Recognizing Human Actions from 3D Skeleton Sequences

Flavien Vidal
École Polytechnique
flavien.vidal@polytechnique.edu

Abstract

The recognition of human actions is a topic in the field of computer vision, which includes various subtopics such as pedestrian detection, segmentation or pose estimation. To recognise an action, a human being would first locate the person, analyse the position of the body parts, understand how they move and interact together, and finally classify the action accordingly. In this work, we propose to replicate this process using only the location information of body parts. In particular, to locate these body parts, we use a skeleton representation due to its action-centric nature and compactness. We leave aside the study of pose estimation, which will be the subject of our next work. We therefore develop an action recognition module whose input are the sequences of coordinates from the skeletal representation, while the output is the prediction of the performed action. Considering a simple but effective strategy, we study the influence of the input data configuration and of the internal memory of the network on the quality of the predictions. Our results demonstrate the interest of incorporating hidden joints and truncated sequences into the training data and raise the question of the relevance of training models on sequences of more than a hundred frames. In particular, they show that only a small subset of the last frames is needed in order to not only save computational power and time but also to increase the precision of predictions.

Index Terms *— Human Action Recognition, Skeleton Sequences, Deep Learning, Computer Vision, LSTM.*

1. Introduction

The purpose of this project is to develop tools and methods, accompanied by meaningful metrics, to perform human action recognition (HAR).

Over the last decade, new deep learning approaches have led to significant improvements in many areas such as computer vision, speech and language processing, and even game solving. In computer vision, in particular, progresses have been achieved in tasks such as semantic segmentation, object recognition and object detection. An explanation for such advances, besides the increasing amount of data available and improved GPUs, has been the introduction of Convolutional Neural Networks, or CNNs, allowing to capture the spatial locality assumption of the data structures and images features [1]. Consequently, the HAR task has moved from the study of manually-crafted video representations to the one of neural networks. It has become an attractive research area among the wide range of computer vision applications, and is a central task in video understanding which aims to understand human actions from video or image sequences, and to assign a label to each of them. This task has many real-world applications [7], such as the surveillance of public places, video captioning, sport analysis. However, the more complex the actions or applications are, the more robust and consistent the methods developed or techniques deployed need to be.

The challenges of the human action recognition task are related to the difficulty of defining the movement of the different body parts, but are also due to many real-world artifacts such as camera movements, dynamic backgrounds or recording noises. Although we could prevent some of them using high end devices or more computational power, not all of them are related to material deficiency. Different approaches have been introduced in the literature, but they are insufficient to fully cover the challenges of the topic. As an example, some of them would consider multiple points of view [6] but are unable to make up for occlusions. These studies have explored various modalities of feature representation. Indeed, the action recognition task can be studied using RGB images, infra-red data, optical streams or even human skeleton representation. These methods have their own advantages depending on the specific applications and, as a result, many research studies have attempted to investigate different approaches.

Action recognition based on skeleton representations has received particular attention due to its action-focused nature and its compactness. Indeed, such structured

data allow to avoid having to consider data which are much larger in size and require more processing time and optimised processes. When performing live prediction, this computational and, *a fortiori*, temporal problem must be addressed. Skeleton representations are arrays of human joint coordinates extracted from an image using pose estimation methods, allowing to capture essential information related to the action performed, while filtering noises, background variations or even changes in clothing.

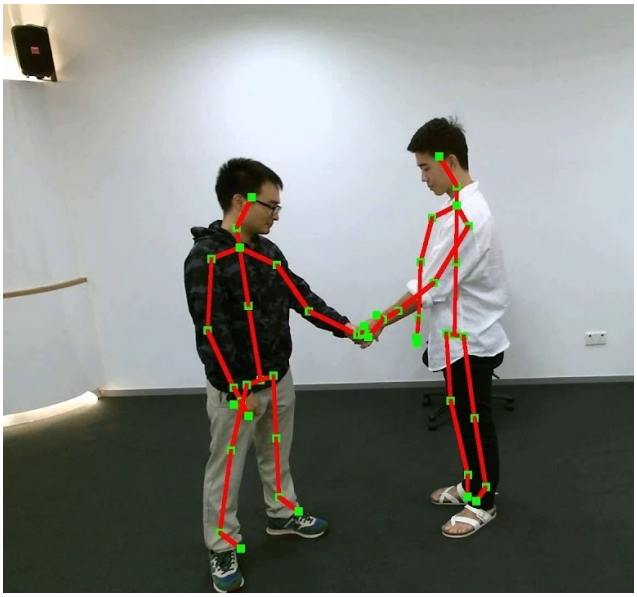


Figure 1. Highlight of the skeleton joints on an RGB image [9].

As a consequence, we decided to work on skeleton data based recognition methods, rather than directly using video data which require both more pre-processing and heavier computation. As we will see later in presenting related work, these methods have proven their worth, particularly in terms of generalizability and resistance to photometric or scenic variability. To compare the performance of these methods, public data sets with various actions under several conditions and constraints were recorded.

In the first section, we start by briefly presenting an overview of recent advances in deep learning-based human action recognition methods, focusing on the one using skeleton data. In the second section, we provide an analysis of the dataset used to build our models. In particular, we discuss the statistics of the different possible actions such as the space occupation or the number of frames needed to perform the action. In the third section, we present the general architecture of our action recognition module. Finally, we perform experiments and analyze the influence of data quality and consecutive action on the predictions to demonstrate the efficiency and flexibility of the proposed method.

2. Related Work

2.1. Graph Neural Networks

A graph is a structure composed of vertices and connected by edges. We can distinguish two types of graph networks: directed graphs, which associate a direction to the edges, and undirected graphs, which do not. Nowadays, a huge amount of information is represented through graphs such as the Google's Knowledge Graph, social media, and even simply chemical molecular structures.

Graph network structures are an emerging topic in deep learning [12], and due to the graph structure of the skeleton representations, research have been led in order to perform the action recognition task using such networks. This is the case of the Directed Graph Neural Network or DGNN [10], developed, trained and tested on the NTU RGB+D dataset [9]. In this model, the joints and bones are represented as the vertices and edges respectively within a directed acyclic[1] graph, which is fed into the DGNN to extract features for action recognition. It is interesting to note that this model was able to achieve an accuracy of almost 95%, which is more than encouraging and invites to look at other possible uses of GNNs. Nevertheless, these neural architectures will not be explored in this report.

2.2. PoseC3D

PoseC3D present a different method for human action recognition which relies on a 3D heatmap stack [3], robust to small perturbations of the skeleton pose, and which allows to use optical flow and to take into account multiple individuals in a single frame. To develop their model, the authors of the above-mentioned article used two-dimensional poses rather than three-dimensional ones, since they established that this allowed them to obtain better results. Such an approach then raises questions about the relevance of spatial information extraction for two-dimensional data. One possible explanation for the use of two-dimensional poses is their ability to carry most of the information while being simple to annotate, especially for scenes involving multiple people.

For research purposes, we decided to test the PoseC3D model using the pre-trained parameters available online. To do so, we used a modified version of the NTU RGB+D 60 dataset to fit the two-dimensional input of PoseC3D. In this framework, we tested 500 samples with an average duration of 22 seconds per sample. We obtained a top 1 precision of about 93%, a top 5 precision of 99%, and an approximate average precision of 93% per class. However, we do not consider this method in our further research because it uses thermal maps, which make intensive use of the computing power of GPUs.

[1] A directed acyclic graph is a directed graph without any loops.

3. NTU RGB+D Action Recognition Dataset

As mentioned earlier, we decided to use the human skeleton data instead of directly using the RGB videos which require too much pre-processing and heavy computation.

3.1. Structure of a skeleton

Here are the 25 joints of the skeletal representation:

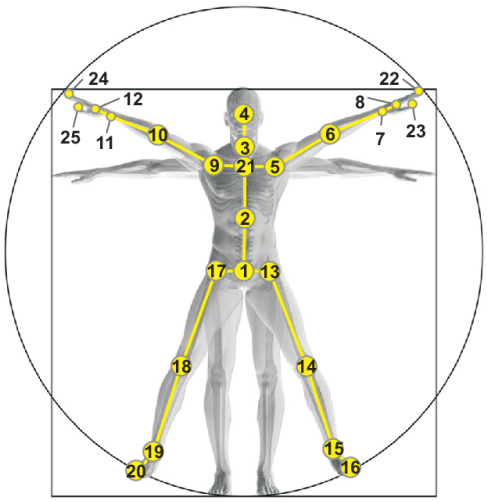


Figure 2. Configuration of 25 body joints in the dataset.

3.2. Presentation of the dataset

The dataset used in our project and presented here is the NTU RGB+D Action Recognition Dataset [9], which counts 56,880 video samples collected from 40 separate subjects. Three Microsoft Kinect V2 cameras (with a 30 frame-rate) were used so as to simultaneously capture different viewpoints from the same action. The cameras were located at the same height with however different horizontal angles: $-45°$, $0°$, $+45°$. The height and the distance of the three cameras change depending on the setup as shows the *Table* 1.

setup	h (m)	d (m)	**setup**	h (m)	d (m)
1	1.7	3.	**10**	1.8	3.0
2	1.7	2.5	**11**	1.9	3.0
3	1.4	2.5	**12**	2.0	3.0
4	1.2	3.0	**13**	2.1	3.0
5	1.2	3.0	**14**	2.2	3.0
6	0.8	3.5	**15**	2.3	3.5
7	0.5	4.5	**16**	2.7	3.5
8	1.4	3.5	**17**	2.5	3.0
9	0.8	2			

Table 1. Height and distance of the cameras in each setup.

Each captured sample finally include four different modalities of data: RGB videos (1920×1080), depth map sequence and infrared video (512×424), as well as the 3D skeleton data that contains the 3D locations of the 25 major body joints at each frame. Each file in the dataset has a *SsssCcccPpppRrrrAaaa* title:

- **sss** is the setup number (between 001 and 017).
- **ccc** is the camera ID (between 001 and 003).
- **ppp** is the performer ID (between 001 and 040).
- **rrr** is the replication number (001 or 002).
- **aaa** is the action class label (between 001 and 060.

3.3. Action classes

The dataset contains 60 action classes listed below:

A1. drink water	A31. pointing to sth with finger
A2. eat meal/snack	A32. taking a selfie
A3. brushing teeth	A33. check time (from watch)
A4. brushing hair	A34. rub two hands together
A5. drop	A35. nod head/bow
A6. pickup	A36. shake head
A7. throw	A37. wipe face
A8. sitting down	A38. salute
A9. standing up from sitting	A39. put the palms together
A10. clapping	A40. cross hands in front (say stop)
A11. reading	A41. sneeze/cough
A12. writing	A42. staggering
A13. tear up paper	A43. falling
A14. wear jacket	A44. touch head (headache)
A15. take off jacket	A45. touch chest (heart pain)
A16. wear a shoe	A46. touch back (backache)
A17. take off a shoe	A47. touch neck (neckache)
A18. wear on glasses	A48. nausea or vomiting condition
A19. take off glasses	A49. use a fan (hand/paper)
A20. put on a hat/cap	A50. punching/slapping others
A21. take off a hat/cap	A51. kicking others
A22. cheer up	A52. pushing others
A23. hand waving	A53. pat on back of others
A24. kicking sth	A54. point finger at the others
A25. reach into pocket	A55. hugging others
A26. hopping (1 foot jump)	A56. giving sth to others
A27. jump up	A57. touch others pocket
A28. phone call/answer	A58. handshaking
A29. playing with phone	A59. walking towards each other
A30. typing on a keyboard	A60. walking apart from each other

Figure 3. The actions available in the dataset.

3.4. Analysis and visualization of the dataset

In this first work, to ease the study, if a sequence of skeletons contains more than one skeleton per frame, we only consider the first one. Moreover, we only consider nine of the above actions:

- Class 0 : Pickup (A6)
- Class 1 : Throw (A7)
- Class 2 : Sitting-down (A8)
- Class 3 : Standing-up (from sitting position) (A9)
- Class 4 : Take-off jacket (A15)
- Class 5 : Reach into pocket (A25)
- Class 6 : Pointing to something with finger (A31)
- Class 7 : Check time (from watch) (A33)
- Class 8 : Falling (A43)

3.4.1 Number of frames

The *Table* 3 summarizes the distribution of the number of frames according to the considered action class.

class	class size	mean	std	min	median	max
0	943	81	14	54	81	131
1	944	64	13	37	62	136
2	941	74	14	46	74	119
3	936	64	12	39	63	120
4	945	141	34	66	138	277
5	946	97	23	54	94	202
6	944	56	12	32	54	110
7	946	64	14	36	62	128
8	946	64	12	40	62	139

Table 2. Number of samples followed by statistics on frames per class.

From these statistics, we observe that each of our 9 action classes has an equivalent number of samples (with a maximum difference of 1% between the smallest and the largest class) thus forming a balanced dataset. Furthermore, for a given action class, the mean and median number of frames are approximately equal. We also observe that the average number of frames is very different from one action class to another (it can double in some cases) which is natural since some actions require more or less movements, and therefore do not have the same duration.

3.4.2 Occupation of the space

In this part, we analyze for each action class, how the space is used (for the three-dimensional dataset) as well as where on the screen the action is most likely to be located (for the two-dimensional dataset).

From *Figure* 4, we observe that the actions most often take place in the center of the frames. This is most likely due to the fact that the experiments were centered on the camera opening. We can therefore assume that a model trained with these data would be unable to correctly predict the action of off-center humans. One solution to this problem is either to manually center the frames or to replace the spatial reference with a directly skeletal-related reference, for example by adopting the value 0 at the pelvis.

From the histograms presented in *Figure* 5, we observe that the distribution of joint positions in 3D space is also very similar between the different classes of actions. Once again, the actions seem most often centered around 0 with respect to the X and Y coordinates. They are performed around a certain depth band close to 3.5m. Similar observations to those above can therefore be made on these results.

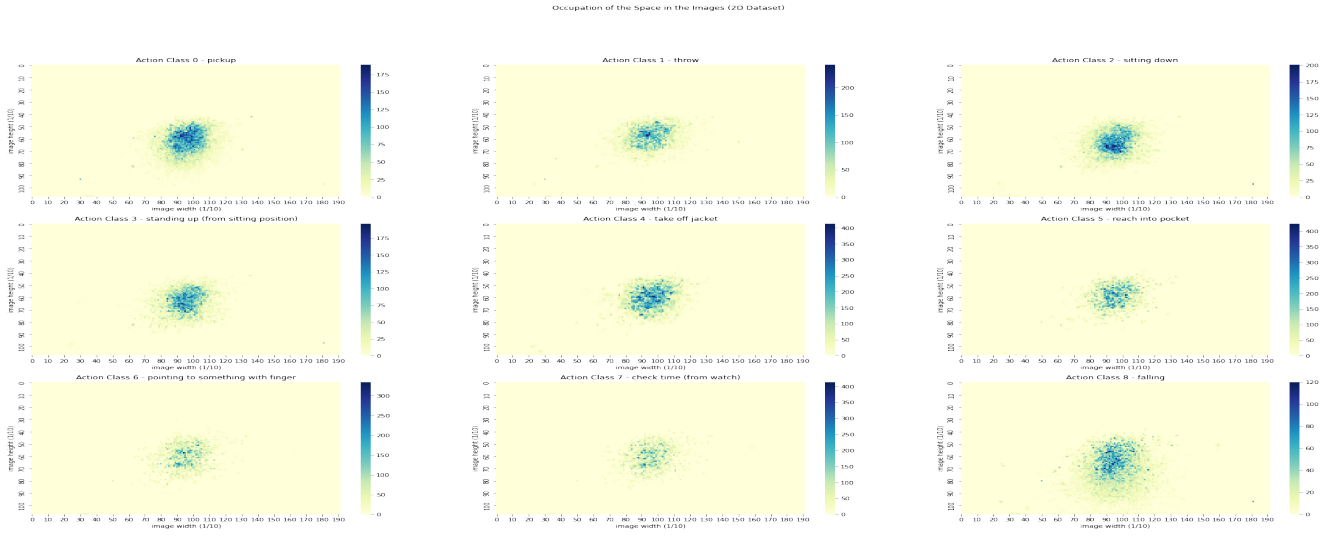


Figure 4. Heatmap of the action location per class (2D data).

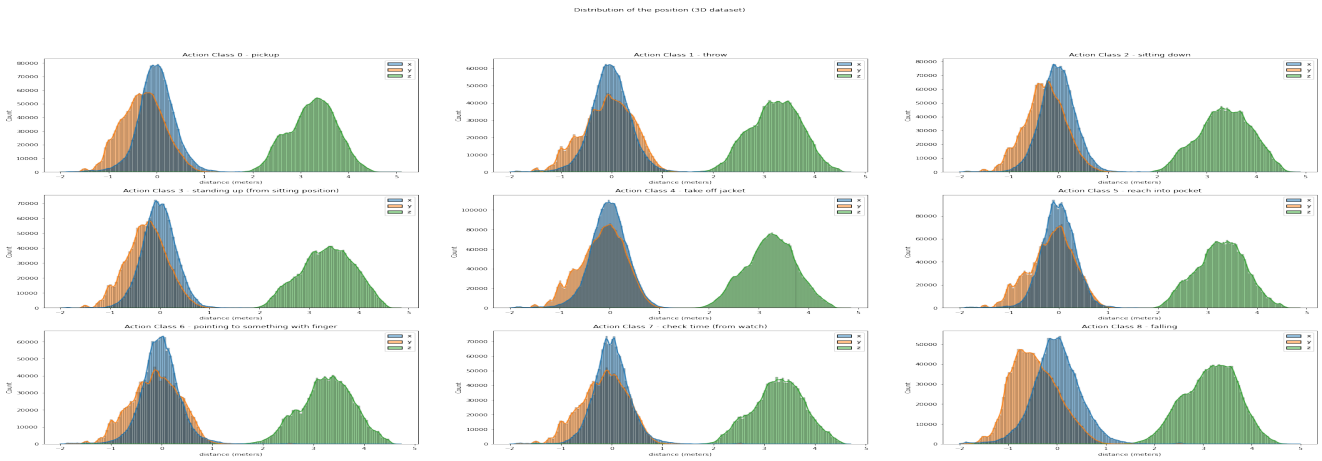


Figure 5. Histogram of the action location per class (3D data).

3.4.3 Visualization

As an illustration, we present four of our nine classes by animating the 3D skeleton data. The color palette from blue to red indicates the depth of the body members. The more blue a joint is, the closer it is to the camera and conversely the more red it is, the further away it is. *Figures* 6 to 9 respectively depict a human throwing an object, sitting down, standing up and then checking the time on his watch.

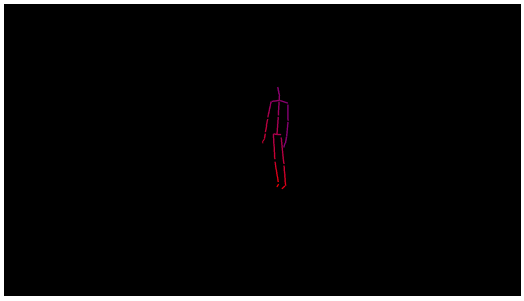


Figure 6. Animation[2] of a skeleton for the class 1 (throw).

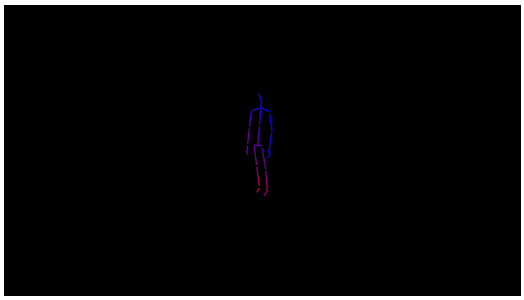


Figure 7. Animation of a skeleton for class 2 (sit down).

[2] Click on the image to watch the corresponding GIF.

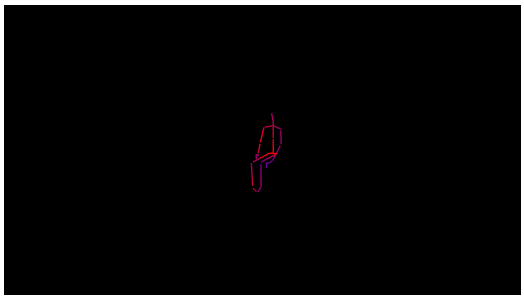


Figure 8. Animation of a skeleton for class 3 (stand up).

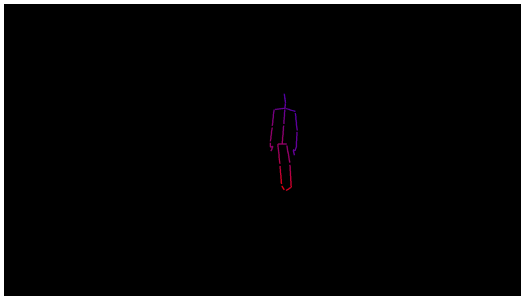


Figure 9. Animation of a skeleton for class 7 (check time).

4. Recurrent Neural Networks

4.1. Dealing with sequential data

Sequential data can be divided into two main categories: time series and ordered sequences. Since they inherently are the result of a temporal processes, sequences of skeleton representation are part of the former category. Given their versatility, Recurrent Neural Networks [11], or RNNs, are popular models used in a great variety of problems related to sequences.

Thus, we decided to use RNNs as a basis of our own whose architecture is described in the next section. Using standard neural network on sequence data such as skeleton data is most often not a good idea. Indeed, they are unable to take into account inputs of variable lengths, and thus to to take advantage of the sequential structure of the input data of a skeleton.

4.2. Description of RNNs

Since the input structure is a sequence of skeleton data, the inputs must be ordered. In order to transmit information through time, the architecture of a standard neural network is not appropriate because all the elements of the sequence input would be connected to all the neurons of the hidden layer. Unlike the standard neural network, in the RNN each input is sent to one of the hidden layer neurons which also takes as input the output of the previous hidden state.

Bidirectional RNNs are a variant of standard RNNs used to process sequential data with possibly interdependent time steps [8]. Their main characteristic is to rely on the use of two sub networks: one of them performing operations and sequence processing from left to right (or past to the future) and the other one from right to left (or future to the past). In other words, a prediction can result from both the information before and after the corresponding position in the input sequence. Such networks proved to be particularly useful in our problem, as the prediction at a given time may depend both on the data preceding it, but also on data following it.

4.3. Memory issues in RNNs

Most challenges in RNNs are related to memory issues. More concretely, predicting an outputs from a sequence of data can sometimes be difficult, given the nature of the task and the output. For example, it is possible that one may need the data from the first element of the sequence to be perfectly relevant. Nevertheless, depending o,n the architecture of the RNN, it may be difficult to achieve decent results if the input sequence is too long. The last phenomenon can be seen as a lack of long term memory in RNNs.

4.4. Variants of the RNN

So as to overcome the problem of the lack of memory of classical RNNs and therefore to take into account the long-term dependencies of our skeleton data, we will now briefly study two types of units that can replace the hidden units in RNNs. Both are based on the following equation:

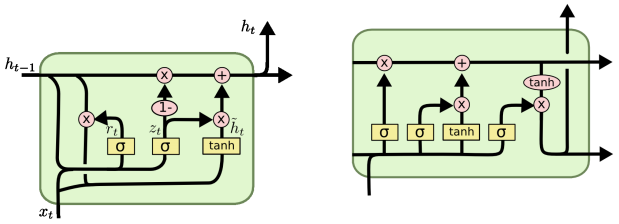
$$h_t = f_t \odot h_{t-1} + i_t \odot \tilde{h}_t$$

where f_t (forgetting gate) and i_t (entering gate) are sigmoid functions used on a linear combination of h_{t-1} (the internal memory), x_t (the input at time step t), and a bias, and where $\tilde{h}_t$ represents a possible candidate for h_t. The idea of this process is to allow the neural network to remember the components of the previous hidden states, which can help improve the long-term dependencies within the network. Indeed, the forgetting gate f_t is used for memorization and the candidate hidden state $\tilde{h}_t$ incorporates an output gate that allows to give more weight to some components of the previous hidden state.

4.4.1 Gated Recurrent Units (GRU)

A solution to maintain the stability of the model is to use Gate Recurrent Units, or GRUs [2], which specify relation between the forgetting and input gates, such that $f_t = 1 - i_t$. We then obtain the following equation:

$$h_t = (1 - i_t) \odot h_{t-1} + i_t \odot \tilde{h}_t$$



(a) GRU Architecture. (b) LSTM Architecture.

4.4.2 Long Short Term Memory Units (LSTM)

An other solution is to use Long Short Term Memory units, or LSTMs [4]. The idea is to apply an hyperbolic tangent to the hidden state to bind its values. Such model are useful in this project since they help remembering the long-term dependencies of the skeleton sequences.
Now that we have outlined the theoretical aspects and details of the models we used, and tried to explain why some specific models were interesting in our case, and what were the dangers of using them as they were, we will dive into the concrete approach we adopted for all our experiments.

5. Approach

5.1. Prior considerations

At the beginning of our research, we had to choose between a 2D coordinate system (pixel coordinates) and a 3D coordinate system (from the Kinect camera). We favored the latter on the assumption that tridimensional data were able to provide more relevant information about the positions of the joints. Moreover, we assumed that we could establish a way to generalize our architecture to two-dimensional data more efficiently. This assumption became even more meaningful when we measured the correlation

coefficients[3] between the 2D and 3D values of (x, y), which were approximately equal to 97%. As previously explained, we decided to limit ourselves to 9 classes of actions[4] when presenting the dataset, notably in order to reduce the computation time.

5.2. Architecture of the model

The architecture of our model is relatively simple and remains the same throughout the different analyses and experiments performed in this work. The results obtained from this architecture seem sufficiently good to be satisfied. Nevertheless, more studies on other possible structures will be done later in a future work. The model we implemented is illustrated on *Figure* 14.

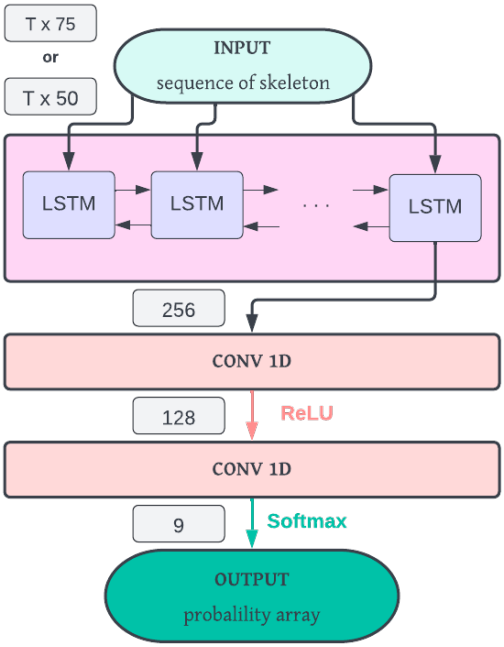


Figure 11. Architecture of our model.

The input is a sequence of 25 key points (one point per body joint) that are either in 2D or 3D. For ease of data manipulation, we will use a $(T, 50)$ shape if we consider two-dimensional data or a $(T, 75)$ shape in case of three-dimensional data, where the integer T represents the length of the sequence.

Given the explicit sequentiality of the data, we naturally decided to include a recursive part in our model to take into account the ordered aspect of the images. The input is therefore given to a single-layer bidirectional LSTM network with a hidden size of 256. Once the input sequence is processed, the final hidden state of the model is given as input to a classifier composed of two one-dimensional convolutional layers with a ReLU activation function in between, and a softmax operator at the end that allows us to categorize the action accordingly.

The training of our models on the given dataset is performed using an NVIDIA® GeForce® GTX 1660 Ti and considering a split ratio of 80% for the training dataset, a cross-entropy loss, an Adam optimizer, a number of epochs equal to 200, a batch size equal to 32 and finally a learning rate starting at 10^{-4} and divided by 10 every 10 epoch.

5.3. Video stream

Before exploring different configurations and implementing any model, we have developed an interface to visualize our predictions in real time. Screenshots of this interface are available in the *Figures* 12 and 13. The interface is relatively simple. On the left side, we display the evolution bars that translate the probability that an action is performed. On the right side, and whatever the model used, we display the skeletal representation using the pixel coordinates.

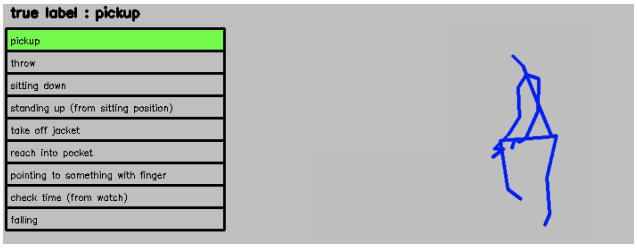


Figure 12. (A) Visualization of predictions (from 3D data).

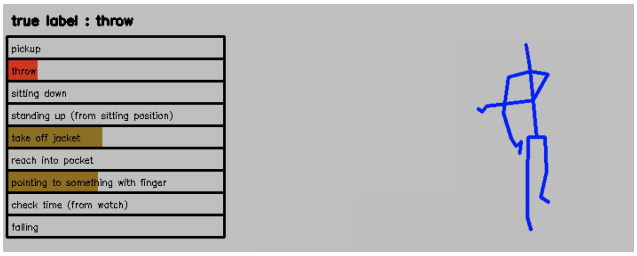


Figure 13. (B) Visualization of predictions (from 3D data).

5.4. Influence of the dataset

5.4.1 Complete, masked and truncated configurations

In the context of our research we decided to explore three settings corresponding to three criteria that make sense to consider in the prediction of actions.

[3]Pearson correlation value.
[4]Representing actions that are relevant to monitor in a public place.

1. Complete samples: the first situation is relatively simple. We consider as input the set of sequences taken individually in our dataset. Therefore, the task of our network is simply to take as input a sequence of skeletal representations and to return the index of the corresponding action. We will refer to this as the complete samples situation.

2. Masked samples: the second situation echoes a real application of action recognition. Indeed, sometimes, for technical or practical reasons, we do not have access to the exact position of some of the joints. Therefore, given a dataset with a certain random amount of hidden joints, we want to know to what extent the model can adapt and learn to make valid predictions. We will refer to this as the masked samples situation.

3. Truncated samples: the third situation proposes to study the effect of the length of the input sequence on the predictive ability of the model. Indeed, on some occasions, such as during a fall, it can be interesting to be able to quickly detect the action performed. We will call this situation truncated samples.

5.4.2 Building the datasets

Since the complete samples situation corresponds to our base case, there is no need to transform our initial dataset for it. The processed samples all had different sequence lengths in terms of frames, therefore, we had to adjust their sizes in the same batch using a PyTorch padding function dedicated to RNNs. In this last processing, we decided to use 0 padding to start with.

For the hidden samples, given a percentage of the number of joints to be hidden, we randomly sample the joints to be hidden for each frame of each sequence in the original dataset with the goal of creating a new set suitable for this situation. The percentages considered are all multiples of ten, ranging from 0% to 90%. As seen previously on the *Figure* 5, the values of the considered coordinates are all very far from the value $\alpha = -100$, hence we decided to use this value to designate a missing joint.

For truncated samples, given a percentage p of the number of frames to consider, we only consider the first p percent of frames for each sequence in the original data set. In this case, the range of percentages varies from 10% to 100%.

5.4.3 Results

For each of the three data sets we trained an instance of the neural network that we will name respectively **basic model**, **masked model** and **truncated model**. The evolution of losses and accuracy is presented for each model in *Figure* 14. Due to a very slow convergence process, we have been forced to double the number of training epochs when training on masked samples.

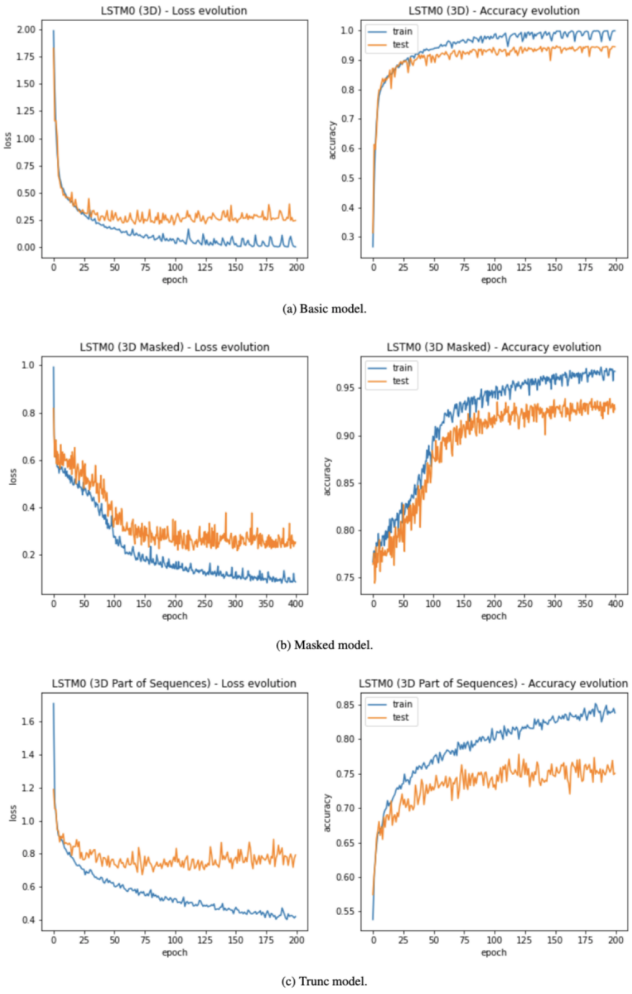


Figure 14. Evolution of the loss and accuracy for each of the three models.

We first present below in *Table* 3 the results obtained by evaluating the three pre-trained models on the test set of the first configuration (complete samples). Although the basic model seems to perform slightly better, the other two models are very close, suggesting that the first situation can easily be handled by considering one of the other situations.

We then decided to confront the three pre-trained models by attempting to make predictions on the samples in the second dataset from the masked sample situation. A graph of the accuracy obtained per percentage of the visible skeleton is shown in *Figure* 16 of the Appendix. The results we obtain are relatively disappointing since we observe that the second model, although trained specifically for this task, is not able to significantly outperform the other two models when the percentage of hidden joints is too high. In partic-

model	**0**	**1**	**2**	**3**	**4**
basic	99	98	99	98	98
masked	98	96	96	97	99
part	98	95	98	97	99
model	**5**	**6**	**7**	**8**	**avg**
basic	97	98	97	99	98
masked	97	95	94	94	96
part	95	93	88	97	96

Table 3. Accuracies over the validation set of the first situation.

ular, when less than 90% of the skeleton is available, there appears to be little point in attempting to make a prediction.

Finally, we decided to also confront the three pre-trained models by considering this time the last dataset of the third situation. As for the previous situation, a similar graph of the accuracies by percentage of the visible sequence is available in the *Figure* 17 in the Appendix. As expected, the truncated model performs significantly better when evaluated on a portion of the sequences (with only 40% of the sequences, it is able to achieve 75% accuracy).

From this study of datasets, we understand the importance of incorporating masked or truncated data when training our model to ensure that it is able to generalize to very specific cases, and to have more finesse in the way it predicts the current action. Since we want to get the most flexible model possible, we consider from now on a default dataset with the following characteristics:

- A random proportion p_1, up to 10% (in agreement with the previous results), of the skeletal joints is hidden,
- A random proportion p_2 of the sequences, ranging from 10% to 100%, is considered.

5.5. Influence of the memory

Now that we have studied the influence of the data structure (in terms of 3D key points and frames obstruction) on the predictive ability of our model, we study the influence of its memory on its ability to adapt when information about previous events is kept in memory.

5.5.1 Consecutive actions

In a first configuration, considering the same three previously trained models, we wish to determine their ability to make predictions in a given context. More specifically, if another action has just been performed, are these networks able to correctly predict the last action. The results obtained are presented in *Figure* 18 of the Appendix. The first three heatmaps show the accuracy of the predictions given a first context action while the last three show the average number of frames of the second action needed to correctly predict the action.

As we can quickly see, the three models perform rather poorly even when the contextual action is identical to the target action. We also observe that when the first contextual action is action 6 (pointing at something) and especially action 7 (checking the time), the predictions are systematically very poor. However, an exception, common to all three models, is still visible. Indeed, for the last class which corresponds to a fall, the three models seem to maintain a correct behavior. This can be explained by the fact that it is a singular action for which all the skeletal joints have a very similar dynamics.

5.5.2 Time window

Similarly, we are interested in the ability of the three models to predict two consecutive actions, but this time considering a temporal sliding window. We try to predict the last action performed, which allows, depending on the respective length of the two sequences, either to consider only the end of the first action, or to consider only the beginning of the second action. The *Figures* 19 and 20 in Appendix, present the results obtained for time windows respectively of 75 frames and 100 frames.

There is a significant difference, in terms of precision, in the efficiency of the three models compared to the results obtained previously. This phenomenon is understandable because here the long-term memory of the different models is only slightly affected since we restrict ourselves to the last frames that contain the essential of the target action, taking into account the average and median number of frames for a given action (which we studied when presenting the dataset).

These good results obtained using a sliding window raise the question of the relevance of training a model on sequences of more than 100 frames. In particular, our results show that in the context of live prediction on potentially long sequences we can restrict ourselves to a finite number of the last images in order to not only save in power and computation time but also to increase precision.

5.6. Final models

Finally we choose to use two models that take the results of the previous experiments into account and whose basic structure is identical to the previous ones. In fact, it is entirely based on the architecture available in the *Figure* 14. We believe that the study of the training dataset and how to make a prediction has a potentially larger impact on the accuracy and success of the results.

5.6.1 Dataset and training

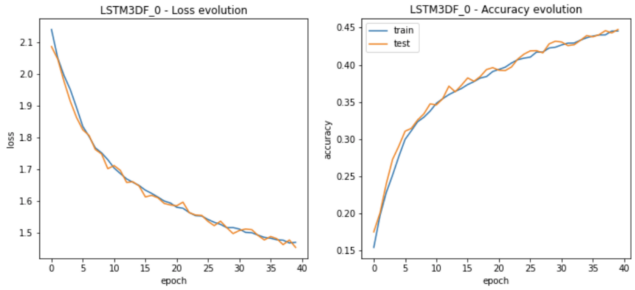
The processing of the data provided as input to the two models is slightly different from before. It takes up the previous analysis on the interest of using consecutive actions during

training. We are now considering successions of events for both models: we concatenate a complete action, which we call a context, to a percentage of the sequence of a second action, which we call a situation.

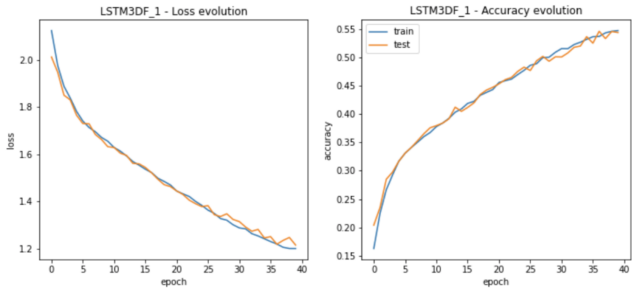
In the first model, whether for training or prediction, we consider the last 100 frames of the association of a context and a situation. In the second model, only during training, we consider only a given situation but do not reset the internal memory of the LSTM network.

5.6.2 Results

In contrast to the previous models that we trained on 200 or 400 epochs, we only train our two final models on 40 epochs because the size of the new dataset is such that the computational time per batch is now much more important. Despite this, we obtain some very interesting results. The evolution of the loss and accuracy of the two models may look mediocre when referring to *Figure* 15, but the training process has simply not yet converged after 40 epochs and these values are computed on a potentially very truncated sequence or with hidden parts dataset and therefore do not necessarily make much sense. It is rather the shape of the curves obtained that is meaningful and that attests to the good evolution of the learning process. The results obtained, similar to those of the first three models, are available in the Appendix (*Figures* 21 to 24).



(a) Basic model.



(b) Masked model.

Figure 15. Evolution of the loss and accuracy for each of the two models.

First, when only complete situations are considered (100% of the sequences), the first models are impressive for their efficiency, as shown in the *Table* 4. In comparison, the second model is not very convincing, but this phenomenon will be explained later.

model	**0**	**1**	**2**	**3**	**4**
first model	100	99	99	100	100
second model	95	73	85	87	98
model	**5**	**6**	**7**	**8**	**avg**
first model	99	99	99	99	99
second model	73	78	34	96	80

Table 4. Accuracies over the validation set of the first situation.

Then, the results around the prediction according to a certain percentage of hidden skeleton (*Figure* 21) are as expected not very promising, regardless of the two models.

Finally, the results of the predictions according to the percentage of sequence given as input (*Figure* 22), approximately follow the results that were also expected. We should precise that the prediction limit is no longer 40% of the sequence but a little higher because in the dataset that we now consider the lengths of the sequences are potentially even shorter.

When it comes to studying the memory influence of these two models the results are mixed. Indeed, assuming a context and a full situation (100% of a sequence), we observe that the first model is not very efficient, which is to be expected given its nature. However, where the surprise lies is that the second model does not achieve the performance that one might hope for (Figure 23). By using the visualization interface, we realize that it takes some time to adapt to the second model. More particularly, it seems difficult for the latter to make predictions starting from a new or recent long memory. Thus we have developed a more accurate metric to assess the predictive capacity of this model.

When we restrict ourselves to a window of 100 frames the results of the second model are not very interesting, but it makes sense because it was not developed for that. The results of the first model are very interesting (*Figures* 24) and attest the relevance of restricting to a finite time window.

5.6.3 Inference methods

In order to perform real-time predictions, we differentiate two prediction modes (one for each of the two final models considered).

To perform a prediction at a given time step, the first model uses the following mode: when a new frame is received at the given time step, it is placed at the end of the current sequence, and when this sequence has a length greater than 100, the first element is removed. This dynamic structure can be described as a FIFO (First In First Out) structure.

To make predictions, the second model is based on this mode: each considered image is submitted to the input of the model which keeps its short and long term memory.

In terms of computation, the first method is more complex because it requires the processing of 100 frames for each prediction, while the second method requires the processing of only one frame.

6. Conclusion and future work

In this work, we proposed a simple but effective strategy to perform human action recognition from two- or three-dimensional skeletal data. The adoption of these skeletal data was motivated by the computational time savings generated.

We started by defining and experimentally validating an architecture that served as a basis for the rest of our work. All our experiments were performed using this common architecture but considering different training configurations.

We then studied the influence of the data structure by first considering the dataset as it was, then by hiding some joints in the frames, and finally by truncating the sequences. Our results demonstrated the value of incorporating hidden joints and truncated sequences into data during training.

Next, we studied the influence of the network's internal memory on its predictive capacity when information about previous events is kept in memory. By first considering the succession of a contextual action followed by a target action, then by considering a time window of different sizes moving over this succession of actions, we evaluated the ability of the models to predict the target action. Our results raised the question of the relevance of training a model on sequences of more than 100 frames. In particular, our results show that in the context of live prediction on potentially long sequences, we can restrict ourselves to a small number of the last frames in order to not only save computational power and time but also increase the accuracy of predictions.

Finally, we trained two new models in light of these observations by considering two successive truncated actions containing hidden joints. One of the two models does not reset its memory during training, which reduces the computation time during inference. Note that this computation time remains a priority in human action recognition applications, for example in video surveillance. The results obtained support the previous analyses and lead to more efficient models.

Nevertheless, relevant remarks have been made throughout our work and can therefore be considered as future work. First of all, we have only considered one model architecture without ever modifying it under the assumption that the study of training configurations had a greater impact on the precision of the results. For example, it would be interesting to study the impact of transformers on our predictions or to consider a CNN upstream of the LSTM to process the images individually and possibly extract more information. Finally, it would be interesting to test a full prediction pipeline to verify the robustness of our model. To do this, and in a future work, we will use the OpenPose library [5] to extract the skeletons from an image and use our models to make a prediction.

References

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *international conference on engineering and technology (ICET)*. IEEE, 2017.

[2] Kyunghyun Cho, Bart Van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8)*, 2014.

[3] Haodong Duan, Yue Zhao, Kai Chen, Dian Shao, Dahua Lin, and Bo Dai. Revisiting skeleton-based action recognition. 2021.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[5] Yaadhav Raaj, Haroon Idrees, Gines Hidalgo, and Yaser Sheikh. Efficient online multi-person 2d pose tracking with recurrent spatio-temporal affinity fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019.

[6] Hossein Rahmani, Ajmal Mian, and Mubarak Shah. Learning a deep model for human action recognition from novel viewpoints. *IEEE transactions on pattern analysis and machine intelligence*.

[7] Suneth Ranasinghe, Fadi Al Machot, and Heinrich C Mayr. A review on applications of activity recognition systems with regard to performance and evaluation. *International Journal of Distributed Sensor Networks*, 2016.

[8] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 1997.

[9] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[10] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with directed graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[11] Ah Chung Tsoi. Recurrent neural network architectures: an overview. In *International School on Neural Networks, Initiated by IIASS and EMFCSC*. Springer, 1997.

[12] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Appendix

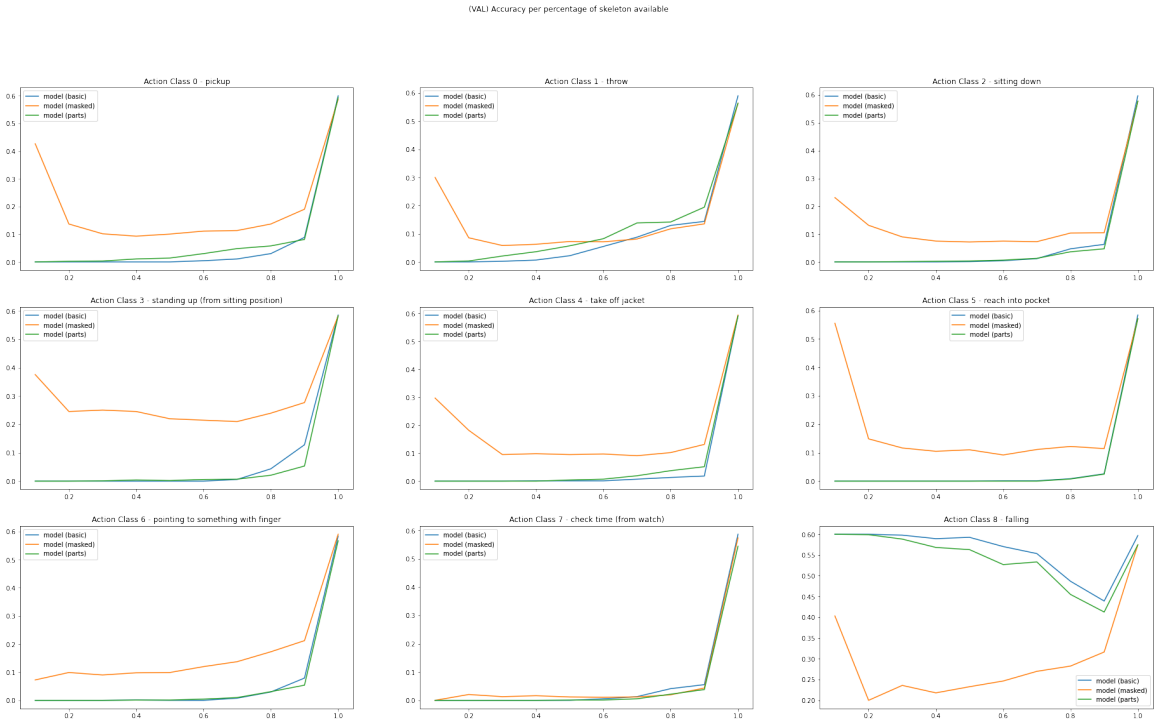


Figure 16. Evolution of the accuracy per percentage of skeleton available.

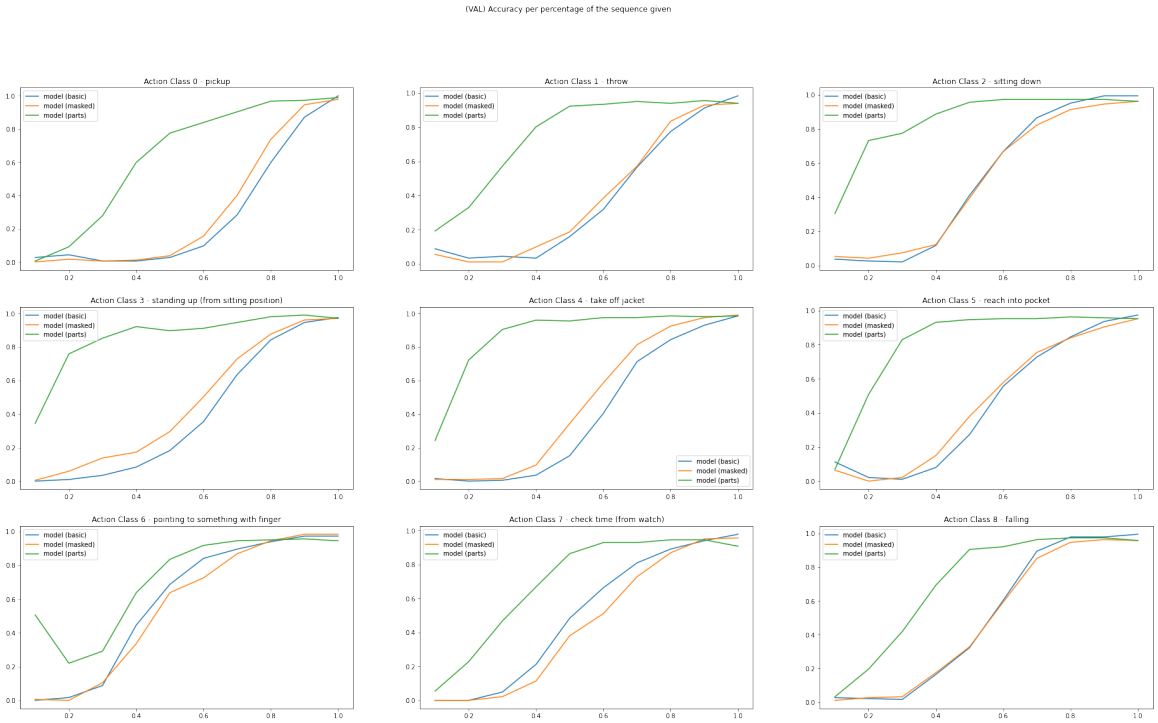


Figure 17. Evolution of the accuracy per percentage of the sequence available.

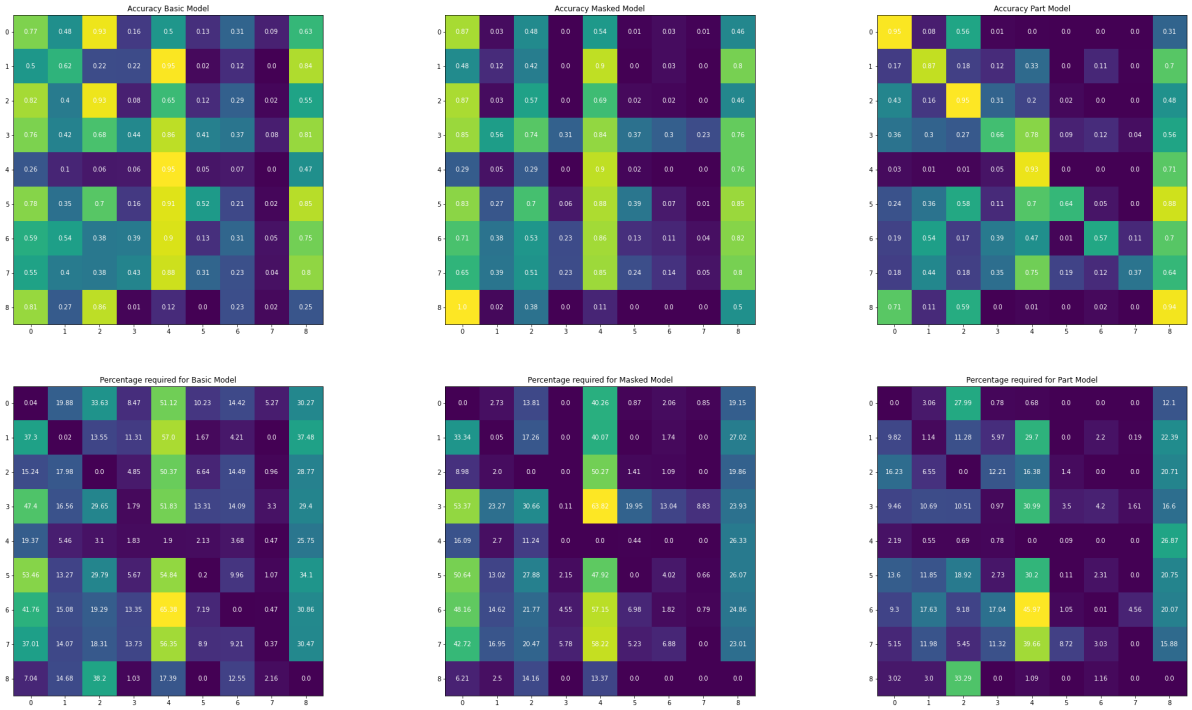


Figure 18. Accuracy per targeted action (lines) given a previous contextual one (columns).

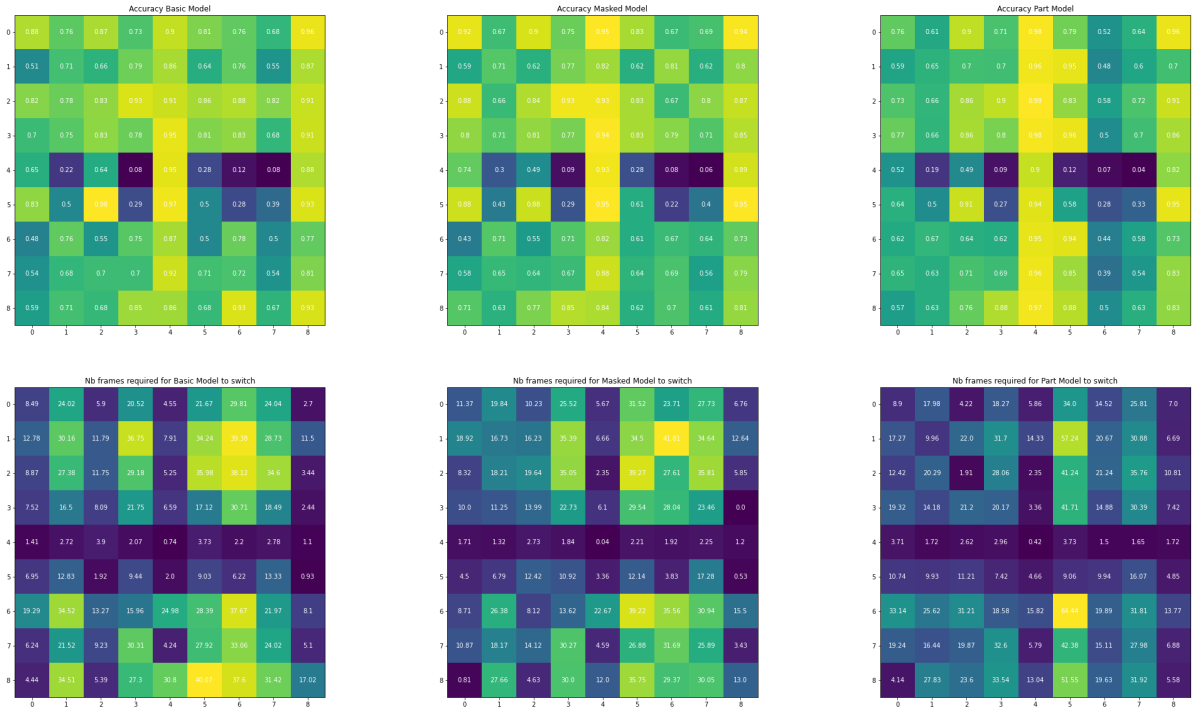


Figure 19. Accuracy per targeted action (lines) given a previous contextual one (columns) using a time window of 75 frames.

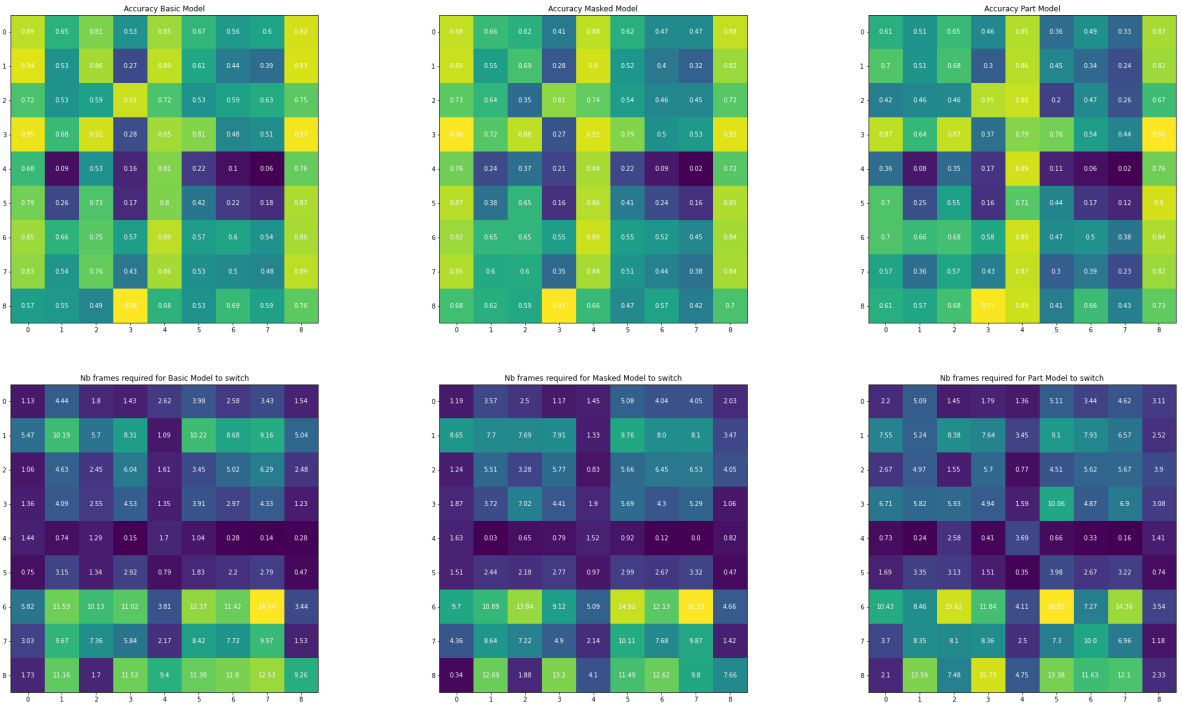


Figure 20. Accuracy per targeted action (lines) given a previous contextual one (columns) using a time window of 100 frames.

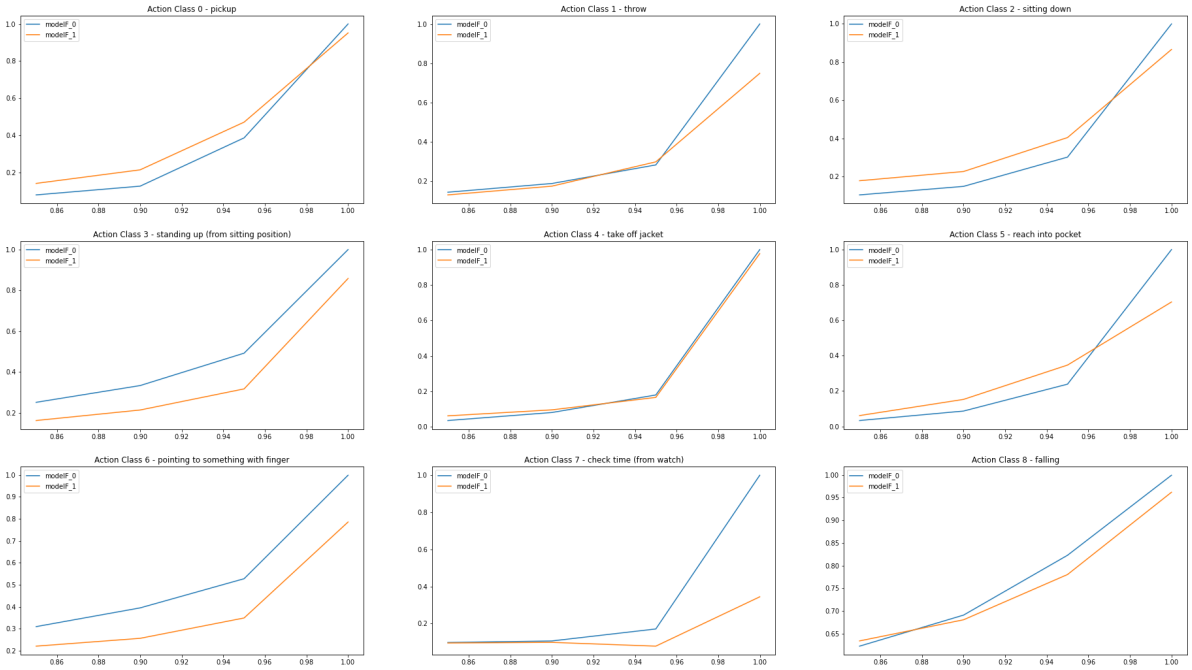


Figure 21. Evolution of the accuracy per percentage of skeleton available.

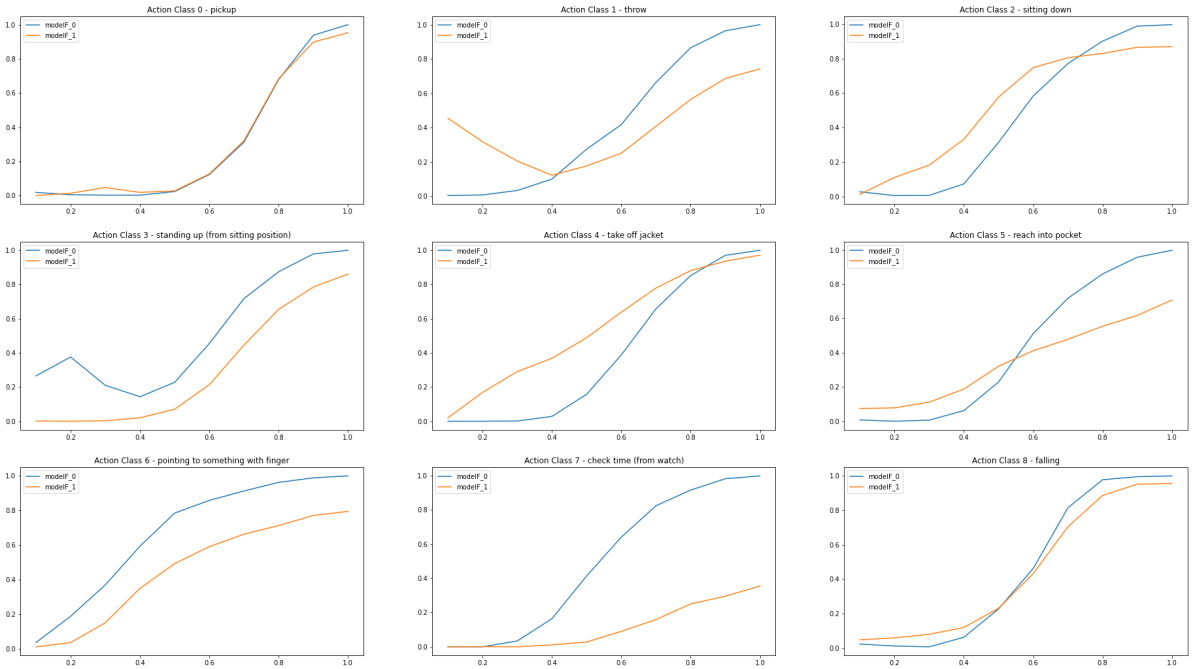


Figure 22. Evolution of the accuracy per percentage of the sequence available.

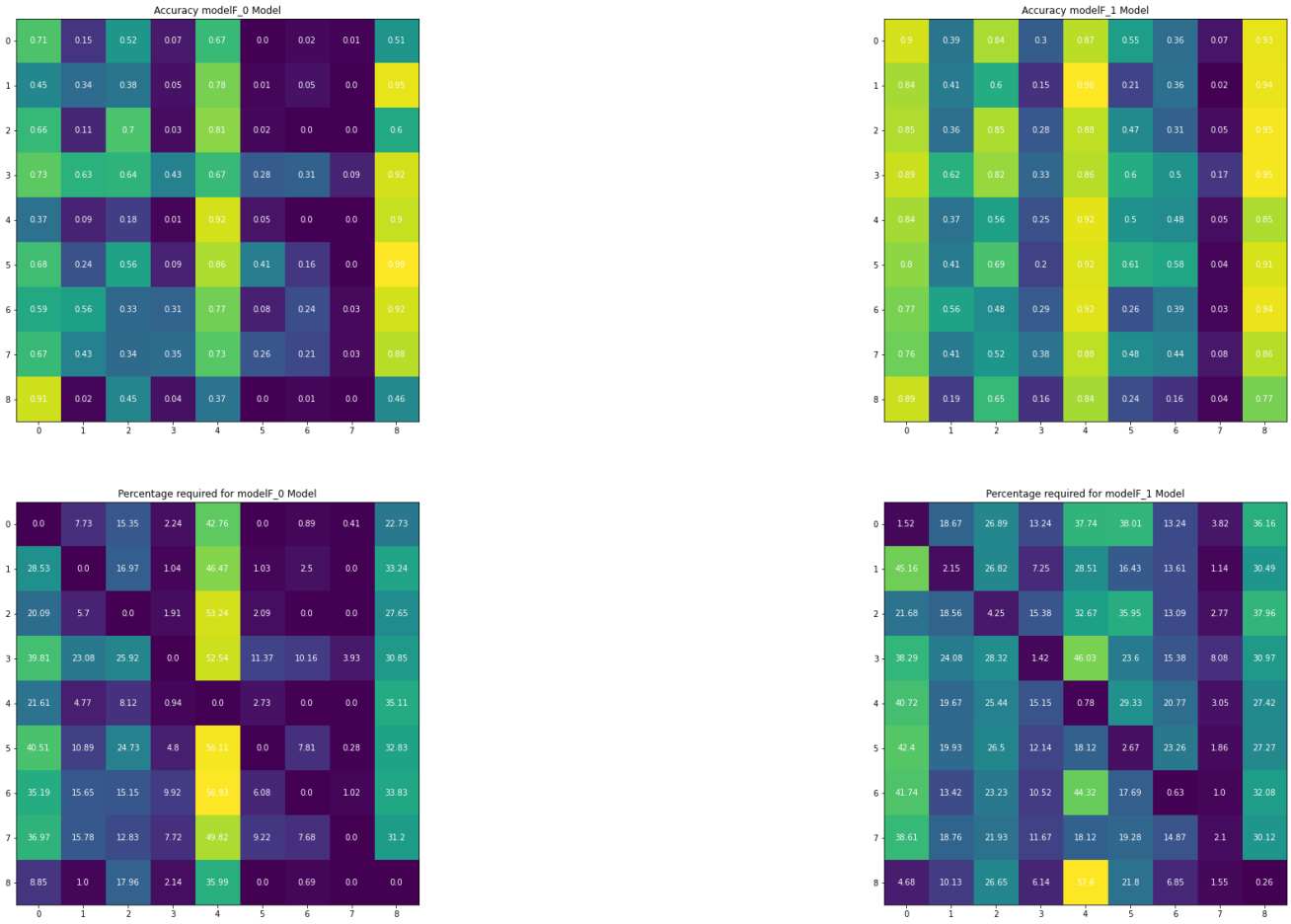


Figure 23. Accuracy per targeted action (lines) given a previous contextual one (columns).

(a) Accuracy.

(b) Average number of frames to switch.

Figure 24. Accuracy per targeted action (lines) given a previous contextual one (columns) using a time window of 100 frames.