#CodeGen2021

@cloudgen_verona

TOPIC

(Machine) Learning to Social Distance

How to get away with a pandemic: a social distance ML guide

Federico Cunico

About me

2014

2019

2021

UNIVERSITÀ di VERONA — Dipartimento di INFORMATICA

HUMATICS + UNIVERSITÀ di VERONA — Dipartimento di INFORMATICA

About me

2014

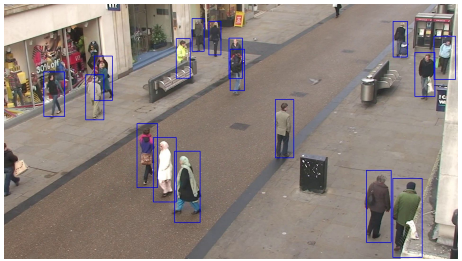
2019

2021

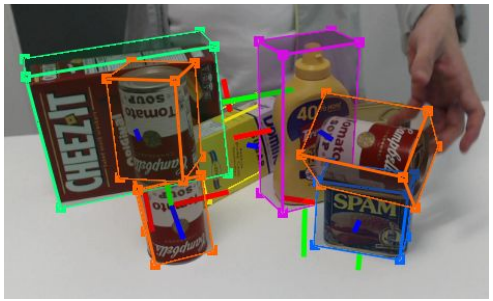
UNIVERSITÀ di VERONA Dipartimento di INFORMATICA

HUMATICS + UNIVERSITÀ di VERONA Dipartimento di INFORMATICA

People Detection and Tracking



Object Pose Estimation



Overview

- The Social Distance problem
- Machine Learning for SD
- The “Old” Computer Vision Techniques
- What about some code?
- Problems

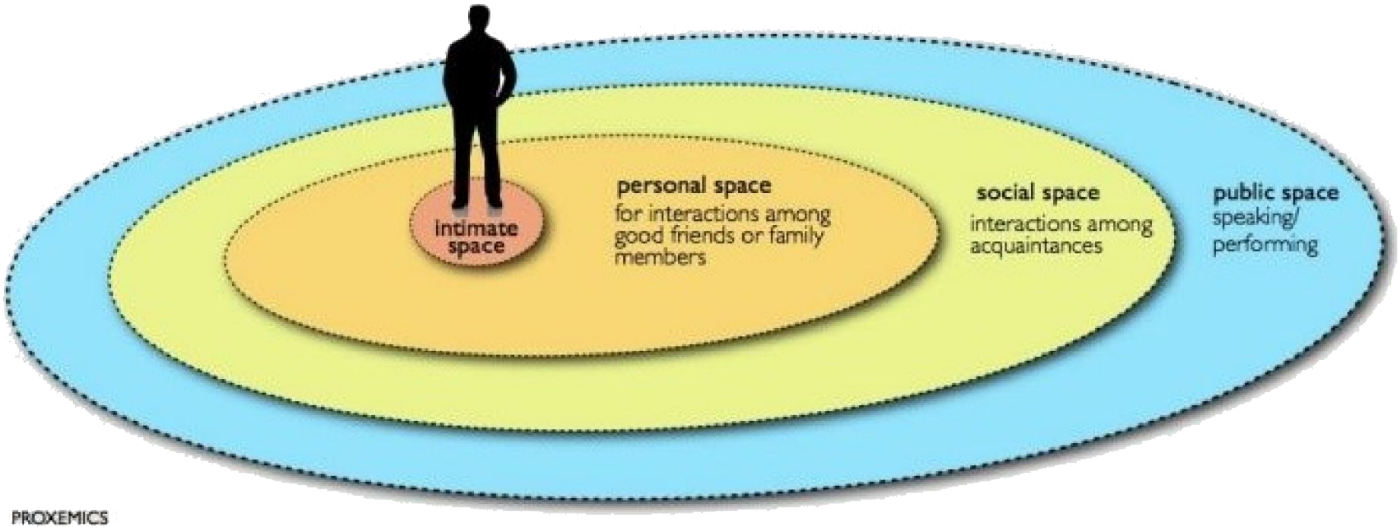
Overview

Our goal is to highlight people too close to each other.

In other words: verify that the “social distance” is preserved.

The Social Distance Problem

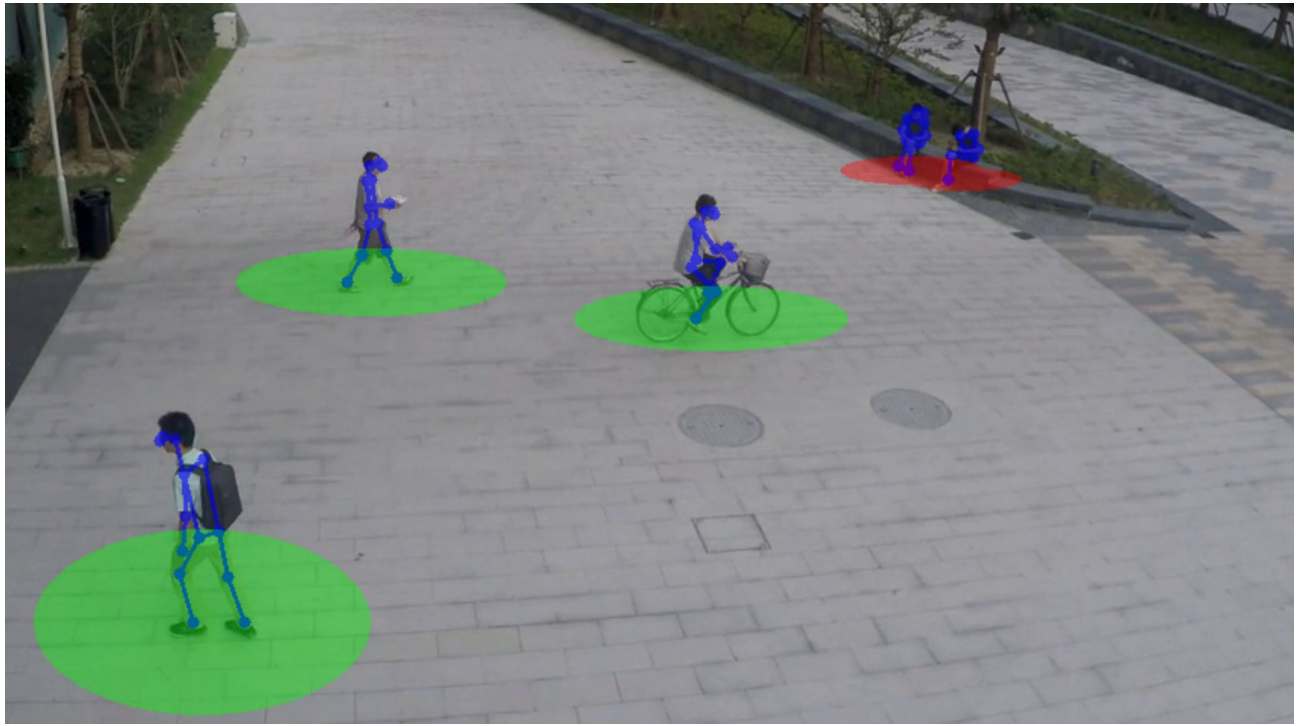
The Social Distance problem



PROXEMICS

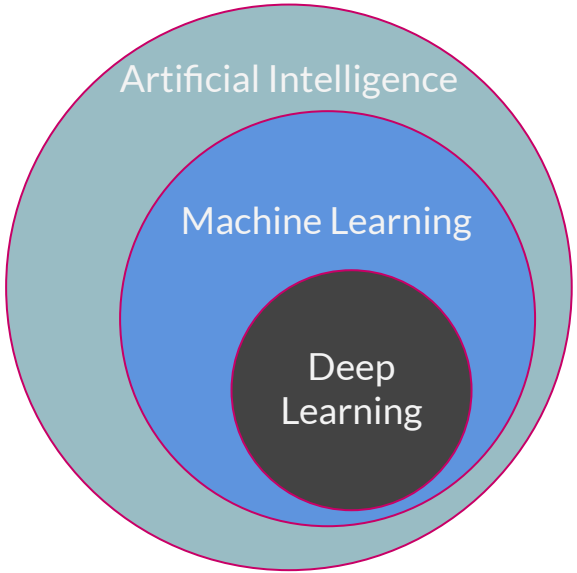
REFERENCE: EDWARD T. HALL THE HIDDEN DIMENSION (1966)

The Social Distance problem

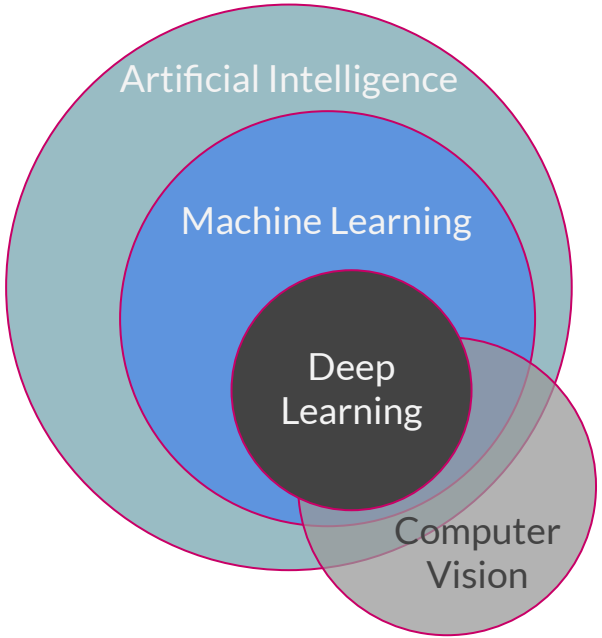


Machine Learning for Social Distance

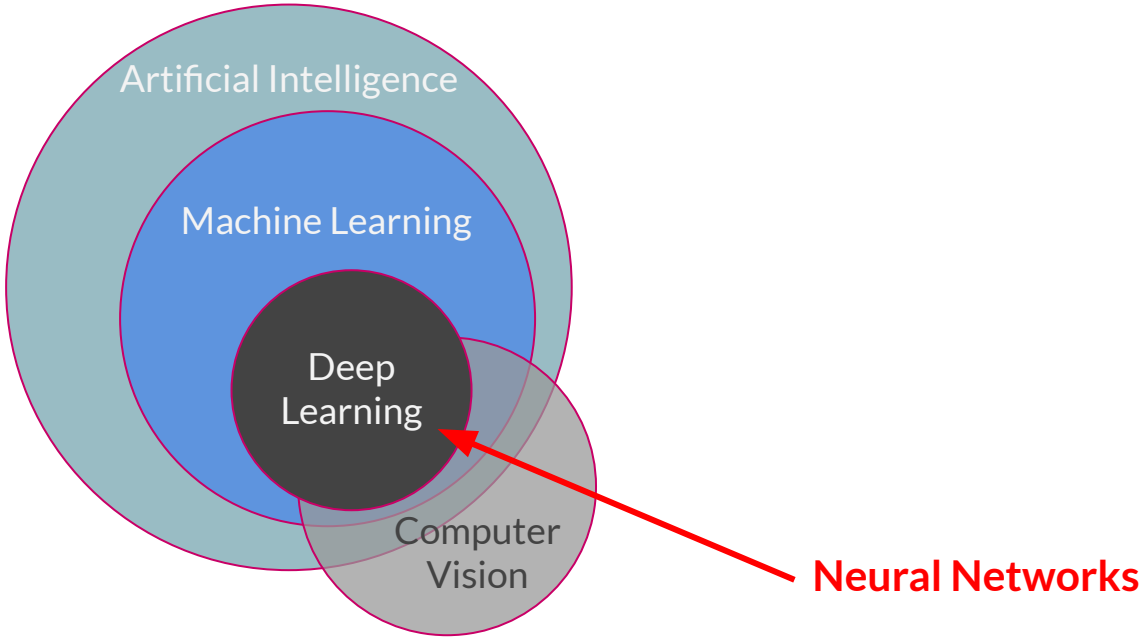
AI? Machine Learning?
Deep Learning? Neural Networks?



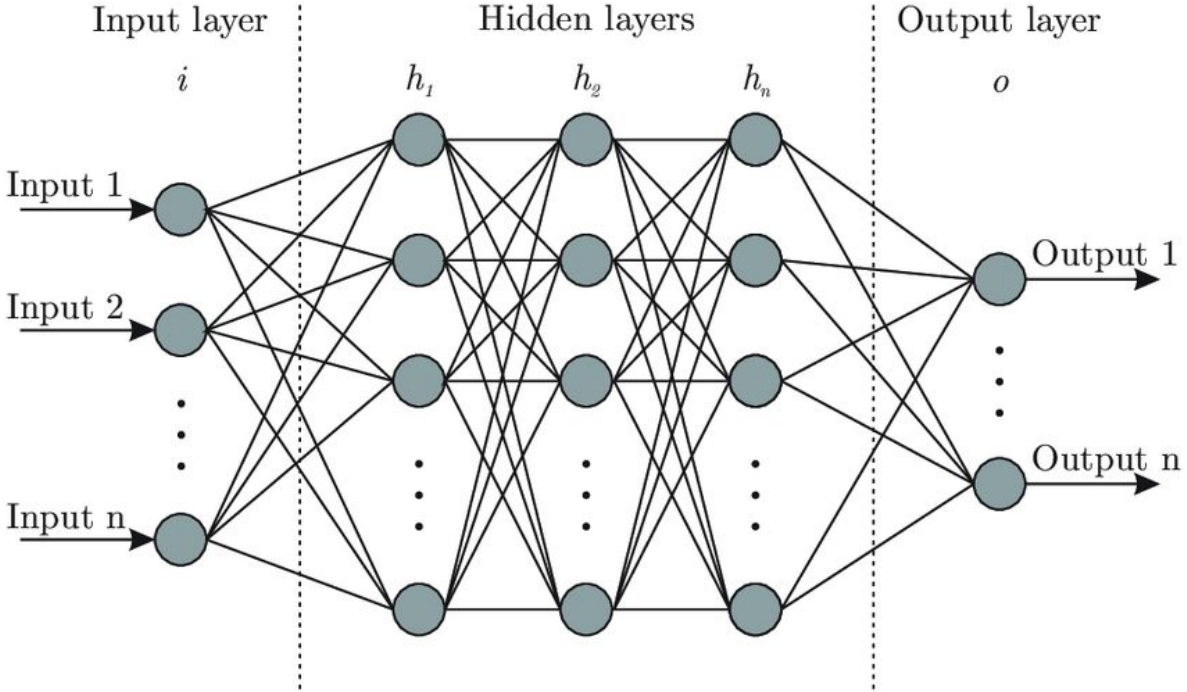
AI? Machine Learning?
Deep Learning? Neural Networks?



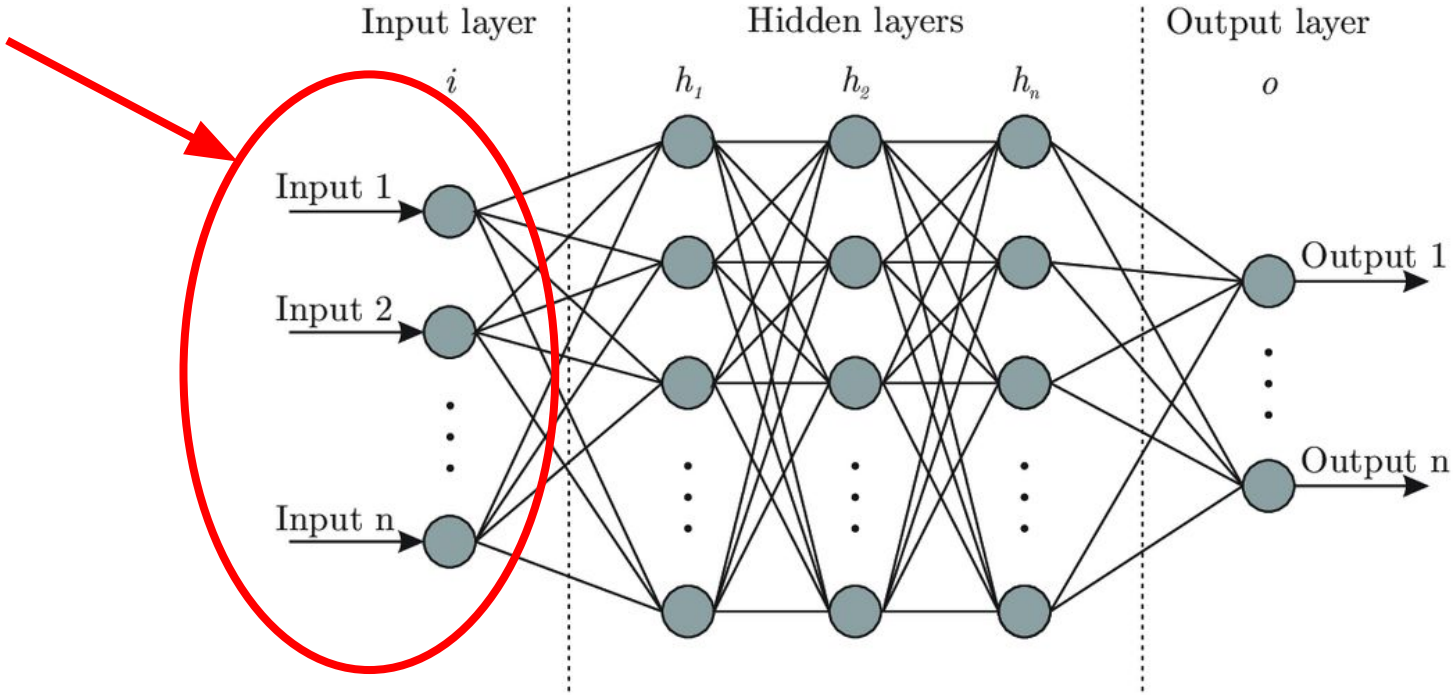
AI? Machine Learning? Deep Learning? Neural Networks?



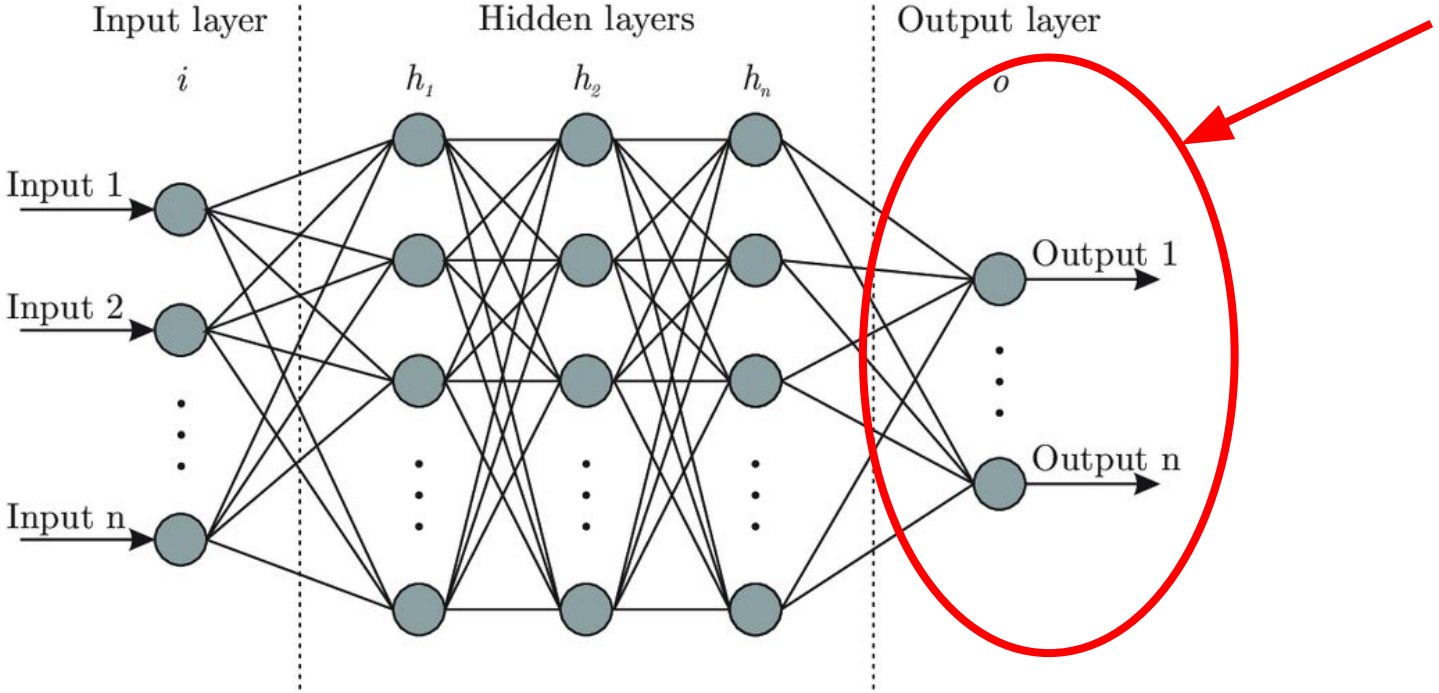
Neural Networks perform very well in various tasks!



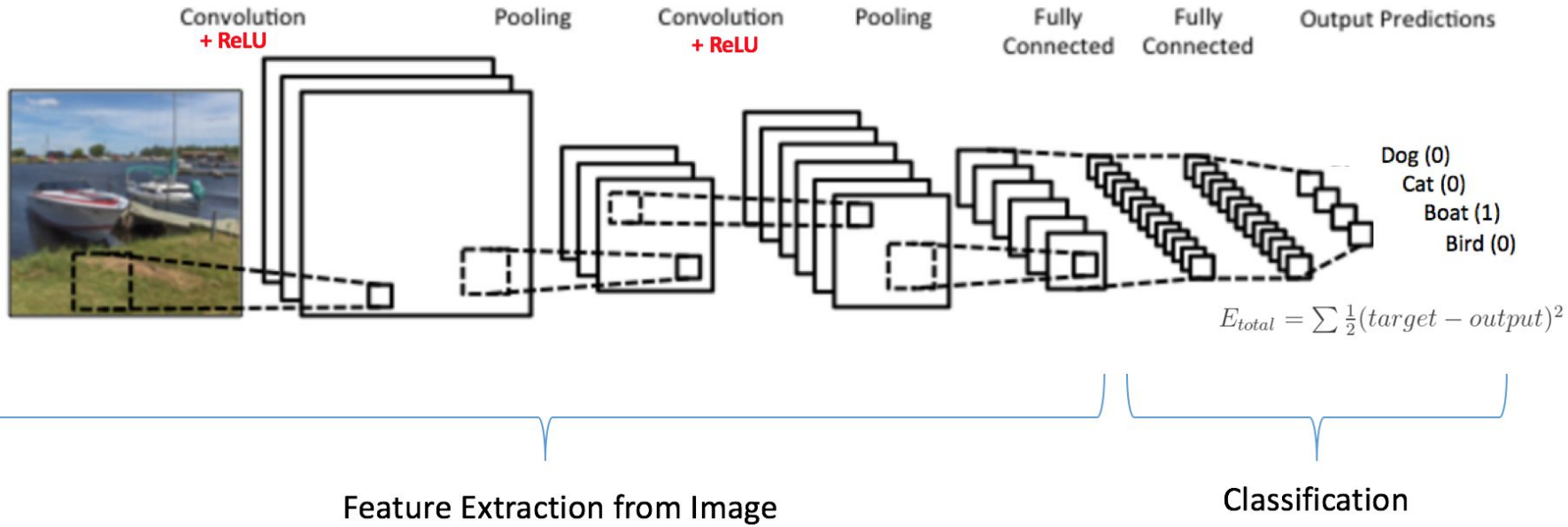
Neural Networks perform very well in various tasks!



Neural Networks perform very well in various tasks!



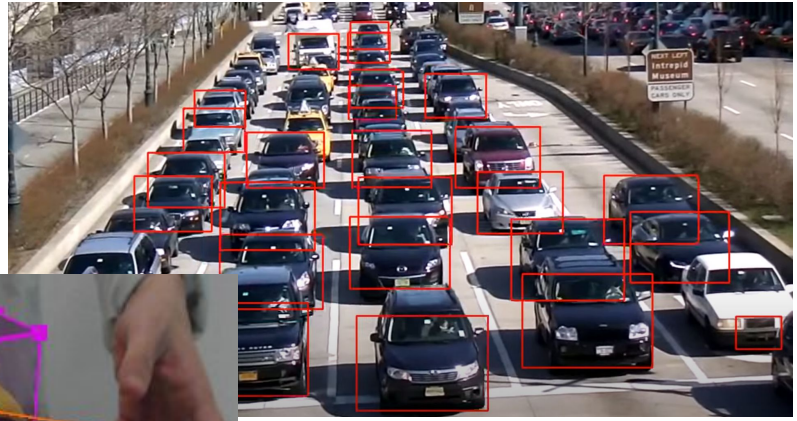
Neural Networks perform very well in various tasks!

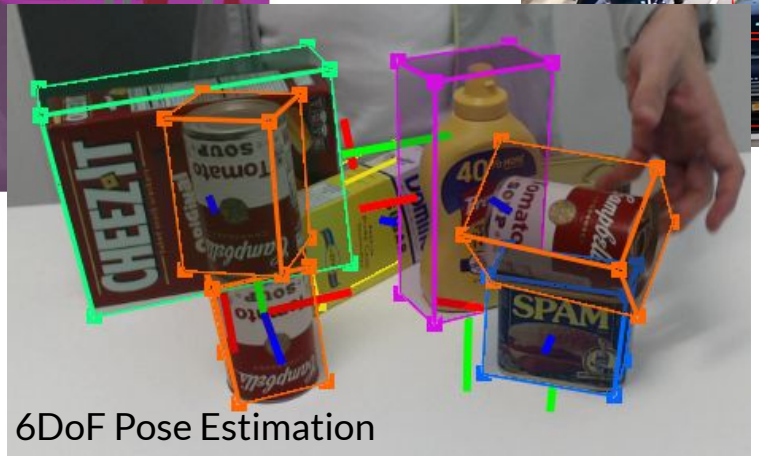


What kind of Neural Network can be useful?

Image Segmentation

Object Detection





6DoF Pose Estimation

We need to get people in the image (2D)

People Detection (2D)

https://github.com/CMU-Perceptual-Computing-Lab/openpose

So we are done, right?

People detection / pose estimation

So we are done, right?

People detection / pose estimation

… well, actually not! People detection may not be enough!

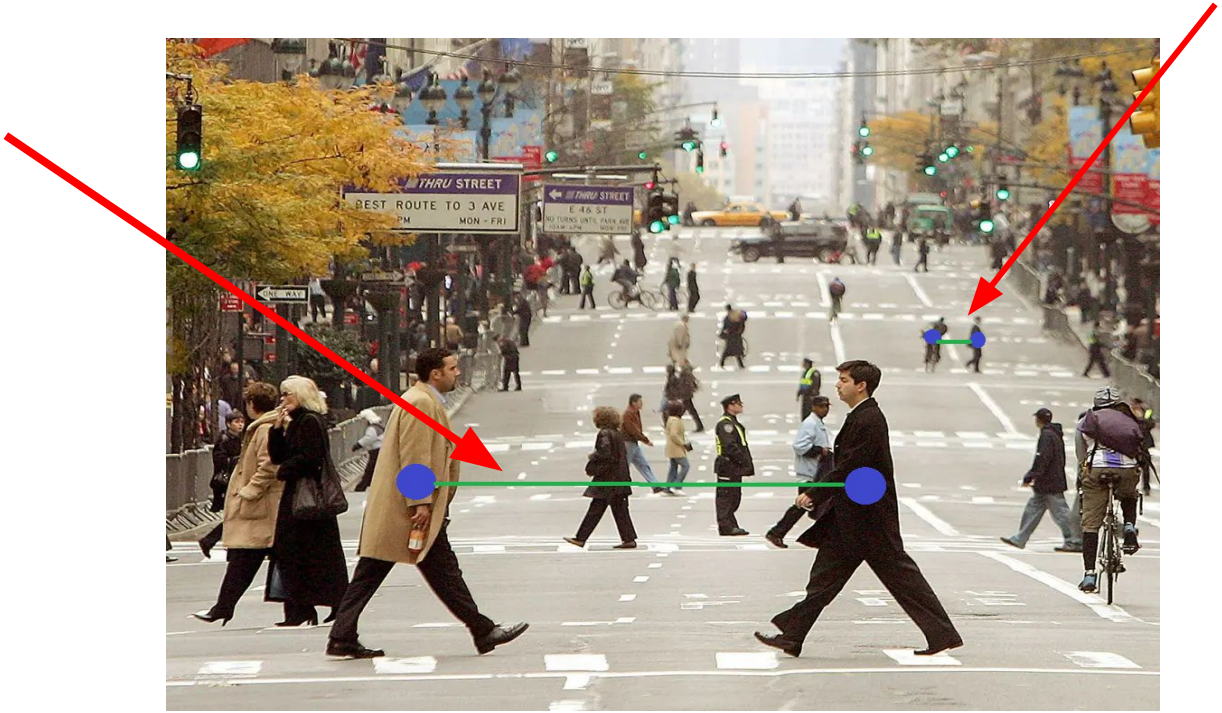
Computer Vision Techniques

What’s the problem?

What's the problem?

PERSPECTIVE

What’s the problem?



We need a knowledge of the geometry of the scene

Scene geometry understanding

?

People detection / pose estimation

- 2D People Detection
- 3D Pose Estimation

We need a knowledge of the geometry of the scene

Scene geometry understanding

- Ground plane estimation
- Homography estimation
- Depth estimation

People detection / pose estimation

- 2D People Detection
- 3D Pose Estimation

We will use the Homography

The planar homography relates the transformation between two planes (up to a scale factor):

$$s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

We will use the Homography

The planar homography relates the transformation between two planes (up to a scale factor):

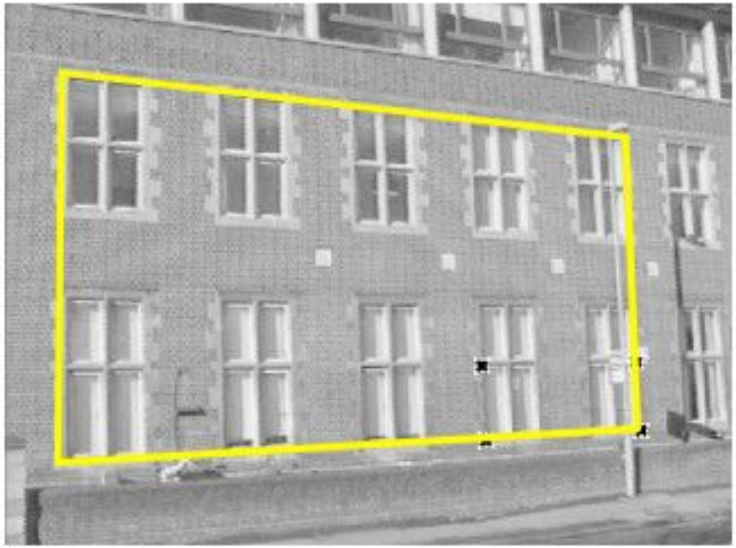
$$s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

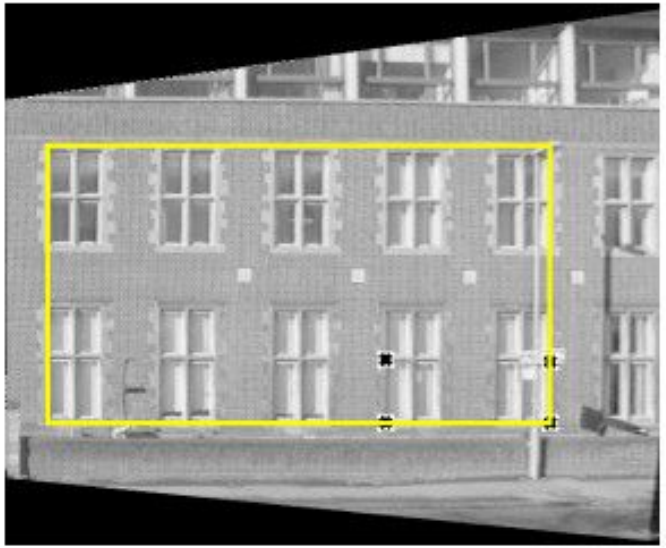
Basically it means that the homography matrix (H) transforms a pixel position into another pixel position.

i.e. it transforms* an image to another image, moving the pixels.

*homography is a special case of transformation

We will use the Homography

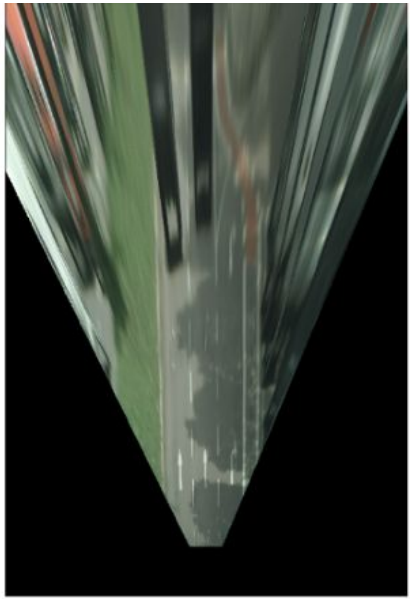




from Hartley & Zisserman

https://docs.opencv.org/master/d9/dab/tutorial_homography.html

We will use the Homography



https://towardsdatascience.com/a-hands-on-application-of-homography-ipm-18d9e47c152f

Let’s code it!

About the code

The code is written in **python**. More specifically, python3.6

We will be using **OpenPose** for people detection, and it's written in C++.
Luckily for us, OpenPose also exposes the python bindings.

Python modules requirements:

- **OpenCV**
- **Numpy**

The code is available in a **Google Colab** Notebook. I will place here snippets of the code, but if you want you can also follow the code on the notebook.

https://colab.research.google.com/drive/1X2Sc0XBhOxAJ5gSuTo5qon3VzagLs3PI?usp=sharing

Downloading and installing OpenPose

In this section we will see the procedure to install this particular neural network for people detection. We will be using the latest version (as Jan 2021 is the 1.7.0).

Building from source

This step requires a lot of time (5-10 minutes at least), so make yourself a coffe or a tea while waiting for the compiler.

```
[ ]  import os
     from os.path import exists, join, basename, splitext

     git_repo_url = 'https://github.com/CMU-Perceptual-Computing-Lab/openpose.git'
     project_name = splitext(basename(git_repo_url))[0]

     if not exists(project_name):
       # see: https://github.com/CMU-Perceptual-Computing-Lab/openpose/issues/949
       # install new CMake becaue of CUDA10
       !wget -q -O cmake-3.13.0-Linux-x86_64.tar.gz https://cmake.org/files/v3.13/cmake-3.13.0-Linux-x86_64.tar.gz
       !tar xfz cmake-3.13.0-Linux-x86_64.tar.gz --strip-components=1 -C /usr/local
       # clone openpose
       !git clone -q --depth 1 $git_repo_url
       # !git clone $git_repo_url

       # install system dependencies
       !apt-get -qq install -y libatlas-base-dev libprotobuf-dev libleveldb-dev libsnappy-dev libhdf5-serial-dev
       !apt-get -qq install -y protobuf-compiler libgflags-dev libgoogle-glog-dev liblmdb-dev opencl-headers ocl-icd-opencl-dev libviennacl-dev

       # openpose version 1.6
       # !cd openpose && git checkout 3d057691b219dddf264c6e412a4560ac8a12dedb

       # fix to the caffe lib
       !sed -i 's/execute_process(COMMAND git checkout master WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}\/3rdparty\/caffe)/
       'execute_process(COMMAND git checkout f019d0dfe86f49d1140961f8c7dec22130c83154 WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}\/3rdparty\/caffe)/g' openpose/CMakeLists.txt

       # build openpose
       !cd openpose && rm -rf build || true && mkdir build && cd build && cmake -DBUILD_PYTHON=ON .. && make -j`nproc` && sudo make install
```

Add to path and import pyopenpose

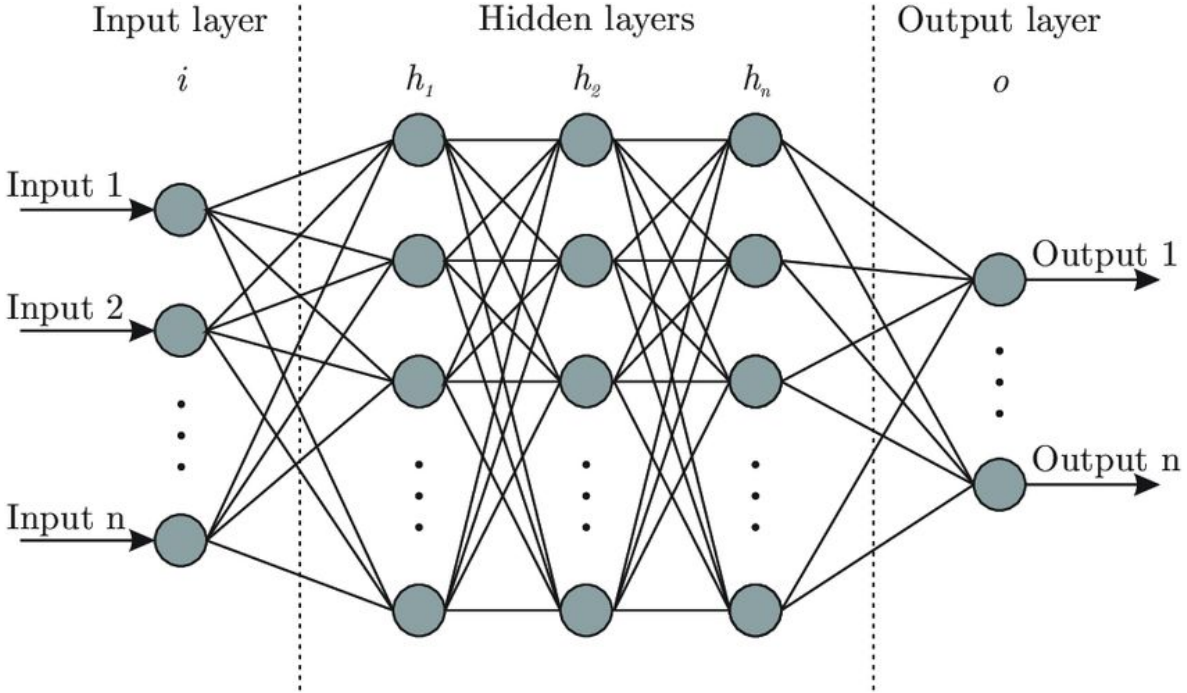
Now, in order to let the python interpreter know where the openpose bindings are located, we need to add to the pythonpath the installation folder.

This can be done during runtime with the sys module. Sometimes, a runtime reset (!!NOT FACTORY RESET!! or you will have to rebuild, losing a lot of time) may be necessary if the import does throw error.

```
[ ]  # YOU MAY HAVE TO REBOOT RUNTIME AFTER BUILD; Runtime > Reset Runtime

     import sys
     sys.path.append('/usr/local/python')

     # Installation verify (optional)
     try:
       from openpose import pyopenpose as op
     except ImportError:
       sys.stderr.writelines('Error during installation of OpenPose, are you sure about the BUILD_PYTHON option? Maybe try resettng the Runtime')
       sys.exit(-1)
```

Remember how the Neural Networks works

IMAGE



PEOPLE

```
## Using python API bindings WrapperPython from pyopenpose
# opWrapper = op.WrapperPython()
# params = dict()
# params["model_folder"] = "openpose/models/"

# opWrapper.configure(params)
# opWrapper.start()
# datum = op.Datum()

# datum.cvInputData = frame

# opWrapper.emplaceAndPop(op.VectorDatum([datum]))
```

```
## Using python API bindings WrapperPython from pyopenpose
# opWrapper = op.WrapperPython()
# params = dict()
# params["model_folder"] = "openpose/models/"

# opWrapper.configure(params)
# opWrapper.start()
# datum = op.Datum()

# datum.cvInputData = frame

# opWrapper.emplaceAndPop(op.VectorDatum([datum]))
```

OpenPose python wrapper

```
## Using python API bindings WrapperPython from pyopenpose
# opWrapper = op.WrapperPython()
# params = dict()
# params["model_folder"] = "openpose/models/"

# opWrapper.configure(params)
# opWrapper.start()
# datum = op.Datum()

# datum.cvInputData = frame

# opWrapper.emplaceAndPop(op.VectorDatum([datum]))
```

OpenPose python wrapper

Execution Parameters: #GPU, network location on disk, resolution parameters, etc..

```
## Using python API bindings WrapperPython from pyopenpose
# opWrapper = op.WrapperPython()
# params = dict()
# params["model_folder"] = "openpose/models/"

# opWrapper.configure(params)
# opWrapper.start()
# datum = op.Datum()

# datum.cvInputData = frame

# opWrapper.emplaceAndPop(op.VectorDatum([datum]))
```

OpenPose python wrapper

Execution Parameters: #GPU, network location on disk, resolution parameters, etc..

The input image frame (e.g. single frame of the video)

```
## Using python API bindings WrapperPython from pyopenpose
# opWrapper = op.WrapperPython()
# params = dict()
# params["model_folder"] = "openpose/models/"

# opWrapper.configure(params)
# opWrapper.start()
# datum = op.Datum()

# datum.cvInputData = frame

# opWrapper.emplaceAndPop(op.VectorDatum([datum]))
```

OpenPose python wrapper

Execution Parameters: #GPU, network location on disk, resolution parameters, etc..

The input image frame (e.g. single frame of the video)

Extracts the people skeletons

Here I am defining a custom wrapper. This wrapper is not strictly necessary, but it's a shortcut for performing the inference (i.e. extracting the human bodies coordinates from the image).

```
[ ]  from openpose import pyopenpose as op

     class OpenPoseWrapper:
         def __init__(self, params=None):
             if params is None:
                 params = self.default_config()

             # Starting OpenPose
             self.opWrapper = op.WrapperPython()
             self.opWrapper.configure(params)
             self.start()

             self.datum = op.Datum()
             self.preprocessing = None

         def start(self):
             self.opWrapper.start()

         def set_image(self, image):
             """
             Set the opencv image (numpy ndarray [h,w,3] into the openpose datum.
             If self.preprocessing function is set, the function is applied.
             :param image:
             :return:
             """
             if self.preprocessing is not None and callable(self.preprocessing):
                 image = self.preprocessing(image)

             # Assign input image to openpose
             self.datum.cvInputData = image
```

```
def default_config(self):
    """
    Compute openpose params from arguments
    :return:
    """
    # Custom Params (refer to include/openpose/flags.hpp for more parameters)
    params = dict()

    # Openpose params
    # Model path
    params["model_folder"] = "openpose/models"

    # Face disabled
    params["face"] = False

    # Hand disabled
    params["hand"] = False

    # Net Resolution
    params["net_resolution"] = "-1x368"

    # Gpu number
    params["num_gpu"] = 1  # Set GPU number

    # Gpu Id
    params["num_gpu_start"] = 0  # Set GPU start id (not considering previous)

    return params

def get_keypoints(self, image):
    """
    Set the input ndarray image [h,w,3] as datum cvInput and compute the keypoints prediction.
    :param image:
    :return: keypoints with prediction associated as [n_body, n_keypoints, 3], where [:,:,0:2] are keypoints and [:,:,2] are the confidences
    """
    self.set_image(image)
    # self.opWrapper.emplaceAndPop([self.datum])
    self.opWrapper.emplaceAndPop(op.VectorDatum([self.datum]))
    keypoints = self.datum.poseKeypoints

    return keypoints

def stop(self):
    self.opWrapper.stop()
```

Download a test video from YouTube

For this demo we will use a widely known video called TownCentreXVID from a famous dataset used in a lot of benchmarks.

```
[ ]  # YouTube TownCentreXVID video
     # https://www.youtube.com/watch?v=ZVmPaCZY3Mc
     YOUTUBE_ID = 'ZVmPaCZY3Mc'


     # YouTube Mall
     # https://www.youtube.com/watch?v=WvhYuDvH17I
     # YOUTUBE_ID = 'WvhYuDvH17I'

     from IPython.display import YouTubeVideo
     YouTubeVideo(YOUTUBE_ID)
```

Saving the video file locally

The video will be downloaded and saved locally.

Also for convenience we will extract only a few seconds (optional step ofc).

```
[ ]  !pip install -q youtube-dl

     max_seconds = 30
     file_video_target = "video.mp4"
     fname = file_video_target.split('.')[0]

     !rm -rf $file_video_target &> /dev/null
     # download the youtube with the given ID
     !youtube-dl -f 'bestvideo[ext=mp4]' --output $fname".%(ext)s" https://www.youtube.com/watch?v=$YOUTUBE_ID
     # cut the first X seconds
     if max_seconds is not None:
       print(f'Extracting only {max_seconds} seconds')
       !ffmpeg -y -loglevel info -i $file_video_target -t $max_seconds $file_video_target &> /dev/null

     print('File ready to use')
```

```
[youtube] ZVmPaCZY3Mc: Downloading webpage
[youtube] ZVmPaCZY3Mc: Downloading MPD manifest
[dashsegments] Total fragments: 48
[download] Destination: video.mp4
[download] 100% of 77.35MiB in 00:06
Extracting only 30 seconds
File ready to use
```

Utilities for OpenPose

Here we will implement some utility functions that will be used later in the code.

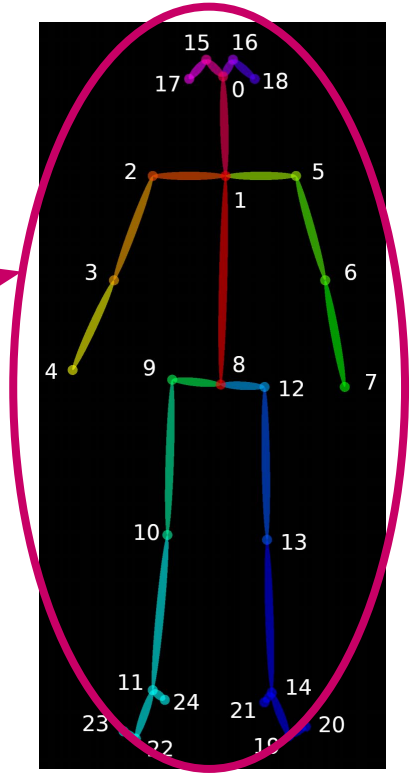
```
[ ]  import numpy as np
     import cv2

     # Openpose visualization
     # ================================================================

     def draw_skeleton(frame, keypoints, colour=(255,0,0), dotted=False):
         connections = [(0, 16), (0, 15), (16, 18), (15, 17),
                        (0, 1), (1, 2), (2, 3), (3, 4),
                        (1, 5), (5, 6), (6, 7), (1, 8),
                        (8, 9), (9, 10), (10, 11),
                        (8, 12), (12, 13), (13, 14),
                        (11, 24), (11, 22), (22, 23),
                        (14, 21), (14, 19), (19, 20)]

         for x, y in keypoints:
             if 0 in (x, y):
                 continue
             center = int(round(x)), int(round(y))
             cv2.circle(frame, center=center, radius=4, color=colour, thickness=-1)

         for keypoint_id1, keypoint_id2 in connections:
             x1, y1 = keypoints[keypoint_id1]
             x2, y2 = keypoints[keypoint_id2]
             if 0 in (x1, y1, x2, y2):
                 continue
             pt1 = int(round(x1)), int(round(y1))
             pt2 = int(round(x2)), int(round(y2))
             cv2.line(frame, pt1=pt1, pt2=pt2, color=colour, thickness=2)
         return frame
```

```
def centroid(pts, exclude_zero=True):
    pts = np.asarray(pts)
    assert len(pts.shape) == 2 and pts.shape[1] == 2, 'Array must be [n,2]'

    def centroidnp(arr):
        length = arr.shape[0]
        sum_x = np.sum(arr[:, 0])
        sum_y = np.sum(arr[:, 1])
        res = np.asarray([sum_x / length, sum_y / length])
        return res

    if exclude_zero:
        # true if there is at least one non-zero element.
        # with axis=1 checking the 1 axis, so columns.
        # i.e. looking for rows that has column with at least one true (!= 0)
        # valid_pts = np.any(pts, axis=1)
        valid_pts = pts[np.any(pts, axis=1), :]

        if valid_pts.shape[0] > 0:
            return centroidnp(valid_pts)
        else:
            raise Exception('No valid point in pts. Set exclude_zero to False')
    else:
        return centroidnp(pts)
```

Utilities for OpenPose

Here we will implement some utility functions that will be used later in the code.

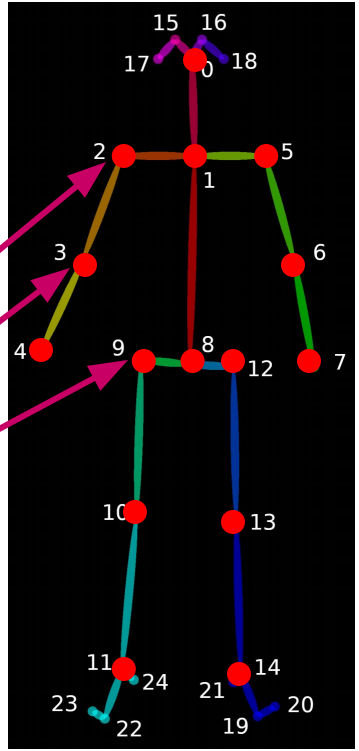
```
[ ]  import numpy as np
     import cv2

     # Openpose visualization
     # ================================================================

     def draw_skeleton(frame, keypoints, colour=(255,0,0), dotted=False):
         connections = [(0, 16), (0, 15), (16, 18), (15, 17),
                        (0, 1), (1, 2), (2, 3), (3, 4),
                        (1, 5), (5, 6), (6, 7), (1, 8),
                        (8, 9), (9, 10), (10, 11),
                        (8, 12), (12, 13), (13, 14),
                        (11, 24), (11, 22), (22, 23),
                        (14, 21), (14, 19), (19, 20)]

         for x, y in keypoints:
             if 0 in (x, y):
                 continue
             center = int(round(x)), int(round(y))
             cv2.circle(frame, center=center, radius=4, color=colour, thickness=-1)

         for keypoint_id1, keypoint_id2 in connections:
             x1, y1 = keypoints[keypoint_id1]
             x2, y2 = keypoints[keypoint_id2]
             if 0 in (x1, y1, x2, y2):
                 continue
             pt1 = int(round(x1)), int(round(y1))
             pt2 = int(round(x2)), int(round(y2))
             cv2.line(frame, pt1=pt1, pt2=pt2, color=colour, thickness=2)
         return frame
```



Utilities for OpenPose

Here we will implement some utility functions that will be used later in the code.

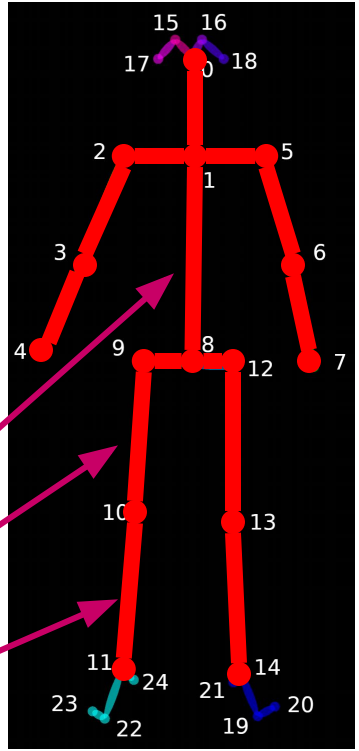
```
[ ]  import numpy as np
     import cv2

     # Openpose visualization
     # ================================================================

     def draw_skeleton(frame, keypoints, colour=(255,0,0), dotted=False):
         connections = [(0, 16), (0, 15), (16, 18), (15, 17),
                        (0, 1), (1, 2), (2, 3), (3, 4),
                        (1, 5), (5, 6), (6, 7), (1, 8),
                        (8, 9), (9, 10), (10, 11),
                        (8, 12), (12, 13), (13, 14),
                        (11, 24), (11, 22), (22, 23),
                        (14, 21), (14, 19), (19, 20)]

         for x, y in keypoints:
             if 0 in (x, y):
                 continue
             center = int(round(x)), int(round(y))
             cv2.circle(frame, center=center, radius=4, color=colour, thickness=-1)

         for keypoint_id1, keypoint_id2 in connections:
             x1, y1 = keypoints[keypoint_id1]
             x2, y2 = keypoints[keypoint_id2]
             if 0 in (x1, y1, x2, y2):
                 continue
             pt1 = int(round(x1)), int(round(y1))
             pt2 = int(round(x2)), int(round(y2))
             cv2.line(frame, pt1=pt1, pt2=pt2, color=colour, thickness=2)
         return frame
```



Utilities for OpenPose

Here we will implement some utility functions that will be used later in the code.

```
[ ]  import numpy as np
     import cv2

     # Openpose visualization
     # ================================================================

     def draw_skeleton(frame, keypoints, colour=(255,0,0), dotted=False):
         connections = [(0, 16), (0, 15), (16, 18), (15, 17),
                        (0, 1), (1, 2), (2, 3), (3, 4),
                        (1, 5), (5, 6), (6, 7), (1, 8),
                        (8, 9), (9, 10), (10, 11),
                        (8, 12), (12, 13), (13, 14),
                        (11, 24), (11, 22), (22, 23),
                        (14, 21), (14, 19), (19, 20)]

         for x, y in keypoints:
             if 0 in (x, y):
                 continue
             center = int(round(x)), int(round(y))
             cv2.circle(frame, center=center, radius=4, color=colour, thickness=-1)

         for keypoint_id1, keypoint_id2 in connections:
             x1, y1 = keypoints[keypoint_id1]
             x2, y2 = keypoints[keypoint_id2]
             if 0 in (x1, y1, x2, y2):
                 continue
             pt1 = int(round(x1)), int(round(y1))
             pt2 = int(round(x2)), int(round(y2))
             cv2.line(frame, pt1=pt1, pt2=pt2, color=colour, thickness=2)
         return frame
```

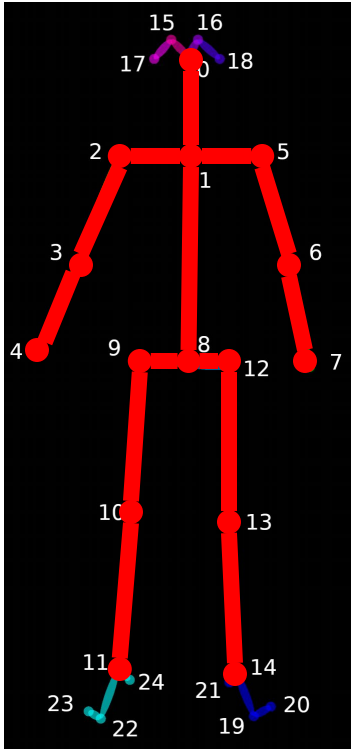


Utilities for OpenPose

Here we will implement some utility functions that will be used later in the code.

```
[ ]  import numpy as np
     import cv2

     # Openpose visualization
     # ======================================

     def draw_skeleton(frame, keypoints, colour=
         connections = [(0, 16), (0, 15), (16, 1
                        (0, 1), (1, 2), (2, 3),
                        (1, 5), (5, 6), (6, 7),
                        (8, 9), (9, 10), (10, 11),
                        (8, 12), (12, 13), (13, 14),
                        (11, 24), (11, 22), (22, 23),
                        (14, 21), (14, 19), (19, 20)]

         for x, y in keypoints:
             if 0 in (x, y):
                 continue
             center = int(round(x)), int(round(y))
             cv2.circle(frame, center=center, radius=4, color=colour, thickness=-1)

         for keypoint_id1, keypoint_id2 in connections:
             x1, y1 = keypoints[keypoint_id1]
             x2, y2 = keypoints[keypoint_id2]
             if 0 in (x1, y1, x2, y2):
                 continue
             pt1 = int(round(x1)), int(round(y1))
             pt2 = int(round(x2)), int(round(y2))
             cv2.line(frame, pt1=pt1, pt2=pt2, color=colour, thickness=2)
         return frame
```

For OpenPose if a joint has x or y coordinate equals to zero, it means the joint either is not reliable or not found by the ML algorithm

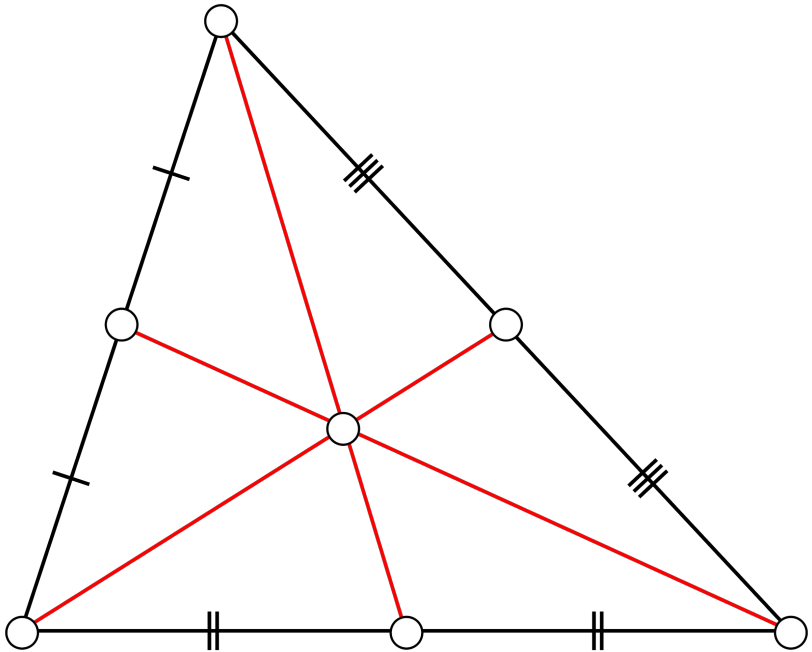


The centroid of a finite set of k points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ in $\mathbb{R}^n$ is

$$\mathbf{C} = \frac{\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_k}{k}$$

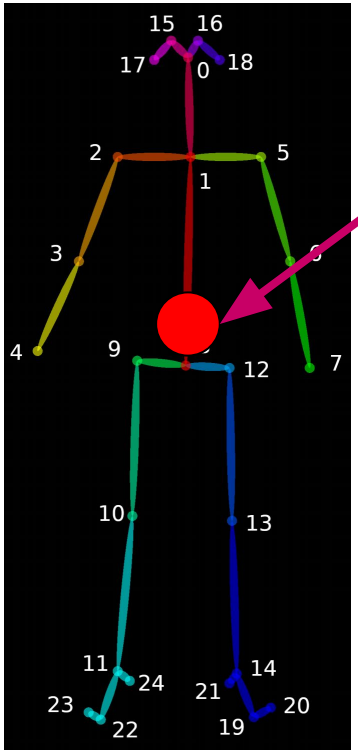
```python
def centroid(pts, exclude_zero=True):
    pts = np.asarray(pts)
    assert len(pts.shape) == 2 and pts.shape[1] == 2, 'Array must be [n,2]'

    def centroidnp(arr):
        length = arr.shape[0]
        sum_x = np.sum(arr[:, 0])
        sum_y = np.sum(arr[:, 1])
        res = np.asarray([sum_x / length, sum_y / length])
        return res

    if exclude_zero:
        # true if there is at least one non-zero element.
        # with axis=1 checking the 1 axis, so columns.
        # i.e. looking for rows that has column with at least one true (!= 0)
        # valid_pts = np.any(pts, axis=1)
        valid_pts = pts[np.any(pts, axis=1), :]

        if valid_pts.shape[0] > 0:
            return centroidnp(valid_pts)
        else:
            raise Exception('No valid point in pts. Set exclude_zero to False')
    else:
        return centroidnp(pts)
```

e.g. Centroid of a Triangle



```
def centroid(pts, exclude_zero=True):
    pts = np.asarray(pts)
    assert len(pts.shape) == 2 and pts.shape[1] == 2, 'Array must be [n,2]'

    def centroidnp(arr):
        length = arr.shape[0]
        sum_x = np.sum(arr[:, 0])
        sum_y = np.sum(arr[:, 1])
        res = np.asarray([sum_x / length, sum_y / length])
        return res

    if exclude_zero:
        # true if there is at least one non-zero element.
        # with axis=1 checking the 1 axis, so columns.
        # i.e. looking for rows that has column with at least one true (!= 0)
        # valid_pts = np.any(pts, axis=1)
        valid_pts = pts[np.any(pts, axis=1), :]

        if valid_pts.shape[0] > 0:
            return centroidnp(valid_pts)
        else:
            raise Exception('No valid point in pts. Set exclude_zero to False')
    else:
        return centroidnp(pts)
```

Centroid (or barycenter if you prefer)




```python
def centroid(pts, exclude_zero=True):
    pts = np.asarray(pts)
    assert len(pts.shape) == 2 and pts.shape[1] == 2, 'Array must be [n,2]'

    def centroidnp(arr):
        length = arr.shape[0]
        sum_x = np.sum(arr[:, 0])
        sum_y = np.sum(arr[:, 1])
        res = np.asarray([sum_x / length, sum_y / length])
        return res

    if exclude_zero:
        # true if there is at least one non-zero element.
        # with axis=1 checking the 1 axis, so columns.
        # i.e. looking for rows that has column with at least one true (!= 0)
        # valid_pts = np.any(pts, axis=1)
        valid_pts = pts[np.any(pts, axis=1), :]

        if valid_pts.shape[0] > 0:
            return centroidnp(valid_pts)
        else:
            raise Exception('No valid point in pts. Set exclude_zero to False')
    else:
        return centroidnp(pts)
```

Reading the video frame. How is it?

Reading the video, to extract a frame.

For semplicity, we use the frame number 0.

```
[ ]  from google.colab.patches import cv2_imshow

     cap = cv2.VideoCapture("video.mp4")

     frame_n = 0
     counter = 0

     while True:
       ret, frame = cap.read()
       if not ret or counter >= frame_n:
         break
       else:
         counter+=1

     im_size = frame.shape
     cv2_imshow(frame)
```

OpenPose in action!

Ok now it's time for the ML part. First of all, we have to extract the people's positions inside a certain image using the OpenPose neural network.

We will use the python bindings provided with OpenPose, and in particular our previously defined wrapper. Then we will check what the network output is, by displaying the bodies in the image.

```
## I am using a custom defined wrapper (over the already existing wrapper.. yeah, I know, let me be)
custom_wrapper = OpenPoseWrapper()
keypoints = custom_wrapper.get_keypoints(frame)  # KxJx3

skeletons = keypoints[:, :, 0:2]  # KxJ

cp_frame = frame.copy()
for skeleton in skeletons:
  # print(skeleton)
  draw_skeleton(cp_frame, skeleton)

cv2_imshow(cp_frame)
```

OpenPose in action!

Ok now it's time for the ML part. First of all, we have to extract the people

We will use the python bindings provided with OpenPose, and in particula
output is, by displaying the bodies in the image.

```
## I am using a custom defined wrapper (over the already exist
custom_wrapper = OpenPoseWrapper()
keypoints = custom_wrapper.get_keypoints(frame)  # KxJx3

skeletons = keypoints[:, :, 0:2]  # KxJ

cp_frame = frame.copy()
for skeleton in skeletons:
  # print(skeleton)
  draw_skeleton(cp_frame, skeleton)

cv2_imshow(cp_frame)
```

Extracts the people's coordinates storing them in a multidimensional array of size: **K** x **J** x **3**
with **K** number of people in the image, **J** number of joints for each person, and **three** values for each joint (X coord, Y coord and Confidence value)

OpenPose in action!

Ok now it's time for the ML part. First of all, we have to extract the people

We will use the python bindings provided with OpenPose, and in particula
output is, by displaying the bodies in the image.

```
## I am using a custom defined wrapper (over the already exist
custom_wrapper = OpenPoseWrapper()
keypoints = custom_wrapper.get_keypoints(frame)  # KxJx3

skeletons = keypoints[:, :, 0:2]  # KxJ

cp_frame = frame.copy()
for skeleton in skeletons:
  # print(skeleton)
  draw_skeleton(cp_frame, skeleton)

cv2_imshow(cp_frame)
```

Extracts the people's coordinates storing them in a multidimensional array of size: **K** x **J** x **3**
with **K** number of people in the image, **J** number of joints for each person, and **three** values for each joint (X coord, Y coord and Confidence value)

Get only the X and Y coordinates from the NN output

OpenPose in action!

Ok now it's time for the ML part. First of all, we have to extract the people

We will use the python bindings provided with OpenPose, and in particula
output is, by displaying the bodies in the image.

```
## I am using a custom defined wrapper (over the already exist
custom_wrapper = OpenPoseWrapper()
keypoints = custom_wrapper.get_keypoints(frame)  # KxJx3

skeletons = keypoints[:, :, 0:2]  # KxJ

cp_frame = frame.copy()
for skeleton in skeletons:
  # print(skeleton)
  draw_skeleton(cp_frame, skeleton)

cv2_imshow(cp_frame)
```

Extracts the people's coordinates storing them in a multidimensional array of size: **K** x **J** x **3**
with **K** number of people in the image, **J** number of joints for each person, and **three** values for each joint (X coord, Y coord and Confidence value)

Get only the X and Y coordinates from the NN output

Draw each person's skeleton in the image


30% off
sweaters

And here comes the magic

Here we will use the homography ~~magic~~... matrix.

Usually, you would use the function called `cv2.findHomography()` as part of the `cv2` module, which does some math operations between two sets of points, obtaining the matrix that we will use for the transformation into the planar projective space.

This is how to do it mathematically, but since some steps are required in order to get a good homography, we use a previous calculated (but not optimized) homography matrix on that video. This matrix has been obtained with a method that is part of an enterprise software.

As a result, we will see the perspective view.

```
[ ]  # HOMOGRAPHY MAGIC

     import json

     # this is how to do it mathematically, but since some steps are required in
     # order to get a correct homography, we use a previous calculated (but not
     # optimized) homography matrix
     def compute_homography(pts1, pts2):
       h, status = cv2.findHomography(pts1, pts2)
       return h

     previous_config_str = '''
     {"homography_matrix":[[1.6467088390692677,5.1418249047541575,-0.455209332524646],
     [0.03714008190854082,4.510218260097977,0.019514961022878197],
     [0.0004313776114372802,0.0025703686312520285,1]],
     "xmax":476,"xmin":-1417,"ymax":428,"ymin":-862}'''
     previous_config = json.loads(previous_config_str)

     h0 = np.asarray(previous_config['homography_matrix'], dtype=np.float32)
     xmax = previous_config['xmax']
     xmin = previous_config['xmin']
     ymax = previous_config['ymax']
     ymin = previous_config['ymin']

     image_rec = cv2.warpPerspective(frame.copy(), h0, (xmax - xmin, ymax - ymin))
     cv2_imshow(image_rec)
```

And here comes the magic

Here we will use the homography ~~magic~~... matrix.

Usually, you would use the function called `cv2.findHomography()` as part of the `cv2` module, which does some math operations between two sets of points, obtaining the matrix that we will use for the transformation into the planar projective space.

This is how to do it mathematically, but since some steps are required in order to get a good homography, we use a previous calculated (but not optimized) homography matrix on that video. This matrix has been obtained with a method that is part of an enterprise software.

As a result, we will see the perspective view.

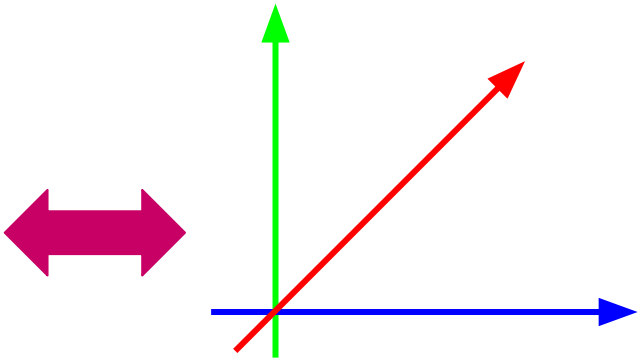
$$s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

```
[ ]  # HOMOGRAPHY MAGIC

     import json

     # this is how to do it mathematically, but since some steps are required in
     # order to get a correct homography, we use a previous calculated (but not
     # optimized) homography matrix
     def compute_homography(pts1, pts2):
       h, status = cv2.findHomography(pts1, pts2)
       return h

     previous_config_str = '''
     {"homography_matrix":[[1.6467088390692677,5.1418249047541575,-0.455209332524646],
     [0.03714008190854082,4.510218260097977,0.019514961022878197],
     [0.0004313776114372802,0.0025703686312520285,1]],
     "xmax":476,"xmin":-1417,"ymax":428,"ymin":-862}'''
     previous_config = json.loads(previous_config_str)

     h0 = np.asarray(previous_config['homography_matrix'], dtype=np.float32)
     xmax = previous_config['xmax']
     xmin = previous_config['xmin']
     ymax = previous_config['ymax']
     ymin = previous_config['ymin']

     image_rec = cv2.warpPerspective(frame.copy(), h0, (xmax - xmin, ymax - ymin))
     cv2_imshow(image_rec)
```


30% off


30% off

Old reference system



New reference system

New reference system

Compact representation

In this section, we will see how to use the actual ML output. First, we will adapt the ML output to our task.

We will transform the body joint of a person to a more compact representation we can use in a single frame. One way to do this is to compute the centroid of the body joints.

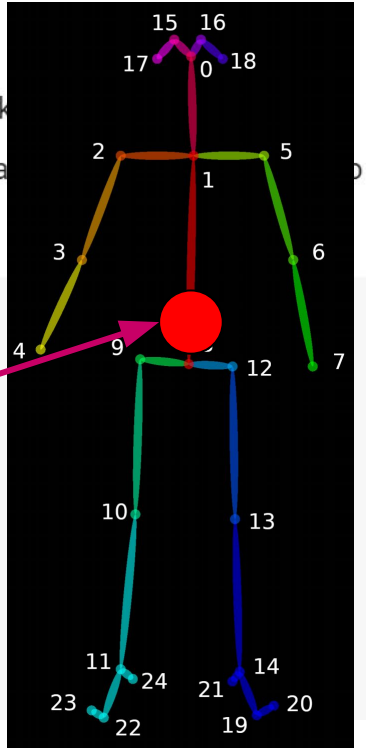
```
[ ]  blue = (255, 0, 0)
     red = (0, 0, 255)

     centroid_frame = frame.copy()

     people = []
     for skeleton in skeletons:
       person_point = list(map(int, centroid(skeleton)))  # compute the centroid of skeleton
       people.append(np.asarray(person_point))  # as a numpy array, for later use
       cv2.circle(centroid_frame, (person_point[0], person_point[1]), 10, blue, -1)

     cv2_imshow(centroid_frame)
```

Compact representation

In this section, we will see how to use the actual ML output. First, we will adapt the ML output to our task

We will transform the body joint of a person to a more compact representation we can use in a single fra o compute the centroid of the body joints.

```
[ ]  blue = (255, 0, 0)
     red = (0, 0, 255)

     centroid_frame = frame.copy()

     people = []
     for skeleton in skeletons:
       person_point = list(map(int, centroid(skeleton)))  # compute the centroid of skeleton
       people.append(np.asarray(person_point))  # as a numpy array, for later use
       cv2.circle(centroid_frame, (person_point[0], person_point[1]), 10, blue, -1)

     cv2_imshow(centroid_frame)
```




30% off

Visualize the people in the perspective view

Now, using the homography matrix, we use project the centroid of each person into the perspective planar view.

```
[ ]  perspective_people = []
     for p in people:
       tmp = p.reshape((-1,1,2)).astype(np.float32)
       perspective_float = cv2.perspectiveTransform(tmp, h0)
       perspective_coords = perspective_float.reshape(1, -1)[0].astype(np.int)
       perspective_people.append(perspective_coords)

     for p in perspective_people:
       cv2.circle(image_rec, (p[0], p[1]), 10, blue, -1)

     cv2_imshow(image_rec)
```

Visualize the people in the perspective view

$$s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Now, using the homography matrix, we use project the centroi ew.

```
[ ]  perspective_people = []
     for p in people:
       tmp = p.reshape((-1,1,2)).astype(np.float32)
       perspective_float = cv2.perspectiveTransform(tmp, h0)
       perspective_coords = perspective_float.reshape(1, -1)[0].astype(np.int)
       perspective_people.append(perspective_coords)

     for p in perspective_people:
       cv2.circle(image_rec, (p[0], p[1]), 10, blue, -1)

     cv2_imshow(image_rec)
```

One last step

At this point we want to check who is the closest person detected in the perspective view for each person.

Compute the distances

Let's create a dictionary (key value pairs), in which we have all the people indexes as keyes, and the index of their closest person as values. We also add the distance from that person (we will use it later).

```
[ ]  def normL2(x, y):
       return np.linalg.norm(np.asarray(x) - np.asarray(y))

     closests_idxs = {}

     for i, p0 in enumerate(perspective_people):
       min_d = float('inf')
       candidate = None
       candidate_idx = None
       for j, p1 in enumerate(perspective_people):
         if np.all(p0 == p1):
           continue
         curr_d = normL2(p0, p1)
         if curr_d < min_d:
           candidate = p1
           candidate_idx = j
           min_d = curr_d

       if candidate is None:
         # handle this case
         continue
       else:
         closests_idxs[i] = (candidate_idx, min_d)
```

One last step

At this point we want to check who is the closest person detected in the perspective view for each person.

Compute the distances

Let's create a dictionary (key value pairs), in which we have all the people indexes as keyes, and the index of their closest person as values. We also add the distance from that person (we will use it later).

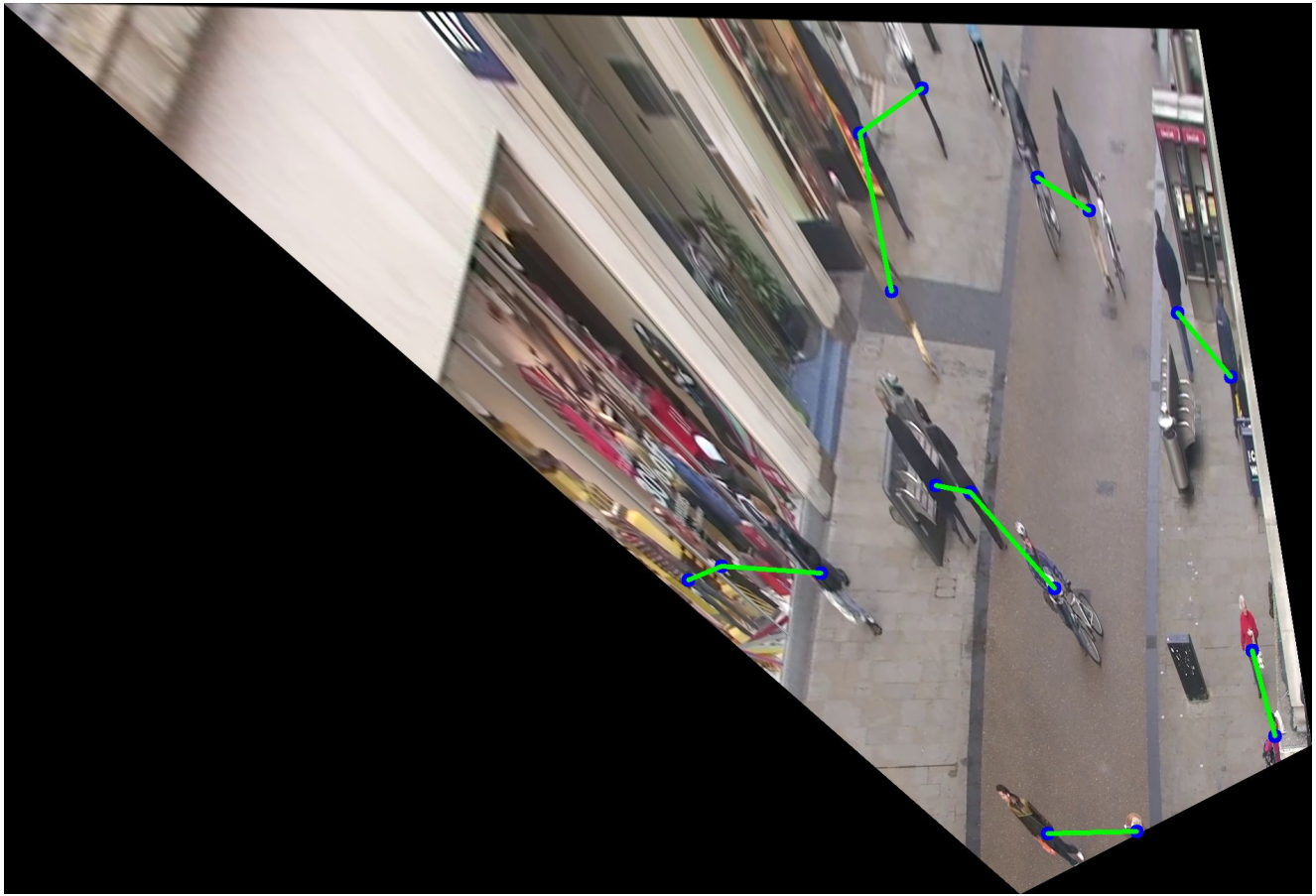
```
[ ]  def normL2(x, y):
       return np.linalg.norm(np.asarray(x) - np.asarray(y))

     closests_idxs = {}

     for i, p0 in enumerate(perspective_people):
       min_d = float('inf')
       candidate = None
       candidate_idx = None
       for j, p1 in enumerate(perspective_people):
         if np.all(p0 == p1):
           continue
         curr_d = normL2(p0, p1)
         if curr_d < min_d:
           candidate = p1
           candidate_idx = j
           min_d = curr_d

       if candidate is None:
         # handle this case
         continue
       else:
         closests_idxs[i] = (candidate_idx, min_d)
```

```
{
   ppl_idx: (closest_idx, distance_val),
   ppl_idx: (closest_idx, distance_val),
   ...
}
```

Draw in the perspective view

Let's draw a line from each person to their closest in the perspective view

```
[17] perspective_final_frame = image_rec.copy()

     green = (0, 250, 0)
     orange = (0, 140, 255)

     for k in closests_idxs.keys():
       closest_idx = closests_idxs[k][0]
       closest = perspective_people[closest_idx]
       pt1 = tuple(map(int, perspective_people[k]))
       pt2 = tuple(map(int, closest))
       cv2.line(perspective_final_frame, pt1, pt2, green, 5)

     cv2_imshow(perspective_final_frame)
```



Draw in the original frame

Let's draw this in the original (unprojected) frame

```
[ ]  final_frame = centroid_frame.copy()

     for k in closests_idxs.keys():
       closest = closests_idxs[k][0]
       pt1 = tuple(map(int, people[k]))
       pt2 = tuple(map(int, people[closest]))
       cv2.line(final_frame, pt1, pt2, green, 5)

     cv2_imshow(final_frame)
```


30% off
Sweaters

Finally...

Ok, but this is not useful by itself. We need to know which of them are too close, how we do it?

Well, sounds like it's time for the step all data scientists most hate, but it is (sometimes) inevitable: thresholding!

I want to highlight the people only when they come closer than 100 units.

The unit can be meters, decimeters, centimeters, pixels, etc... depending on method used in the h0 (homography) calibration.

```
[ ]  real_final_frame = centroid_frame.copy()

     th = 100  # arbitrary, yes, but also customizable ;)

     for k in closests_idxs.keys():
       data = closests_idxs[k]
       closest = data[0]
       value = data[1]
       if value >= th:
         continue

       pt1 = tuple(map(int, people[k]))
       pt2 = tuple(map(int, people[closest]))
       cv2.line(real_final_frame, pt1, pt2, red, 5)

     cv2_imshow(real_final_frame)
```


30% off

Problems

Problems

Limitation of homography approach (math related):

- The main problem is that **camera position must not change**;
- An **initial but crucial calibration** to compensate the scale factor of the H matrix is required;
- Plane is **estimated** (does not work for complex envs with different planes);
- What about image **distortion**?

Problems related to this demo:

- **Centroid** is dependent of people's height: a small variation in height may leads to false perspective distance
- What about groups of people for whom it's "ok" to be close? (**families** with child, group of **friends**, ecc..)
- **Time analysis**: we only have the current instance without identification or density over time
- **Accuracy of neural network** directly affects the overall performances (false positives - false negatives)

Some enterprise application

Thank you

Any questions?