AI for Astrophysics: Advanced Neural Networks

David Cornu

LERMA, Observatoire de Paris, PSL

Doctoral course – ED AAIF 2024/2025

Lessons materials

Slides, exercises, codes, corrections and datasets are **available on GitHub** and will be updated regularly:

http://github.com/Deyht/ML_OSAE_M2

git clone https://github.com/Deyht/ML_OSAE_M2
git pull

Or download the repository in zip file

Avoid losing your work on forced pull updates by copying all files from the cloned repository into a working directory!

Do not copy and past content from git-hub pages (lead to format errors).
Use python up to 3.10 but not more recent.

Neural Networks for images

Fully connected networks has shown one weakness

➡ **They are inefficient for handling images !**

- Images are highly dimensional (lots of pixels!)
- They have a very high degree of invariance
  (mainly translation but also luminosity, color, rotation, ...)

Classical ANN can deal with images by considering each pixel of an image as an individual input but it is STRONGLY inefficient.

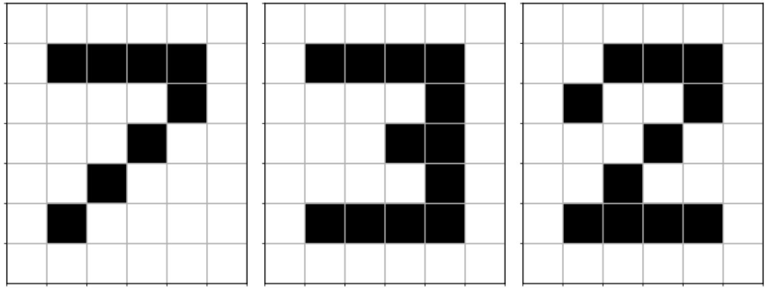
*A **highly dimensional** "dog" with ~0.5 Million pixels.*
Quite difficult to classify ...

Driven by the computer vision and pattern recognition community these issues have found a solution in the 90s with:

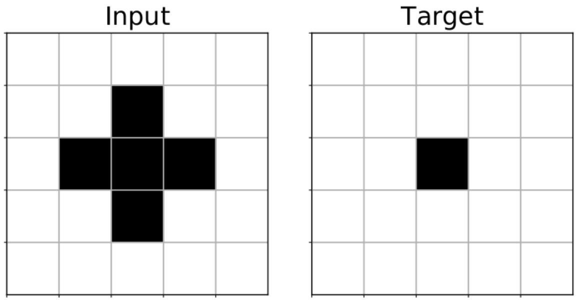
➡ **Convolutional Neural Networks** !

Spatially coherent information

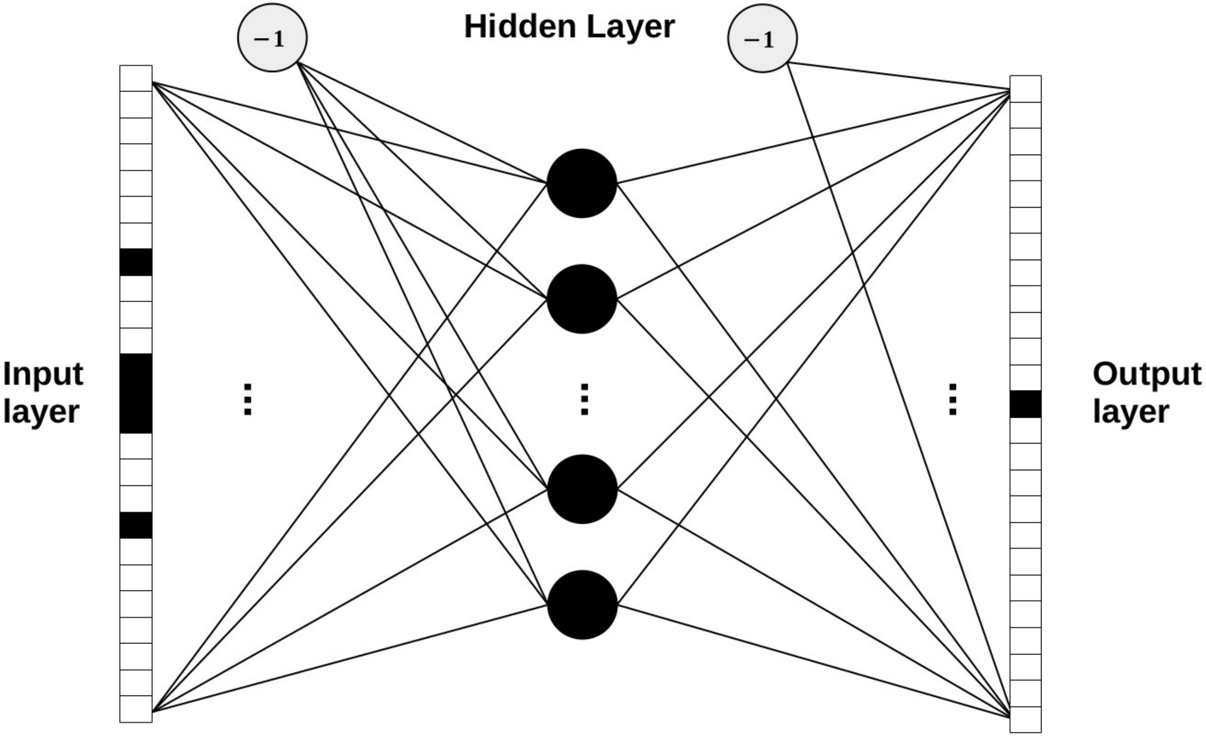
*Classical ANN can deal with images by considering each pixel of an image as an individual input but it is **STRONGLY inefficient**.*



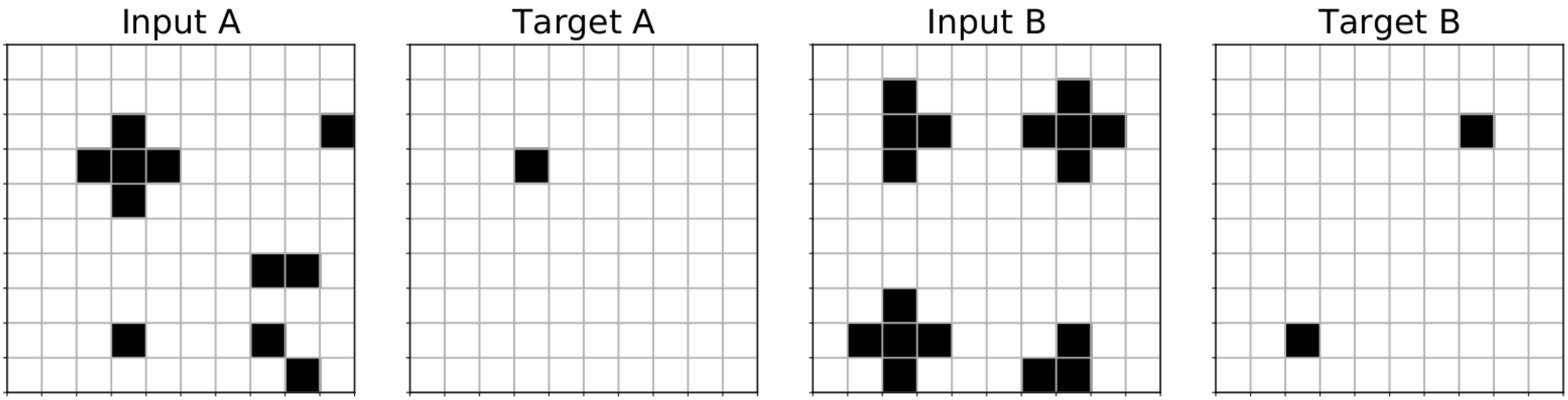
Simple digit representation as a 6x7 binary pixel image



Representation of a simple cross pattern on a 5 × 5 image as input, and the corresponding localization prediction on an equivalent size output image



Spatially coherent information : Pattern recognition

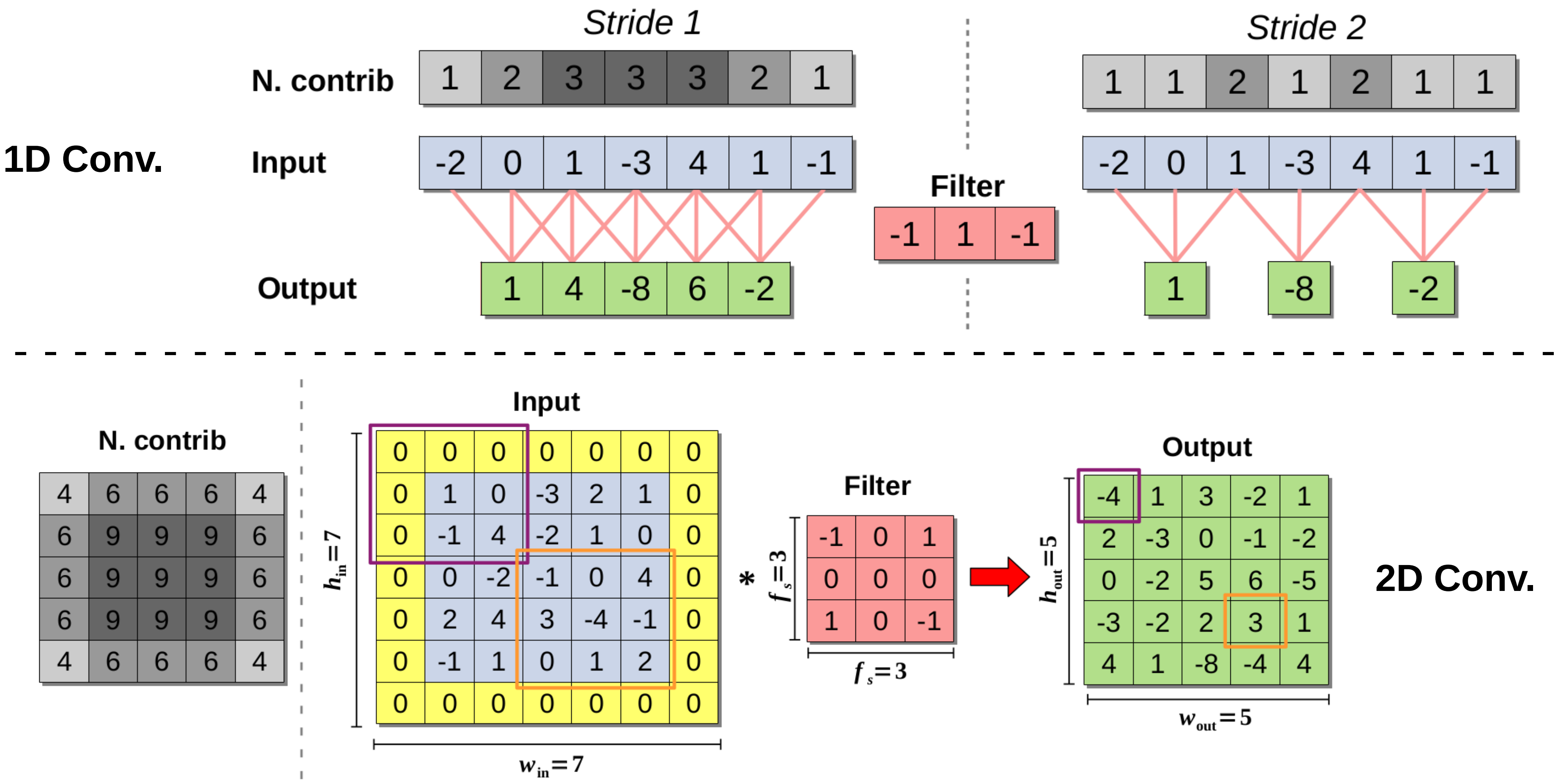


Looking for specific patterns can be automatized by scanning all the possible positions in the image.

In contrast, training a fully connected network to do the same task would require learning the presence or non-presence of the pattern at every possible position instead of learning the pattern once and only checking its presence at every position.

How to circumvent this behavior ? ➡ Use **Convolutional layers** !

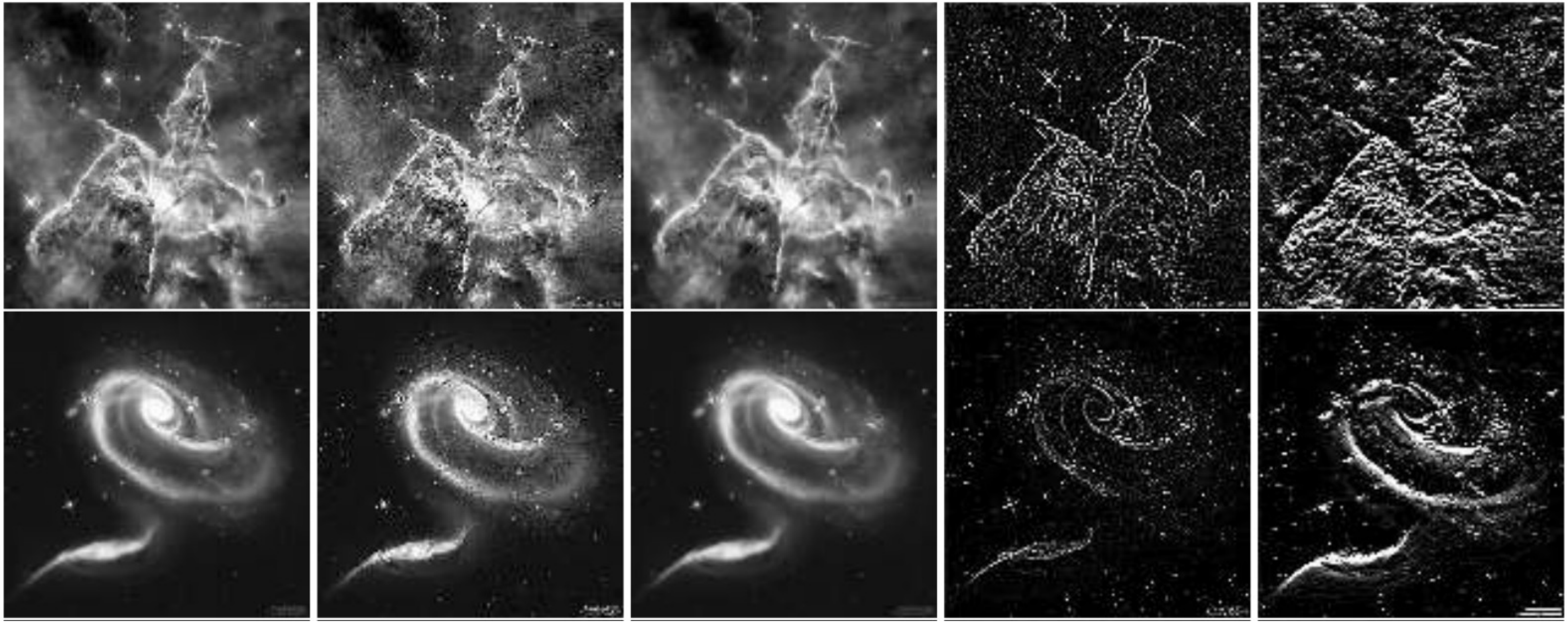
Convolution filter



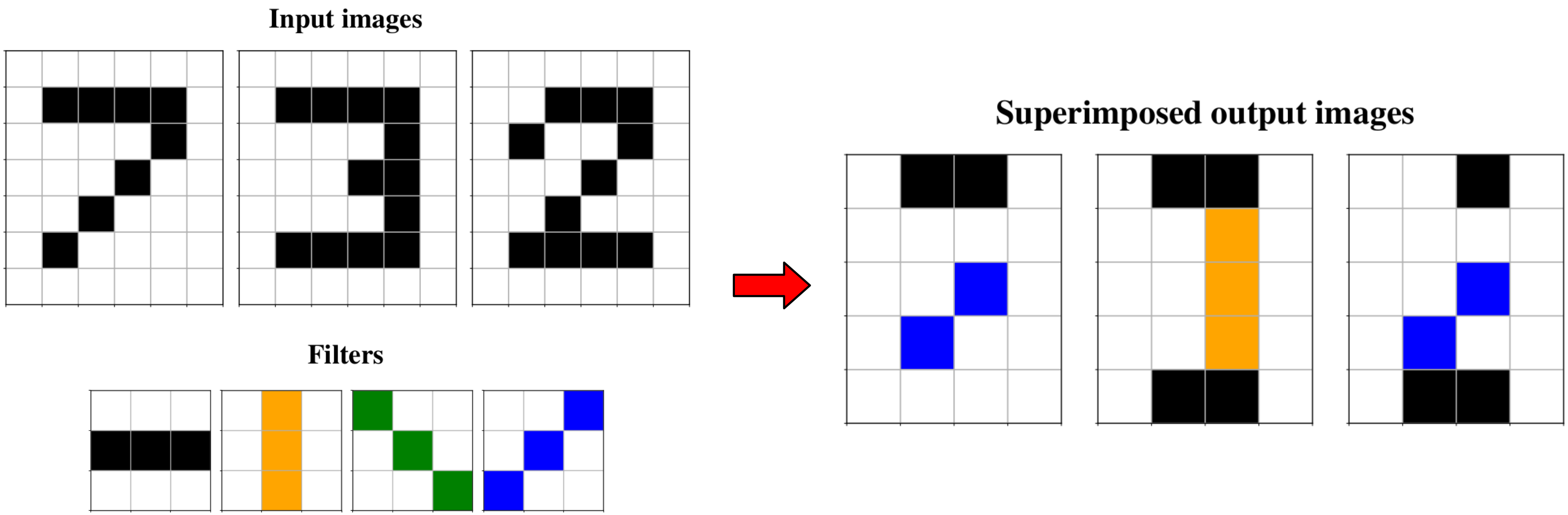
Stride 1 and Padding 1

Filter effect examples

No filter	Sharpen	Gaussian blur	Edge detector	Axis elevation
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$



Pattern recognition with several filters

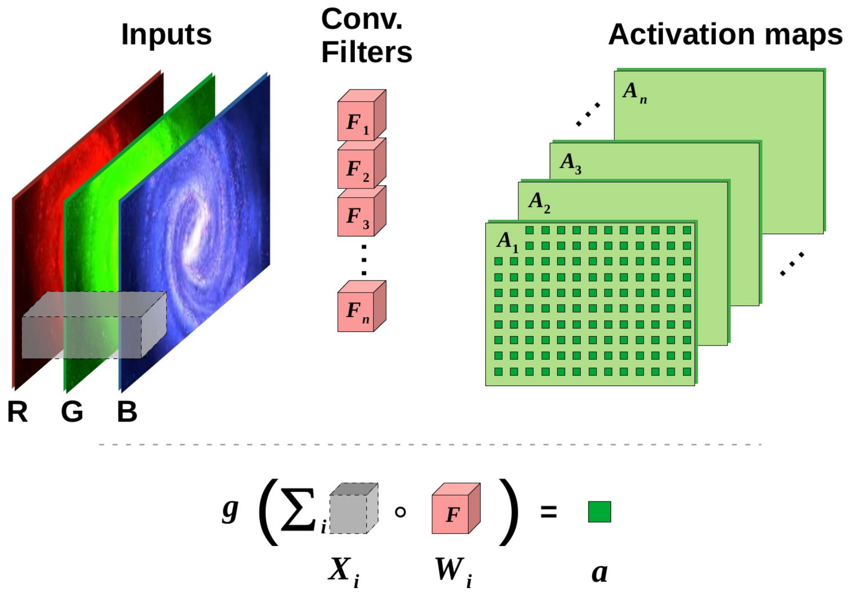


Each filter will produce its own activation map.

Combining the information from different activation maps allows to construct more complex patterns.

**Here the different activation maps are superimposed using color coding per filter*

Convolutional Neural Networks

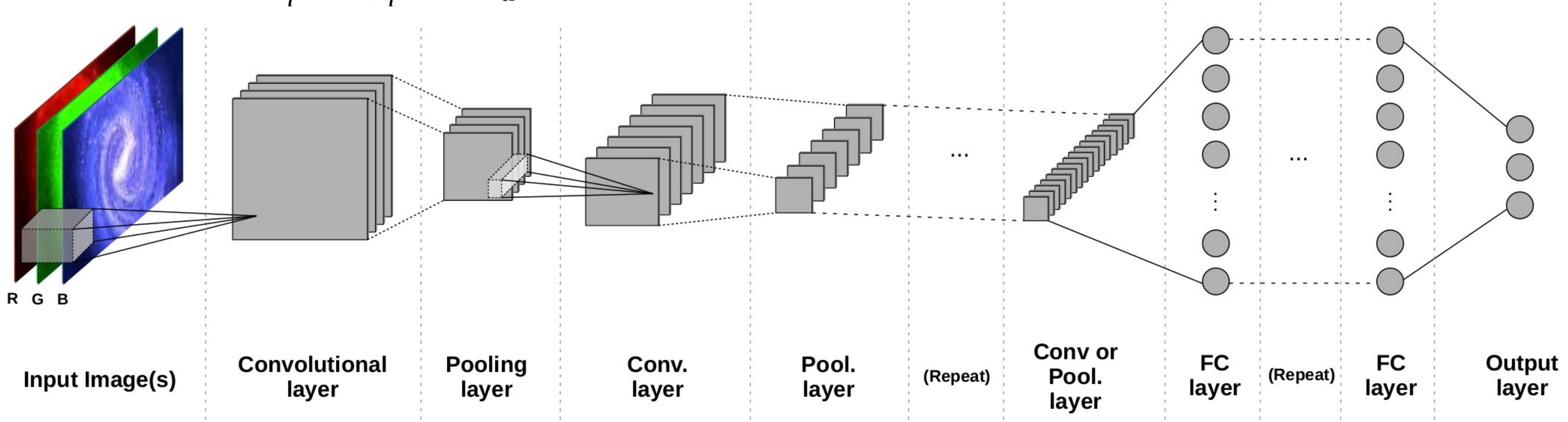


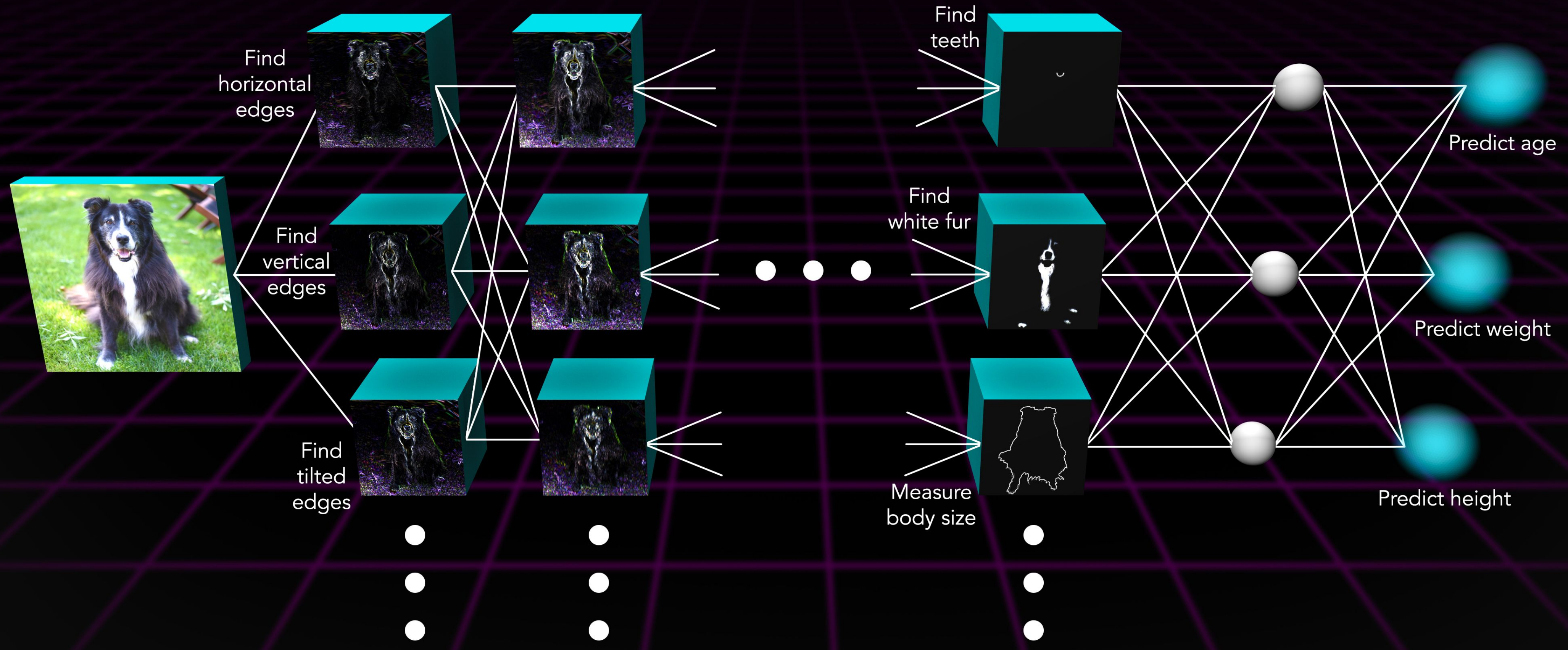
Each filter can be seen as a **single neuron** with one weight per input dimension in the filter.

BUT, the same weights are used at every position and the outputs are independent.

➡ **Translational equivariance !**

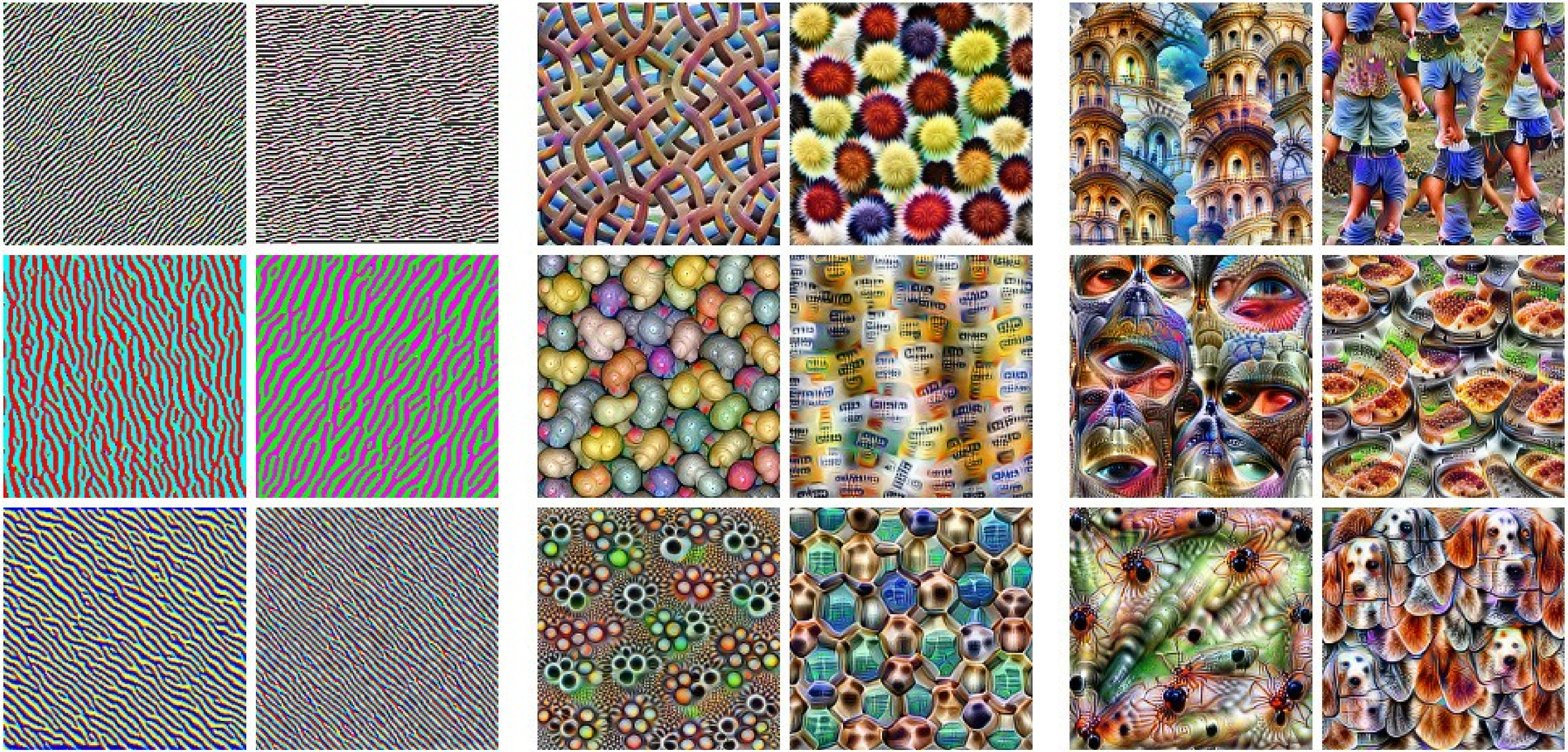
A network made of stacked convolutional layers can be tuned for Translation invariance.





Greg Stewart, SLAC National Accelerator Laboratory

Examples of filter maximization

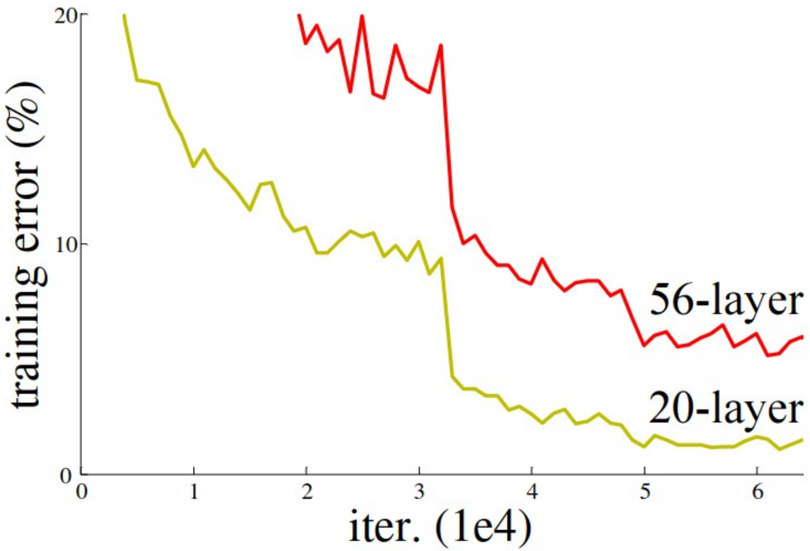


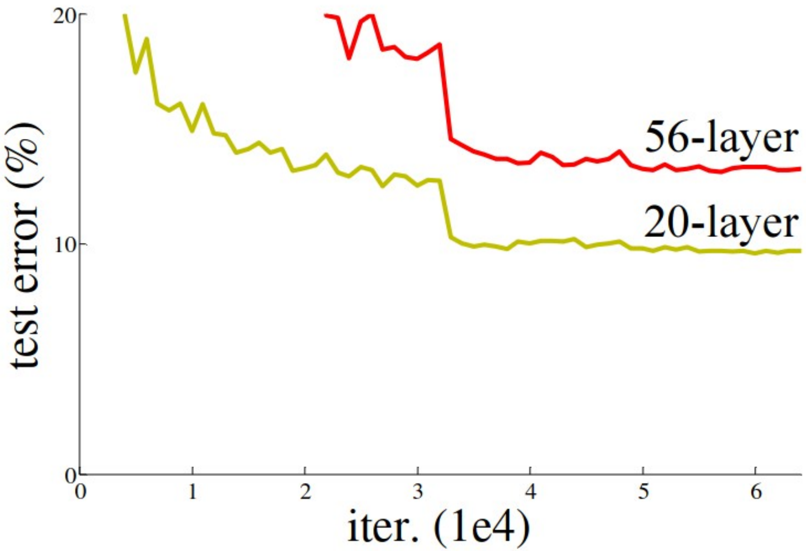
Edges (layer conv2d0) **Patterns** (layer mixed4a) **Objects** (layers mixed4d & mixed4e)

Example of input images that maximize specific filters activation at different depth in a classification network.

Vanishing Gradient Issue

From He et al. 2015





In principle, the deeper the network, the higher its expressivity should be as long as it is trained with enough data. However, it is nos the case in practice due to the gradient slowly getting smaller and smaller as it goes through more layers.

Still many approaches can mitigate this issue to construct network with hundreds of layers (e.g., changing the activation or having pass through connections between layers that are far away in the network).

The Rectified Linear Unit (ReLU)

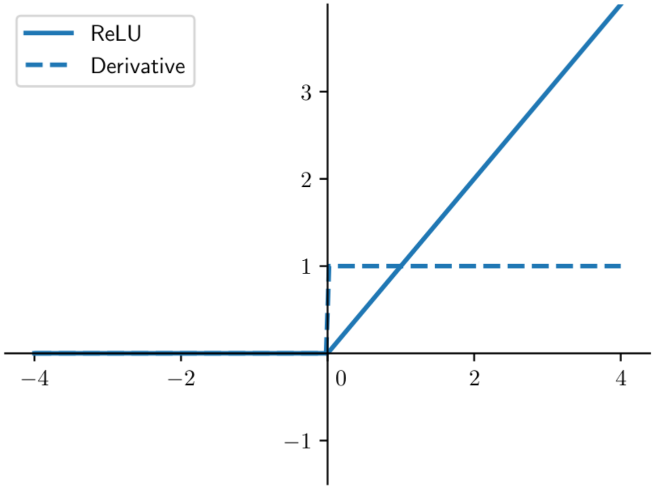
The ReLU (or its variance, the leaky-ReLU) has proven more efficient for CNN. It preserves a form of non-linearity and its constant derivative reduces **vanishing gradient problems**.

It is scale-invariant, and much faster to compute than other activation functions.

Using this activation, it becomes possible to construct much deeper networks.

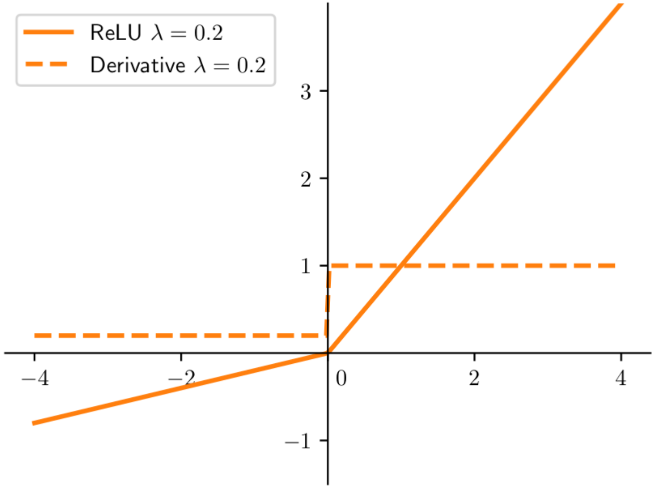
$$a_j = g(h_j) = \begin{cases} h_j & \text{if} \quad h_j \geq 0 \\ 0 & \text{if} \quad h_j < 0 \end{cases}$$

or $$a_j = g(h_j) = \max(0, h_j)$$



$$a_j = g(h_j) = \begin{cases} h_j & \text{if} \quad h_j \geq 0 \\ \lambda h_j & \text{if} \quad h_j < 0 \end{cases}$$

or $$a_j = g(h_j) = \max(0, h_j) + \min(0, \lambda h_j)$$

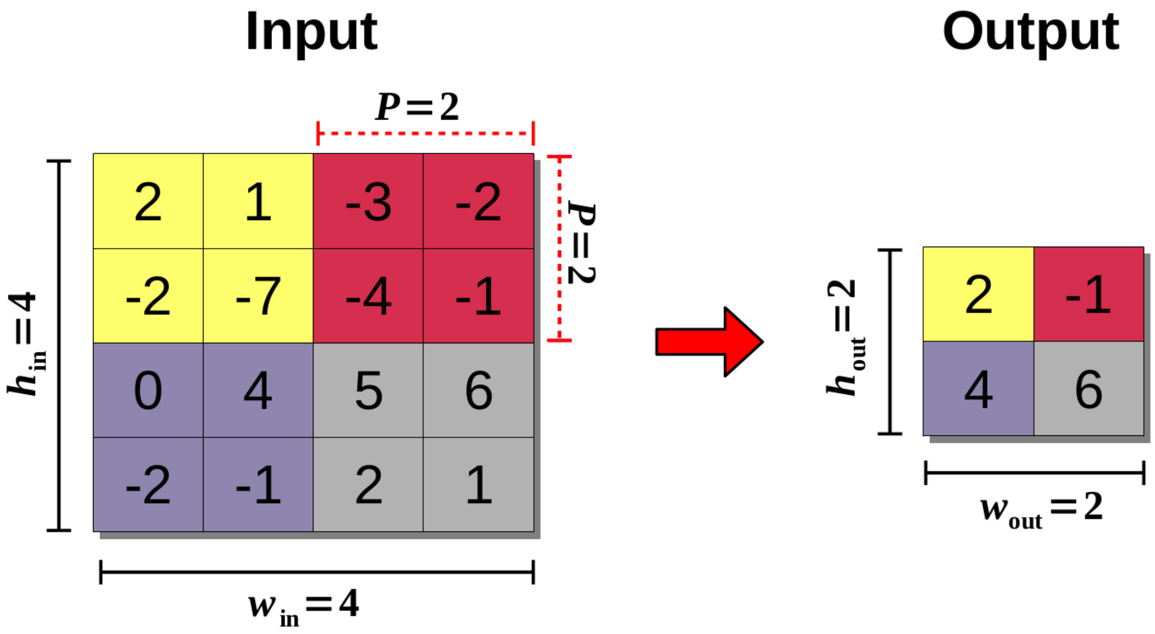


Dimensionality reduction: Pooling

A classical convolution operation is tuned to preserve the spatial dimensionality.
Still, it is most of the time necessary to **reduce the "image" size progressively**.

For classification tasks, the output layer is often reduced to a dense layer with a few neurons.

One way to reduce the spatial dimensionality it to use **Pooling layers** !

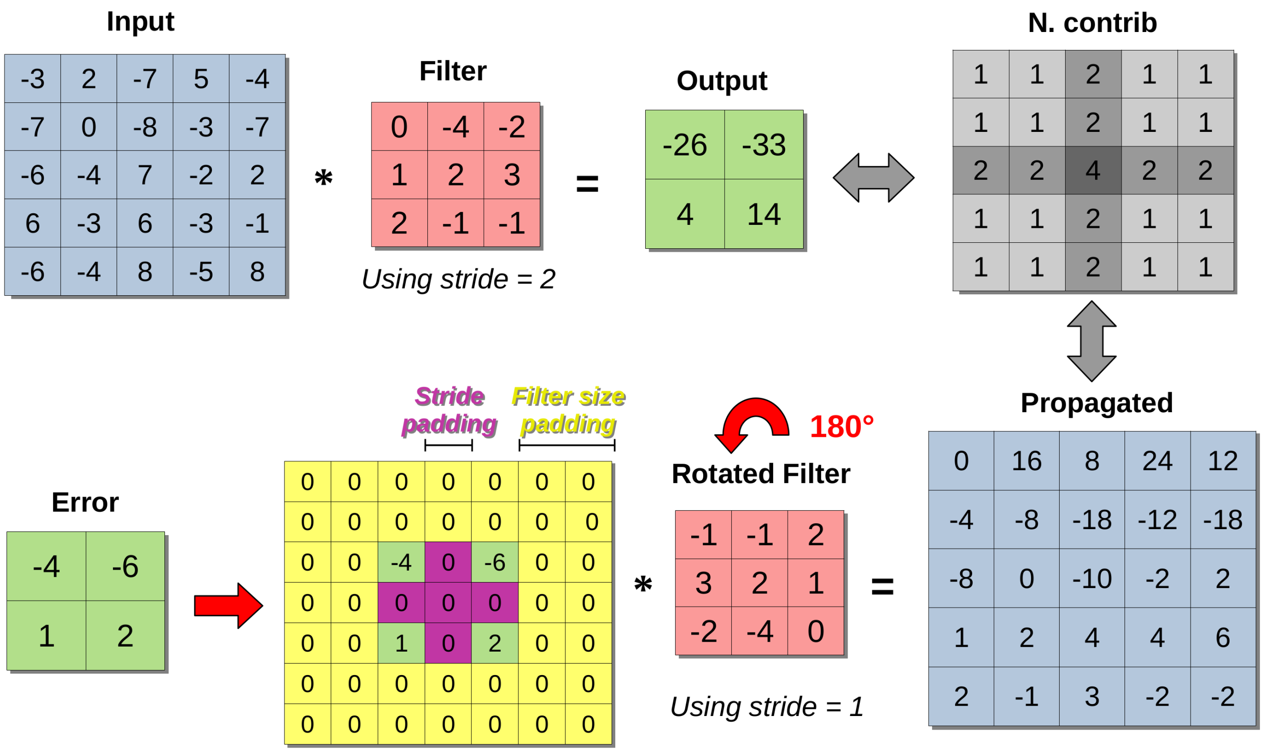


Different pooling methods exist, the most common being Max-Pooling and Average-Pooling. The pooling size can be modified, but most of the network architectures reduce each spatial dimension by a factor of two.

Learning the filters

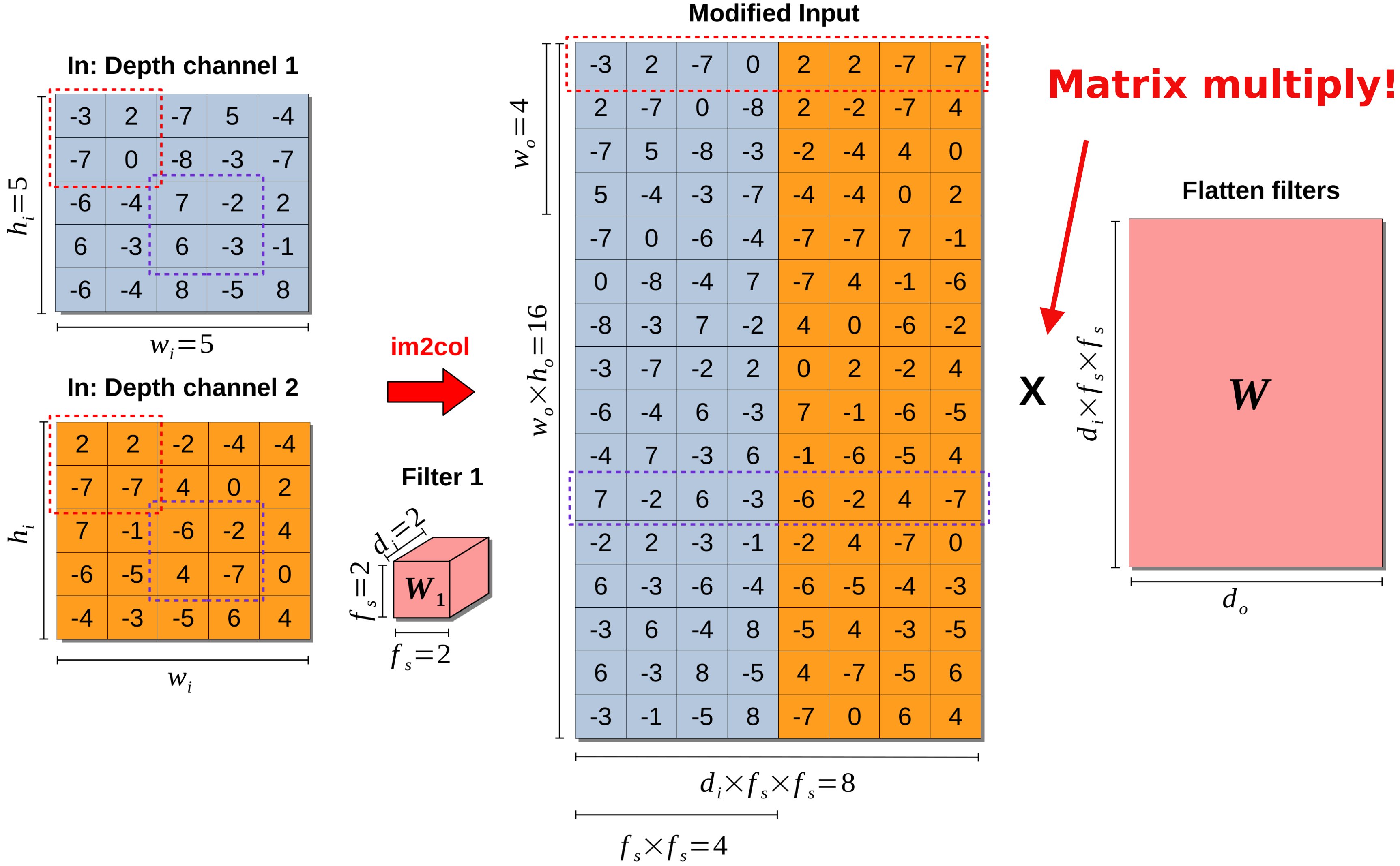
Like fully connected layers, the **convolutional filters** can be learned using **backpropagation** of the error measured at the output layer.

The error is propagated using a transposed convolution operation, which can be expressed with a classical convolution operation using simple transformations on specific layer elements.

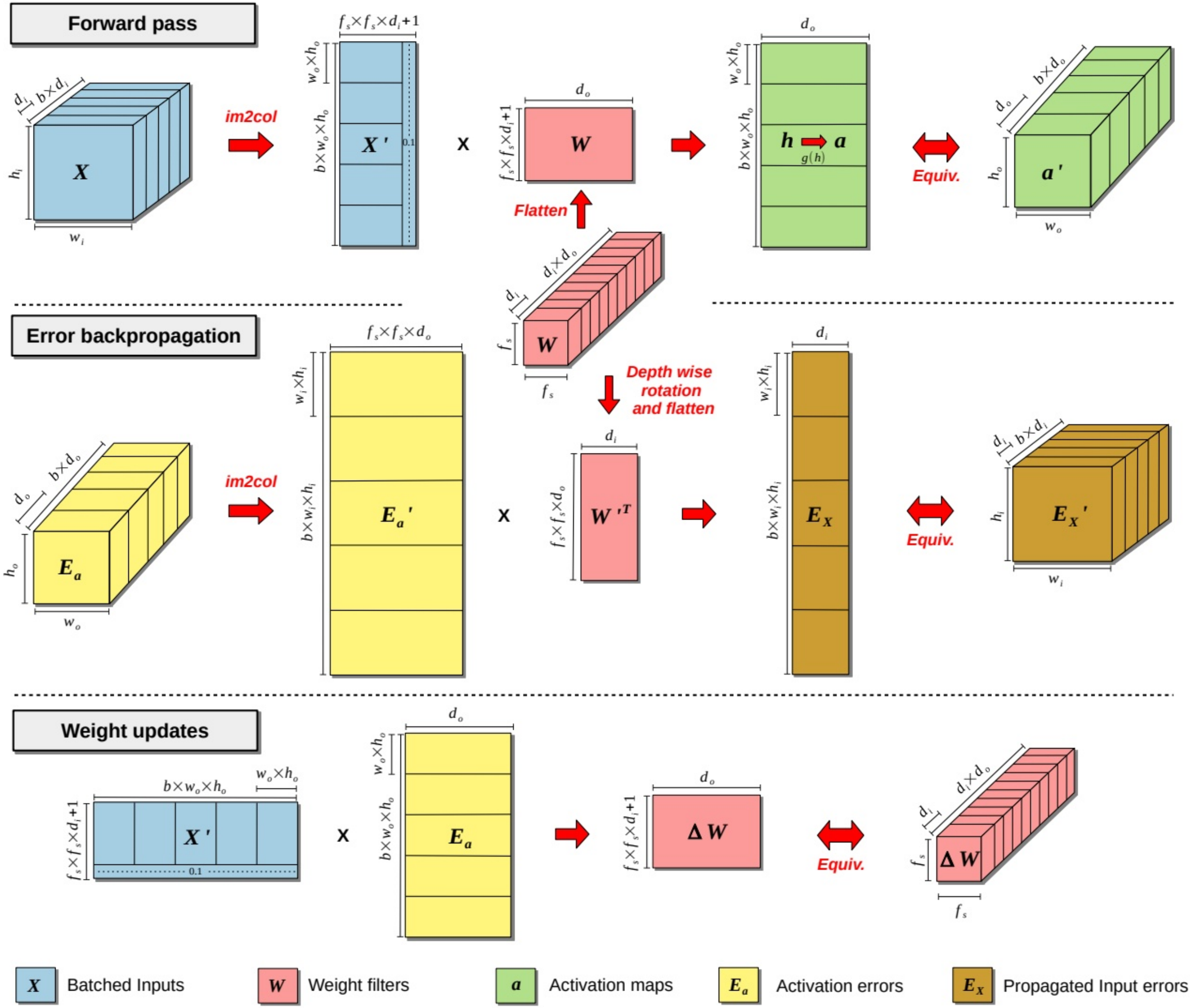


*Learning the convolutional filters is often considered to be the definition of “**Deep Learning**”.*

The Im2col transformation



Convolutional layer complete matrix formalism

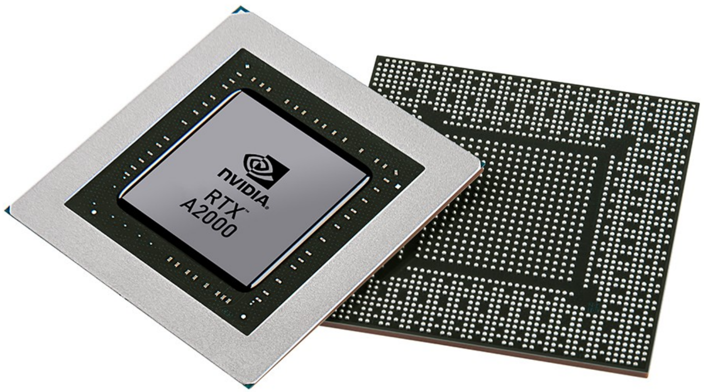


What about computing on GPU?

GPU (Graphical Processing Unit) are massively parallel computing chips dedicated to SIMD like operations (thousands of cores). Most image processing algorithm apply the same transformation to millions of pixels, hence the SIMD formalism.

GPU have the same form factor than CPU but usually come as a dedicated daughter board with their own large cooling system as they can have a much higher power draw than CPU!

GPUs are not suited for all tasks, but for those they where designed for, they pack a huge amount of computing power, which include matrix multiplication!



Nvidia H100 GPU spec-sheet (AI dedicated)

Graphics Processor	
GPU Name:	GH100
Architecture:	Hopper
Foundry:	TSMC
Process Size:	4 nm
Transistors:	80,000 million
Density:	98.3M / mm²
Die Size:	814 mm²

Graphics Features	
DirectX:	N/A
OpenGL:	N/A
OpenCL:	3.0
Vulkan:	N/A
CUDA:	9.0
Shader Model:	N/A

Graphics Card	
Release Date:	Mar 21st, 2023
Generation:	Tesla Hopper (Hxx)
Predecessor:	Tesla Ada
Production:	Active
Bus Interface:	PCIe 5.0 x16

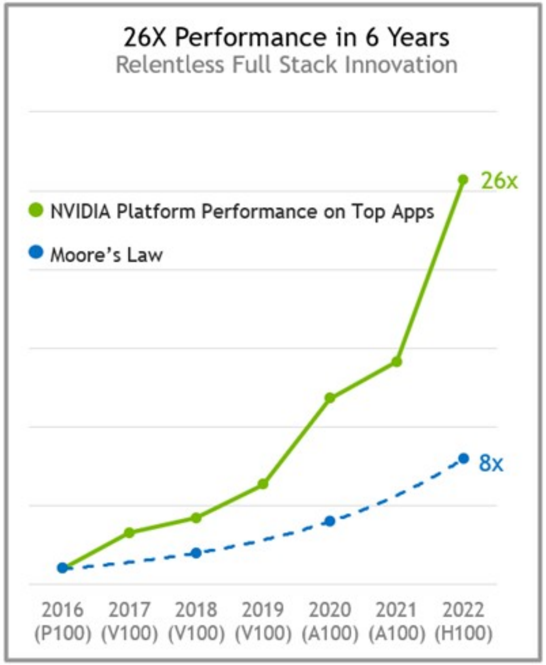
Board Design	
Slot Width:	Dual-slot
Length:	268 mm 10.6 inches
Width:	111 mm 4.4 inches
TDP:	350 W
Suggested PSU:	750 W
Outputs:	No outputs
Power Connectors:	1x 16-pin
Board Number:	P1010 SKU 200

Clock Speeds	
Base Clock:	1095 MHz
Boost Clock:	1755 MHz
Memory Clock:	1593 MHz 3.2 Gbps effective

Render Config	
Shading Units:	14592
TMUs:	456
ROPs:	24
SM Count:	114
Tensor Cores:	456
L1 Cache:	256 KB (per SM)
L2 Cache:	50 MB

Memory	
Memory Size:	80 GB
Memory Type:	HBM2e
Memory Bus:	5120 bit
Bandwidth:	2,039 GB/s

Theoretical Performance	
Pixel Rate:	42.12 GPixel/s
Texture Rate:	800.3 GTexel/s
FP16 (half):	204.9 TFLOPS (4:1)
FP32 (float):	51.22 TFLOPS
FP64 (double):	25.61 TFLOPS (1:2)



From Nvidia

From Techpowerup

GPU architecture

The GPU is specialized for highly parallel computations and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control. The schematic Figure 1 shows an example distribution of chip resources for a CPU versus a GPU.

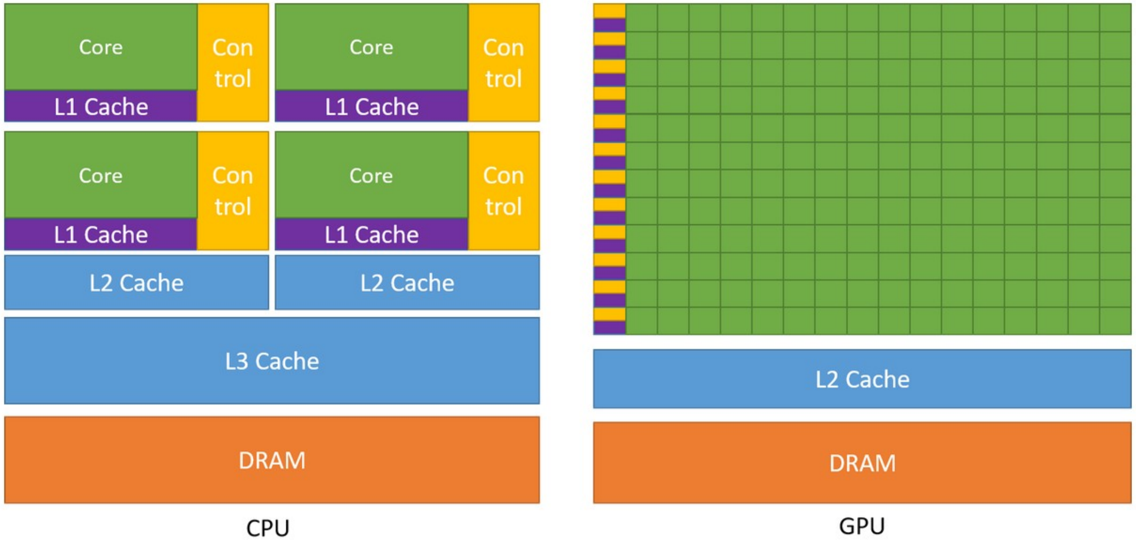


Figure 1: *The GPU Devotes More Transistors to Data Processing*

From Nvidia CUDA programming guide

GPU architecture

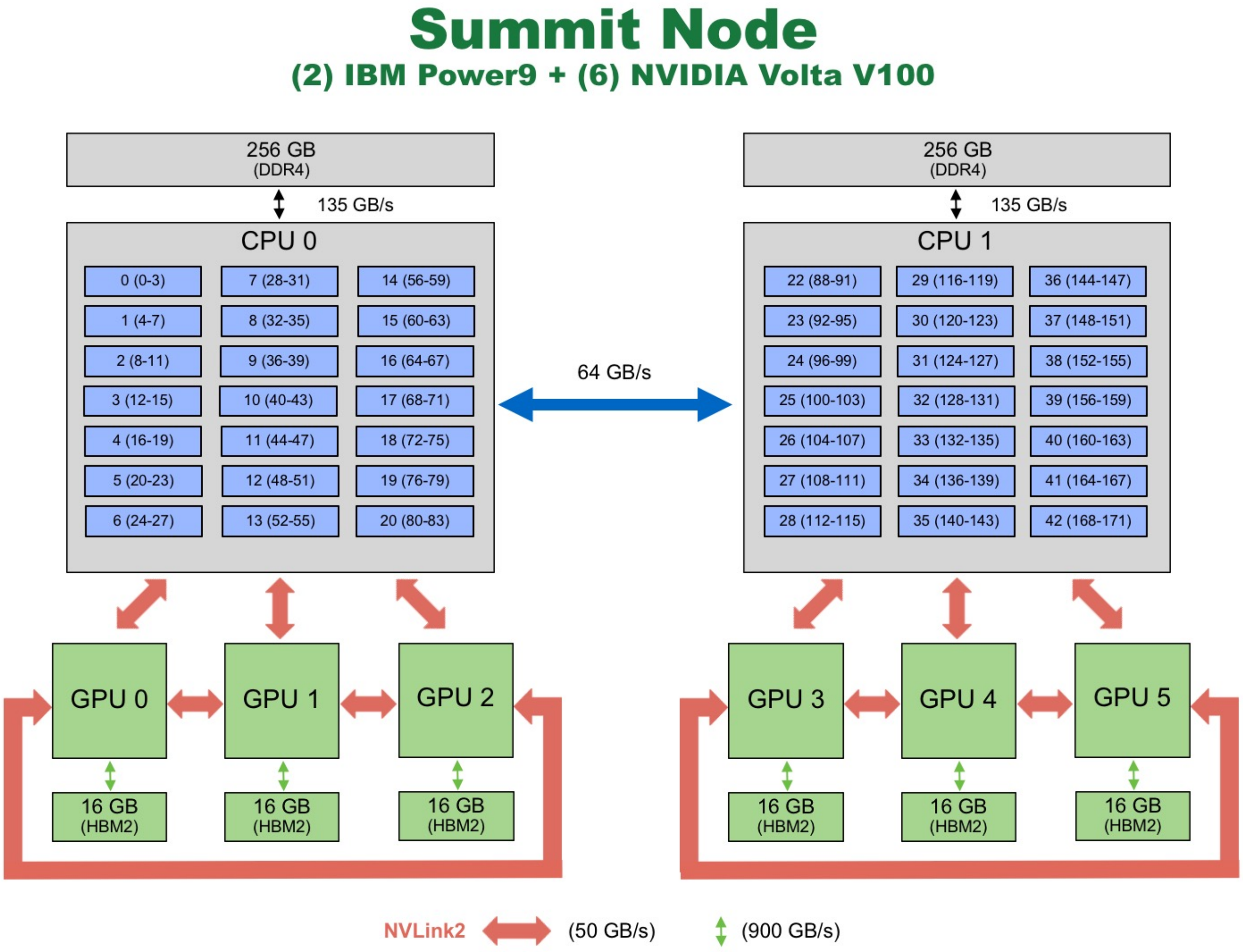
Ex. of a GP104 – Tesla P100

The GPU is equipped with a shared “on board” memory. All SM share the same large L2 cache.

Each SM has a dedicated cache and can launch multiple instruction blocs through multiple warp schedulers.

Inside each SM, there are several CUDA cores and other dedicated compute units.

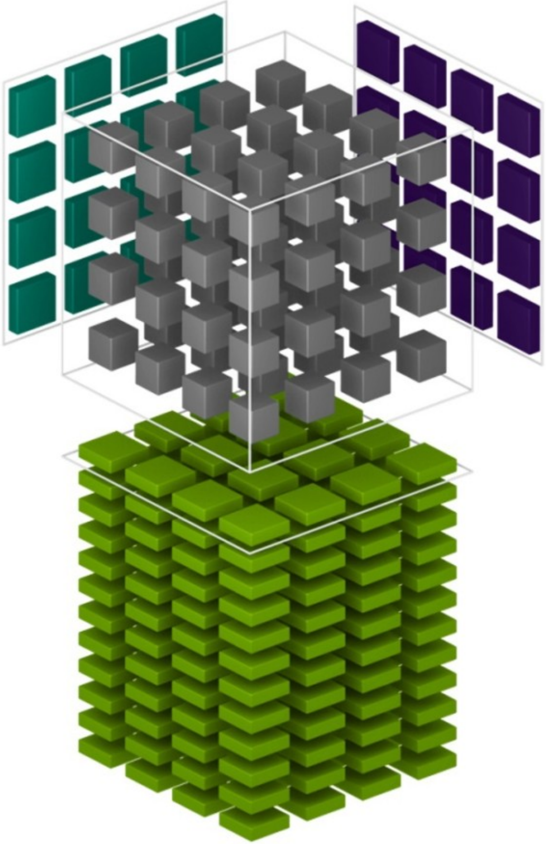
Distribution in GPU clusters

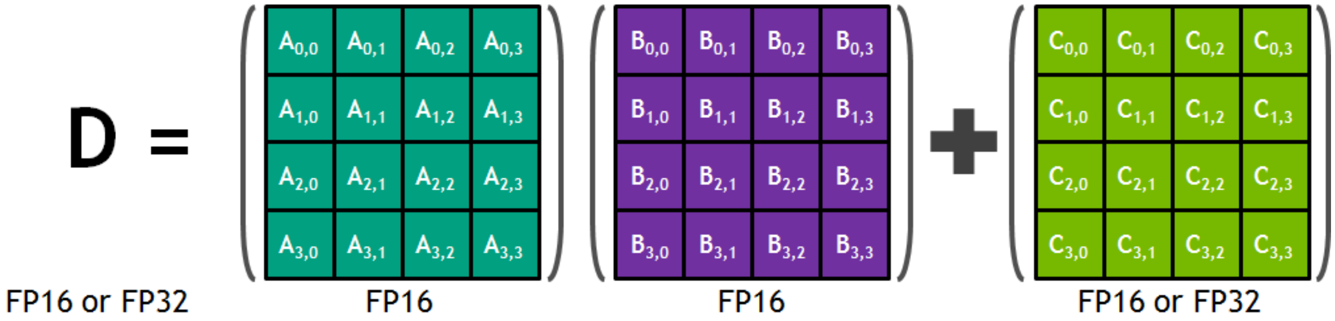


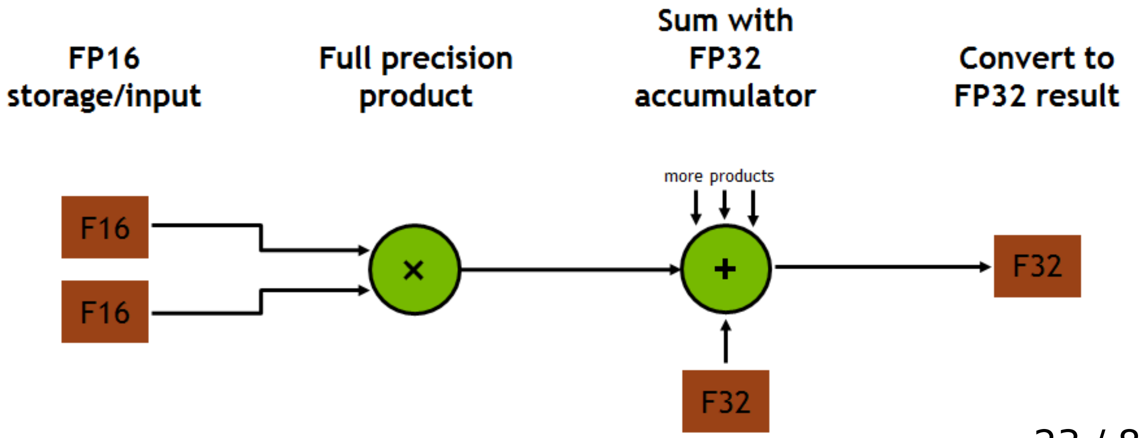
Nvidia Tensor Cores

Optimized Warp Matrix Multiply Add (WMMA) instructions !

CuBLAS can be set to used tensor core through the gemmEX function.

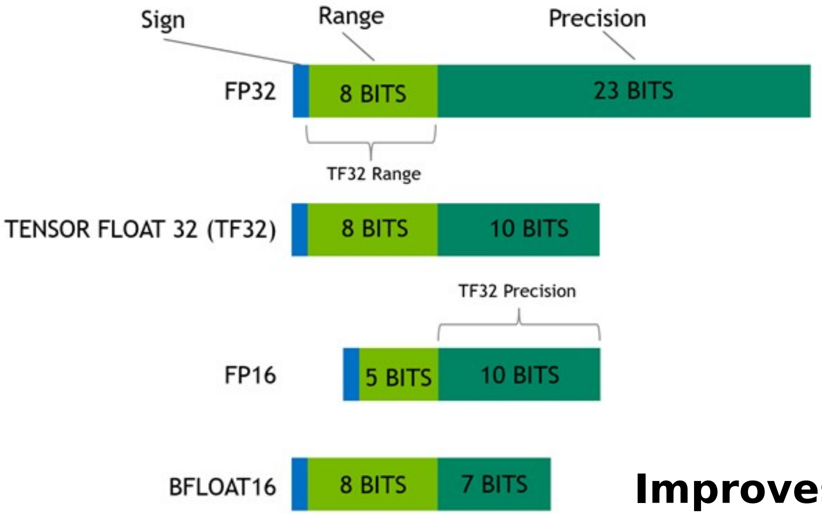


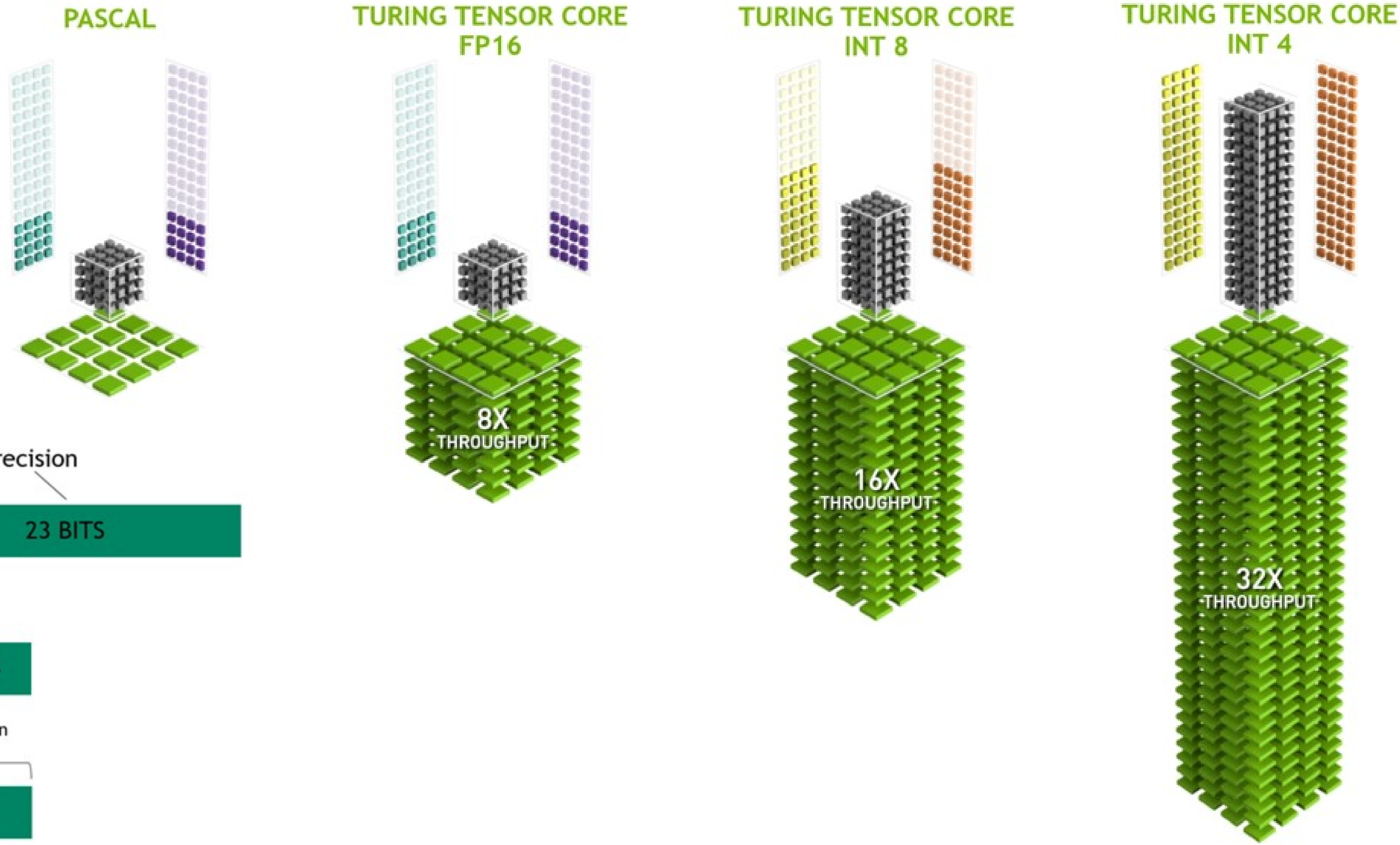




Nvidia Tensor Cores reduced quantization

$$1.2345 = \underbrace{12345}_{\text{mantissa}} \times 10^{\overbrace{-4}^{\text{exponent}}}$$

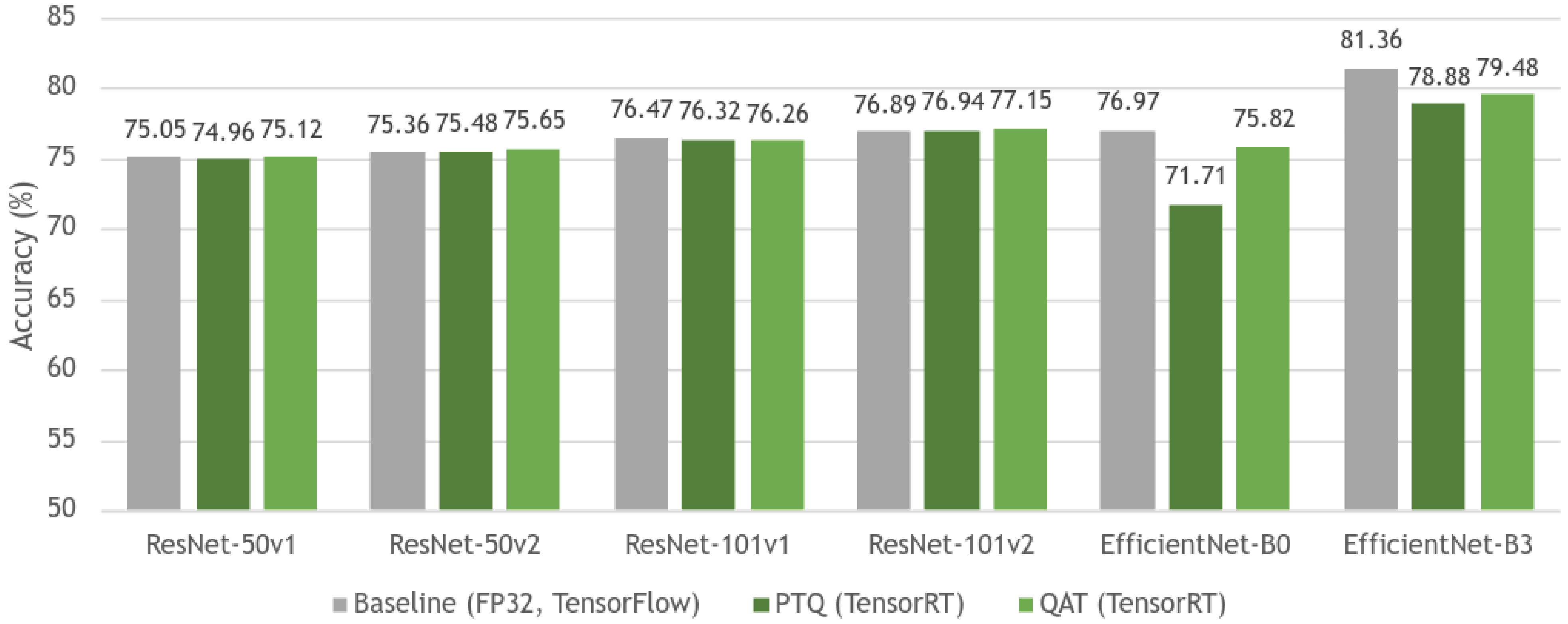




Improves compute throughput AND reduces data size,
→ reduce memory footprint and required bandwidth

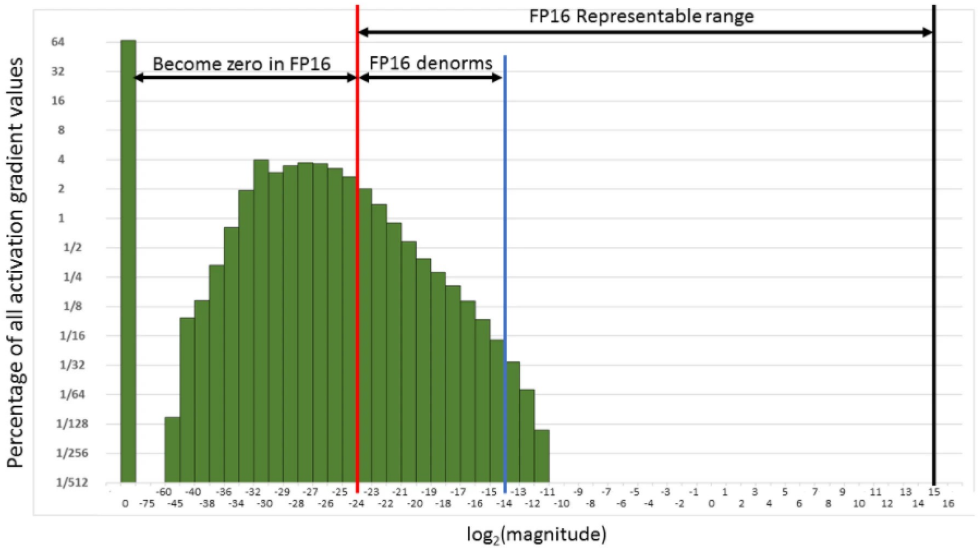
Quantization effect on AI model accuracy

All models quantized to INT8, accuracy for ImageNET-2012



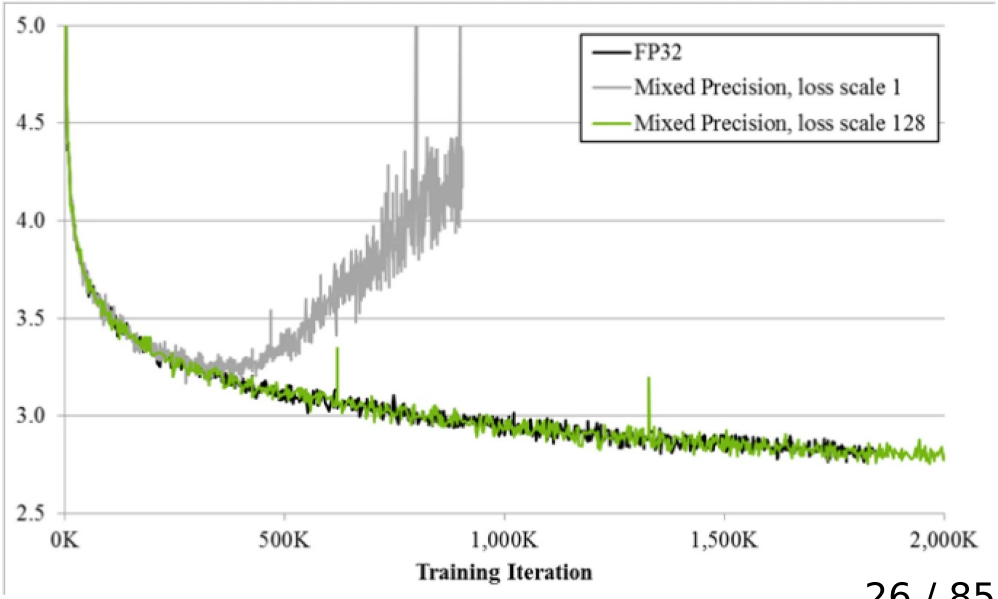
PTQ = Post training quantization
QAT = Quantization aware training

Mixed precision training



Reduced bit count variables have smaller representable ranges. This can lead to strong gradient vanishing problems.

This problem can first be mitigated by preserving an FP32 copy of the weights for accumulating the updates.



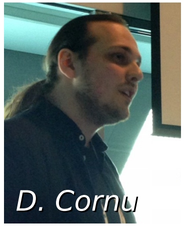
To further reduce this issue, a scaling is applied on the output loss. All propagated values are naturally scaled so they are more likely to be in the proper range.

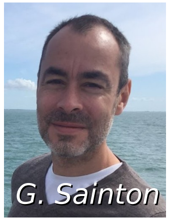
At weight update time the correction is scaled down by the same factor.

From Nvidia « Mixed precision training »

CIANNA

Development team





Convolutional Interactive Artificial Neural Networks by/for Astrophysicists

General purpose framework BUT developed for **astronomical applications**

CPU OpenMP OpenBLAS / GPU NVIDIA CUDA

x86 Intel/AMD & ARM **Supports modern TC acceleration** **Python interface**

Work on a wide variety of hardware from IoT to super-computing facilities

github.com/Deyht/CIANNA

Open source – Apache 2 license

July 24, 2024 (V-1.0.0.0) Software Open

Deyht/CIANNA: CIANNA V-1.0

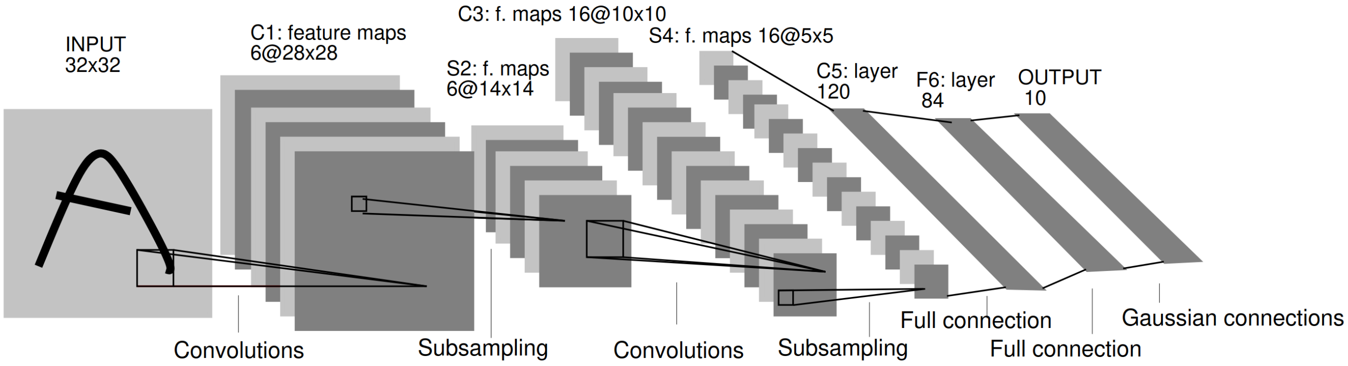
Simple examples: MNIST

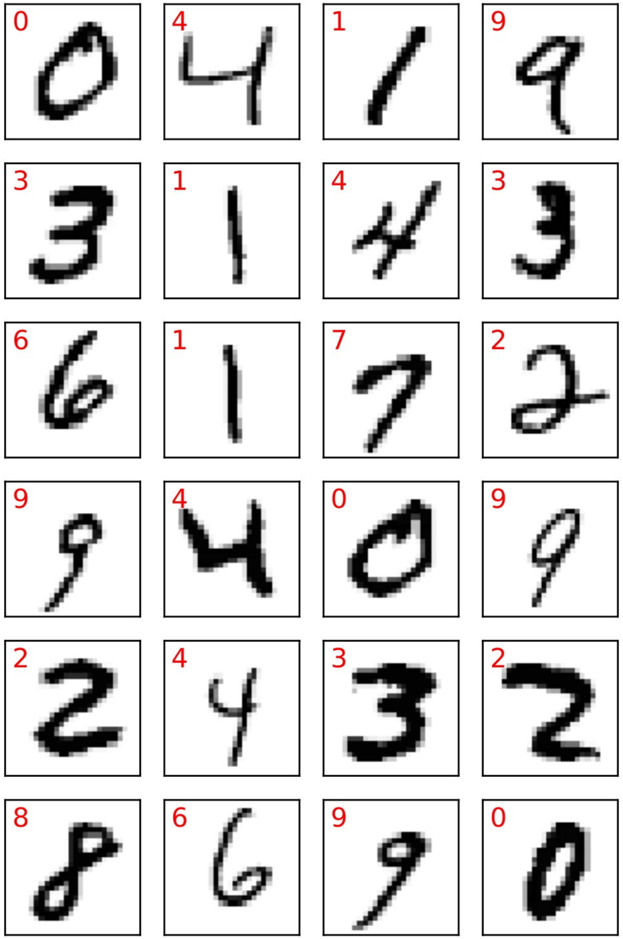
The well-known **MNIST** (Modified NIST's special dataset) dataset consists of handwritten digits from 500 different writers expressed as 28x28 grayscale images.

It is freely accessible in the form of a 60000 image training set, 10000 images validation, set and a 10000 image test set.

Very simple CNN architectures can achieve over **99.3 classification accuracy** on this dataset.

LeNet 5 network architecture (from LeCun et al. 1998):





How to use CIANNA for MNIST ?

Example script over MNIST, with an LeNet5 *like* network.

Interface is similar to widely adopted frameworks.

The **full interface documentation** is available on the github repo as a Wiki page listing all available functions with their descriptions.

Several example scripts are provided as Google Colab notebooks.

```
#CIANNA initialization
cnn.init(in_dim=i_ar([28,28]), in_nb_ch=1, out_dim=10,
    bias=0.1, b_size=16, comp_meth="C_CUDA",
    dynamic_load=1, mixed_precision="FP32C_FP32A")

#Create data subsets (from numpy arrays)
cnn.create_dataset("TRAIN", size=60000, input=data_train, target=target_train)
cnn.create_dataset("VALID", size=10000, input=data_valid, target=target_valid)
cnn.create_dataset("TEST", size=10000, input=data_test, target=target_test)

#Define the network structure sequentially
cnn.conv(f_size=i_ar([5,5]), nb_filters=8 , padding=i_ar([2,2]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([5,5]), nb_filters=16, padding=i_ar([2,2]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.dense(nb_neurons=256, activation="RELU", drop_rate=0.5)
cnn.dense(nb_neurons=128, activation="RELU", drop_rate=0.2)
cnn.dense(nb_neurons=10, strict_size=1, activation="SMAX")

#Training loop configuration and launch
cnn.train(nb_iter=20, learning_rate=0.004, momentum=0.8, confmat=1, save_every=0)

#Evaluate network prediction after training
cnn.forward(repeat=1, drop_mode="AVG_MODEL")
```

Example results on MNIST

Predicted

Class	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	Recall
C0	976	0	1	0	0	0	1	1	1	0	99.6%
C1	0	1132	1	0	1	0	0	0	1	0	99.7%
C2	1	1	1027	0	1	0	0	1	1	0	99.5%
C3	0	0	1	1004	0	3	0	1	1	0	99.4%
C4	0	0	1	0	972	0	1	0	1	7	99.0%
C5	0	0	0	4	0	886	1	0	0	1	99.3%
C6	3	2	0	0	1	2	949	0	1	0	99.1%
C7	0	2	3	0	0	0	0	1020	1	2	99.2%
C8	0	0	1	1	0	1	1	1	968	1	99.4%
C9	0	0	0	0	3	1	0	4	0	1001	99.2%
Precision	99.6%	99.6%	99.2%	99.5%	99.4%	99.2%	99.6%	99.2%	99.3%	98.9%	99.35%

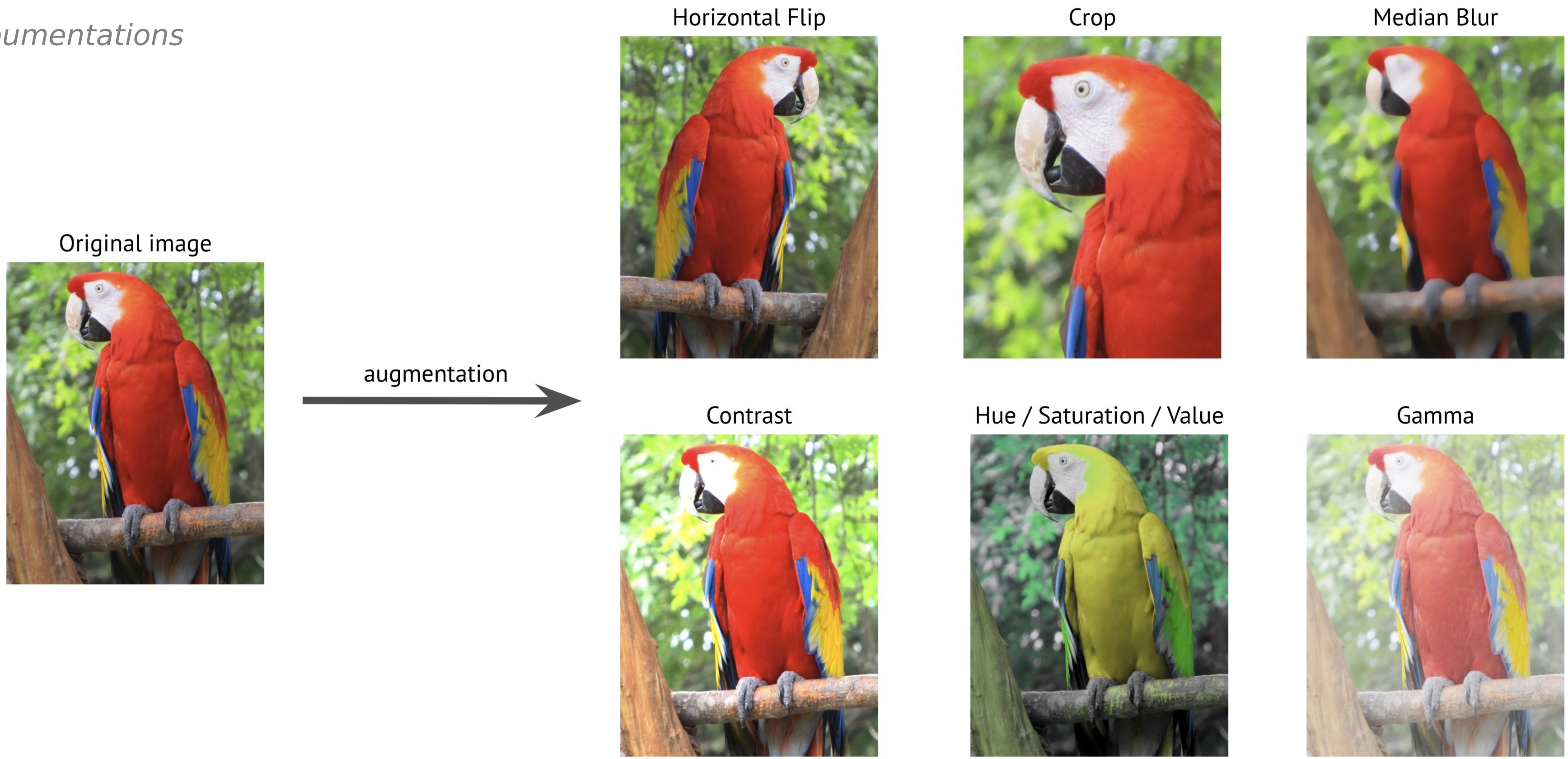
Actual

Practical work:

Reproduce this result using the provided notebooks. Try to modify the architecture *(filter size, #filters, #neurons, batch size, learning rate, etc.)* and estimate the impact of individual changes on the training / inference time and final accuracy.

Data augmentation

From Albumentations



Depending on the task, one can choose a set of transforms that do not alter the labeling corresponding to the image (here the class). This allow to increase the coverage of the feature space without the need of new data.

Augmentation does not solve all the dataset size issues, it usually **cannot create new contexts** nor it can generate features that are simply missing in the original dataset.

More advanced image classification

ASIRRA (Animal Species Image Recognition for Restricting Access): 25000 images, 50 % cats, 50 % dogs.

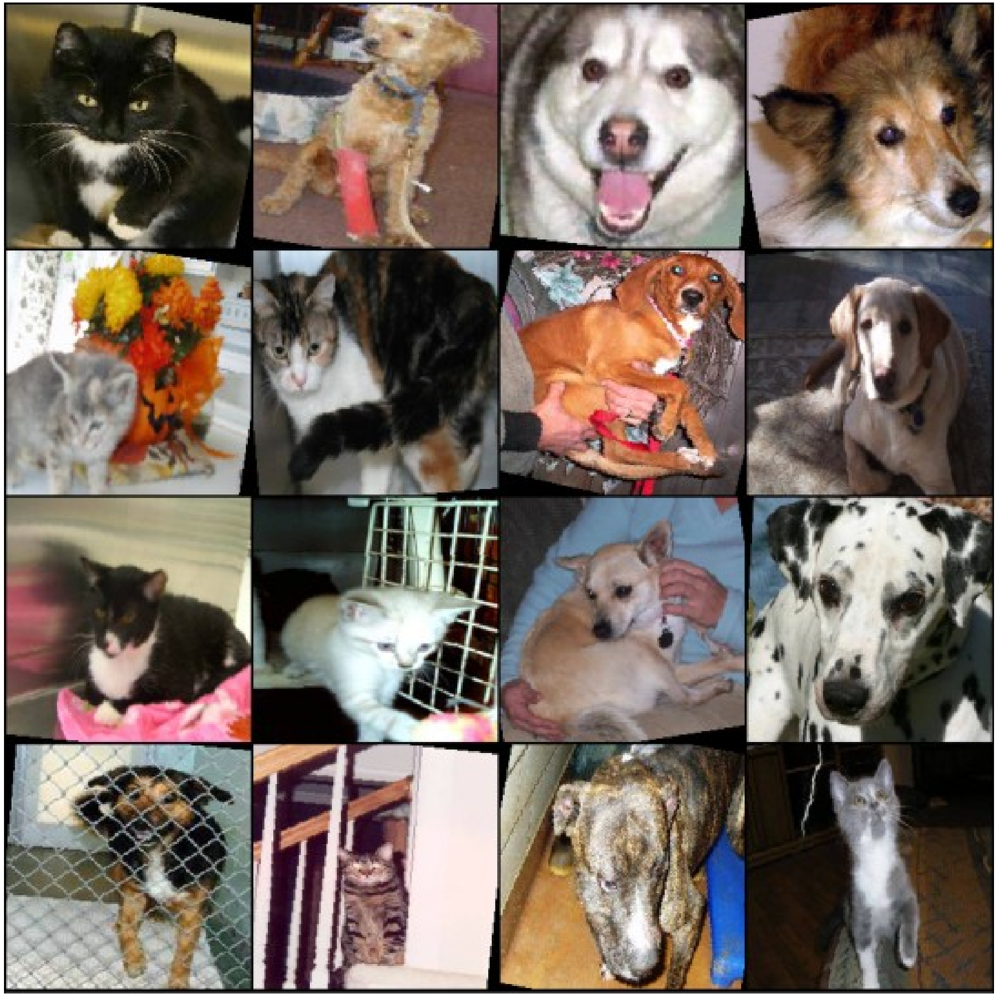
Typical of a **CAPTCHA** task or **HIP** (Human Interactive Proof) because the task is easy and quick for humans.

→ Nowadays modern CNN tools have more than 90% accuracy on ASIRRA !

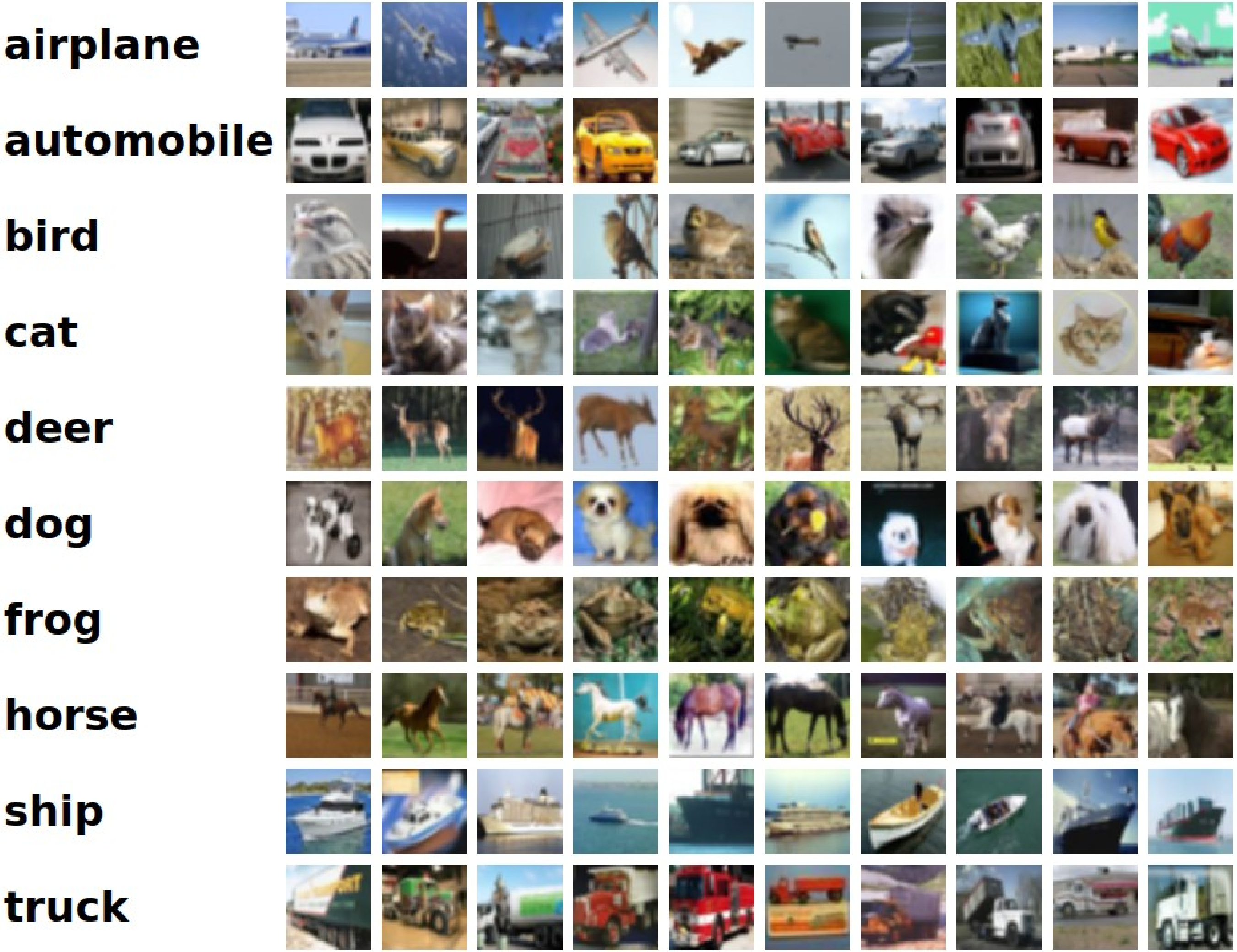
We can generate standardize and augmented images from the dataset using dynamic image augmentation.

Practical work:

Using the provided dataset and data loading script, implement a CNN (with either framework) for this classification. Explore the architectures, hyper-parameters, and augmentation setups by yourself and try to optimize the prediction result.



A more complex task, the CIFAR-10 dataset



60000 images of 32x32 pixels labeled into 10 classes.

50000 images for training and 10000 for testing.

Modern methods can reach 97% accuracy on this dataset.

The lower resolution combined to the increase in number of classes and image context make it a more difficult task to solve that for ASIRRA.

Practical work:

Do the same work you’ve done for ASIRRA on this new dataset. Try to identify the key differences between the dataset and explore how it affect the optimal architecture.

1D CNN for spectra

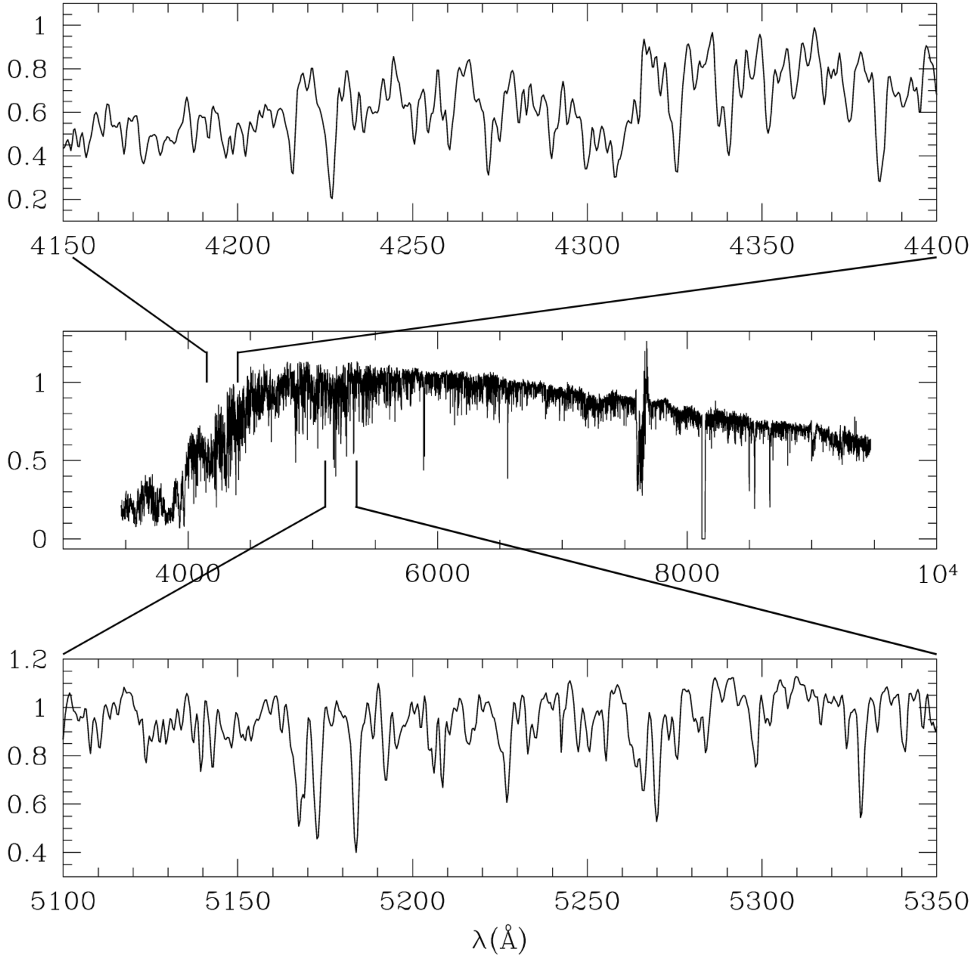
A spectra is a continuous 1D data structure decomposed into many frequency bins.

Like an image it can exhibit recurrent patterns that could be search in a **translation equivariant** way.

Practical work:

Using the stellar spectra dataset from the Perceptron and MLP parts of the course, build a 1D CNN that performs the classification. As before, a notebook is provided as a starting point.

As the number of frequency bins is much higher than in a typical image, you might need to use larger filters and to reduce the convoluted dimensionality more aggressively.



1D CNN for spectra

From Kessler et al. 2025 (in prep)

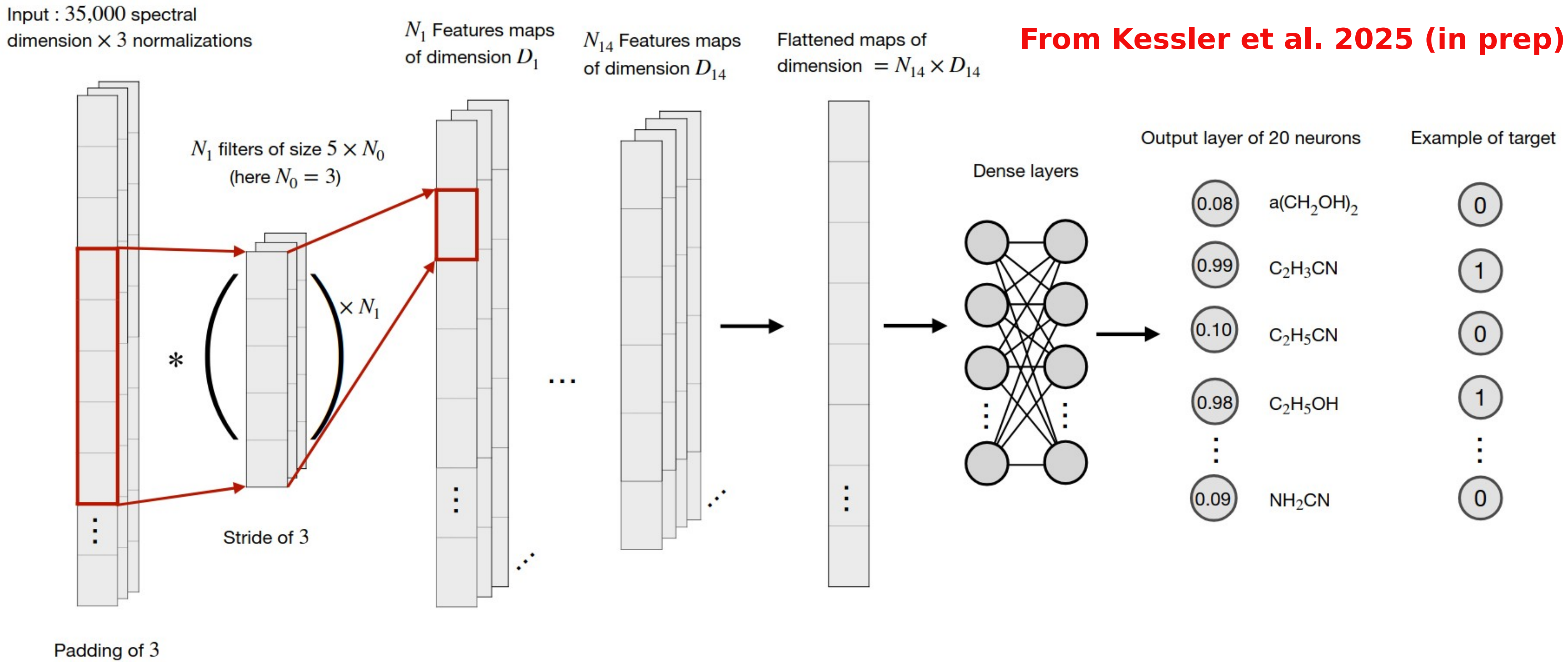
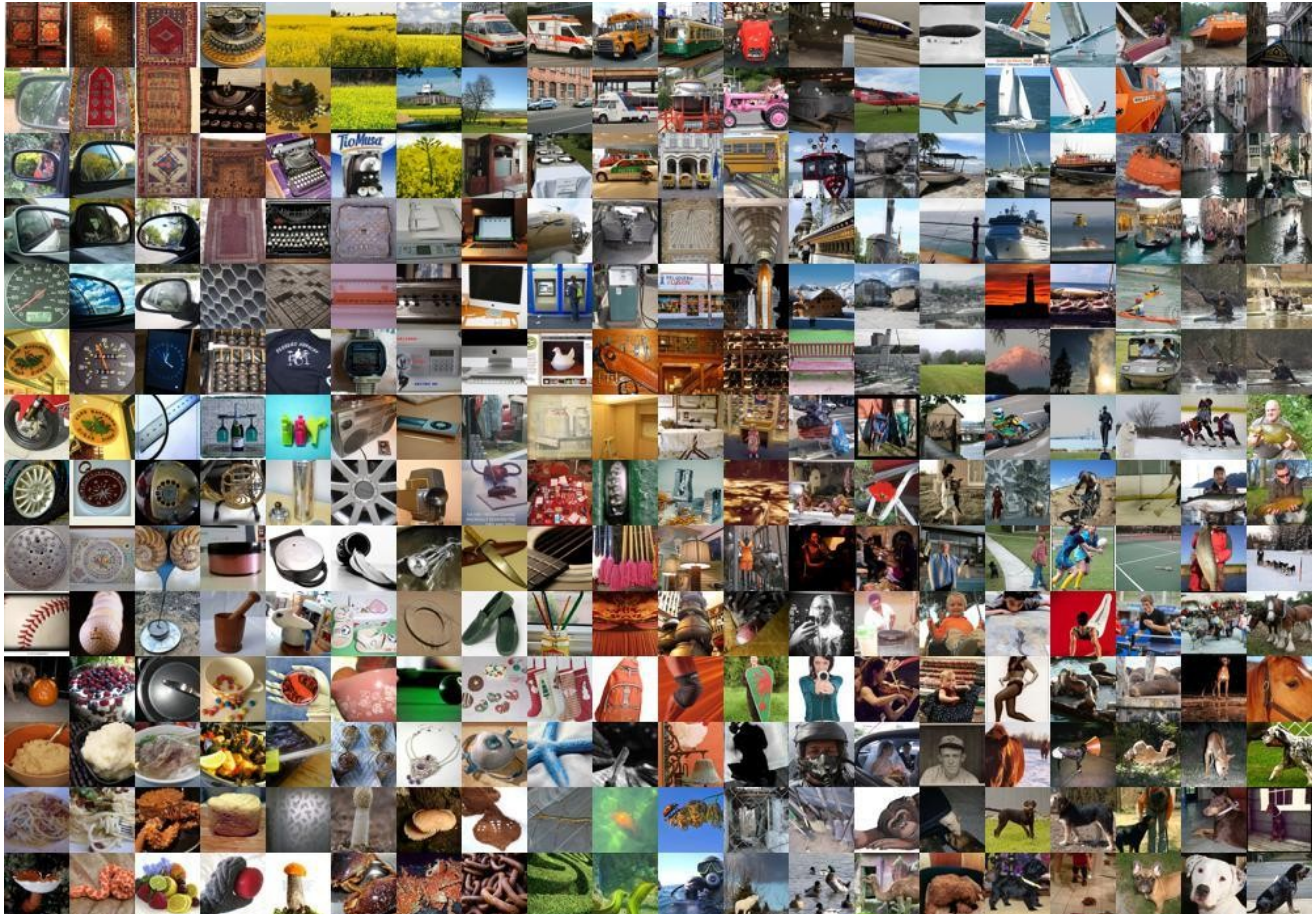


Fig. 5. Scheme of the ANN architecture. Filters are applied to the input data to convolve the information and produce features maps. This operation is done for each of the convolutional layers. Dense layers then combine the extracted features and learn how to label the spectra depending on the provided target. The output layer is composed of one neuron per class giving a score between 0 and 1 independent between each other.

The canonical ImageNet-2012 dataset



ImageNet is a famous dataset as it is large and diverse enough to pre-train large models for a large variety of applications (classification, generation, detection, etc.)

- 1,281,167 training images (150 GB)
- 50,000 validation images
- 100,000 test images,
- Each image is associated with one of 1000 possible classes.

Images are of variable resolution with an average around 500x400.

Practical work:

This dataset is way too heavy for training a model during the course. Use the provided script to run a pre-trained model, identify its architecture and visualize some results. You can try apply it to your own images.

The canonical ImageNet-2012 dataset

Result over ImageNet using a darknet-19 backbone, **91.7 Top5 Accuracy** over the 1000 classes, at a 448p resolution. The network run at 740 ips on an RTX 4090.

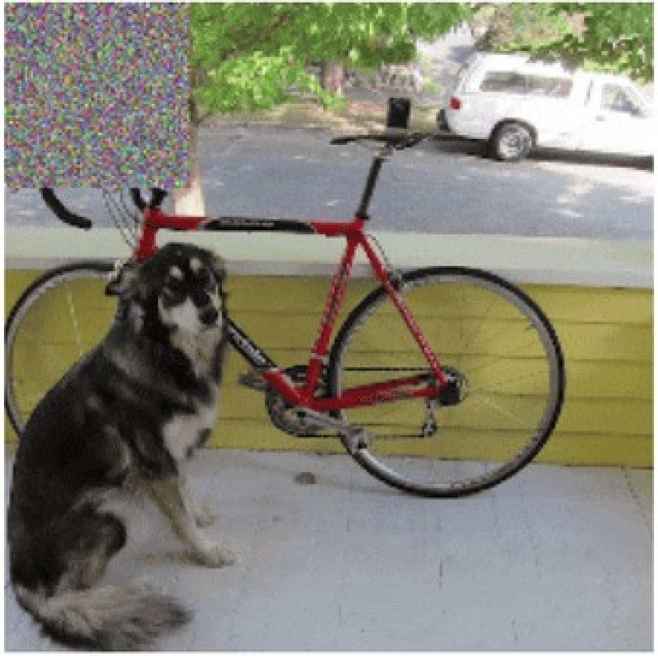
Explaining a model decision with occlusion analysis

An occlusion analysis is done by replacing a portion of the image by noise and measuring the impact it has on the model prediction. An occlusion map can be created by repeating moving the occlusion window over the original image.

Allow to identify image features that contribute positively or negatively to a given prediction.

Here the occlusion map is done with the ImageNet trained model for the « malamute » class.

Occulted input image



Pred difference map