Crushing on DL

Artificial Intelligence and Data Mining Methods in Ecology

University of Tuscia – Viterbo, July 21–25, 2025

Alex Falcon
Beatrice Portelli
University of Udine

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 "Education and Research" - Component 2: "From research to business" - Investment 3.1: "Fund for the realisation of an integrated system of research and innovation infrastructures"

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani

Where We Left Off

ML models are picky eaters (but powerful)

✅ We explored classical ML models: regression, classification, clustering

✅ Saw how these models learn from well-prepared data

✅ Learned to encode and normalize data correctly
→ One-hot, cyclical encoding, Z-score, min-max, etc.

⚠️ Key insight: ML models don't "understand" your data
you have to feed them the right format

Where We're Going Today

Deep Learning and letting the models do the work

- 🧠 From ML to DL: what changes when features are learned automatically
- 🖼️ Vision-focused DL models:
  VGG, ResNet, Vision Transformers
- 💻 Computational costs: data, compute, and time
- 🔁 Transfer learning: reusing models trained on massive datasets
  (Why reinvent the wheel?)

📌 DL models can learn features directly from raw data...
but they come with tradeoffs

By the end of today, we should be able to…

✅ Understand the difference between ML and DL

✅ Recognize key DL architectures in vision (VGG, ResNet, ViT)

✅ Appreciate the computational demands of deep learning

✅ Explain what transfer learning is, and how it helps

✅ Know when DL is worth it… and when it might be overkill!

Day 1

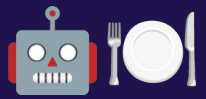
Why AI for Environment?

Day 2

Best Practices

Day 3

ML & Preprocessing



Day 4

Deep Learning

The Supervised ML Pipeline

Feature Extraction

The model is trained to learn a function mapping input features to the target

ML Model

N

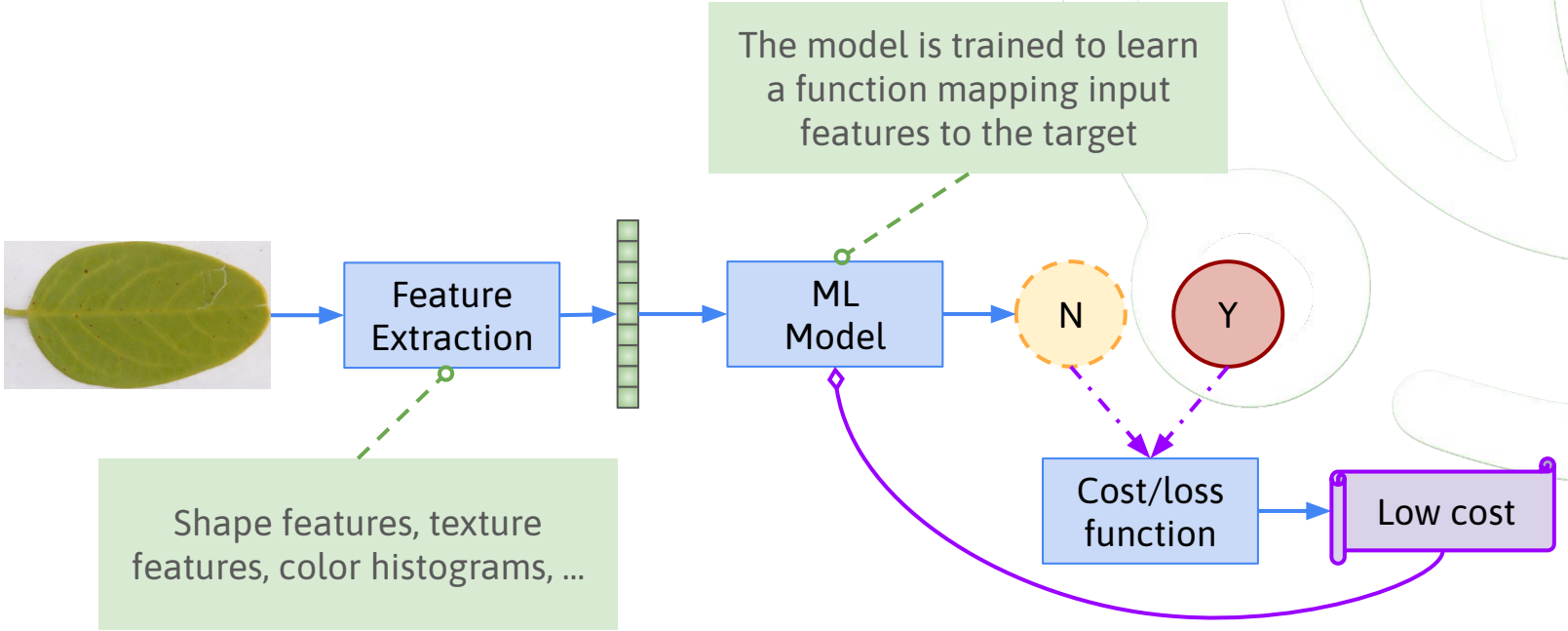
Y

Cost/loss function

Low cost

Shape features, texture features, color histograms, ...

The Supervised ML Pipeline



We said feature engineering is difficult, requires expertise, time, ...

The Supervised ML Pipeline

Feature Extraction

Shape features, texture features, color histograms, ...

The model is trained to learn a function mapping input features to the target

ML Model

N

Y

Cost/loss function

Low cost

We said feature engineering is difficult, requires expertise, time, ...

We could try to make the feature extraction process ***automatic***!

The Supervised ~~ML~~ DL Pipeline

The model is trained to learn a function mapping input features to the target

This part often is called "problem solving" part of the DL pipeline

Feature Extraction

ML Model

N

Y

Cost/loss function

Low cost

Now, the supervision also updates the feature extraction part!

AI is an umbrella term

From 2 days ago

Machine Learning - ML

- statistical models and algorithms
- **hand-crafted** features
- needs experts
- relies on **clean** data
- can work on **small** datasets
- **white box** models

Deep Learning - DL

- complex relationships and patterns
- **automated** feature extraction
- less expert intervention
- can deal with **noisy** data
- needs **large** datasets
- **black box** models

AI is an umbrella term

From 2 days ago

Machine Learning - ML

- statistical models and algorithms
- **hand-crafted** features
- needs experts
- relies on **clean** data
- can work on **small** datasets
- **white box** models

Deep Learning - DL

- complex relationships and patterns
- **automated** feature extraction
- less expert intervention
- can deal with **noisy** data
- needs **large** datasets
- **black box** models

Statistical features from tabular;
Shape/texture/... from images;
...

AI is an umbrella term

From 2 days ago

Machine Learning - ML

- statistical models and algorithms
- **hand-crafted** features
- needs experts
- relies on **clean** data
- can work on **small** datasets
- **white box** models

Deep Learning - DL

- complex relationships and patterns
- **automated** feature extraction
- less expert intervention
- can deal with **noisy** data
- needs **large** datasets
- **black box** models

Statistical features from tabular;
Shape/texture/... from images;
...

Key novelty in DL

AI is an umbrella term

From 2 days ago

Machine Learning - ML

- statistical models and algorithms
- **hand-crafted** features
- needs experts
- relies on **clean** data
- can work on **small** datasets
- **white box** models

Deep Learning - DL

- complex relationships and patterns
- **automated** feature extraction
- less expert intervention
- can deal with **noisy** data
- needs **large** datasets
- **black box** models

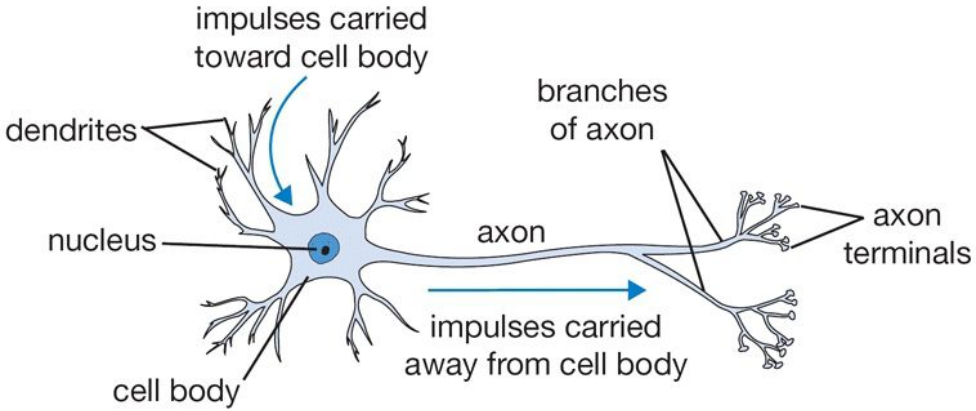
Statistical features from tabular;
Shape/texture/... from images;
...

Key novelty in DL

What does this mean?

Quick detour on DL

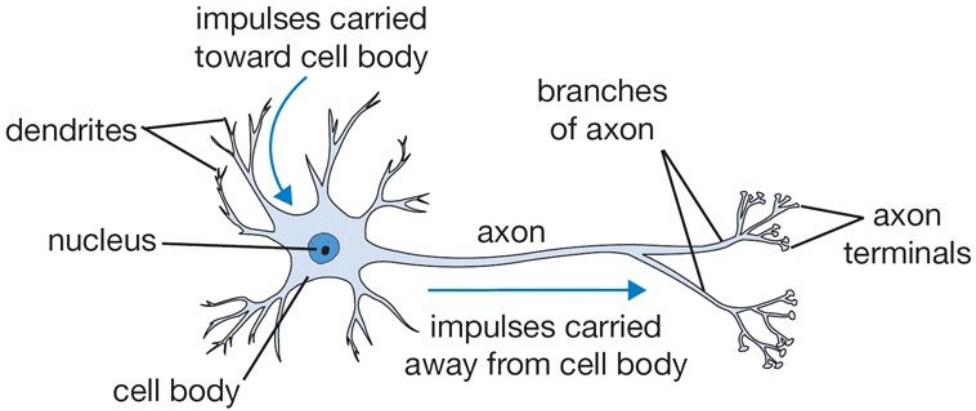
- DL → Artificial Neural Networks
- Neural → built upon "**neurons**"
  - Inspiration from our brains

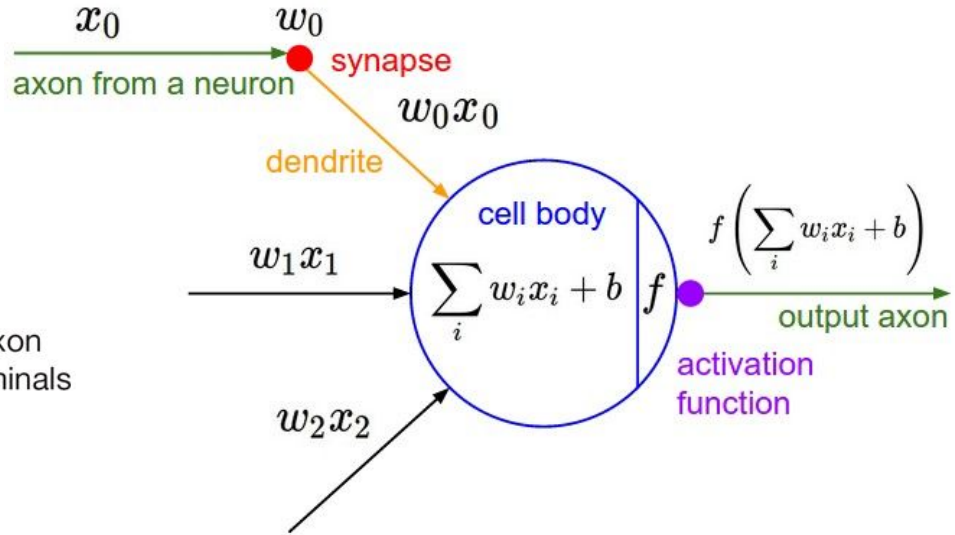


https://cs231n.github.io/neural-networks-1/

Quick detour on DL

- DL → Artificial Neural Networks
- Neural → built upon "**neurons**"
  - Inspiration from our brains

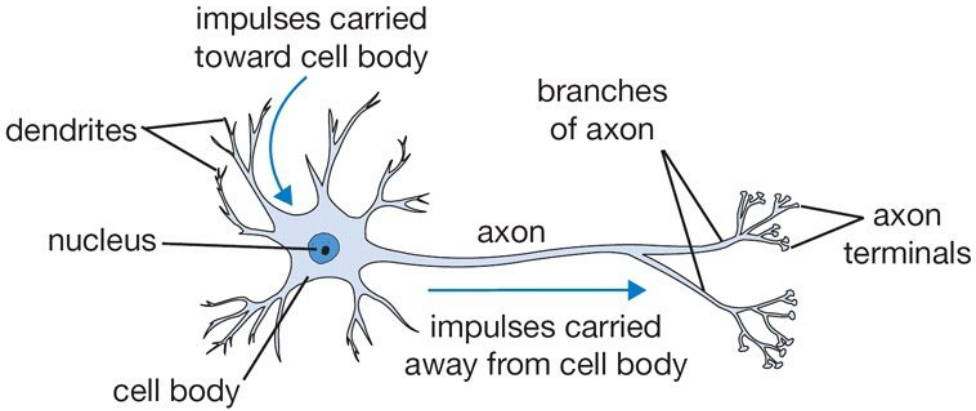


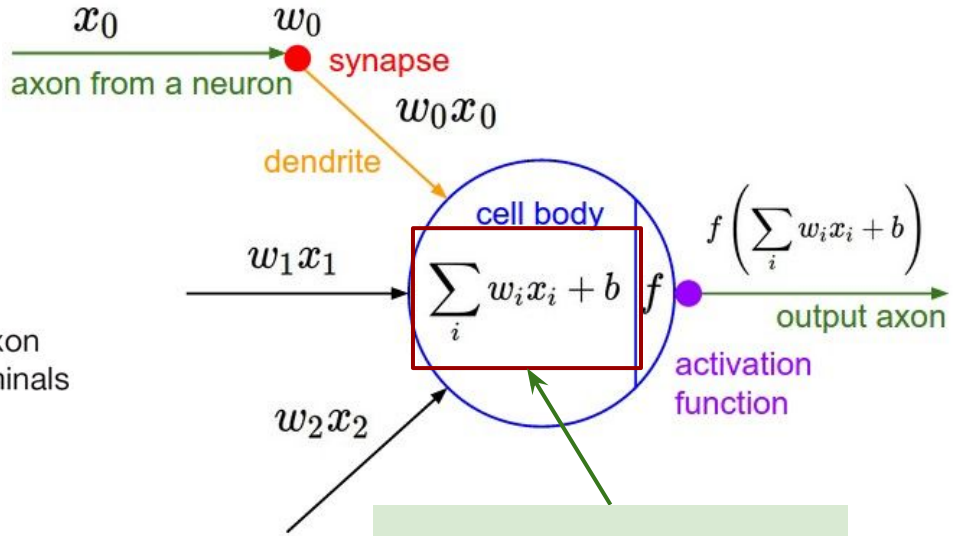


https://cs231n.github.io/neural-networks-1/

Quick detour on DL

- DL → Artificial Neural Networks
- Neural → built upon "**neurons**"
  - Inspiration from our brains





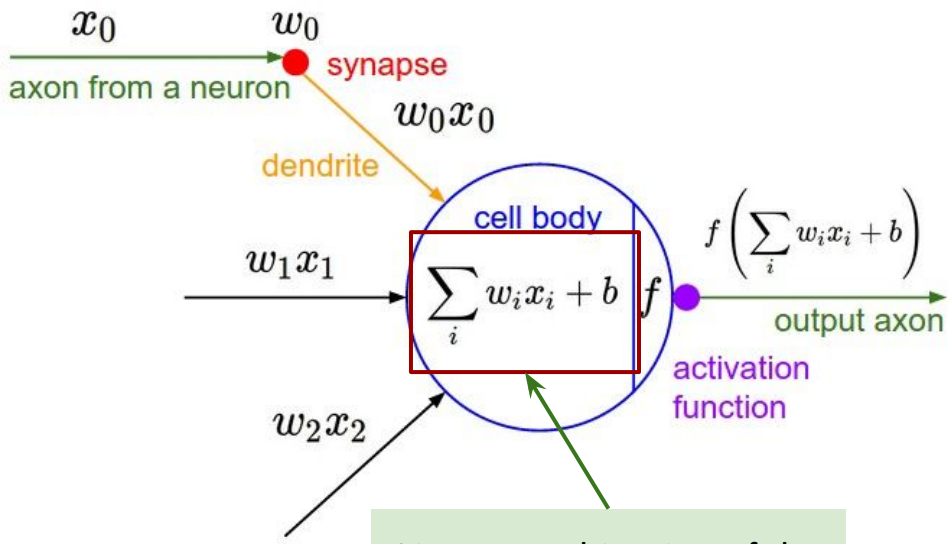
Linear combination of the input features ***x***

https://cs231n.github.io/neural-networks-1/

Quick detour on DL

- DL → Artificial Neural Networks
- Neural → built upon "**neurons**"
  - Inspiration from our brains

So: input $\boldsymbol{x} = (x_1, x_2, ..., x_N)$ and we compute the output with parameters/*weights* $\boldsymbol{w} = (w_1, ..., w_N)$



Linear combination of the input features $\boldsymbol{x}$

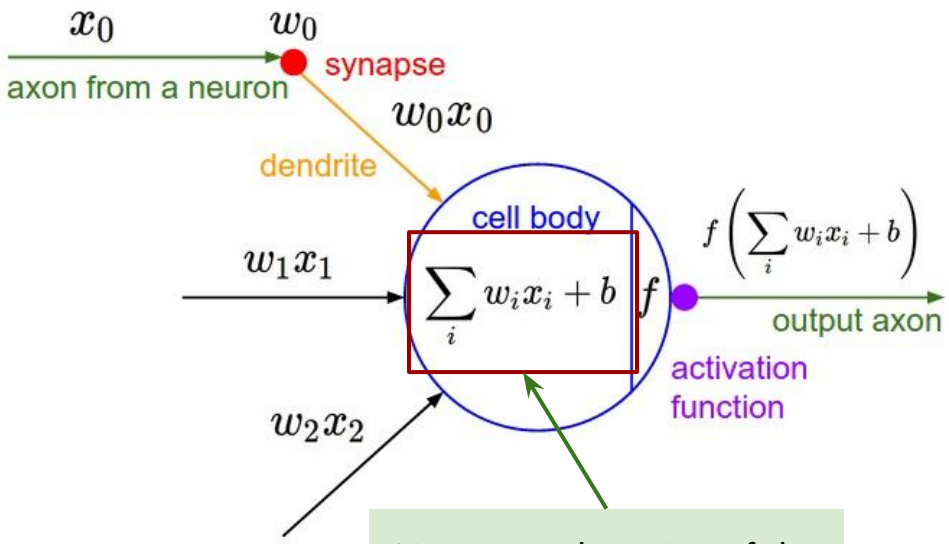
https://cs231n.github.io/neural-networks-1/

Quick detour on DL

- DL → Artificial Neural Networks
- Neural → built upon "**neurons**"
  - Inspiration from our brains

So: input $\boldsymbol{x} = (x_1, x_2, ..., x_N)$ and we compute the output with parameters/*weights* $\boldsymbol{w} = (w_1, ..., w_N)$

<u>*Note*</u>: if $x_1, x_2, ..., x_N$ are *features* → through ML we are learning $\boldsymbol{w}$ (e.g. this neuron looks very similar to a linear classifier/regressor)



https://cs231n.github.io/neural-networks-1/

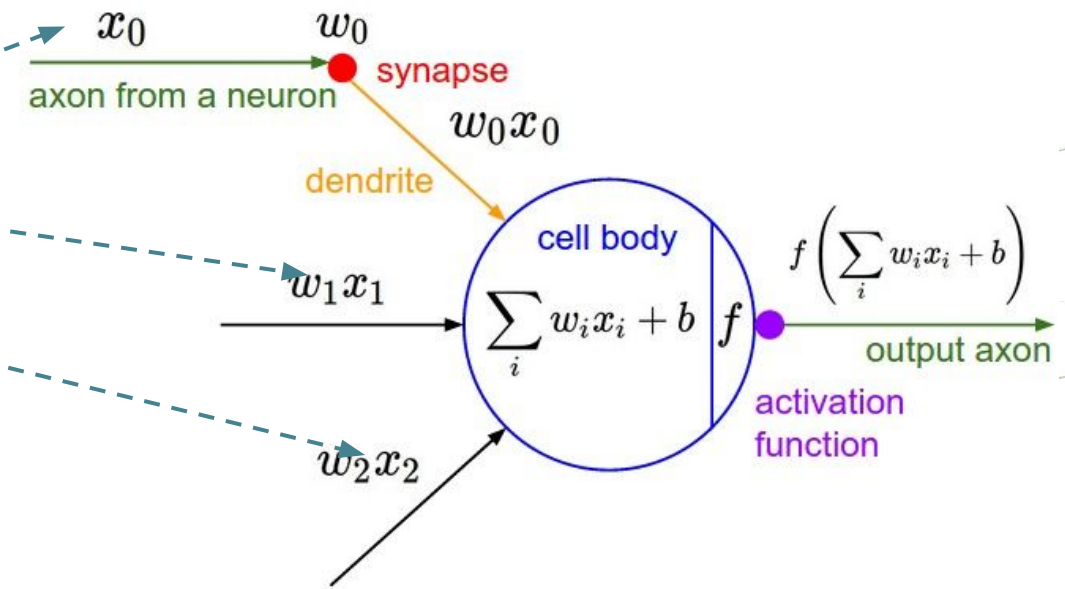
Quick detour on DL

eccentricity

circularity

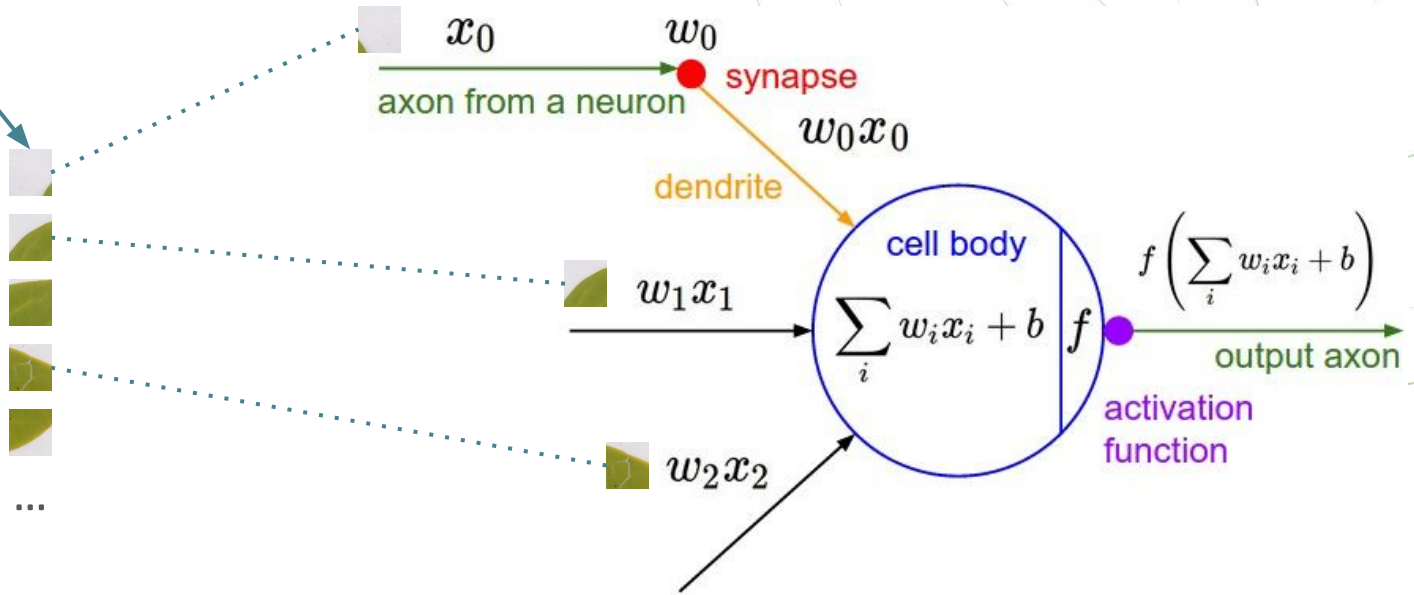
GLCM avg

...



What if we put raw data as input features?

Imagine these as pixels

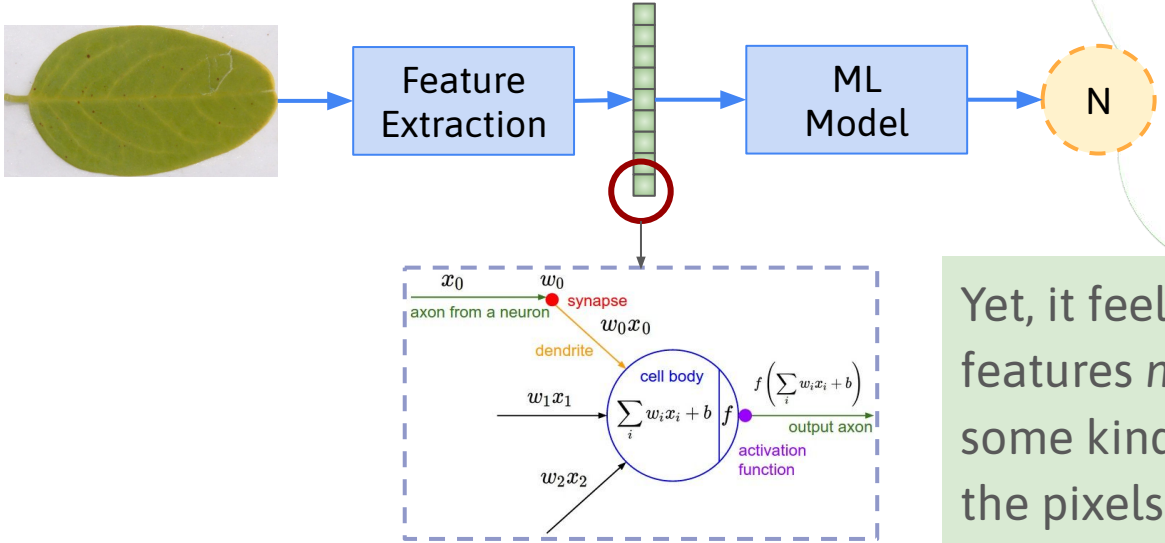


The output would become a linear combination of pixels.

Not as “powerful” as combining image features from two days ago.

A look back at the pipeline

Indeed, image features look pretty complex compared to linearly combining pixels.

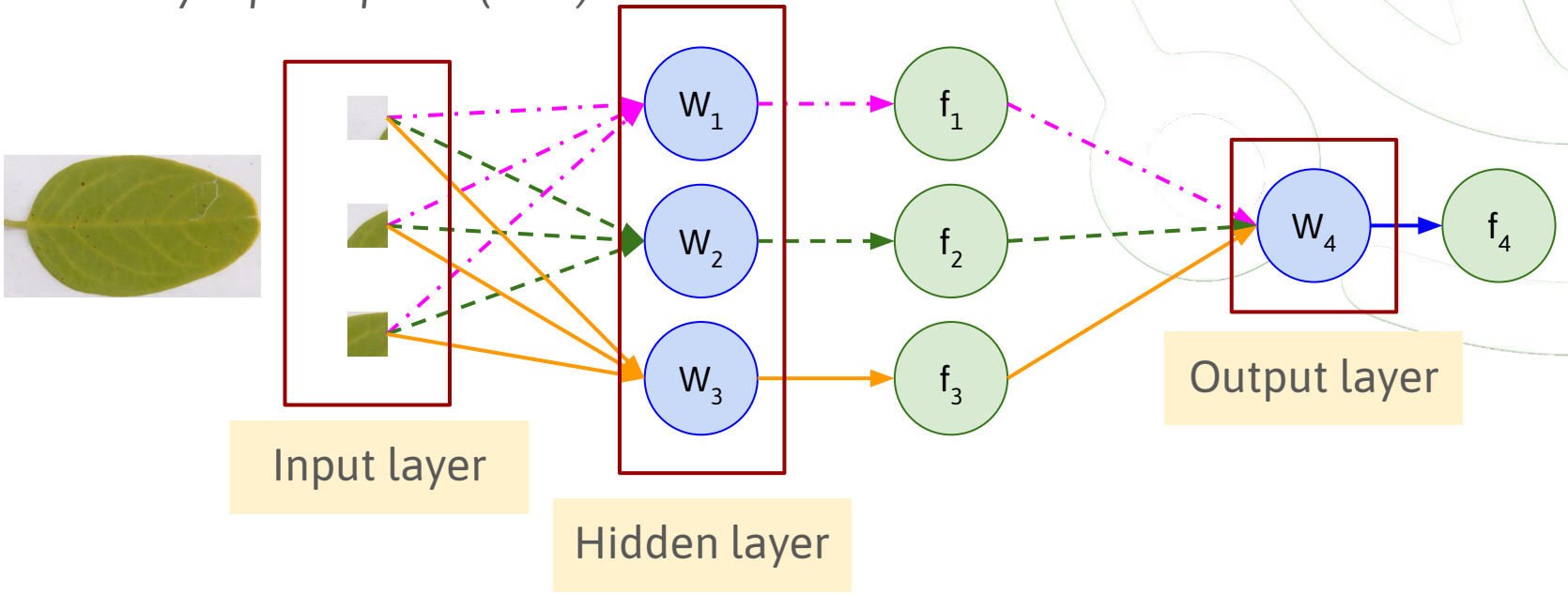


Yet, it feels like each of those features *might* be extracted as some kind of weighted mean of the pixels.

What if we tried to "recreate" some kind of feature extraction process by stacking multiple neurons?

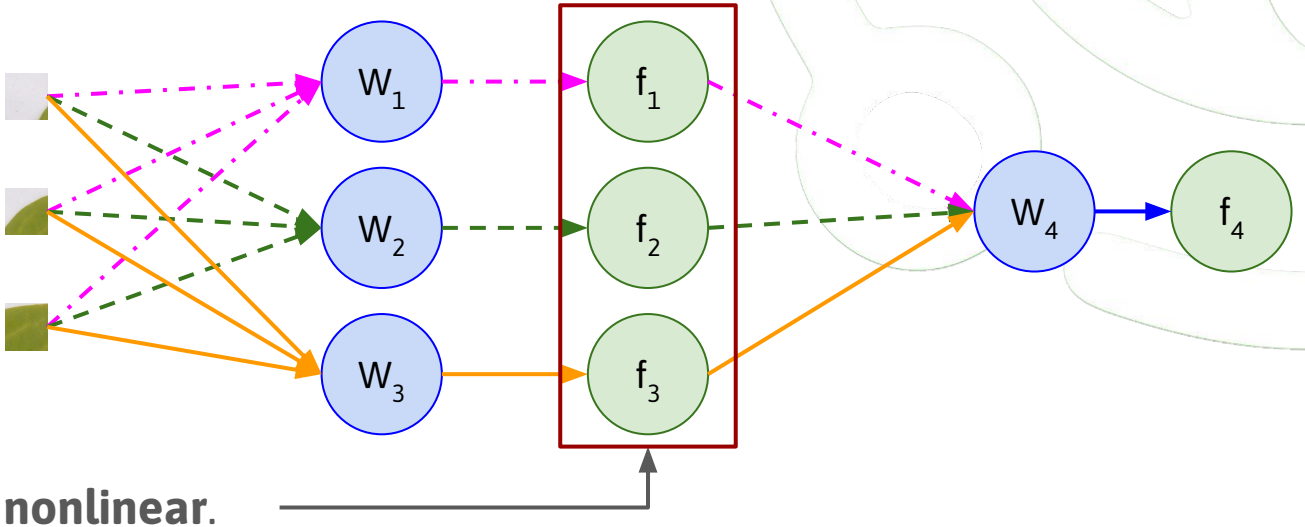
Let’s stack multiple of these “neurons”!

Multi-layer perceptron (MLP)



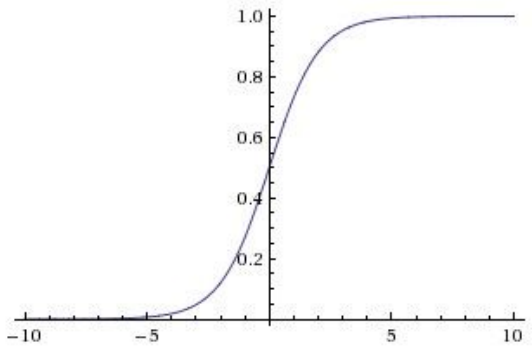
Let's stack multiple of these "neurons"!

Multi-layer perceptron (MLP)

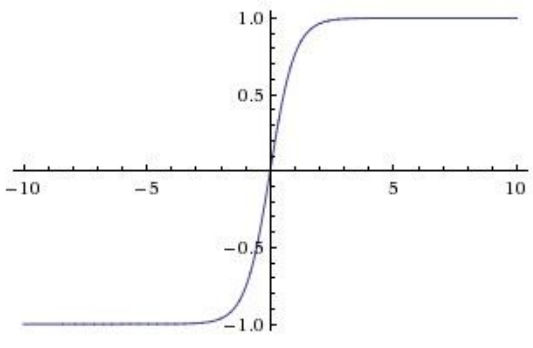


Note: f1, f2, … are **nonlinear**.
Linear → $W_4(W_1 \boldsymbol{x})$ is a linear transform of ***x*** (i.e. a single W would suffice)
Nonlinear → the model can learn "any" function

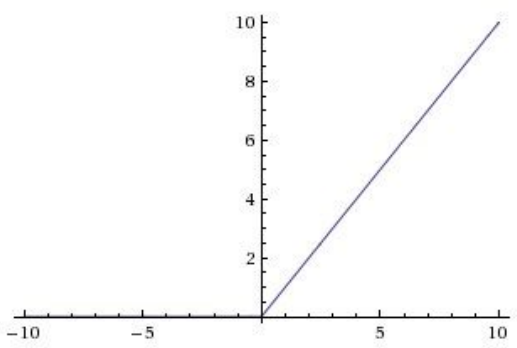
Nonlinear functions?



Sigmoid



Tanh



ReLU

Mathematical functions that transform a single input number.

Sigmoid: intuitively, for binary problems; some suggest to avoid it (not zero-centered).
Tanh: preferred.
ReLU: most common choice.

Is the MLP effective for images?

Short answer: **no**.

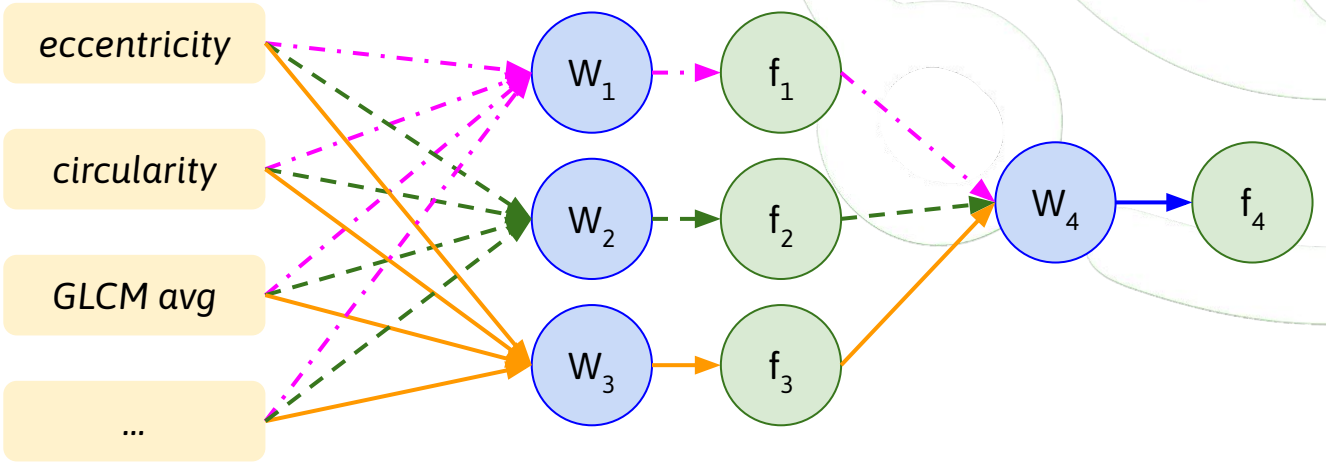
Longer answer: *definitely* **no**.

Main issue:
input = pixels... how many pixels do we have?

- Small/low res image 256x256 → 65.536 (input features)
  - RGB image? 65.536*3 ...
- → each neuron's W has **65.536 parameters** to learn
  - It will very likely learn the *noise* in the image

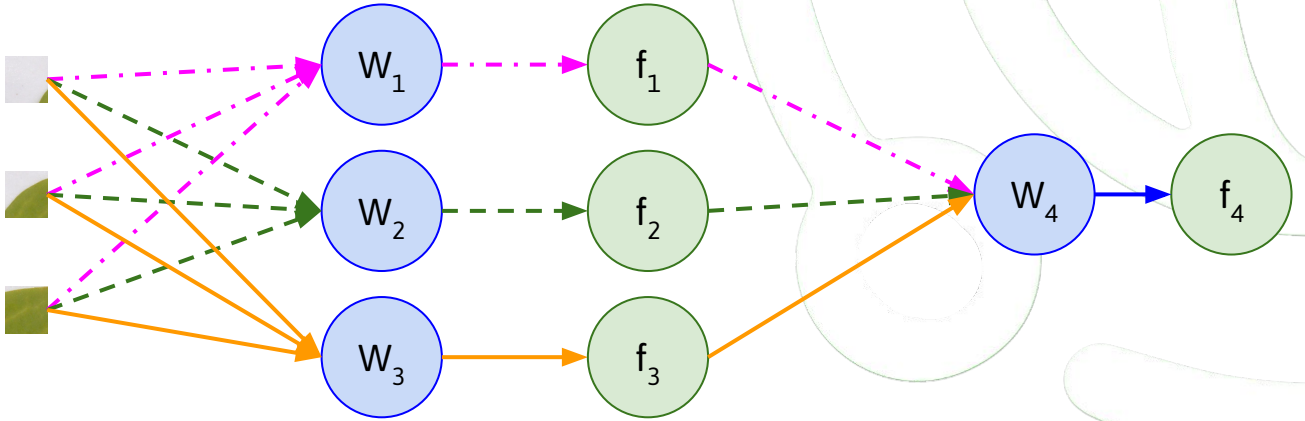
Is the MLP effective for images?

Yet, it fits nicely on tabular-like data.

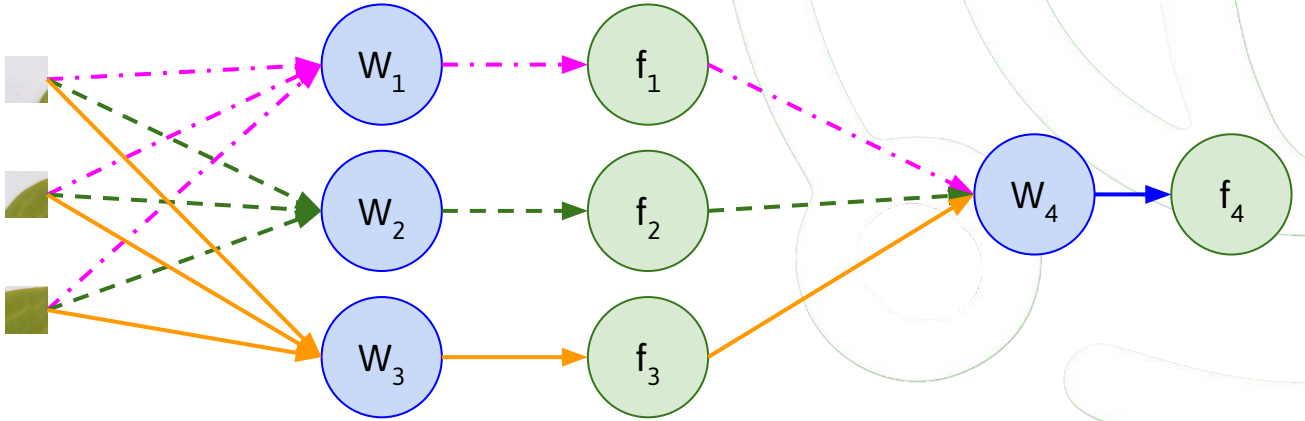


But, this is again “standard” ML.

A look back at the MLP on pixels

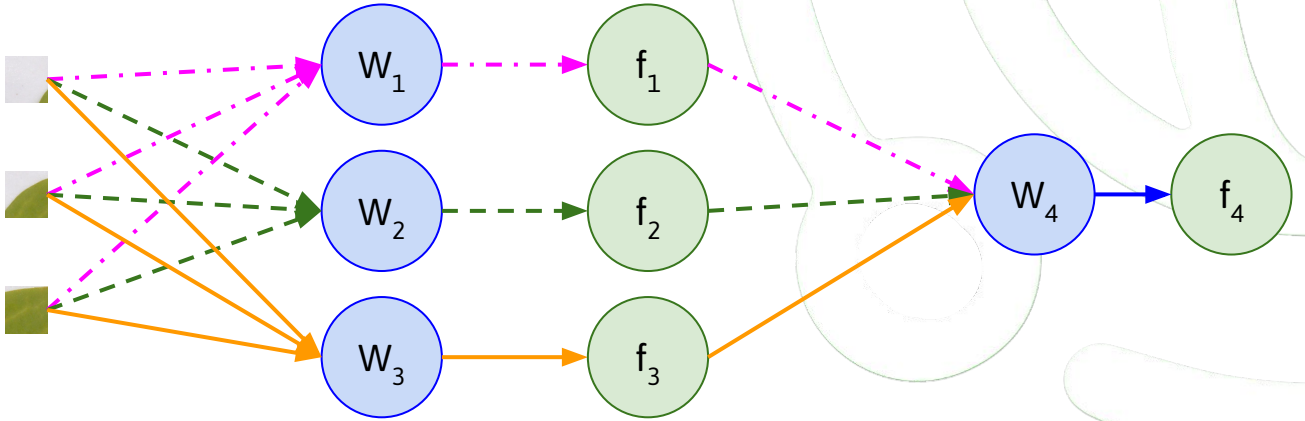


A look back at the MLP on pixels



What is each of these neurons receiving as input?

A look back at the MLP on pixels



What is each of these neurons receiving as input?

Actually... these are *not* pixels, but *blocks of 32x32 pixels*.

The lack of spatiality

Key insight: the MLP has no direct access to spatial information, yet images make sense because pixels are arranged in some clear way, along both dimensions.

The lack of spatiality

Key insight: the MLP has no direct access to spatial information, yet images make sense because pixels are arranged in some clear way, along both dimensions.




E.g. these small regions make (some) sense to us.

The lack of spatiality

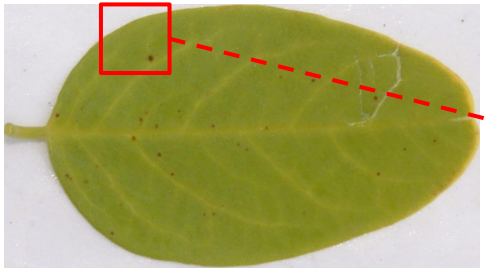



E.g. these small regions make (some) sense to us.

So... what if we had a “neuron” that looks at small regions of pixels?

Towards *Convolutional* neural networks (CNN)

CNNs are networks that use convolutions, i.e. small “filters” that sequentially scan the image and produce an output.



Imagine this is a 3x3 region

$p_{1,1}$	$p_{1,2}$	$p_{1,3}$
$p_{2,1}$	$p_{2,2}$	$p_{2,3}$
$p_{3,1}$	$p_{3,2}$	$p_{3,3}$

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$
$w_{3,1}$	$w_{3,2}$	$w_{3,3}$

Convolutional filter/kernel of size 3x3

First, element-wise multiplication

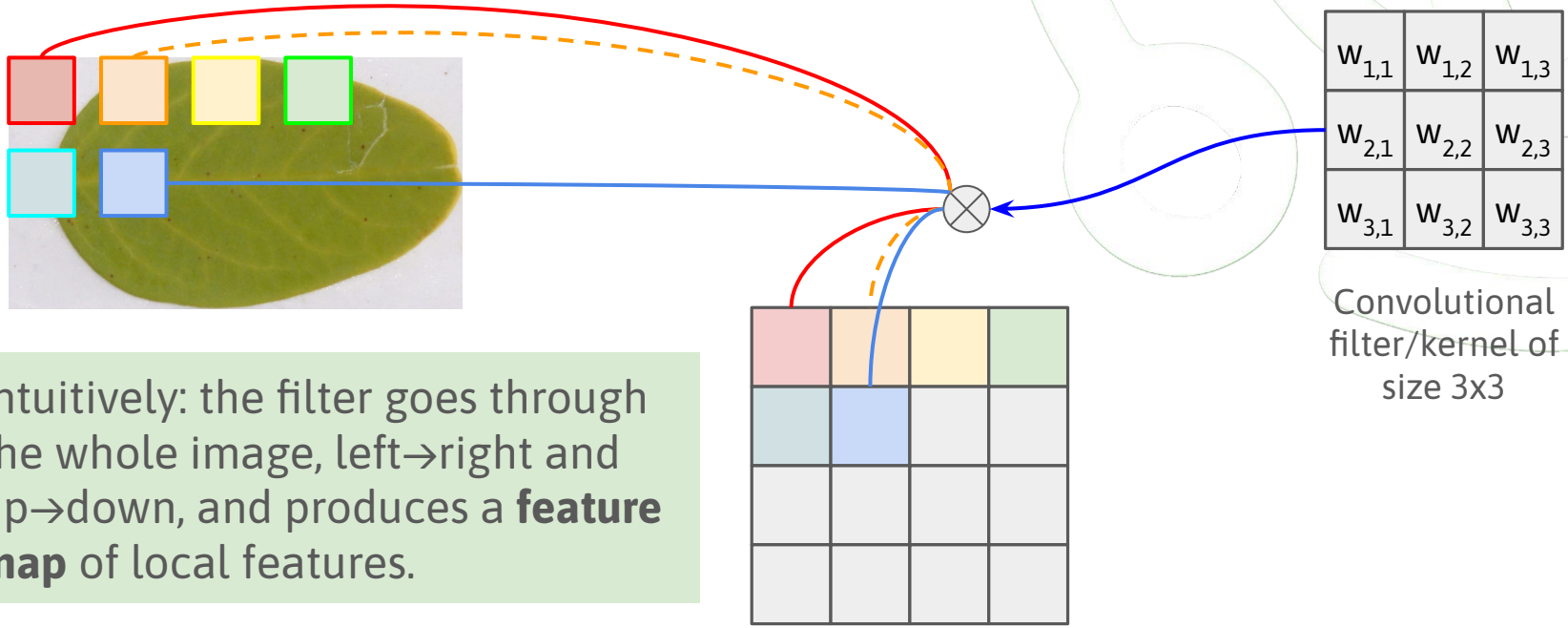
$w_{1,1}p_{1,1}$	$w_{1,2}p_{1,2}$	$w_{1,3}p_{1,3}$
$w_{2,1}p_{2,1}$	$w_{2,2}p_{2,2}$	$w_{2,3}p_{2,3}$
$w_{3,1}p_{3,1}$	$w_{3,2}p_{3,2}$	$w_{3,3}p_{3,3}$

Then, output the sum:

$w_{1,1}p_{1,1} + w_{1,2}p_{1,2} + \ldots$

Towards *Convolutional* neural networks (CNN)

How is the output produced by the filter used?

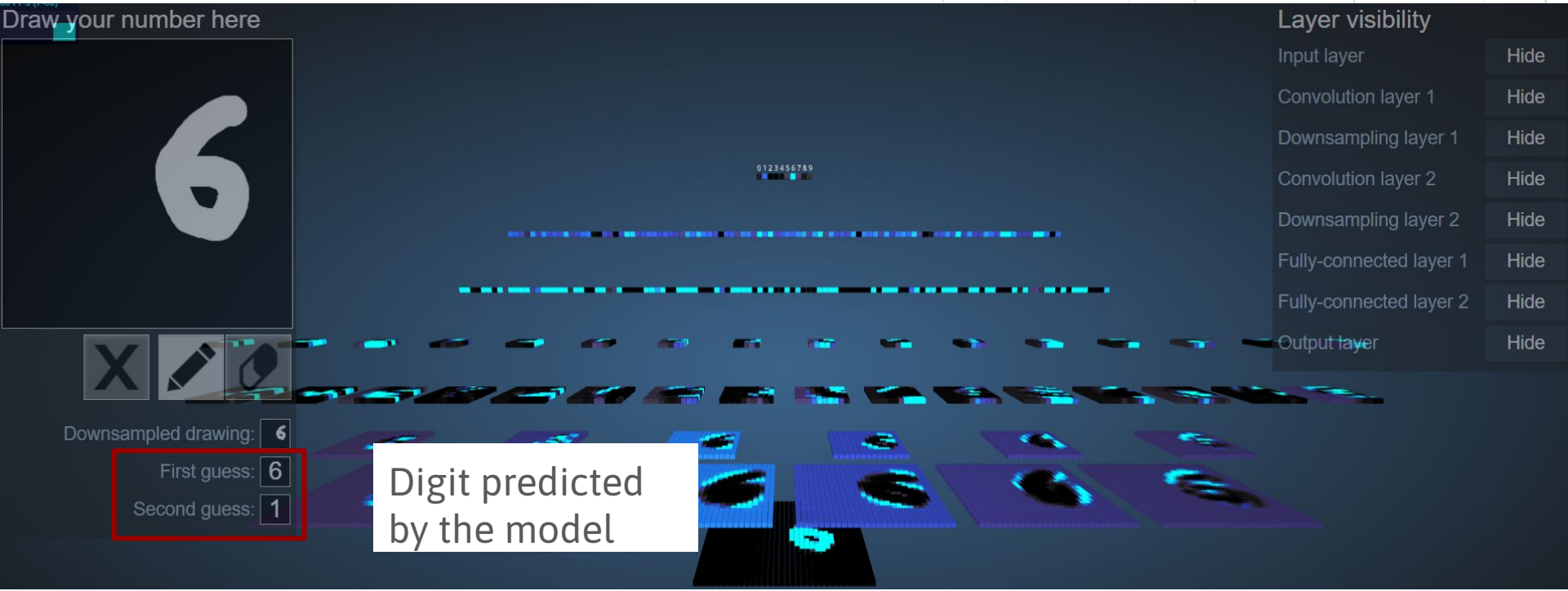


Intuitively: the filter goes through the whole image, left→right and up→down, and produces a **feature map** of local features.

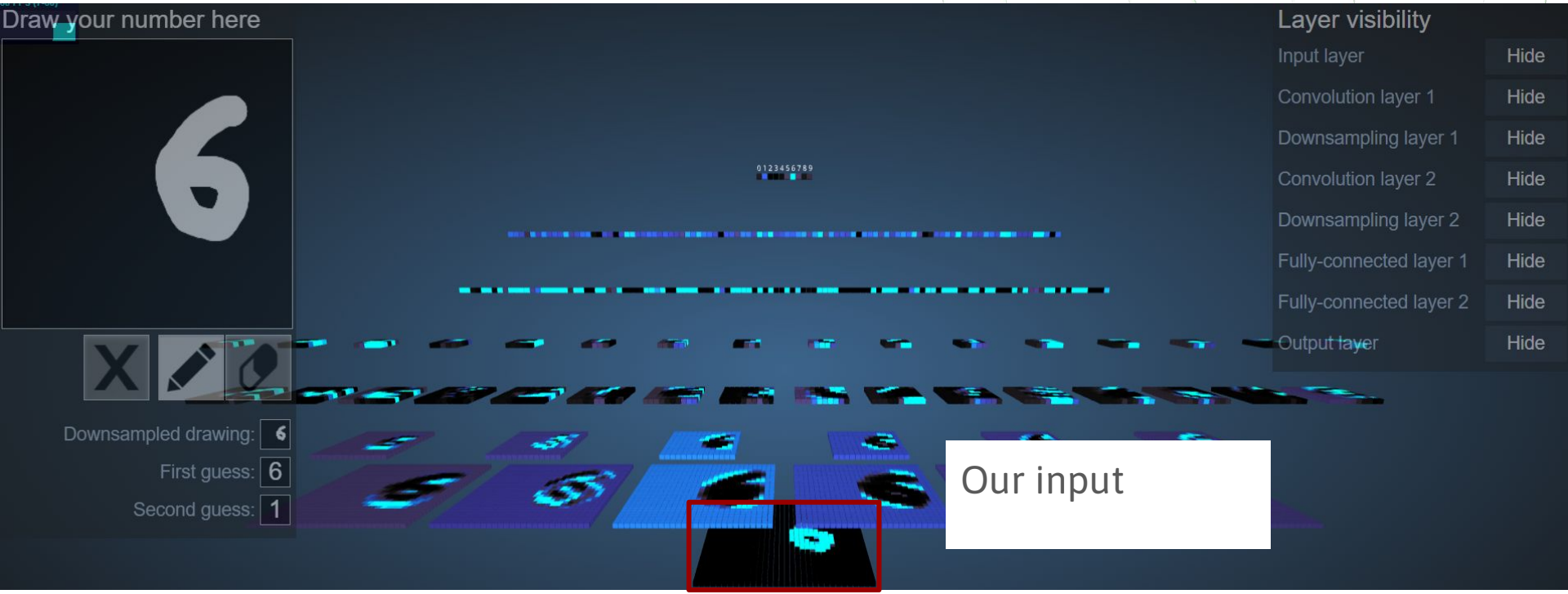
Let’s see a live example!

https://adamharley.com/nn_vis/cnn/3d.html

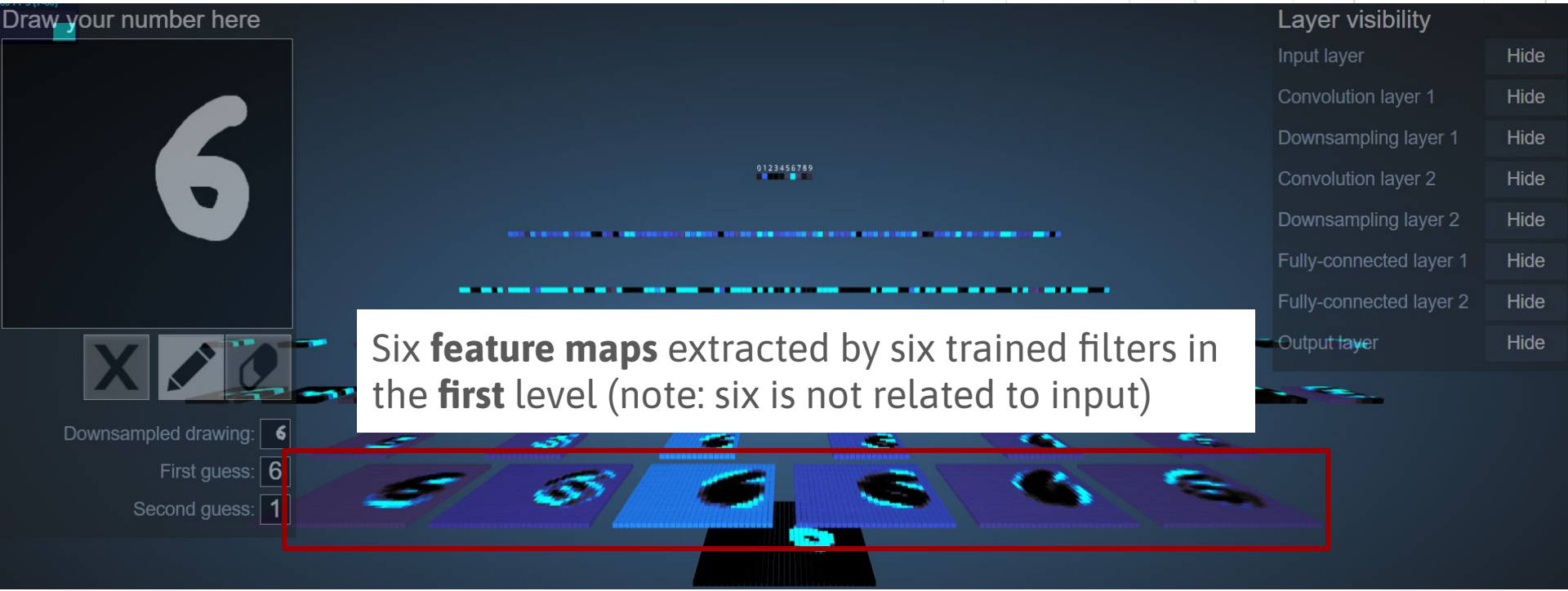
Let’s see a live example!



Let’s see a live example!

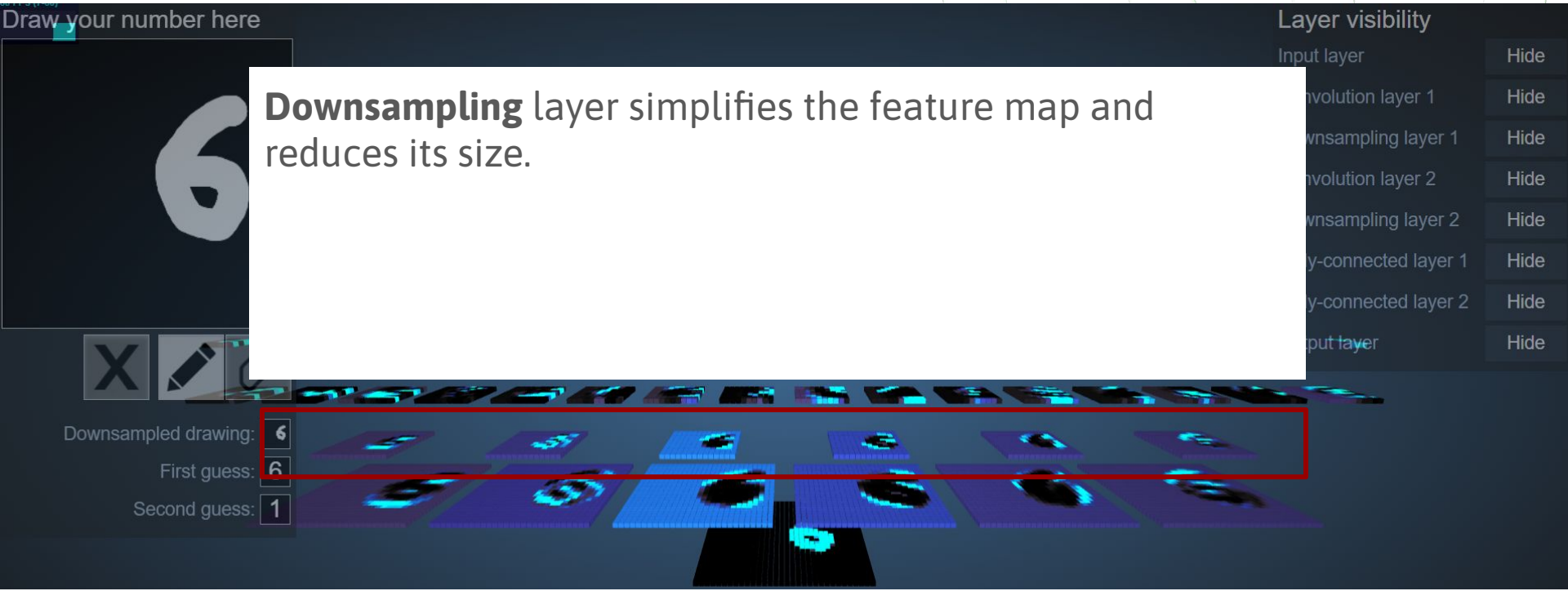


Let’s see a live example!



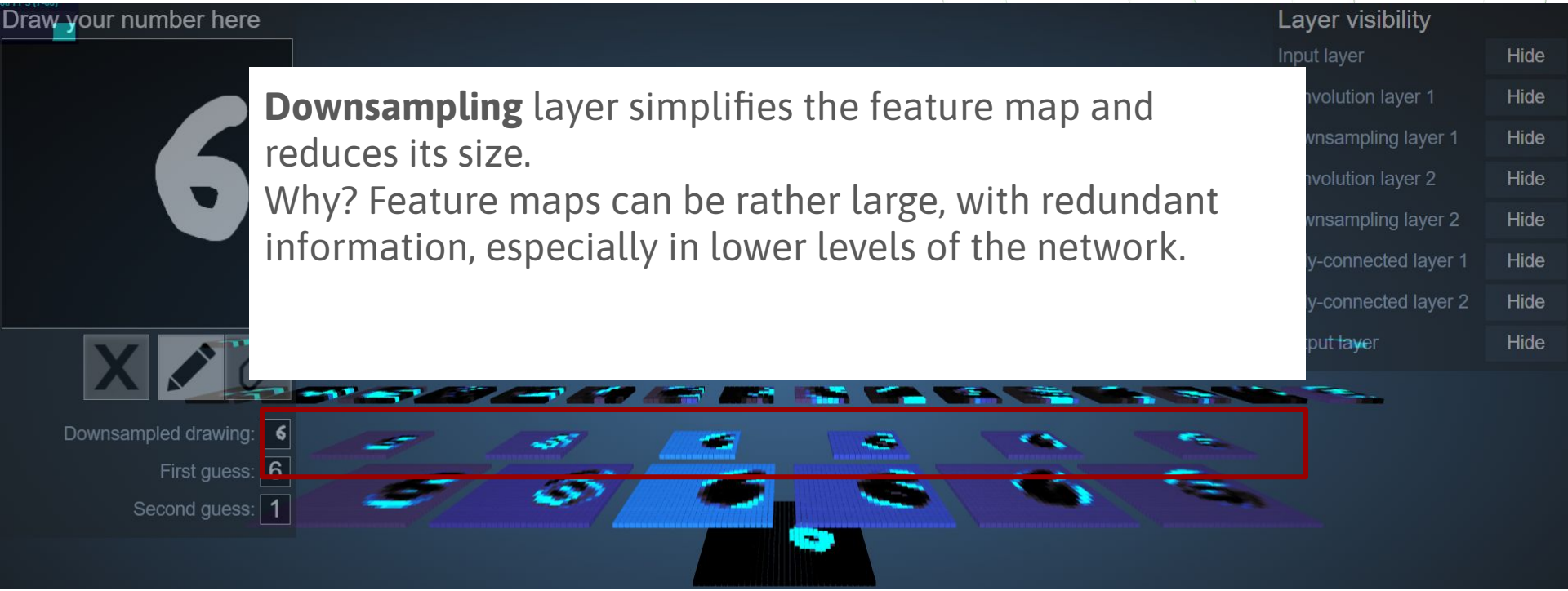
Six **feature maps** extracted by six trained filters in the **first** level (note: six is not related to input)

Let’s see a live example!



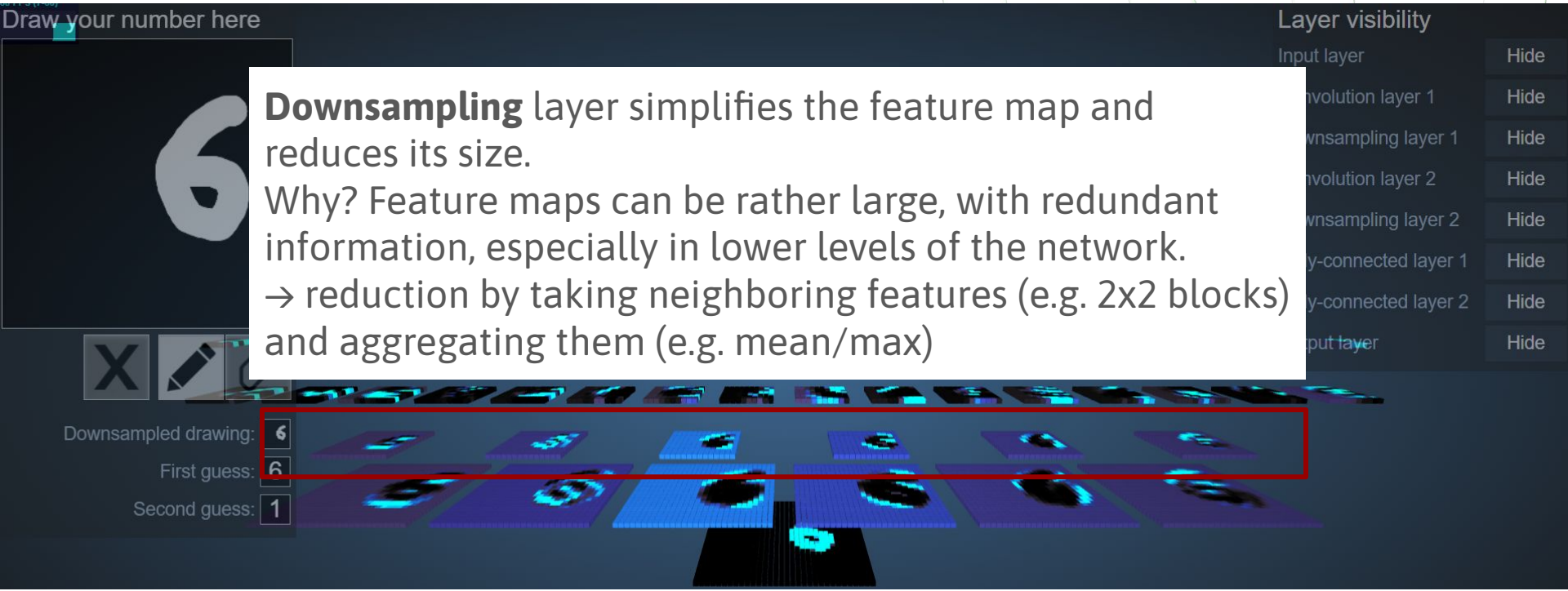
Downsampling layer simplifies the feature map and reduces its size.

Let’s see a live example!



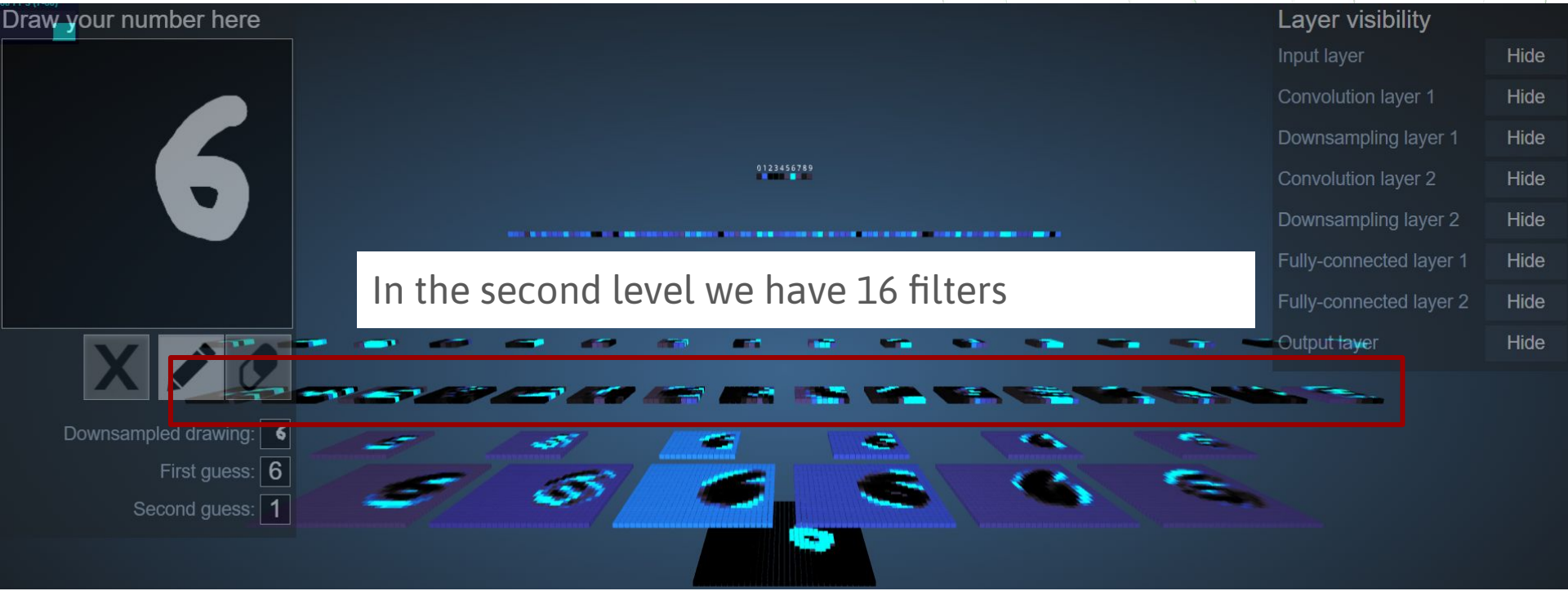
Downsampling layer simplifies the feature map and reduces its size.
Why? Feature maps can be rather large, with redundant information, especially in lower levels of the network.

Let’s see a live example!

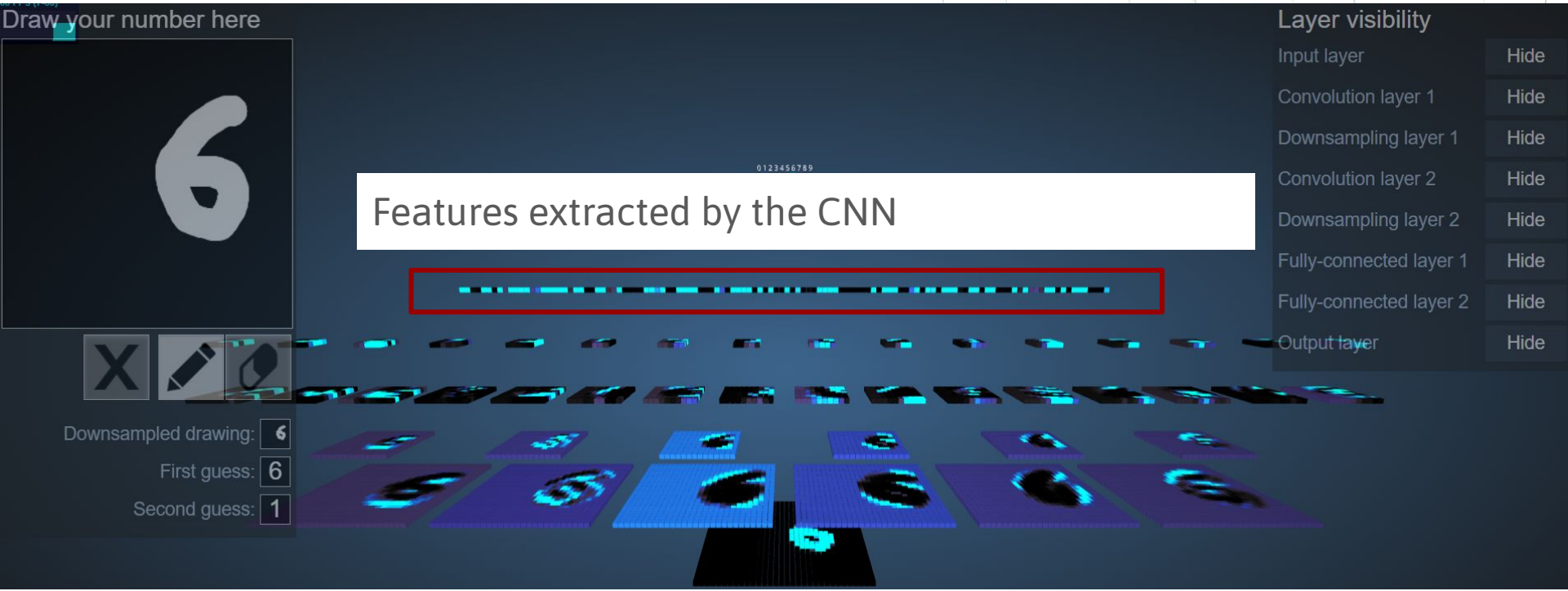


Downsampling layer simplifies the feature map and reduces its size.
Why? Feature maps can be rather large, with redundant information, especially in lower levels of the network.
→ reduction by taking neighboring features (e.g. 2x2 blocks) and aggregating them (e.g. mean/max)

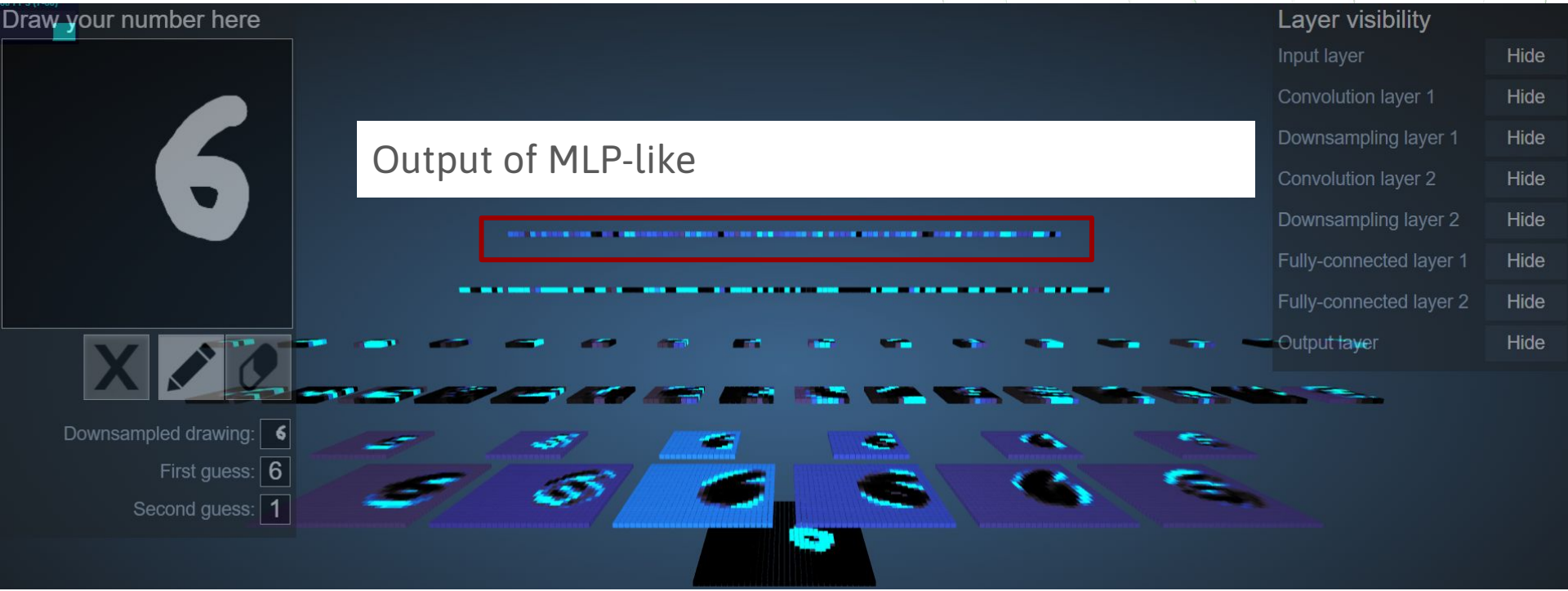
Let’s see a live example!



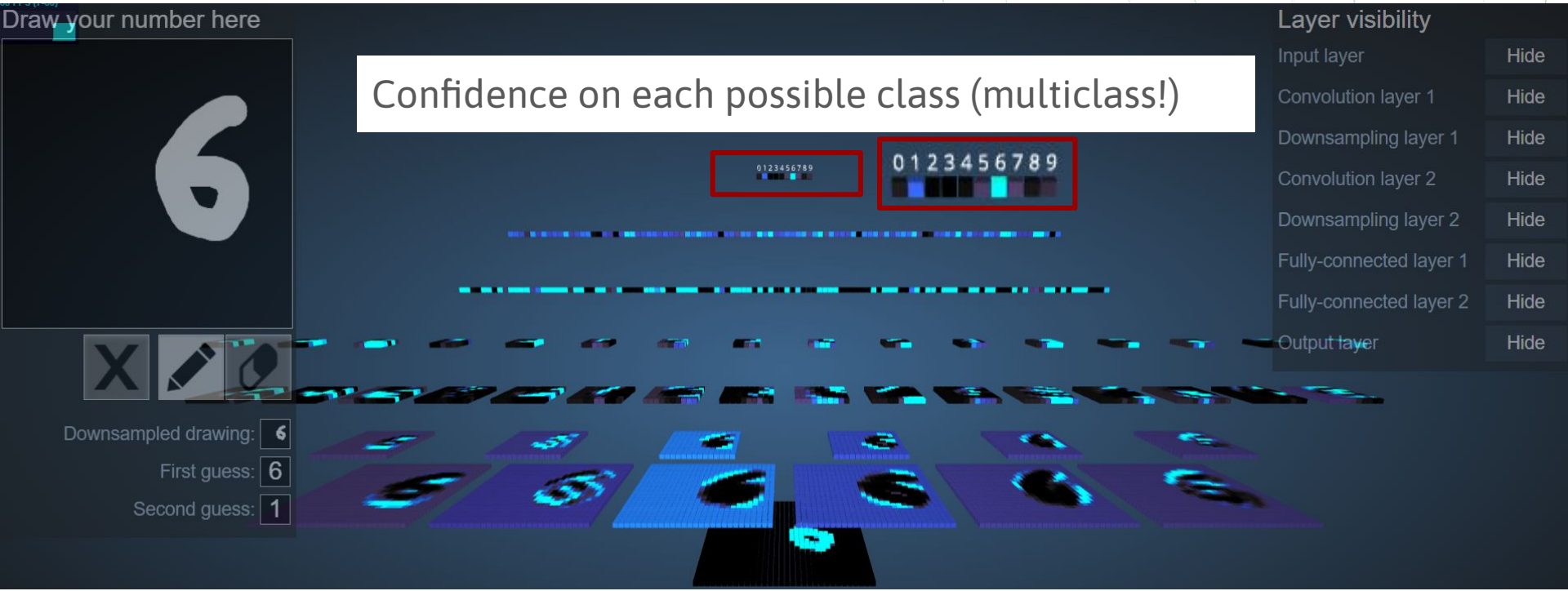
Let’s see a live example!



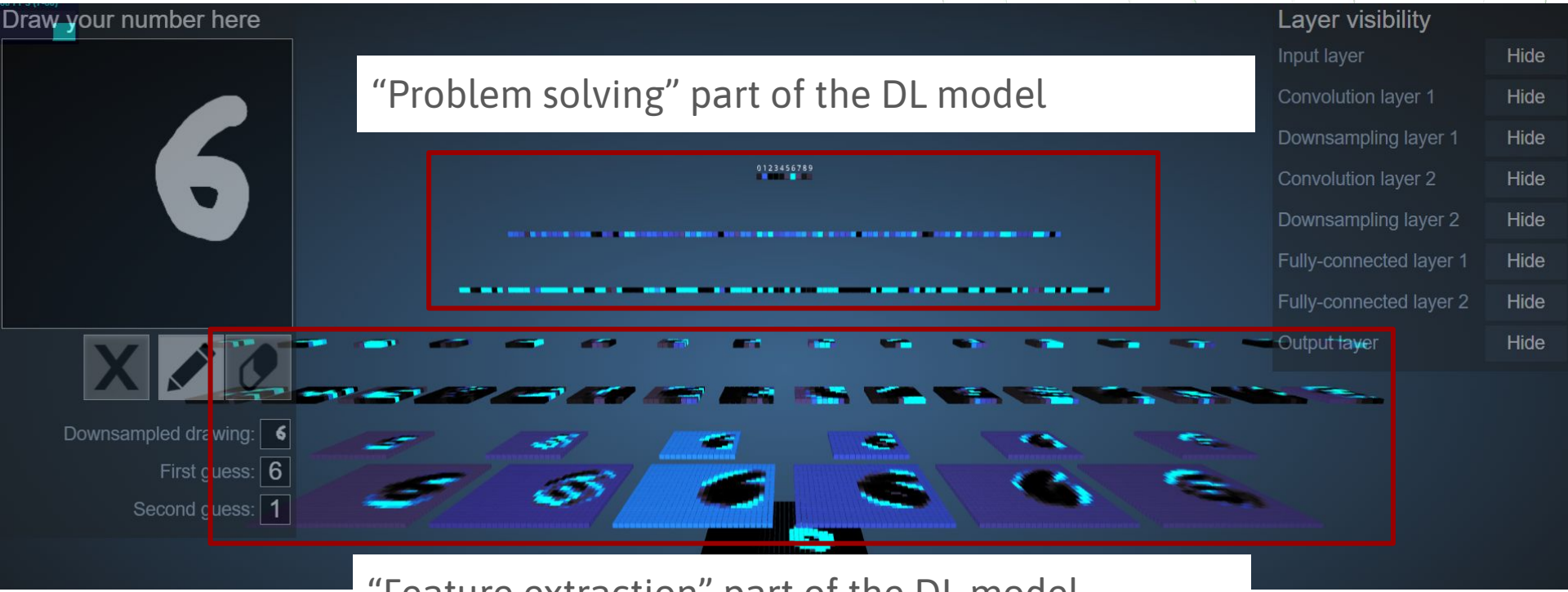
Let’s see a live example!



Let’s see a live example!



Let’s see a live example!



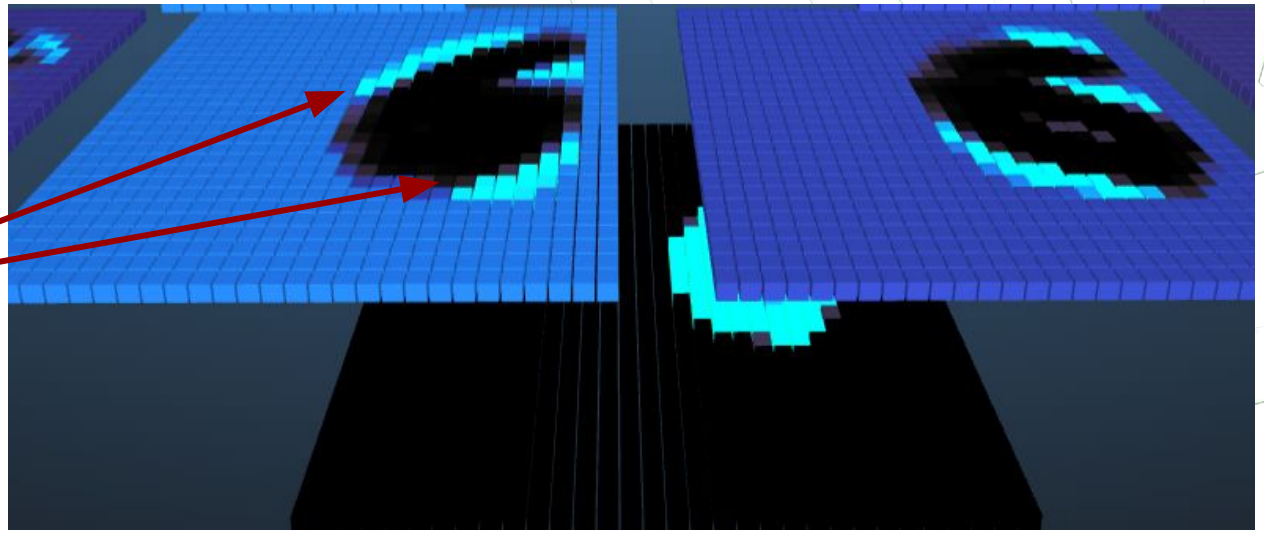
“Problem solving” part of the DL model

“Feature extraction” part of the DL model

Let’s see a live example!

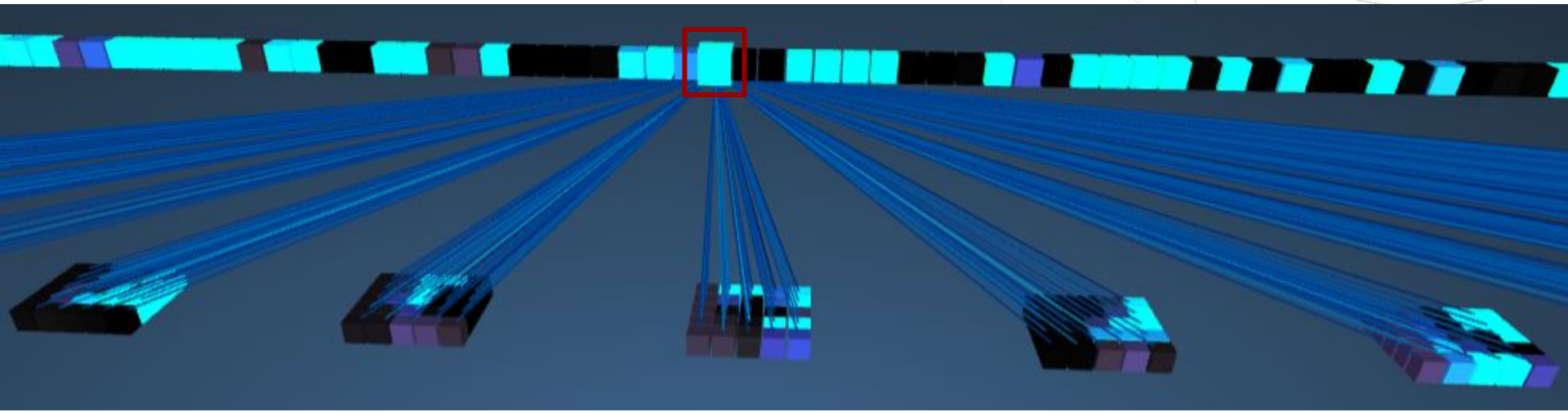
White-ish blue = high importance to that part of the image, i.e.: this filter is searching for borders in ***that*** direction (45° or something like that)!

Hopefully this reminds you of Gabor filters!



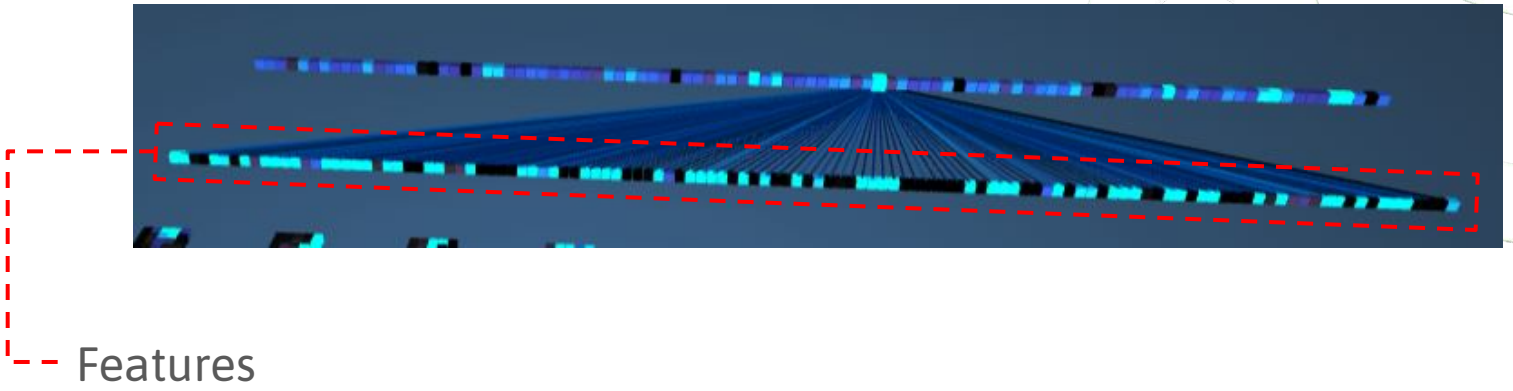
Let’s see a live example!

This is a **neuron** receiving all those spatial features, and attributing each a different amount of importance (light- or dark-blueish colors) in the edges/arrows

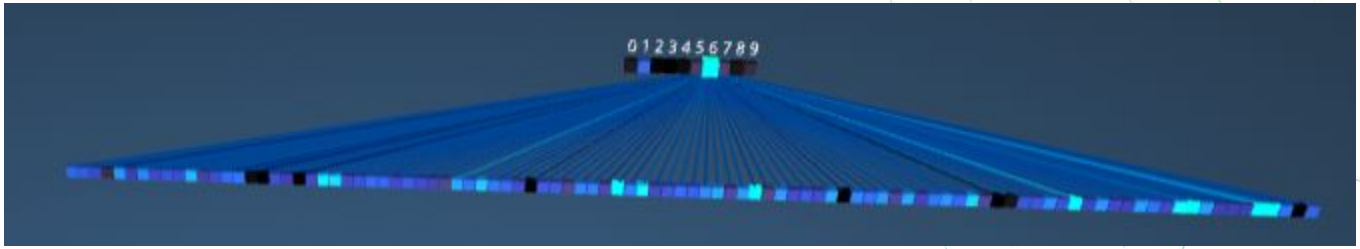


Let’s see a live example!

Features obtained by aggregating the spatial information are given different weights by each neuron in the upper layer



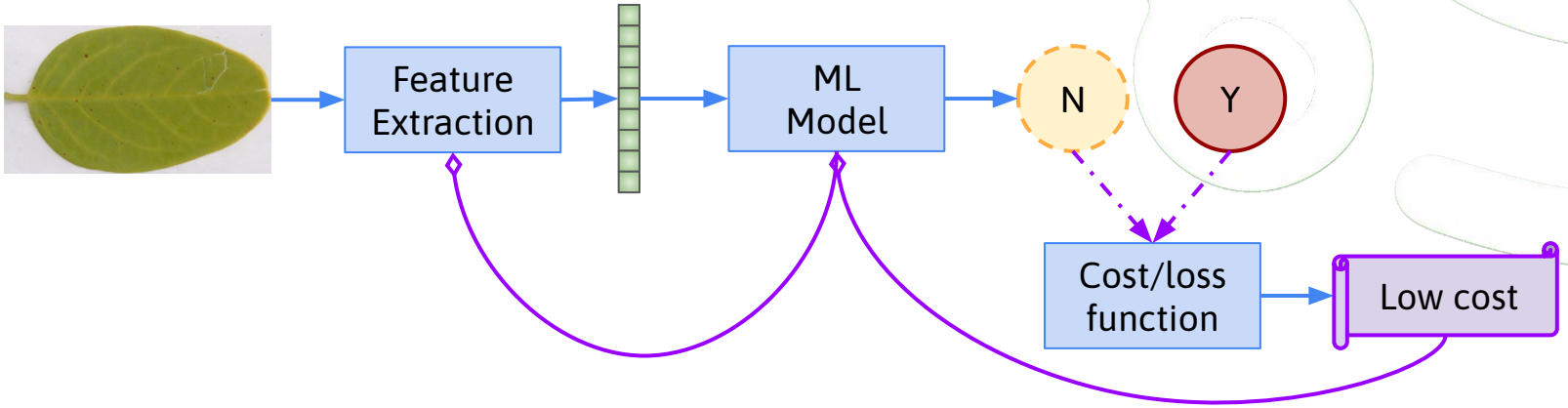
Let’s see a live example!



Finally, a **confidence score** is computed for each class/digit

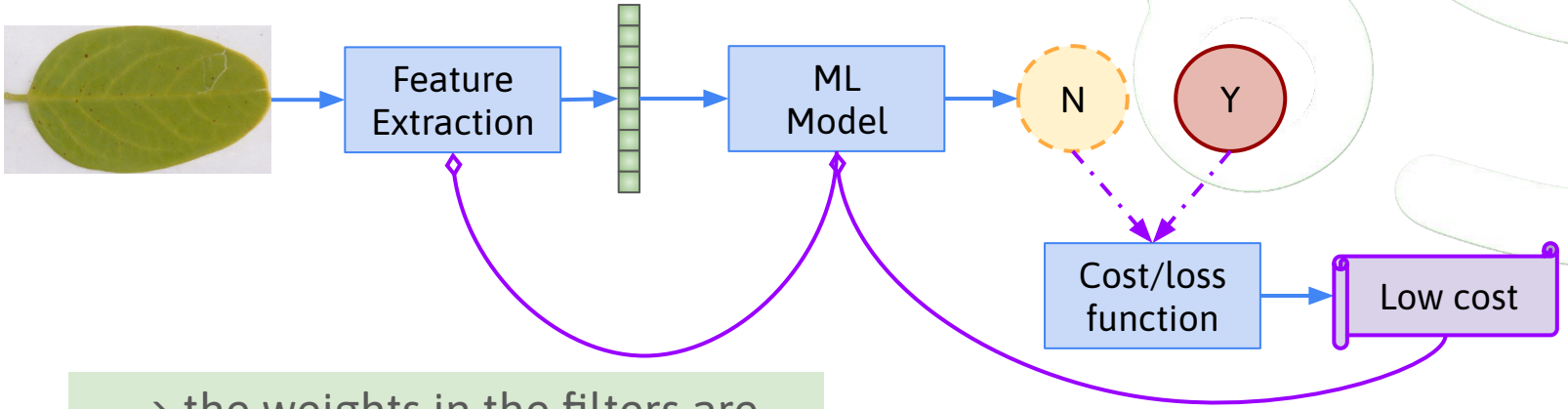
A look back at the DL Pipeline

All these convolutions act as the neurons ***learning to extract features***!



A look back at the DL Pipeline

All these convolutions act as the neurons ***learning to extract features***!

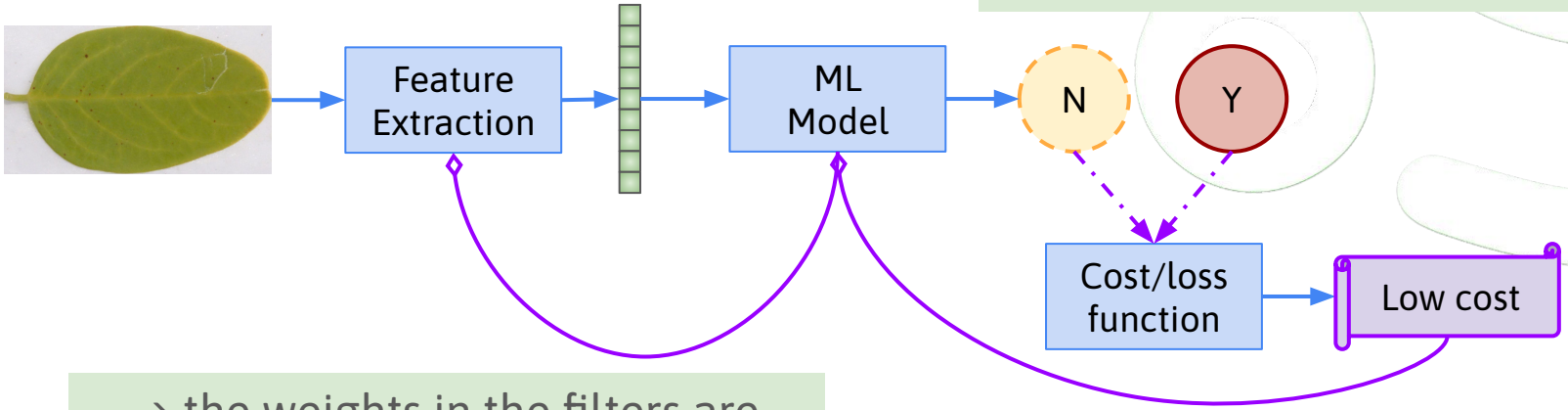


→ the weights in the filters are updated thank to the cost-driven update loop

A look back at the DL Pipeline

All these convolutions act as the neurons ***learning to extract features***!

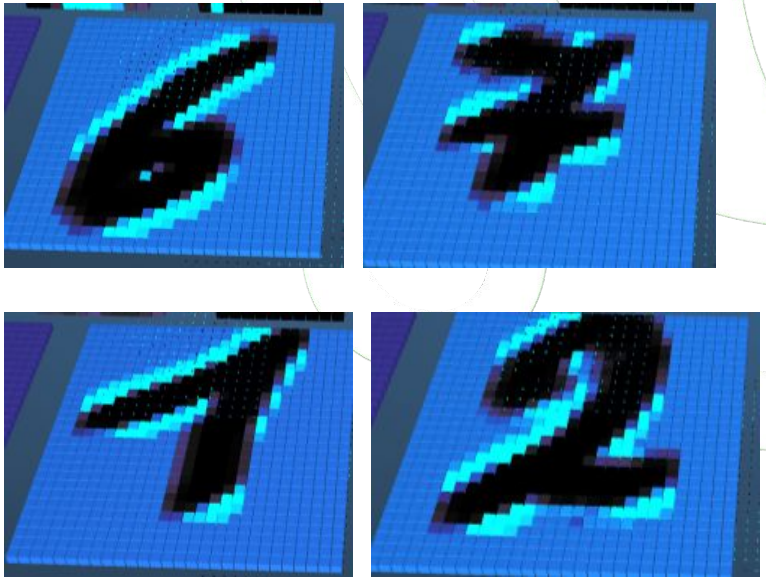
Small remark: "ML Model" could be a MLP on top of CNN features.



→ the weights in the filters are updated thank to the cost-driven update loop

What’s inside these neurons?

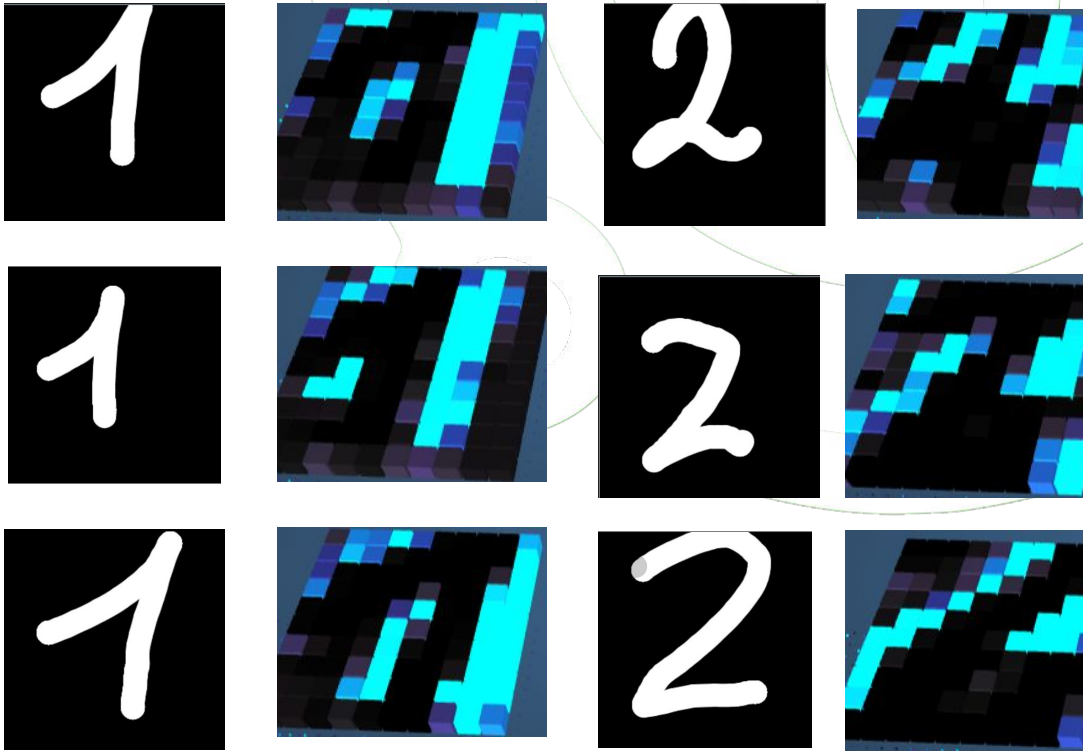
1st level features seem to capture patterns in a fixed direction



Intuition on learned features and compositionality

1st level features seem to capture patterns in a fixed direction

2nd level features seem to be more responsive if you input the same digit multiple times with different “styles”

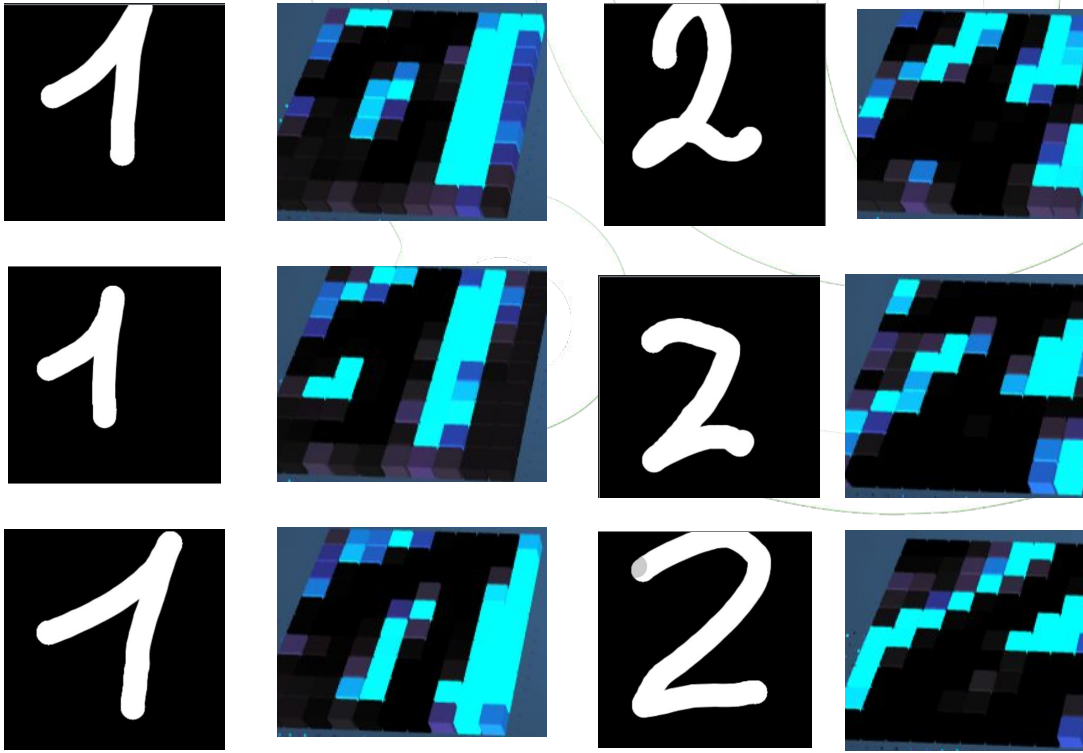


Intuition on learned features and compositionality

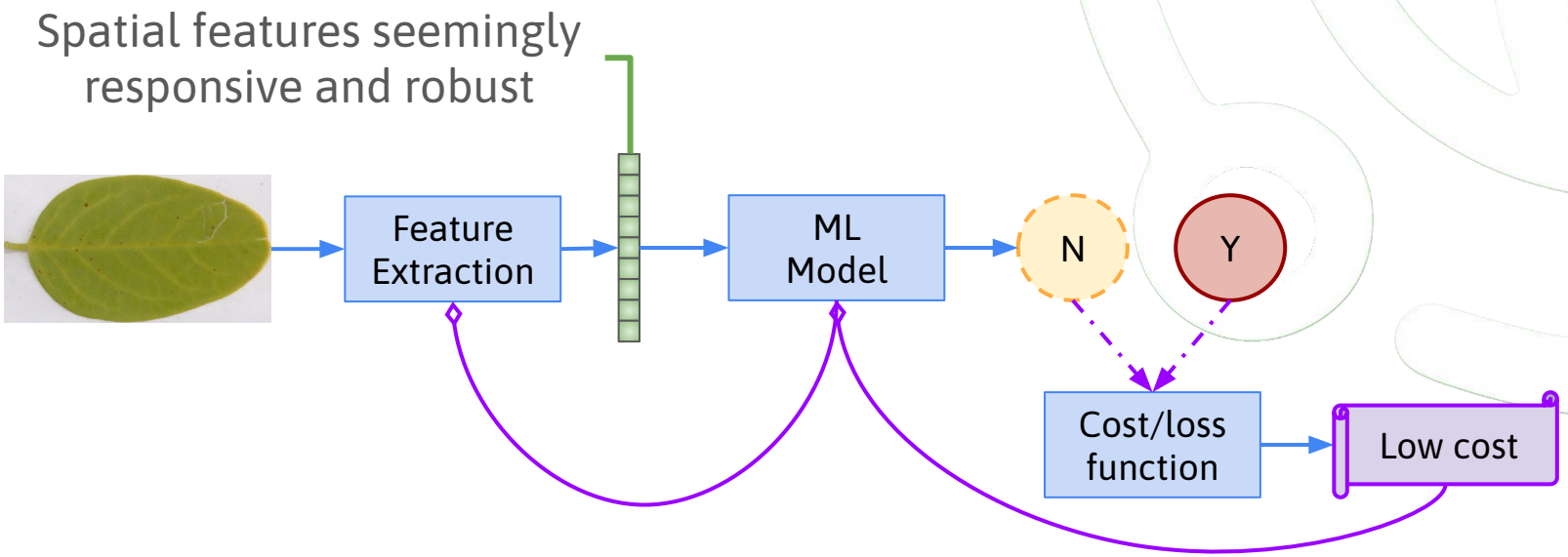
1st level features seem to capture patterns in a fixed direction

2nd level features seem to be more responsive if you input the same digit multiple times with different “styles”

What if we have “more complex” images, and maybe a “deeper” network?



A look back at the DL Pipeline



A look back at the DL Pipeline

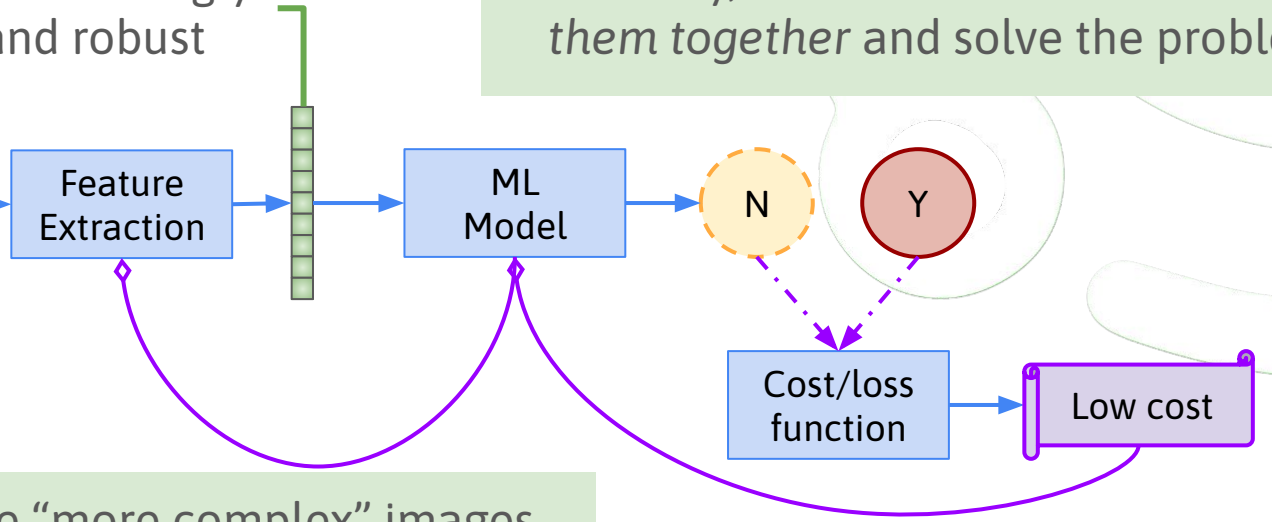
Spatial features seemingly responsive and robust

Intuitively, "ML model" *learns how to put them together* and solve the problem!

A look back at the DL Pipeline

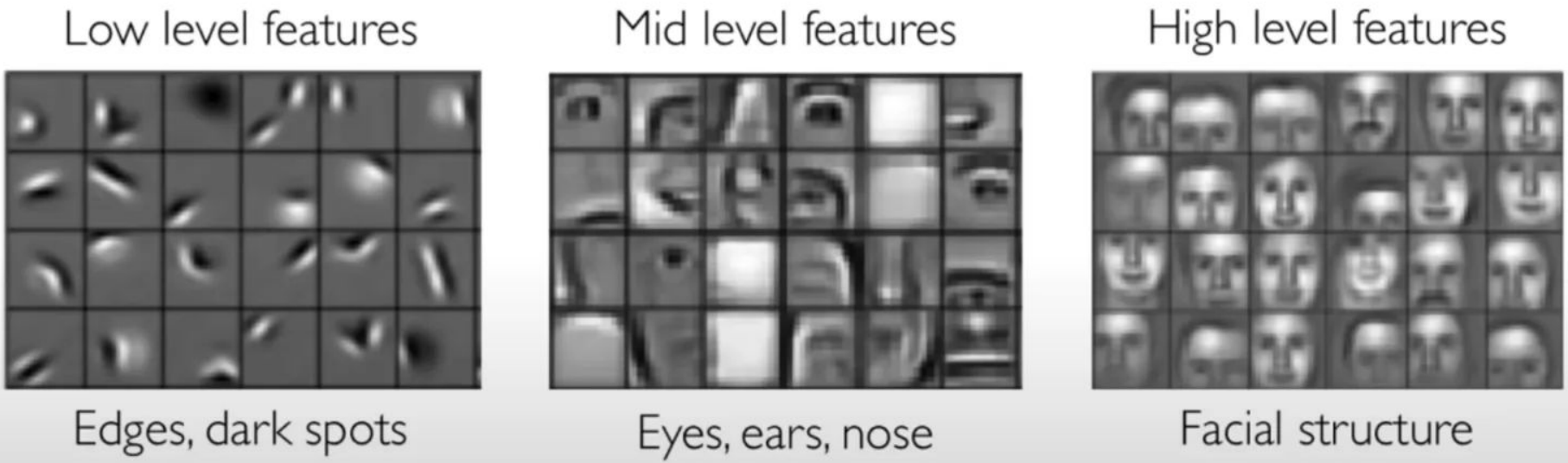
Spatial features seemingly responsive and robust

Intuitively, "ML model" *learns how to put them together* and solve the problem!

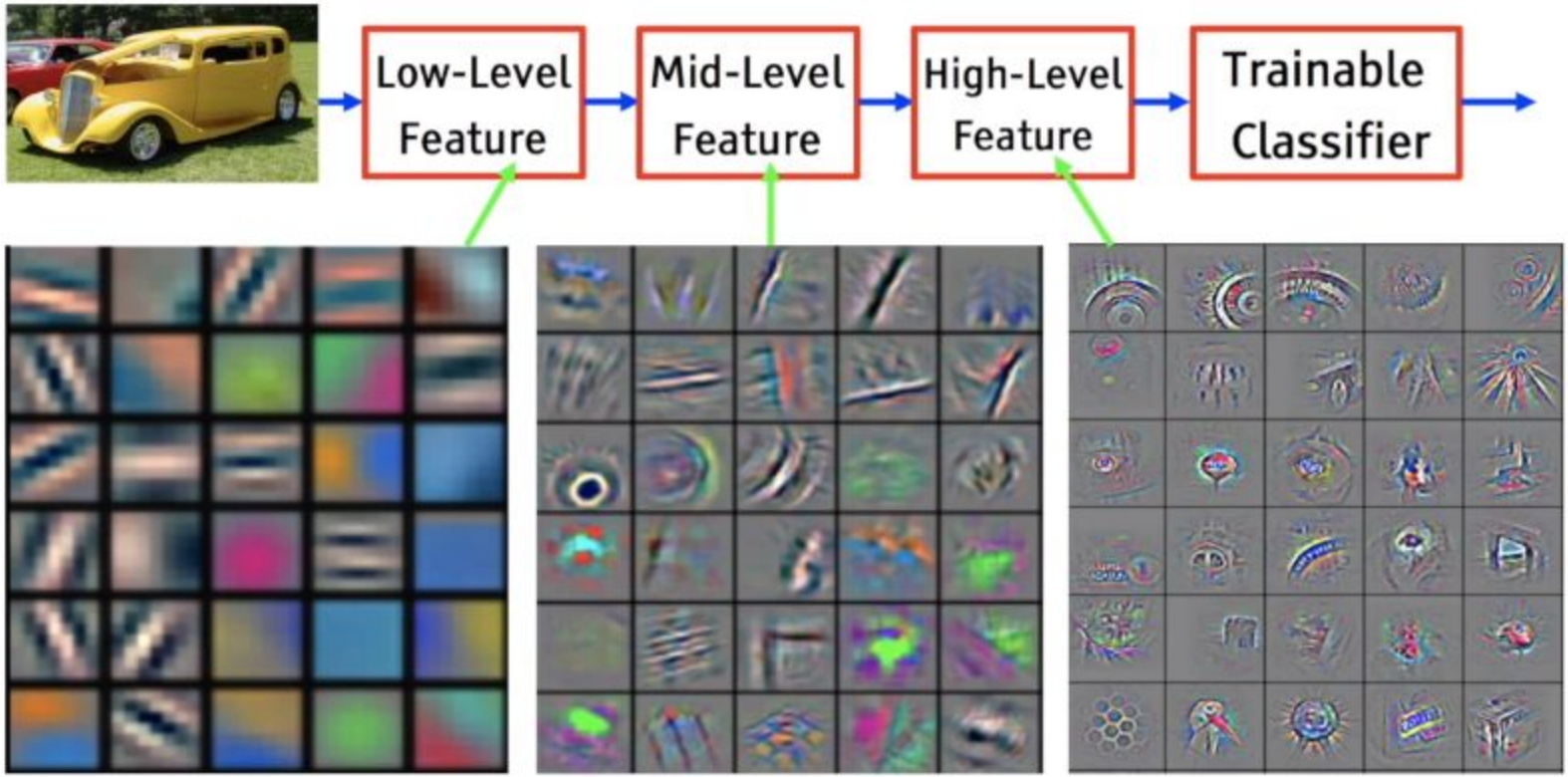


What if we have "more complex" images, and maybe a "deeper" network?

Examples from the Web



https://www.knime.com/blog/convolutional-neural-networks-computer-vision



https://medium.com/analytics-vidhya/the-world-through-the-eyes-of-cnn-5a52c034dbeb

Advantages of CNNs (recap)

- They can work directly on images
- They are aware of spatial relations between pixels

Advantages of CNNs (recap)

- They can work directly on images
- They are aware of spatial relations between pixels
- Compared to MLPs, they can have fewer trainable parameters (less prone to overfitting data)

Advantages of CNNs (recap)

- They can work directly on images
- They are aware of spatial relations between pixels
- Compared to MLPs, they can have fewer trainable parameters (less prone to overfitting data)
- They can automatically learn complex, compositional features from images
    - Especially, with deeper networks

Advantages of CNNs (recap)

- They can work directly on images
- They are aware of spatial relations between pixels
- Compared to MLPs, they can have fewer trainable parameters (less prone to overfitting data)
- They can automatically learn complex, compositional features from images
    - Especially, with deeper networks

Generally speaking, it’s not easy to build very deep networks (although the intuition seems to say “the deeper the better”)

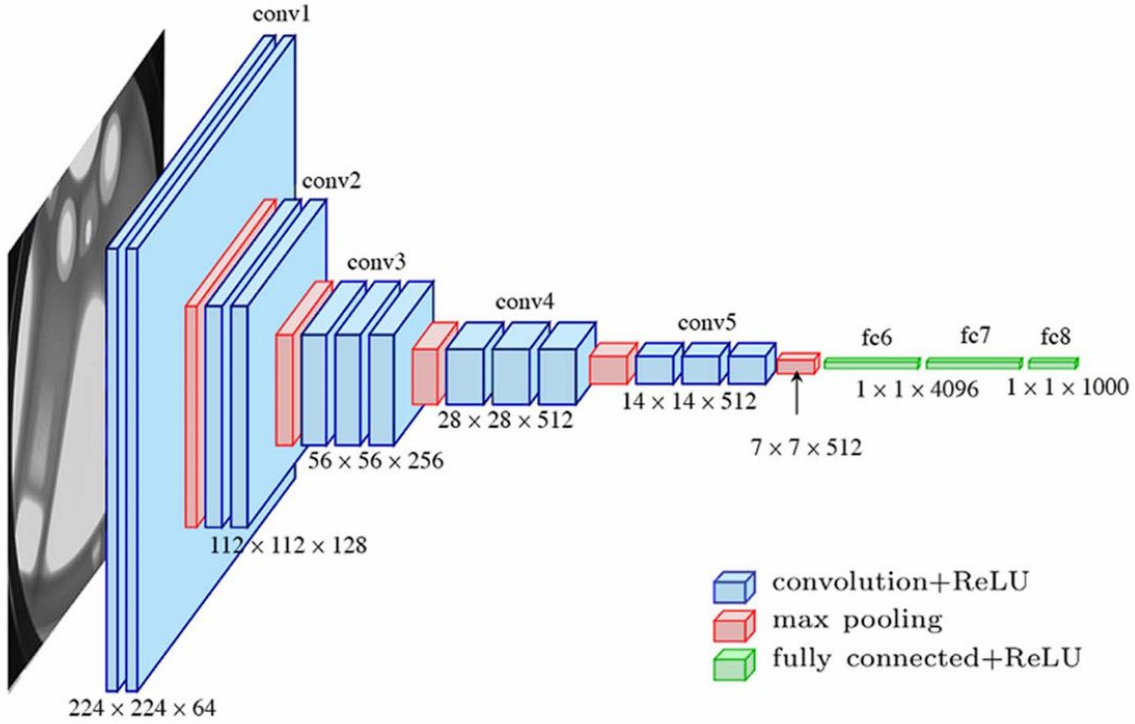
Advantages of CNNs (recap)

- They can work directly on images
- They are aware of spatial relations between pixels
- Compared to MLPs, they can have fewer trainable parameters (less prone to overfitting data)
- They can automatically learn complex, compositional features from images
    - Especially, with deeper networks

Generally speaking, it's not easy to build very deep networks (although the intuition seems to say "the deeper the better")
We look at popular examples (which also work well in practice)

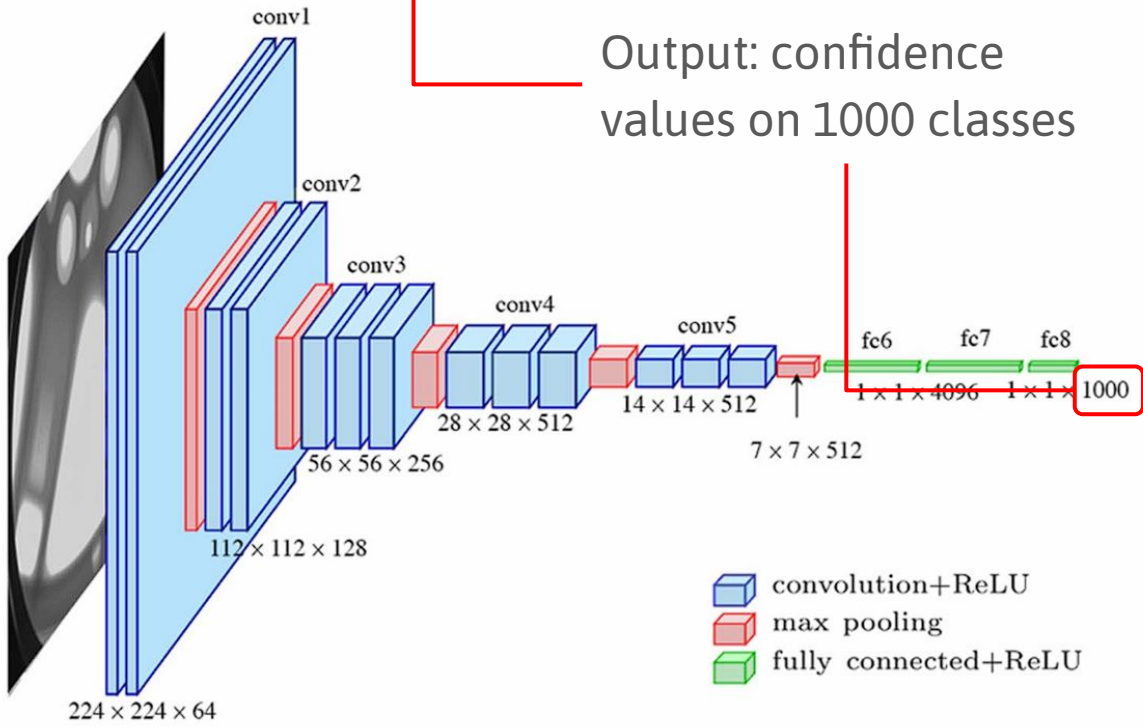
Stacking more conv layers - VGGNet (2014)

VGGNet came up as a runner-up for a popular “multiclass problem” on images.



Stacking more conv layers - VGGNet (2014)

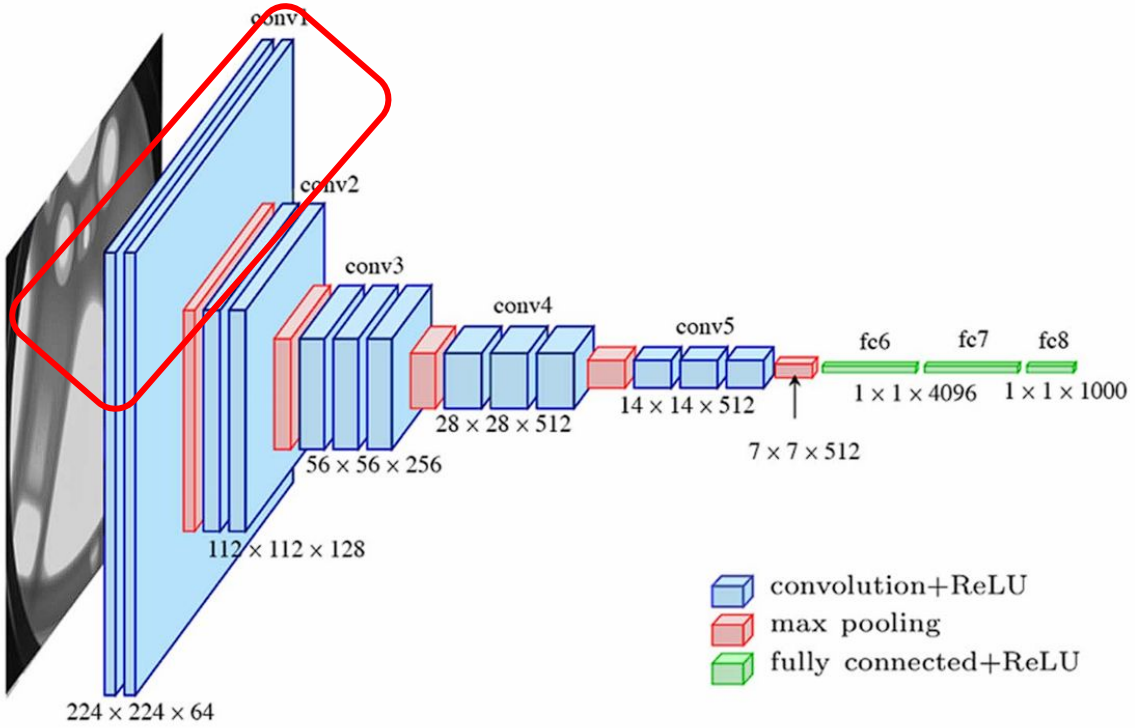
VGGNet came up as a runner-up for a popular “multiclass problem” on images.



Stacking more conv layers - VGGNet (2014)

VGGNet came up as a runner-up for a popular “multiclass problem” on images.

Regular structure: convolutions and maxpool (downsampling)

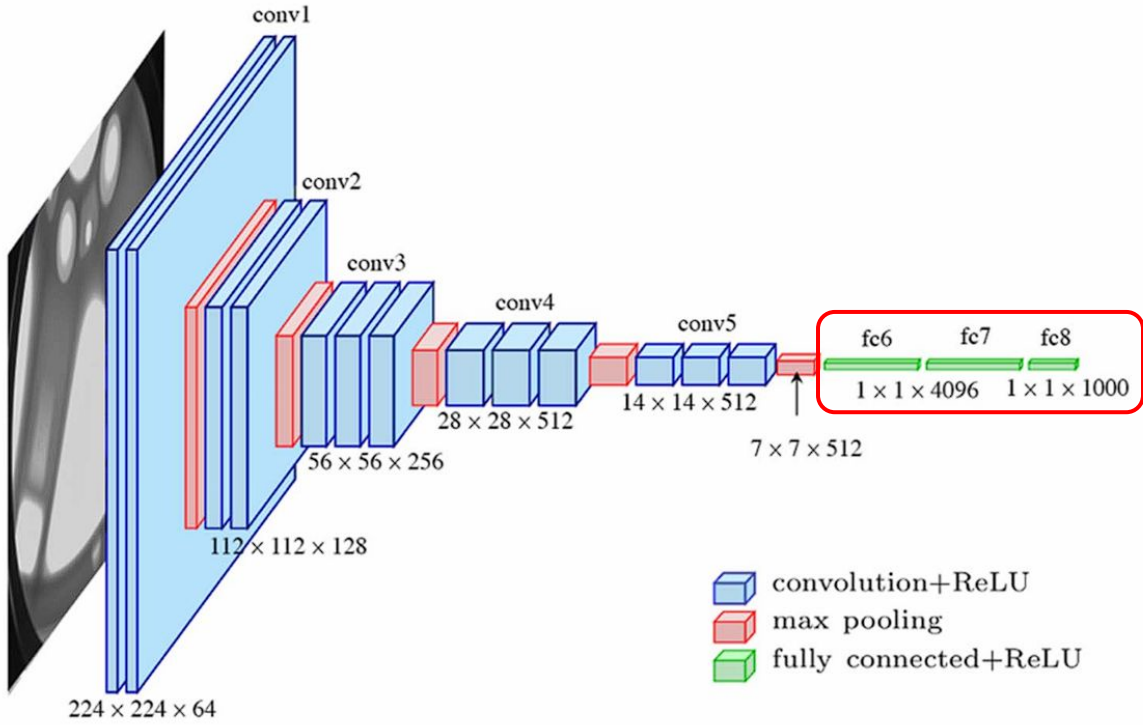


Stacking more conv layers - VGGNet (2014)

VGGNet came up as a runner-up for a popular “multiclass problem” on images.

Regular structure: convolutions and maxpool (downsampling)

You can see a MLP on top of the final “high level” features extracted from the conv layers to solve the classification!



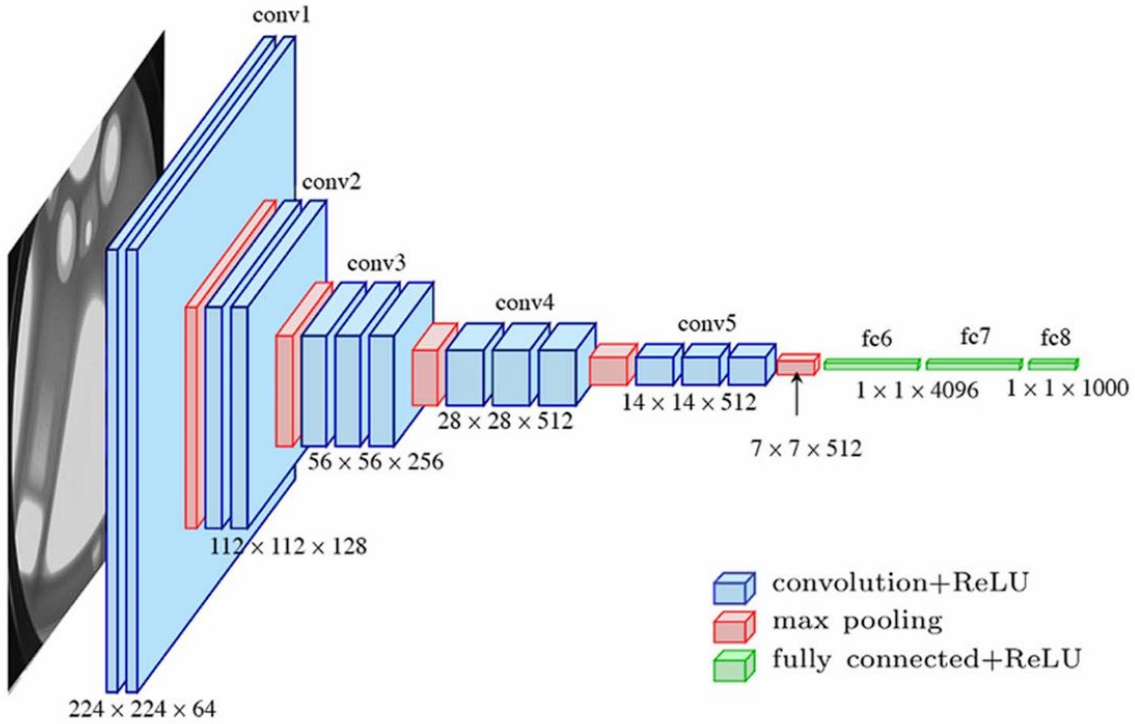
Stacking more conv layers - VGGNet (2014)

VGGNet came up as a runner-up for a popular “multiclass problem” on images.

Regular structure: convolutions and maxpool (downsampling)

You can see a MLP on top of the final “high level” features extracted from the conv layers to solve the classification!

16 layers (blue+green→trainable ones). Also a 19 layers variant.



More layers?

It seems that “more layers → better performance”.

More layers?

It seems that “more layers → better performance”.

Can we stack more?

More layers?

It seems that “more layers → better performance”. *Can we stack more?*

Some problems arise:

- Performance degradation

More layers?

It seems that “more layers → better performance”.

Can we stack more?

Some problems arise:

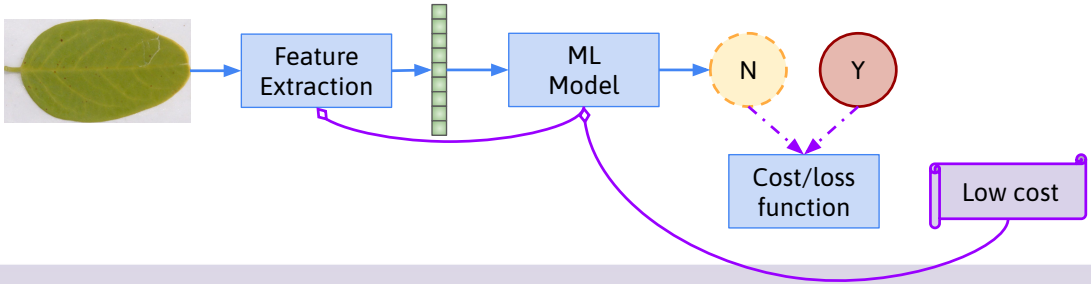
- Performance degradation
- Models do not seem to train that well anymore

More layers?

It seems that “more layers → better performance”. *Can we stack more?*

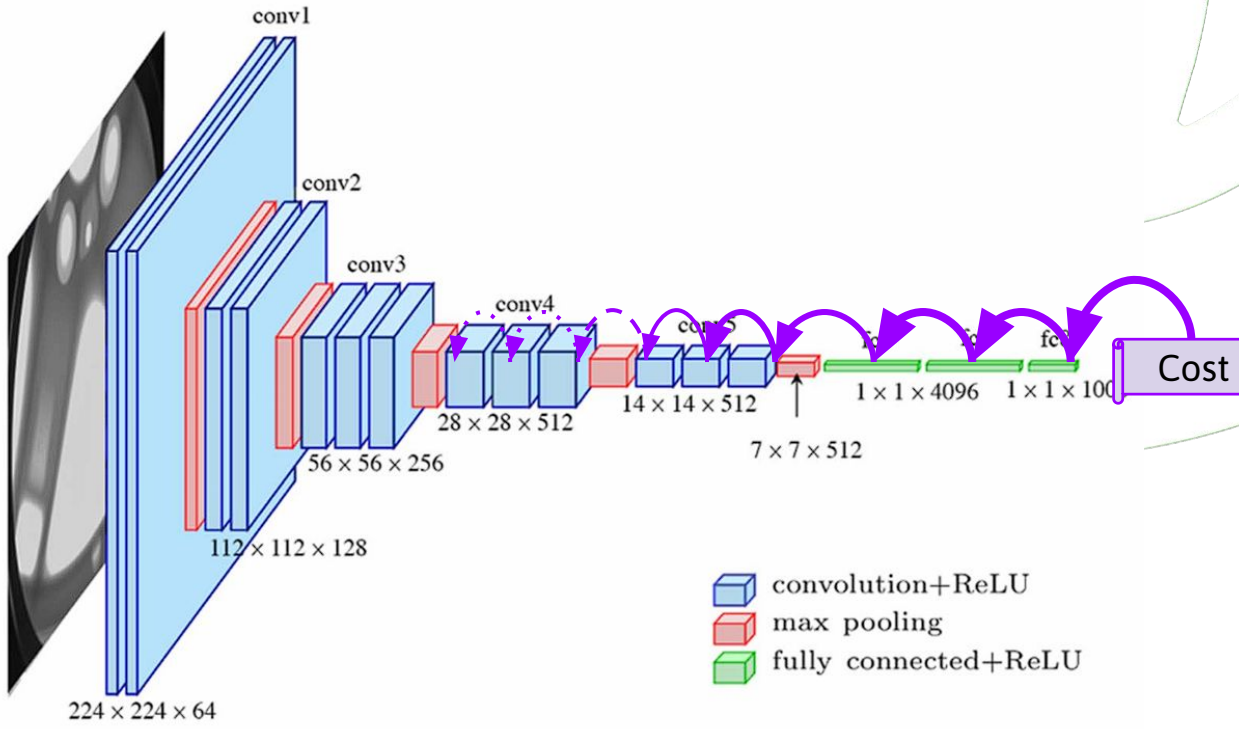
Some problems arise:

- Performance degradation
- Models do not seem to train that well anymore
  - Analysing the *“backward” information flow* we see that the one going to the feature extraction module (e.g. CNN layers) is actually **vanishing**



Vanishing gradients

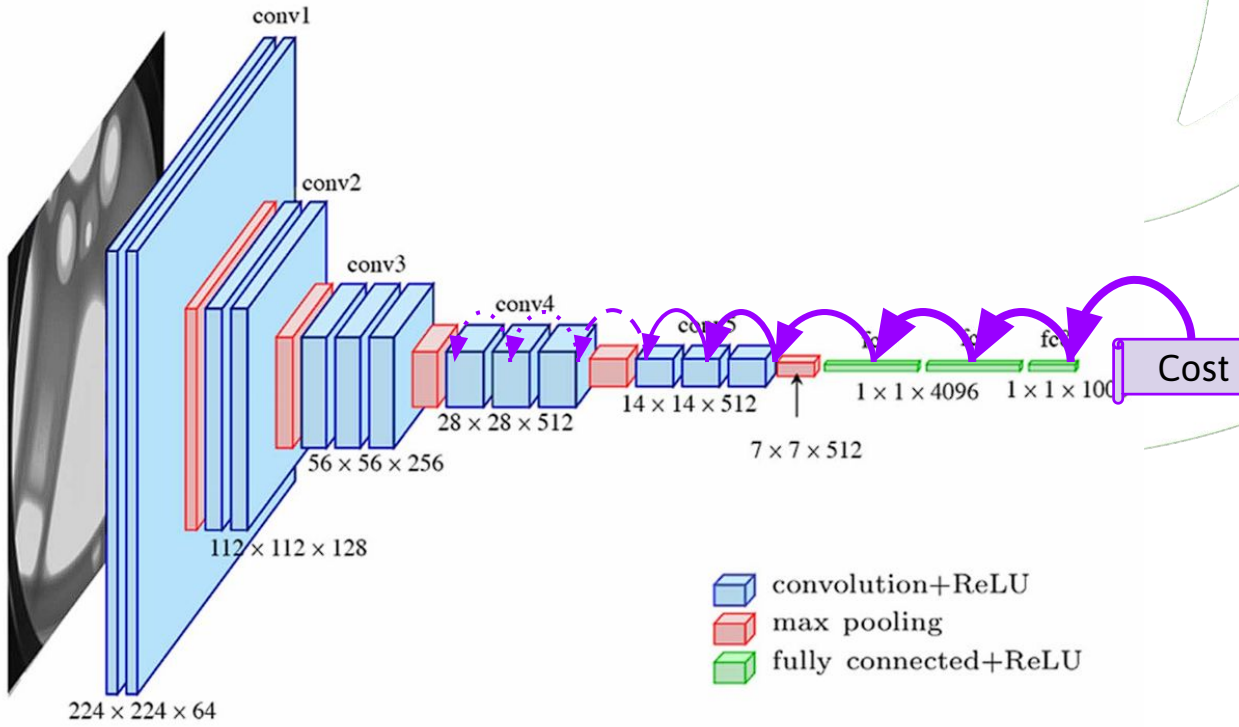
Cost/loss function → useful information (*gradients*) suggesting *how to adjust the weights* to lower the error on the current training samples.



Vanishing gradients

Cost/loss function → useful information (*gradients*) suggesting *how to adjust the weights* to lower the error on the current training samples.

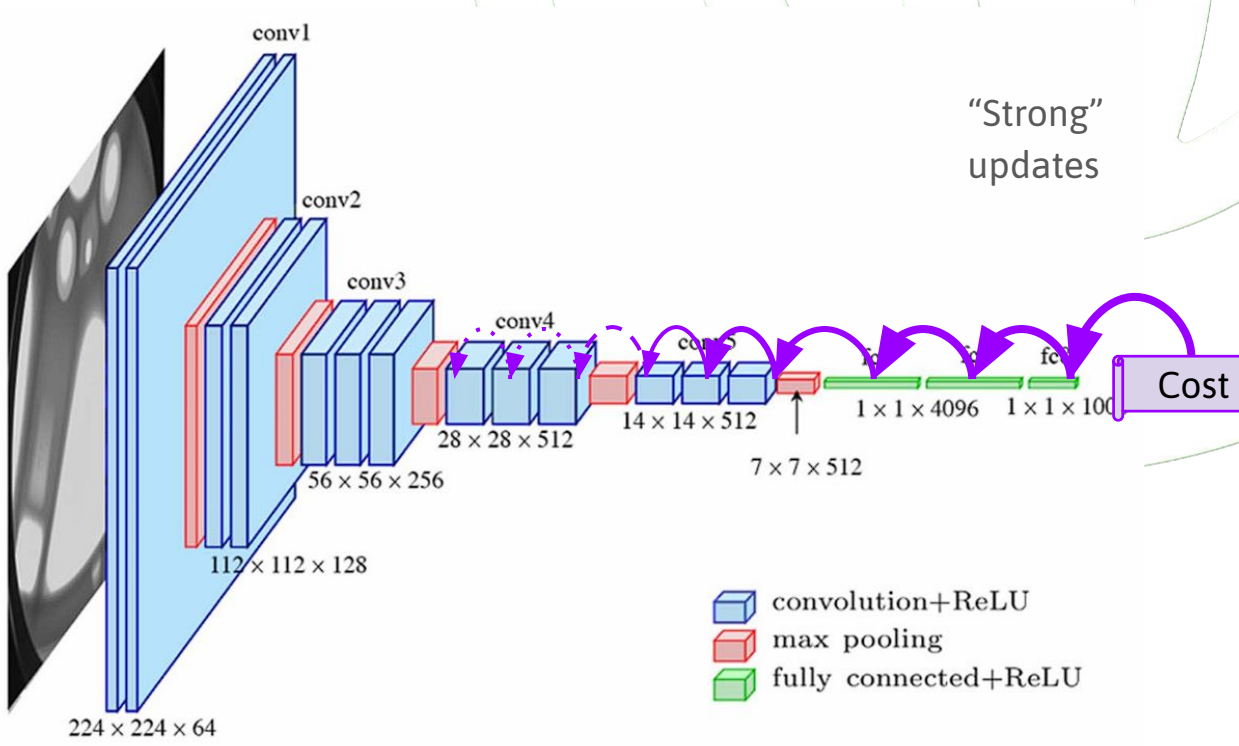
As gradients are tailored for each layer, they decrease in magnitude.



Vanishing gradients

Cost/loss function → useful information (*gradients*) suggesting *how to adjust the weights* to lower the error on the current training samples.

As gradients are tailored for each layer, they decrease in magnitude.

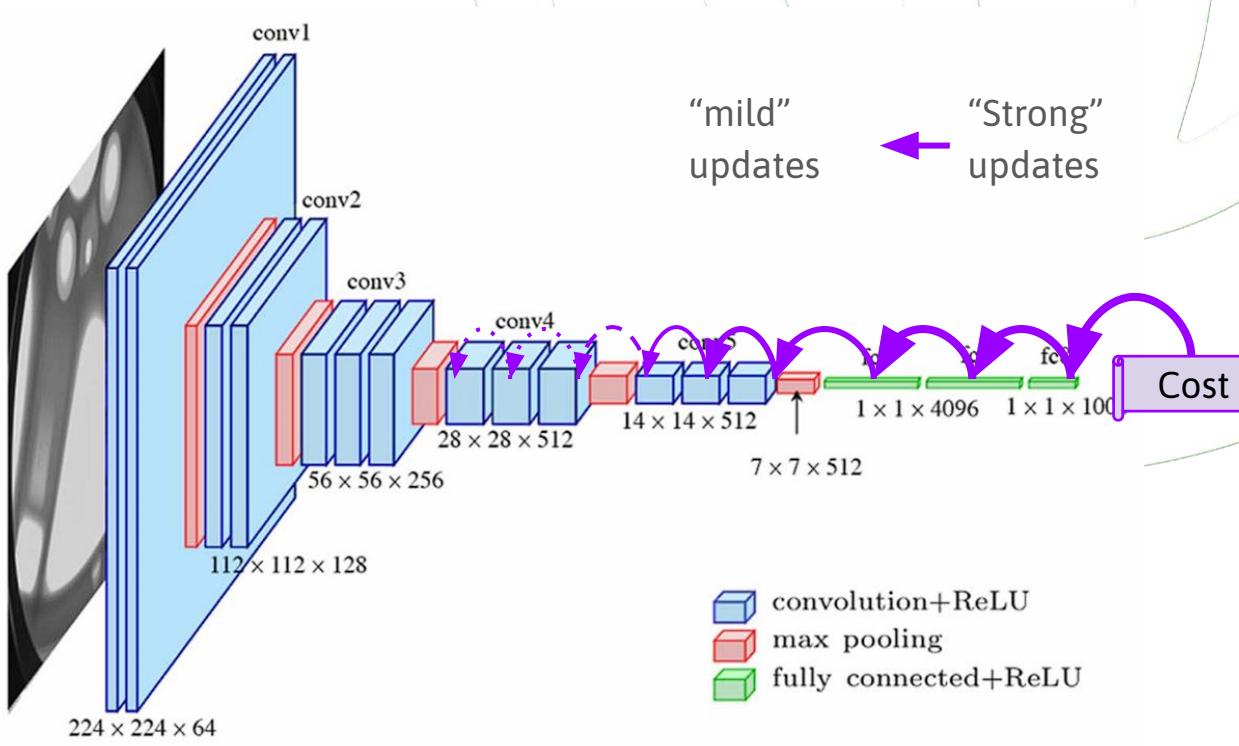


"Strong" updates

Vanishing gradients

Cost/loss function → useful information (*gradients*) suggesting *how to adjust the weights* to lower the error on the current training samples.

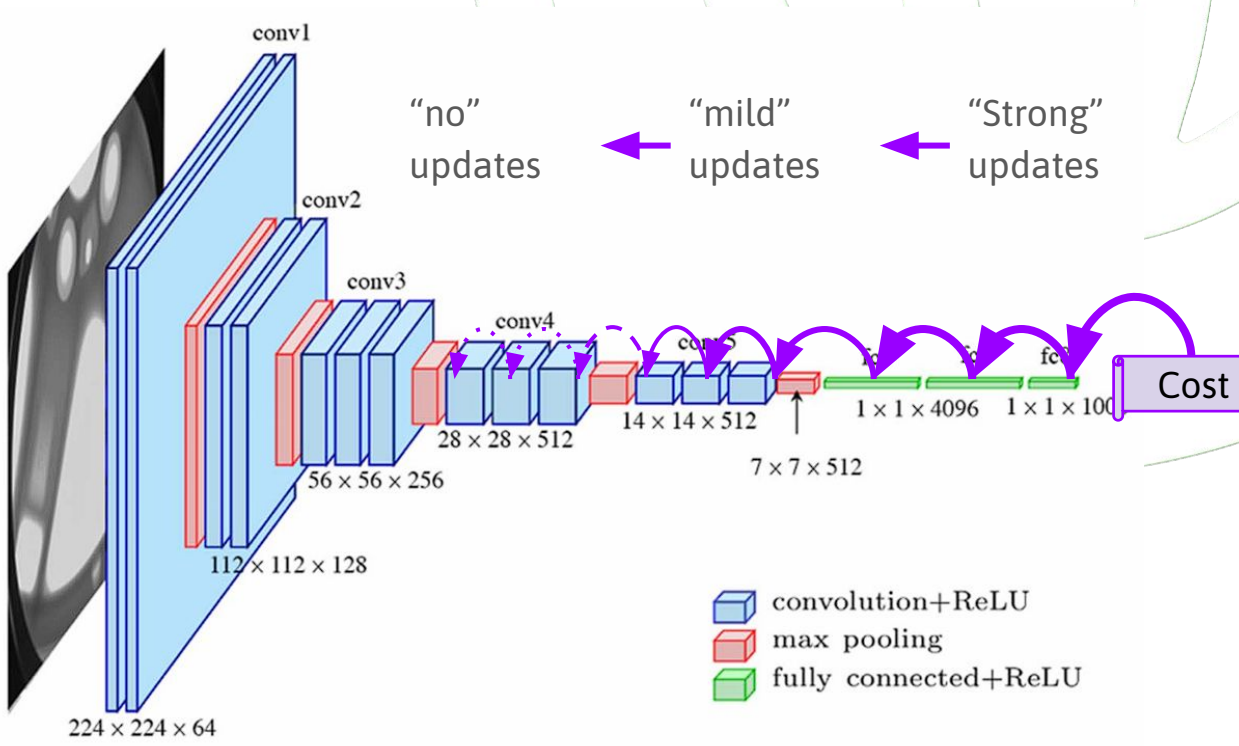
As gradients are tailored for each layer, they decrease in magnitude.



Vanishing gradients

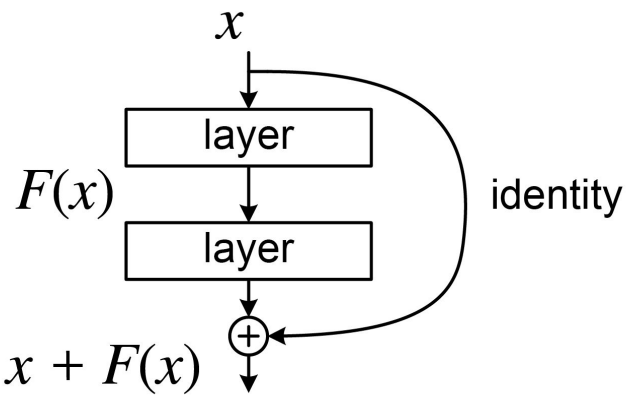
Cost/loss function → useful information (*gradients*) suggesting *how to adjust the weights* to lower the error on the current training samples.

As gradients are tailored for each layer, they decrease in magnitude.



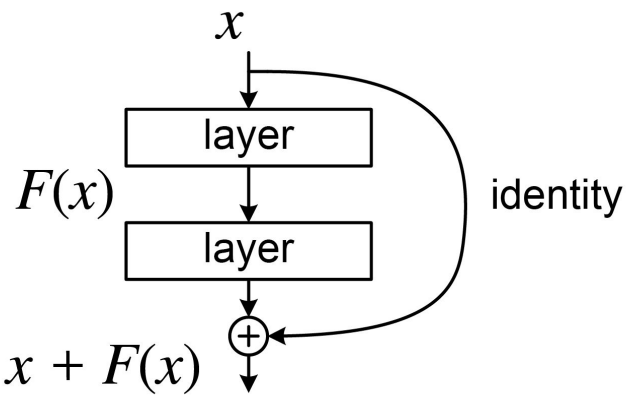
ResNet (2015) - addressing the vanishing gradient

Key idea in ResNet: **skip connections**.



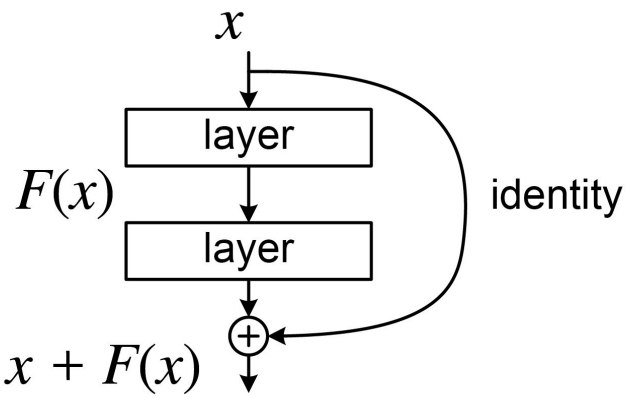
ResNet (2015) - addressing the vanishing gradient

Key idea in ResNet: **skip connections**.
After every “few” layers, we “bring back” the previous input x.



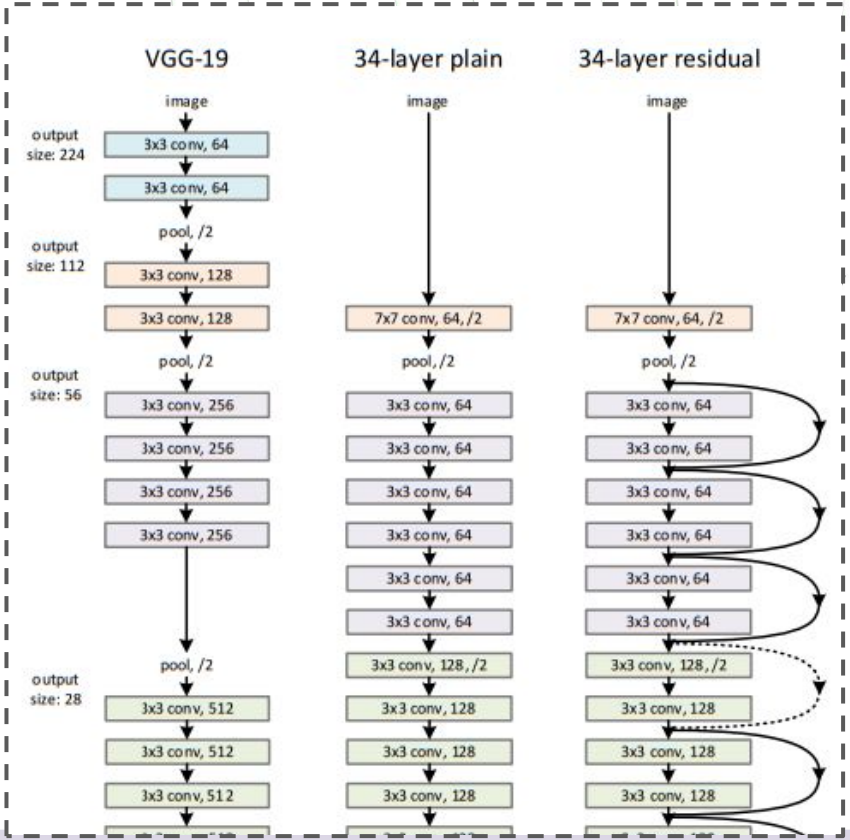
ResNet (2015) - addressing the vanishing gradient

Key idea in ResNet: **skip connections**.
After every “few” layers, we “bring back” the previous input x.
Some of the gradient information, will go ***through*** the skip connection, instead of being “only” tailored for each of the layers that would result in progressive vanishing.



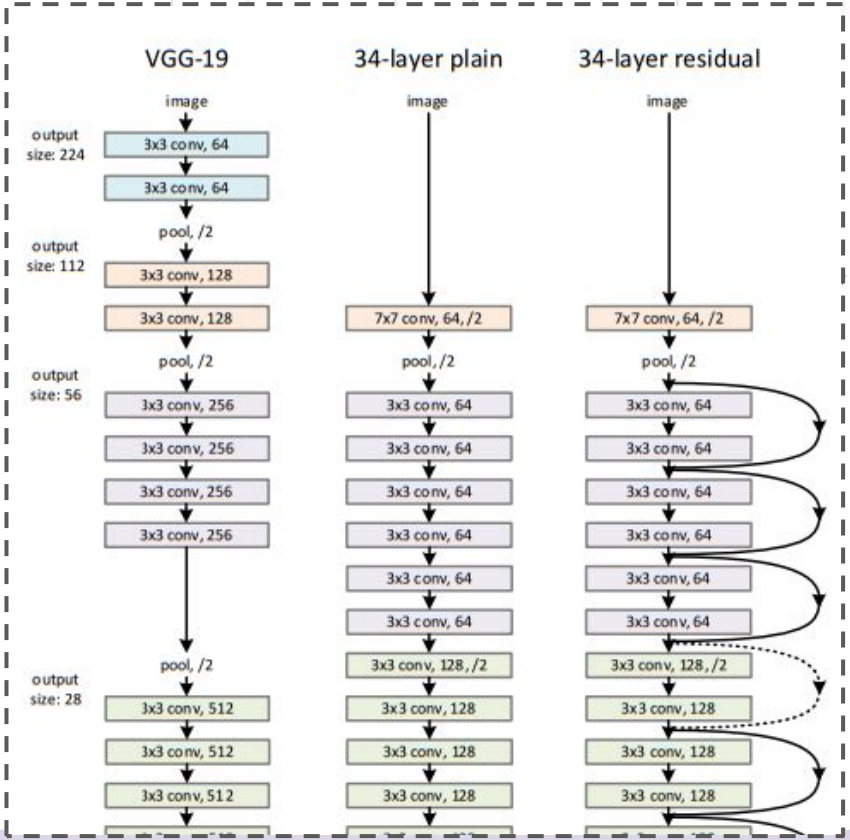
ResNet (2015) - addressing the vanishing gradient

Key idea in ResNet: **skip connections**.
After every “few” layers, we “bring back” the previous input *x*.
Some of the gradient information, will go ***through*** the skip connection, instead of being “only” tailored for each of the layers that would result in progressive vanishing.



ResNet (2015) - addressing the vanishing gradient

Key idea in ResNet: **skip connections**.
After every “few” layers, we “bring back” the previous input *x*.
Some of the gradient information, will go ***through*** the skip connection, instead of being “only” tailored for each of the layers that would result in progressive vanishing.
With this trick, they got progressive improvements up to *152* layers.



What if we have *sequential* data?

Some data are naturally *sequential* or with some *temporal* relations.
E.g. growth of plants

What if we have *sequential* data?

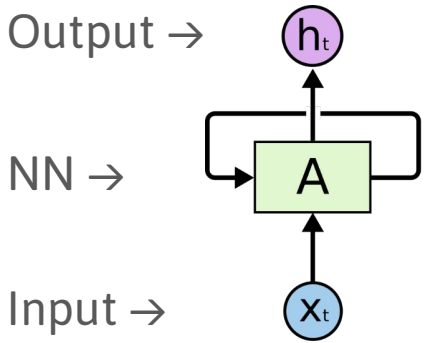
Some data are naturally *sequential* or with some *temporal* relations.
E.g. growth of plants

First intuition: a *sequence* of data, so why don't we process the elements of the sequence one by one?

Recurrent Neural Networks (RNN)

Some data are naturally *sequential* or with some *temporal* relations.
E.g. growth of plants

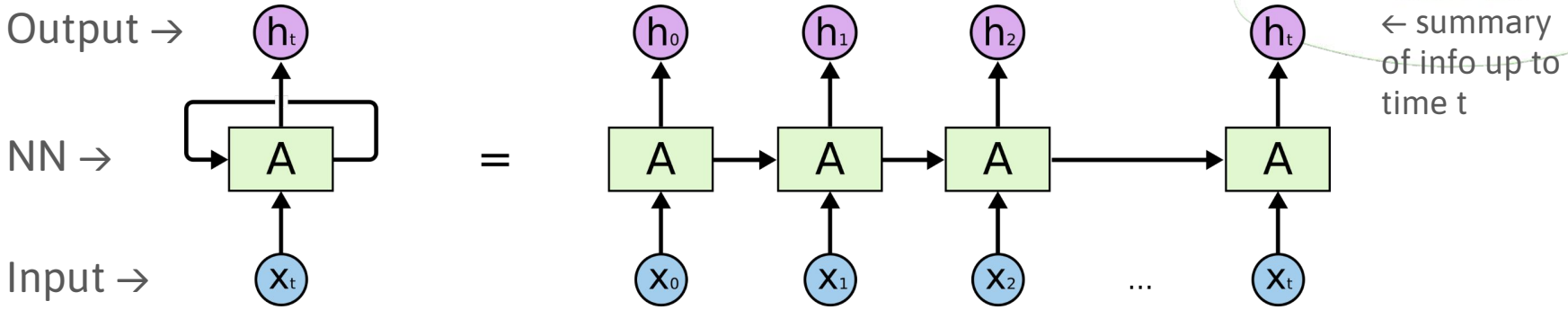
First intuition: a *sequence* of data, so why don't we process the elements of the sequence one by one?



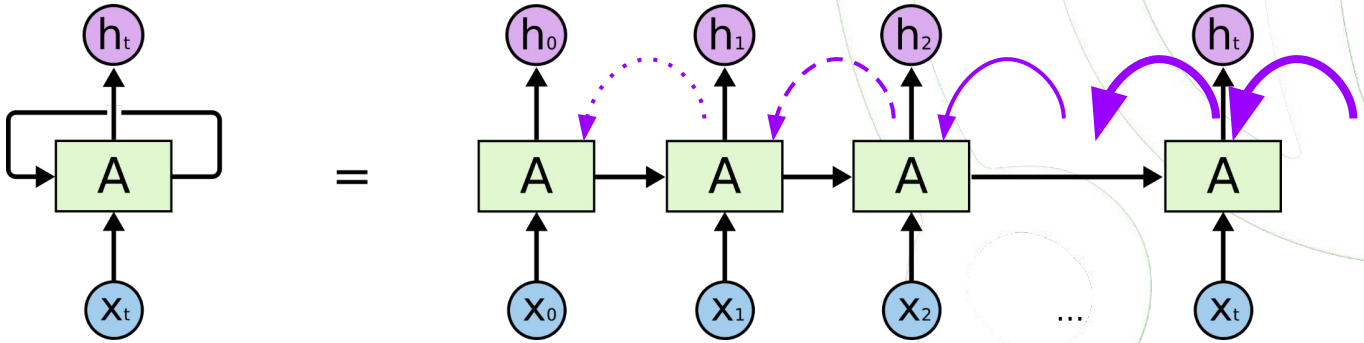
Recurrent Neural Networks (RNN)

Some data are naturally *sequential* or with some *temporal* relations.
E.g. growth of plants

First intuition: a *sequence* of data, so why don't we process the elements of the sequence one by one?

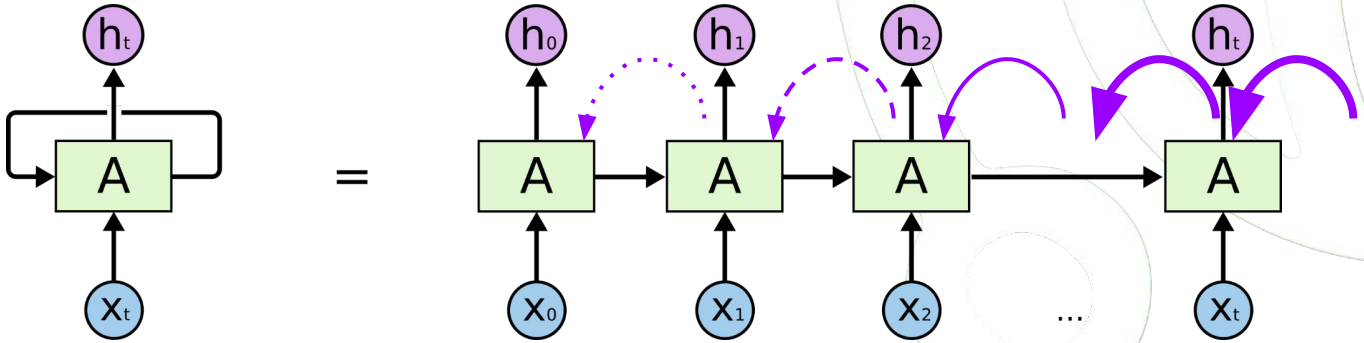


Vanishing gradient in RNNs



As with other deep networks → vanishing gradients.

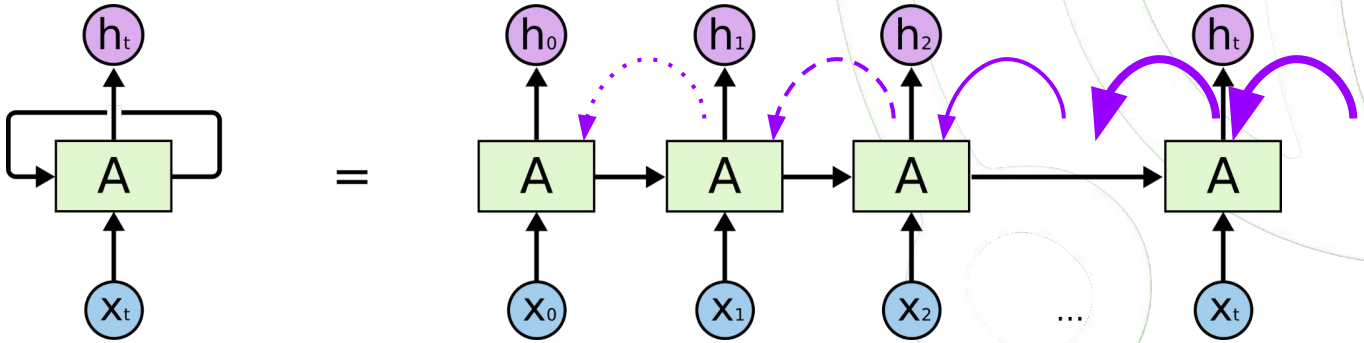
Vanishing gradient in RNNs



As with other deep networks → vanishing gradients.

There is also a problem with *long-term dependencies*.

Vanishing gradient in RNNs

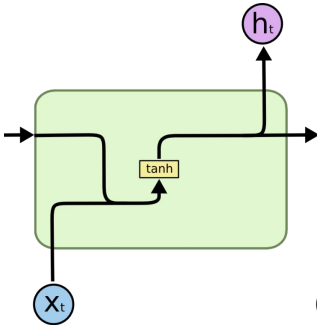


As with other deep networks → vanishing gradients.

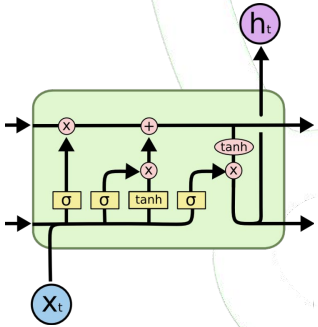
There is also a problem with *long-term dependencies*.

E.g. “I grew up in France, then I moved to England. I studied English, … Nonetheless, I speak fluent [ ]” to predict the next word [*French*], the model needs to remember the initial part of the sentence.

LSTM and GRU

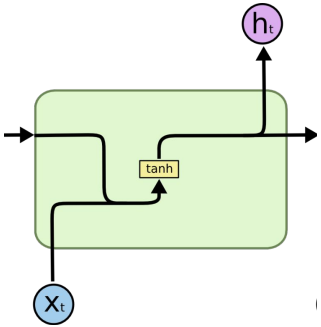


Basic RNNs use a single network (MLP-like) to update the "summary" h_t

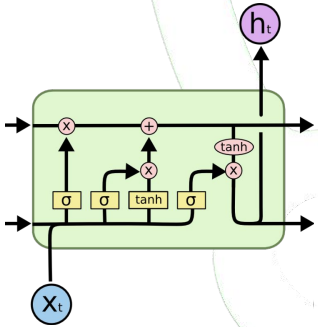


LSTMs are more complex and use **four** networks to decide which features to update and how.

LSTM and GRU

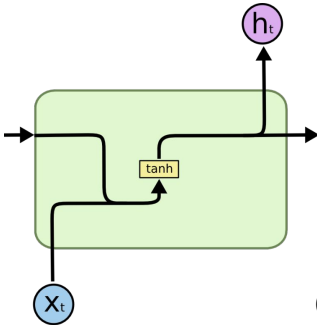


Basic RNNs use a single network (MLP-like) to update the "summary" h_t

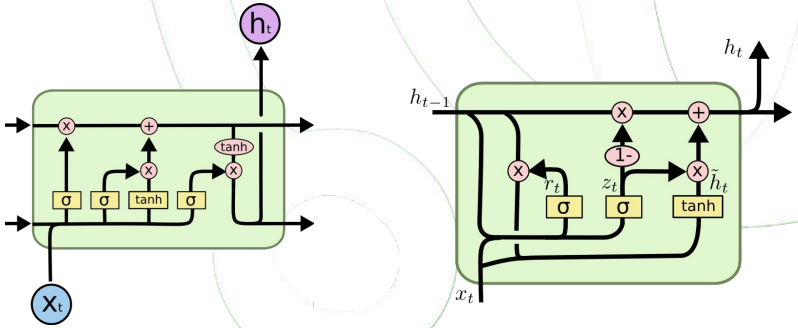


LSTMs are more complex and use **four** networks to decide which features to update and how. This makes them a little prone to overfitting.

LSTM and GRU



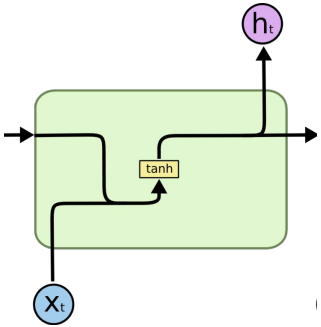
Basic RNNs use a single network (MLP-like) to update the "summary" h_t



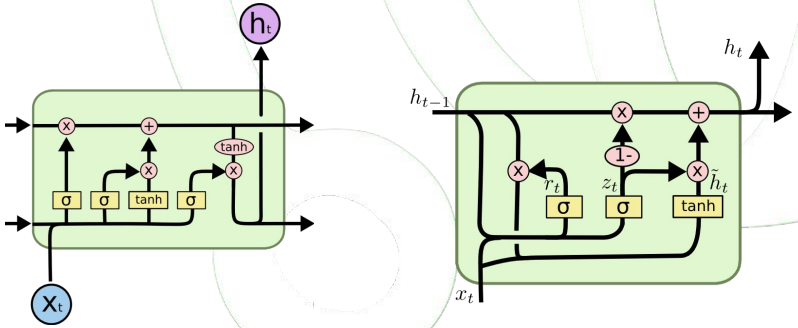
LSTMs are more complex and use **four** networks to decide which features to update and how. This makes them a little prone to overfitting.

GRUs are simpler (3 nets) yet they usually lead to more general features and better results.

LSTM and GRU



Basic RNNs use a single network (MLP-like) to update the "summary" h_t



LSTMs are more complex and use **four** networks to decide which features to update and how. This makes them a little prone to overfitting.

GRUs are simpler (3 nets) yet they usually lead to more general features and better results.

Great reference: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Transformers

Although great and high performing, RNNs need to process words one by one.

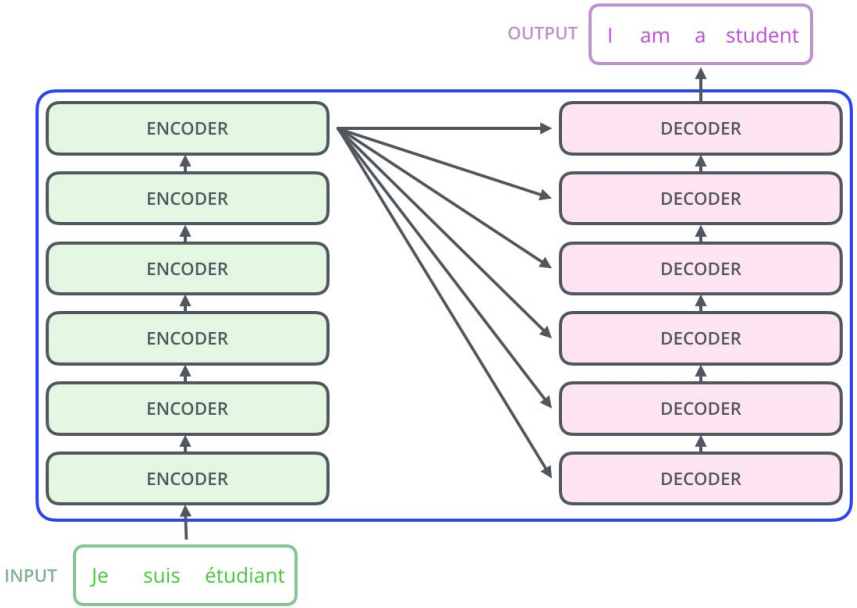
Transformers

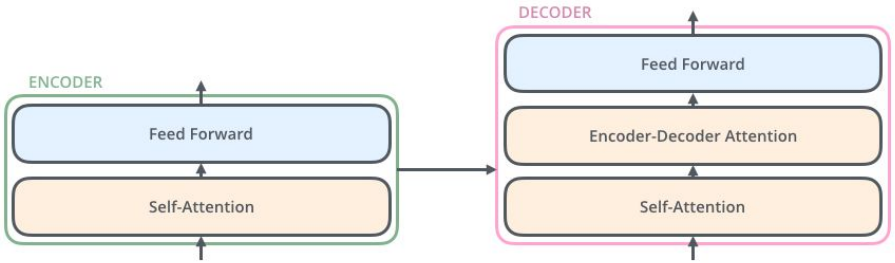
Although great and high performing, RNNs need to process words one by one. *Problematic* if we need to train our models on very long descriptions of millions of words.

Transformers

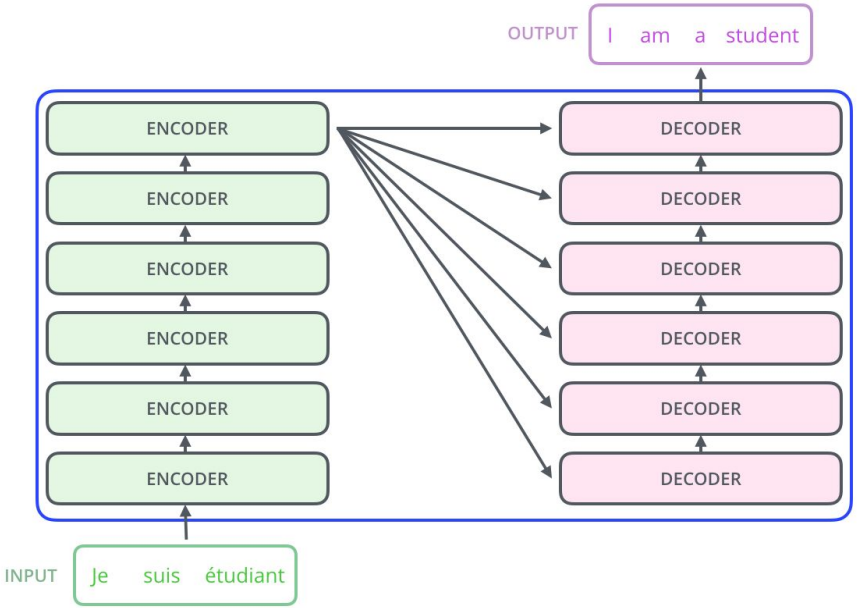
Although great and high performing, RNNs need to process words one by one.
Problematic if we need to train our models on very long descriptions of millions of words.
Transformers → good mix of easy-to-parallelize components (e.g. MLP) → no need for sequential processing of words.

Transformers

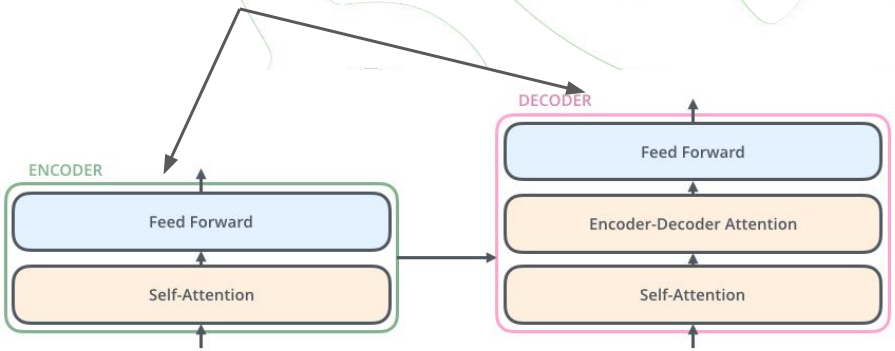




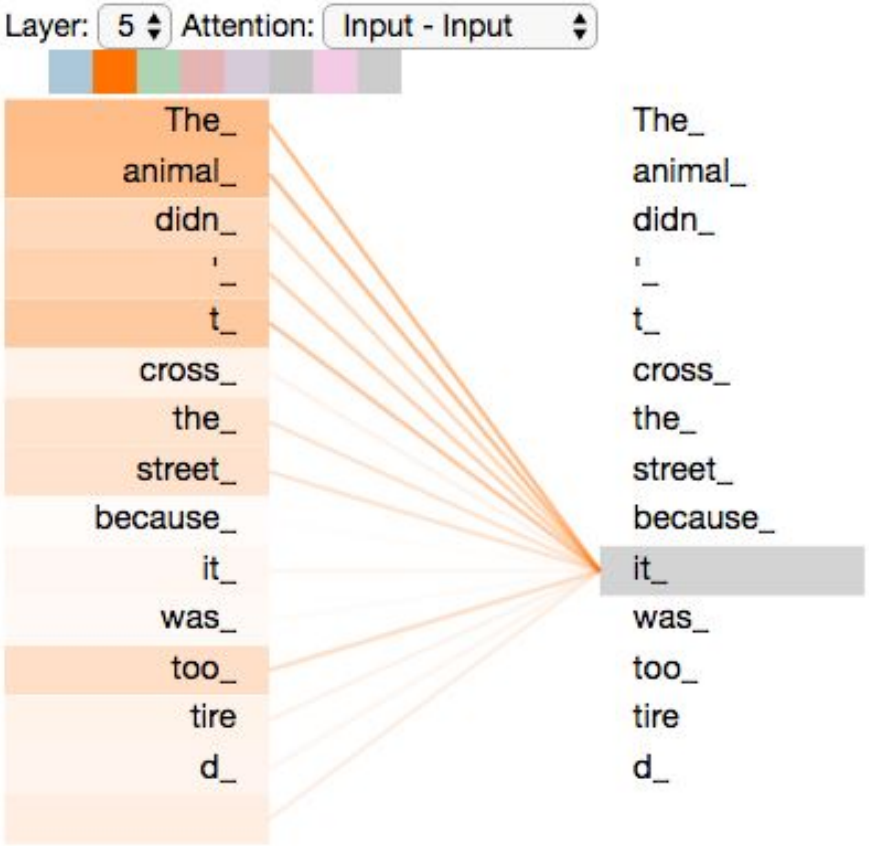
Transformers



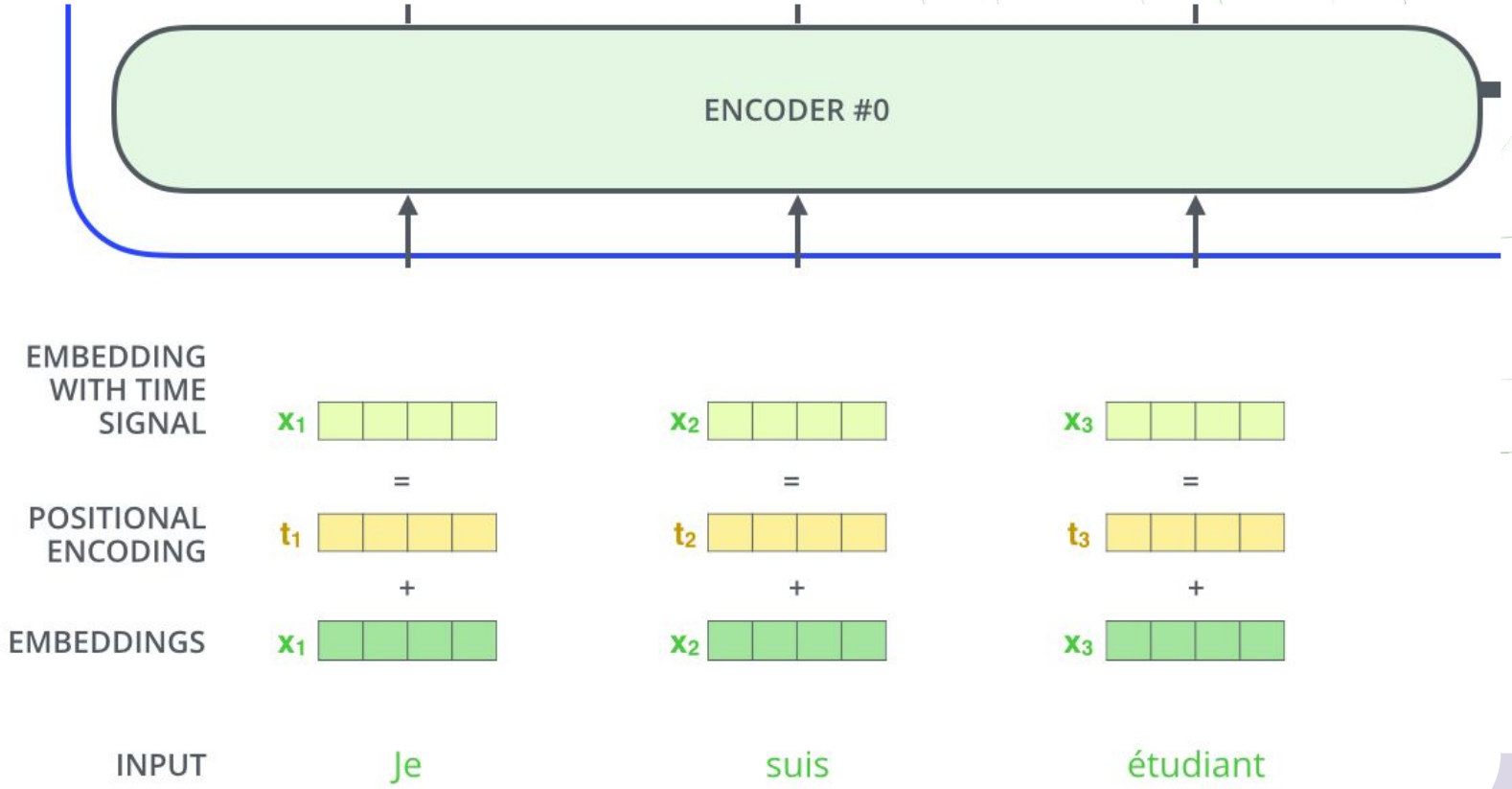
MLP-like



Transformers - self-attention

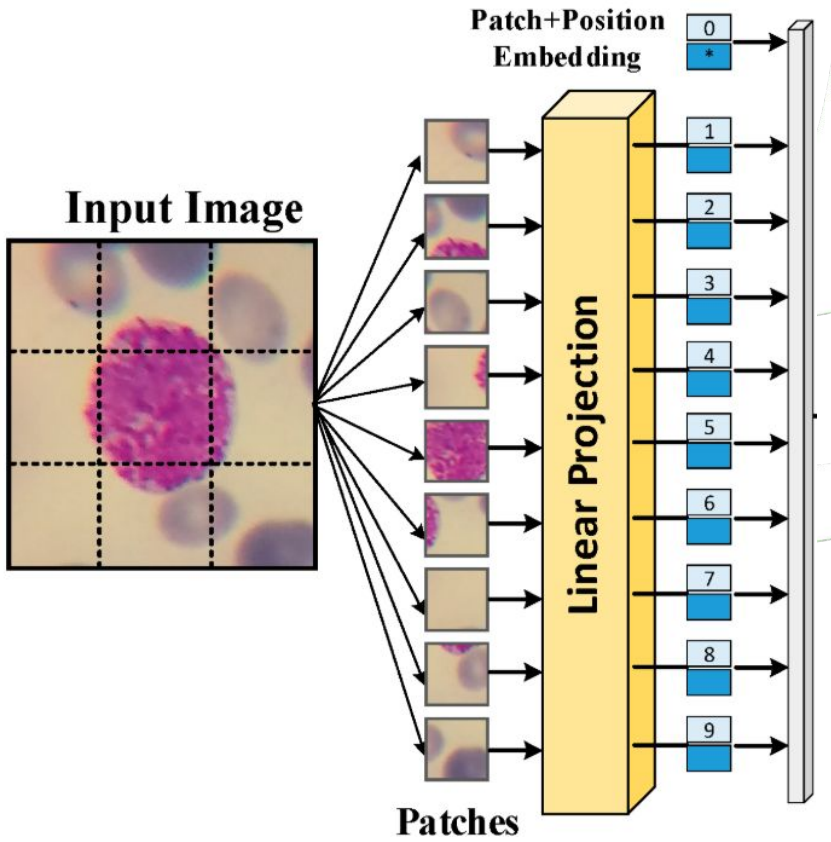


Transformers - positional encoding

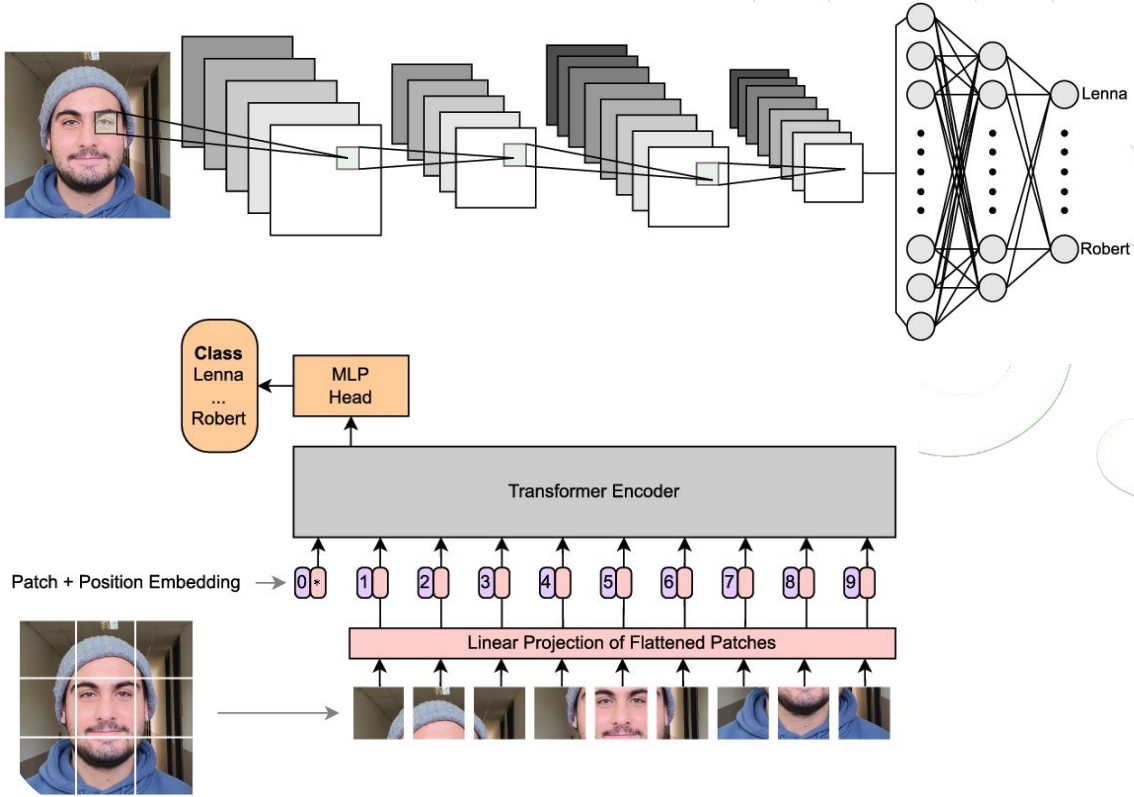


Vision Transformers

- We can borrow the Transformer architecture (that works on texts) for visual data!
- The “tokens” are no longer “words” but “patches” of the image
- Vision Transformer (ViT)



Vision Transformers



Brief recap

Why showing all these network “architectures”?

- Building a deep network from scratch is not easy → better to rely on “already tested” stuff
- Different architectures for different needs
  - VGGNet, ResNet, ViT → images
  - RNNs, Transformer → sequential data (text, sensor measurements, …)
- But there is also another important point: training DL models **costs**

Limitations

🧮 Training DL models is costly

- ⏱️ Time: Large models can take days or weeks to train
- ⚡ Electricity: Training requires lots of energy, especially for GPUs and TPUs
- 💾 Hardware: You need specialized resources (GPUs, clusters…)
- 💸 Money: All of the above adds up

Limitations

🧮 Training DL models is costly

🌍 Not accessible to everyone

- Especially a challenge in smaller labs, NGOs, or field-based environmental research

Limitations

🧮 Training DL models is costly

🌍 Not accessible to everyone

🌱 Environmental impact

- Training large models = large carbon footprint
  (Training GPT-3 reportedly used as much electricity as 120 U.S. homes in a month)

Limitations

🧮 Training DL models is costly

🌍 Not accessible to everyone

🌱 Environmental impact

🔍 Interpretability

- DL models are often seen as “black boxes”
- In many environmental applications, we don’t just want a prediction, we want to understand:
  - Why was this tree species selected?
  - What part of the drone image indicates a weed?
  - Which variables most influence yield predictions?

Limitations

- 🧮 Training DL models is costly
- 🌍 Not accessible to everyone
- 🌱 Environmental impact
- 🔍 Interpretability

Limitations - Energy consumption from DL training

Text-based DL models

Model	BERT finetune	BERT pretrain	6B Transf.
GPU	4·V100	8·V100	256·A100
Hours	6	36	192
kWh	3.1	37.3	13,812.4

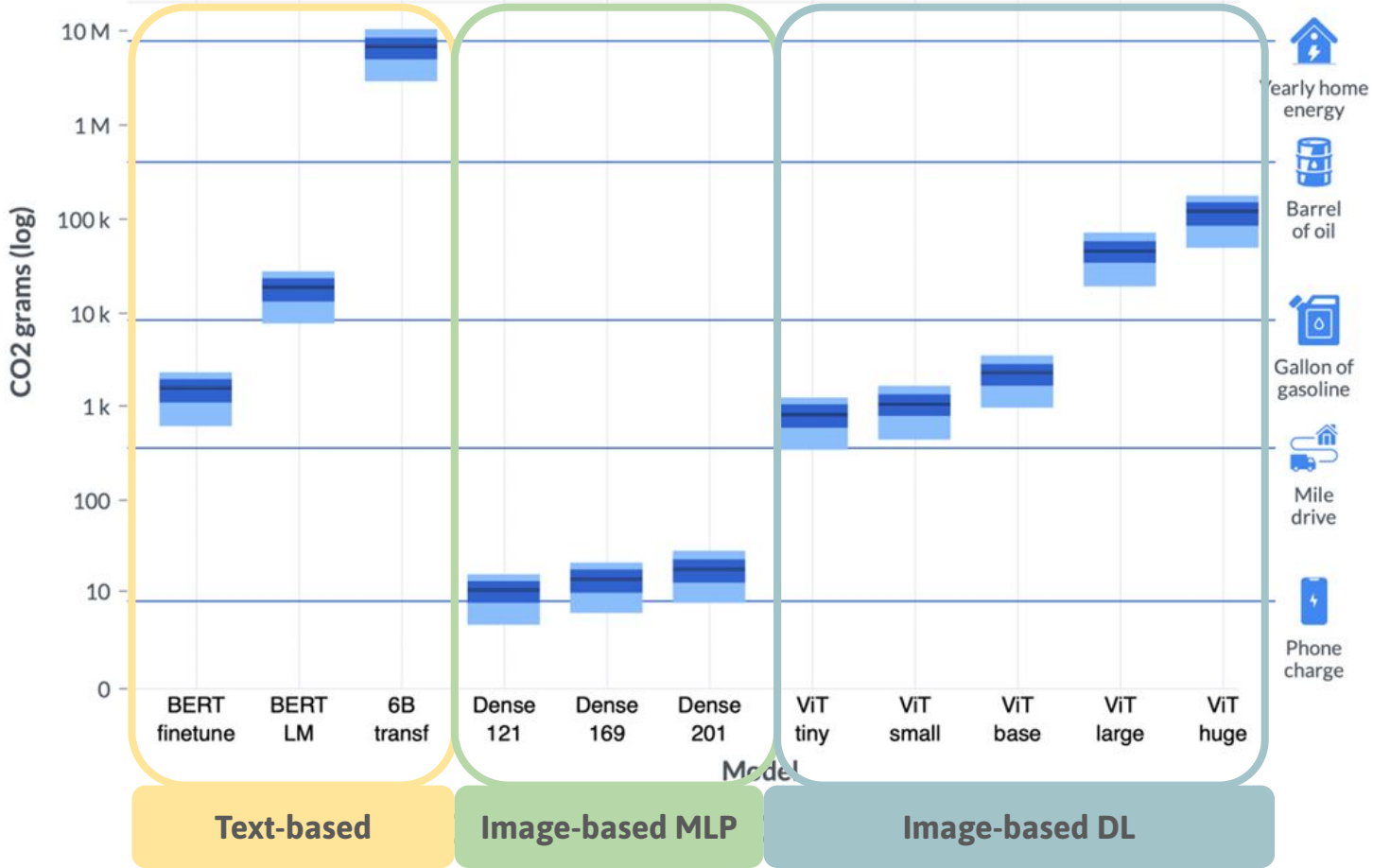
Image-based MLP models (Multilayer Perceptron)

Model	Dense 121	Dense 169	Dense 201
GPU	1·P40	1·P40	1·P40
Hours	0.3	0.3	0.4
kWh	0.02	0.03	0.04

Image-based DL models

Model	ViT Tiny	ViT Small	ViT Base	ViT Large	ViT Huge
GPU	1· V100	1·V100	1·V100	4·V100	4·V100
Hours	19	19	21	90	216
kWh	1.7	2.2	4.7	93.3	237.6

Limitations - Emissions from DL training



Transfer learning

- We train DL models on huge datasets like ImageNet (millions of images, 1000 classes).
- These models learn to detect edges, textures, shapes, object parts, etc.
- 💡 Turns out these low- and mid-level features are useful across many domains!

Transfer learning

How to Reuse Deep Models Without Training From Scratch
There are two main strategies:

- Feature Extraction
  - Use a pre-trained model to extract features
  - Train a small ML model (e.g., Random Forest, SVM, MLP) on your data
  - ✅ Cheap, fast, effective

- Fine-Tuning
  - Start from a pre-trained model
  - Retrain some layers (or all) on your specific dataset
  - ✅ More powerful but needs more data & compute

Pretraining - Where do I find pretrained models?

To reuse a model, you need:

✅ The architecture (code)

✅ The pretrained weights (on some big dataset)

Pretraining - Where do I find pretrained models?

To reuse a model, you need:

✅ The architecture (code)

✅ The pretrained weights (on some big dataset)

🖼️ For images

→ Look for models pretrained on **ImageNet** (1M images, 1k classes)

→ These models work well even for leaves, bugs, roots, landscapes...

📦 TorchVision makes this easy!

Pretraining - Where do I find pretrained models?

To reuse a model, you need:

✅ The architecture (code)

✅ The pretrained weights (on some big dataset)

Example: load a ResNet model, pretrained on ImageNet

```
from torchvision.models import resnet50
model = resnet50(weights="IMAGENET1K_V2")
```

Or a ViT model

```
from torchvision.models import vit_b_16
model = vit_b_16(weights="IMAGENET1K_V1")
```

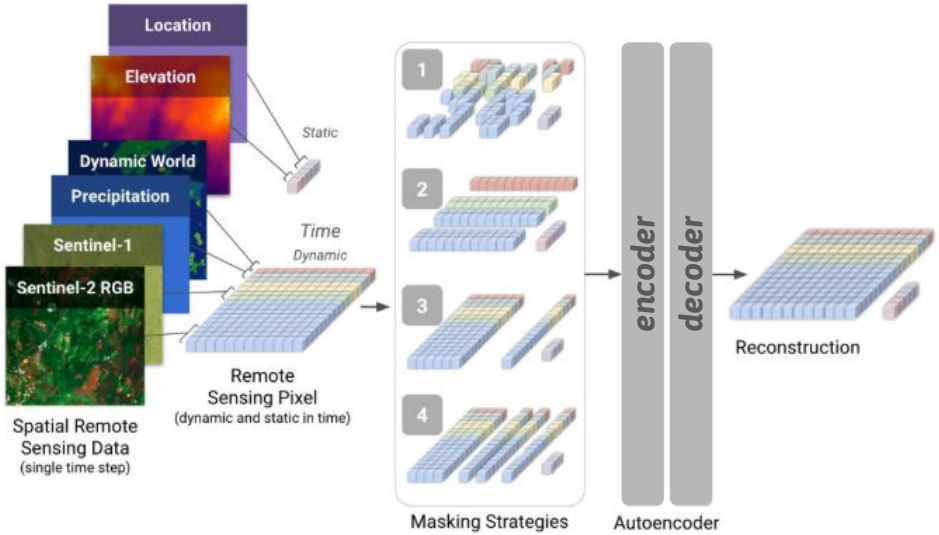
Pretraining - Where do I find pretrained models?

Platform	What It Offers	Notes
TorchVision	Common CNNs, ViTs pretrained on ImageNet	Easy to use, included in PyTorch ecosystem
TIMM	Huge collection of image models (CNNs, ViTs, hybrids)	
Hugging Face 🤗	Image, text, audio, video models	Especially good for NLP + general DL (based on PyTorch ecosystem)
OpenMMLab	Powerful models for object detection, segmentation	Often used in industrial vision
TensorFlow Hub	Pretrained models for many tasks	TensorFlow ecosystem

Pretraining - Where do I find pretrained models?

(randomly on github, other places, in paper...)

Pretraining - example



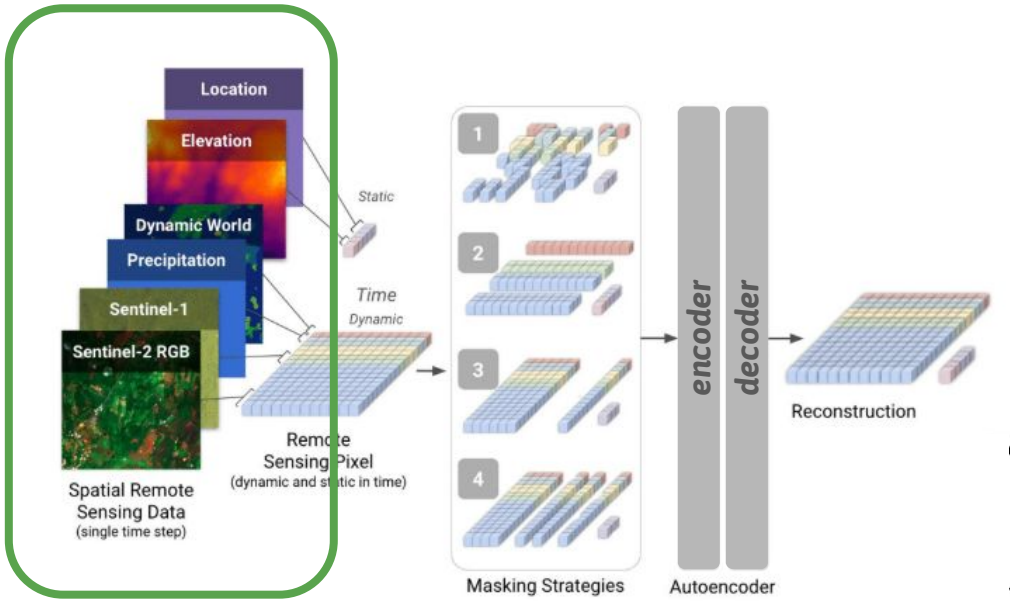
Presto

(Pretrained remote sensing transformer)

Lightweight, Pre-trained Transformers for Remote Sensing Timeseries

Gabriel Tseng[1,2] **Ruben Cartuyvels**[1,3] **Ivan Zvonkov**[4] **Mirali Purohit**[5]
David Rolnick[1,2] **Hannah Kerner**[5]
[1] Mila – Quebec AI Institute
[2] McGill University
[3] KU Leuven
[4] University of Maryland, College Park
[5] Arizona State University

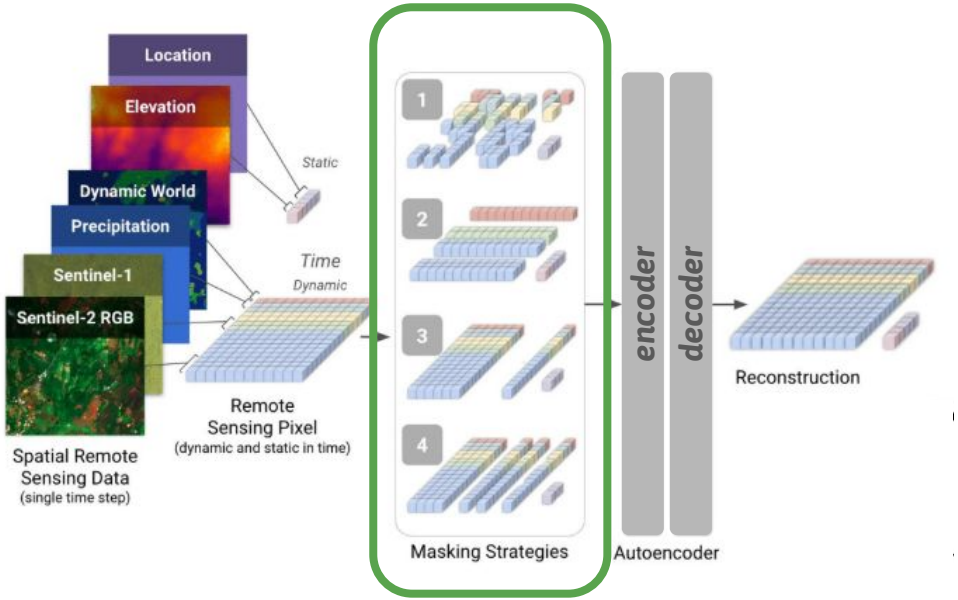
Pretraining - example



Pros of the model

- Uses a wide variety of input sensors at pre-training time

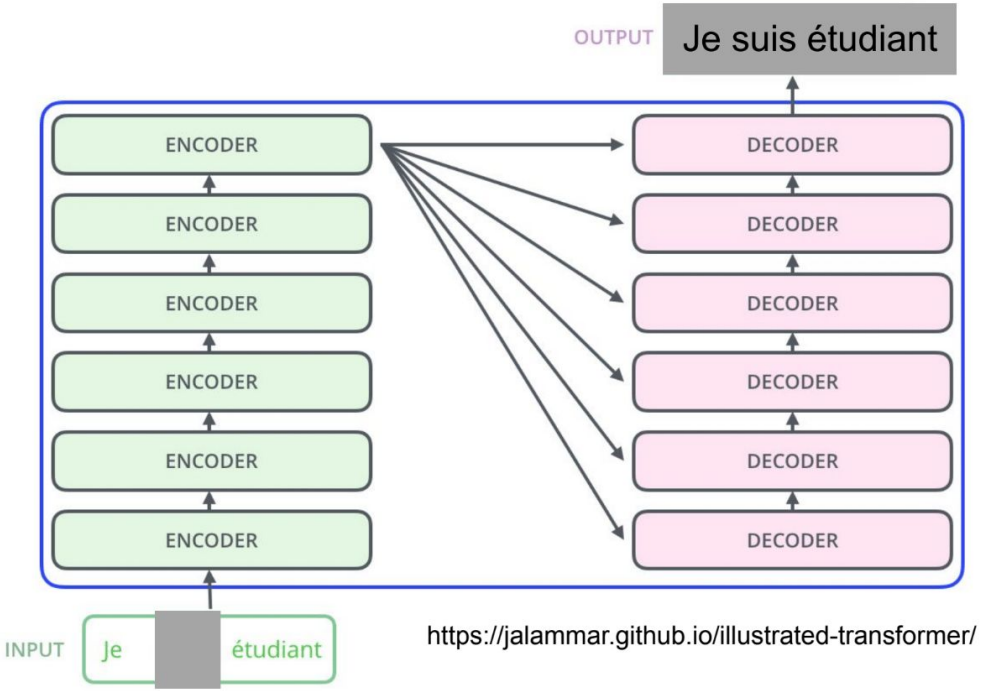
Pretraining - example



Pros of the model

- Uses a wide variety of input sensors at pre-training time
- Applies structured masking strategies so that the model can handle missing channels or timesteps

Pretraining - example

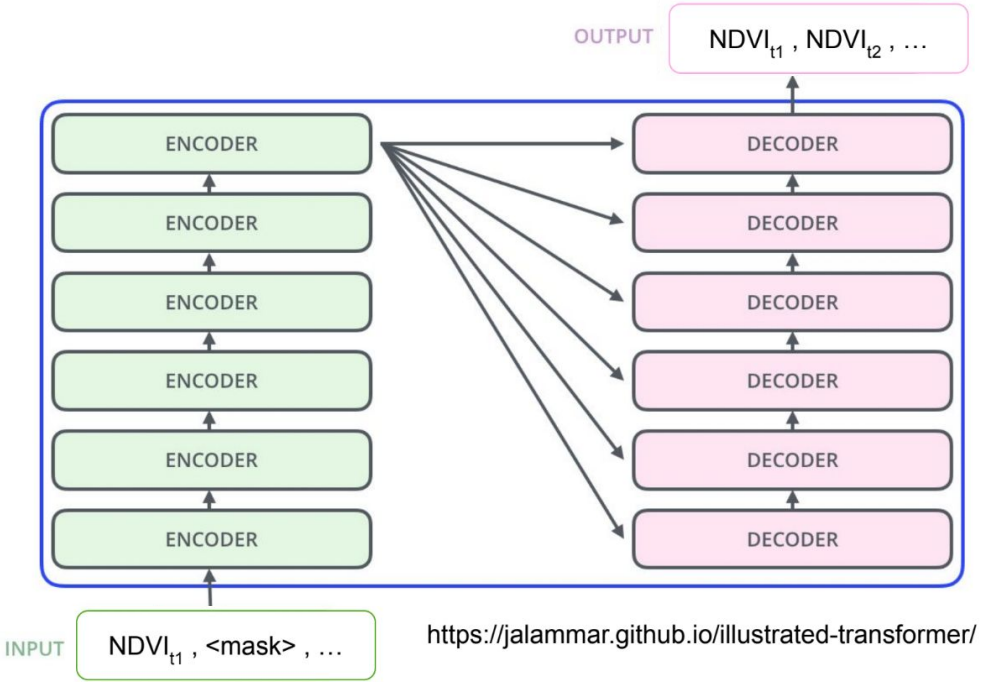


https://jalammar.github.io/illustrated-transformer/

Pros of the model

- Uses a wide variety of input sensors at pre-training time
- Applies structured masking strategies so that the model can handle missing channels or timesteps

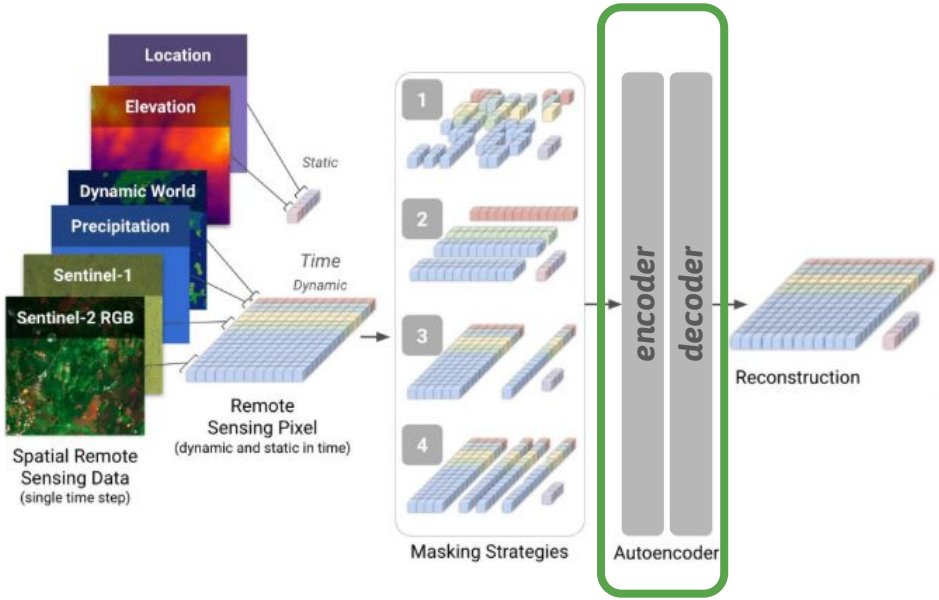
Pretraining - example



Pros of the model

- Uses a wide variety of input sensors at pre-training time
- Applies structured masking strategies so that the model can handle missing channels or timesteps

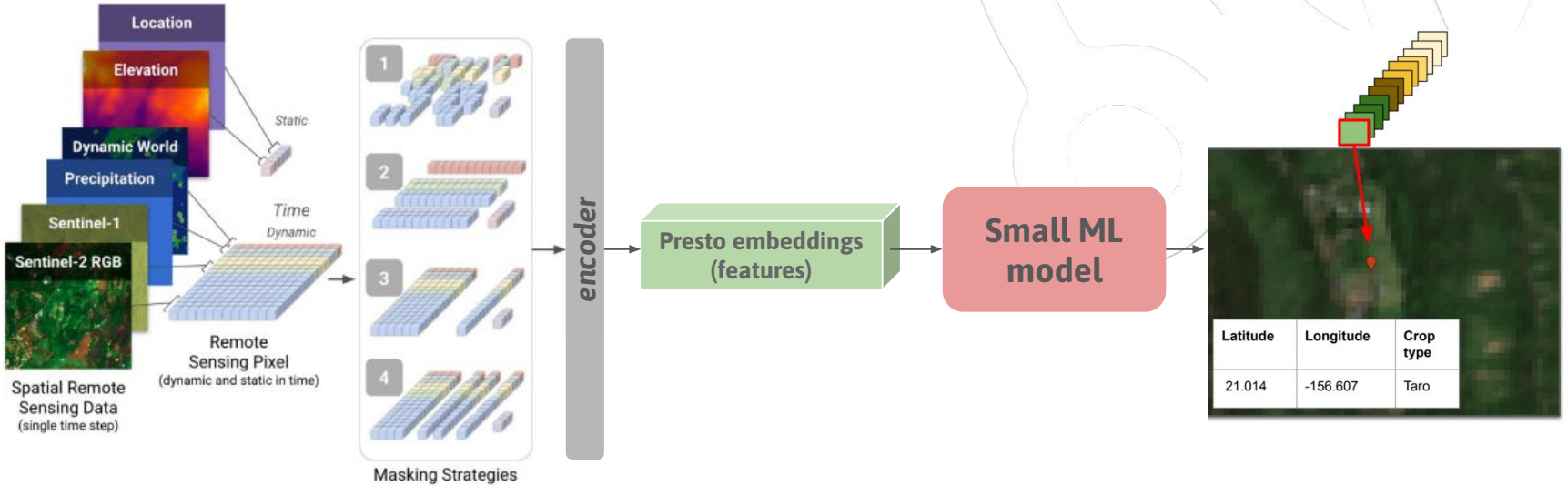
Pretraining - example



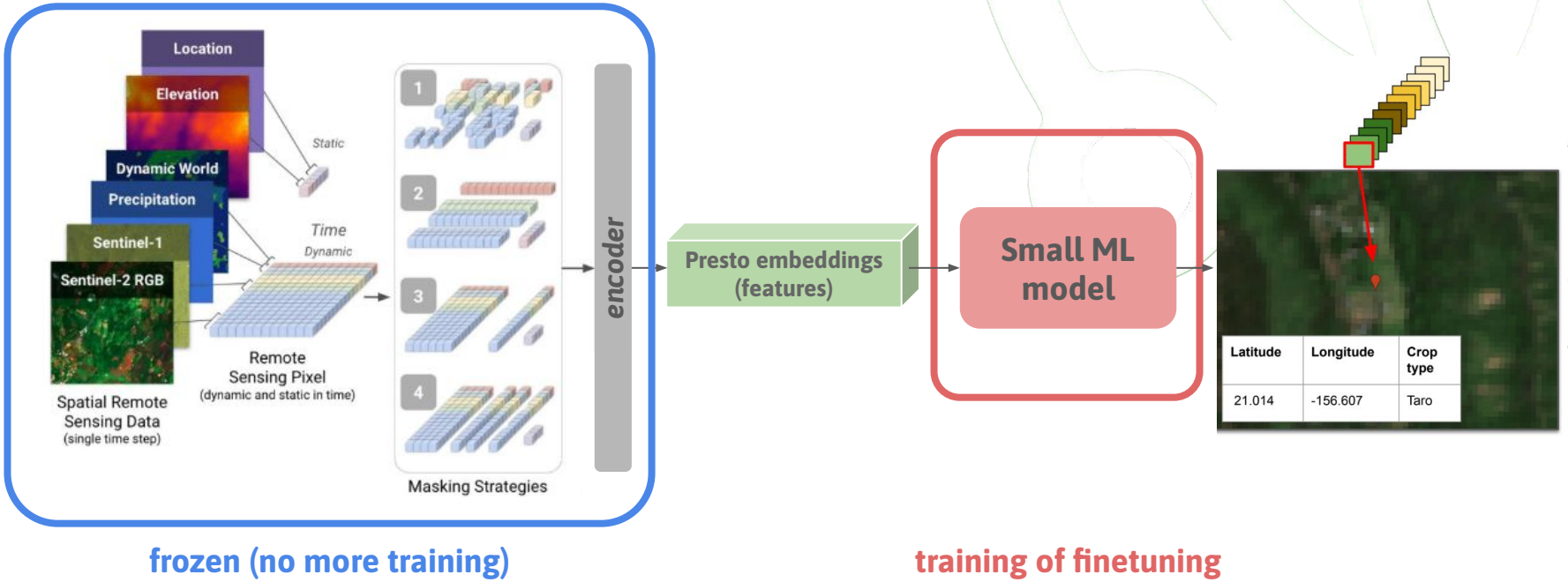
Pros of the model

- Uses a wide variety of input sensors at pre-training time
- Applies structured masking strategies so that the model can handle missing channels or timesteps
- The encoder takes *time series* as inputs instead of *images*.
  "Lighter" inputs → lighter model

Pretraining - example



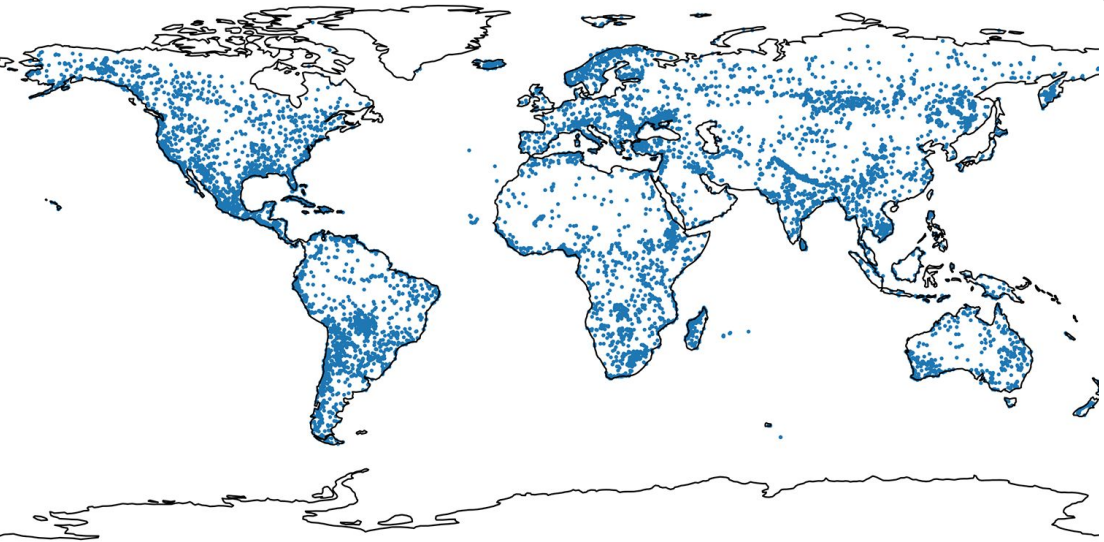
Pretraining - example



frozen (no more training)

training of finetuning

Pretraining - example



Pros of the model

- Global representation and diversity in pre-training data

Pretraining - example

Model	#. parameters Total	Adapted	Mean F1
Random Forest			0.441
MOSAIKS-1D$_R$	418K	8193	0.738
TIML	91K	91K	0.802
Presto$_R$	402K	129	0.835
no DW			**0.836**

Pros of the model

- Global representation and diversity in pre-training data
- Lightweight and quick to train (few parameters to train/adapt compared with other top models)

Pretraining - example

Model	Backbone	Params (M)	MegaFlops
SatMAE (RGB) (Cong et al., 2022)	ViT-Large	303.10	59,685.69
SatMAE (MS) (Cong et al., 2022)	ViT-Large	305.96	535,515.25
ScaleMAE (Reed et al., 2022)	ViT-Large	303.10	59,685.69
ConvMAE (Gao et al., 2022)	ConvMAE-Large	88.78	23,315.58
SeCo (Manas et al., 2021)	ResNet-18	11.69	149.37
GASSL (Ayush et al., 2021)	ResNet-18	11.69	149.37
Presto RGB pixel (image)	Presto	0.40	0.79 (3,235.84)
Presto MS pixel (image)	Presto	0.40	2.37 (9,707.52)

Space requirement

Time requirement

Pretraining - example

```
# either import works. The single_file_presto has no load_pretrained function, since this
# requires knowing where the pretrained file is. The state dict can be loaded directly
# from data/default_models.pt
from single_file_presto import Presto
from presto import Presto

# to make a randomly initialized encoder-decoder model
encoder_decoder = Presto.construct()
# alternatively, the pre-trained model can also be loaded
encoder_decoder = Presto.load_pretrained()

# to isolate the encoder
encoder_only = encoder_decoder.encoder
# to add a linear transformation to the encoder's output for finetuning
finetuning_model = encoder_decoder.construct_finetuning_model(num_outputs=1, regression=True)
```

Pros of the model

- Global representation and diversity in pre-training data
- Lightweight and quick to train
- Takes 3.17 MB of disk space
- You can finetune it in 5-10 minutes on a 2017 Macbook Pro (on CPU)
- Easy to load into python (I didn't try, but it looks easy)

Pretraining - example

Using Presto for downstream tasks

The purpose of this notebook is to demonstrate how Presto (and utility functions in the Presto package) can be used for downstream tasks.

To demonsrate the usefulness of Presto even when the input looks very different then what Presto was pre-trained on, we will consider tree-type mapping using single-timestep images.

To do this, we will use the TreeSat benchmark dataset. This tutorial requires the S2 data to be downloaded from Zenodo and unzipped in the treesat folder.

In [1]:
```
import xarray
from pyproj import Transformer
import numpy as np
from scipy import stats
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score

from tqdm import tqdm

import torch
from torch.utils.data import DataLoader, TensorDataset

import presto

# this is to silence the xarray deprecation warning.
# Our version of xarray is pinned, but we'll need to fix this
# when we upgrade
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
/Users/gabrieltseng/anaconda3/envs/lem/lib/python3.9/site-packages/geopandas/_compat.py:106: UserWarning: The Shapely GEOS version (3.11.1-CAPI-1.17.1) is incompatible with the GEOS version PyGEOS was compiled with (3.10.4-CAPI-1.16.2). Conversions between both will be slow.
  warnings.warn(
```

If the TreeSat data has been correctly downloaded from Zenodo (see the Markdown cell above), these assert statements should pass.

In [2]:
```
treesat_folder = presto.utils.data_dir / "treesat"
assert treesat_folder.exists()

# this folder should exist once the s2 file from zenodo has been unzipped
s2_data_60m = treesat_folder / "s2/60m"
assert s2_data_60m.exists()
```

For simplicity, we will only consider classification between 2 tree species: Abies_alba and Acer_pseudoplatanus.

The TreeSatAI S2 data contains the following bands: ["B2", "B3", "B4", "B8", "B5", "B6", "B7", "B8A", "B11", "B12", "B1", "B9"]

In [3]:
```
TREESATAI_S2_BANDS = ["B2", "B3", "B4", "B8", "B5", "B6", "B7", "B8A", "B11", "B1

SPECIES = ["Abies_alba", "Acer_pseudoplatanus"]
```

https://github.com/nasaharvest/presto/blob/main/downstream_task_demo.ipynb

Pros of the model

- Global representation and diversity in pre-training data
- Lightweight and quick to train
- Takes 3.17 MB of disk space
- You can finetune it in 5-10 minutes on a 2017 Macbook Pro (on CPU)
- Easy to load into python (I didn't try, but it looks easy)

Limitations

🧮 Training DL models is costly

🌍 Not accessible to everyone

🌱 Environmental impact

🔍 Interpretability

Limitations

🧮 Training DL models is costly

🌍 Not accessible to everyone

🌱 Environmental impact

🔍 Interpretability

Interpretability

✅ Workarounds exist:

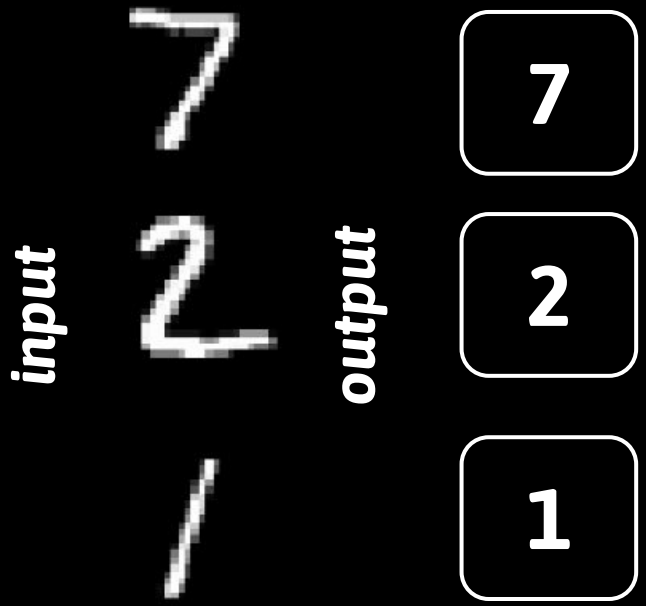
- Visualization tools (e.g., saliency maps for images)
- Post-hoc explainability methods (e.g., SHAP, LIME)

Let’s see an example of interpretability with SHAP

Interpretability

Why?

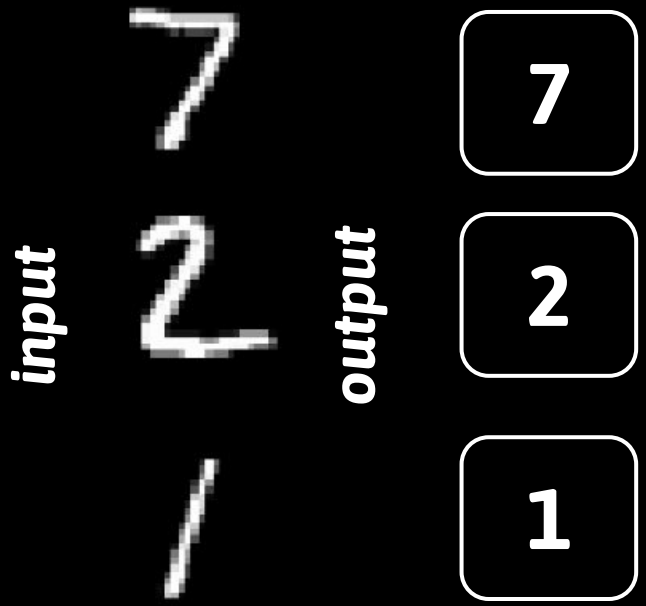
(Why that number and not another one?)



Interpretability

Why?

(Why that number and not another one?)



Using an explainability method (SHAP) we ask the model

“Is it a 0?”

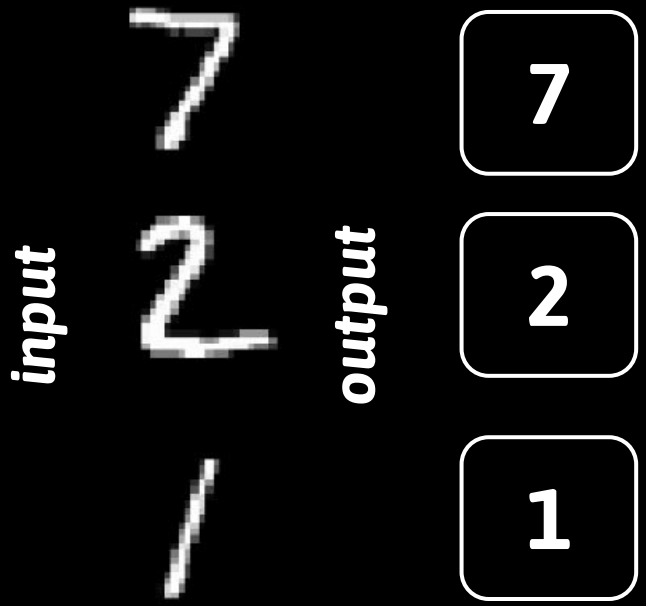
Blue pixels are why probably “no”

Red pixels are why probably “yes”

Interpretability

Why?

(Why that number and not another one?)



Using an explainability method (SHAP) we ask the model

“Is it a 1?”

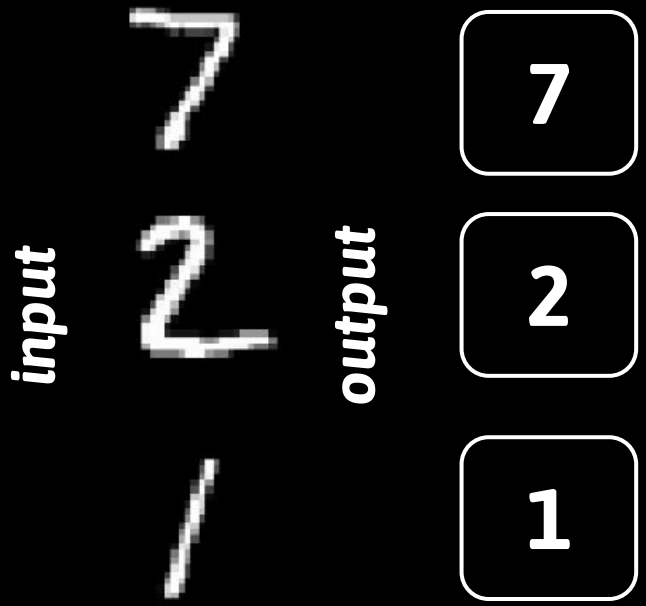
Blue pixels are why probably “no”

Red pixels are why probably “yes”

Interpretability

Why?

(Why that number and not another one?)



Using an explainability method (SHAP) we ask the model

“Is it a 2?”

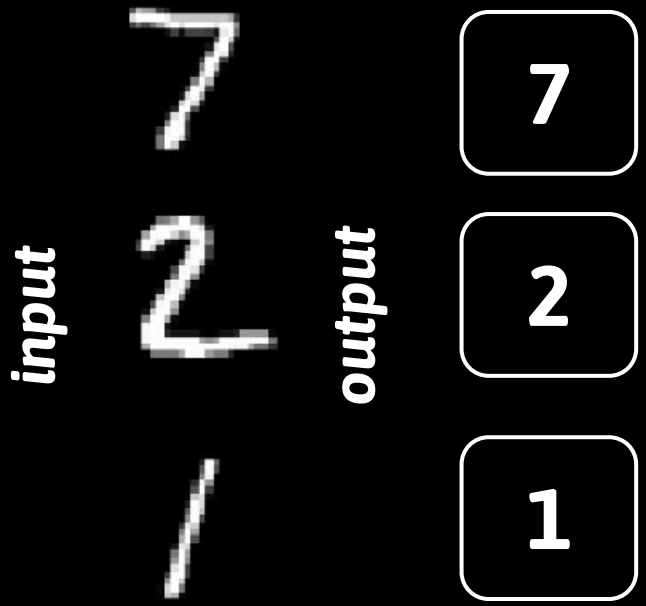
Blue pixels are why probably “no”

Red pixels are why probably “yes”

Interpretability

Why?

(Why that number and not another one?)



Using an explainability method (SHAP) we ask the model

“Is it a 3?”

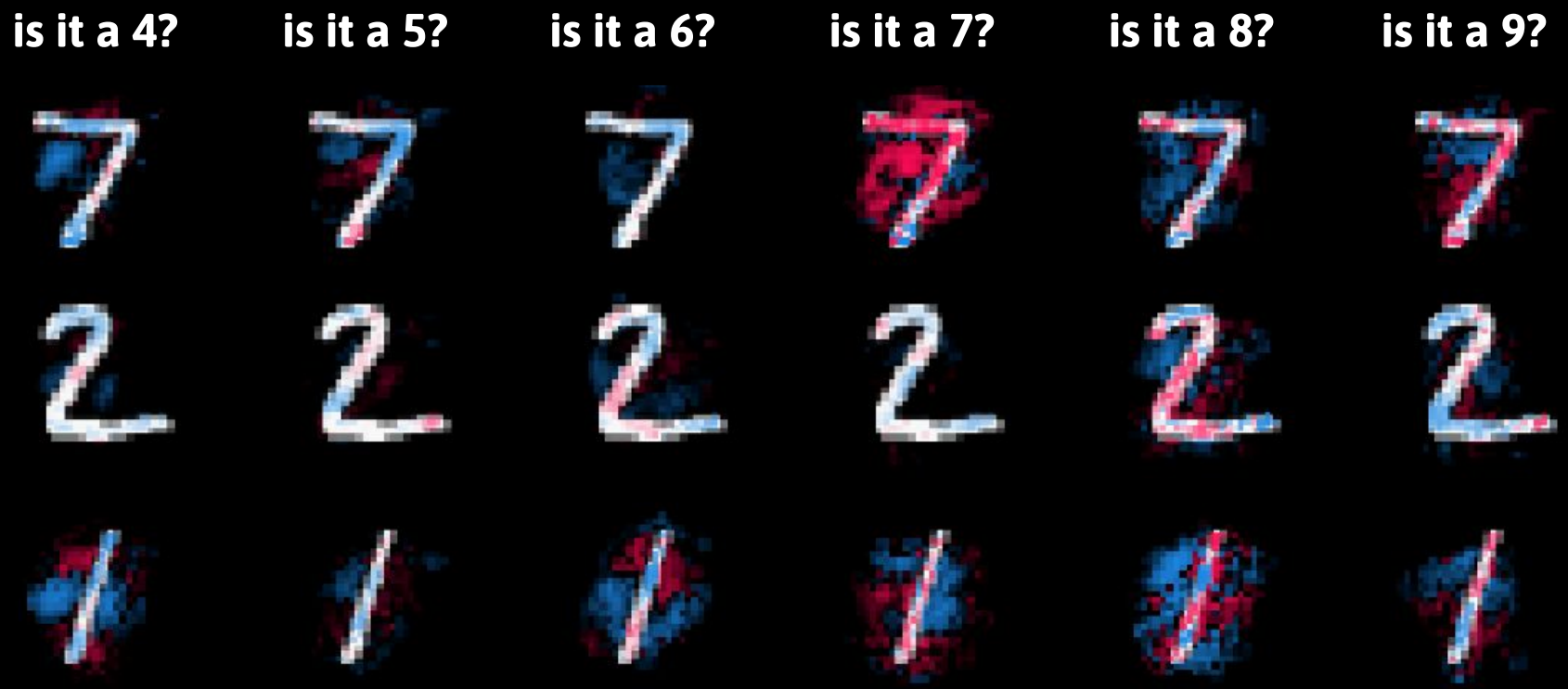
Blue pixels are why probably “no”

Red pixels are why probably “yes”

Interpretability

Why?

(Why that number and not another one?)



Final recap

- Building a deep network from scratch is not easy → better to rely on “already tested” stuff
- Different architectures for different needs
  - VGGNet, ResNet, ViT → images
  - RNNs, Transformer → sequential data (text, sensor measurements, …)
- Keep in mind the limitations and drawbacks of using Deep Learning methods
  - Costs, time, environmental footprint, interpretability

THANKS!

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani