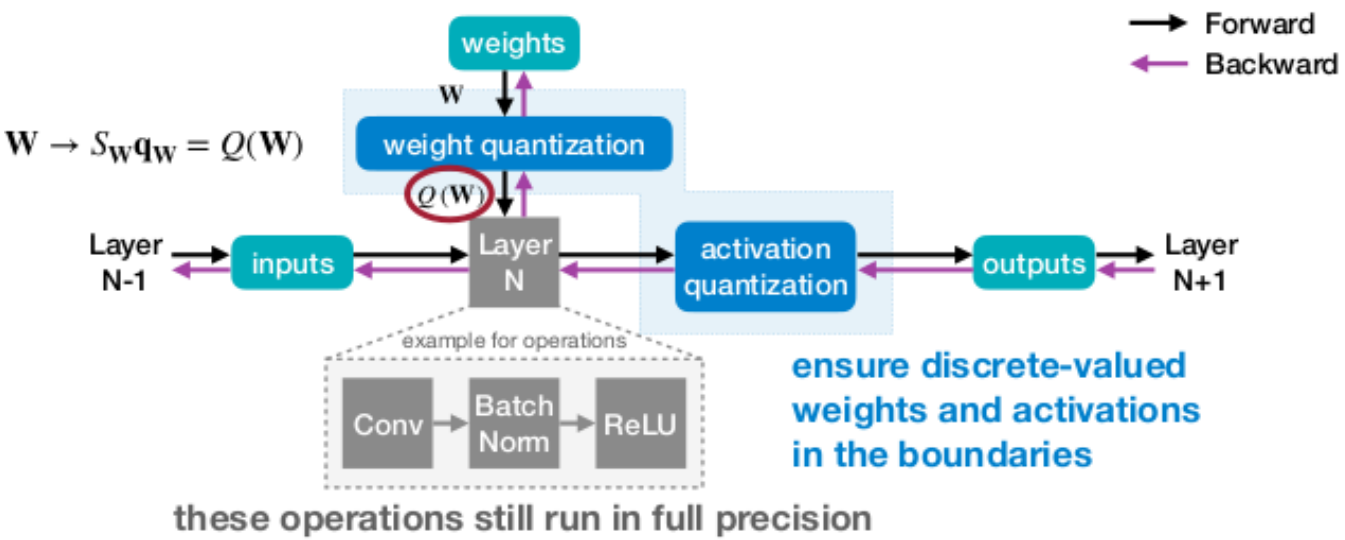
- A full precision copy of the weights W is maintained throughout the training.
- The small gradients are accumulated without loss of precision.
- Once the model is trained, only the quantized weights are used for inference.
- $\Longrightarrow$ use a "SimulatedFake Quantization"
- $\Longrightarrow$ use a "SimulatedFake Quantization"

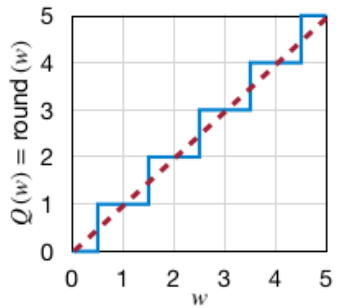


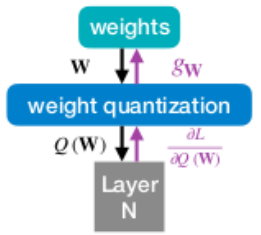
Quantization
Quantization-aware training

Train the model taking quantization into consideration

- A full precision copy of the weights W is maintained throughout the training.
- The small gradients are accumulated without loss of precision.
- Once the model is trained, only the quantized weights are used for inference.
- $\Longrightarrow$ use a "SimulatedFake Quantization"
- $\Longrightarrow$ use a "SimulatedFake Quantization"
- $\Longrightarrow$ Straight-Through Estimator (STE) : gradients are passed through Quant. as if identity







Quantization
Quantization-aware training

Train the model taking quantization into consideration

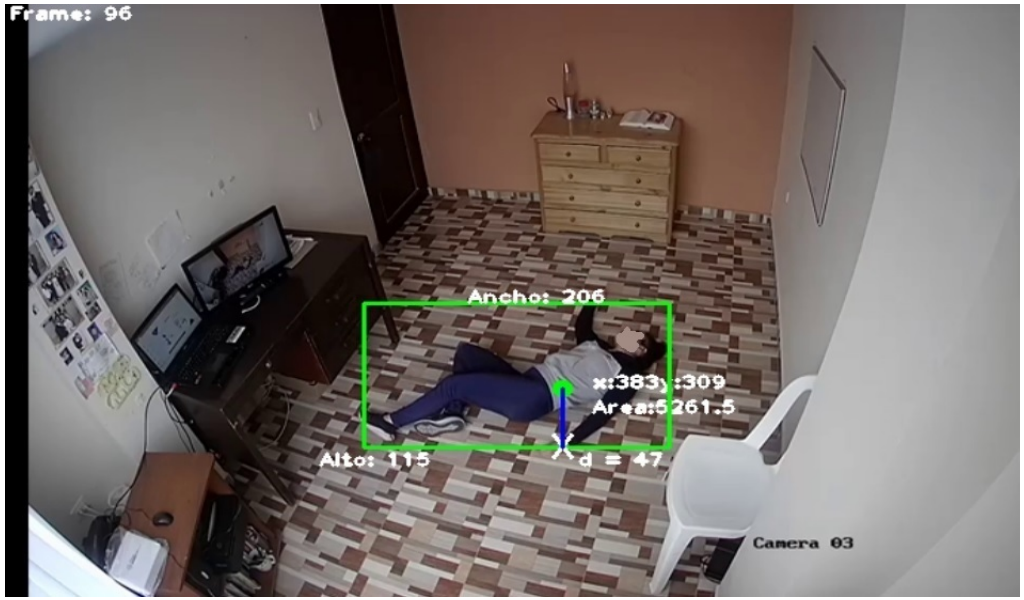
- A full precision copy of the weights W is maintained throughout the training.
- The small gradients are accumulated without loss of precision.
- Once the model is trained, only the quantized weights are used for inference.
- $\Longrightarrow$ use a "SimulatedFake Quantization"
- $\Longrightarrow$ use a "SimulatedFake Quantization"
- $\Longrightarrow$ Straight-Through Estimator (STE) : gradients are passed through Quant. as if identity

Neural Network	Floating-Point	Post-Training Quantization		Quantization-Aware Training	
		Asymmetric	Symmetric	Asymmetric	Symmetric
		Per-Tensor	Per-Channel	Per-Tensor	Per-Channel
MobileNetV1	70.9%	0.1%	59.1%	70.0%	70.7%
MobileNetV2	71.9%	0.1%	69.8%	70.9%	71.1%
NASNet-Mobile	74.9%	72.2%	72.1%	73.0%	73.0%

Section 5

Practical sessions : Lab 1

Practical sessions : Lab 2



Objective

Recognize human falls using images of subjects in a room.

Practical sessions : Lab 2
Data

Dataset : https://data.mendeley.com/datasets/7w7fccy7ky/4
The dataset of falls contains data from 10 different activities :

1. Fall backwards,
2. Fall forward,
3. Fall left,
4. Fall right,
5. Fall sitting,
6. Hop,
7. Kneel,
8. Pick up object,
9. Sit down,
10. Walk.

The images were obtained from videos taken in conditions of an uncontrolled home environment (occlusions, lights, clothes, etc.).
Each of the 10 subjects had to follow a specific protocol, containing the 10 different "activities".

Practical sessions : Lab 2
Data

Dataset : https://data.mendeley.com/datasets/7w7fccy7ky/4

Images :

- 20 000 pictures in .png format,
- The pictures are labelled by activity

The dataset was preprocessed :

- A YOLO model was used to segment the person in each picture, resulting in square images of size ranging between 200 and 350,
- The images were resized to be 96×96,
- Labels are merged to get : fall vs. non-fall activity.

Two sets are available

- Training set : images of subjects 1-8,
- Final est set : images of subjects 9 and 10.

Practical sessions : Lab 2
In a nutshell

Steps :

Lab 1 – Human activity recognition :

Lab 2 – Fall recognition :

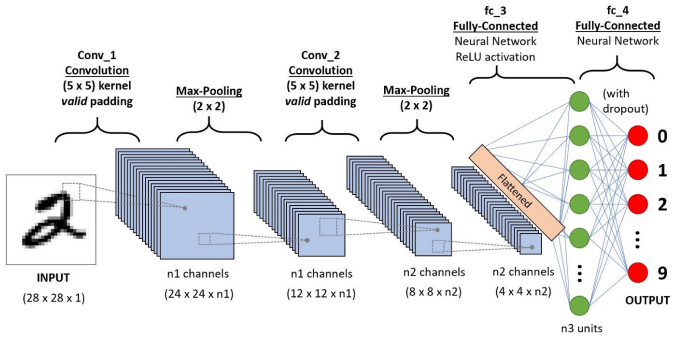
1. Build a CNN for fall recognition using TensorFlow,
2. Optimize/compress the CNN,
3. Transfer-Learning for fall recognition.

In practice :

- **Langage :** Notebook Python
  - Tutorials python on moodle,
  - Possibility to share the notebook via *Google Colab*,
- Work in **groups of 2**,
- **Deliverable :** a notebook by group per Lab containing your code and analysis,
- **Deadline : 23th january**.

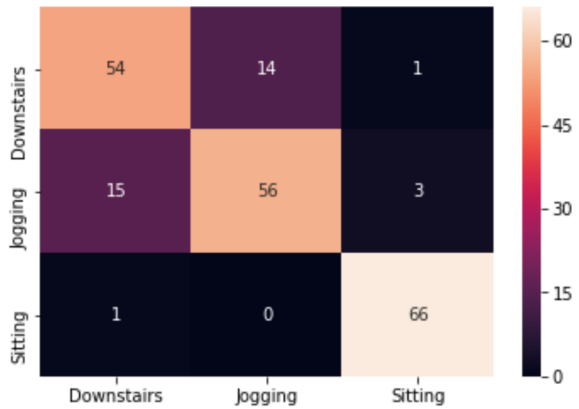
Practical sessions : Lab 2
Part 1 : Build a CNN for fall recognition using TensorFlow

CNN architecture



⇒ Learn how to use the TensorFlow library and CNN methods

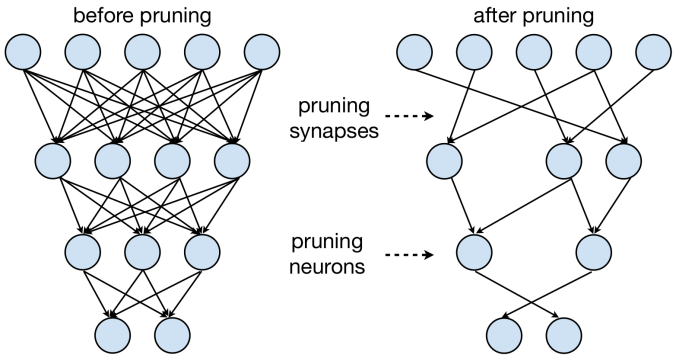
- Use of **Tensorflow.keras** *layers* and *models* to build a CNN capable of fall recognition : validation of hyperparameters,
- Evaluation with *confusion_matrix* and *accuracy_score* from scikit-learn.

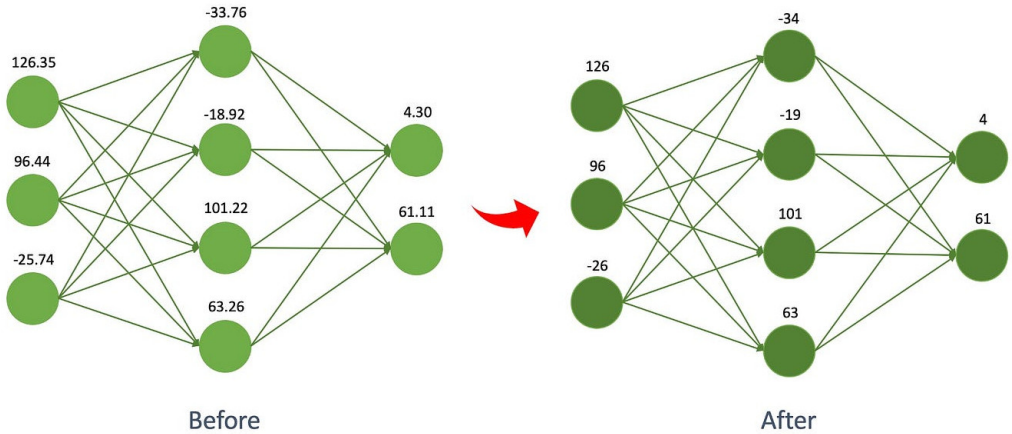


Practical sessions : Lab 2
Part 2 : Optimize/compress the CNN

Use of the Python *TensorFlow Lite* library

- Compress the model with pruning and quantization,
- Conservation of the model accuracy,
- Estimate size of the model.

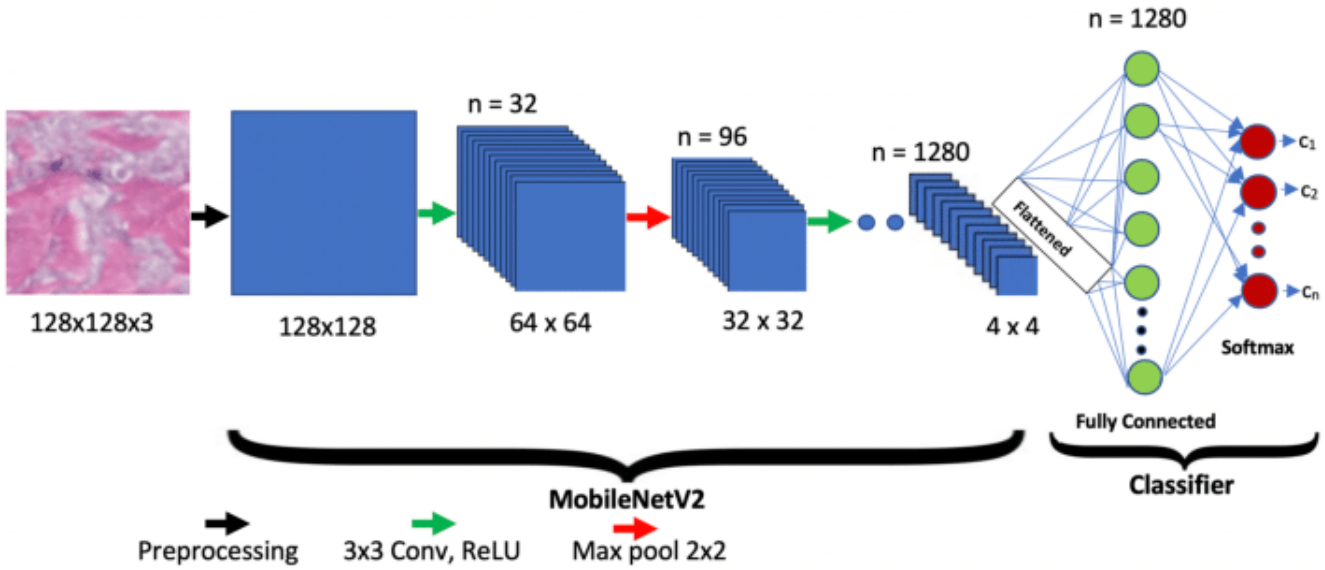




Practical sessions : Lab 2
Part 3 : Transfert-Learning for fall recognition (Bonus)

Use a pre-trained efficient solver : MobileNetV2 and perform fall recognition.

- Load pre-trained model and modify it for the application,
- Training of the dense layer,
- Fine-tuning of the last convolutional layers.



Practical sessions : synthesis
Use your brain

At the end of the sessions :

- Lab 1 : activity recognition from sensors,
- Lab 2 : fall detection from images.

Question to develop : Using your knowledge from the lectures, and your imagination, how would you build a fall detection system for old people's homes ("maisons de retraite") ?
Be creative !