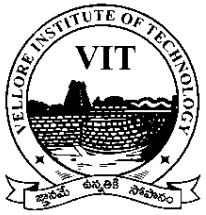



VIT-AP UNIVERSITY

# Computer Vision

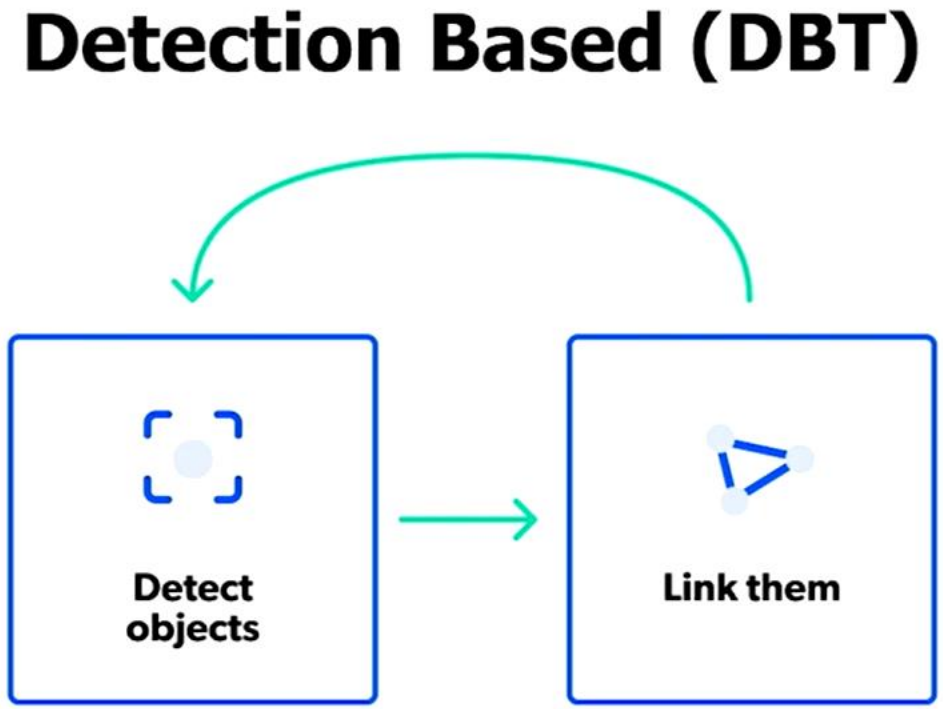
(Course Code: 4047)

## Module-4:Lecture-4: Object Tracking

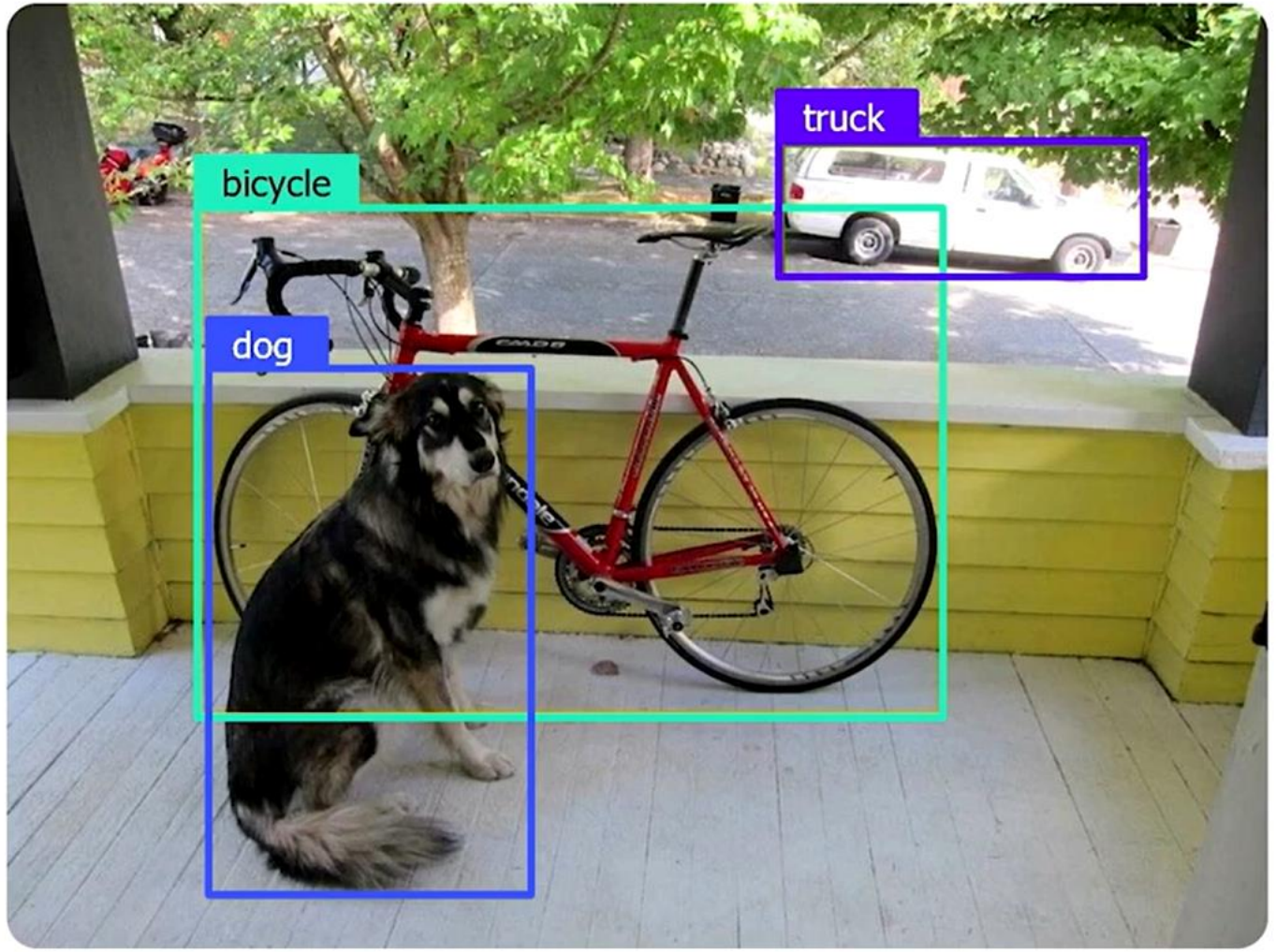
Gundimeda Venugopal, Professor of Practice, SCOPE

# Detection based Tracking

- Detect objects in **each** frame
- New objects are **automatically** discovered
- Object classes need to be **predefined**



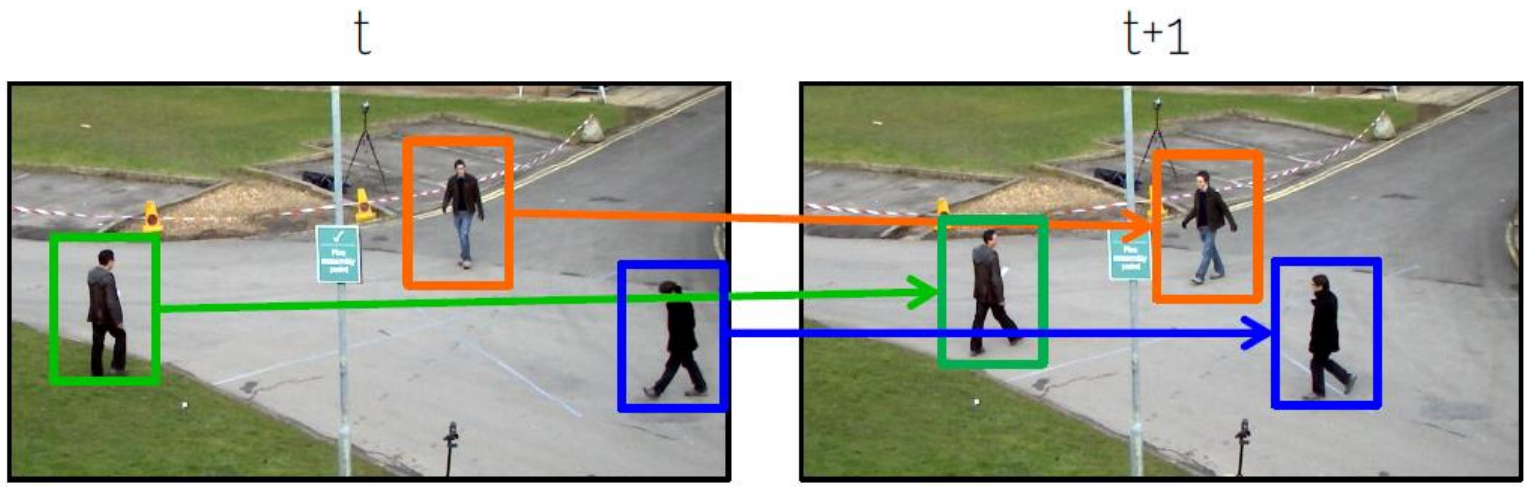
# Object Tracking using a Deep learning model (e.g., YOLO)



# Online tracking

# Problem Statement

- Given a video, find out which parts of the image depict the same object in different frames
- Often we use detectors as starting points



# Why do we need tracking?

- To model objects when detection fails:
  - Occlusions
  - Viewpoint/pose/blur/illumination variations (in a few frames of a sequence)
  - Background clutter
- To reason about the dynamic world, e.g., trajectory prediction (is the person going to cross the street?)

# Tracking is….

- Similarity measurement
- Correlation
- Correspondence
  - Story time: „A young graduate student asked Takeo Kanade what are the three most important problems in computer vision. Kanade replied: “Correspondence, correspondence, correspondence!”
- Matching/retrieval
- Data association

# Tracking is also …

- Learning to model our targets

- Appearance: we need to know how the target looks like
  - Single object tracking
  - Re-identification

- Motion: to make predictions of where the targets goes
  - Trajectory prediction

# Single Object Tracking

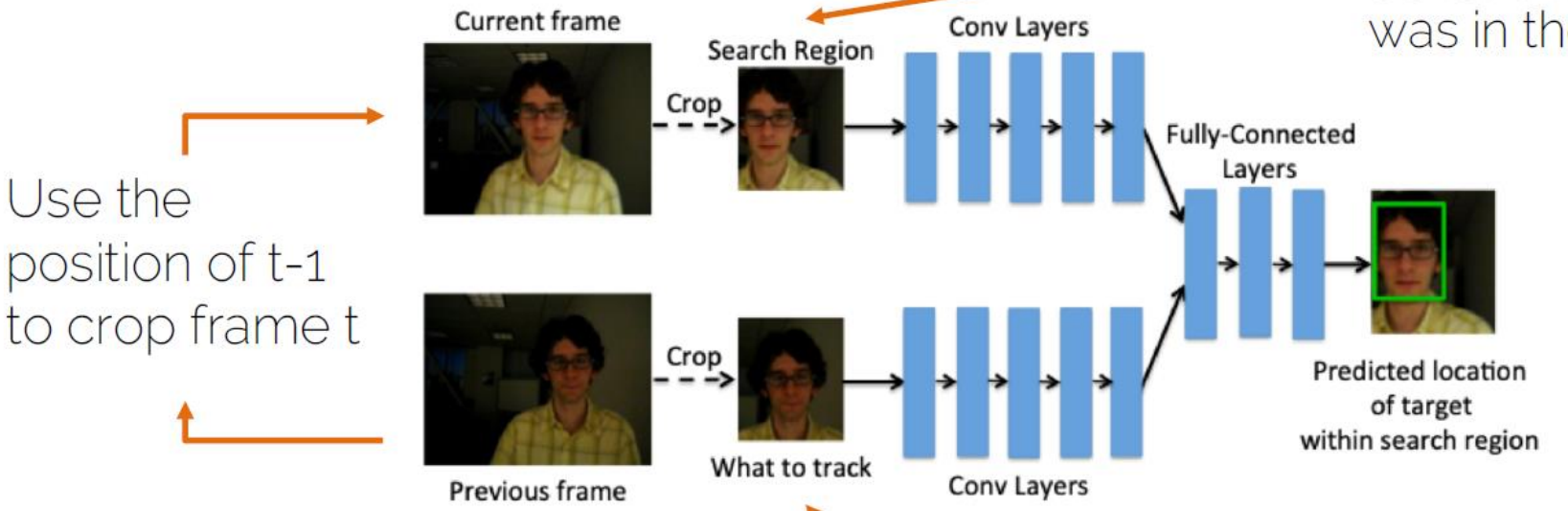
- STT (1) as a matching/correspondence problem:
  - GOTURN: no online appearance modeling
- STT (2) as an appearance learning problem:
  - MDNet: quick online finetuning of the network
- STT (3) as a (temporal) prediction problem:
  - ROLO = CNN + LSTM

# Single Object Tracking 1

- Architecture: conv + concatenate + FC
- Input: what to track?
- Where do I search for the object?

Check the original paper for the exact parameterization of the output

Assume smooth "slow" motion. The object cannot be far away from where it was in the previous frame.




Crop the object to be tracked – Initialization of our tracker

Siamese Neural Network
Both the Conv layers are same
Their weights are same and they extract same features

D. Held, S. Thrun, S. Savarese. "Learning to Track at 100 FPS with Deep Regression Networks". ECCV 2016.

# Single Target Tracking 1

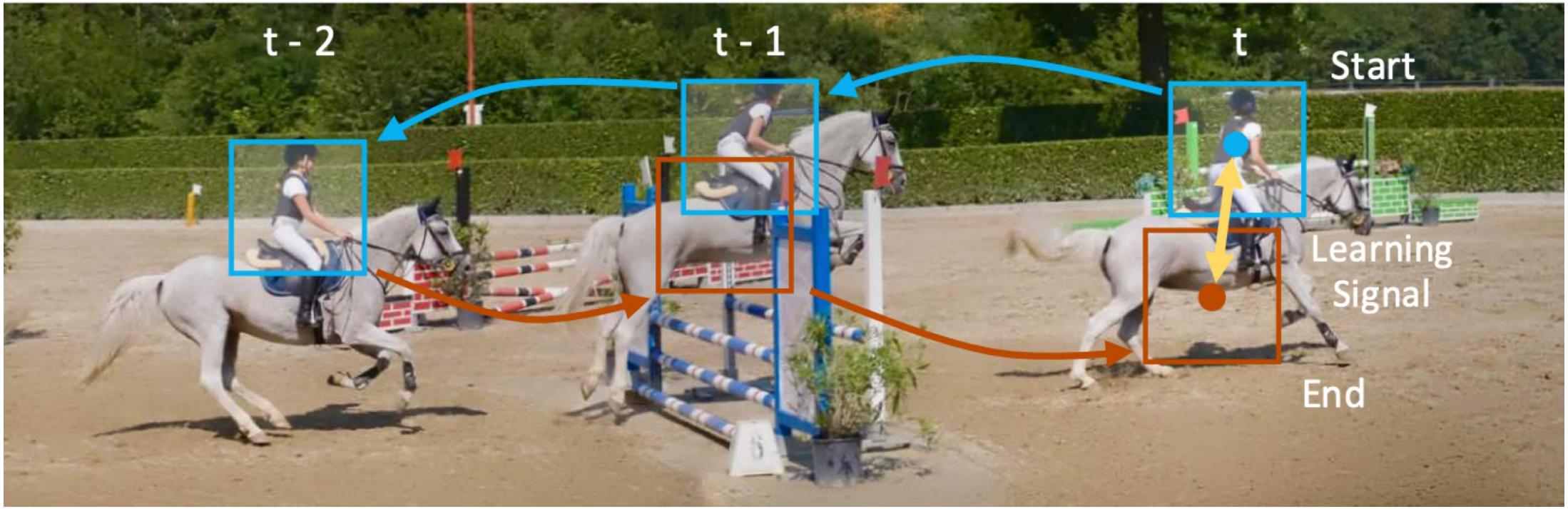
- PROS of GOTURN:
  - No online training required.
  - Tracking is done by comparison, so we do not need to retrain or finetune our model for every new object.
  - Close to the template matching approach that we saw in the first lectures for object detection
  - This makes it very fast!

- CONS:
  - We have a motion assumption. If the object moves fast and goes out of our search window, we cannot recover.

D. Held, S. Thrun, S. Savarese. "Learning to Track at 100 FPS with Deep Regression Networks". ECCV 2016.

# Single Object Tracking 1.2 - Unsupervised

❖ Forward cycle and backward cycle should be consistent!



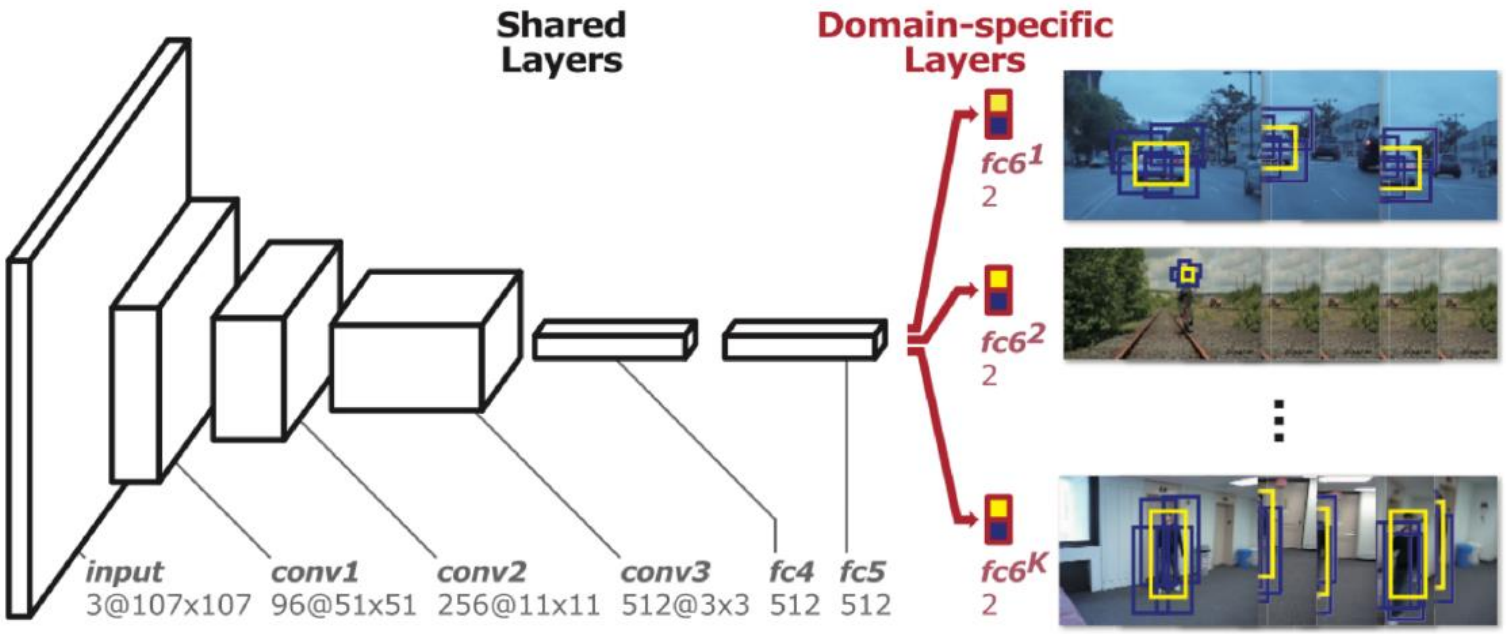

X. Wang, A. Jabri, A. Efros. “Learning correspondence from the cycle-consistency of time”. CVPR 2019

# Single Object Tracking 2

- Online appearance model learning entails training your CNN at test time.
  - Slow: not suitable for real-time applications
  - Solution: train as few layers as possible

# Single Target Tracking 2

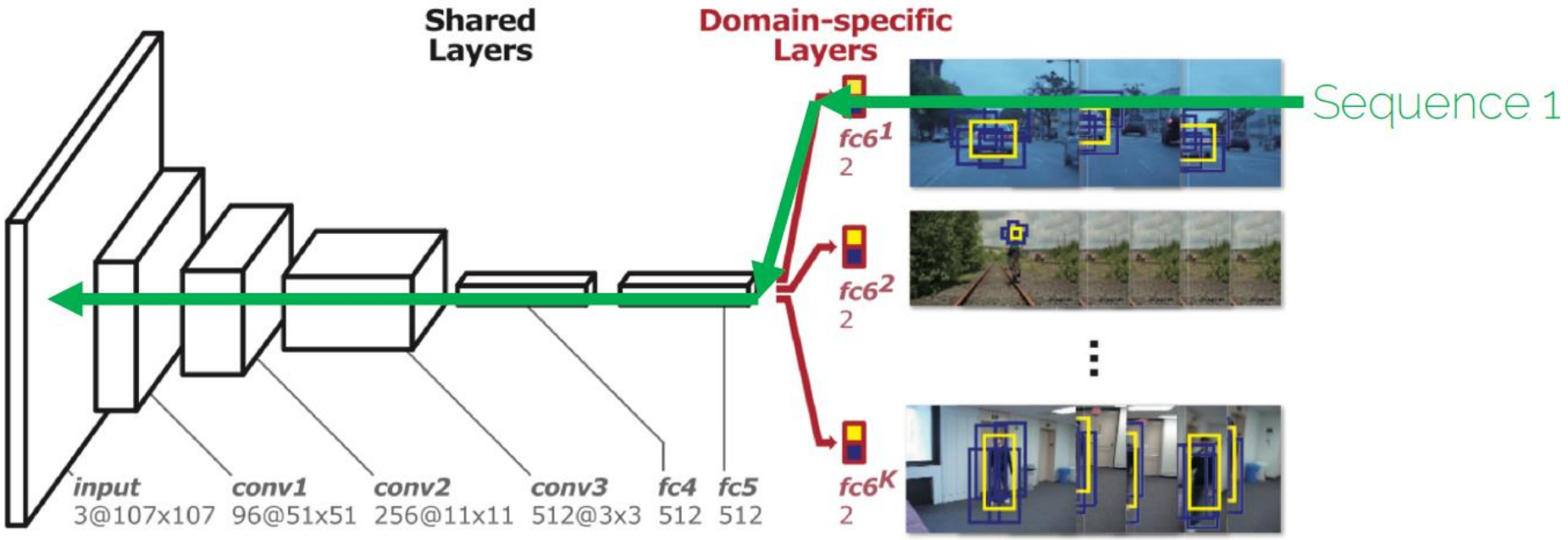
- Shared layers + scene-specific layers




H. Nam and B. Han. „Learning Multi-Domain Convolutional Neural Networks for Visual Tracking". CVPR 2016

# Single Target Tracking 2

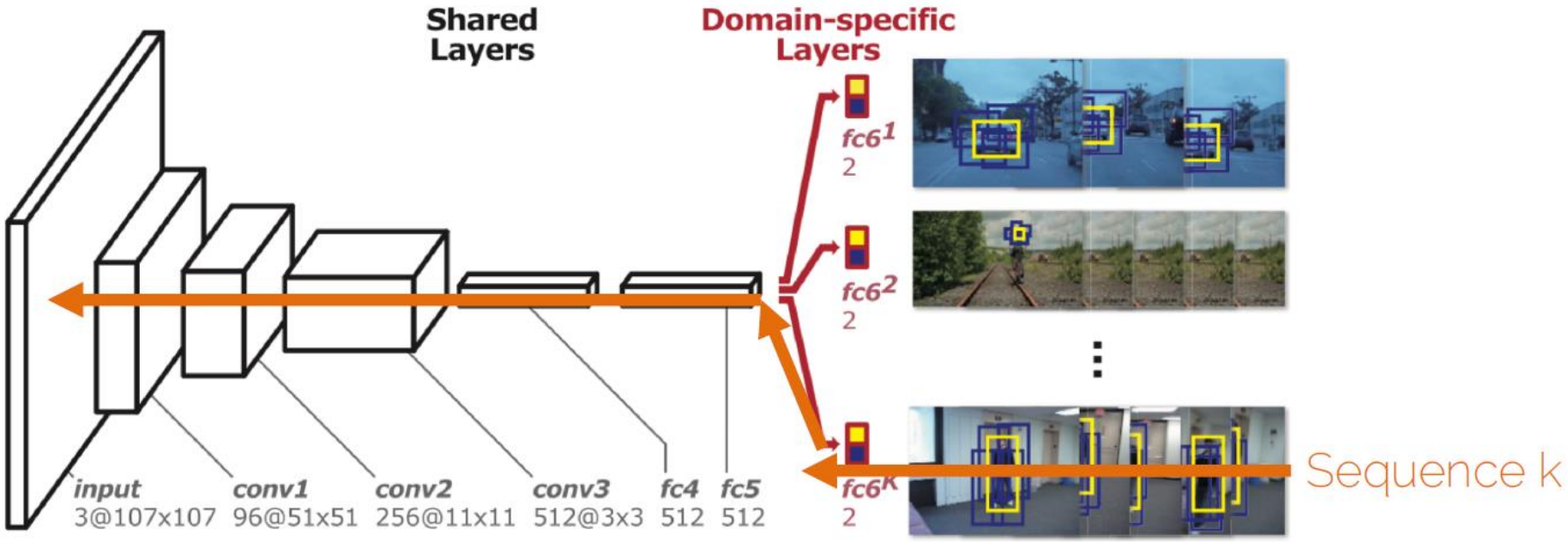
- Backpropagation is independent per sequence




H. Nam and B. Han. „Learning Multi-Domain Convolutional Neural Networks for Visual Tracking". CVPR 2016

# Single Target Tracking 2

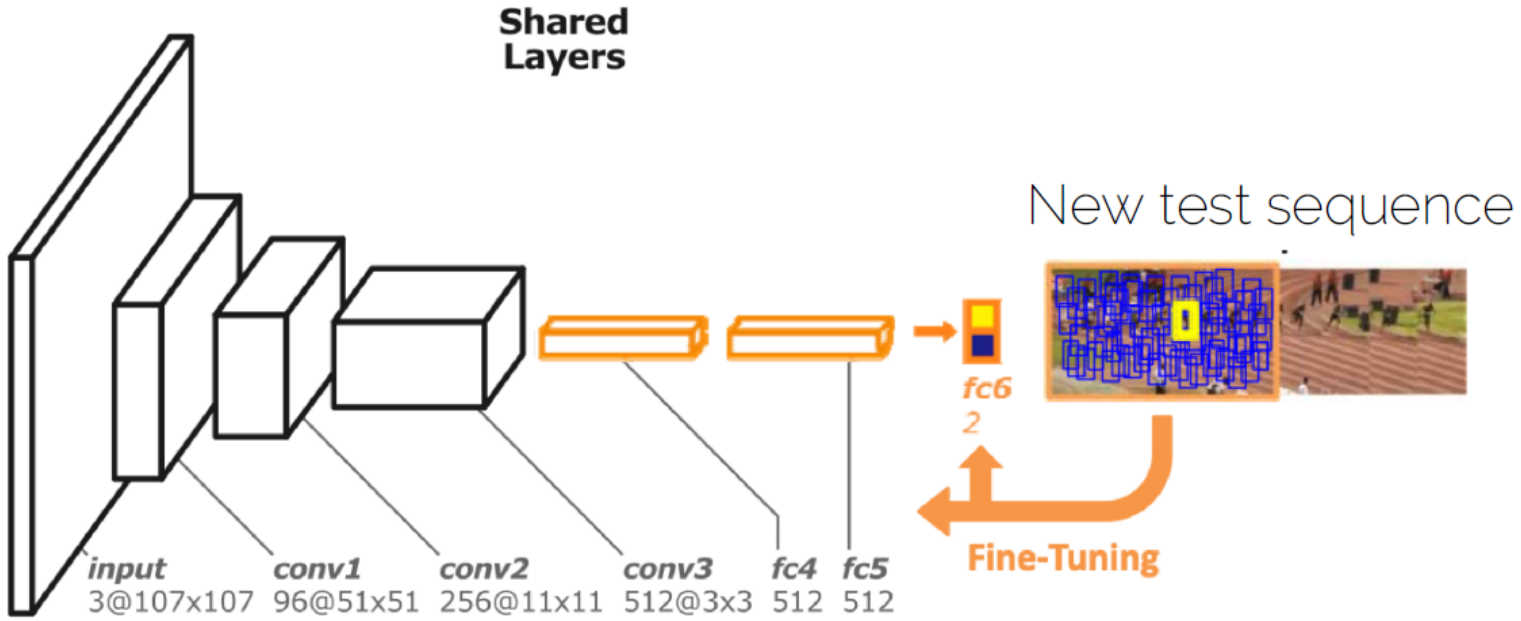
- Backpropagation is independent per sequence




H. Nam and B. Han. „Learning Multi-Domain Convolutional Neural Networks for Visual Tracking". CVPR 2016

# Single Target Tracking 2

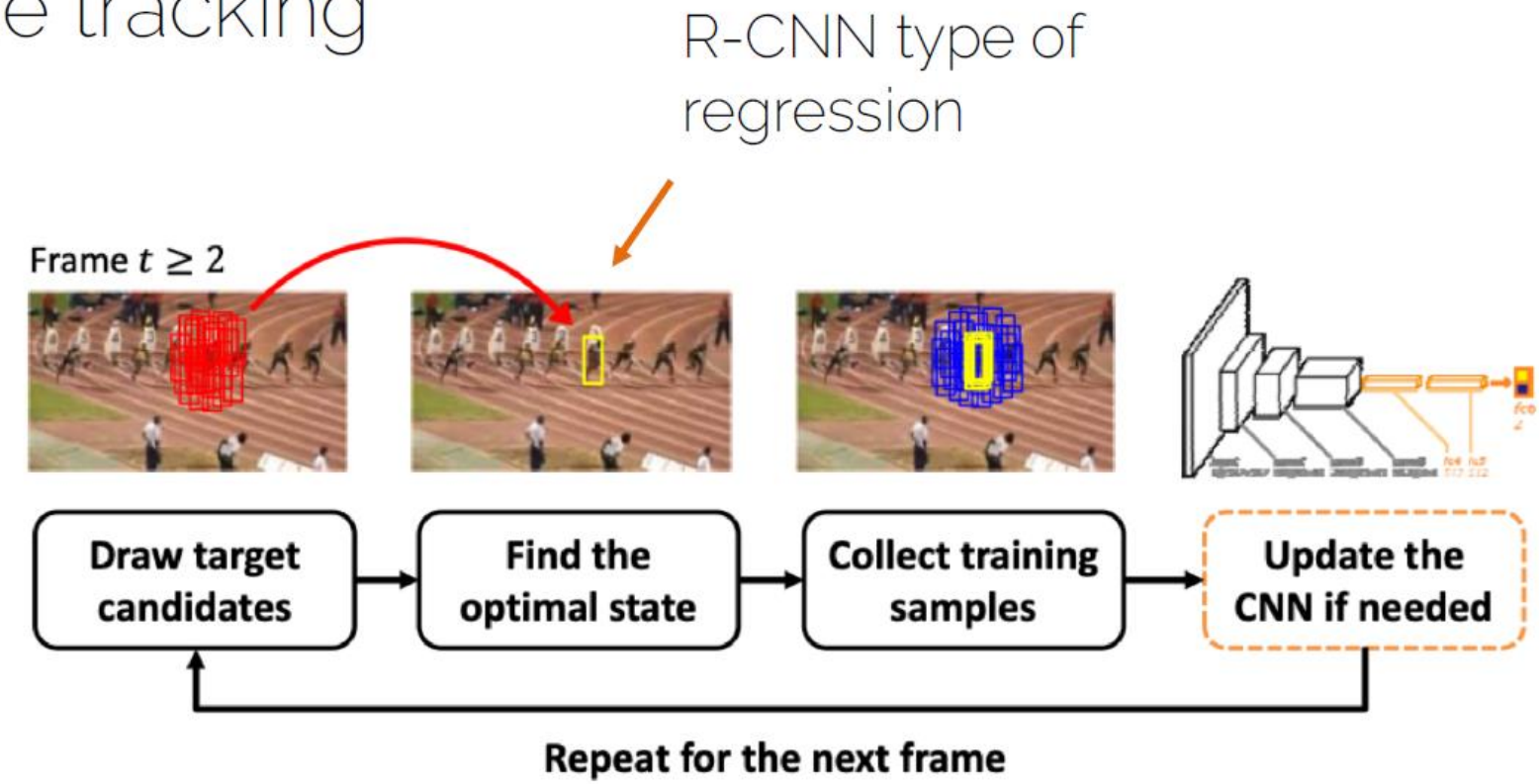
- At test time, we need to train fc6 (up to fc4 if wanted).




H. Nam and B. Han. „Learning Multi-Domain Convolutional Neural Networks for Visual Tracking". CVPR 2016

# Single Target Tracking 2

- Online tracking




H. Nam and B. Han. „Learning Multi-Domain Convolutional Neural Networks for Visual Tracking". CVPR 2016

# Single Target Tracking 2

❖ PROS of MDNet:

- No previous location assumption, the object can move anywhere in the image
- Fine-tuning step is comparatively cheap
- Winner of the VOT Challenge 2015 (http://www.votchallenge.net)

VOT: Visual object Tracking Challenge

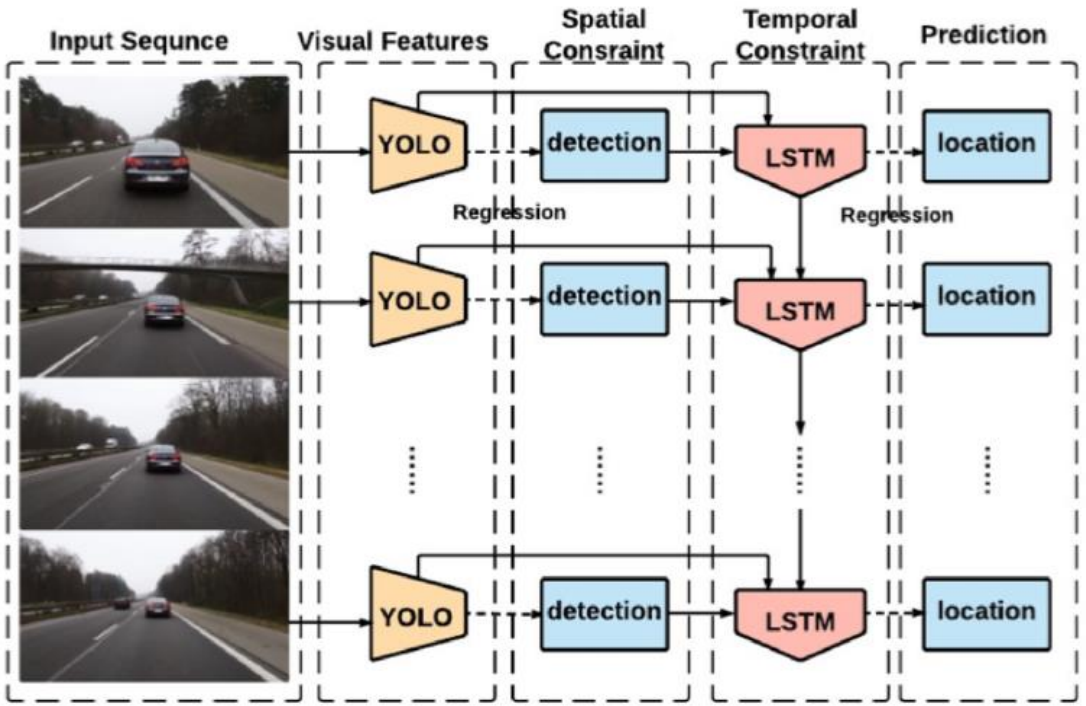
❖ CONS:

- Not as fast as GOTURN

H. Nam and B. Han. „Learning Multi-Domain Convolutional Neural Networks for Visual Tracking“. CVPR 2016

# Single Target Tracking 3

- CNN for appearance + LSTM for motion




Recurrent YOLO

ROLO

G. Ning, Z. Zhang, C. Huang, Z. He. „Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking". arXiv:1607.05781. 2016

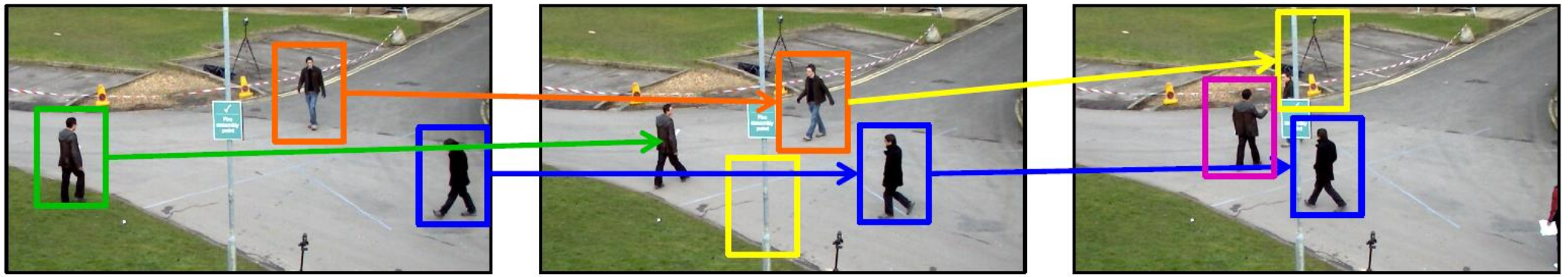
# Multiple Object Tracking

# MOT Challenges

- Multiple objects of the same type
- Heavy occlusions
- Appearance is often very similar

# Tracking by Detection

- We will focus on algorithms where a set of detections is provided
  - Remember detections are not prefect!

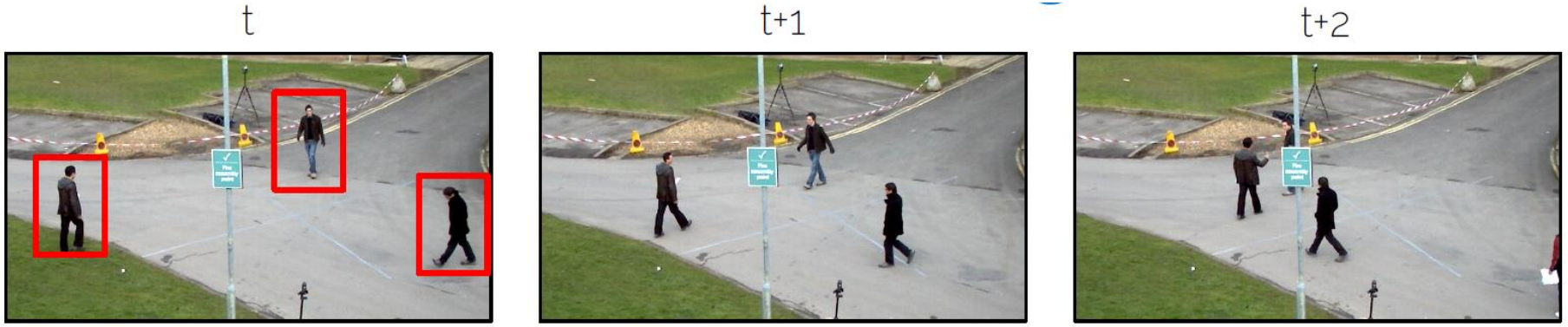


Find detections that match and form a trajectory

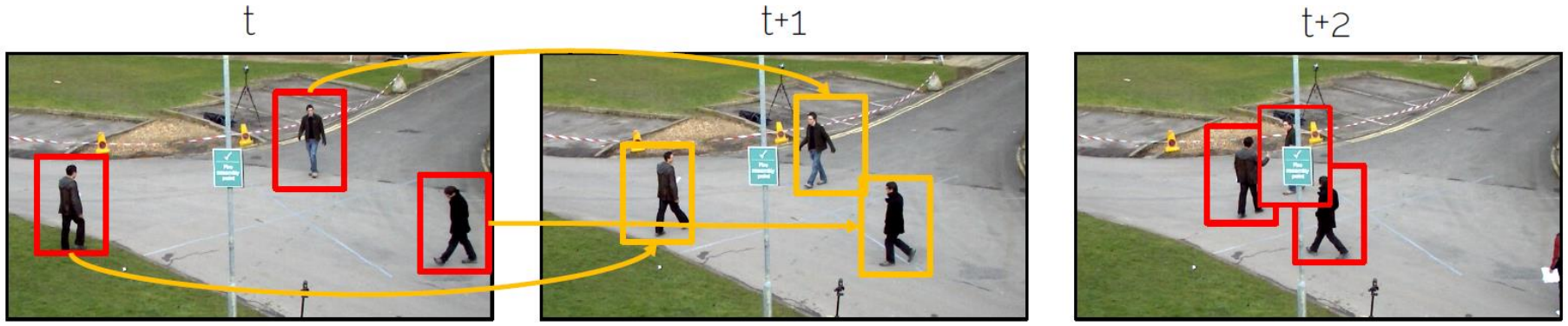
# Online vs offline tracking

- Online tracking
  - Processes two frames at a time
  - For real-time applications
  - Prone to drifting → hard to recover from errors or occlusions
- Offline tracking
  - Processes a batch of frames
  - Good to recover from occlusions (short ones as we will see)
  - Not suitable for real-time applications
  - Suitable for video analysis

# Online tracking

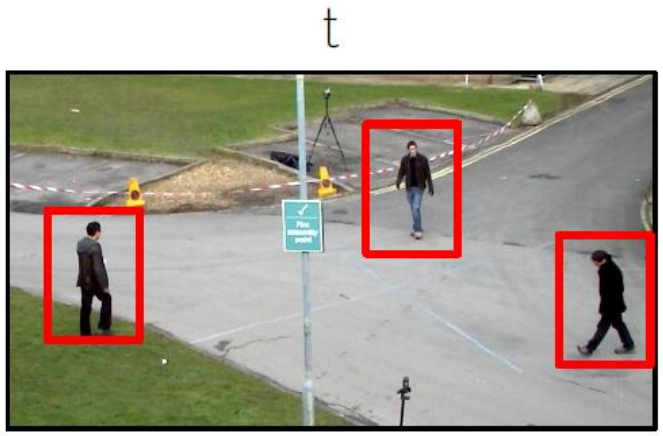


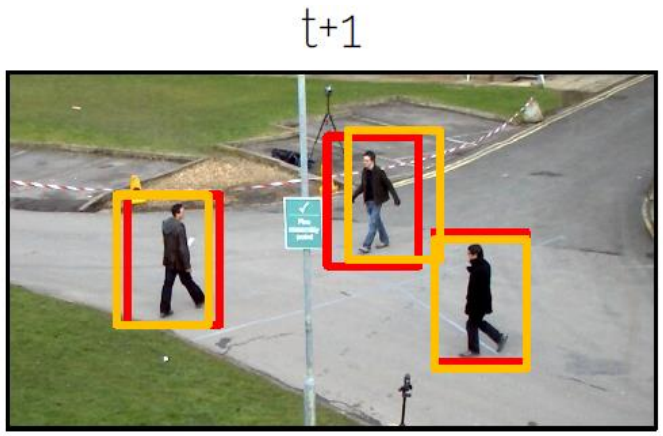

- 1. Track **initialization** (e.g. using a detector)

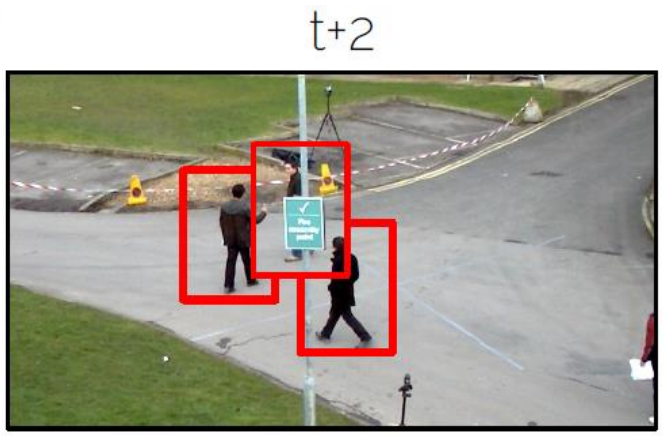



- 2. **Prediction** of the next position (motion model) →

# Online Tracking

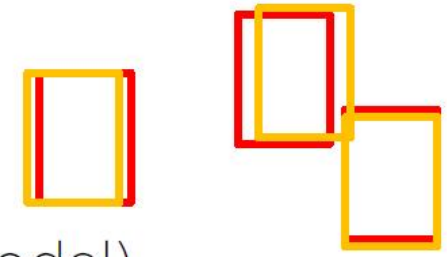



- 1. Track initialization (e.g. using a detector)

- 2. Prediction of the next position (motion model)



- 3. **Matching** predictions with detections (appearance model)

# Online Tracking

- 2. Prediction of the next position (motion model)
  - Classic: Kalman filter
  - Nowadays: Recurrent architecture
  - For now: we will assume a constant velocity model (spoiler alter: it works really well at high framerates and without occlusions!)

# Online tracking

- Bipartite matching
  - Define distances between boxes

    (e.g., IoU, pixel distance, 3D distance)
  - Solve the unique matching with e.g., the Hungarian algorithm*
  - Solutions are the unique assignments that minimize the total cost

Predictions

From motion model from time t

Detections

| | | | |
|---|---|---|---|
| 0.9 | 0.8 | 0.8 | 0.1 |
| 0.5 | 0.4 | 0.3 | 0.8 |
| 0.2 | 0.1 | 0.4 | 0.8 |
| 0.1 | 0.2 | 0.5 | 0.9 |

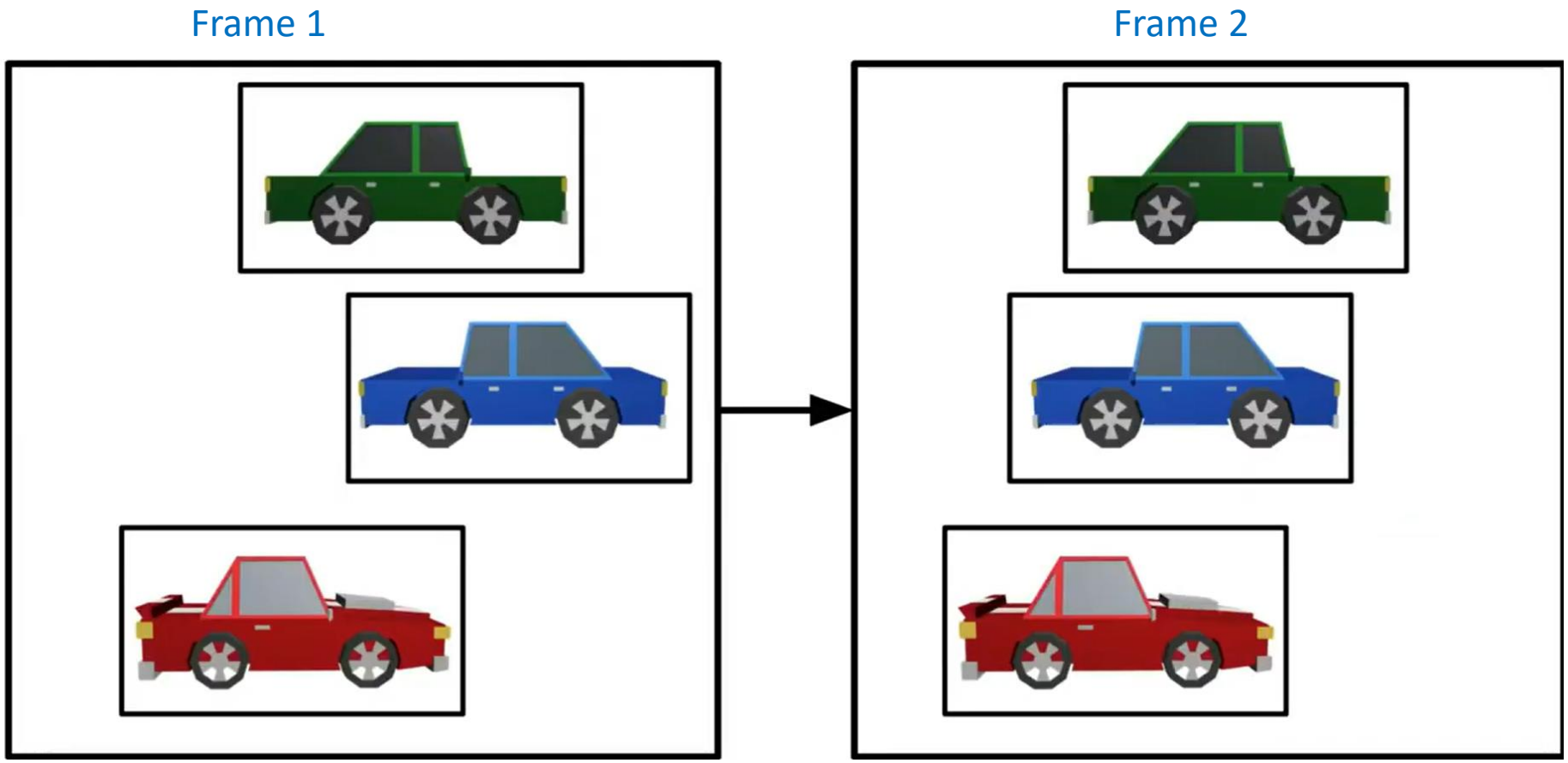
at time t+1

*Demo: http://www.hungarianalg

# Hungarian Matching Algorithm

# Object Tracking:  Matching Objects across Frames

The Hungarian Algorithm is also named differently: **Bipartite Graph Matching**. The idea of Bipartite Graph Matching is to build a graph with distances, and to assign nodes from one side of the graph to the other.

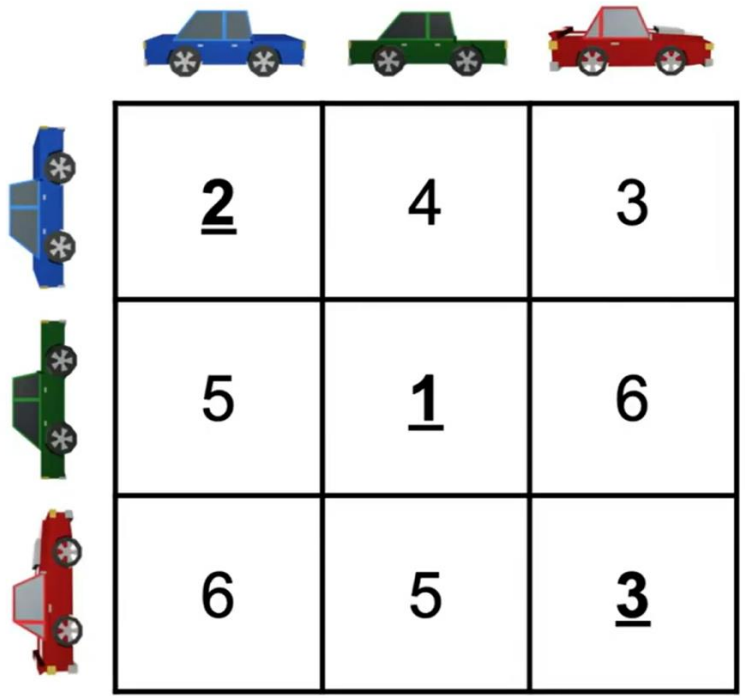



Tracking the bounding boxes across frames by matching across frames

# Hungarian Algorithm: Cost Matrix

Frame2

Frame1



| | | |
|---|---|---|
| **2** | 4 | 3 |
| 5 | **1** | 6 |
| 6 | 5 | **3** |

The cost is based on:

- distance between the centers of the boxes
- the overlap between the boxes
- the similarity of the features

etc…

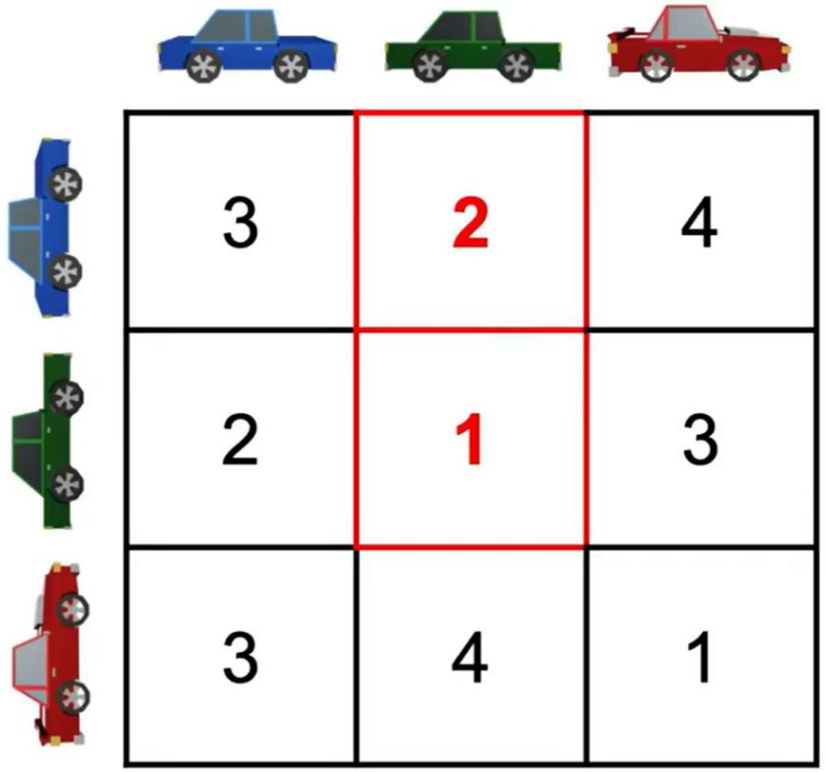
IoU (Need: max IoU)
Pixel Distance (Need: min. Euclidian distance)
3D distance

Hungarian Algorithm assigns a cost to each pair of bounding boxes in consecutive frames.

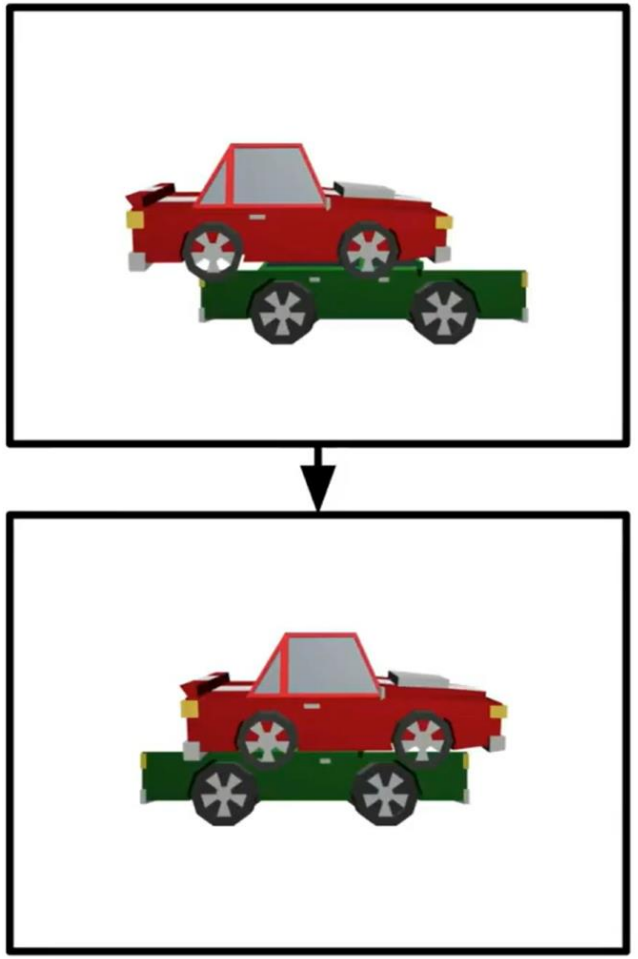
The cost is based multiple factors that help us measure the similarity or dissimilarity between the objects.

By analysing these costs, the algorithm aims to find the best possible matches between the bounding boxes that reduce the global assignment cost enabling accurate object tracking across frames.

# Can we  take the min cost directly?



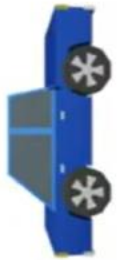

Suboptimal  assignments



# Hungarian Algorithm: Step 1

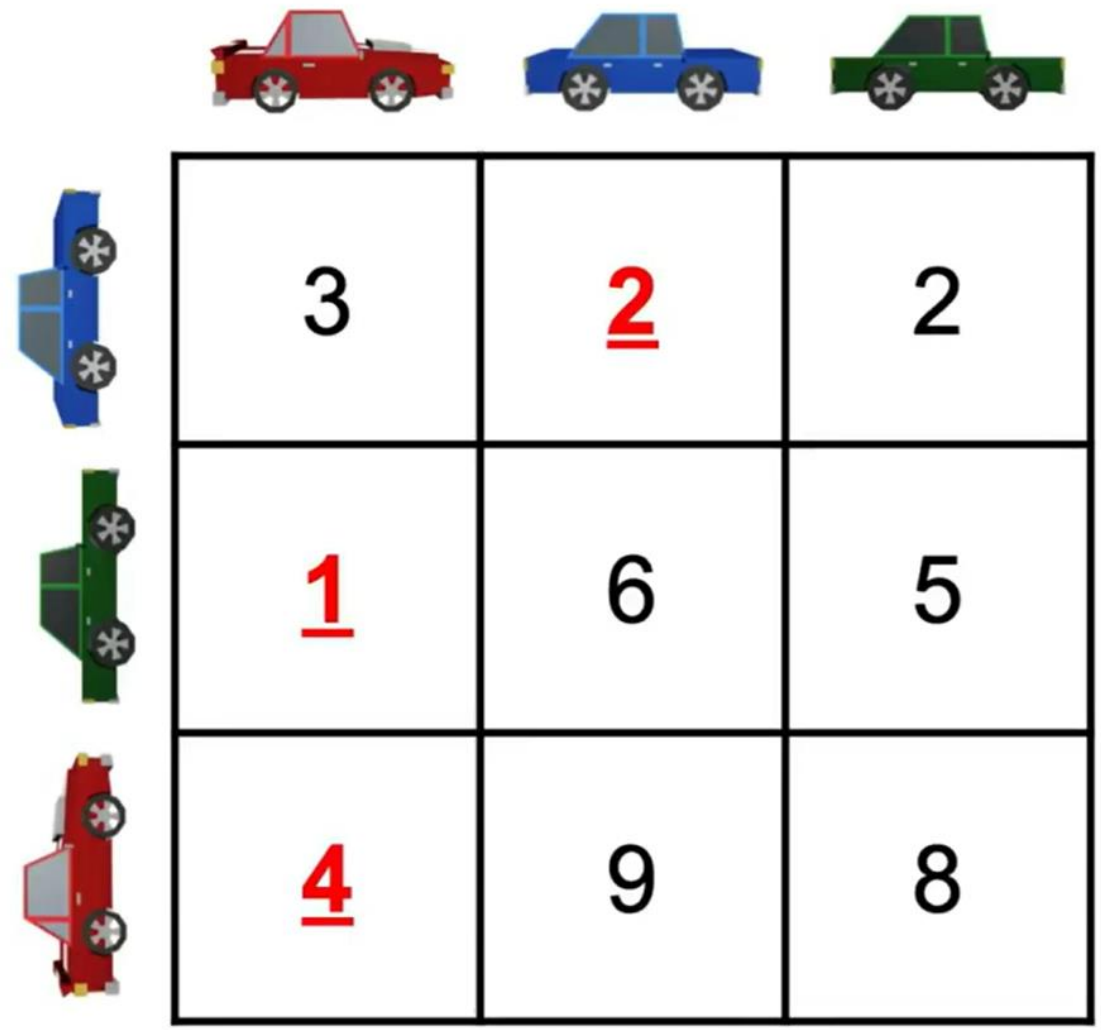
## Step 1

Calculate the cost matrix.



| | | |
|---|---|---|
| 3 | 2 | 2 |
| 1 | 6 | 5 |
| 4 | 9 | 8 |

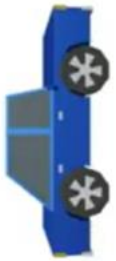
# Hungarian Algorithm: Step 2-1



| | | |
|---|---|---|
| 3 | **2** | 2 |
| **1** | 6 | 5 |
| **4** | 9 | 8 |

Find the lowest cost entry in each row and subtract it from all other entries in that row.

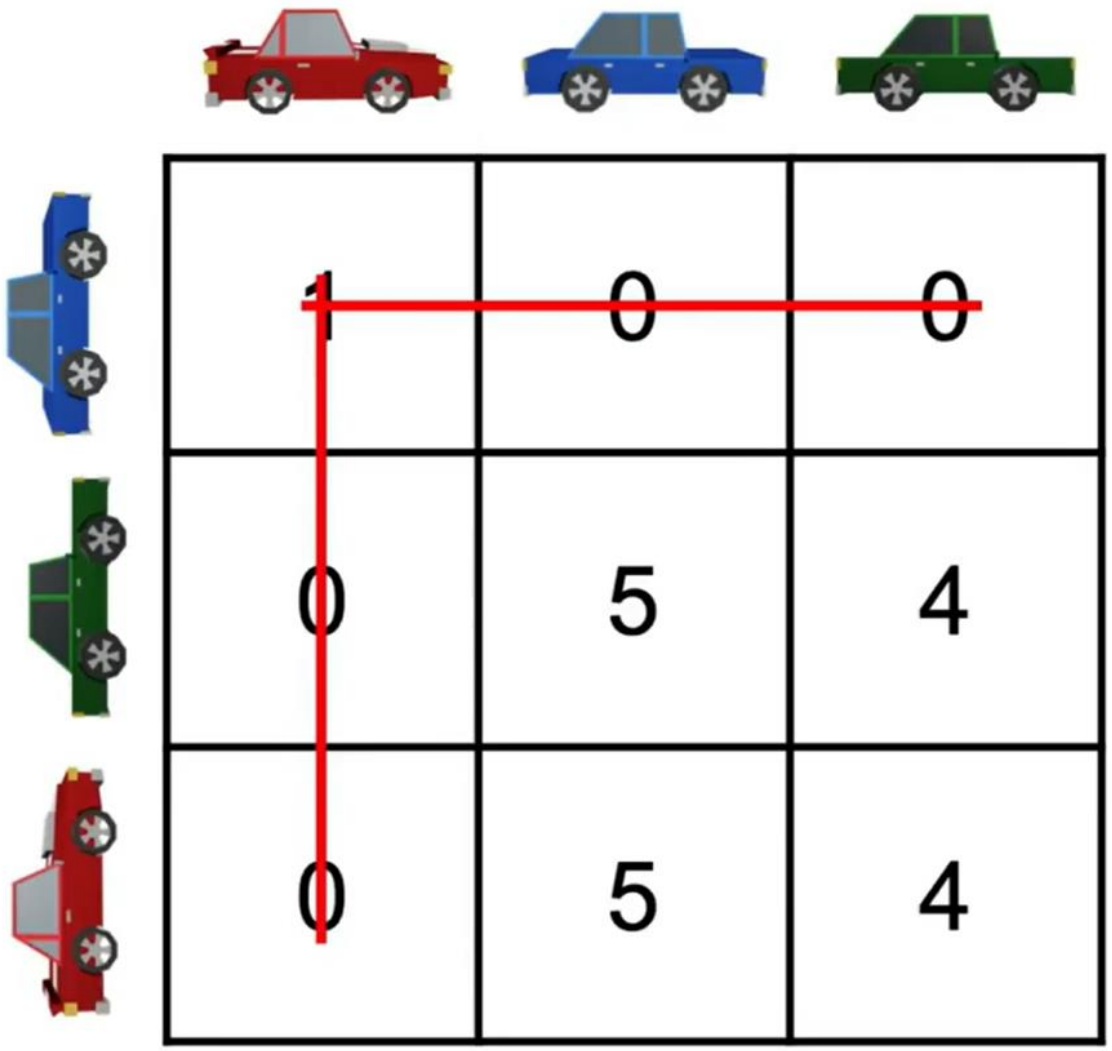
# Hungarian Algorithm: Step 2-2



| | | |
|---|---|---|
| 1 | **0** | **0** |
| **0** | 5 | 4 |
| 0 | 5 | 4 |

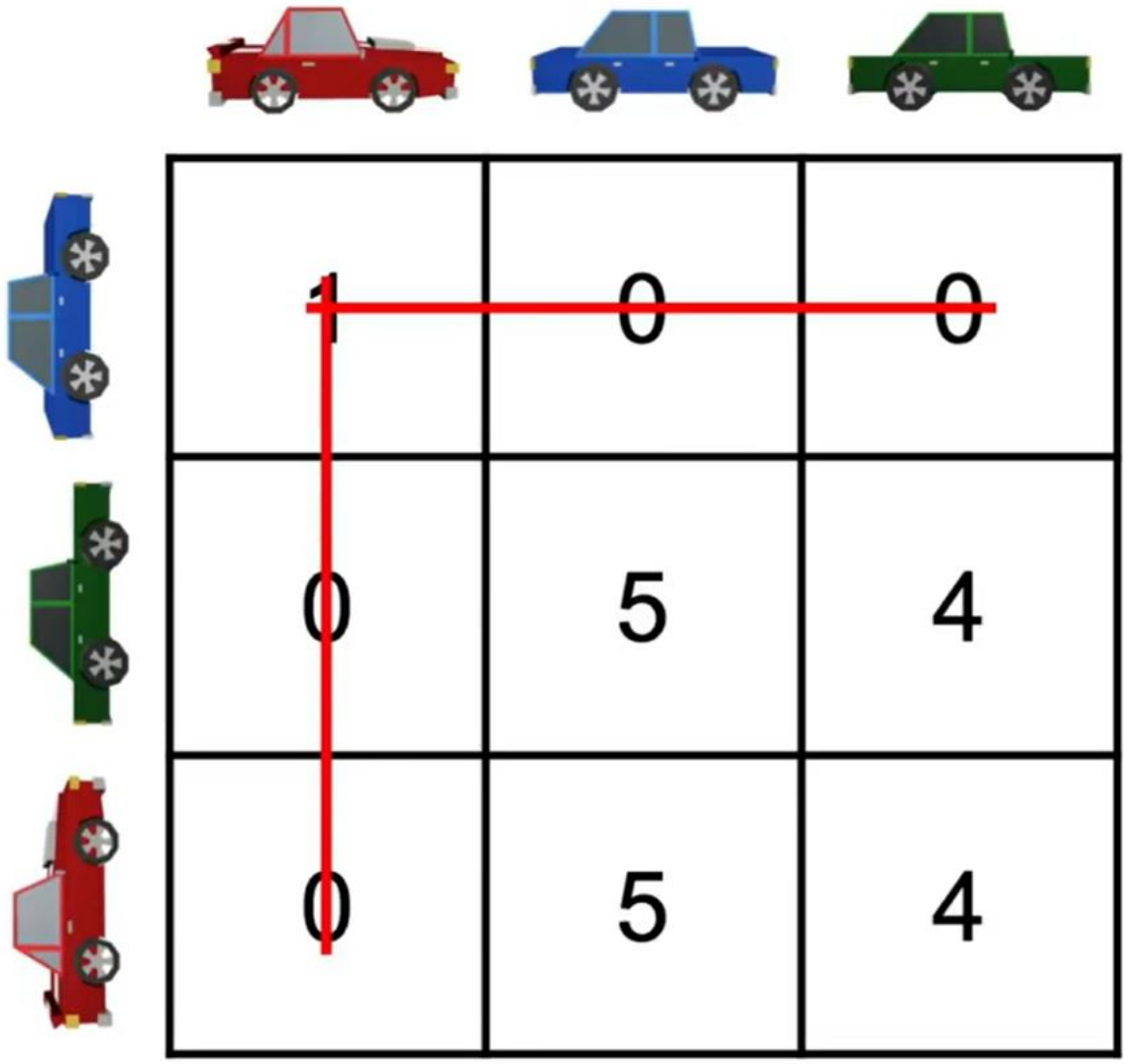
Find the lowest cost entry in each column and subtract it from all other entries in that column.

# Hungarian Algorithm: Step 2-3



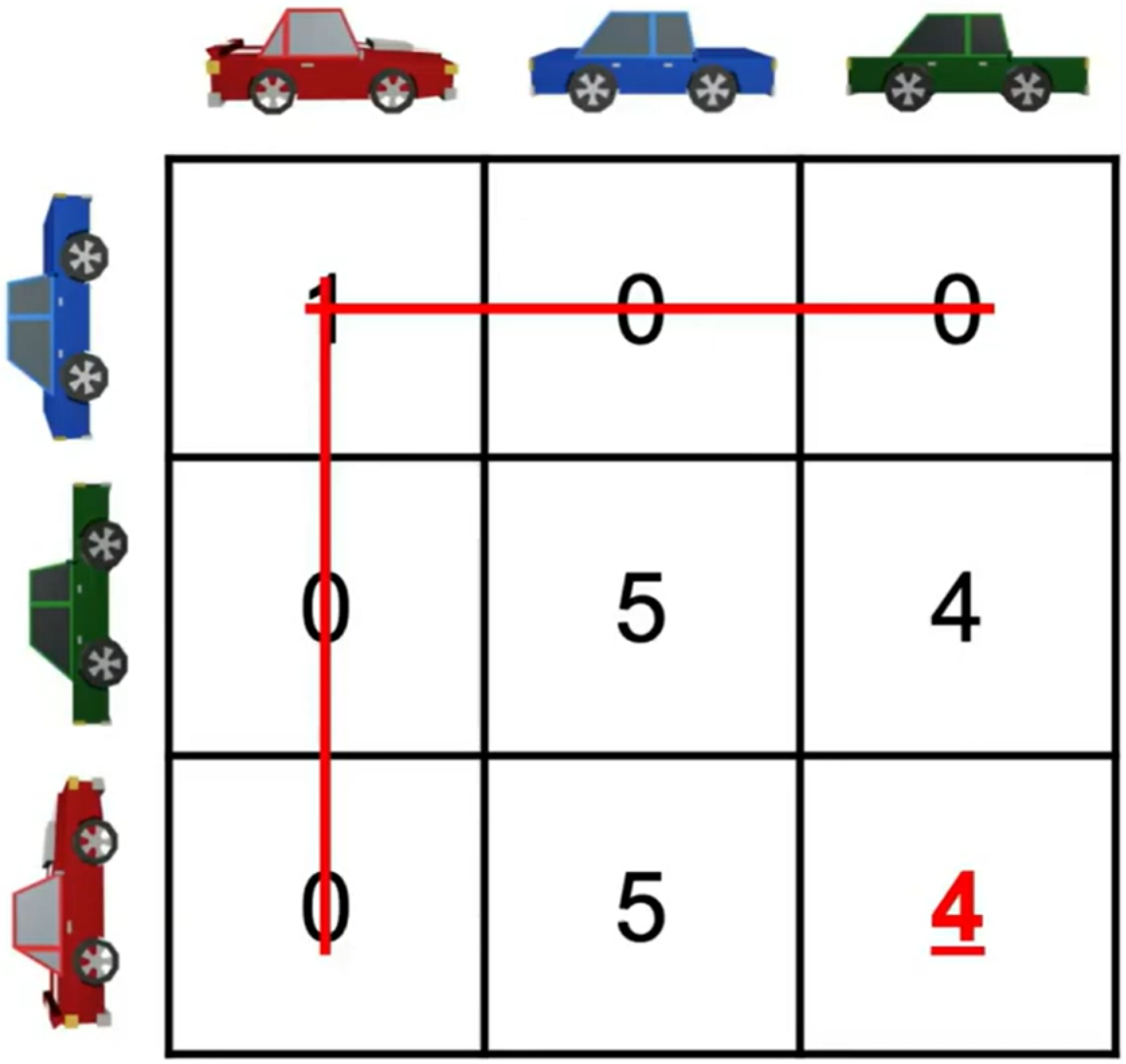

Cover the matrix with the minimum number of lines (horizontal or vertical) so that all zeros are covered.

# Hungarian Algorithm: Step 2-4



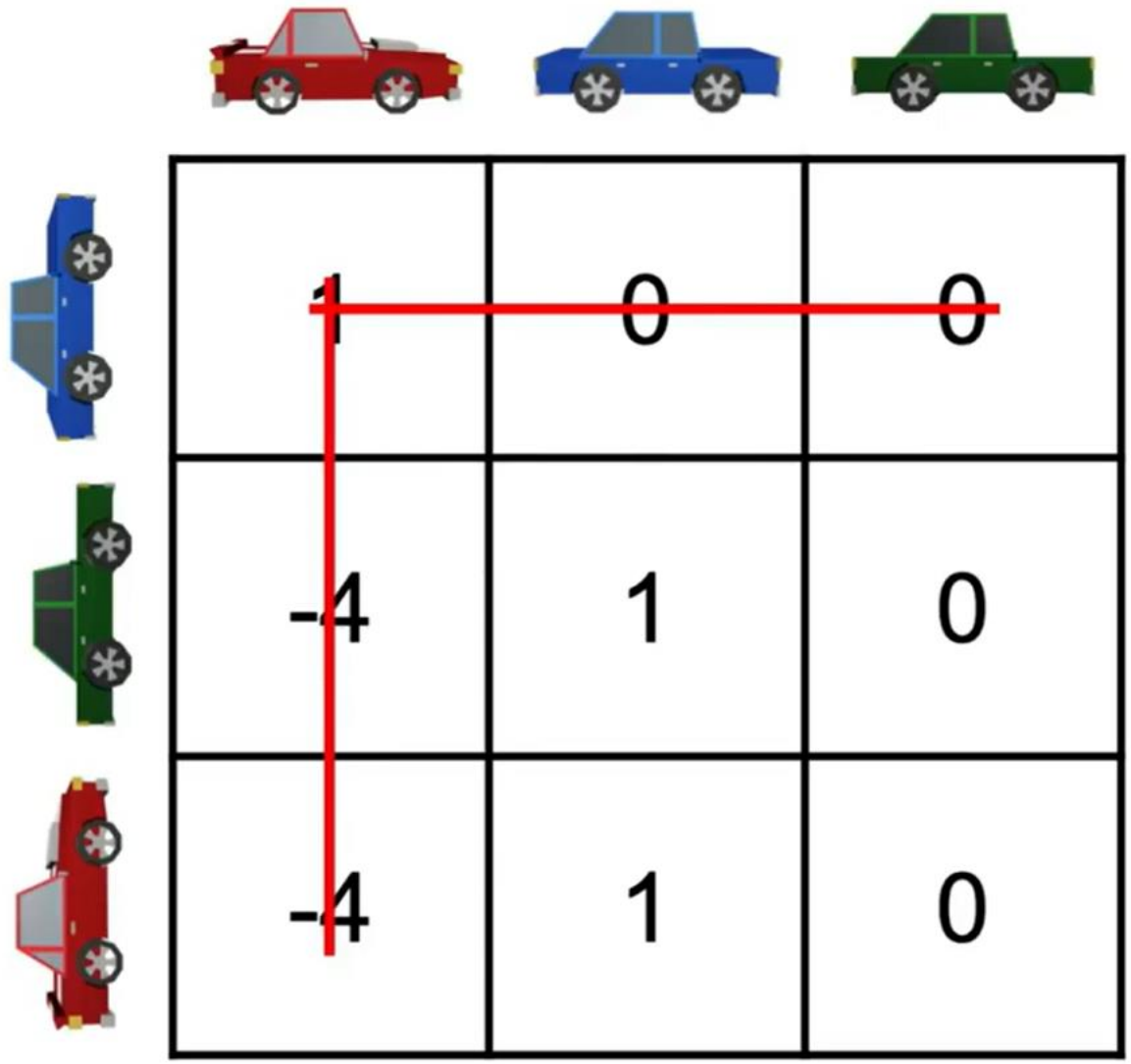

If there are n lines drawn, an optimal assignment of zeros is possible and the algorithm is finished.

# Hungarian Algorithm: Step 2-5 (a)



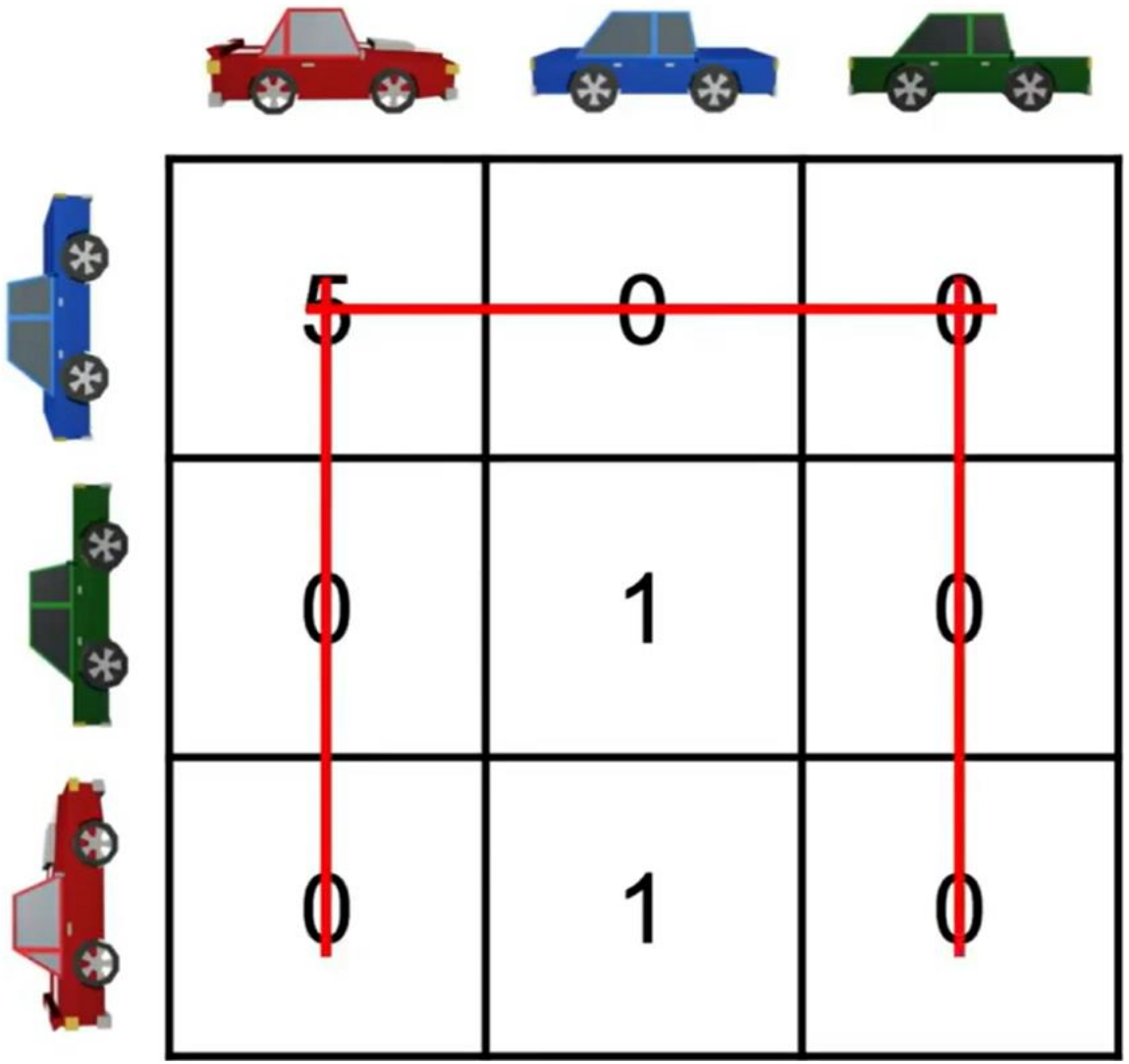

Find the smallest uncovered entry in the matrix and subtract it from all other uncovered rows that are not crossed…

# Hungarian Algorithm: Step 2-5(b)



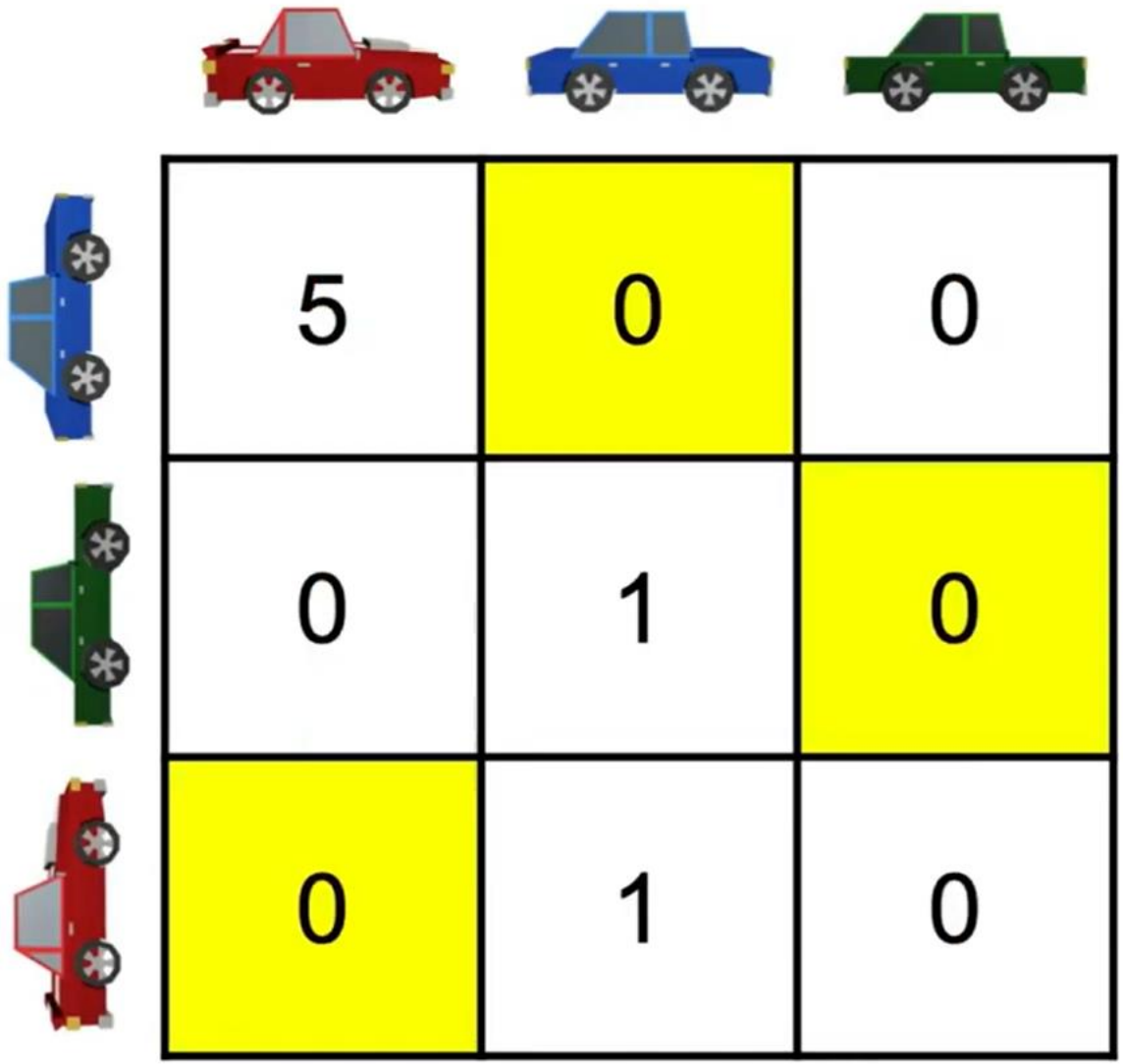

… and then add it to each column that is crossed out.

# Hungarian Algorithm: Step 2-5 (c)




Go back to step 3 of the Hungarian Algorithm.

# Hungarian Algorithm: Step 3



| | | |
|---|---|---|
| 5 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |

The assignment will be where the 0's are in the matrix such that only one 0 per row and column is part of the assignment.

# The 5 Steps of the Hungarian Matching Algorithm

Assume the cost matrix that has n lines and n columns (say n = 3 ), here's the pipeline of steps :

1. **Subtract the minimum from every element of each row** (this will make the smallest entry in the row now equal to 0).
2. **Subtract the minimum from every element of each column** (this will make the smallest entry in the row now equal to 0).
3. **Cross the 0s with the minimum number of lines needed**. If the number of lines is less than *n* (n=3), continue to step 4; otherwise, the optimal number of zeros has been reached and we can start pairing (go to step 5).
4. **Find the smallest entry not covered by any line, and subtract this entry to the entire matrix.** If an element has been covered by any line twice, add it to the place where it’s double crossed. Then, go back to Step 3.
5. **Assign Object in one frame to Object in another frame starting with the line with only one zero!** Every time we're matching object in one frame with an object in another frame, we cross its row and column to make it unavailable.

# Disadvantages

$O(N^3)$

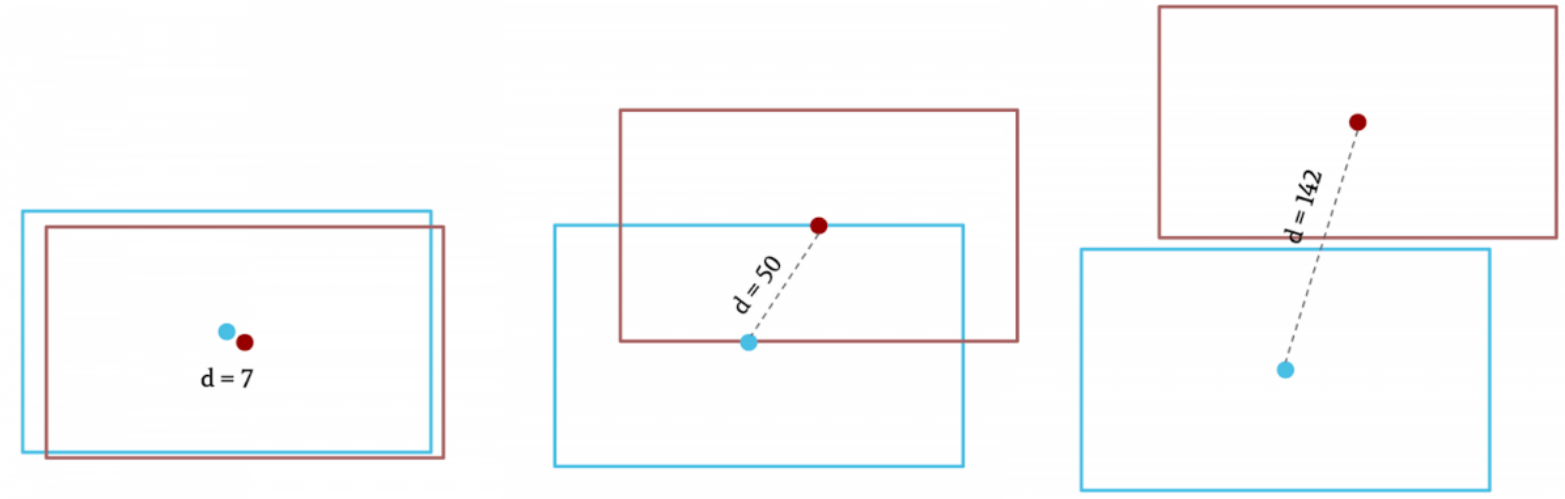
time complexity

# Costs with bounding box matching

# Euclidean Distance

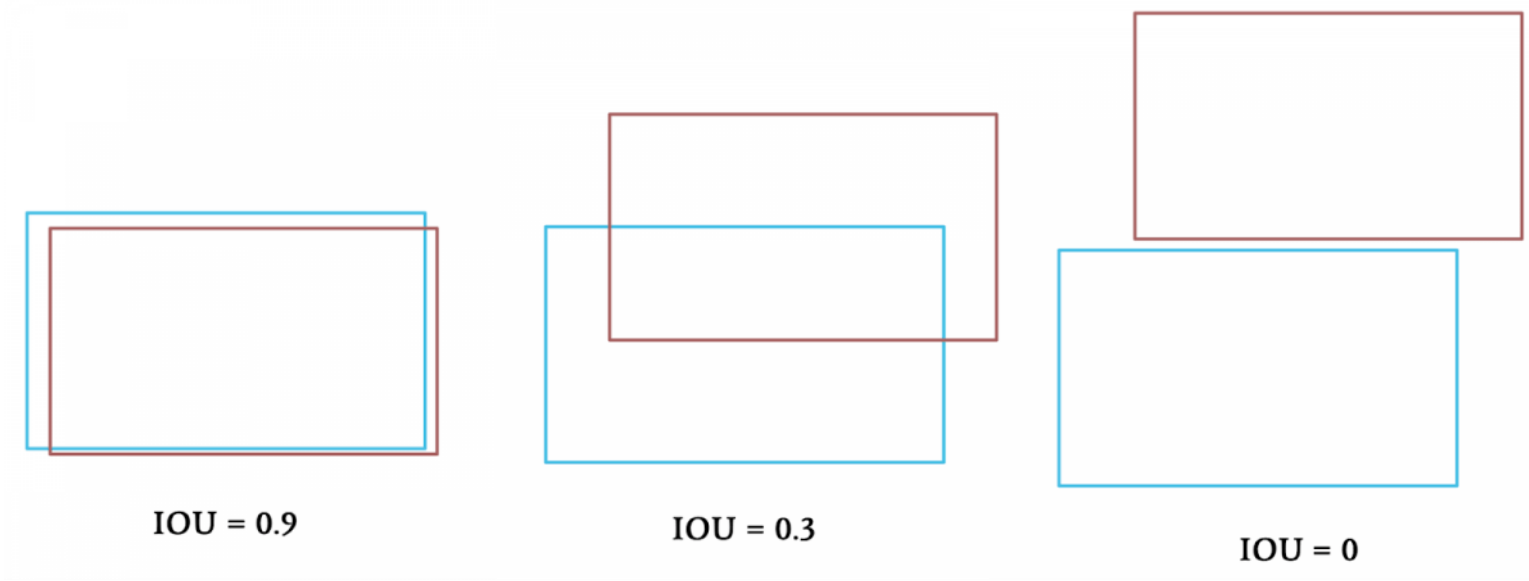
First, we could assign bounding boxes to their closest centers by calculating the Euclidean distance.

Something like: $\mathbf{d = [(x2 - x1)^2 + (y2 - y1)^2 ]^{1/2}}$



# Intersection Over Union (IOU)

❖ Apply the Intersection Over Union. It's a metric used to calculate how much overlap there are between two bounding boxes. Therefore, a cyclist wouldn't match a car!




Although IOU is great, notice how it changes our problem from minimization (find the smallest euclidean distance) to maximization (find the biggest overlap)

# From NxN to NxM

- what happens when we have 3 obstacles in image T, and 4 in image T+1? The trick here is conversion: take the maximum of the entire graph, and add a new column full of this value. If your maximum is 90, add a column full of 90.
- **Put differently, we want to add new edges to our graph**. Let's say we're tracking obstacles and using IOU as a metric. On the detection side (time t), we have 4 obstacles. But on the tracking side (time t-1), we only had 3: a new detection has arrived.

In case of NxM, make it NxN by adding columns with the maximum value in each cell

| | Tracking 1 | Tracking 2 | Tracking 3 |
|---|---|---|---|
| Detection 1 | 21 | 10 | 0 |
| Detection 2 | 56 | 62 | 11 |
| Detection 3 | 20 | 77 | 90 |
| Detection 4 | 10 | 80 | 20 |

| | Tracking 1 | Tracking 2 | Tracking 3 | Tracking 4 |
|---|---|---|---|---|
| Detection 1 | 21 | 10 | 0 | **90** |
| Detection 2 | 56 | 62 | 11 | **90** |
| Detection 3 | 20 | 77 | 90 | **90** |
| Detection 4 | 10 | 80 | 20 | **90** |

If we have an NxM matrix, we can turn it into an NxN matrix by adding a dummy column!

# From Maximization to Minimization

- The second problem is maximization. If you're having **IOU values**, you want the **highest values** to match. If you have **similarity costs**, you want the **most similar examples** to match. Here, the trick is to take the maximum value, and then remove the value of each cell to that maximum.
- In the following example, the value of every cell on the left has been subtracted to 90, the maximum.

Subtract the value of each cell to the maximum

| | Tracking 1 | Tracking 2 | Tracking 3 | Tracking 4 |
|---|---|---|---|---|
| Detection 1 | 21 | 10 | 0 | **90** |
| Detection 2 | 56 | 62 | 11 | **90** |
| Detection 3 | 20 | 77 | 90 | **90** |
| Detection 4 | **10** | 80 | 20 | **90** |

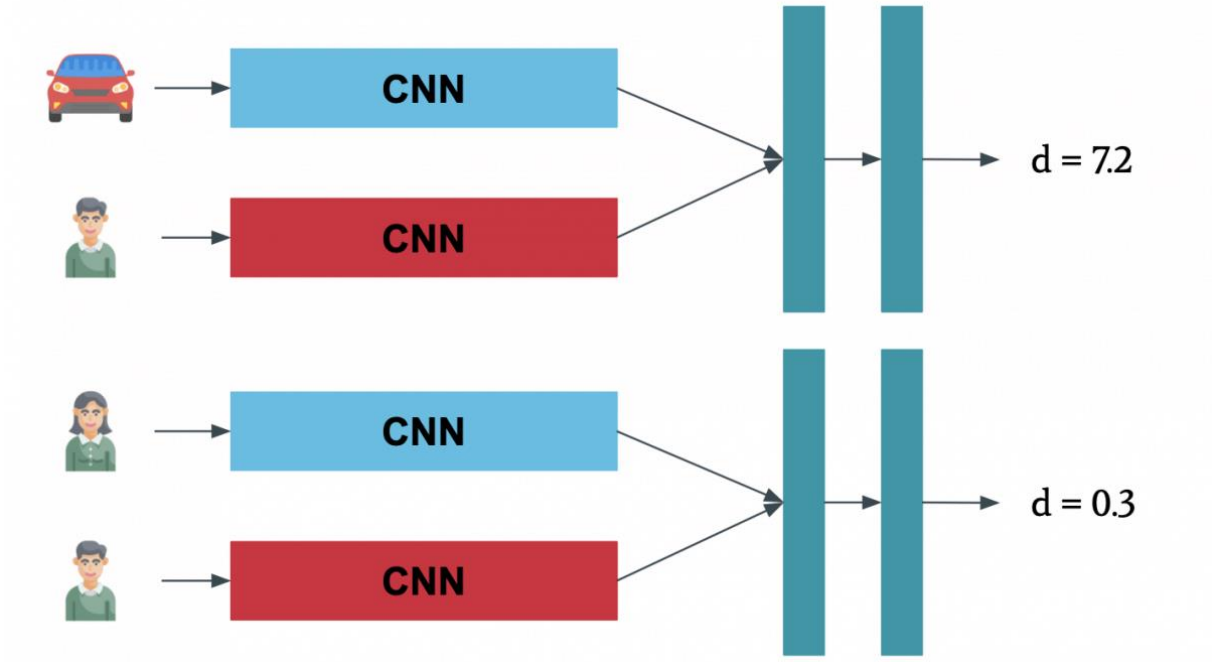
| | Tracking 1 | Tracking 2 | Tracking 3 | Tracking 4 |
|---|---|---|---|---|
| Detection 1 | 69 | 80 | 90 | 0 |
| Detection 2 | 34 | 28 | 79 | 0 |
| Detection 3 | 70 | 13 | 0 | 0 |
| Detection 4 | **80** | 10 | 70 | 0 |

80 = 90 - 10

How to turn a maximization problem into a minimization? Simply remove each value from the maximum!

# Siamese Networks

- Going deeper than calculating the overlap, we could also "look" inside the boxes using CNNs. This is what the DeepSORT algorithm uses, and although computationally expensive, this can help with overlaps.
- To implement this, we use a Siamese Network, calculating the distances between two patches of images. Here, we can use **minimization** (how dissimilar are they?) or **maximization** (how similar are they?).

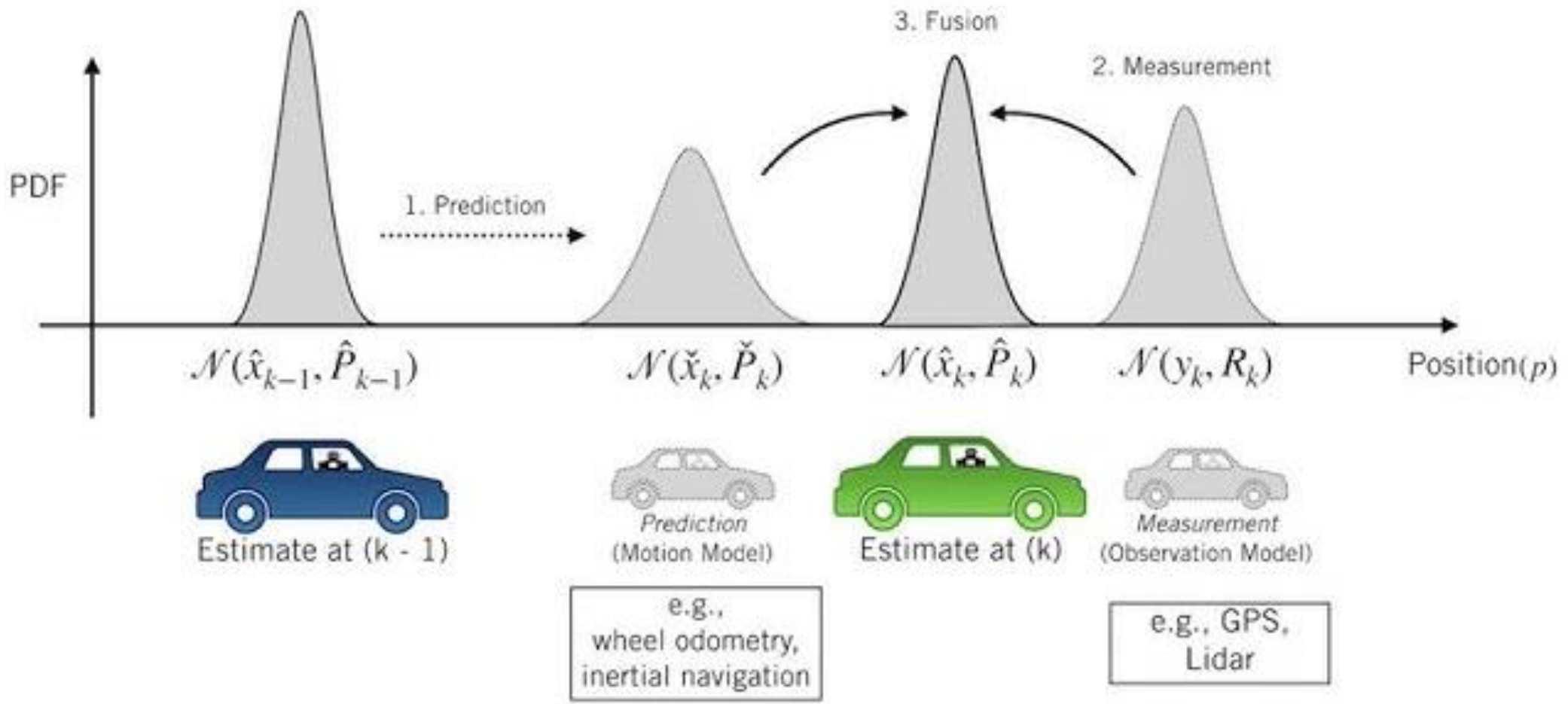



The Siamese Network calculates the cosine distances between convolutional features, so if 2 boxes overlap, we can still tell the difference!

# Kalman Filter

- A Kalman filter is an algorithm that uses a series of measurements to estimate the state of a system over time. It's a common data and sensor fusion algorithm used in many applications, including computer vision, navigation systems, and signal processing.
- The Kalman filter was developed by Hungarian engineer Rudolf Kalman and is named after him. It works by:
  - Predicting: Predicting the state of the system
  - Updating: Using noisy measurements to refine the estimate of the system state
- The Kalman filter is an efficient algorithm that can:
  - Determine the state of a discrete-data controlled process
  - Estimate the uncertainty of the estimates
  - Take into account the different states of time, direction, and acceleration

# The Kalman Filter : Prediction and Correction



# References


- CV3DST - computer Vision 3: Detection, Segmentation, and Tracking - Technical University of Munich - Prof. Leal-Taixé (WS21/22)

- Object Detection Part 6: The Hungarian Matching Algorithm, Tracking, Bounding Box Matching

- https://www.thinkautonomous.ai/blog/hungarian-algorithm/

- Multiple Object Tracking Systems - a Presentation from Tryolabs

https://www.youtube.com/watch?v=dRFoF5pbP7Q&list=PLd3hlSJsX_Im0zAkTX3ogoiDN9Y7G6tSx&index=55