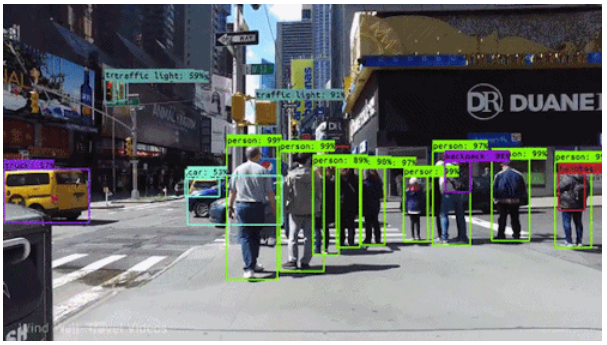
EE 046746 - Technion - Computer Vision

Tutorial 13 - Deep Object Tracking



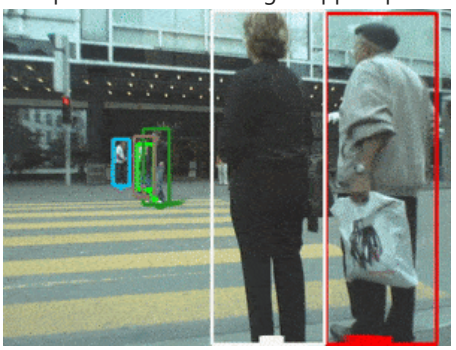
- Image Source

Agenda

In [1]:
```
# imports for the tutorial
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
```

What is Object Tracking?

- Object tracking in videos, or video object tracking, is the process of **detecting an object as it moves through space** in a video.
- Object tracking has a wide range of applications in Computer Vision, such as surveillance, human-computer interaction, and medical imaging, traffic flow monitoring, human activity recognition, etc.
  - For example, if the FBI wants to track a criminal running away in a car using citywide surveillance cameras.
  - Or analyze a soccer game and the performance of the players.

- Another example would be tracking shoppers path in a mall and anlayzing the number of people that entered/exited the mall. 

Object Tracking Vs. Object Detection

- Why can't we use object detection in each frame in the whole video and track the object?
  - If the image has multiple objects, then we have no way of connecting the objects in the current frame to the previous frames.
  - If the object you were tracking goes out of the camera view for a few frames and another one appears, we have no way of knowing if it's the same object.
  - Essentially, during detection, we work with **one image at a time** and we have no idea about the *motion* and past movement of the object, so **we can't uniquely track objects in a video**.
- Whenever there is a moving object in the videos, there are certain cases when the visual appearance of the object is not clear.
- In all such cases, *detection* would fail while *tracking* succeeds as it also has the motion model and history of the object.

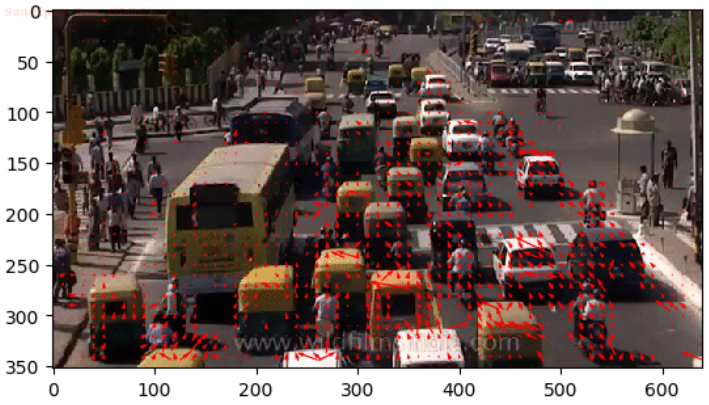
Detection Failure Cases

- **Occlusion**- the object in question is partially or completely occluded.
- **Identity Switches** - after two objects cross each other, can't tell the correct identity.
- **Motion Blur** - object is blurred due to the motion of the object or camera. Hence, visually, the object doesn't look the same anymore.
- **Viewpoint Variation** - different viewpoint of an object may look very different visually and without the context. It becomes very difficult to identify the object using only visual detection.
- **Scale Change** - huge changes in object scale may cause a failure in detection.

- **Background Clutters** - background near object has similar color or texture as the target object. Hence, it may become harder to separate the object from the background.
- **Illumination Variation** - illumination near the target object is significantly changed. Hence, it may become harder to visually identify it.
- **Low Resolution** - when the number of pixels inside the ground truth bounding box is low, it may be too hard to detect the objects visually.

Under the Hood of Object Tracking

- There are various techniques and algorithms which try to solve the tracking problem in many different ways.
- A good object tracker has two basic models: **Motion Model** and **Visual Appearance Model**.



- Image Source - Introduction to Motion Estimation with Optical Flow

Motion Model

- The ability to understand and model the motion of the object.
  - A good motion model captures the dynamic behavior of an object.
  - It predicts the potential position of objects in the future frames, hence, reducing the search space.
  - However, the motion model alone can fail in scenarios where motion is caused by things that are not in a video or abrupt direction and speed change.
  - Some of the classic methods understand the motion pattern of the object and try to predict that.
    - However, the problem with such approaches is that they can't predict the abrupt motion and direction changes.
    - Examples of such techniques are Optical Flow, Kalman Filtering, Kanade-Lucas-Tomashi (KLT) feature tracker, mean shift tracking.

Visual Appearance Model

- The ability to understand the appearance of the object that is tracked.
  - Trackers need to learn to discriminate the object from the background.
  - In single object trackers (one object), visual appearance alone could be enough to track the object across frames, while In multiple-object trackers, visual appearance alone is not enough.

Object Tracking Procedure

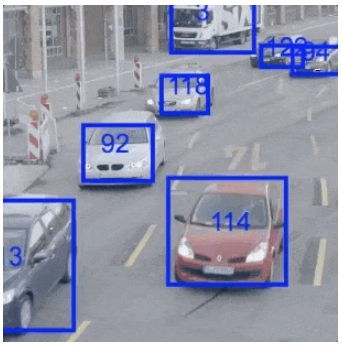
In general, the object tracking procedure is composed of **4 main modules**:

- **Target initialization/object detection**: an initial set of object detections is created. This is typically done by taking a set of bounding box coordinates and using them as inputs for the network.
  - The idea is to draw bounding box of the target in the initial frame of the video and tracker has to estimate the target position in the remaining frames in the video.
- **Appearance modeling**: learning the visual appearance of the object by using (deep) learning techniques. In this phase, the model learns the visual features of the object while in motion, various view-points, scale, illuminations etc.

- **Motion estimation**: the objective of motion estimation is learning to predict a zone where the target is most likely to be present in the subsequent frames.

- **Target positioning**: motion estimation predicts the possible region where the target could be present, thus, yielding an area to search to lock down the exact location of the target.

It is usually the case that tracking algorithms don't try to learn all the variations of the object. Hence, most of the tracking algorithms are much faster than regular object detection.



- Image Source - Tracking Things In Object Detection Videos

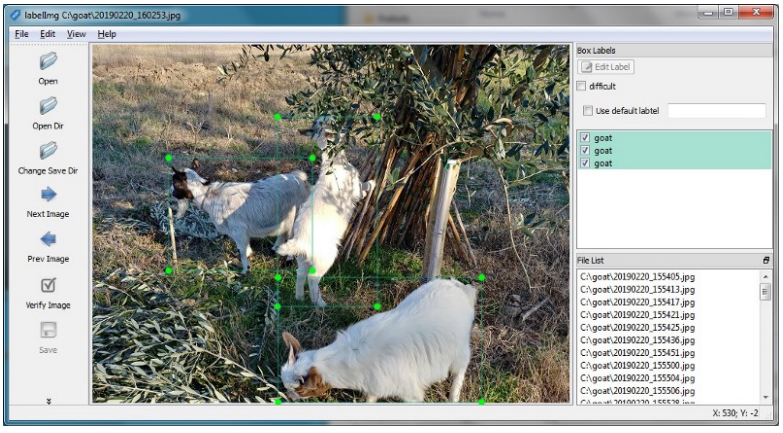
Types of Tracking Algorithms

We can classifiy object trackers according to whether they are based on automatic object detection or manual, whether they track a single object or capable of tracking multi objects and whether they operate *online* or *offline*.

Detection-Based Vs. Detection-Free

- **Detection-based**: the consecutive video frames are given to a pretrained object detector that forms a detection hypothesis which in turn is used to form tracking trajectories.
  - It is more popular because new objects are detected and disappearing objects are terminated automatically.
  - In these approaches, the tracker is used for the failure cases of object detection.
  - In an another approach, object detector is run every n frames and the remaining predictions are done using the tracker.
  - Suitable approach for tracking for a long time.

- **Detection-free**: requires manual initialization of a fixed number of objects in the first frame. It then localizes these objects in the subsequent frames.
  - Cannot deal with the case where new objects appear in the middle frames.



- Image Source

Single Object Vs. Multi Object

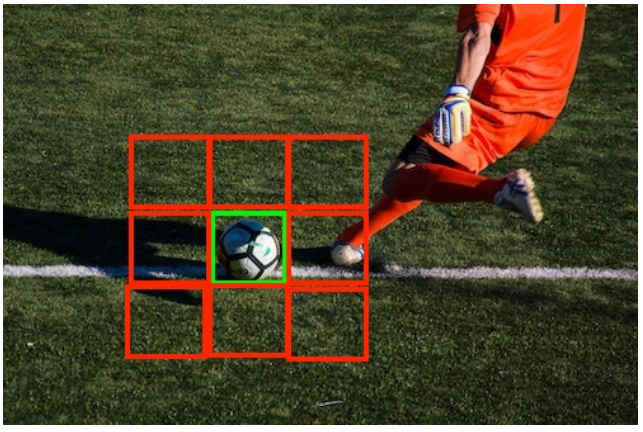
- **Single Object Tracking**: only a single object is tracked even if the environment has multiple objects in it. The object to be tracked is determined by the initialization in the first frame.
- **Multi Object Tracking**: all the objects present in the environment are tracked over time. If a detection based tracker is used it can even track new objects that emerge in the middle of the video.

Offline Vs. Online

- **Offline Trackers** - used when you have to track an object in a recorded stream.
  - For example if you have recorded videos of a soccer game of an opponent team which needs to be analyzed for strategic analysis. In such case, you can not only use the past frames but also future frames to make more accurate tracking predictions.
- **Offline Learning Trackers** - the training of these trackers only happen offline.
  - As opposed to online learning trackers, these trackers don't learn anything during run time.
  - We can train a tracker to identify persons and then these trackers can be used to continuously track all the persons in a video stream. Pre-trained.

- **Online Trackers** - online trackers are used where predictions are available immediately and hence, they can't use future frames to improve the results.
- **Online Learning Trackers** - typically learn about the object to track using the initialization frame and few subsequent frames, making these trackers more general because you can just draw a bounding box around any object and track it.
  - For example, if you want to track a person with red shirt in the airport, you can just draw a bounding box around that person in 1 or few frames. The tracker would learn about the object using these frames and would continue to track that person.

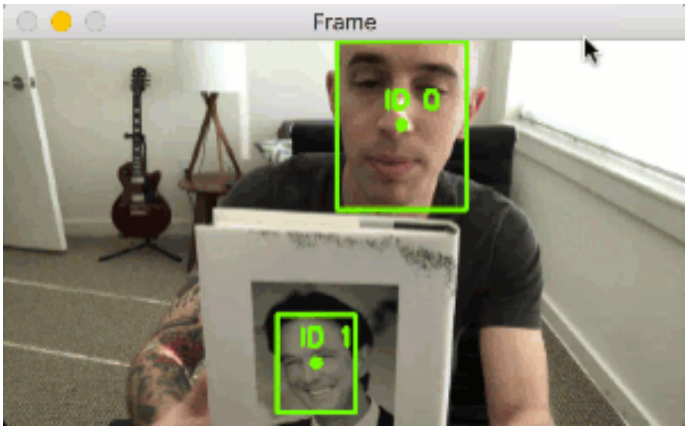
- In online learning trackers, Center Red box is specified by the user, it is taken as the positive example and all the boxes surrounding the object are taken as negative class and a classifier is trained which learns to distinguish the object from the background.



From Classic to Deep Learning Algorithms

- Most of the classic trackers are not very accurate due to the limitations mentioned in the beginning.

    - However, some times they can be useful to run in a resource constraint environment like an embedded system.
- **Kernelized Correlation Filters (KCF)** tracker is a very fast, well-performing tracker. Read more: High-Speed Tracking with Kernelized Correlation Filters
- A lot of classic tracking algorithms are integrated in OpenCV's tracking API.
    - A very simple tracking technique is based on finding centroids and can be easily implemented with OpenCV.
        - Simple object tracking with OpenCV (Code is available).
- Deep learning based trackers are now miles ahead of traditional trackers in terms of accuracy.

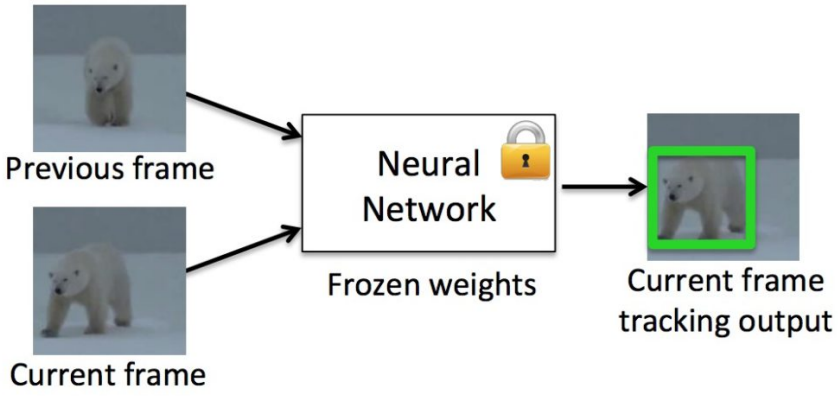


Deep Object Tracking Algorithms

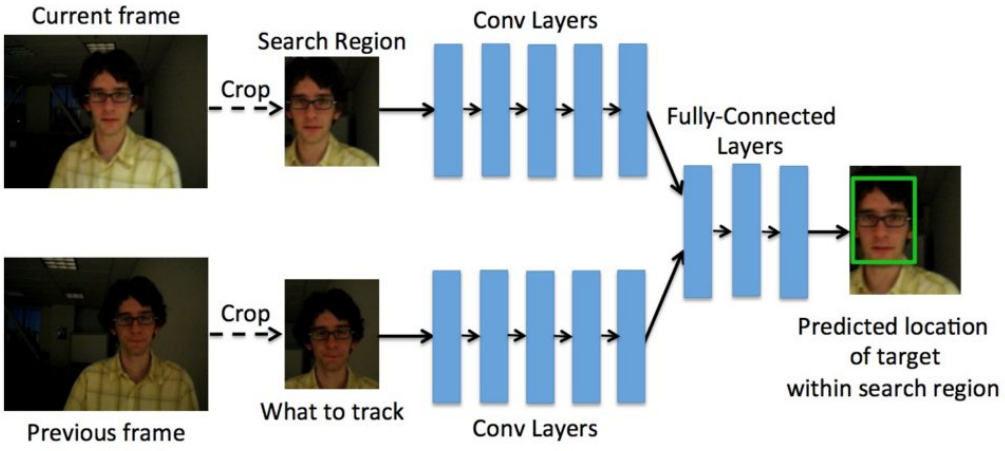
- We will present 3 popular deep learning based trackers, but there are more.
- Only one of them is implemented in OpenCV, but we will provide PyTorch code for the rest.

Generic Object Tracking Using Regression Networks (GOTURN)

- GOTURN was introduced by David Held, Sebastian Thrun, Silvio Savarese in their paper titled "Learning to Track at 100 FPS with Deep Regression Networks".
- GOTURN uses deep neural networks to track objects in an **offline** fashion.
    - Most tracking algorithms train *online*, which is to say the algorithm learns how the object appears *only at runtime*.
    - In contrast, GOTURN is trained on thousands of chunks of video before runtime, and as a result, it doesn't need to train at all during runtime.
- GOTURN is trained using a pair of *cropped* frames from thousands of videos and outputs the bounding box around the object in the second frame.



- In the first frame (also referred to as the previous frame), the location of the object is known, and the frame is cropped to two times the size of the bounding box around the object.
- The object in the first cropped frame is *always centered*.
- The location of the object in the second frame (also referred to as the current frame) needs to be predicted.
- The bounding box used to crop the first frame is also used to crop the second frame. Because the object might have moved, the object is not centered in the second frame.
- A **Convolutional Neural Network (CNN)** is trained to predict the location of the bounding box in the second frame.



- GOTURN is the only deep-learning algorithm implemented in OpenCV.
- GOTURN can run very fast i.e. 100fps on a GPU powered machine.
- We will now see how it works!
- If you prefer to work with PyTorch, here are 2 repositories:
  - Source 1
  - Source 2

---------- </> CODE TIME ----------

The first step is downloading the pre-trained model files. There are 2 ways to download it:

1. Downloading the files separately and merging them locally, instructions can be found here: LINK
2. Download the merged files from Dropbox: LINK

You should end up with 2 files: `goturn.caffemodel` and `goturn.prototxt` and they should be placed in the current working directory (`./`).

In [64]:

```
# initialize tracker
tracker = cv2.TrackerGOTURN_create()

# Read video
video = cv2.VideoCapture("./datasets/tracking/slow_traffic_small.mp4")

# Exit if video not opened
if not video.isOpened():
    print("Could not open video")
    raise SystemError

# Read first frame
ok, frame = video.read()
if not ok:
```

```
    print("Cannot read video file")
    raise SystemError
```

In [65]:
```
# Define a bounding box
# bbox = (276, 23, 86, 320)
# bounding box parameters: (y1, y2, x1, x2)

# Uncomment the line below to select a different bounding box
bbox = cv2.selectROI(frame, False)
# False is to say that we want to draw rectangle from top left (and not from the center)
```

In [66]:
```
# Initialize tracker with first frame and bounding box
ok = tracker.init(frame, bbox)
# `ok` is a boolean which is False until the frame is tracked.

# close all windows
cv2.destroyAllWindows()
```

Predict the bounding box in a new frame

We loop over all frames in the video and find the bounding box for new frames using `tracker.update`. The rest of the code is simply for timing and displaying.

In [67]:
```
while True:
    # this loop will stop once we reached the final frame
    # Read a new frame
    ok, frame = video.read()
    if not ok:
        break

    # Start timer
    timer = cv2.getTickCount()

    # Update tracker
    ok, bbox = tracker.update(frame)

    # bbox is the returned bounding box, you can do all sorts of stuff with it

    # Calculate Frames per second (FPS)
    fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer);

    # Draw bounding box
    if ok:
        # Tracking success
        p1 = (int(bbox[0]), int(bbox[1]))
        p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
        cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
        # mark the object
        # cv2.putText(frame, "Object", (int(bbox[0]), int(bbox[1])), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0,
    else:
        # Tracking failure
        cv2.putText(frame, "Tracking failure detected", (100, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 2

    # Display tracker type on frame
    cv2.putText(frame, "GOTURN Tracker", (100, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50),2);

    # Display FPS on frame
    cv2.putText(frame, "FPS : " + str(int(fps)), (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50),

    # Display result
    cv2.imshow("Tracking", frame)

    # Exit if ESC pressed
    k = cv2.waitKey(1) & 0xff
    if k == 27:
        break
```

In [68]:
```
# close all windows
cv2.destroyAllWindows()
```

Multi-Domain Convolutional Neural Network Tracker (MDNet)

- MDNet, an **online** video object tracker, was introduced by Hyeonseob Nam and Bohyung Han in their paper titled "Learning Multi-Domain Convolutional Neural Networks for Visual Tracking".
  - Winner of VOT2015 challenge.
- Because training a deep neural network is computationally expensive, small networks are used for training around deployment time.
- The drawback to small networks is that they lack the classification/discrimination power of larger networks.
- In order to deal with the fact that networks which train at runtime have lower discriminatory power, the training of the network can be split into different steps.

- For instance, the entire network can be trained *before* runtime, but during runtime, the first few layers of the network are used as feature extractors and only the last few layers of the network have their weights adjusted.
- Essentially, the **CNNs are trained beforehand** and used to extract features, while the **last layers can quickly be trained online**.
- Theoretically, this creates a multi-domain CNN that can be used in many different scenarios, capable of discriminating between background and target.

- In practice, the *background* of one video could be the *target* of a different video, and so the CNN must have some method of discriminating between these two situations.
- MDNet handles possible confusion from similar targets and backgrounds by dividing the network into two portions, a shared portion and a portion that remains independent for every domain.
- **Every domain has its own training video**, and the network is trained for the total number of different domains.
  - The network is first trained over K-domains iteratively where each domain classifies between its target and background.

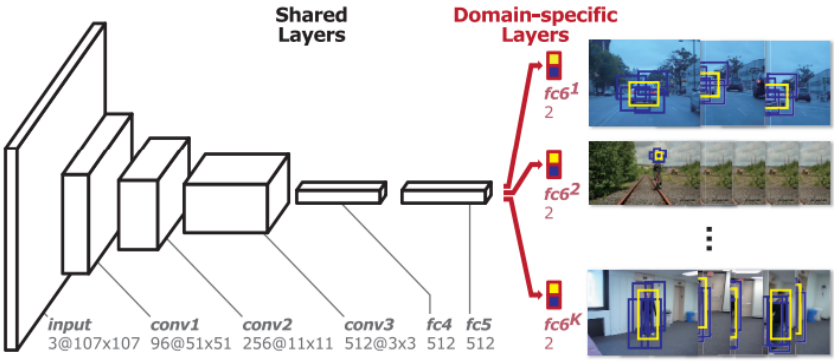


Figure 1: The architecture of our Multi-Domain Network, which consists of shared layers and K branches of domain-specific layers. Yellow and blue bounding boxes denote the positive and negative samples in each domain, respectively.

- In the k^{th} iteration, the network is updated based on a minibatch that consists of the training samples from the $(k mod K)^{th}$ sequence, where only a single branch `fc6(k mod K)` is enabled.

- After training is complete, the layers specific to the different domains are removed and as a result, a feature extractor capable of interpreting any given background/object pairs is created.
- During the process of inference, a binary classification layer (a single fully-connected layer, `fc6`) is created by removing the domain-specific layers and adding a binary classifier.

- To estimate the target state in each frame, N target candidates $x_1, \ldots, x_N$ sampled around the previous target state are evaluated using the network, and we obtain their positive scores $f^+(x^i)$ and negative scores $f^-(x^i)$ from the network. The optimal target state x^* is given by finding the example with the maximum positive score as

$$x^* = arg\max_{x^i} f^+(x^i)$$

- MDNet is one of the most accurate deep learning based online training, detection free, single object tracker.
- PyTorch Code

---------- </> CODE TIME ----------

- Run from Terminal

In [ ]:
```
import sys
sys.path.append("./models/pyMDNet/tracking")
from models.pyMDNet.tracking.run_tracker import track_mdnet
```

In [ ]:
```
vid_path = './datasets/tracking/ironman.mp4'
img_path = './datasets/tracking/ironman_frames'
seq_name = "iron_man_custom"
track_mdnet(vid_path, img_path, display=False, seq_name=seq_name)
```

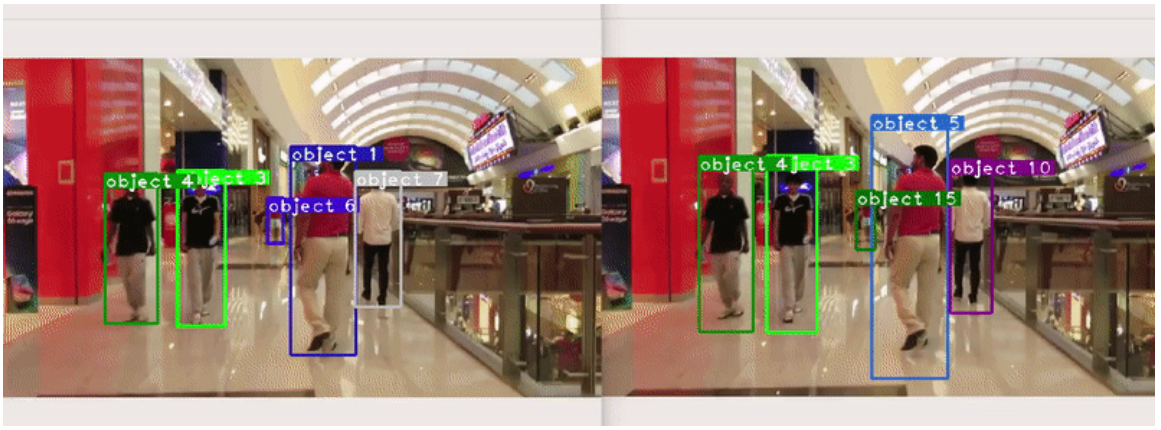
- Converting in in between frames and mp4

In [74]:
```
image_folder = './models/pyMDNet/results/ironman_frames/figs'
video_name = 'ironman_mdnet.mp4'

images = [img for img in os.listdir(image_folder) if img.endswith(".jpg")]
frame = cv2.imread(os.path.join(image_folder, images[0]))
height, width, layers = frame.shape

video = cv2.VideoWriter(video_name, 0, 30, (width,height))

for image in images:
    video.write(cv2.imread(os.path.join(image_folder, image)))

cv2.destroyAllWindows()
video.release()
```

Deep Simple Online and Realtime Tracking (Deep SORT)

- Deep SORT, an **online** multiple object tracker which is an extenstion of the SORT algorithm, was introduced by Nicolai Wojke, Alex Bewley and Dietrich Paulus in their paper titled "Simple Online and Realtime Tracking with a Deep Association Metric".
- Deep SORT integrates appearance information to improve the performance of SORT.
  - Due to this extension it is possible to track objects through longer periods of occlusions, effectively reducing the number of *identity switches*.
- Much of the computational complexity is placed into an offline pre-training stage where a deep association metric is learned on a large-scale person re-identification dataset.
- During online application, measurement-to-track associations is established using nearest neighbor queries in visual appearance space.

- Experimental evaluation shows that these extensions reduce the number of identity switches by 45%, achieving overall competitive performance at high frame rates.



- Image Source

- The original SORT algorithm (Paper, Code) proposed the following steps to perform tracking:
  - **Detection**: using a CNN-based detection architecture (VGG16) to extract regions.
  - **Estimation Model**: the representation and the motion model used to propagate a target's identity into the next frame.
    - Inter-frame displacements of each object are estimated with a linear constant velocity model which is independent of other objects and camera motion. The state of each target is modeled as:

$$x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]$$

    - u and v represent the horizontal and vertical pixel location of the centre of the target, while the scale s and r represent the scale (area) and the aspect ratio of the target's bounding box respectively.
    - When a detection is associated to a target, the detected bounding box is used to update the target state where the velocity components are solved optimally via a **Kalman filter** framework.

- Steps continued:
  - **Data Association**: In assigning detections to existing targets, each target's bounding box geometry is estimated by predicting its new location in the current frame.
    - The assignment cost matrix is then computed as the intersection-over-union (IOU) distance between each detection and all predicted bounding boxes from the existing targets.
  - **Creation and Deletion of Track Identities**: When objects enter and leave the image, unique identities need to be created or destroyed accordingly.
    - If an object reappears, tracking will implicitly resume under a new identity.

- So where is the *deep learning* in all of that?
- Despite the effectiveness of Kalman filter, it fails in many of the real world scenarios, like occlusions, different view points etc.
- **Deep SORT** replaced the CNNs with YOLO (You Only Look Once) deep object detector and introduced another distance metric based on the *"appearance"* of the object.
- The idea is to obtain a vector that can describe all the features of a given image (here, the crop of the object is used).
  - First, build a classifier over the dataset, train it till it achieves a reasonably good accuracy, and then strip the final classification layer.
  - Assuming a classical architecture, we will be left with a dense layer producing a single feature vector, waiting to be classified.
  - That feature vector becomes the "appearance descriptor" of the object.
  - The loss of appearance vectors is added to the loss of the original SORT.
- PyTorch Code (with YOLO)

Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- Deep Video Object Tracking - Deep Video Object Tracking - Xavier Giro - UPC Barcelona 2019
- GOTURN - GOTURN : Deep Learning based Object Tracker
  - GOTURN - GOTURN - a neural network tracker
- MDNet - Learning Multi-Domain Convolutional Neural Networks for Visual Tracking (MDNet)

Credits

- EE 046746 Spring 21 - Tal Daniel
- Zero to Hero: A Quick Guide to Object Tracking: MDNET, GOTURN, ROLO - Ankit Sachan
- Object Tracking In Videos
- GOTURN : Deep Learning based Object Tracking
- DeepSORT: Deep Learning to Track Custom Objects in a Video
- Icons from Icon8.com - https://icons8.com