Deep Learning for Human Action Recognition

And shoplifting detection application

Course prerequisites

- Master the python programming language.
- Have a good understanding of AI & PyTorch.
- Have access to google colab (or a good GPU).
- Download the data packages before the labs (see the course's github).

Course objectives

- Master deep learning basics for computer vision
- Know the state of the art
  - Know & Understand state of the art architectures for action recognition.
  - Get expertise on the subject.
- Be able to gather and format data
- Use efficient architectures & set your own shoplifting detector

Course layout

1. Computer vision & action recognition theory : ~2 hours
   - 3D CNNs, ResNet, SlowFast, …
2. Experimenting with action recognition (Lab) : ~2 hours
   - Gathering data, test inference, …
3. Shoplifting detection theory : data, fine-tuning, limits : ~2 hours
   - Video data, training processes comparison
4. Set an efficient shoplifting detector (Lab) : ~2 hours

Prerequisites : Google colab access (or fast GPU access)

Before starting

- This course contains labs where students shall get proven hands-on experience by the end of this course.
- Please make sure you prepare yourself well for the first lab after the first lecture (see prerequisites and github).

About the lecturer

(me)

Computer vision & action recognition theory

Computer vision : problem statement

- Our goal is to **categorize image data** using a computer program

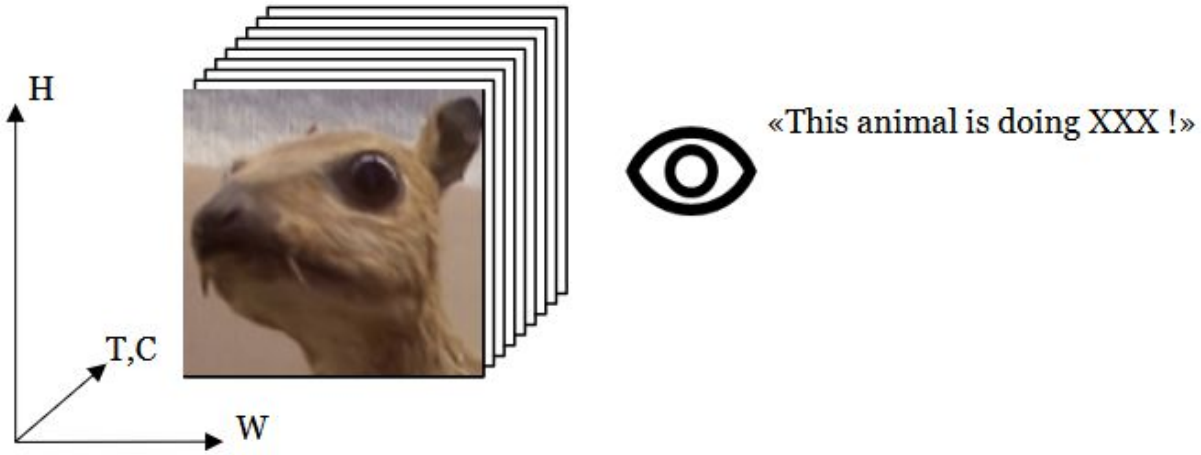
Computer vision & action recognition theory

Reminders and deep dive into existing architectures

Computer vision : problem statement

- Our goal is to **categorize image data** using a computer program

Action recognition : problem statement

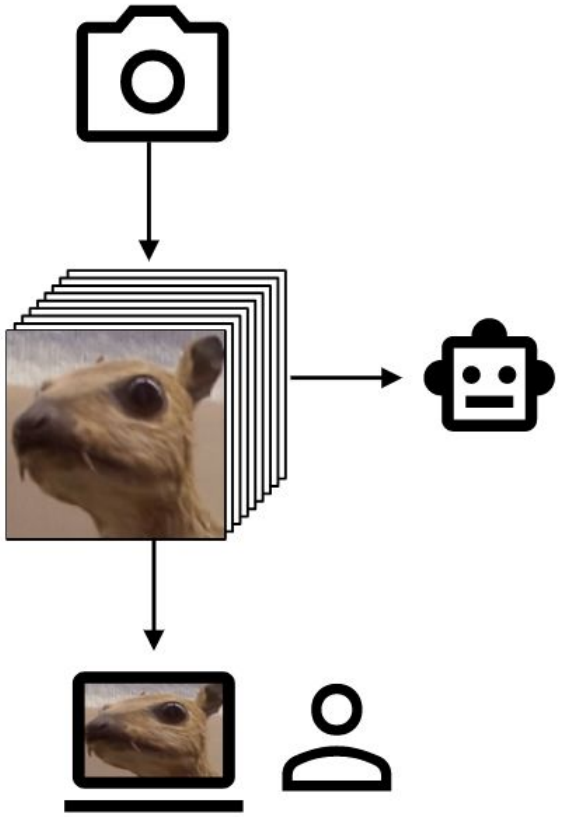


- In this course, we focus on **action recognition**, meaning we need to take **time** into account (video data)

- But how ?
  - *How does a computer pick on an images **features** ?*
  - *How do we take **time** into account ?*
  - *How does a computer **learn** these **features** ?*

Computer vision & action recognition theory

Solution requirement :



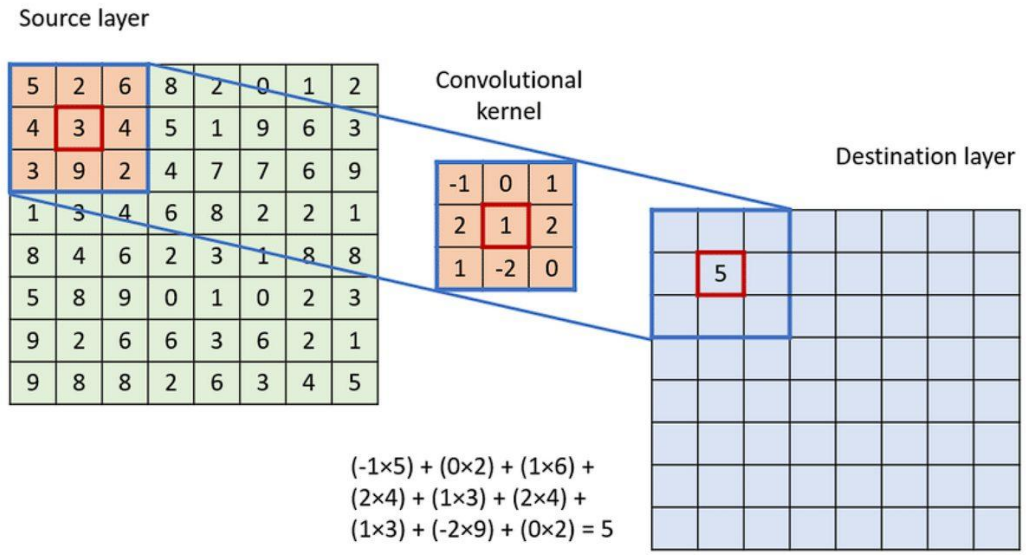
Usually, we use human intervention for action recognition

Goal : Get the data, classify, done !

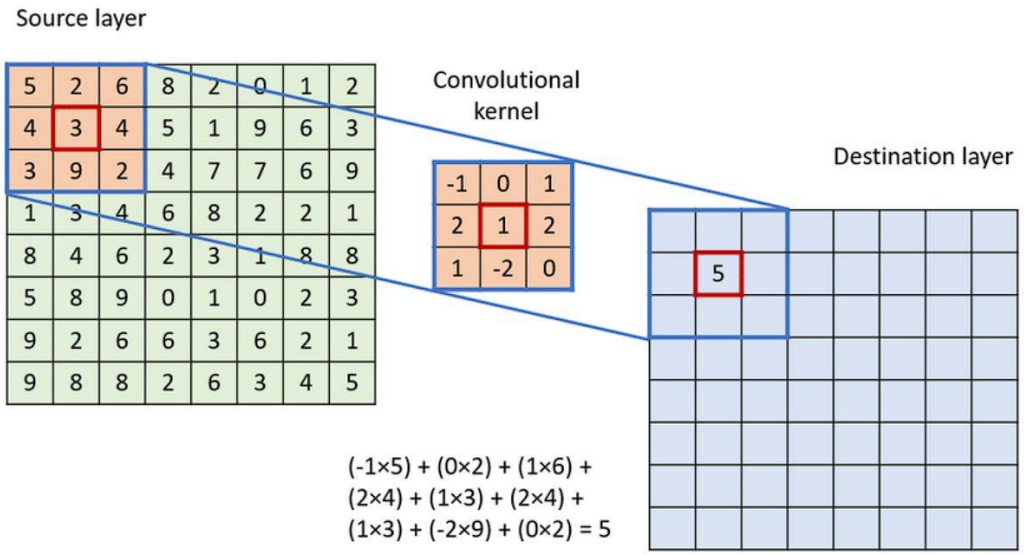
Introducing : The convolution

$$Output = Kernel * Input$$

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(n-m)g(m)$$

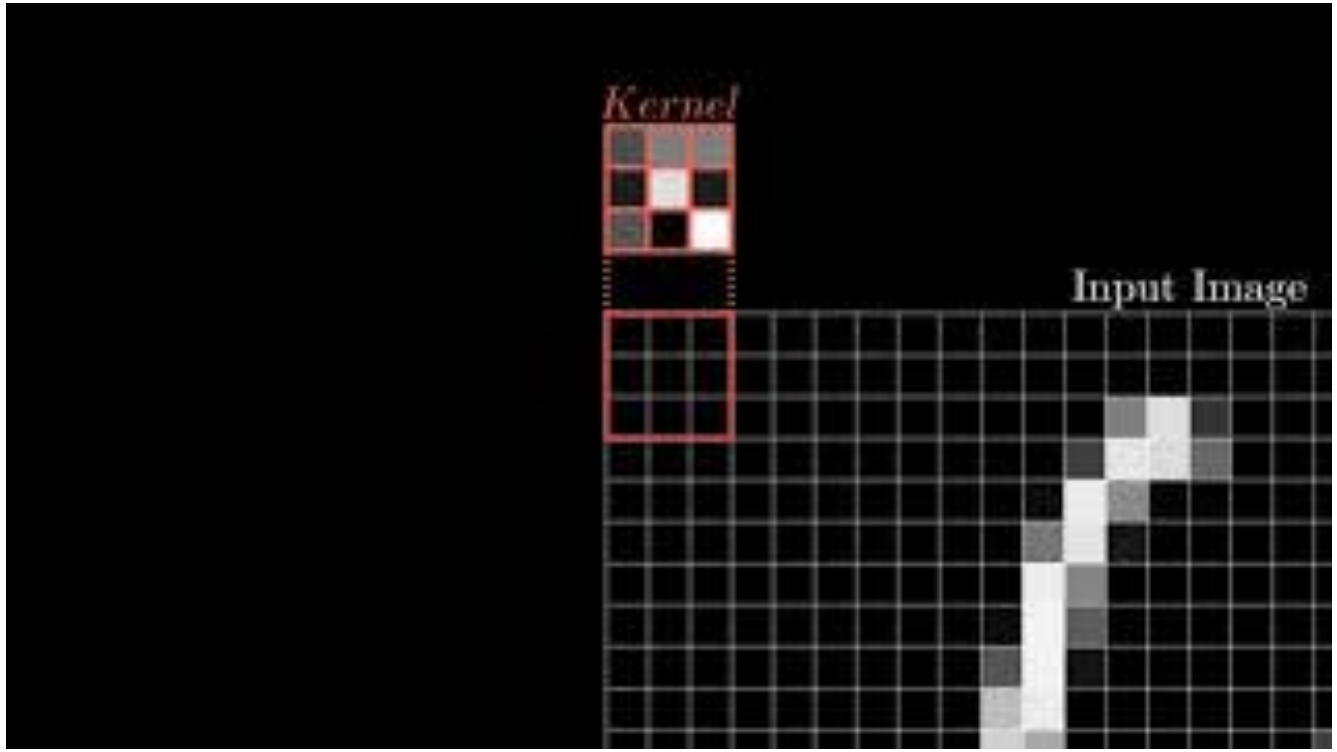


The convolution



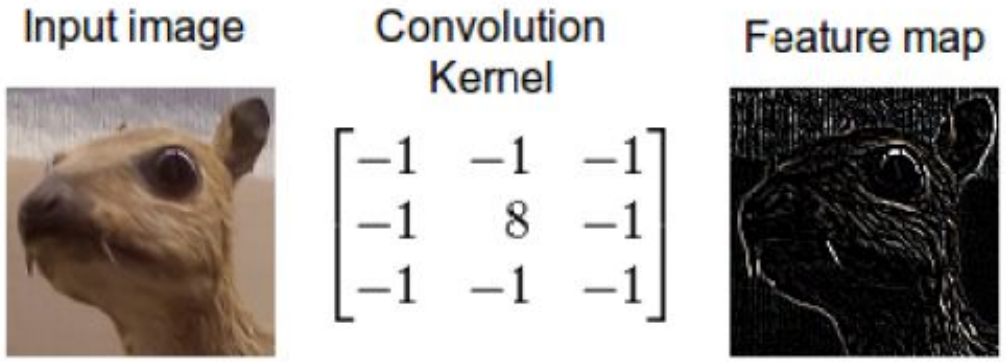
- Computationally intensive.
  - So **<u>why</u>** use convolution then ?

Features extraction



The computer extracts « features » from an image using convolution

Features extraction

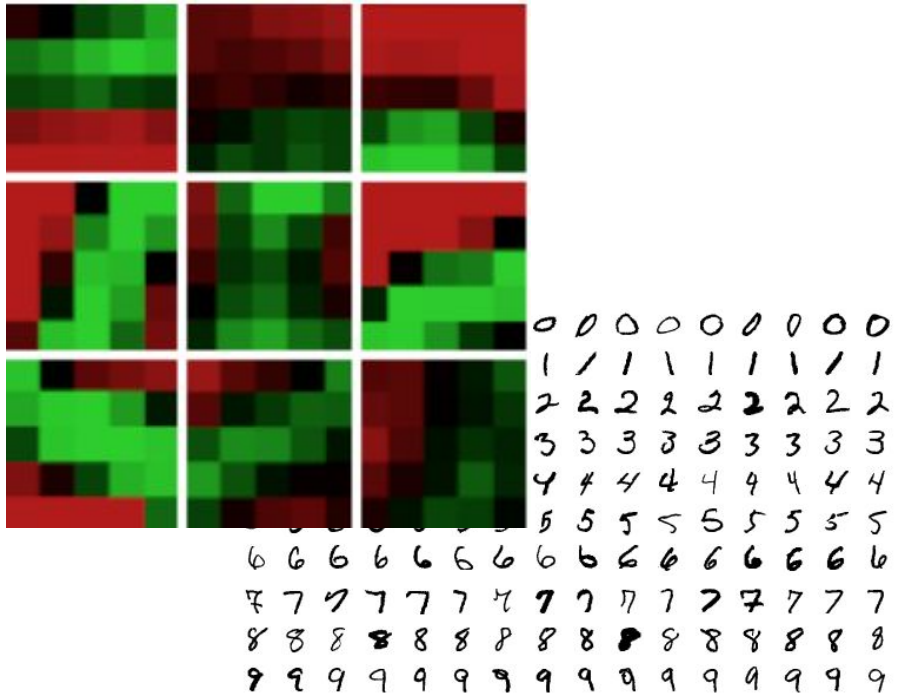


In a nutshell :

Image representation => Abstract representation

Computer vision & action recognition theory

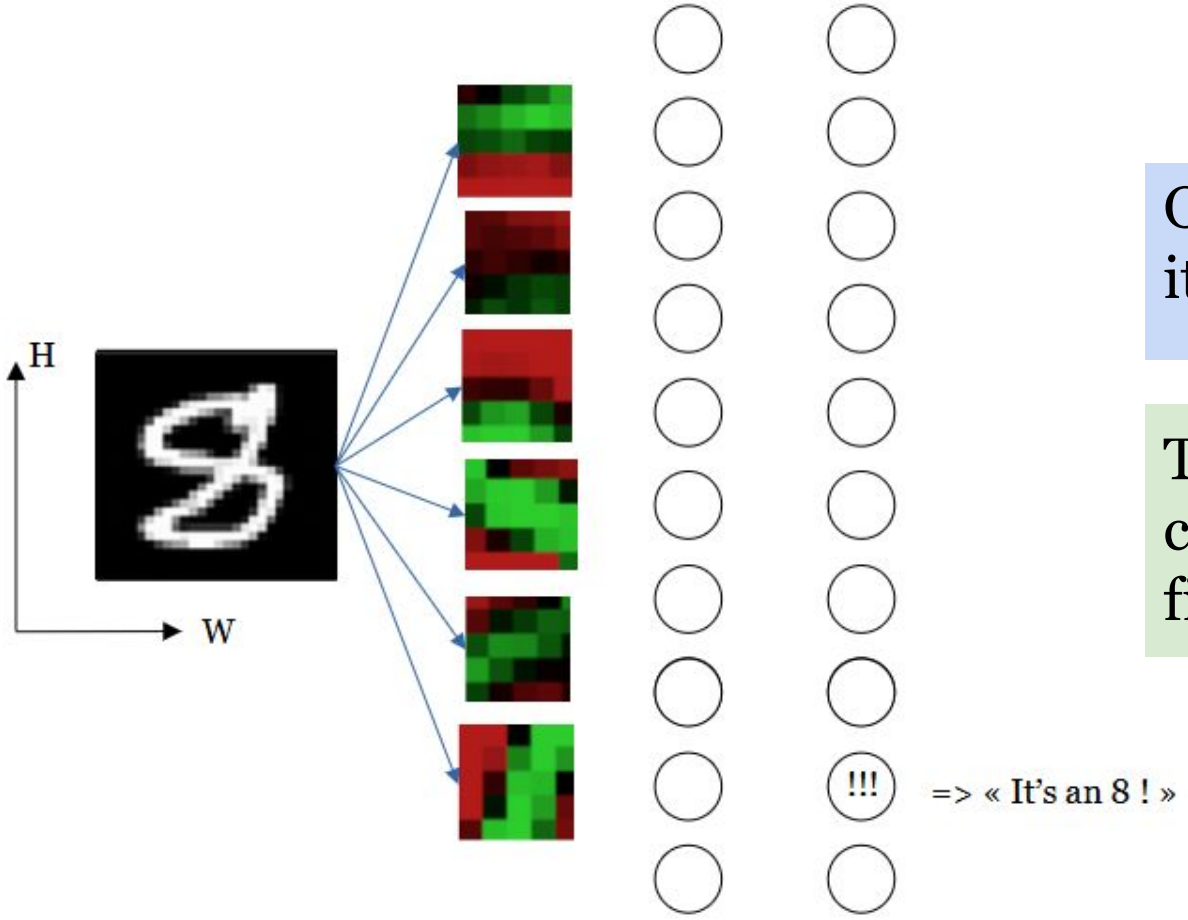
Features extraction



The optimisation algorithm **learns** optimal filters (*kernels*).

These are example filters learnt this classify numbers (MNIST)

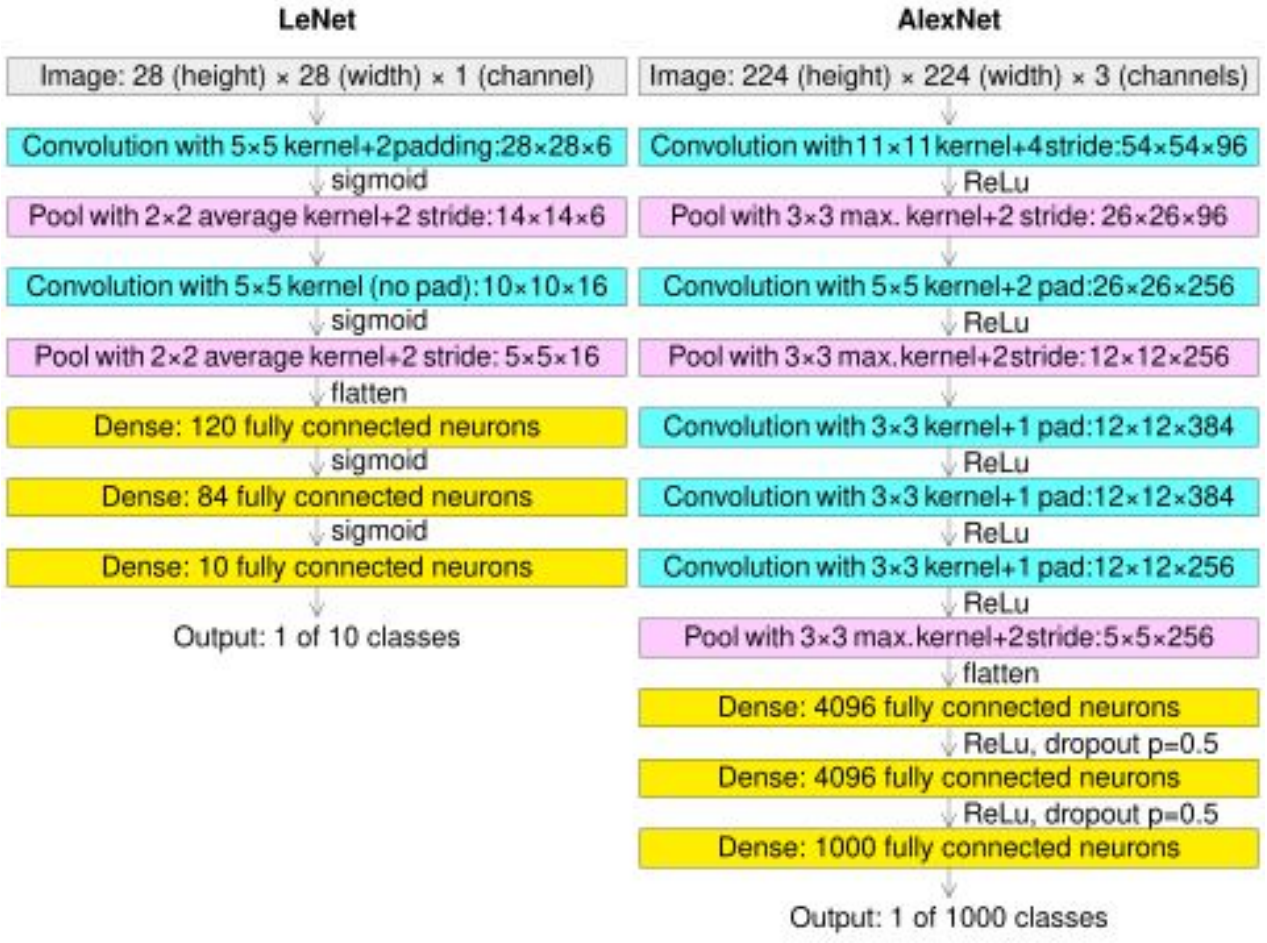
Features classification



Once the computer extracted features, it needs to classify them.

To do so, we use a couple of fully connected layer. We can stack these for fine-tuning applications.

Convolution networks examples



Each architecture is different and is optimized for a specific application.

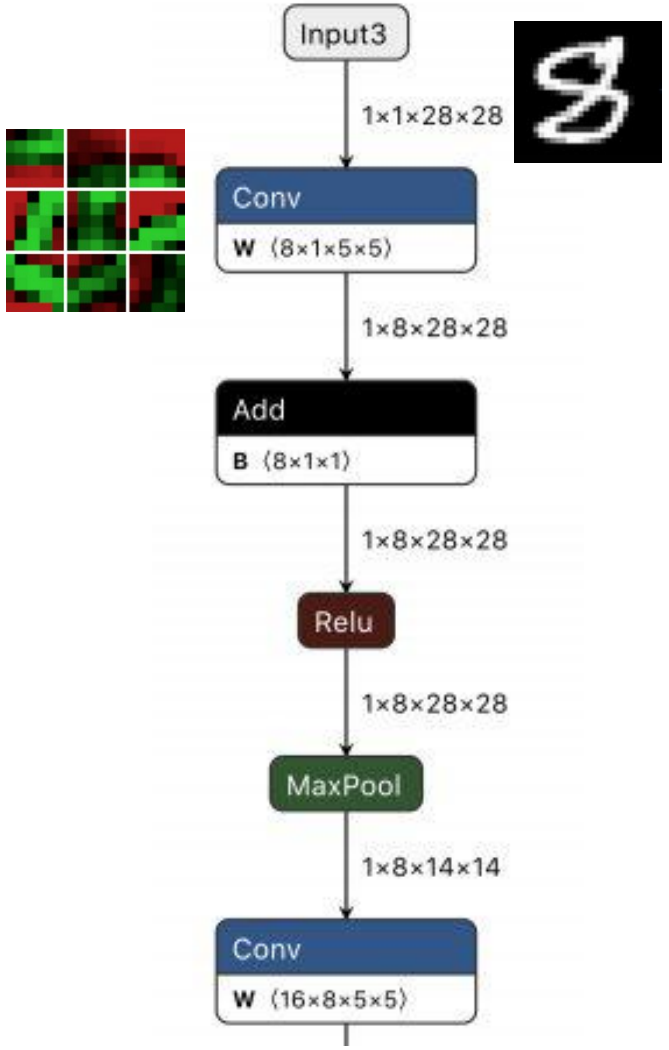
When a model is performant (innovative), it can have its own name (e.g. “AlexNet” for image classification)

Example usage of convolution : ONNX

ONNX

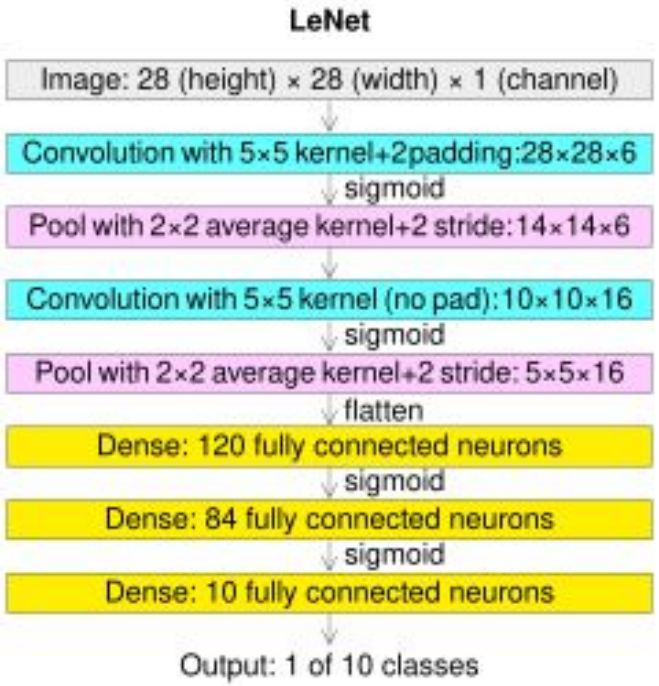
ONNX is a great tool to visualize a network though its dimensions

It also is a binary format to translate a model across different frameworks

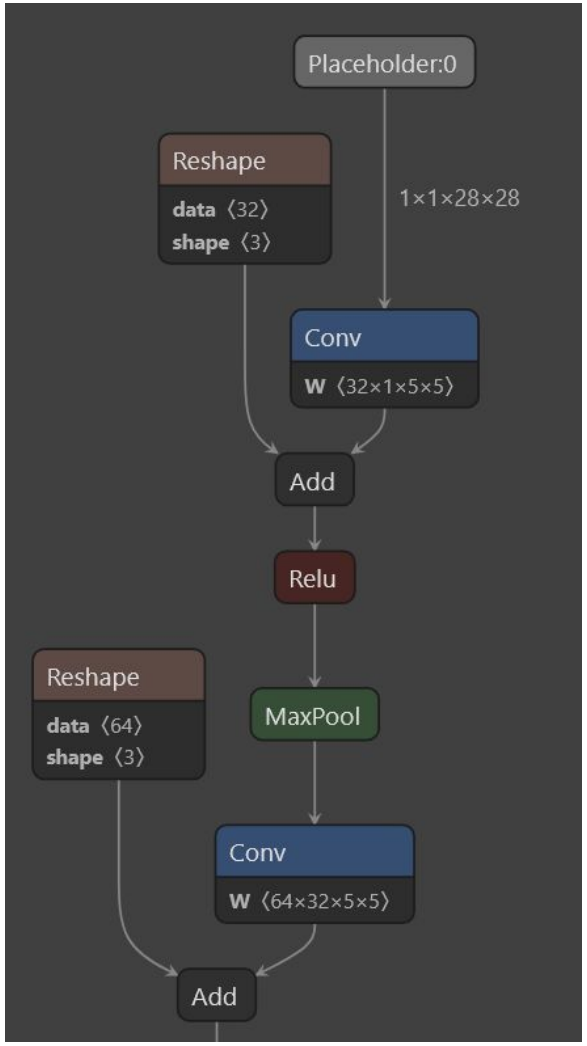


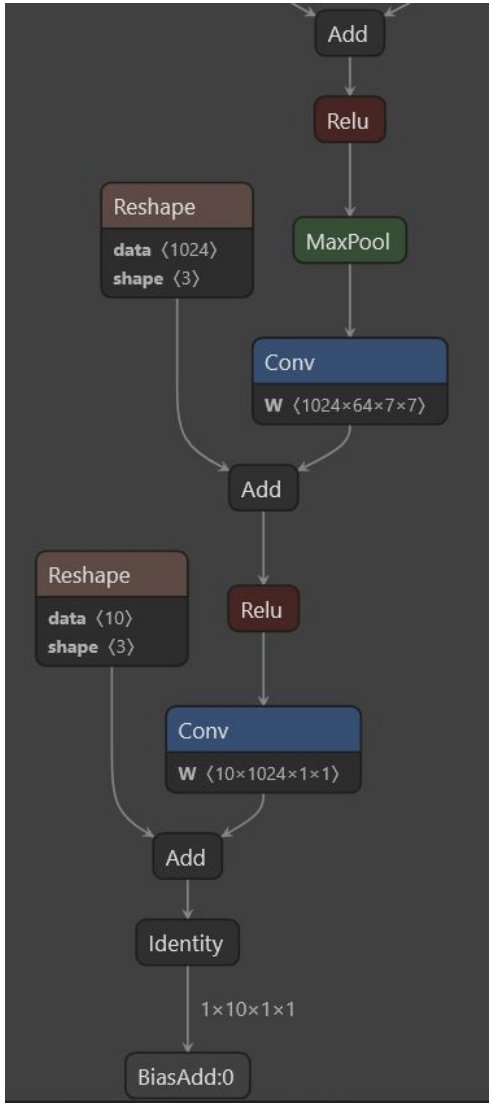
Computer vision & action recognition theory

Example usage : LeNet (1989-1995)



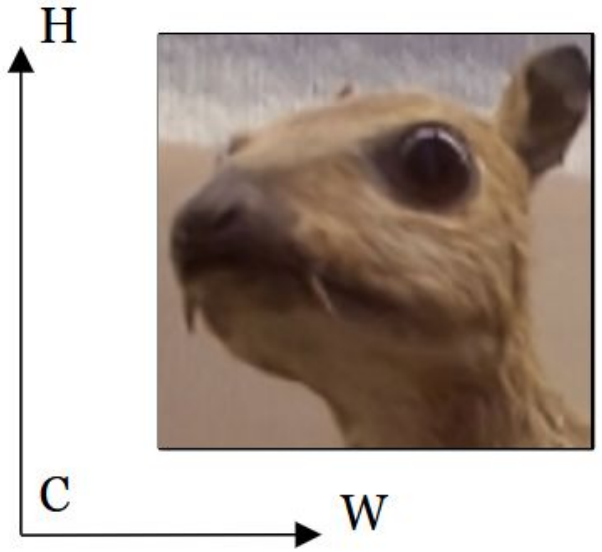
≈





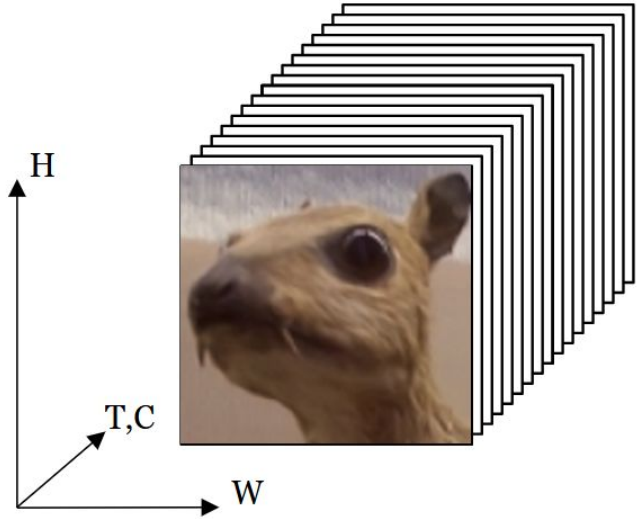
Adding dimensions …

- When dealing with videos, we also want to also **extract temporal features**



- Simple image :

« Spatial features » only



- Video :

New temporal dimension. Carries its own informations, has its own « **temporal features** »

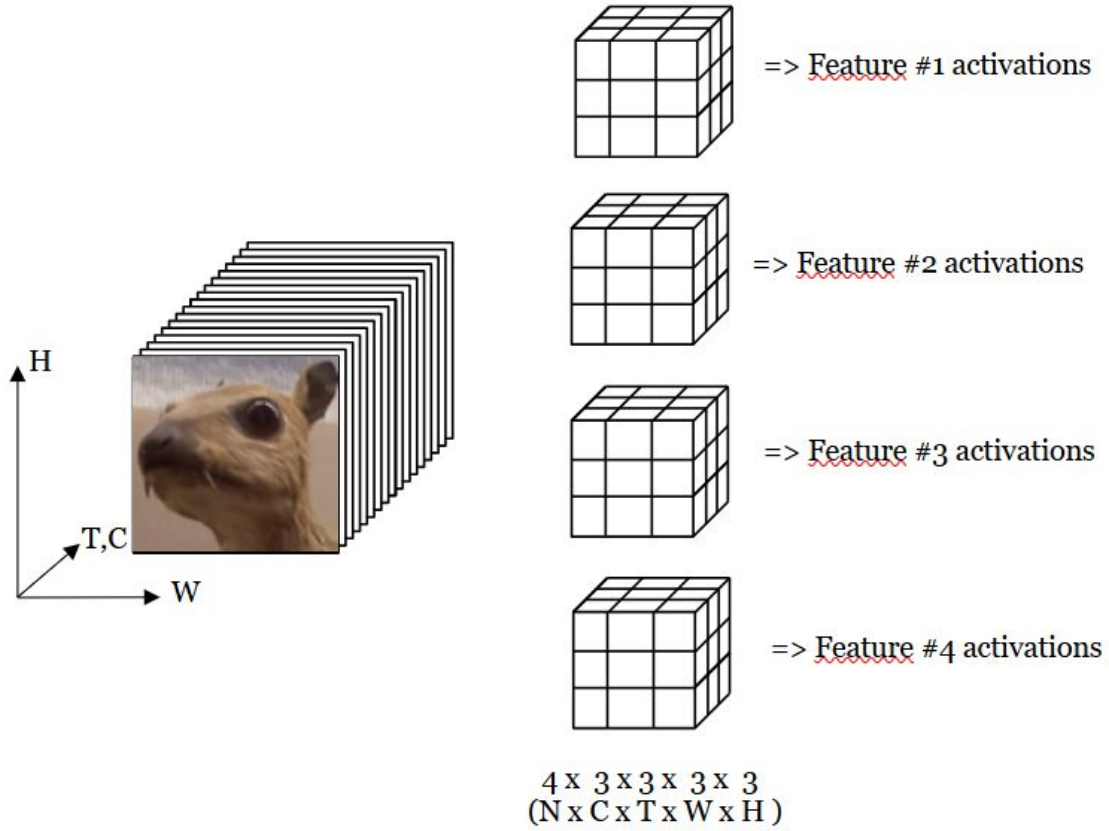
Extracting temporal features

Solution #1 : 3D CNN

In this solution, we add a dimension to the kernels, dedicated to extract T features.

\+ Precision

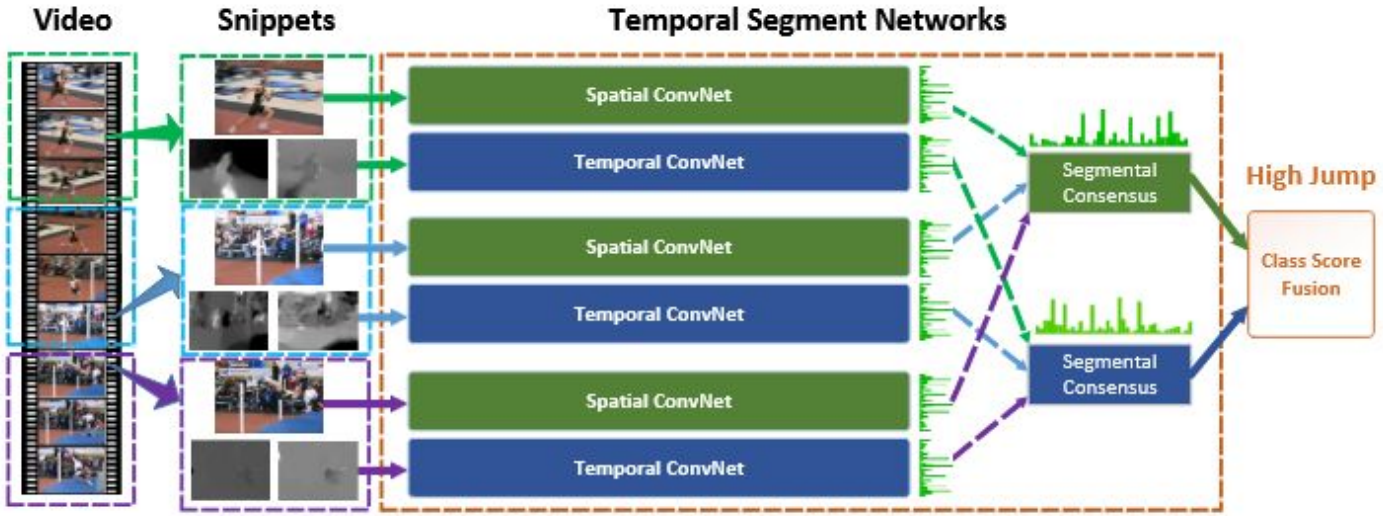
\- Computation cost



Extracting temporal features

Solution #2 : Dividing & Sampling

- Simply divide the video into segments and sample 1 snippet.
  - The snippet can be a frame, 10 frames etc.
  - The end goal is to have an overview of the whole video by extracting key difference in space and time instead of processing everything



Extracting temporal features

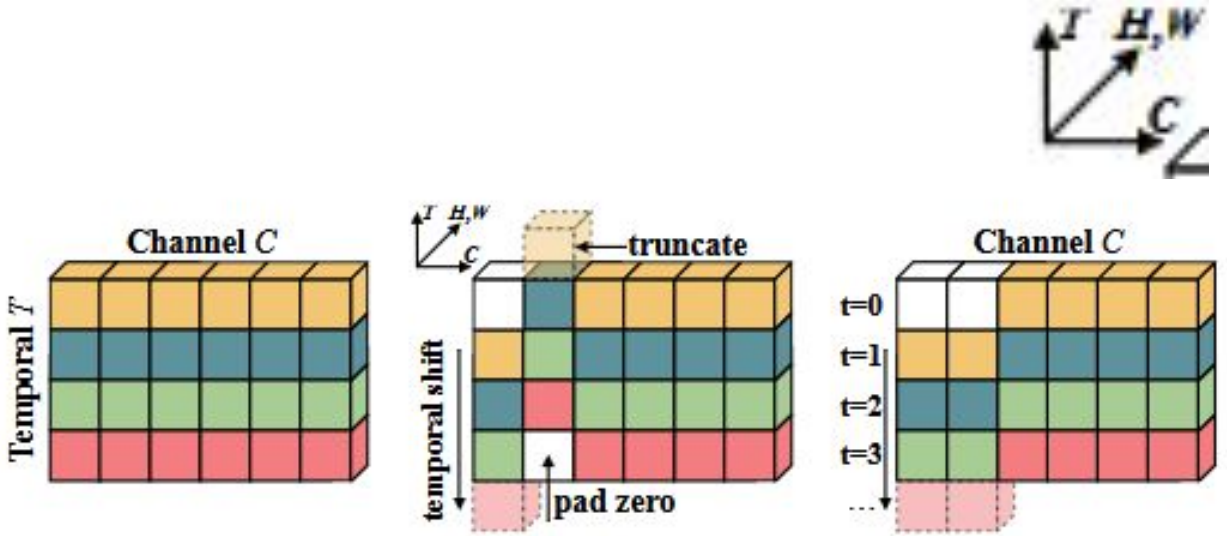
 Solution #3 : Add Shifting

Keeping the kernels 2D but we shift data across the T axis

+ Computation cost

- Complex shifting mechanisms

- Trade-off between spatial and temporal features



(a) The original tensor without shift. (b) Offline temporal shift (bi-direction). (c) Online temporal shift (uni-direction).

Other ways to work on video

 Regular machine learning & others…

We can also use ML algorithms to try to recognize video features.

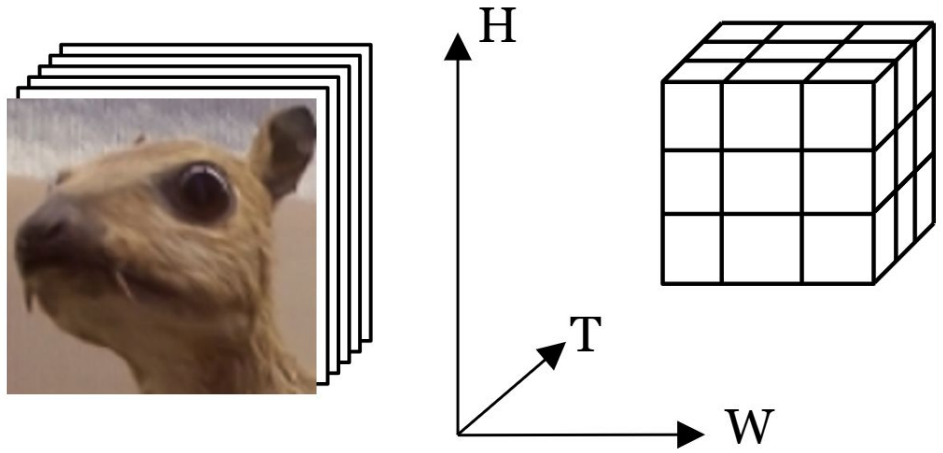
We will not explore these in this course (less efficient).

Depending on the data, anomaly detectors can be developed.

- More lightweight but less precise, notable ML algorithms for videos can be :
  - Decision trees
  - KNN
  - SVM…

Architectures optimized for action recognition

- 3D CNN Based : SlowFast

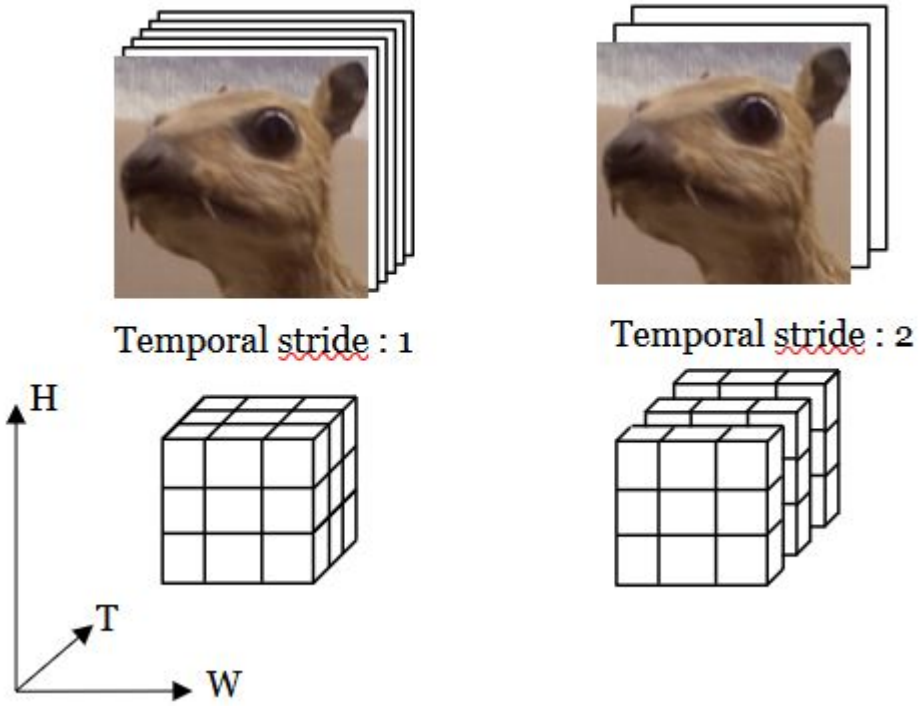


Remember 3D Convolution principle

In this example, temporal stride = 1

Architectures optimized for action recognition

- 3D CNN Based : SlowFast

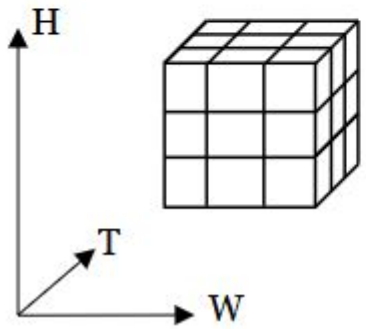


We can make the stride vary to pick on **features on a larger time scale** without more computation

Architectures optimized for action recognition

- 3D CNN Based : SlowFast

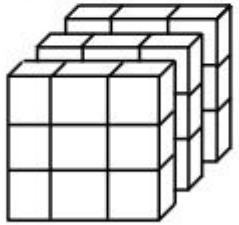
Temporal stride : 1



But we have to make a choice on the time scale… Right ?

Wrong

Temporal stride : 2

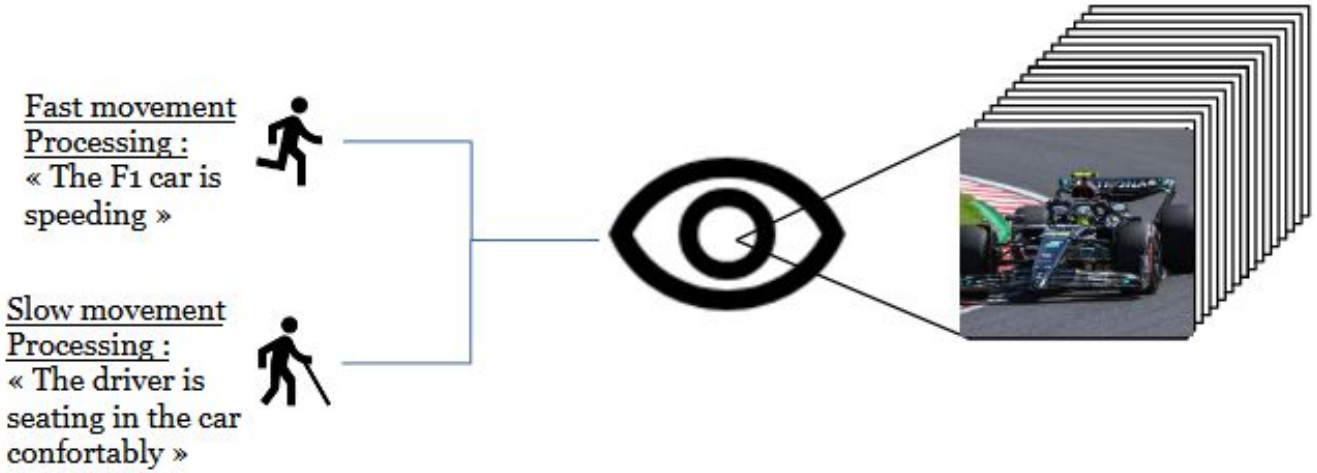


Architectures optimized for action recognition

3D CNN Based : SlowFast

Based on real-world action detection in animals…

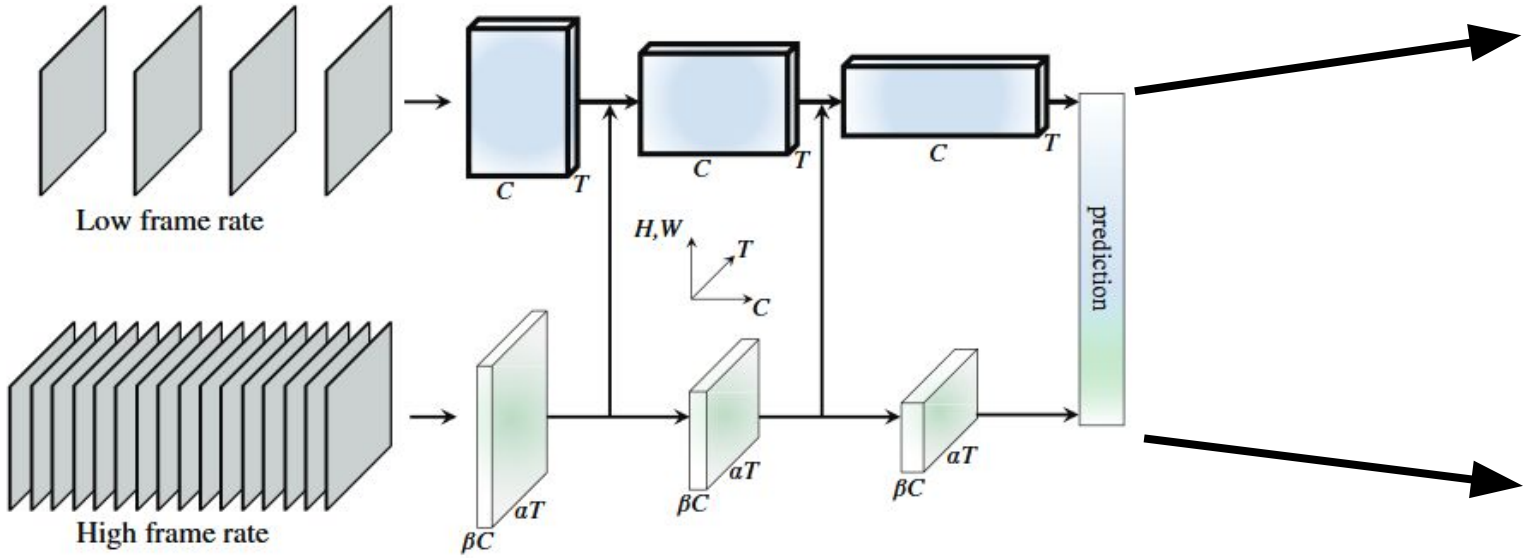
The brain processes stimuli differently based on the time scale.



This induces a « two way » data processing in the brain.

Architectures optimized for action recognition

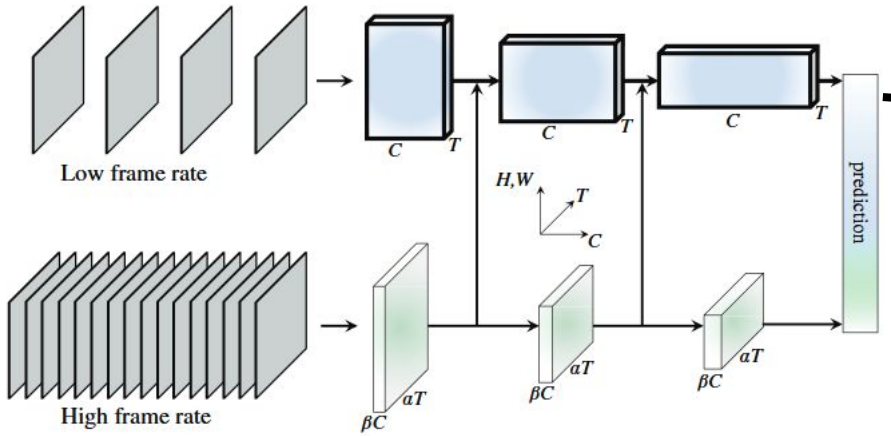
- 3D CNN Based : SlowFast



Based on previous observations… SlowFast !

Architectures optimized for action recognition

- 3D CNN Based : SlowFast



Slow data flow

- stride = S/**α**
- channels = C

Fast data flow

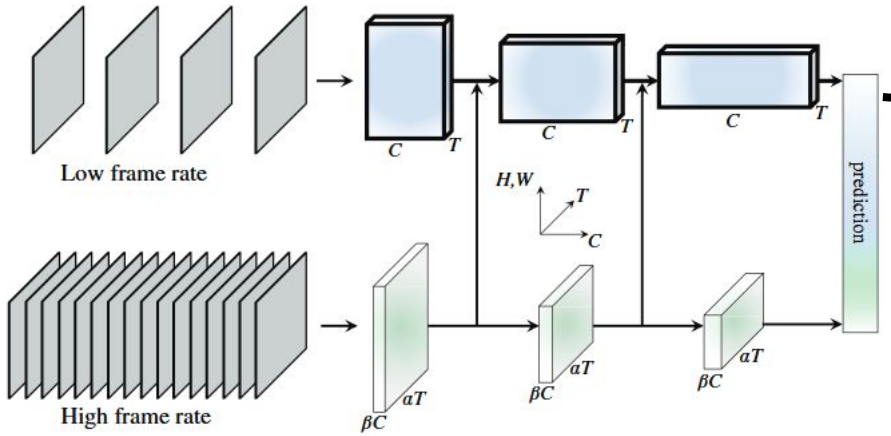
- stride = S
- channels = C/**β**

- **C & S** : The model's temporal stride and feature channels
- **α & β** : The C&S sampling ratios between slow & fast flow

Computer vision & action recognition theory

Architectures optimized for action recognition

- 3D CNN Based : SlowFast



Slow data flow

- 80% of the computation !

Fast data flow

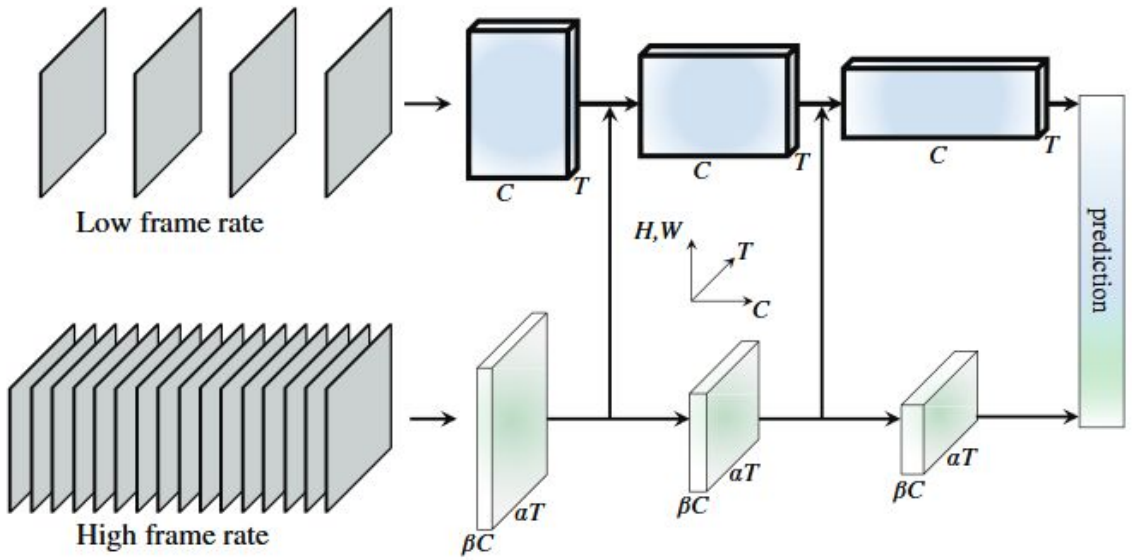
- 20% of the computation

for $\alpha = \beta = 4$

- Adding channels is $O(n^2)$ (quadratic w.r.t. kernels dimension)
- Adding time sampling is $O(n)$, thus the computing ratio.

Architectures optimized for action recognition

- 3D CNN Based : SlowFast

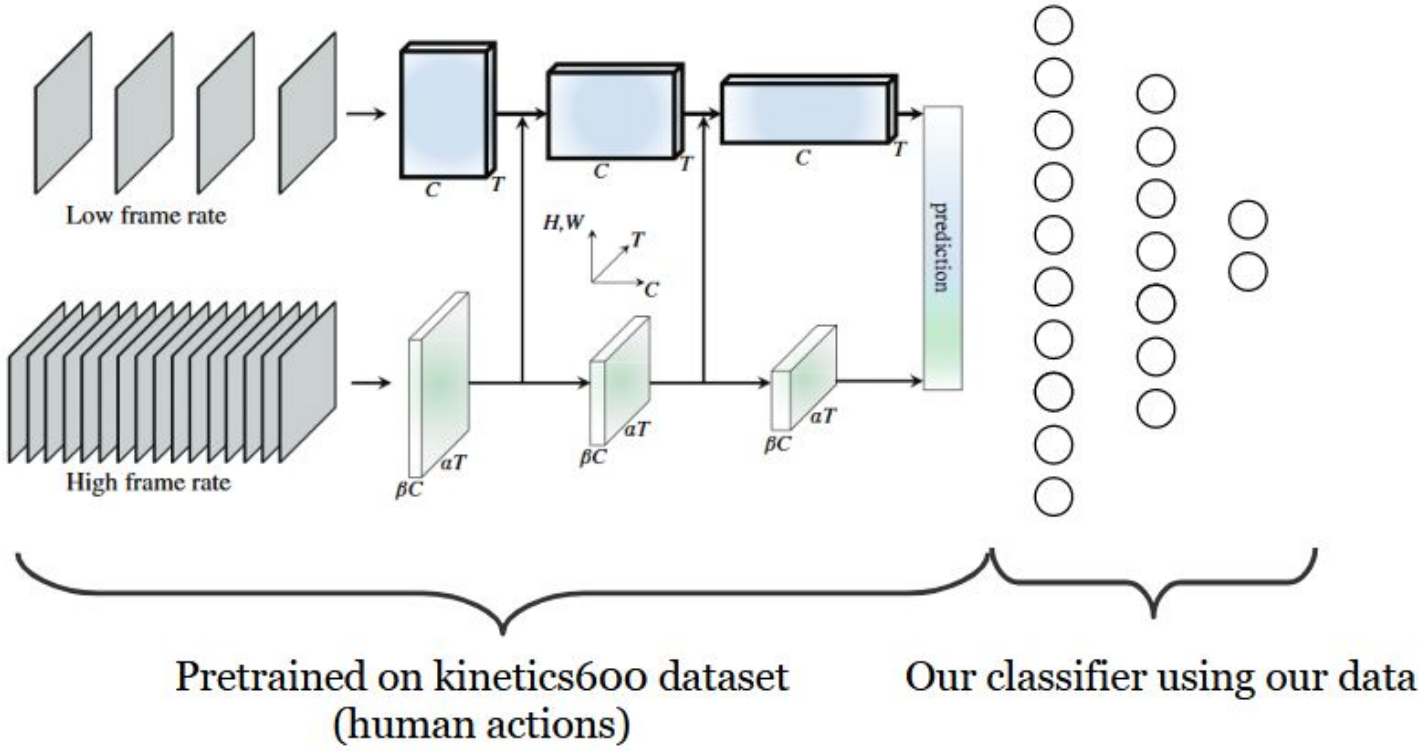


for $\alpha = \beta = 4$

stage	*Slow* pathway	*Fast* pathway	output sizes $T \times S^2$
raw clip	-	-	64×224^2
data layer	stride 16, 1^2	stride 2, 1^2	*Slow* : 4×224^2 *Fast* : 32×224^2
conv_1	1×7^2, 64 stride 1, 2^2	5×7^2, 8 stride 1, 2^2	*Slow* : 4×112^2 *Fast* : 32×112^2
pool_1	1×3^2 max stride 1, 2^2	1×3^2 max stride 1, 2^2	*Slow* : 4×56^2 *Fast* : 32×56^2
res_2	$\begin{bmatrix} 1 \times 1^2, 64 \\ 1 \times 3^2, 64 \\ 1 \times 1^2, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 1^2, 8 \\ 1 \times 3^2, 8 \\ 1 \times 1^2, 32 \end{bmatrix} \times 3$	*Slow* : 4×56^2 *Fast* : 32×56^2
res_3	$\begin{bmatrix} 1 \times 1^2, 128 \\ 1 \times 3^2, 128 \\ 1 \times 1^2, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 3 \times 1^2, 16 \\ 1 \times 3^2, 16 \\ 1 \times 1^2, 64 \end{bmatrix} \times 4$	*Slow* : 4×28^2 *Fast* : 32×28^2
res_4	$\begin{bmatrix} 3 \times 1^2, 256 \\ 1 \times 3^2, 256 \\ 1 \times 1^2, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 3 \times 1^2, 32 \\ 1 \times 3^2, 32 \\ 1 \times 1^2, 128 \end{bmatrix} \times 6$	*Slow* : 4×14^2 *Fast* : 32×14^2
res_5	$\begin{bmatrix} 3 \times 1^2, 512 \\ 1 \times 3^2, 512 \\ 1 \times 1^2, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 1^2, 64 \\ 1 \times 3^2, 64 \\ 1 \times 1^2, 256 \end{bmatrix} \times 3$	*Slow* : 4×7^2 *Fast* : 32×7^2
global average pool, concate, fc			# classes

Architectures optimized for action recognition

- 3D CNN Based : SlowFast

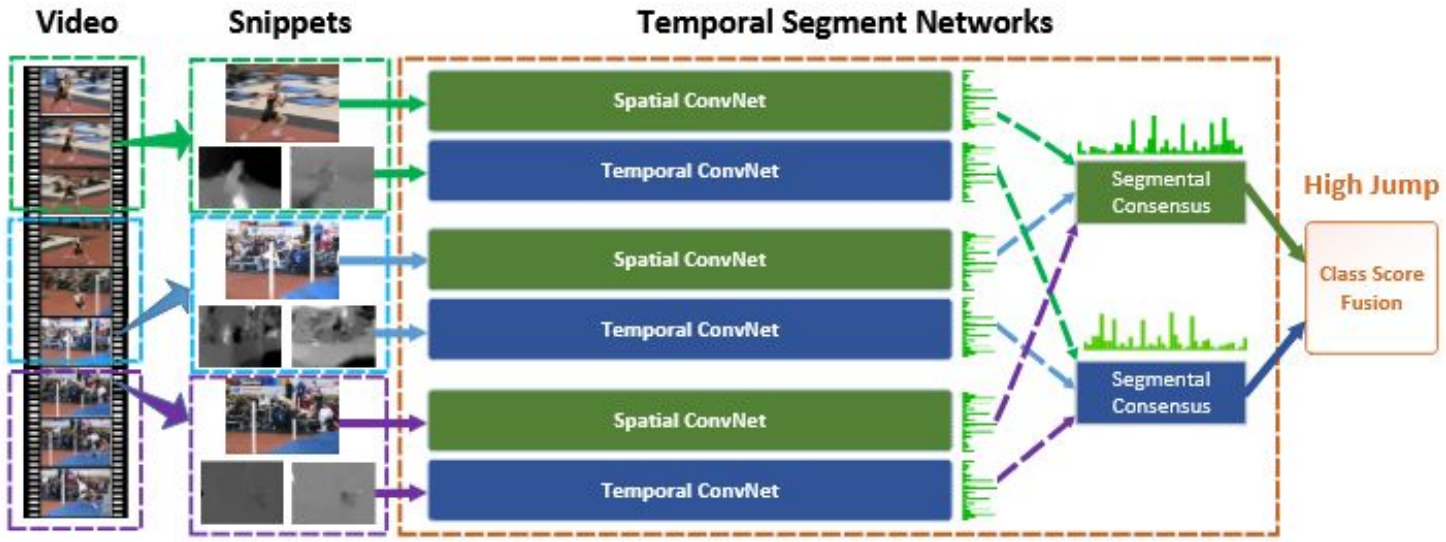


SlowFast exists as a pre-trained model that we can fine-tune using a fully connected layer and some data.

Architectures optimized for action recognition

2D CNN : TSN

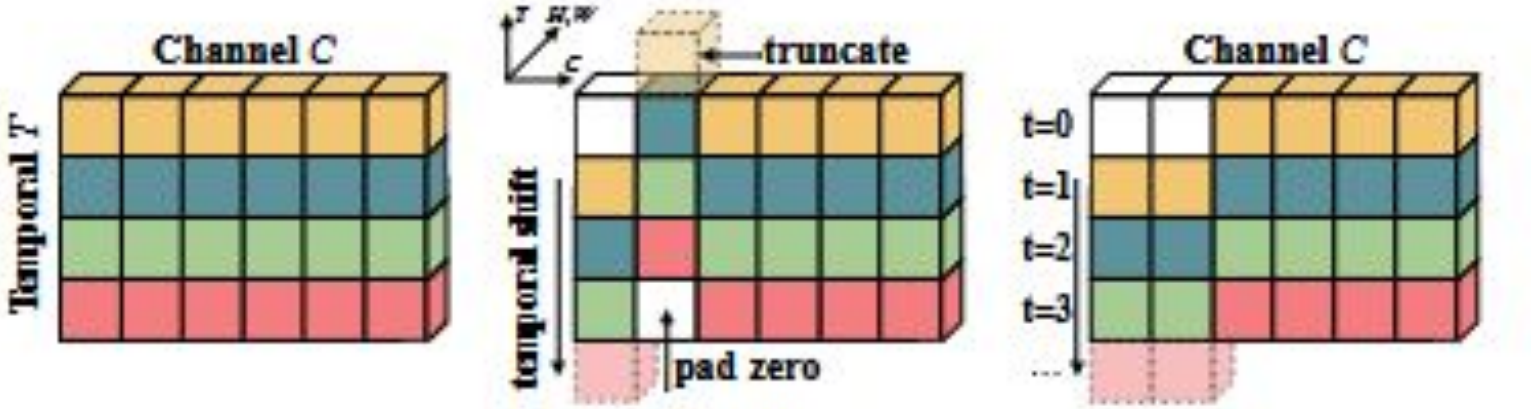
It creates K segments from a video and uses *N* 2D ConvNets on each one to reach a « consensus ».



Architectures optimized for action recognition

- 2D CNN Based : TSM

TSM (Temporal Shift Module) is a 2D CNN based on the shifting mechanism. (improved TSN)



Architectures optimized for action recognition

 2D CNN Based : TSM

On paper …

TSM is an equilibrium between spatial and temporal features extraction.

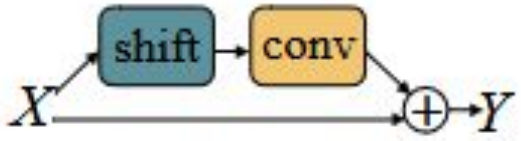
TSM induces memory overhead because shifting means retaining information.

« Naively » shifting data has a **negative impact on all metrics**

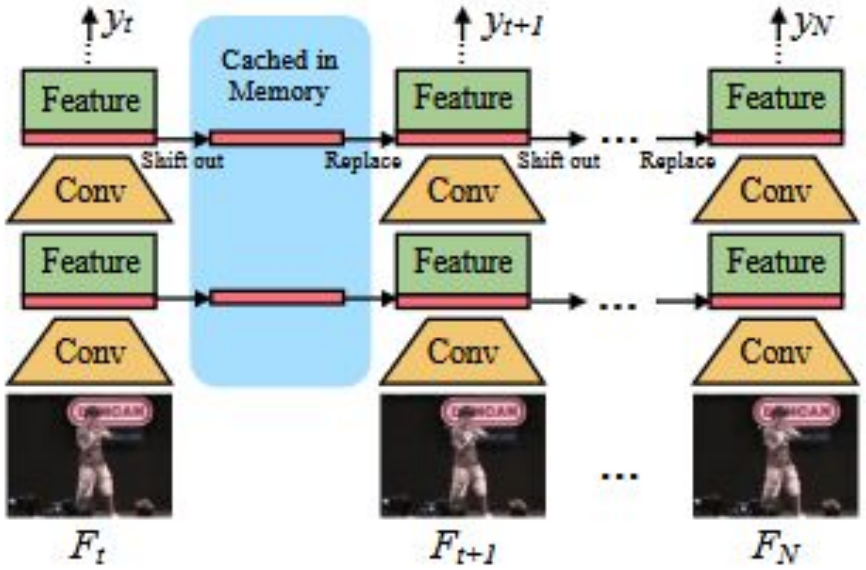
Architectures optimized for action recognition

- 2D CNN Based : TSM (addressing naive shift problems)

These downsides are addressed by using an approach similar to RNNs (Recurrent Neural Networks) by using « **residual** shift » ***and*** by shifting only a **portion** of the channels.



(b) Residual TSM.

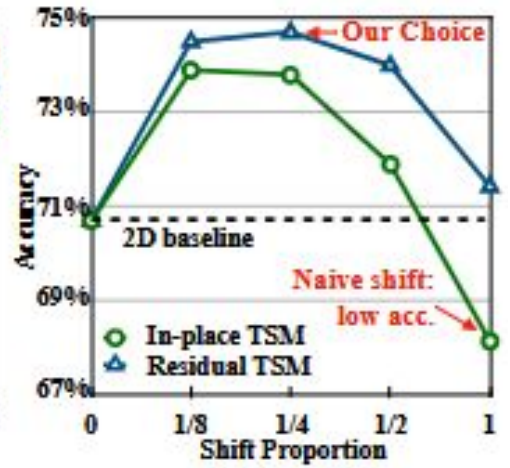


Architectures optimized for action recognition

- 2D CNN Based : TSM (addressing naive shift problems)

Here is how shifting only a portion of the channels translates in real performance impact

The intuition behind this is about reaching a « *spatial/time* » data « *equilibrium* ».

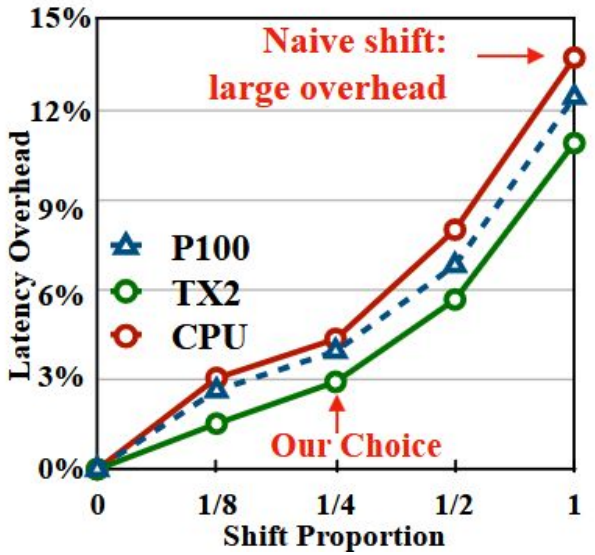


(b) Residual *vs.* in-place.

Architectures optimized for action recognition

- 2D CNN Based : TSM (addressing naive shift problems)

And we also don't shift ALL the channels, but only a few to limit overhead.



Architectures optimized for action recognition

2D CNN Based : TSM

- TSM shares the same global architecture as TSN
  - TSN can miss on some details that can happen between the Time Segments
  - We add a shifting mechanism to enhance temporal perceptions

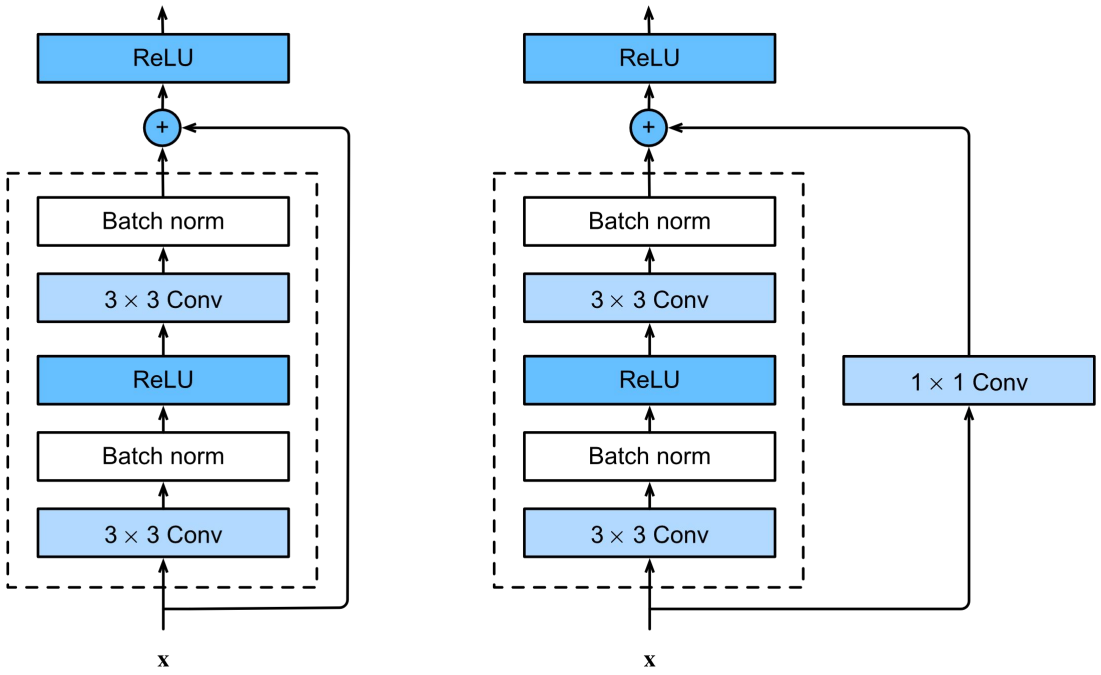
More on ResNets

Most models use residual nodes

Before moving on : Residual Networks.

- What ?
- Why ?
- Why in computer vision ?

$$H(x) = F(x) + x$$



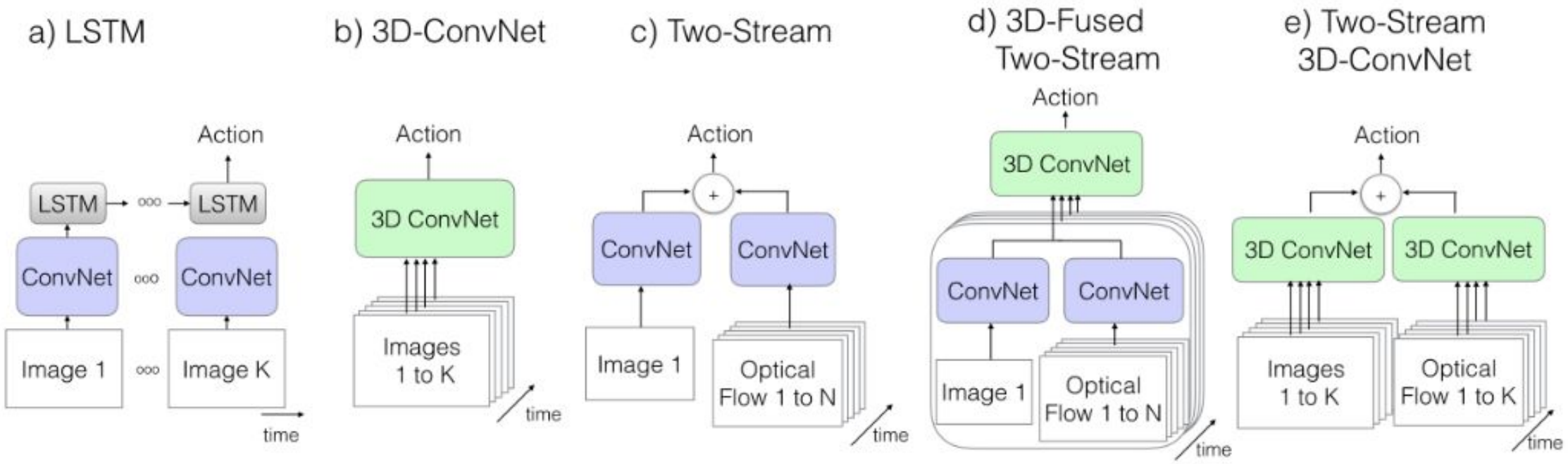
Computer vision & action recognition theory

In a nutshell…

- For action recognition, multiple streams Conv Residual networks are widely used.
- Notable architectures are :
  - SlowFast, 3D CNN based with optimised speed perception (ratio frame/channels)
  - TSN, 2D CNN on segments with consensus
  - TSM : TSN but with « *custom* » shifting mechanism to enhance time perception.

Computer vision & action recognition theory

Networks overview for action recognition



https://pytorch.org/hub/facebookresearch_pytorchvideo_slowfast/

https://pytorch.org/hub/facebookresearch_pytorchvideo_slowfast/

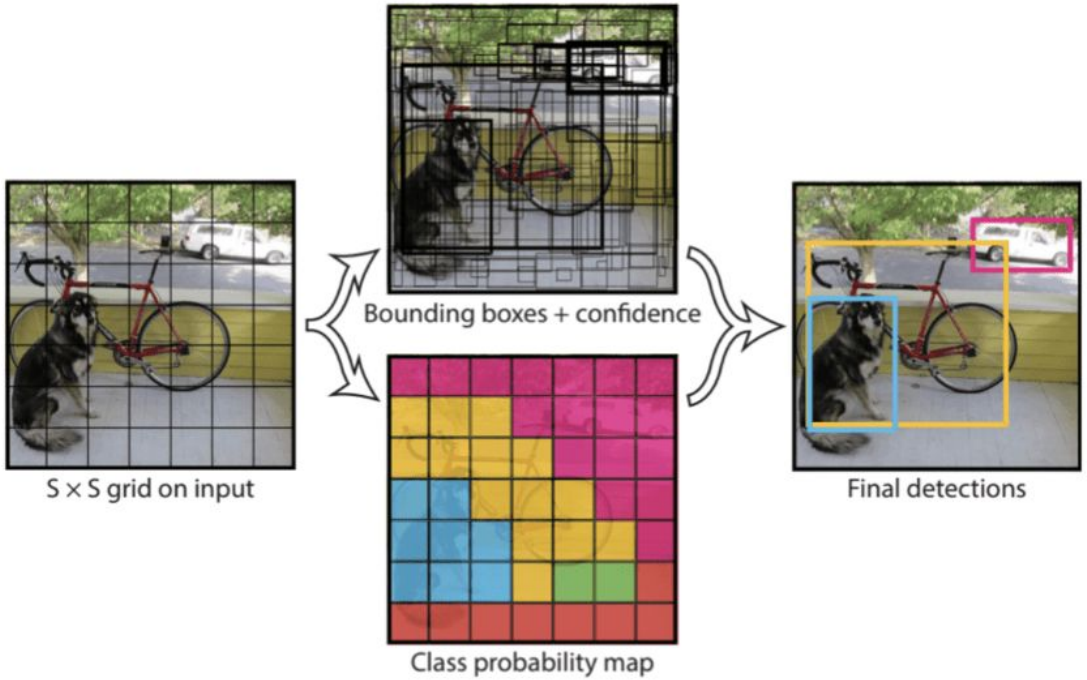
Quo Vadis, Action Recognition?
https://arxiv.org/pdf/1705.07750

https://github.com/open-mmlab/mmaction2/blob/main/configs/recognition/videomaev2/README.md

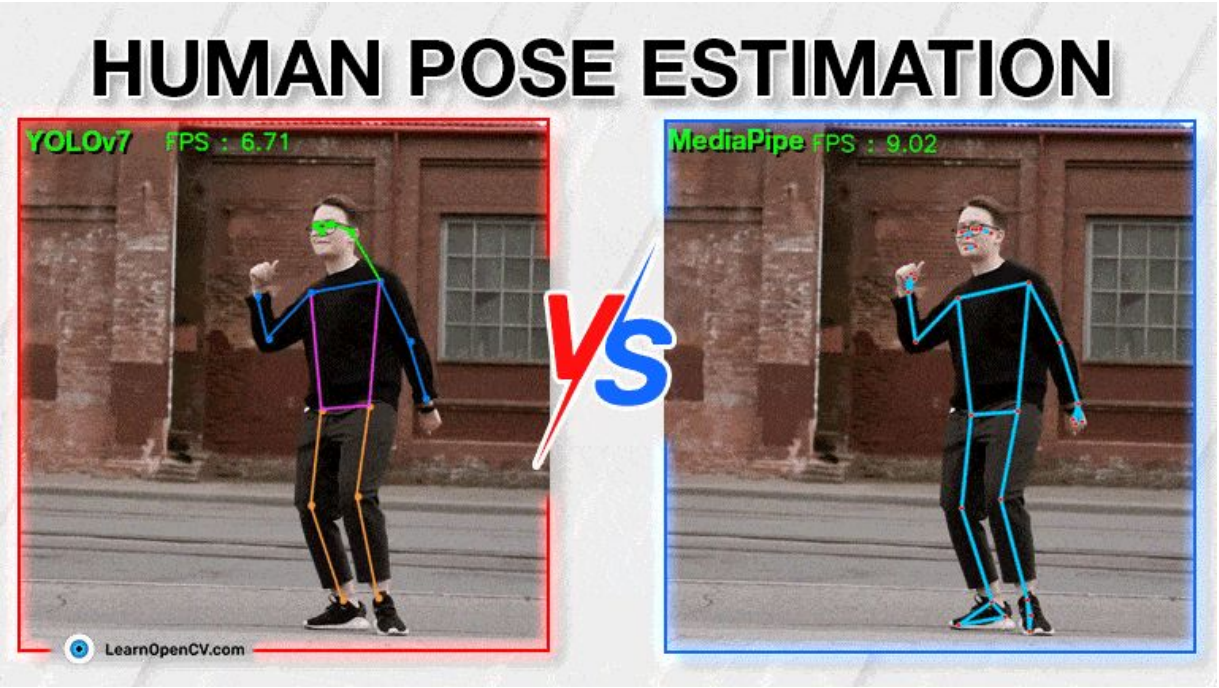
YOLO for action recognition

basic YOLO alike models are NOT a good solution for action recognition

- **Object detection** does not take into account temporal features
- Final outputs are not interpretable for action detection



YOLO for action recognition



Other variations can be used, like pose estimation

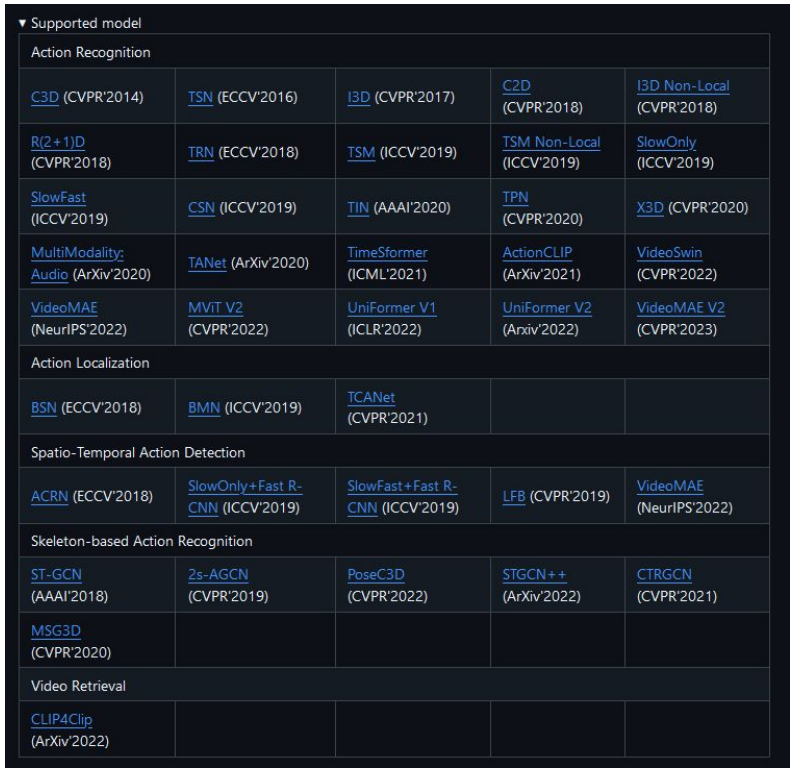
- Personal suggestion : you can reproduce a “SlowFast” like architecture on pose estimation
- The overall model should be lighter but you will be stacking 2 large models.

- This is called “skeleton based” action recognition.

Computer vision & action recognition theory

MMACTION 2

MMACTION2 is a toolbox for video analysis



▾ Supported model

Action Recognition				
C3D (CVPR'2014)	TSN (ECCV'2016)	I3D (CVPR'2017)	C2D (CVPR'2018)	I3D Non-Local (CVPR'2018)
R(2+1)D (CVPR'2018)	TRN (ECCV'2018)	TSM (ICCV'2019)	TSM Non-Local (ICCV'2019)	SlowOnly (ICCV'2019)
SlowFast (ICCV'2019)	CSN (ICCV'2019)	TIN (AAAI'2020)	TPN (CVPR'2020)	X3D (CVPR'2020)
MultiModality: Audio (ArXiv'2020)	TANet (ArXiv'2020)	TimeSformer (ICML'2021)	ActionCLIP (ArXiv'2021)	VideoSwin (CVPR'2022)
VideoMAE (NeurIPS'2022)	MViT V2 (CVPR'2022)	UniFormer V1 (ICLR'2022)	UniFormer V2 (Arxiv'2022)	VideoMAE V2 (CVPR'2023)
Action Localization				
BSN (ECCV'2018)	BMN (ICCV'2019)	TCANet (CVPR'2021)		
Spatio-Temporal Action Detection				
ACRN (ECCV'2018)	SlowOnly+Fast R-CNN (ICCV'2019)	SlowFast+Fast R-CNN (ICCV'2019)	LFB (CVPR'2019)	VideoMAE (NeurIPS'2022)
Skeleton-based Action Recognition				
ST-GCN (AAAI'2018)	2s-AGCN (CVPR'2019)	PoseC3D (CVPR'2022)	STGCN++ (ArXiv'2022)	CTRGCN (CVPR'2021)
MSG3D (CVPR'2020)				
Video Retrieval				
CLIP4Clip (ArXiv'2022)				

MMACTION 2 Contains models to do skeleton based action recognition.

https://github.com/open-mmlab/mmaction2

Lab 1 : Pretrained models practice

Preparing lab 1

We will now explore different solutions for action recognition (2D & 3D CNN)

Goal : compute inference metrics and chose the best solution for our needs

Tested models :

- TSN
- TSM
- SLOWFAST

Computed metrics:

- TOP1 Accuracy
- TOP5 Accuracy
- Inference time

Lab #1

Get hands-on experience with action recognition models

Lab 1 : Pretrained models practice

Notes on Data Flow for the models

Here is a recap of the transform / sampling metrics applied to the data (specific to slowfast) :

Parameter	Value	Description
`side_size`	256	Shorter side of frames resized to this size before cropping.
`crop_size`	256	Final spatial size (height & width) of frames after cropping.
`num_frames`	32	Number of frames per input video clip.
`mean`	[0.45, 0.45, 0.45]	Mean pixel values for RGB channels (used for normalization).
`std`	[0.225, 0.225, 0.225]	Standard deviation for RGB channels (used for normalization).
`sampling_rate`	2	Interval between selected frames in a sequence (every 2nd frame).
`frames_per_second`	30	Original frame rate of input video.
`slowfast_alpha`	4	Ratio of sampling rates between Slow and Fast pathways.
`num_clips`	10	Number of video clips sampled for inference.
`num_crops`	3	Number of spatial crops per clip (for augmentation).
`clip_duration`	(32 × 2) / 30 ≈ 2.13 sec	Temporal duration of each clip in seconds.

Lab 1 : Pretrained models practice

Open the « *Action_recognition_lab1.ipynb* » Notebook.

What you'll learn in lab 1 :

- Gather and pre-process video data.
- Use the discussed networks for inference
- Compare the results

LAB 1 : Recap

MODEL	TOP1 ACCURACY (1C1C)	TOP5 ACCURACY (1C1C)	EXEC TIME/SAMPLE (NVIDIA T4)
SLOWFAST	41%	73%	0.67s
TSM	31%	48%	0.34s
TSN	21%	47%	0.42s

Goals :

- Discuss & interpret these results...
- Compute the vid/s (pred/s) and interpret this metric
- Conclude on possible applications...

Lab 1 : Pretrained models practice

Lab 1 : Bonus #1 : Live webcam recognition demo

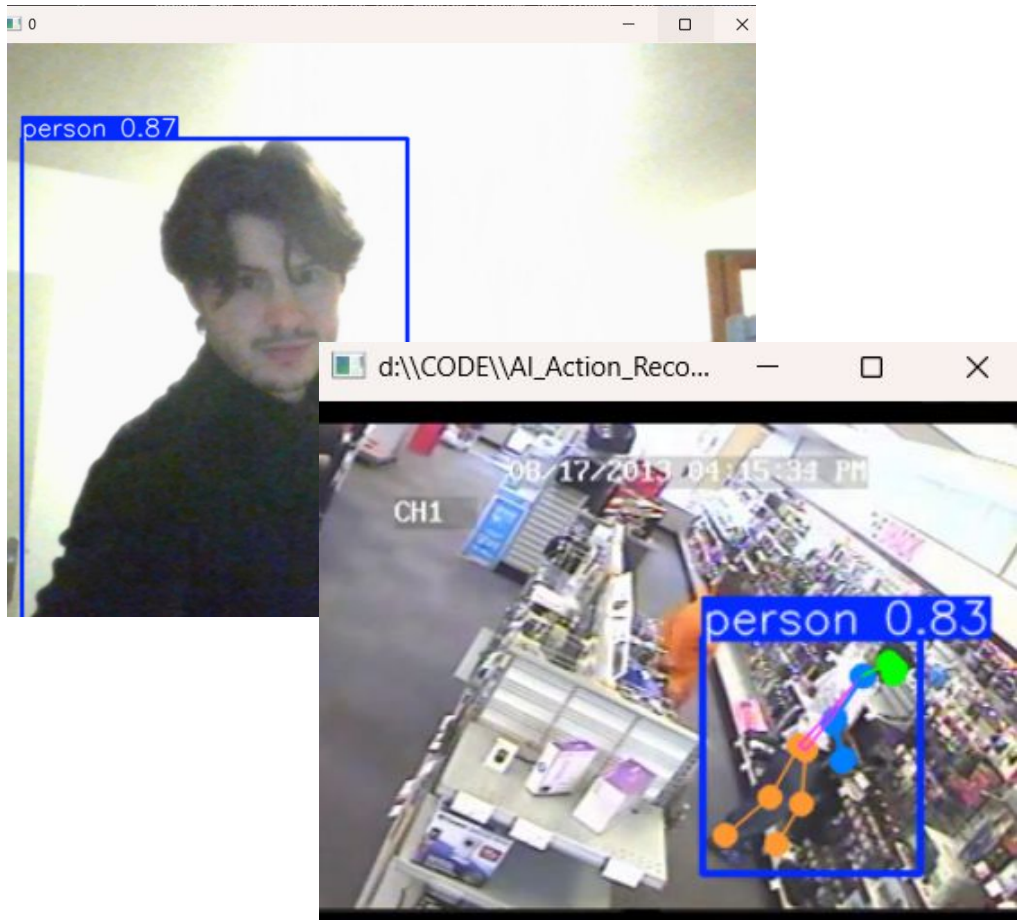
(Not a real cigarette, smoking is bad, do not try this at home)

Open the « live_demo.ipynb » file

- On windows, using openCV2
- Allows to run LIVE recognition on your webcam
- Try to mimic various actions :
  - Smoking
  - Clapping
  - Sign language…

Lab 1 : Pretrained models practice

Lab 1 : Bonus #2 : YOLO Inference



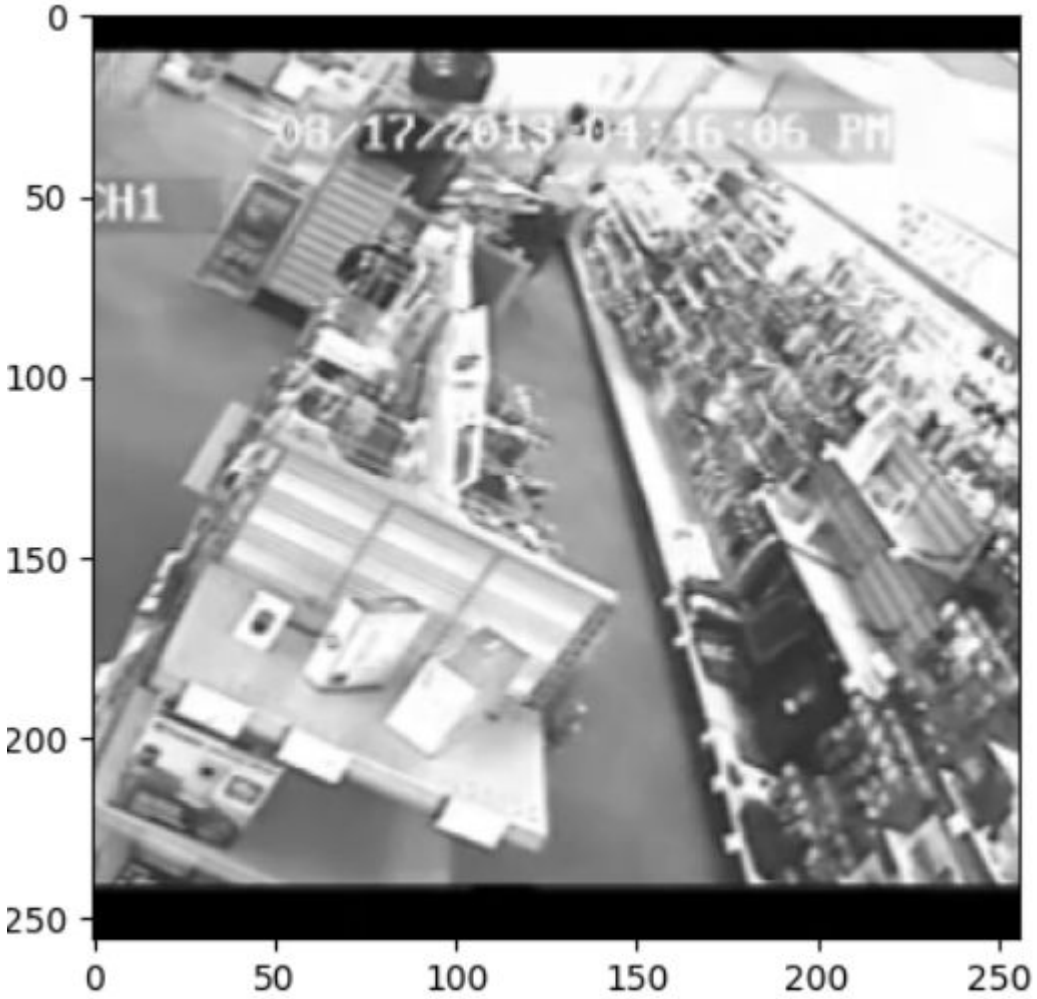
If we have time, we'll also test inference with yolo to get both :

- Regular inference (boxes)
- Skeleton based inference

Action recognition specifics : Shoplifting

End-to-end workflow towards building an AI shoplifting detector

Action recognition specifics : Shoplifting



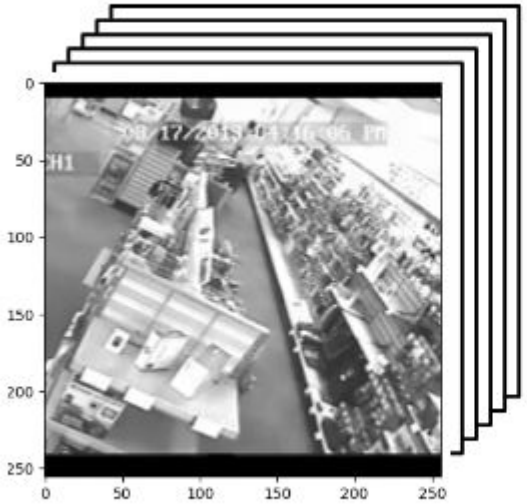
Here is an example of a CCTV frame from a video.

Objective : Determine if…

- Someone's stealing ?
- Or not ?

Action recognition specifics : Shoplifting

Problem Statement

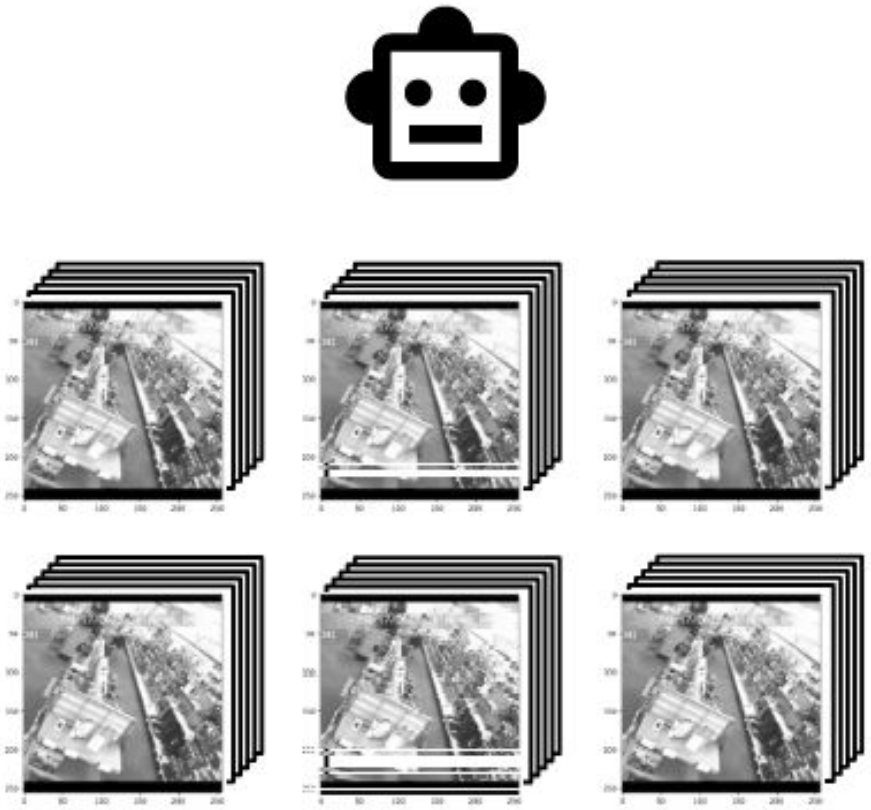


Stores often pay someone to monitor security cameras.

- Expensive
- Not efficient
- Labor intensive

Action recognition specifics : Shoplifting

Our Goal / Main Requirements



Make the shoplifting detection **automatic** across a store.

Allows :

- More CCTV coverage
- Better efficiency
- Cheaper
- Human workforce can focus on providing final judgement / better security

Action recognition specifics : Shoplifting

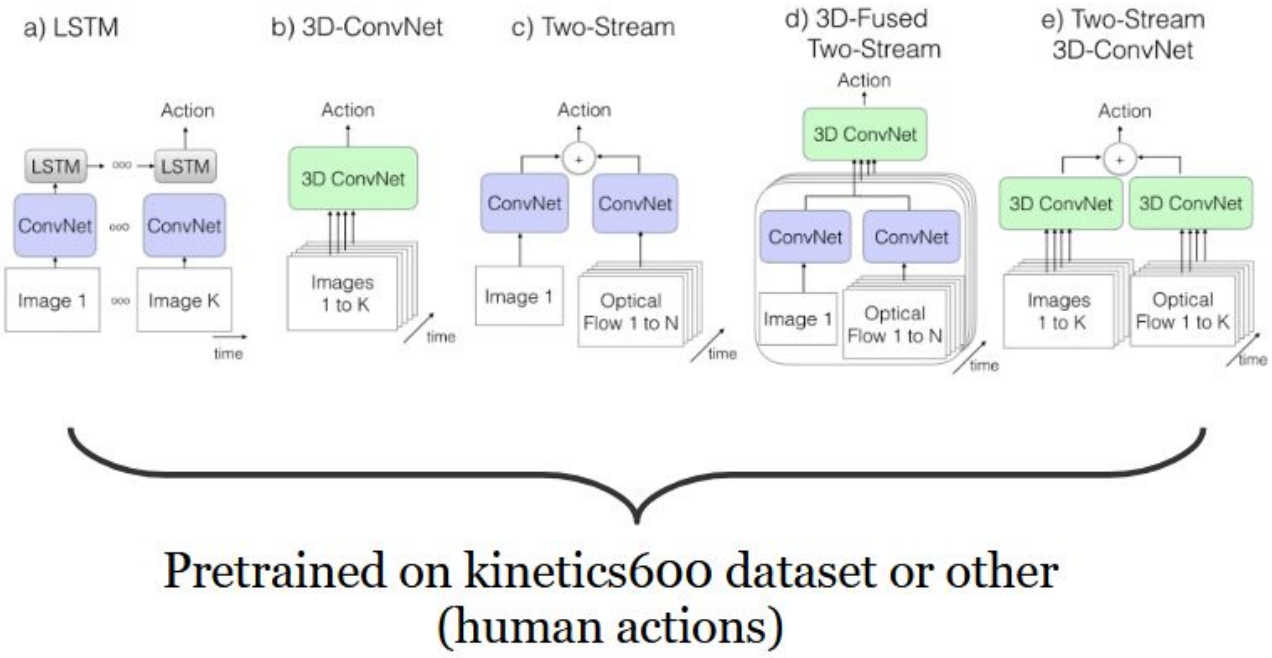
 Our Goal / Main Requirements

 Live prediction example :

Turn this into a **“Crime risk prediction”**

Action recognition specifics : Shoplifting

 How?



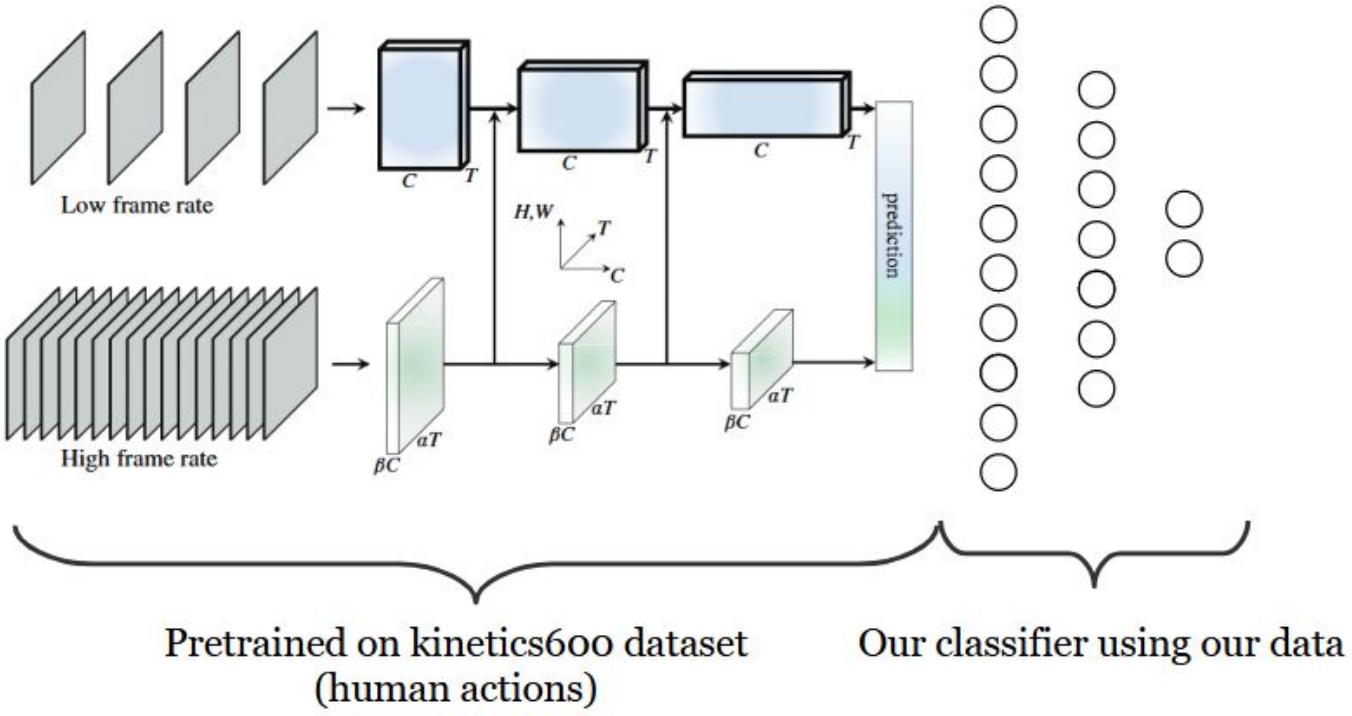
Pretrained on kinetics600 dataset or other (human actions)

We can use the models discussed in lecture #1 pretrained on action recognition

Action recognition specifics : Shoplifting

How?

And simply add a fine-tuning layer after it !




```python
import torch.nn as nn

class CustomClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dims, num_classes=2):
        super(CustomClassifier, self).__init__()

        layers = []
        prev_dim = input_dim

        for dim in hidden_dims:
            layers.extend([
                nn.Linear(prev_dim, dim),
                nn.ReLU(),
                nn.Dropout(0.5)
            ])
            prev_dim = dim

        layers.append(nn.Linear(prev_dim, num_classes))

        self.classifier = nn.Sequential(*layers)

    def forward(self, x):
        return self.classifier(x)
```

Action recognition specifics : Shoplifting

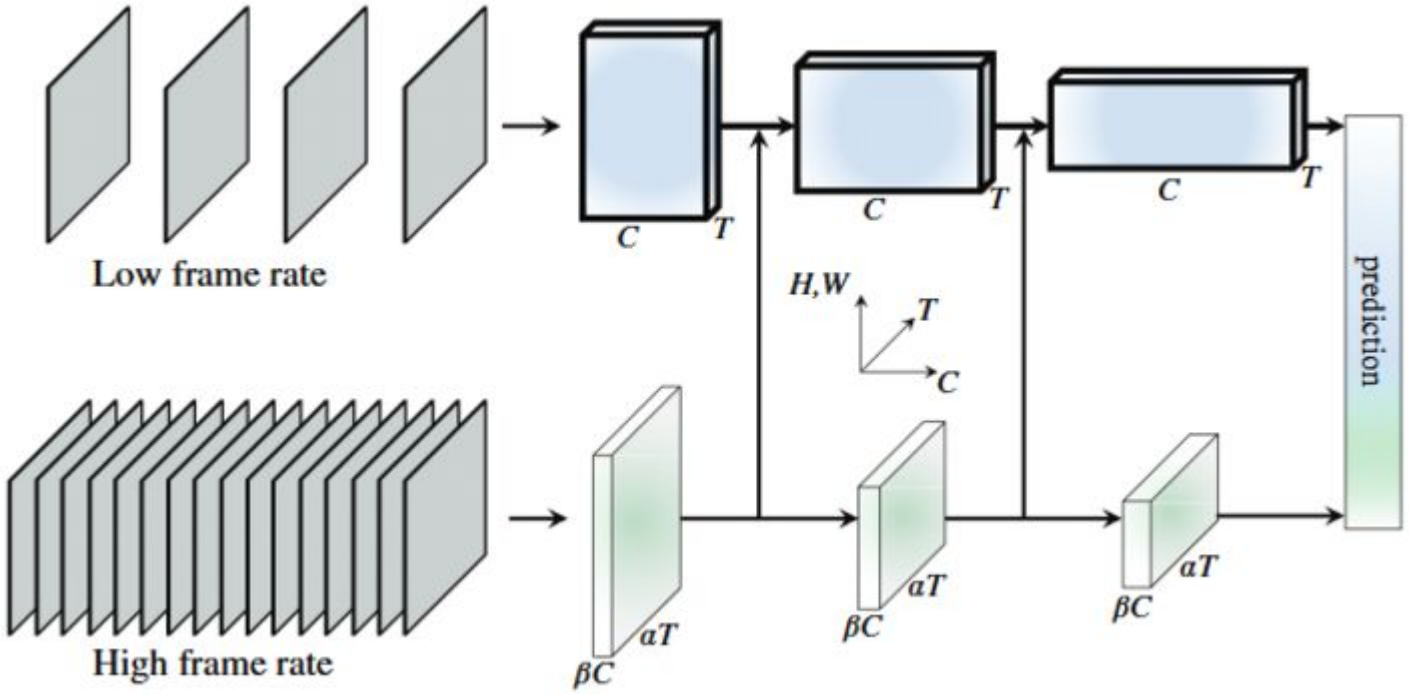
How does fine-tuning works ?

"In deep learning, fine-tuning is an approach to transfer learning in which the parameters of a pre-trained neural network model are trained on new data. Fine-tuning can be done on the entire neural network, or on only a subset of its layers, in which case the layers that are not being fine-tuned are "frozen" (i.e., not changed during backpropagation)."

-Wikipedia

Action recognition specifics : Shoplifting

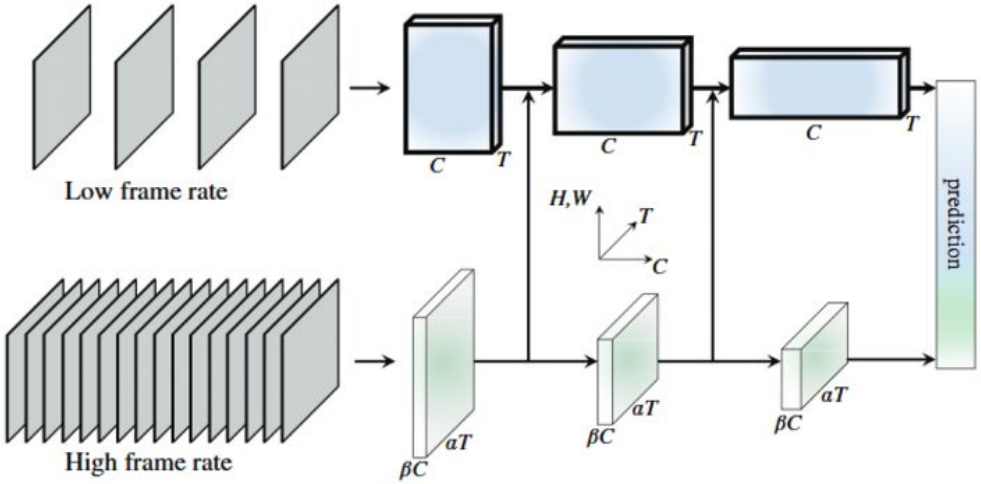
 How does fine-tuning works ?



$$[P_0; P_1; P_2; \ldots; P_{399}]$$

Action recognition specifics : Shoplifting

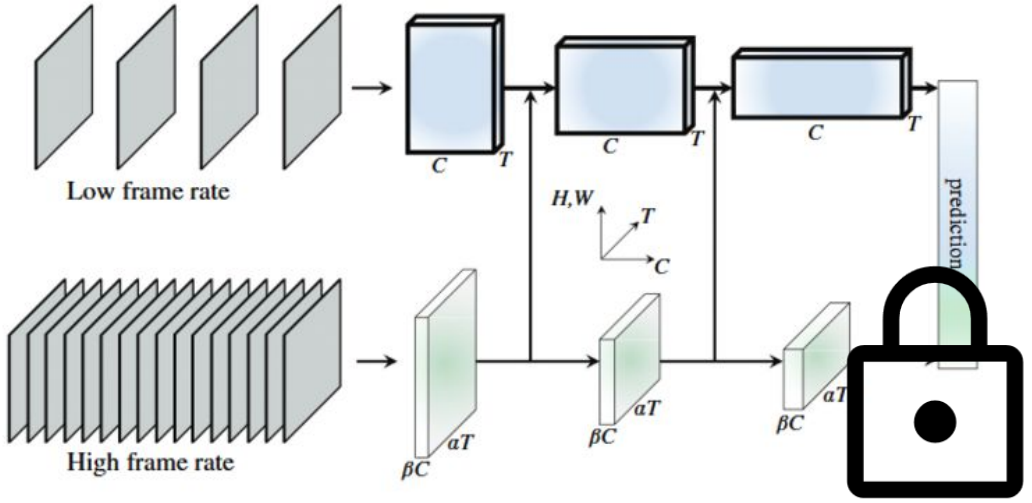
- How does fine-tuning works ?



$$[P_0; P_1; P_2; \ldots; P_{399}]$$

Action recognition specifics : Shoplifting

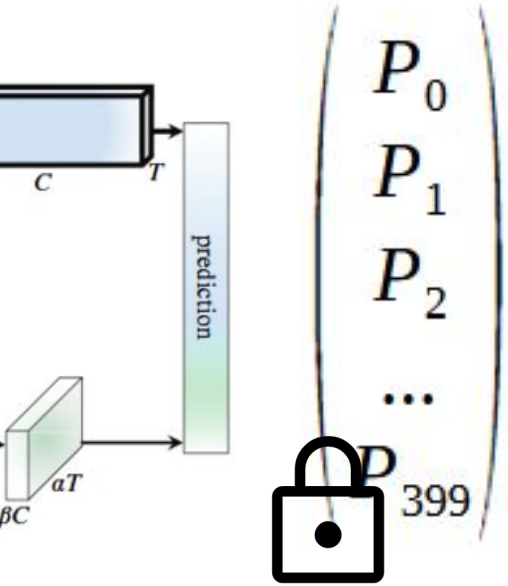
- How does fine-tuning works ?

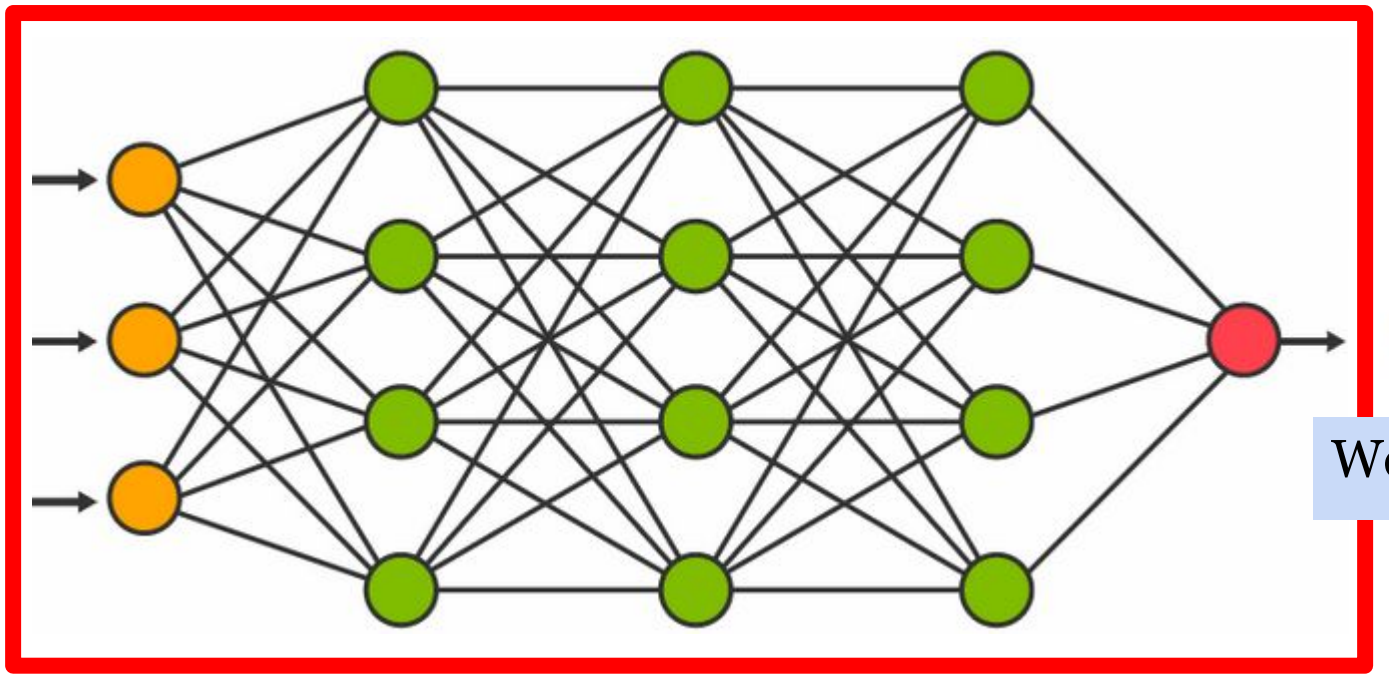


$$[P_0; P_1; P_2; \ldots; P_{399}]$$

Action recognition specifics : Shoplifting

How does fine-tuning works ?





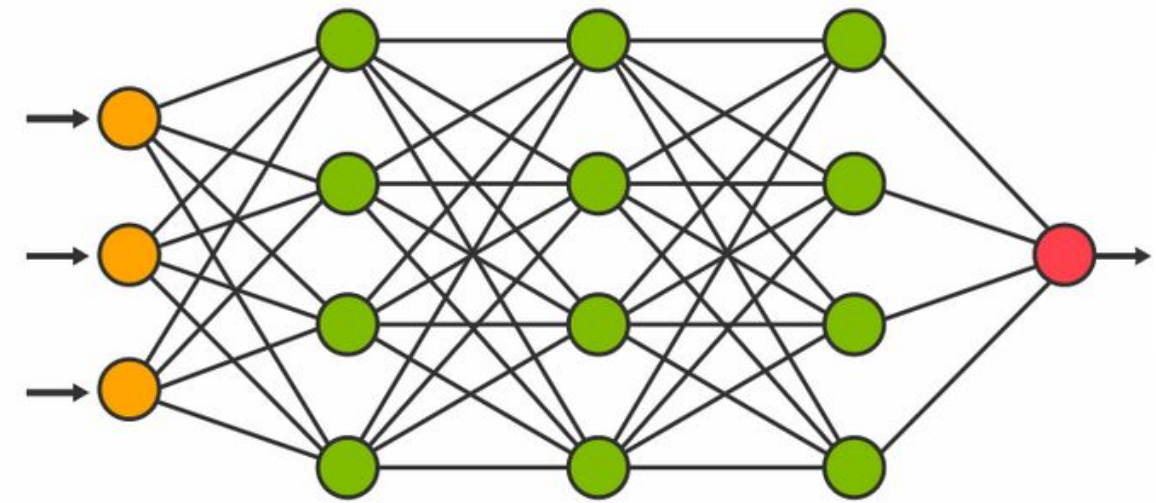
$P_{shoplifting} \in [0,1]$

We get 1 final prediction

This is what we'll "train" using our shoplifting data.

Action recognition specifics : Shoplifting

How does fine-tuning works ?



$P_{shoplifting} \in [0,1]$

LOSS function ?

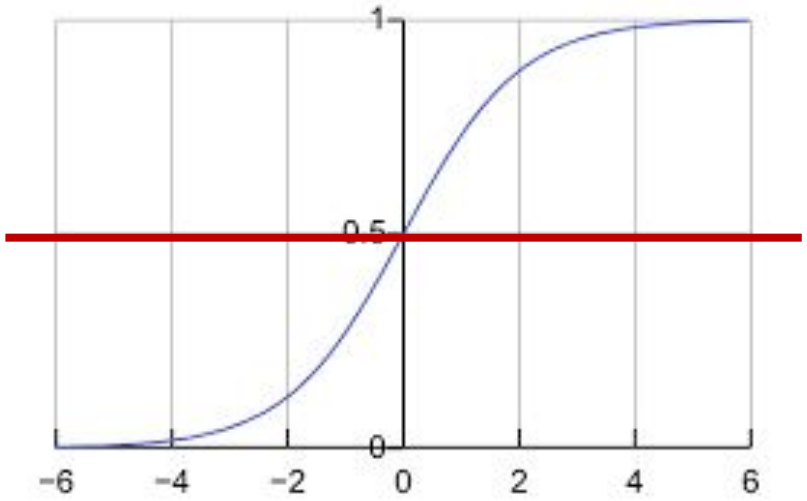
For simple binary application like these, common loss functions does works. We have to use BCE :

```
# criterion = CrossEntropyLoss()
# BINARY CROSS ENTROPY LOSS
criterion = nn.BCEWithLogitsLoss()
optimizer = Adam(fine_tune.parameters(), lr=0.0001)
print(device)
```

Action recognition specifics : Shoplifting

How to use fine-tuning ?

$$P_{shoplifting} \in [0,1]$$



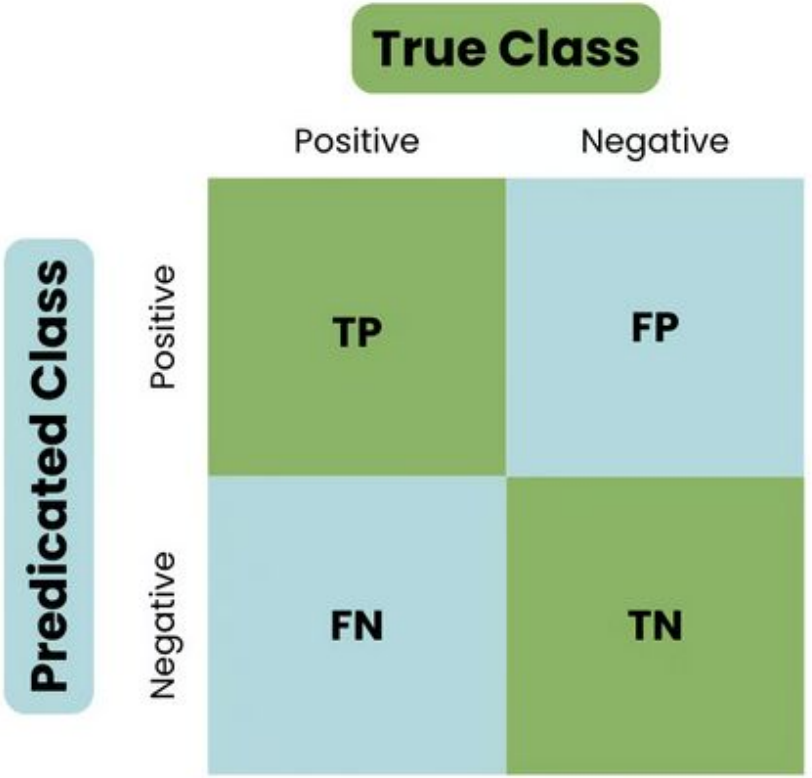
Once the model is trained, we simply have to chose a threshold to raise the alarm !

e.g : *"declare a crime when the crime score is above 50% (0 on the x axis)"* This value is called "threshold

Action recognition specifics : Shoplifting

How to use fine-tuning ?

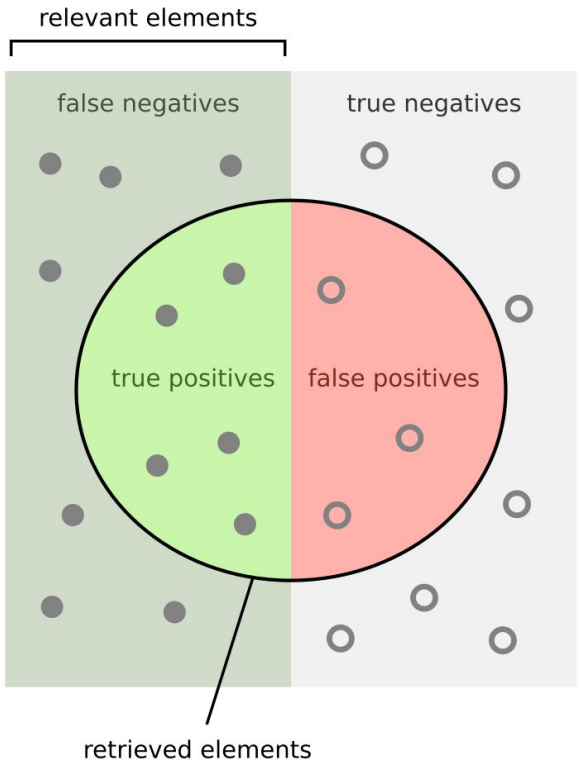
To measure our classifier performances, we'll use a simple **binary confusion matrix**



Action recognition specifics : Shoplifting

Measuring model's performances

From the confusion matrix, we can compute a number of interesting metrics :



How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{retrieved elements}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

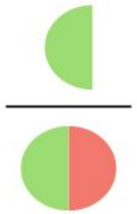
Note : these only work for binary classification problems

Action recognition specifics : Shoplifting

Measuring model's performances

We can easily maximise Precision or recall by craking the threshold to extreme values.

How many retrieved items are relevant?

Precision = 

How many relevant items are retrieved?

Recall = 

But this is not what we want ! we want en equilibrium

Action recognition specifics : Shoplifting

Measuring model's performances

To find the perfect threshold for our final product : F1 Score !

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}.$$

This helps us finding a good equilibrium, which we can bias depending on the criticality of the different metrics

In our shoplifting case, recall is the most important ! We might want to bias the result towards it !

Action recognition specifics : Shoplifting

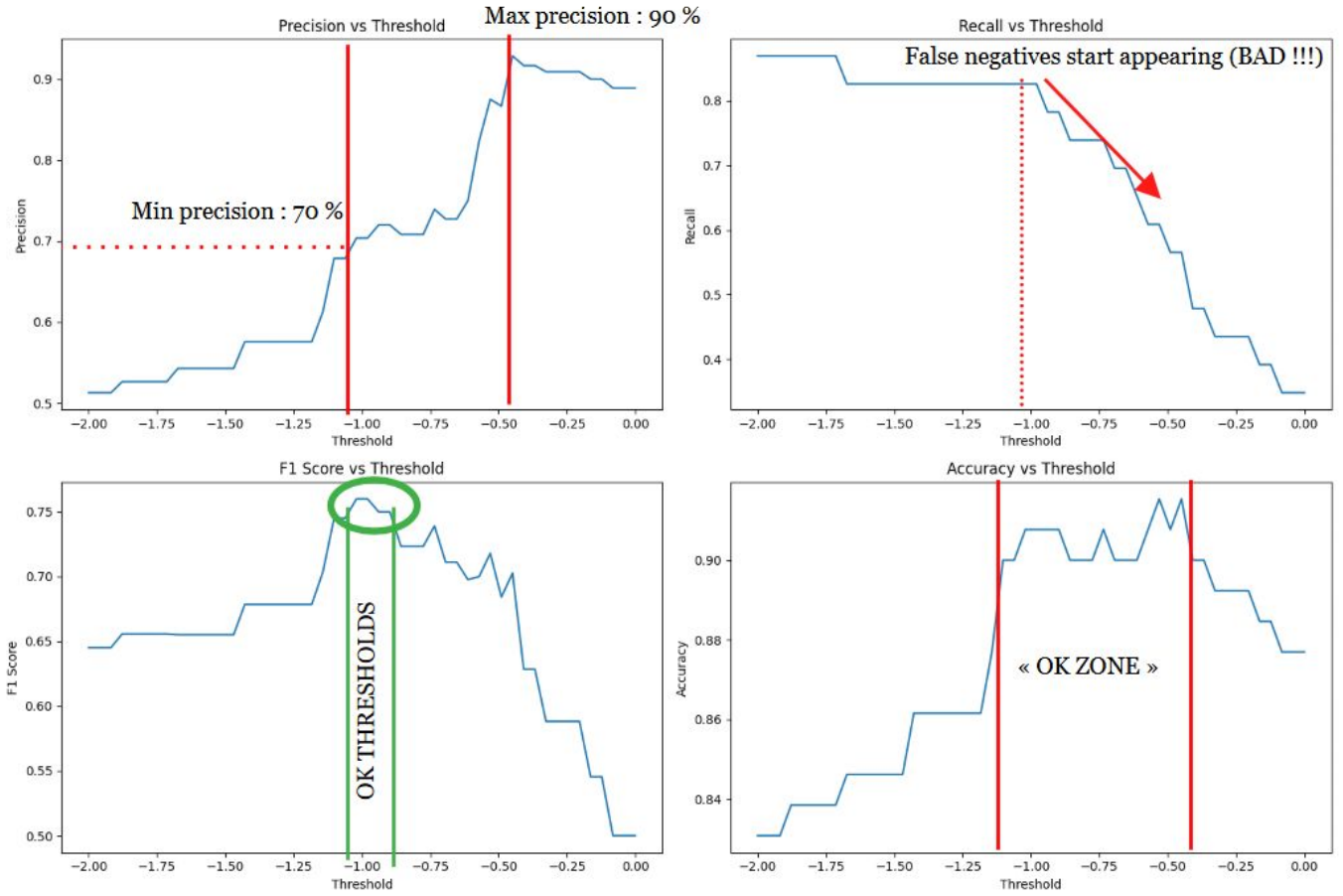
Performances example from LAB2

False negatives = didn't spot crime (BAD!!)

False positive = Alert but NO crime

True positive = Crime detected (success)

True negative = No alert, no crime

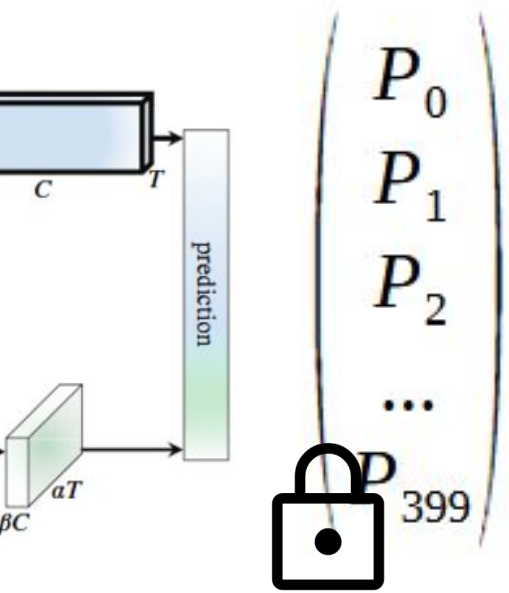


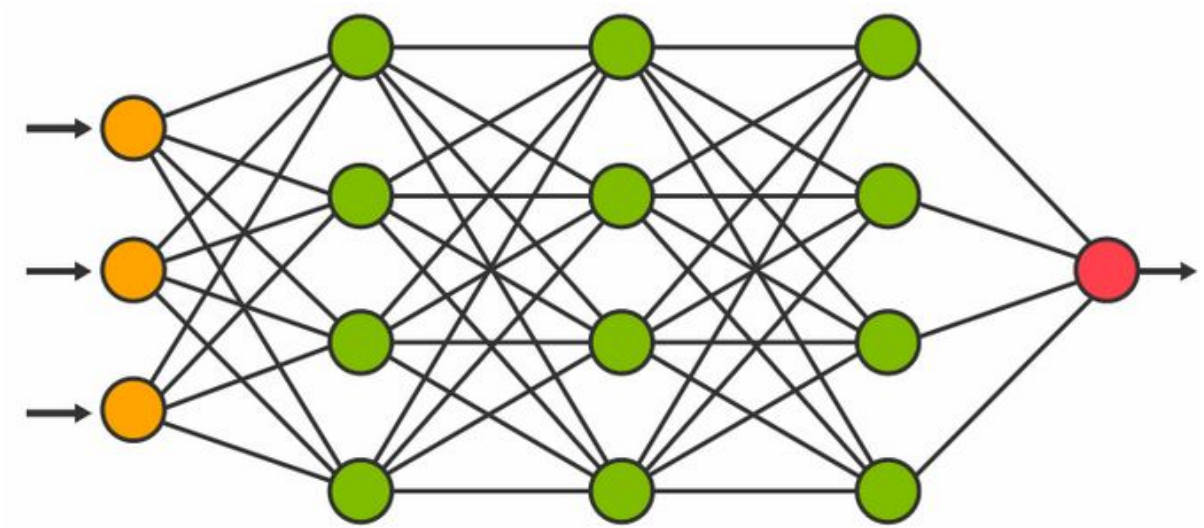
Model precision on this range : 70~73 % => Human intervention conforms the crime before confrontation

Model recall on this range : 80 % => Catches ~80 % of crimes

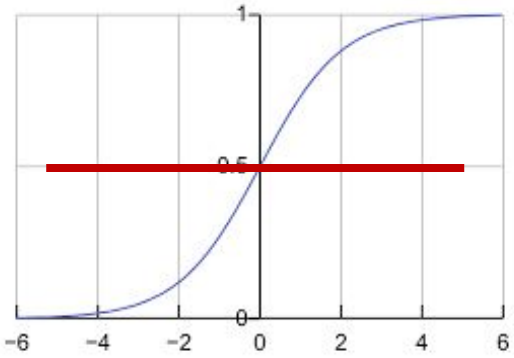
Action recognition specifics : Shoplifting

- How hard/expensive is fine-tuning ?





$$P_{shoplifting} \in [0,1]$$



Training time ?

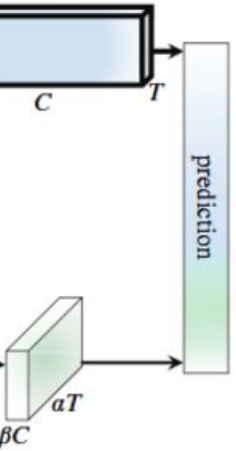
1 - 2 HOURS on a simple T4 GPU !

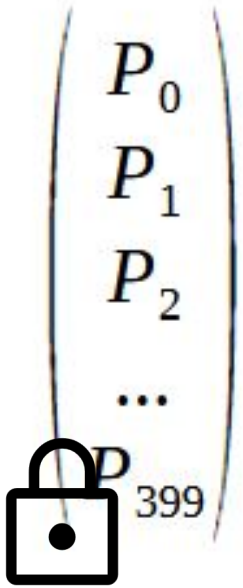
Optimisation by naive exploration included !

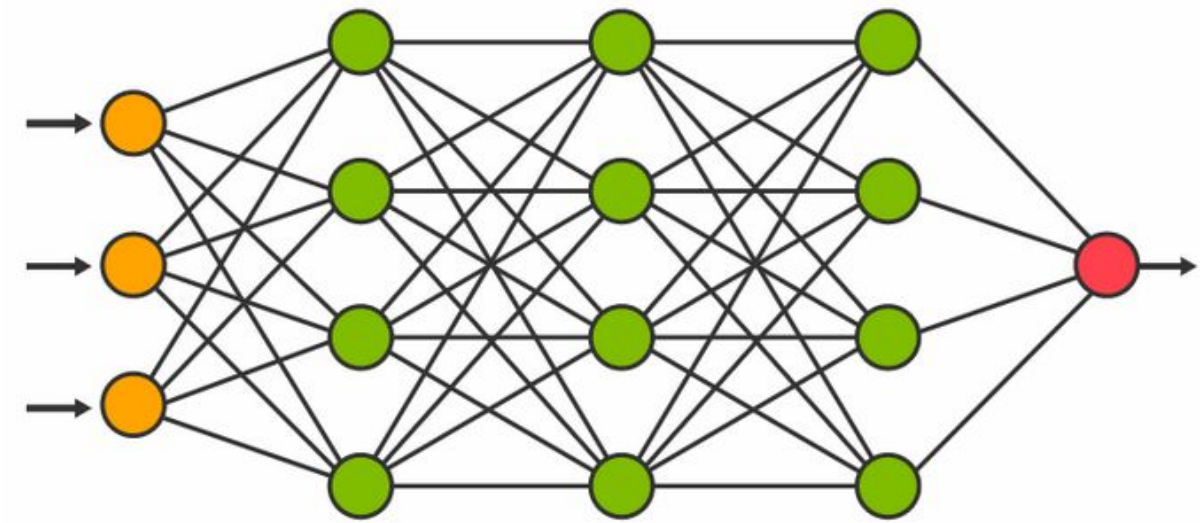
This is why fine-tuning is so great !

Action recognition specifics : Shoplifting

Other fine-tuning scenarios







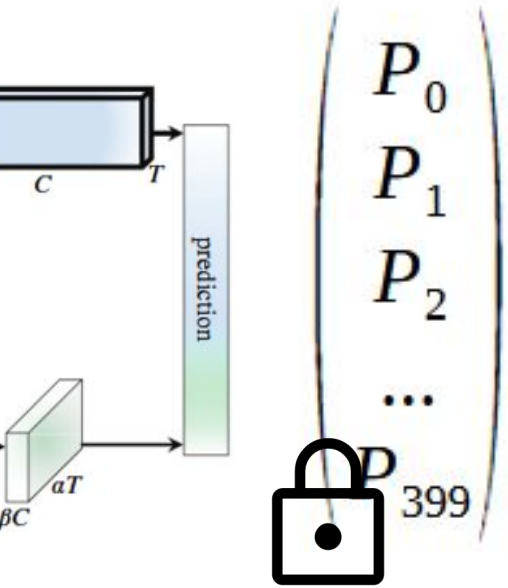
$$Prob_{shoplifting} \in [0,1]$$
$$Prob_{smoking} \in [0,1]$$
$$Prob_{agression} \in [0,1]$$
...

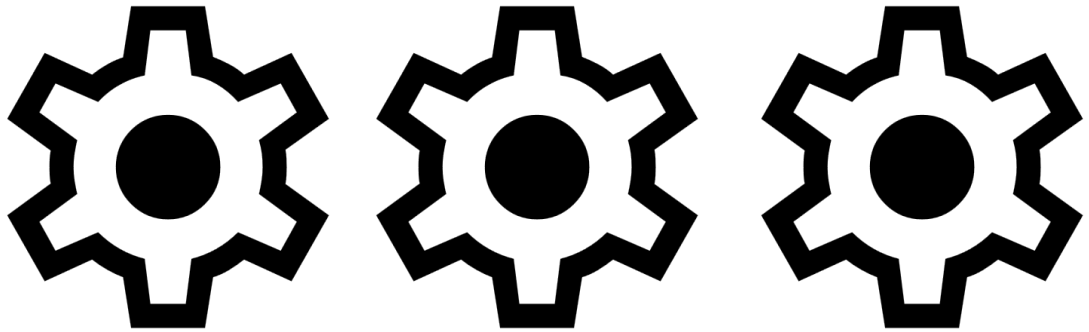
Adding classes is a simple process, but you will need to adapt :

- Loss function
- Data (need more labels)
- More Data (More classes = more complex features)

Action recognition specifics : Shoplifting

Other fine-tuning scenarios





$$Prob_{shoplifting} \in [0,1]$$
$$Prob_{smoking} \in [0,1]$$
$$Prob_{agression} \in [0,1]$$
...

The example we'll cover (Lab2) are just an example, you can use different fine tuning architectures or even run backward propagation on the actual models if it is allowed.

You can even go up to designing you own Loss function and use multiple models for training (not necessary for shoplifting tough).

Action recognition specifics : Shoplifting

Gathering data

How do we get data ?

FREE public datasets
(e.g. DCSASS for occident-like stores)
(e.g. TheftDetectingCCTV for asia self-service stores)

or

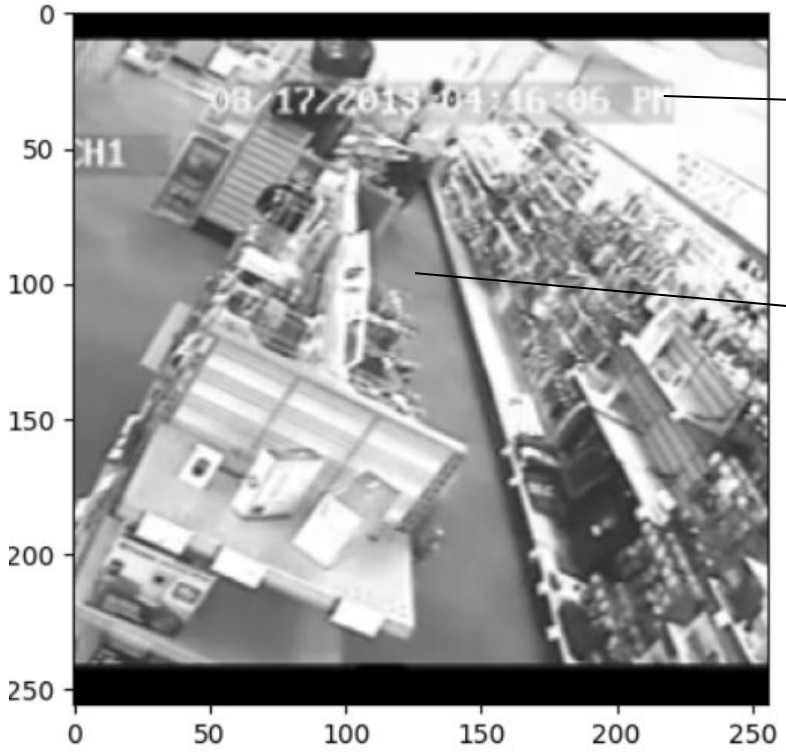
Work with established shops to gather data (Ideal solution)

Fine tuning a model is good but data is often specific to an environment in action recognition.
Working with stores owner allows you to continuously improve a model for a specific application

Action recognition specifics : Shoplifting

Data quality

In lab 2, we'll use DCSASS data, which looks like this :



Watermark

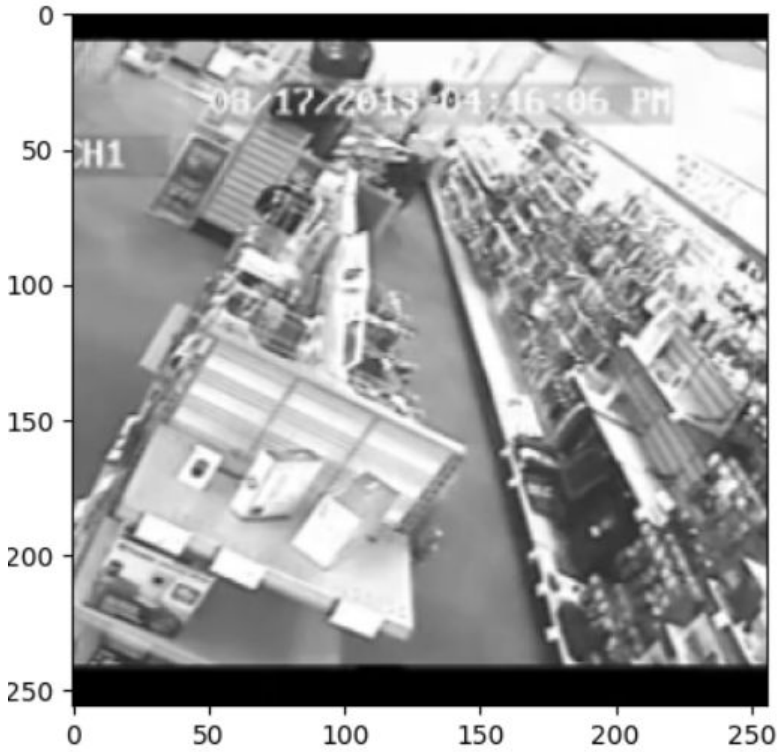
poor resolution

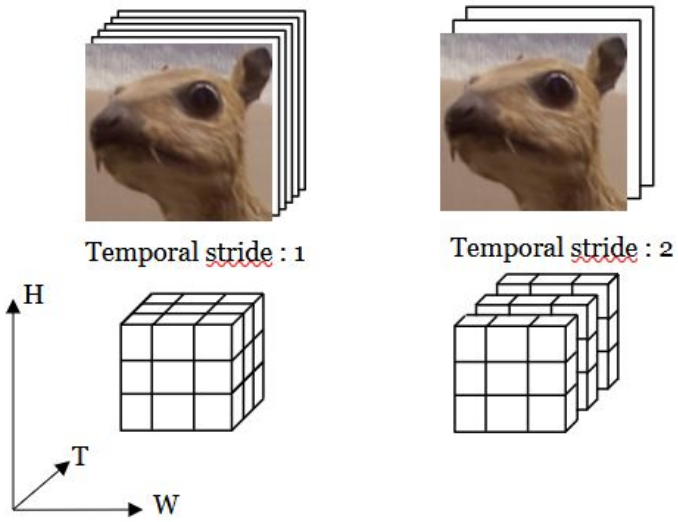
poor and variable angles

What are the effects ?

Action recognition specifics : Shoplifting

Data quality





This is why we use spatio temporal models…

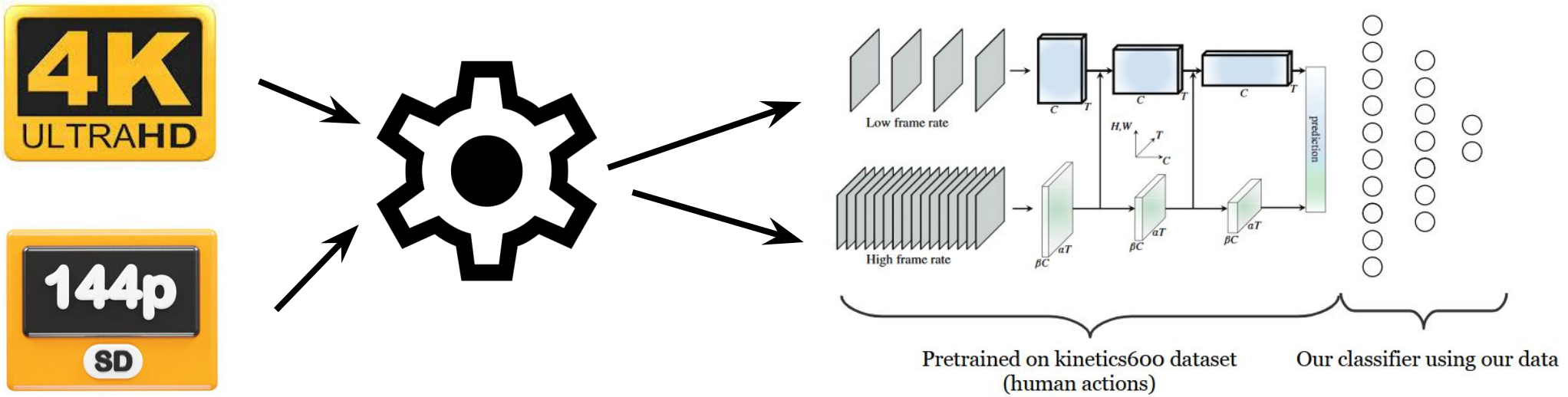
But watch out for artifact granularity ! They have to be “random” !

Action recognition specifics : Shoplifting

Data transformations for the model

Data need to be standard for each model's input

How can we make ANY video a standard one ?



Action recognition specifics : Shoplifting

Data transformations for the model

Here are the transform params we declared in Lab #1.

Parameter	Value	Description
side_size	256	Shorter side of frames resized to this size before cropping.
crop_size	256	Final spatial size (height & width) of frames after cropping.
num_frames	32	Number of frames per input video clip.
mean	[0.45, 0.45, 0.45]	Mean pixel values for RGB channels (used for normalization).
std	[0.225, 0.225, 0.225]	Standard deviation for RGB channels (used for normalization).
sampling_rate	2	Interval between selected frames in a sequence (every 2nd frame).
frames_per_second	30	Original frame rate of input video.
slowfast_alpha	4	Ratio of sampling rates between Slow and Fast pathways.
num_clips	10	Number of video clips sampled for inference.
num_crops	3	Number of spatial crops per clip (for augmentation).
clip_duration	(32 × 2) / 30 ≈ 2.13 sec	Temporal duration of each clip in seconds.

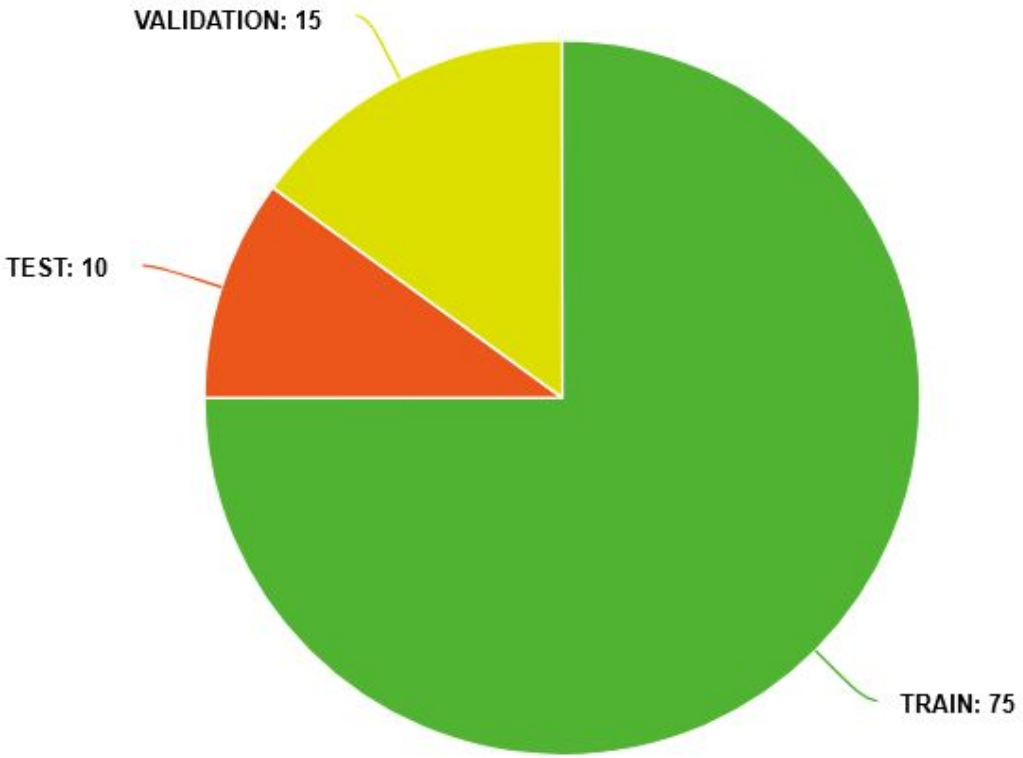
What ?
Why ?
What for ?
Are these relevant for all application (Live feed, Train, Test….)

These transforms can take computation power !

Action recognition specifics : Shoplifting

Lab #2 : Organizing the data

Here is how we'll organize DCSASS data for training and validation



10 Epochs
10 samples per batch (train)
1 sample per batch (val & test)

LAB #2

Develop a shoplifting solution using fine-tuning & naive optimization

Lab #2

 Lab #2 : Open the LAB2 folder and start with

fine_tuning_exmaple.ipynb

What you'll learn in lab 2 :

- Set up a base action recognition model
- Set up a fine tuning layer
- Prepare for training
- Run training
- Test metrics

Lab #2 bonus : Naive optimization

What you'll learn in lab 2 bonus :

- Test different thresholds and plot metrics
- Choose the best threshold depending on the application
- Run a live demo on "live" DCSASS data

Action recognition solution

Deploying an action recognition solution & case studies (test prep)

Action recognition solution

Final solution platform

Now we need to chose how to package our final solution …

Computing power is the heart of AI and its main limiting factor (with data)

Let’s explore solutions for our application

Action recognition solution

Edge devices : JETSON

For such an application, edge devices sound like a great deal

NVIDIA Jetson embedded computer are a good solution

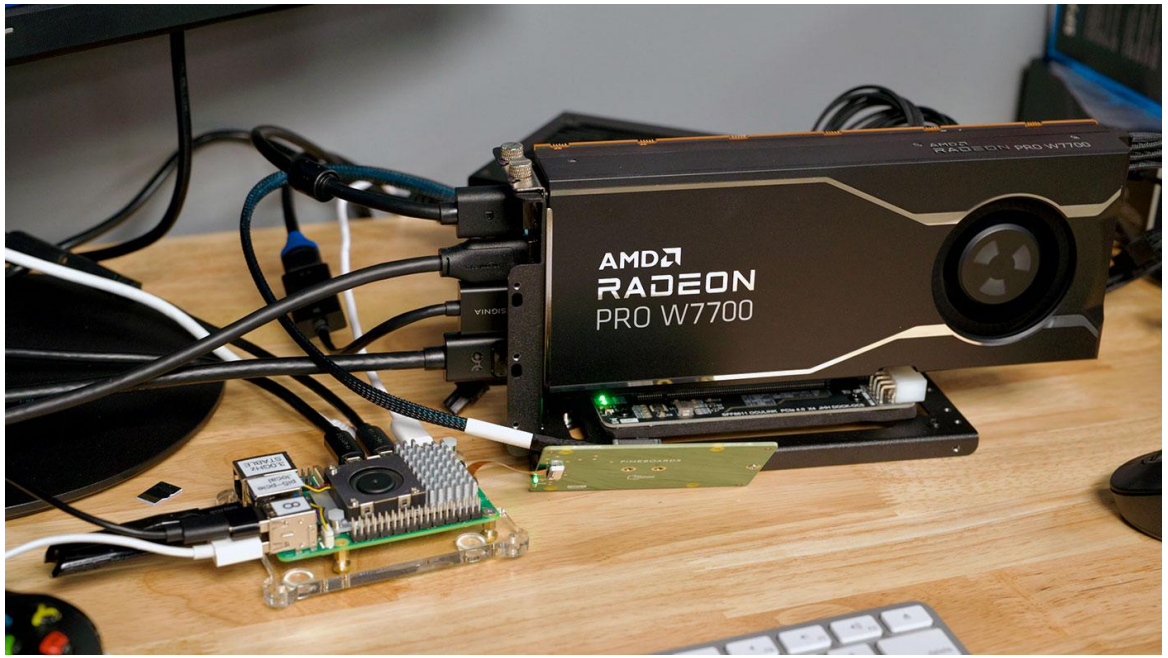
- Less trouble
- Easy to deploy and debug
- Large support
- Low time-to-market time

- 300$ - 350$ **minimum** per fully-equiped unit (Board, camera, case, ...)
- You STILL need a centralised server, at least to receive alarms

Action recognition solution

Edge devices : DIY

You can also make you OWN tiny but powerful AI edge device with a raspberry PI and an AMD GPU



- Low-medium time-to-market time
- You do not depend on NVIDIA hardware
- Very powerful

- More trouble overall…
- Still very expensive (Radeon GPU)
- Still need a server

Action recognition solution

Edge devices : DIY ++ (FPGA, ASIC)

You can also synth your AI algorithm into RTL and use an FPGA (cf. my last course)

- Very efficient
- Low and fixed delay
- Very predictable inference
- Mid to Low-cost operations

- Low time-to-market
- Many troubles
- **Expensive** development

Action recognition solution

Centralized devices : DIY ++ (FPGA, ASIC)

Centralized server are a great solution. You have two choice… #1 : DIY

- Cheap inference
- You own it and can archive the data

- Expensive deployment
- Scaling is up to you

Action recognition solution

Centralized devices : DIY ++ (FPGA, ASIC)

#2 : Rent a server to a cloud provider

- Zero cost to deploy
- No trouble
- Scaling is not your problem

- Expensive inference
- You do not own it
- You pay for scaling

Action recognition solution

BONUS : Preparing for the test : Case study for a critical application

You are approached by a city council.

They have a high criminal activity but can't monitor all the treets efficiently (in despite of all the cameras).

They need a solution and come to you to prepare a product.

Important note : crime happen at night (night vision) AND during the day which we should take into account for training.

- What model would you use ?
- How ? (train from scratch ? fine-tune ?)
- How do you get data ?
- How to you prepare and label data ?
- What classes should we use ?
- What final platform should we use ?
- What optimization metrics should we maximise one the model is trained (recall ? precision ? accuracy ?)
- What notification system would be the best ?
- How these solution would adapt depending on :
  - Urgency of the need ?
  - Budget ?
  - Market cap ?

Action recognition solution

BONUS : Preparing for the test : Case study for a critical application #2

You are approached by a bank company.

They want to catch suspect behavior before anything wrong happens.

Having good data labels is hard as robberies does not happen often.

What solution can we imagine for such a problem ?

References

References

- ResNet 50 in details : https://www.youtube.com/watch?v=mGMpHyiN5lk
- TSM paper : https://arxiv.org/pdf/2109.13227
- TSM Resources github : https://github.com/mit-han-lab/temporal-shift-module
- TSN paper : https://arxiv.org/pdf/1608.00859
- TSN Resources github : https://github.com/yjxiong/temporal-segment-networks
- SlowFast paper : https://arxiv.org/abs/1812.03982
- Kinectics dataset used on pretrained models : https://paperswithcode.com/dataset/kinetics-600
- Quo Vadis, Action Recognition? https://arxiv.org/pdf/1705.07750
- MMACTION2 Github https://github.com/open-mmlab/mmaction2
- BCE loss function explained : https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a
- DCSASS dataset : https://www.kaggle.com/datasets/mateohervas/dcsass-dataset
- TheftDetectingCCTV model (based on mmaction2 & asia data) https://github.com/tiktrimo/TheftDetectingCCTV/tree/master

Institut Supérieur de l'Aéronautique et de l'Espace

10, avenue Edouard Belin – BP 54032
31055 Toulouse Cedex 4 – France
T +33 5 61 33 80 80

www.isae-supaero.fr

isae
Institut Supérieur de l'Aéronautique et de l'Espace
SUPAERO