Università
della
Svizzera
italiana

Faculty
of Informatics

Bachelor Thesis

June 23, 2020

Graph Representations to Model Physical Systems

Transforming images and real-time videos into attributed graphs

Gabriel Henrique Carraretto

Abstract

In this project we build an all-in-one framework to generate a set, or a sequence, of human skeletal graphs describing human poses from images and videos. The framework complies with standard graph machine learning data representations, and it is meant to boost the researcher work in designing new machine learning methods based on graphs. The 2D multi-person detection mechanism allows to identify keypoints of human bodies, as well as, faces, hands and feet. Those keypoints can then be used to feed graph processing systems to address problems such as action detection and classification. The tool is implemented in Python, and is compatible with different input types such as images, video or even real-time data streams, such as those coming from a webcam. The output graphs are stored in a versatile JSON format, that can be further transformed with some built-in parsers in a more practical format such as the NetworkX format or an adjacency matrix. The code used for implementing the pipeline is open-source and available at: `https://github.com/gabecarra/GraphPipe`.

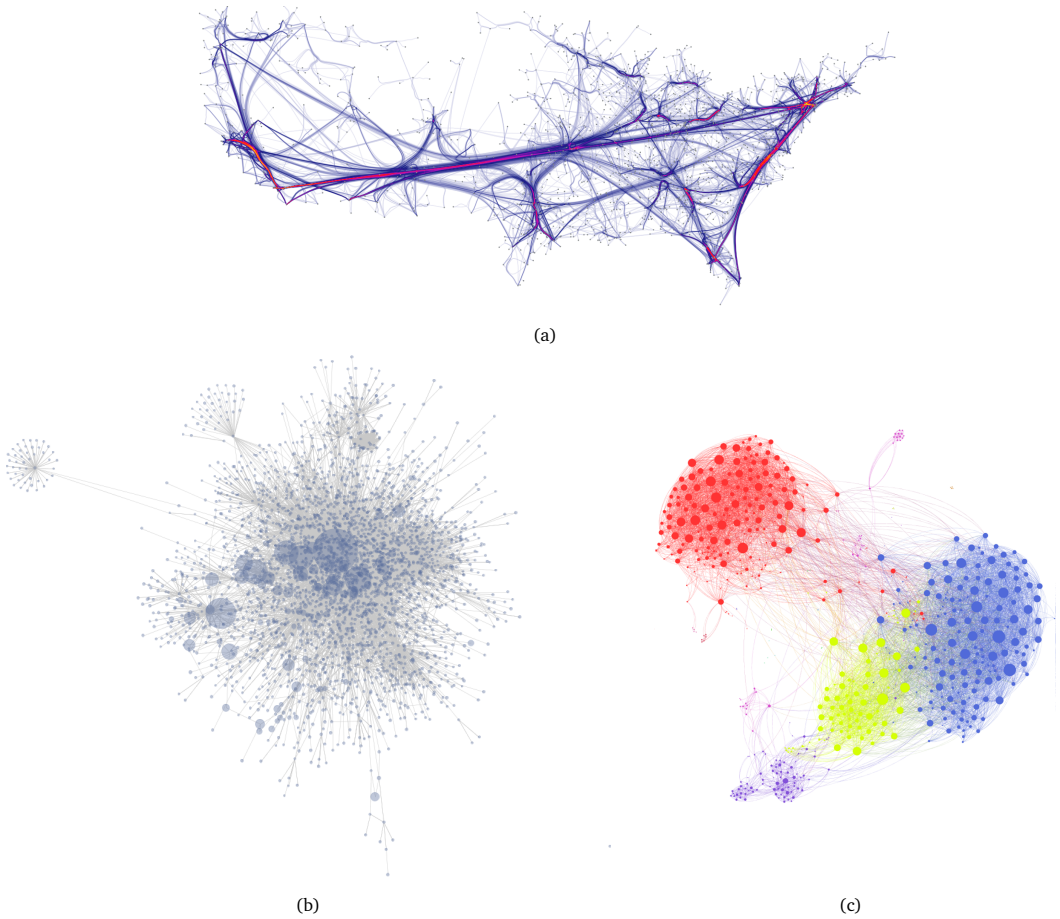
Advisor
Prof. Cesare Alippi
Assistants
Andrea Cini, Daniele Zambon

Advisor's approval (Prof. Cesare Alippi): Date:

1 Introduction

1.1 Graphs

Graphs are mathematical structures used to model pairwise relations between entities. Each entity in a graph is called node, and each relation is called edge. Graphs are one of the objects of study in discrete mathematics, and they can be used to model many types of relations and processes in physical, biological, social and informational systems (see Figure 1), for example: [24, 34, 15, 43]. Usually, dealing with graphs can be a challenging task, but their effectiveness is decisive for solving many complex problems such as combinatorial optimization [23], boolean satisfiability [35], Turing machines modeling [22], and many others. Even though great progress has been made in graph research in the past years, there is still a lot of research to be done, and this project can help accelerating the development of new tecnologies, by provinding an useful tool for graph manipulation.



(a)

(b)

(c)

Figure 1. Some examples of graph structures: a US migrations force-directed graph representation, b Links between pages on a wiki, c Graph clustering of a Facebook network

We can have problems related to optimization, that can be solved using graph theory (e.g. min-cut max-flow) or problems with data representing entities, which have functional or structural relations that are not directly described by the raw data. Representing the given data as a graph or a set of graphs, takes into count the relational information, enriching the whole representation, which sometimes can show interesting patterns that can be analysed or studied (e.g. social networks). We can also have some data were not all the given information is strictly necessary (redundant information), thus adding some inductive bias by representing the data as a graph helps narrow the problem, increasing its solution effectiveness and making it easier to be solved (e.g. from people photos to skeleton graphs).
Graphs have also gained some attention in the field of machine learning (ML). Usually, ML graph tasks can be classified as: node-level, where classification/regression is applied to each node of a graph (from nodes to labels), edge-level, where graph-level, where an input is represented as a graph, and information is extracted taking in count the structure of the input (from graphs to labels) and some other type of tasks that takes a graph as input and returns another graph as output, such as graph sparsification, pooling, generation ecc.
This project deals mainly with the idea of extracting rich but compact representations of the data. Basically, given some data with redundant information, the goal is to transform the data by representing it as a graph or a set of graphs, in order to be used as an input of a graph-processing system. Node-level tasks and some other graph-to-

graph problems can also be tackled, but they are not explicitly addressed here. The given project deals with graph creation for feeding machine learning algorithms, unlike most of the algorithms used nowadays that operates over real number vectors, yielding to greater efficiency in many applications.
Since, in general, graphs have arbitrary topologies, it is not always possible to define an ordering or a criterion on how to traverse it. In order to feed graph data into a graph-processing system, so-called embedding frameworks are commonly used. They basically perform a mapping between each node or edge of a graph to a vector. A large number of frameworks has been designed in order to encode graph information into low-dimensional real number vectors of fixed length. Frameworks like node2vec [19] and DeepWalk [19] gained a lot on popularity since their inception, due to their simplicity and effectiveness in transforming graphs into a convenient format for machine learning processing.
The research in the field of graph-based machine learning experienced a good increment in the past few years. One technique gaining a lot of attention recently is graph neural network (GNN). The GNN model is particularly suited for problems whose domain can be represented by a set of patterns and relationships between them. In those problems, a prediction about a given pattern can be carried out exploiting all the related information, which includes the pattern features, the pattern relationships and, in general, the whole graph that represents the domain. GNN peculiarity consists in its capability of maintaining the processing in the graph domain (see an example of GNN in Figure 2).

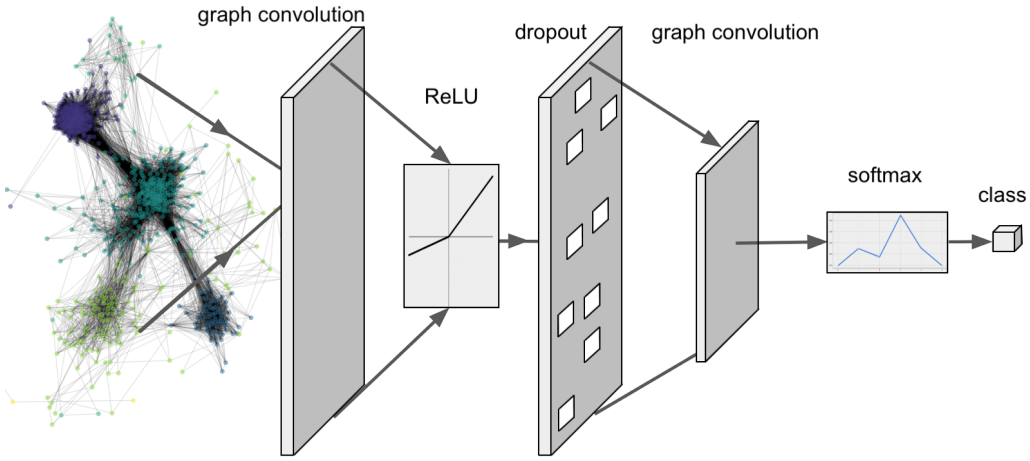


Figure 2. Example of convolutional graph neural network

The idea of GNNs has been introduced in [17], and was based on a recurrent architecture. Apart from the different variants of graph neural networks introduced, general frameworks were proposed to integrate different models into one single framework. In 2018, [5] provided a generalization and extension of various graph neural network approaches, unifying most of the existing methods in a framework called graph networks (GN), that supports constructing complex architectures from simple building blocks called GN blocks(see Figure 3).

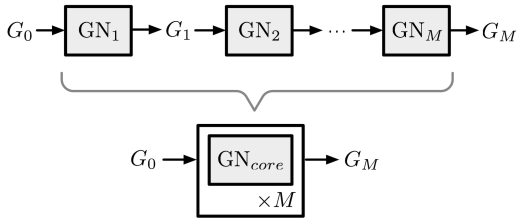
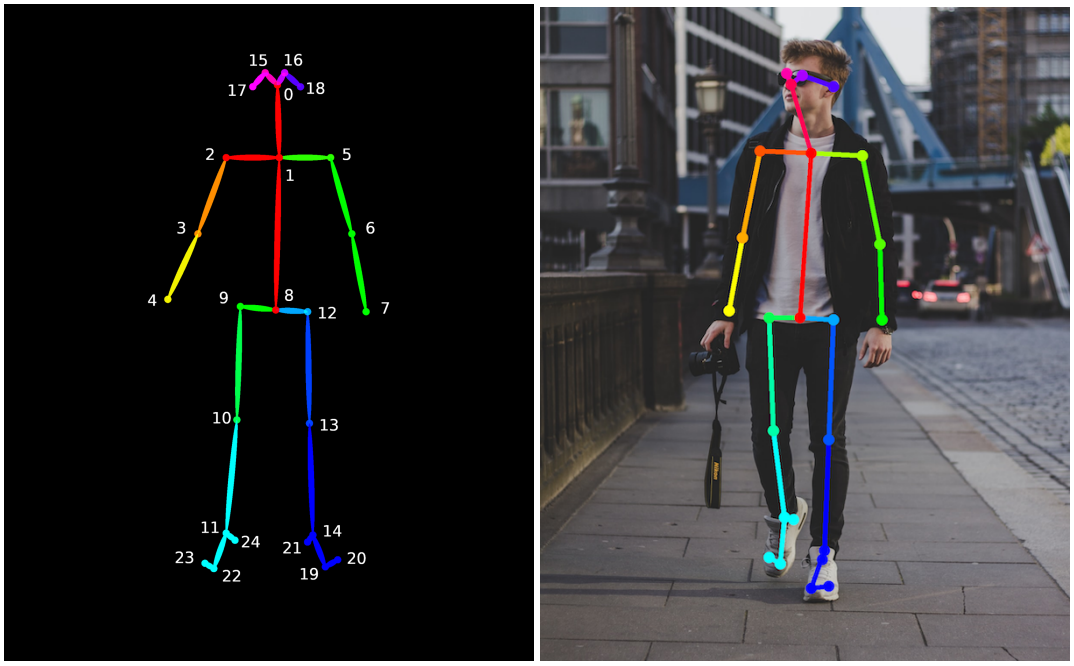


Figure 3. Example of a composition of GN blocks

The communication process between the blocks can be generalized with the concept of messaging passing neural network [16], where for each node, the neighbourhood information is collected, aggregated and passed to the next instance. The framework is extremely flexible and applicable to a wide range of domains ranging from perception, language, and symbolic reasoning.

1.2 Human pose estimation

Human pose estimation is an important problem that has enjoyed the attention of the computer vision community for the past few decades. In general it is the process of making inference of the relative position of keypoints of the human body with respected to some reference system. It's also important to note that pose estimator has various sub-tasks such as single pose estimation, multi-pose estimation and estimation of temporally coherent sequences of poses (tracking). It has a wide range of applications and a key component of systems performing action recognition, animation, gaming, etc.
The result of the pose estimation process is a pose skeleton that represents the orientation of a person in a given frame (see Figure 4). Each coordinate in the skeleton is known as a keypoint (or a joint). A valid connection between two parts is known as a pair (or a limb).
Essentially, the human pose estimation process identifies a series of keypoints, represented as set of 3D (x, y, z) or 2D (x, y) coordinates. Each keypoint contain all the needed information for its identification, and each part is created following a well defined structure. The fact that there is some distinction between body parts, allow to apply a graph-like structure to the skeleton, facilitating the keypoint assignment to the image.



(a) (b)

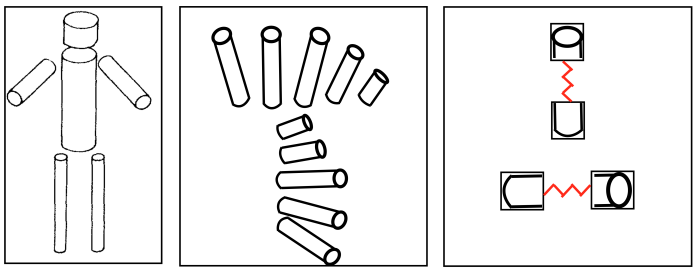
Figure 4. a BODY_25 pose skeleton, b Human pose estimation with BODY_25

The idea of a pose estimation can be extended to individual body parts such as key points of a hand, a face or a foot, and sometimes the body keypoint predictions can help detect more precisely those additional parts.
Depending on the number of people being tracked, pose estimation can be classified into Single-person and Multi-person pose estimation. Single-person pose estimation (SPPE) is the easier of the two, with the assumption of only one person present in the frame. On the other hand, Multi-person pose estimation (MPPE) needs to handle the additional problem of inter-person occlusion.
The variability in the number of subjects can make pose estimation quite challenging, depending on the input given, and involves detecting the number of objects (persons) present in the input frame. For this reason, initial approaches in pose estimation were mostly focused on SPPE, however, with the availability of huge multi-person datasets and the introduction of more complex approaches, the MPPE problem has lately been getting increased attention.
Pose estimation is a difficult problem also due to the human composition: The degree of freedom of an human body, possible partial occlusions of subjects, and a large heterogeneity introduced by differences in appearance such as clothing, body shape, etc..., can generate ambiguity in the results. Finally, most algorithms estimate pose from 2D images, taken from a normal camera, that could include issues due to variability of lighting and camera configurations.

1.2.1 Classical approaches

One of the classical approaches to articulated pose estimation is the pictorial structures framework [3]. The basic idea is to represent an object by a collection of parts arranged in a deformable configuration (see Figure 5a). Each

part can take different orientations, resulting in different joints of the same object (see Figure 5b). Different scales and orientations of the main object can be articulated to scales and orientations of the corresponding parts. A part is an appearance template which is matched in an image. The parts are attached to each other via a set of deformable connections called springs (see Picture 5c) that help control the overall placement of the various parts of the model.



(a) (b) (c)

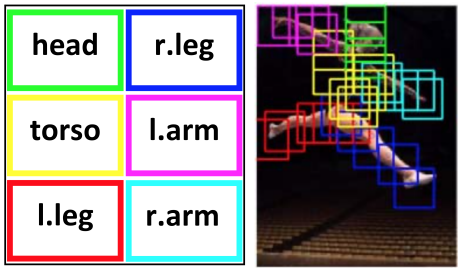
Figure 5. c classic articulated limb model, **??** different orientation and foreshortening states of a limb, **??** pictorial structure approximation

A geometric constraint is set on the orientation of springs, for example limbs of legs cannot move 360 degrees, hence parts cannot have such an extreme orientation, in order to reduce the number of possible permutations, improving the execution time of the algorithm. When parts are parameterized by pixel location and orientation, the resulting structure can model articulation which is very relevant in pose estimation (see Figure 6).

(a) (b) (c)

Figure 6. Examples of human pose estimation using pictorial structures framework

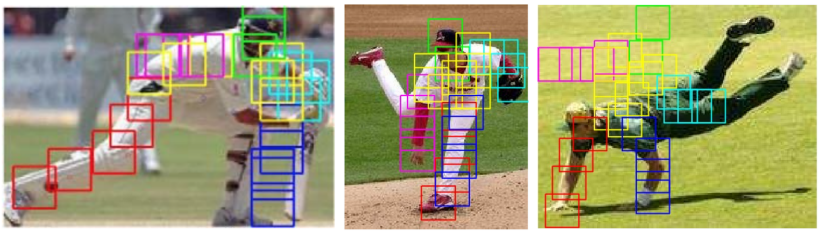
The pictorial structures framework method, however, comes with the limitation of having a pose model not depending on image data. Being an approach based only on engineered features, it often fails in capturing the subtleties present in real world data and does not scale with the complexity of the problem. As a result, research has focused on enriching the representational power of the models.
Another classical approach is called deformable part models [48]. It uses a mixture model of parts which expresses complex joint relationships. Deformable part models are a collection of templates arranged in a deformable configuration and each model has a global template and many part templates (see Figure 7).



(a) (b)

Figure 7. Deformable parts model

The part-based model can model articulations well due to a general, flexible mixture model that augments standard spring models encoding spatial relations. This is however achieved at the cost of limited expressiveness and does not take into account global context (see Figure 8).



(a) (b) (c)

Figure 8. Examples of deformable part models limitations

1.2.2 Deep Learning based approaches

The methods described in 1.2.1 have something in common: they estimates the pose of a single person in an image by identifying its individual parts first, followed by forming connections between them to create the pose. Those methods are not particularly useful in many real-life scenarios where images may contain multiple people or unusual poses. With the introduction of deep learning based approaches, a multi-person pose estimation was made possible. .Most of the times, the location and the number of people in an image are unknown. In multi-person pose estimation, typically, this problem can be solved using one of two approaches: bottom-up and top-down. The top-down approach (see Figure 9) starts by identifying and localizing individual person instances using a bounding box object detector. This is then followed by estimating the pose of a single person. The bottom-up approach (see Figure 10) starts by localizing identity-free semantic entities, then associating/grouping them into distinct person instances.

(a) (b) (c)

Figure 9. Example of a top-down approach: a input image, b object detection application, c human pose estimation

(a) **(b)** **(c)**

Figure 10. Example of a bottom-up approach: a input image, b identity-free semantic entities localization, c human pose estimation

A first attempt to use deep neural networks (DNNs) to tackle this problem was given in 2014, with the DeepPose [40] approach. DeepPose formulates the pose estimation problem as a DNN-based regression task aimed at predicting the position of the keypoints. The location of each body joint is inferred using as input the full image and a convolutional neural network (CNN).
The use of CNNs introduce two main advantages: first, the CNN is capable of capturing the full context of each body joint. Second, the approach is substantially simpler to formulate than methods based on graphical models, since

there is no need to explicitly design a model topology and interactions between joints. Instead, it learns to solve the problem in an end-to-end fashion. In order to increase the precision of joint localization, a cascade of CNN-based pose predictors is used. Starting with an initial pose estimation (see Figure 11a), based on the full image, we learn CNN-based regressors which refines the joint predictions by using higher resolution sub-images (see Figure 11b).

Figure 11. Schematic view of the DNN-based pose regression: a represents the initial state, where the entire image is used to feed the DNN. The convolutional layers are represented in blue, while fully connected ones are in green, b a refining regressor is applied on a sub image to refine a prediction from the previous stage

The application of DNNs to human pose estimation brought the advantage of working with pose estimation in a holistic fashion, allowing a higher precision pose estimate. Many state of the art results where improved on standard benchmarks such as the MPII, LSP, and FLIC datasets. Since then, multiple models where presented, introducing a lot of innovations and variations, such as the usage of heatmap regression models[39] to improve the calculations related with the probability of a joint occurring at a given spatial location in the image, or a self-correcting model [10] that feeds back the error from previous predictions, or even a stacked hourglass model [29], that applies repeated bottom-up and top-down processing with intermediate supervision in order to obtain an improvement in performance.

1.3 Motivation

In section 1.1 we have seen the importance and advantages of representing gathered information as a graph, and analysed the different problems that can be simplified modeling the problem using graph-based abstractions, such as optimization problems, structural and functional data representation, and data limitation/simplification. Also different types of graph application in machine learning tasks were introduced, such as node-level, graph-level and other more complex applications such as pooling and sparsification. In section 1.2, we underlined the interest for human pose estimation and graph theory applied to machine learning tasks and how it increased during the last few years. The research in this field is vast, both in terms of width and depth and the scope of pose estimation is a compelling open research problem. One of the main reasons for the success of deep learning and Human Pose Estimation is the availability of large amounts of training data, especially with the advent of rich and large datasets such as COCO [26] and Human3.6M [21].
We can clearly envision the power of pose estimation by considering its application in automatically tracking human movement. From virtual sports coaches and AI-powered personal trainers to tracking movements on factory floors to ensure worker safety, pose estimation has the potential to create a new wave of automated tools designed to measure the precision of human movement. In addition to tracking human movement and activity, pose estimation opens up applications in a range of areas, such as augmented reality, animation, gaming, robotics, and many others.
We have also seen the advantages introduced by graph neural networks, and their increasing interest in many fields. The power of graph neural networks in modeling complex graph structures is concrete. In view of its effectiveness, GNNs will most probably play an important role in machine learning development in the near future. Also, as argued in [5], graph networks could be the key to achieve human-like performance in relational reasoning and combinatorial generalisation. Thus the idea of an easy-to-use tool that helps creating entire graph-based datasets using images and videos of human subjects. These tool-generated datasets contains a graph representation of the given subjects in an image or video. The subjects are represented as a graph in order to produce a more homogeneous set, and to eliminate all the issues introduced by feeding a CNN directly with a raw image, while maintaining all the relevant structural information of the original subjects.

2 State of the Art

2.1 MediaPipe [27]

MediaPipe is a framework created by Google for building multimodal (eg. video, audio, any time series data), cross platform (i.e Android, iOS, web, edge devices) applied ML pipelines. It allows to create a perception pipeline as a graph of modular components (See figure 12), including model inference, media processing algorithms and data transformations.

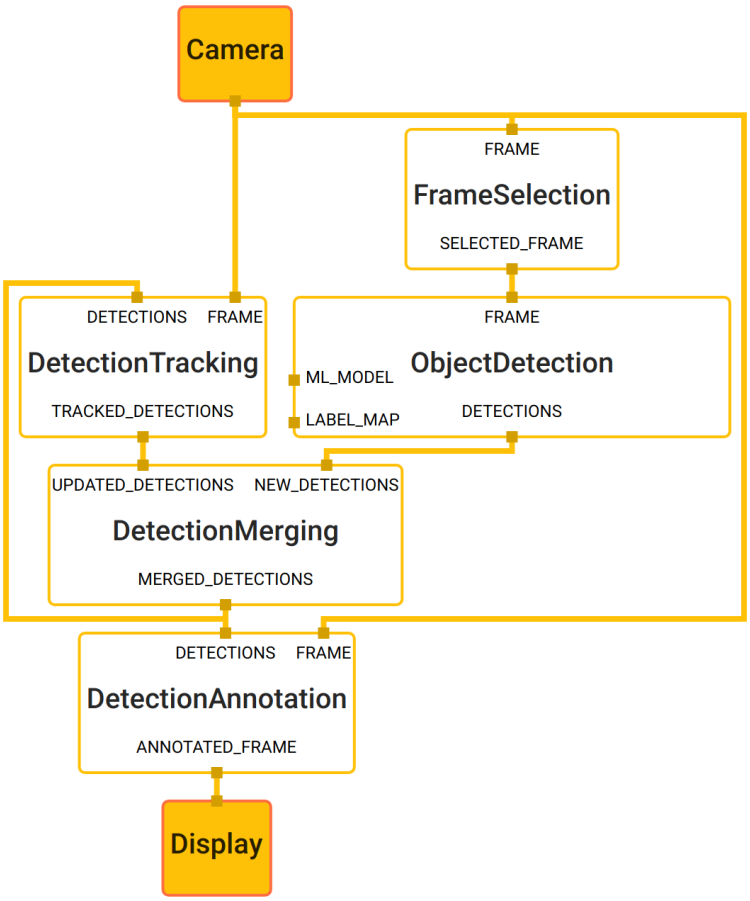
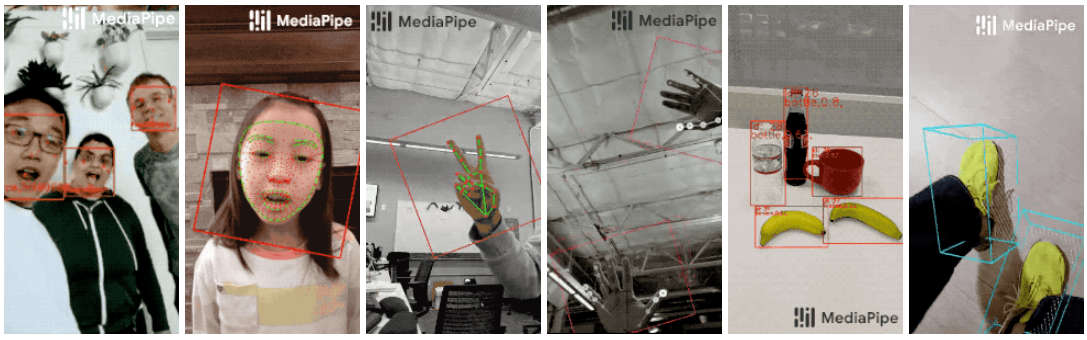


Figure 12. Example of a MediaPipe graph of modular components

The main use case for MediaPipe is rapid prototyping of perception pipelines with inference models and other reusable components. It offers a wide set of machine learning models such as face detection, multi-hand tracking and object detection and tracking (see Figure 13).



(a) (b) (c) (d) (e) (f)

Figure 13. Example of MediaPipe features: a face detection, b face mesh, c hand tracking, d multi-hand tracking, e object detection and tracking, f object tracking

MediaPipe processing takes place inside a set of software components (nodes), arranged in a graph-like structure. The nodes in the structure are connected through a stream that carries information from a node to another. Each node

in the structure represents an inference/processing component where the data processing takes place. MediaPipe was made to be simple and fast to use, as we can see in the following code example 1 and in Figure 14, in order to create a pipeline, is sufficient to declare each node with its inputs and outputs, without really caring about the code implementation of each node.

```
# MediaPipe graph that performs GPU Sobel
# edge detection on a live video stream.

# Images coming into and out of the graph.
input_stream: "input_video"
output_stream: "output_video"

# Converts RGB images into luminance images,
# still stored in RGB format.
node: {
        calculator: "LuminanceCalculator"
        input_stream: "input_video"
        output_stream: "luma_video"
}

# Applies the Sobel filter to luminance
# images stored in RGB format.
node: {
        calculator: "SobelEdgesCalculator"
        input_stream: "luma_video"
        output_stream: "output_video"
}
```

Listing 1. MediaPipe pipeline declaration

Figure 14. pipeline visualization

It also offers some tools that helps users understand the topology and overall behavior of their pipelines, called visualizers (see Figure 15). It consists of a timeline and graph view where a user can see the timing of data moving through threads and calculators.

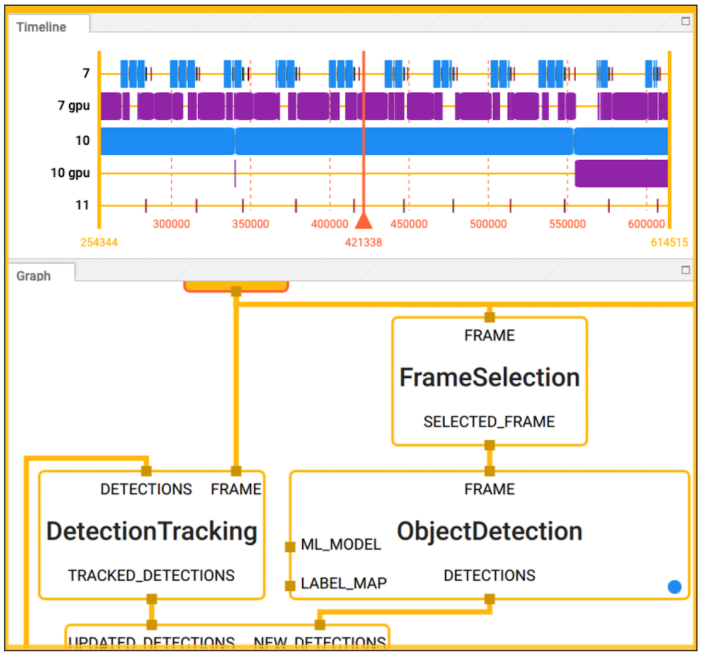


Figure 15. Example of a MediaPipe visualizer

As we can see, MediaPipe is relatively easy to understand and use, but simplicity sometimes comes with some limitations: since it is relatively new and thought to be used for prototyping, keypoints detection for hands and face are not stable, especially if there is some occlusion or the poses are uncommon. Also multiple pipelines execution can be quite challenging, for example if i want to use a hand tracking pipeline and a face mesh together. Furthermore MediaPipe does not implement body pose estimation. Thus, this solution can result inadequate for building applications where high accuracy or body estimation are required.

2.2 MMSkeleton [25]

MMSkeleton is an open source toolbox for skeleton-based human understanding. It is a part of the open-mmlab project in the charge of Multimedia Laboratory of The Chinese University of Hong Kong (CUHK). MMSkeleton provides a flexible framework for organizing codes and projects systematically, with the ability to extend to various tasks and scale up to complex deep models. The toolbox addresses to multiple tasks (see Figure 16) in human understanding such as skeleton-based action recognition and 2D pose estimation.

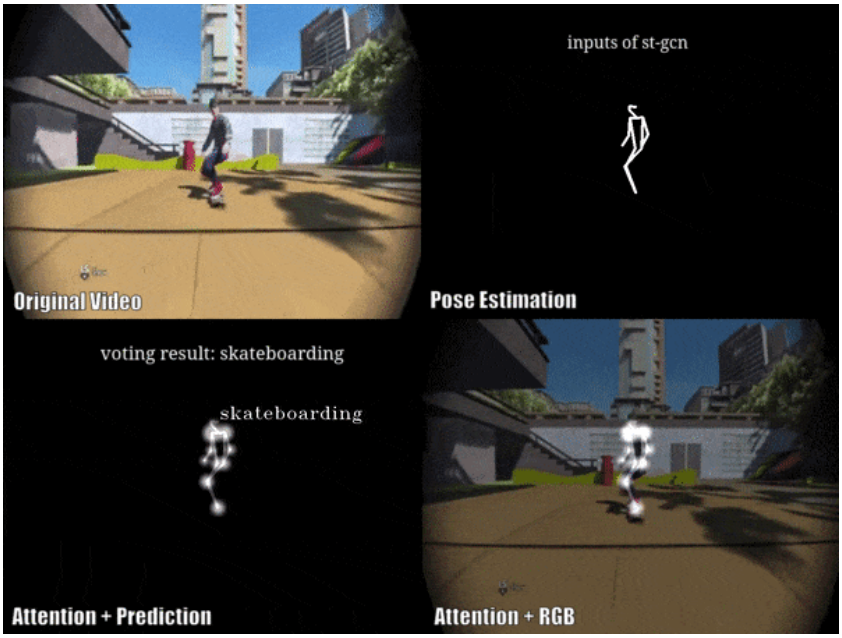


Figure 16. Example of a MMSkeleton features

In order to apply human pose estimation, MMSkeleton first uses a cascade region convolutional neural network [6] (Cascade R-CNN) to find human subjects in the given input. Cascade R-CNN is a multi-stage object detection architecture, consisting of a sequence of detectors trained to be sequentially more selective against close false positives. The detectors are trained stage by stage, leveraging the observation that the output of a detector is a good distribution for training the next higher quality detector.
Then a high-resolution net [37] (HRNet) is used as pose estimator. HRNet starts from a high resolution subnetwork as the first stage, and gradually adds high-to-low resolution subnetworks one by one to form more stages, connecting the multi-resolution subnetworks in parallel. Repeated multi-scale fusions are conducted such that each of the high-to-low resolution representations receives information from other parallel representations over and over, leading to rich high resolution representations (see Figure 17).

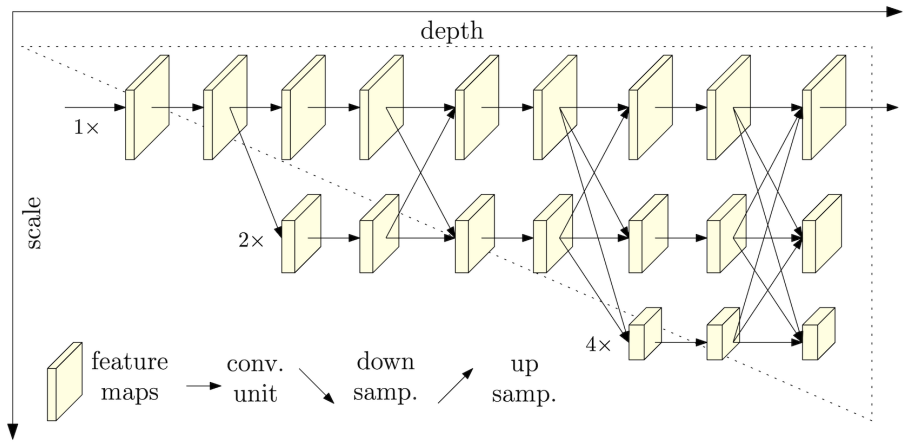


Figure 17. HRNet architecture

So the main idea is that, given a video, MMSkeleton extracts skeleton sequences via performing person detection and pose estimation on all frames. Then a .json is stored with all the data obtained (see Listing 2).

```
{
"info": {
        "video_name": "skateboarding.mp4",
        "resolution": [340, 256],
        "num_frame": 300,
        "num_keypoints": 17,
        "keypoint_channels": ["x", "y", "score"],
        "version": "1.0"
},
"annotations": [
        {
        "frame_index": 0,
        "id": 0,
        "person_id": null,
        "keypoints": [[x, y, score], [x, y, score], ...]
        }, ...
],
"category_id": 0,
}
```

Listing 2. MMSkeleton output structure

MMSkeleton has also a bult-in action recognition. It uses spatial temporal graph convolutional networks (ST-GCN) in order to recognize the given poses (see Figure 18). Usually the data is a sequence of frames, each frame will have a set of joint coordinates. Given the sequences of body joints in the form of 2D or 3D coordinates, a spatial temporal graph is constructed with the joints as graph nodes and natural connectivities in both human body structures and time as graph edges. The input to the ST-GCN is therefore the joint coordinate vectors on the graph nodes. Multiple layers of spatial-temporal graph convolution operations are then applied on the input data and a higher-level feature maps on the graph is generated. Then the images are classified by the standard SoftMax classifier to the corresponding action category.

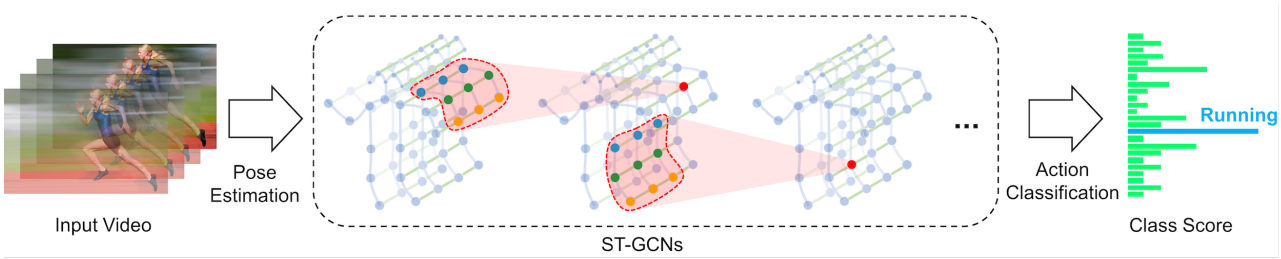


Figure 18. ST-GCN pipeline

MMSkeleton performs very well in overall, being more precise in joint estimation compared to MediaPipe, and also offers an easy-to-use built in action recognizer. But its compatible only with videos, and the output must be parsed before using it as input for CNNs. It does not offer any pose estimation for other body parts like hands or face, and supports only single-person pose estimation. Due to its complexity, it can be hard to integrate its functionalities with other systems.

2.3 OpenPose [7]

OpenPose, developed by researchers at the Carnegie Mellon University can be considered as the state of the art approach for real-time human pose estimation.
This library is able to jointly detect human body, hands and face keypoints (see Figure 19) in real time, while maintaining great accuracy and a relatively low inference time. Unlike the traditional approaches used to articulated multi-person pose estimation, where a top-down strategy is used to first detect people and then estimate the pose of each person independently on each detected region, OpenPose is based on a bottom-up approach, where the spatial dependencies across different people that require global inference are taken into account. This strategy offers robustness and have the potential to decouple runtime complexity from the number of people in the image, basically maintaining the runtime independent from the number of people detected (works only for body/foot keypoint estimation).

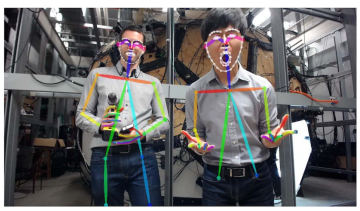


Figure 19. OpenPose features example

OpenPose main architecture is represented as a pipeline (see Figure 20). The system takes as input an RGB image of size $w \times h$ and produces the 2D locations of anatomical keypoints for each person in the image.

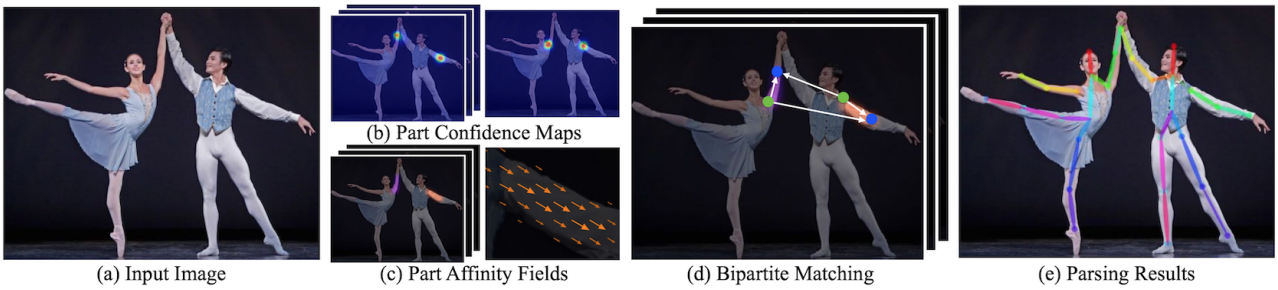


Figure 20. Overall pipeline. (a) RGB image input (b) confidence maps for body part detection and (c) PAFs for part association. (d) bipartite matchings to associate body part candidates. (e) full body poses construction.

In order to find the keypoint locations, OpenPose uses a multistage CNN (see Figure 21) that iteratively predicts affinity fields that encode part-to-part association (shown in blue) and confidence maps (shown in beige). The predictions of each stage and their corresponding image features are concatenated for each subsequent stage.

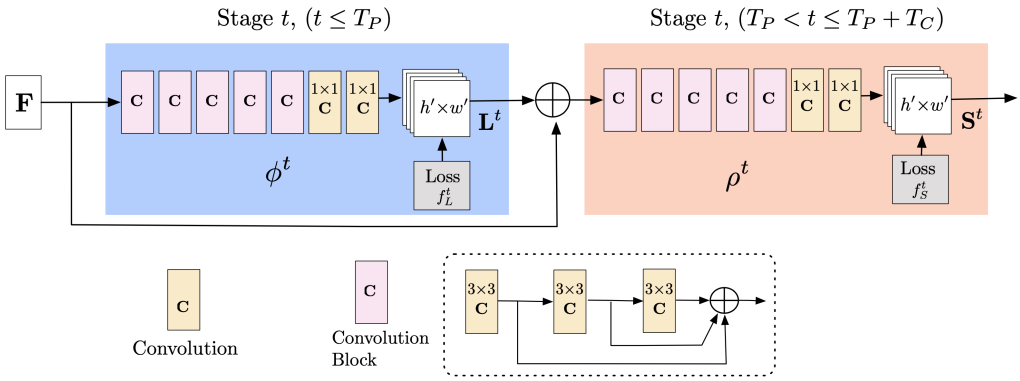
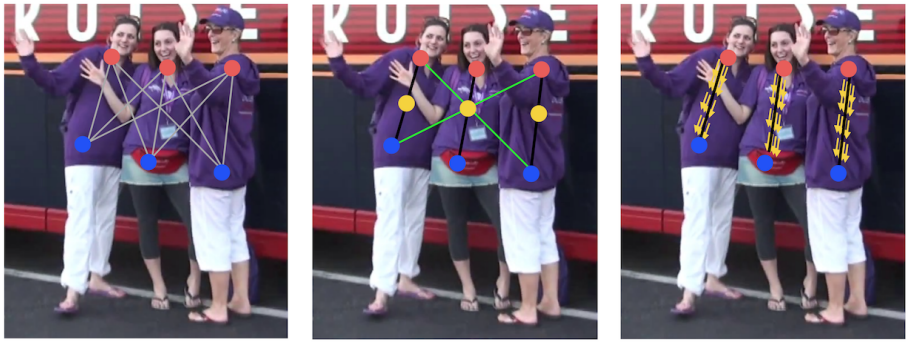


Figure 21. Architecture of the multi-stage CNN

The iterative prediction architecture, refines the predictions over successive stages, with intermediate supervision at each stage. At the first stage, the network produces an initial set of part affinity fields (PAFs), that is a set of 2D vector fields which encodes the degree of association between parts. From stage t to T_p (blue part of Figure 21), it refines the predictions of PAFs from previous stage using the feature maps F and the previous PAFs (see an example in Figure 22).

Figure 22. PAFs of right forearm across stages

After T_p iterations, the process is repeated for the confidence maps detection (beige part of Figure 21), starting in the most updated PAF prediction. Once the network generates the confidence maps and PAFs, OpenPose uses the former for part detection and the latter to part association (see Figure 22).
In order to associate the parts for each subject in the image it first consider some candidates (red and blue nodes) for two body part types and all connection candidates (grey edges) (see Figure 23a). Then it calculates the midpoint (yellow nodes) for each two parts (see Figure b). The correct connections (black edges) and incorrect connections (green edges) are generated, dividing the graph in two parts. Then the right connections are obtained by using the PAFs (yellow arrows) over the correct connections (see Figure 23c). By encoding position and orientation over the support of the limb, PAFs eliminate false associations (green edges). OpenPose can run on different platforms,



(a) (b) (c)

Figure 23. PAFs of right forearm across stages

including Ubuntu, Windows, Mac OSX, and embedded systems, and is compatible with different types of input such as image, video and webcam. It offers a command line interface and a set of flags for body parts selection and output generation. Even thought it offers a solid set of tools for using all the implemented features, in order to interface with the library via code, it is still necessary to spend some time learning and using the given APIs and also creating parsers for transforming the output in a more standard and usable structure. This can be a problem if the library is used for rapid prototyping.

3 Project requirements and analysis

3.1 Pipeline overview

The entire pipeline structure can be visualized in Figure 24. Basically, it takes an input, performs a pose estimation using the pyOpenPose API 2.3, process the keypoints and transform them in a set of graphs.

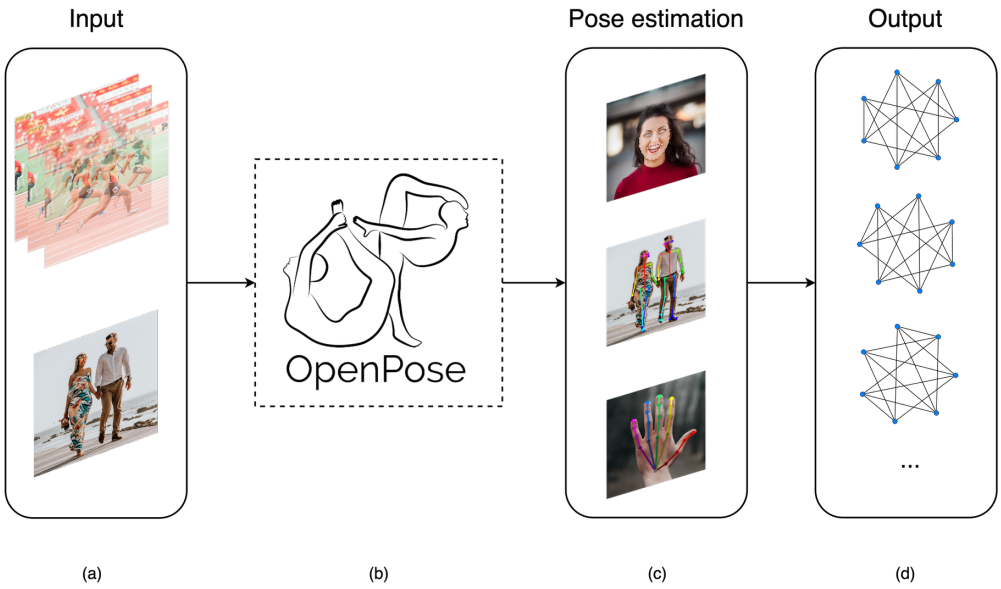


Figure 24. GraphPipe pipeline overview: (a) input. (b) OpenPose pipeline. (c) pose estimation output. (d) Graph parsing

More specifically, in 24a the pipeline can take an image or set of images, a video, or can use an input stream such as a camera. Then, depending on the type of input, a preprocessing is done, and the frames are given as input to OpenPose (see Figure 24b). OpenPose then processes all the data and returns 24c represented as a set of keypoints and the given accuracy for each point. Finally, those keypoints are parsed to graphs and 24d is saved in a JSON.

3.2 Programming language

The initial choice for this project was between Python and C++. Since OpenPose is mostly implemented in C++ and CUDA, using this language for the project would bring a great advantage in terms of library interfacing. Considering that C++ represents a more complex and low-level language, and that OpenPose released a new improved Python API, Python was the final choice, since it is probably the most comfortable language for a large range of data scientists and machine learning experts, and it offers a really simple syntax and a rich set of frameworks for neural networks creation. It offers also a huge set of packages such as NumPy that makes easy to perform pre-processing and manage arrays and matrices with high performance.

3.3 Modules and dependencies

The following is a list of all the modules and dependencies used in my project, with a brief description about the module itself and its purpose in my code:

- `os`

  This module provides a portable way of using operating system dependent functionality. It was used for path parsing and for creating directories.

- `sys`

  This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It was used for command line args parsing and for importing pyOpenPose.

- `time`

This module provides various time-related functions. It was used for tracking the execution time of part of the pipeline operations/stages.

- `math`

  This module provides access to the mathematical functions defined by the C standard. It was used for hands detection.

- `statistics`

  This module provides functions for calculating mathematical statistics of numeric data. It was used for hands data parsing.

- `cv2`

  This module is an open source computer vision and machine learning software library (OpenCV). It was used for many tasks, from output visualization to image pre-processing.

- `progress`

  This module provides an easy progress reporting for Python. It was used for operation progress tracking, together with `time`.

- `scipy`

  This module provides a Python-based ecosystem of open-source software for mathematics, science, and engineering. It was used for reading `.mat` files from datasets.

- `glob`

  This module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell. It was used for graph parsing and for gathering data information from input.

- `ujson`

  This module is an ultra fast JSON encoder and decoder written in pure C that provides a set of instruments for JSON manipulation. It was used for storing the output generated by the pipeline and for parsing the OpenPose raw data.

- `mmap`

  This module provides memory-mapped file objects that behave like both strings and file objects but, unlike normal string objects, these are mutable. It was used for seeking image resolutions from `.txt` files.

- `PIL`

  This module is a python imaging library that provides a set of tools for image manipulation. It was used for loading the images in order to get important information such as the resolution.

- `networkx`

  This module provides a set of tools for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It was used to model the OpenPose data into graphs.

- `tensorflow`

  This module is an end-to-end open source platform for machine learning. It was used to create models for testing my project.

- `spektral`

  This module provides a flexible framework for creating graph neural networks based on the Keras API and TensorFlow 2. It was used to feed the neural networks created with tensorflow with graphs.

- `sklearn`

  This module provides a set of tools for machine learning. It was used to apply preprocessing to the graphs.

- `matplotlib`

  This module provides a library for creating static, animated, and interactive visualizations. It was used plot the testing model results.

3.4 Output format

The idea for the output was to make it as complet and versatile as possible, while maintaining a format that can be easily converted and imported into projects. Given these requirements, we was led to choose the JSON format, since it uses human-readable text to store and transmit data objects and it is a very common data format. My choice was weighed also due to the fact that Python has a built-in package called `json`, which can be used to work with JSON data, allowing to parse the graphs easily. The graph/JSON structure is really simple: it has a couple of fields containing some useful information such as the *filename* and the image *resolution*, and an array of *people* containing a graph structure for each person, with its own incremental *id* and a set of *nodes* and *edges*. Both *nodes* and *edges* are divided in parts, such as *pose*, *hands* and *face*. Each *node* has an *id*, *x* and *y* coordinates, and the *confidence* (normalized from 0 to 1) of the given point, while each edge has it *ref_node* and a list of *linked_nodes*. We can see an example of a JSON graph structured data in Listing 3:

```
{
  "filename": "000065339",
  "resolution": "1280x720",
  "people": [
    {
      "id": 0,
      "nodes": {
        "pose": [
          {
            "id": 0,
            "x": 721.46,
            "y": 147.322,
            "confidence": 0.855369
          },...],
        "hands": {
          "left": [...],
          "right": [...],
        },
        "face": [...]
      },
      "edges": {
        "pose": [
          {
            "ref_node": 0,
            "linked_nodes": [
              1,
              15,
              16
            ]
          }, ...],
        "hands": {
          "left": [...],
          "right": [...],
        },
        "face": [...]
      }
    }
  ...]
}
```

Listing 3. GraphPipe output structure

4 Project design

4.1 Input

The first step of the pipeline is to deal with different input types. The three main inputs are: image, video and input stream. An input stream is considered any kind of device that is able to capture in real time a set of images, such as a webcam, Machine Vision cameras (Flir/Point Grey) or an IP camera.
In order to feed the pipeline with an image, a path to a folder containing the image/images is required. The flag used for the image input is `--image_dir [PATH]` and can be used both in the CLI and module function. Each image in the given `PATH` is then decoded using OpenCV, representing each image as structure made of numpy n-arrays with the given shape `(x,y,3)`, where `x` and `y` represents the resolution of the image, and the array of 3 elements represents each pixel, stored in `BGR` order (see Figure 25).

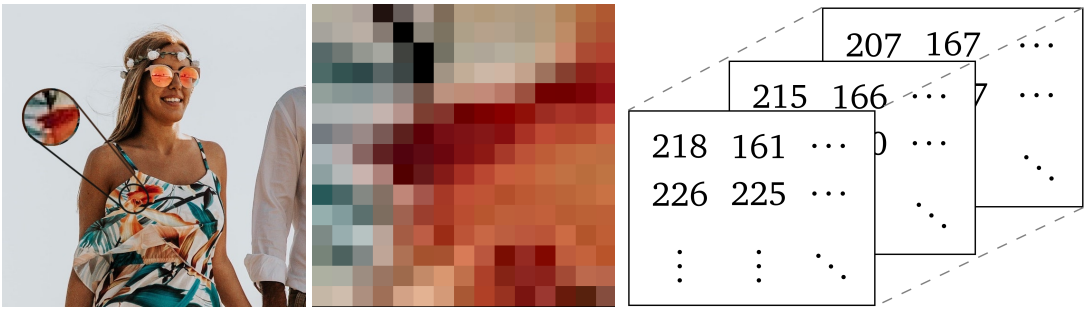


Figure 25. Example of an image decoding: a Input image, b portion of the input, c BGR matrix representation

Raw inputs such as a video or a continuous stream requires a sort of pre-processing before applying the OpenPose recognition. In order to work with a video or stream, it is first necessary to "capture" each frame by using OpenCV and then processing one frame at time (same procedure as if an image was given as input).

4.2 OpenPose processing

The OpenPose API is made of 3 main parts: a basic module named `core`, a multi-threading module `thread` and multi-person keypoint detection module `pose` (see Figure 26). All the processing of my pipeline is done by interacting directly with the `core` module, and indirectly with `thread` and `pose`. The processing part is done using a specific class from `core` module called `Datum`. This class represents the OpenPose basic piece of information and it has all the variables that the thread workers from the other two modules need to share to each other.

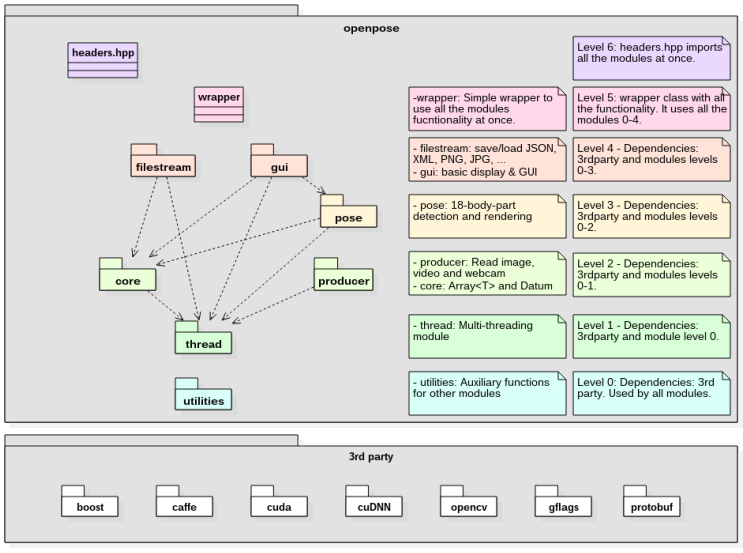
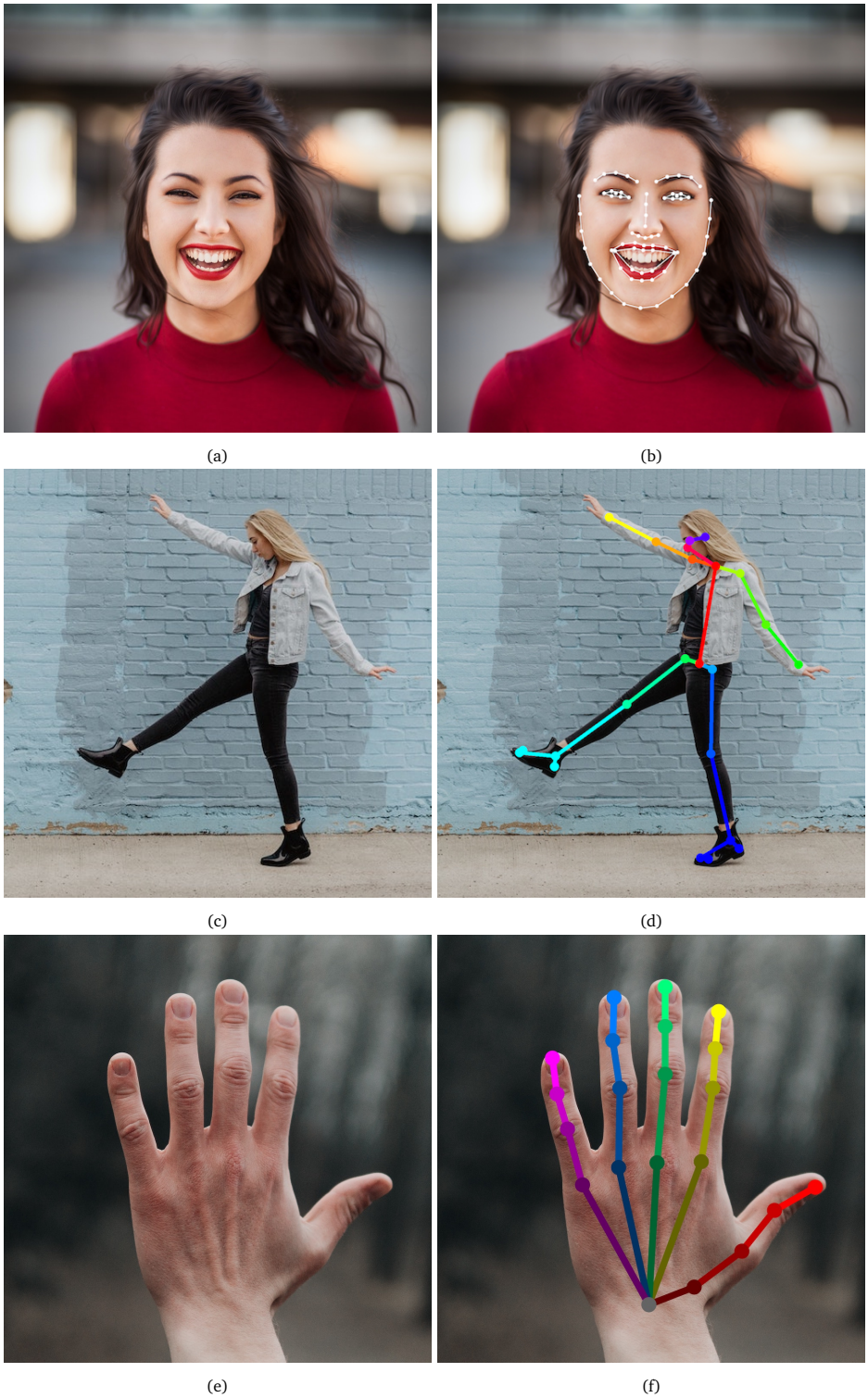


Figure 26. OpenPose API overview

After instantiating a `Datum` object, the frame is passed to the OpenPose pipeline (see subsection 2.3) in order to get the pose estimation.
The pipeline is able to process hands, body and face keypoints and each keypoint type can be specified in the *args* of both the CLI and module function. In order to detect face and hands, OpenPose uses other parts from the body keypoints such as shoulder and elbow. Therefore, if an estimation of hands or face keypoints is needed, without using the body keypoints, a detector is required, since OpenPose is not able to correctly detect the given parts. In order to solve this problem, my pipeline uses a detector implemented with OpenCV before passing the frames to the `Datum` object in order to estimate the keypoints. Due to the limitations of detection without the body keypoints, images used for only hands or only face must have only those parts in the frame and preferably centered (see Figures 27a and 27e). Figure 27 shows a sample processed images.



(a) (b)

(c) (d)

(e) (f)

(g) (h)

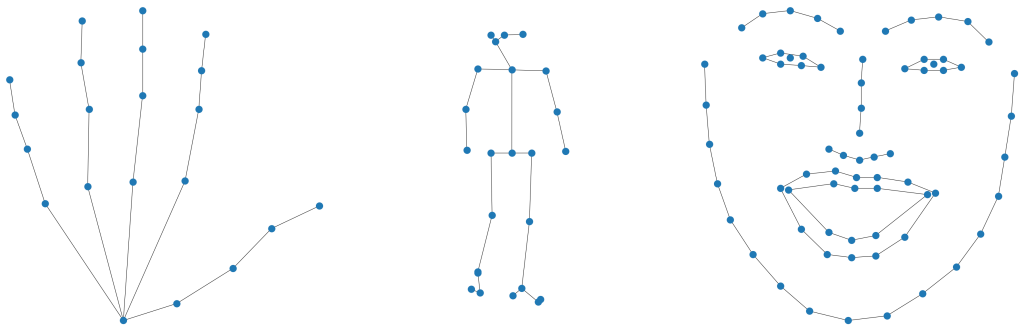
Figure 27. Examples of OpenPose outputs. b Only face keypoint estimation. d Only body keypoint estimation. f Only hand keypoint estimation. h body + hand + face keypoint estimation.

4.3 Output generation

After OpenPose processing, the `Datum` object saves all the keypoints for each part estimated. Those keypoints are contained inside arrays (one for each person) having each n, $n+1$ and $n+2$ position representing the `x`, `y` and `accuracy` respectively. Since OpenPose needs to distinguish between left and right for hands, in case of an only hand estimation, the same subject is considered to be both right and left-handed, and after the estimation, the set of keypoints with the best accuracy is considered.
After dealing with the different types of keypoints, a parsing process initiates, in order to generate the output. First of all, an empty dictionary structure (as the one shown in 3) is created. The name of the file and its resolution is added to the structure. Then, for each person in the image, an `id` is generated, and the nodes and edges are parsed/inserted into the structure. Each node also receives an `id` representing the node order, and the fields `x`, `y` and `accuracy` are filled. In order to create the edges, an utility containing all the relations between the nodes is retrieved. The main problem here is that sometimes not all the nodes are estimated (for example if some joints are not in the frame), therefore the number of nodes and edges are not always the same. Hence, before inserting the edge relations for each node, an algorithm is run in order to cut out all the invalid relations. After parsing the keypoints, the dictionary is saved as a JSON file. The request for output saving must be specified in the *args* before running the pipeline.

4.4 Graph parsing

In order to use the graphs for machine learning related tasks, since most of the graph processing frameworks uses data in a specific format, an additional step is required. For this reason, after generating and parsing the pose estimation keypoints, the data is then converted into a series of Networkx graph structures. After parsing the information, the structure results in a list containing for each frame another list of attributed graphs containing for each person a set of nodes and edges representing their body, hands and face pose (see some examples in Figure 28).



(a) (b) (c)

Figure 28. Example of Networkx pose estimated graphs. a hand. b body. c face.

5 Results

5.1 Human activity classification

The first demo used for testing the pipeline was done using the MPII Human Pose Dataset [2]. This dataset includes around 25K images taken from YouTube containing over 40K people with annotated body joints and each image is provided with an activity label.
The idea was to provide an example of classification using GNNs. Since each image in the dataset is labeled with a specific category, we decided to build a classifier in order to distinguish between people running and bicycling (see Figure 29).

(a) (b)

Figure 29. Example of images from different categories. a bicycling. b running.

The neural network was built using tensorflow [1] and keras [11]. In order to feed our keras model with graphs, we used spektral [18], since it implements some popular layers for geometric deep learning and offers also some useful utilities.

First of all, all the dataset was passed through my pipeline. Using a Microsoft Azure virtual machine *NC6* with an *Intel Xeon E5-2690 V3 2.60 GHz* and an *NVIDIA Tesla K80*, it took around 6 hours to process and save all the dataset as a set of graphs. After constructing the dataset, we implemented a script in Matlab for extracting the labels for each image, in oder to obtain only the ones with the label *running* and *bicycling*.
In Spektral, graphs are represented as matrices: the matrix A represents the adjacency matrix of shape (N, N), where N is the number of nodes, and matrix X is the node attributes matrix of shape (N, F), where F is the size of the node attributes. In order to convert our Networkx data into matrices A and X, we used an utility given by spektral. In order to maximize the results of our learning process, we applied some preprocessing to the matrices. First, we considered only the graphs that had at least 50% of their nodes with a confidence greater or equal than 0.7. Then an one hot was added to the data, in order to convert our categorical data into a numerical form. Finally we standardized our data by subtracting the mean and dividing by the standard deviation, in order to have a mean of zero and a standard deviation of 1. After the preprocessing we created our convolutional model (see Figure 30). After training and evaluating our model, we obtained an acceptable result, considering the fact that the MPII Human Pose Dataset is really hard to deal with since it is composed by YouTube video frames, where some poses are barely recognizable even by humans. Our model achieved a test accuracy of 82.6%, as we can see in Figure 31.

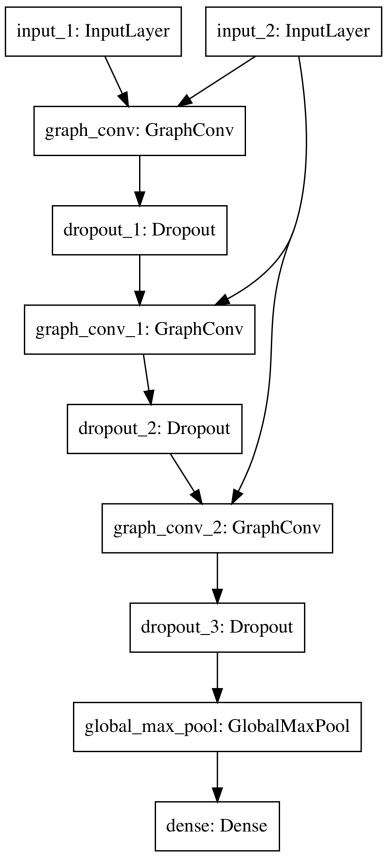


Figure 30. GNN model

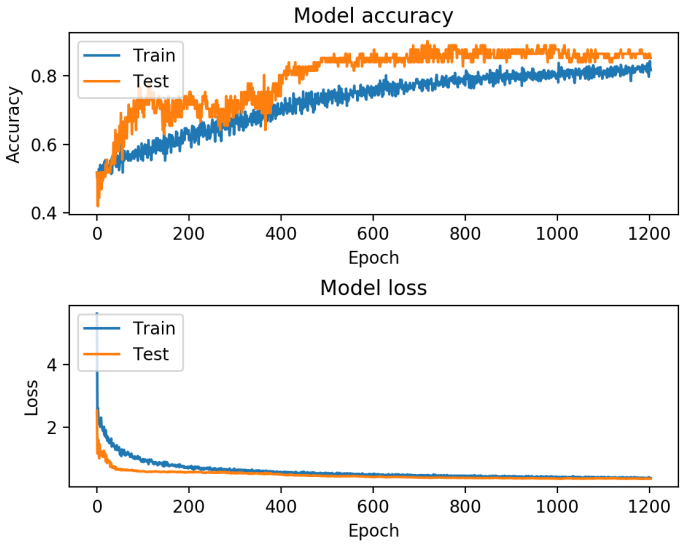


Figure 31. Human activity classification results

5.2 Hand gesture classification

For the second demo, we used a dataset collected in a machine learning class experiment, in order to obtain some hand images for rock, paper and scissors (RPS) classification (see Figure 32).

(a) (b) (c)

Figure 32. Example of images from rock paper scissors.

The process here was pretty much the same: we converted our Networkx graph structures to the matrices A and X using the Spektral utilities, then we applied the preprocessing to our matrices, and finally we defined the same model as in the first demo (see Figure 30).

In this case the advantage of using graphs instead of raw images are more visible: since the given dataset was created by collecting photos using a telegram bot, the size of the dataset was quite contained, compared with the thousand of images given by the MPII Human Pose Dataset, Nevertheless, our model was able to learn effectively from the data (see Figure 33), obtaining an accuracy of 84.3% and a loss of 0.3749 during the evaluation part. Those results where made possible by the fact that, using graph structures, we are actually simplifying images, removing all the major problems introduced in subsection 1.2, allowing the model learn in a more precise way.

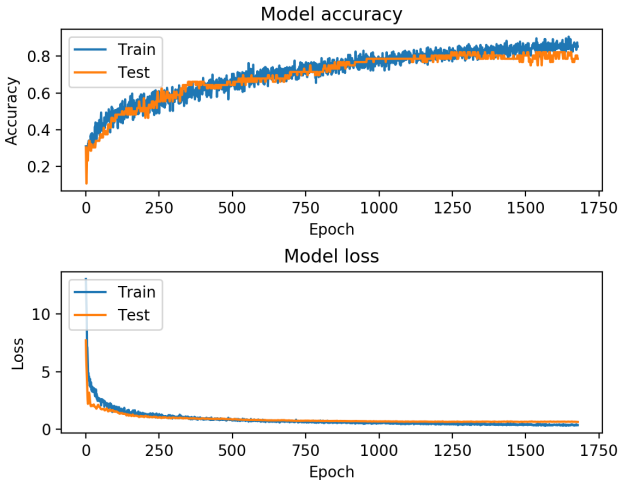


Figure 33. Hand gesture classification results

6 Conclusions

We have implemented a pipeline for extracting graph structures from images and videos. This pipeline is able to simplify the raw image, while preserving relevant information in order to be used in graph-processing systems. It uses one of the most advanced libraries for real time 2d pose estimation, and offers several utilities in terms of fast prototyping and testing. We have demonstrated that most of the state of art solutions suffers from precision or are limited in terms of estimation and flexibility. The pipeline produces good results on a wide variety of inputs, giving a simple but effective way for creating datasets for graph neural networks.

Step 1 Step 2 Step 3

Figure 34. Image processing steps

7 Future work

There are many possibilities for future work. The implementation still suffers from some limitations which are being corrected or improved, but also more features are being added to increase the number of different applications for this pipeline.

7.1 Improve hand and face detection

In section 4.2 we talked about the fact that OpenPose requires body keypoints in order to detect correctly both face and hands. Even though we were able to work around the problem by implementing our own detector, it is still inferior in terms of detection accuracy, compared with OpenPose method. A good way for improving it is to use some good hand and face detector implementations, that are able to track the subject in real time, allowing to do only hands or only face detection in input streams, and giving also the possibility to process only hands or face in images with more than one subject or with different body parts included. An example of real time hand and face detection are [42] and [28].

7.2 Improve real-time stream FPS

As stated already in section 4.2, the pipeline can work with a real time input stream, by applying pose estimation and graph conversion to each frame. Since the estimation and conversion part can take some time, the FPS rate is usually really low, depending on the hardware used. Video compression and frame decoding can also influence the whole processing time of the pipeline. An idea for improving this whole process could be to utilize threading and a queue data structure in order to obtain a reduction in latency. The read method of OpenCV is a blocking operation, which means that the thread operating on the function call remains entirely blocked until the frame is read and decoded from the output stream. Therefore, by creating a new thread that deals with frame reading and decoding and stores those frames in a queue, the main thread is able to handle the OpenPose processing and graph conversion without waiting for any input. This idea is explained in [33].

7.3 Improve conversion utilities

In Section 4.4 we explained the pipeline conversion stage, where the keypoints estimated are parsed to a list of Networkx graphs. An improvement could be to give the option to convert the keypoints to multiple formats, such as python-igraph [12] format, graph-tool [30] or even to an adjacency matrix [45].

7.4 Extend the keypoint detection idea to objects

The purpose of this project is to create datasets based on human pose estimations and organized as graphs, but the same idea can be extended to objects such as vehicles [14] or even to animals [13].

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[3] M. Andriluka, S. Roth, and B. Schiele. Pictorial structures revisited: People detection and articulated pose estimation. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1014–1021, 2009.

[4] S. C. Babu. A 2019 guide to human pose estimation with deep learning, May 2019. [Online; accessed 17-May-2020].

[5] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

[6] Z. Cai and N. Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018.

[7] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields, 2018.

[8] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[9] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.

[10] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik. Human pose estimation with iterative error feedback, 2015.

[11] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[12] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[13] P. T. D, A. D. E, W. Lindsay, K. Mikhail, W. S. S-H, M. Mala, and S. J. W. Fast animal pose estimation using deep neural networks. *nature*, 2019.

[14] W. Ding, S. Li, G. Zhang, X. Lei, and H. Qian. Vehicle pose and shape estimation through multiple monocular vision, 2018.

[15] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc., 2015.

[16] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

[17] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005.

[18] D. Grattarola. Spektral. `https://github.com/danielegrattarola/spektral`, 2019.

[19] A. Grover and J. Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.

[20] D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum (Proceedings of Eurographics/IEEE-VGTC Symposium on Visualization)*, 28(3), 2009.

[21] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(7):1325–1339, July 2014.

[22] D. D. Johnson. Learning graphical state transitions. In *Proceedings of the International Conference on Learning Representations*. ICLR, 2017.

[23] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6348–6358. Curran Associates, Inc., 2017.

[24] Y. Li, R. Yu, C. Shahabi, and Y. Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting, 2017.

[25] S. Y. Y. X. J. W. D. Lin. Mmskeleton. `https://github.com/open-mmlab/mmskeleton`, 2019.

[26] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2014.

[27] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann. Mediapipe: A framework for building perception pipelines, 2019.

[28] Mjrovai. Opencv face recognition. `https://github.com/Mjrovai/OpenCV-Face-Recognition`, 2020.

[29] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation, 2016.

[30] T. P. Peixoto. The graph-tool python library. *figshare*, 2014.

[31] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014.

[32] B. Raj. An overview of human pose estimation with deep learning, April 2019. [Online; accessed 17-May-2020].

[33] A. Rosebrock. Increasing webcam fps with python and opencv, December 2015. [Online; accessed 10-June-2020].

[34] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control, 2018.

[35] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a sat solver from single-bit supervision, 2018.

[36] T. Simon, H. Joo, I. Matthews, and Y. Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*, 2017.

[37] K. Sun, B. Xiao, D. Liu, and J. Wang. Deep high-resolution representation learning for human pose estimation. *arXiv preprint arXiv:1902.09212*, 2019.

[38] P. Tanugraha. Understanding openpose (with code reference)— part 1, September 2019. [Online; accessed 3-June-2020].

[39] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks, 2014.

[40] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2014.

[41] D. Valput. Machine learning with graphs: the next big thing?, March 2019. [Online; accessed 17-May-2020].

[42] D. Victor. Handtrack: A library for prototyping real-time hand trackinginterfaces using convolutional neural networks. *GitHub repository*, 2017.

[43] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[44] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *CVPR*, 2016.

[45] Wikipedia. Adjacency matrix, 2020. [Online; accessed 10-June-2020].

[46] Wikipedia. Json, 2020. [Online; accessed 2-June-2020].

[47] Y. Yang and D. Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *CVPR 2011*, pages 1385–1392, 2011.

[48] Y. Yang and D. Ramanan. Articulated human detection with flexible mixtures of parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2878–2890, 2013.